ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΓΡΑΦΟΘΕΩΡΗΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΠΡΟΓΡΑΜΜΑΤΩΝ

ΚΩΝΣΤΑΝΙΝΟΣ ΠΑΤΑΚΑΚΗΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:

Γεωργιάδης Λουκάς

ΕΠΙΒΛΕΠΩΝ ΣΥΝΕΡΓΑΤΗΣ:

Χρήστος Τζώρτζης,

ΚΟΖΑΝΗ(ΜΑΡΤΙΟΣ, 2011)

ΠΡΟΛΟΓΟΣ

Η δημιουργία ενός αποδοτικού προγράμματος και συνάμα απλού στην σύνταξη και στην κατανόηση είναι ιδιαίτερα δύσκολη για μεγάλα προγράμματα. Η ανάγκη για την υλοποίηση αποδοτικών αλλά και κατανοητών προγραμμάτων, οδήγησε στην ανάπτυξη αυτοματοποιημένων τεχνικών που χρησιμοποιούνται από σύγχρονους μεταγλωττιστές. Στόχος αυτών των τεχνικών είναι να βελτιώσουν το απλό, στην σύνταξη, πρόγραμμα, έτσι ώστε ο χρόνος εκτέλεσης του να πλησιάζει όσο το δυνατόν περισσότερο εκείνου του βέλτιστου χρόνου. Αυτές οι τεχνικές δεν αλλοιώνουν καθόλου τη σημασιολογία του προγράμματος και εκτελούνται στο επίπεδο της μεταγλώττισης. Σημαντικό ρόλο σε αυτές τις τεχνικές παίζουν οι κόμβοι κυριαρχίας οι οποίοι και αναλύονται διεξοδικά στο κείμενο της διπλωματικής. Το πρόβλημα της γρήγορης εύρεσης των κόμβων κυριαρχίας είναι αρκετά περίπλοκο είτε το μελετάμε στη στατική του μορφή είτε το μελετάμε στη προσαυξητική του μορφή. Στα πλαίσια αυτής της διπλωματικής παρουσιάζουμε τεχνικές και αλγόριθμους που υπολογίζουν τόσο το στατικό όσο και το προσαυξητικό πρόβλημα των κόμβων κυριαρχίας. Επιπλέον παρουσιάζουμε πειραματικά δεδομένα για την αποδοτικότητα αυτών των προσαυξητικών αλγορίθμων στη πράξη.

Εισαγωγή

Όταν ένα πρόγραμμα υλοποιείται απρόσεκτα ή σε κάποια γλώσσα προγραμματισμού υψηλού επιπέδου, τότε η απόδοσή του μπορεί να είναι αρκετά χειρότερη σε σύγκριση με την απόδοση που μπορεί να έχει μία προσεκτική υλοποίηση σε γλώσσα μηχανής. Έτσι, οι τεχνικές βελτιστοποίησης κώδικα ενός προγράμματος, εκτός από σημαντικές, κρίνονται, σε κάποιο επίπεδο, και απαραίτητες. Για να καταφέρουμε όμως να τις κατανοήσουμε και να τις περιγράψουμε κατάλληλα θα πρέπει αρχικά να αποκτήσουμε οικειότητα με μία σειρά άλλων τεχνικών και ιδεών στις οποίες βασίζονται οι αλγόριθμοι βελτιστοποίησης.

Ένα κομμάτι κώδικα, εκτός από τη μορφή κειμένου, μπορεί να αναπαρασταθεί και ως γράφημα. Αυτό το γράφημα στην γλώσσα των προγραμματιστών είναι γνωστό και ως το γράφημα ροής του προγράμματος. Όταν ένας μεταγλωττιστής αναλύει ένα πρόγραμμα το μετασχηματίζει από κείμενο σε γράφημα ροής. Από αυτό και μόνο μπορούμε να καταλάβουμε ότι αυτή η τεχνική είναι πολύ σημαντική, αν όχι η σημαντικότερη, για να ανιχνεύσουμε τις περιοχές τις οποίες μπορούμε να βελτιστοποιήσουμε ένα πρόγραμμα. Αυτός είναι και ο λόγος που στο πρώτο κεφάλαιο κάνουμε μία εισαγωγή στην θεωρία των γραφημάτων, μελετώντας βασικές έννοιες και προτείνοντας μία σειρά από δομές δεδομένων με τις οποίες μπορεί να αναπαρασταθεί ένα γράφημα προγραμματιστικά.

Βέβαια εσκεμμένα παραλείψαμε να αναφέρουμε ότι αν ο κώδικας είναι συνταγμένος σε μία γλώσσα υψηλού επιπέδου τότε, το κείμενο σε αυτή τη γλώσσα, δεν είναι κατάλληλο για την μετάβαση στο γράφημα ροής του. Οι μεταγλωττιστές των γλωσσών υψηλού επιπέδου, πριν προχωρήσουν στον μετασχηματισμό του κειμένου αρχικά το μεταφράζουν σε μία ενδιάμεση γλώσσα. Αυτή η γλώσσα είναι κατάλληλη για τη δημιουργία του γραφήματος ροής αφού περιέχει μόνο εντολές χαμηλού επιπέδου. Στο κεφάλαιο δύο μελετάμε μία από τις πλέον πιο γνωστές ενδιάμεσες γλώσσες, εκείνη των τριών διευθύνσεων, καθώς επίσης και τον τρόπο με τον οποίο κατασκευάζεται το γράφημα ροής ενός προγράμματος. Στο τελευταίο τμήμα αυτού

του κεφαλαίου παίρνουμε μία ιδέα των αλγορίθμων με τους οποίους μπορούμε να βελτιώσουμε ένα πρόγραμμα, αφού εξετάσουμε την τοπική βελτιστοποίηση.

Στο τρίτο κεφάλαιο εισάγεται η ιδέα των κόμβων κυριαρχίας και επίσης περιγράφονται οι τρόποι με τους οποίους μπορούμε να τους υπολογίσουμε σε ένα στατικό γράφημα. Στο ίδιο κεφάλαιο εξετάζουμε και μία σχετική έννοια η οποία ονομάζεται όριο κυριαρχίας. Αυτό το κεφάλαιο αποτελεί ένα προπαρασκευαστικό κεφάλαιο το οποίο θα μας βοηθήσει στην περιγραφή μίας νέας και πολύ ισχυρής, στις βελτιστοποιήσεις, ενδιάμεσης γλώσσας η οποία φέρει το όνομα «δομή στατικής και μοναδικής ανάθεσης» (single static assignment form), ή πιο απλά δομή SSA. Στο πέμπτο κεφάλαιο γίνεται παρουσίαση των τεχνικών βελτιστοποίησης που λαμβάνουν χώρα σε ολόκληρο το πρόγραμμα. Εδώ πάλι εμφανίζεται η έννοια των κόμβων κυριαρχίας, αφού σε αυτούς βασίζονται οι μέθοδοι που βελτιώνουν τους βρόχους.

Σε αρκετές εφαρμογές το γράφημα ροής ενός προγράμματος μπορεί να μεταβάλλεται με το πέρασμα του χρόνου. Το να διατηρήσουμε τα σωστά σύνολα των κόμβων κυριαρχίας είναι κρίσιμης σημασίας. Στο κεφάλαιο έξι μελετάμε το προσαυξητικό πρόβλημα των κόμβων κυριαρχίας, δηλαδή πως αλλάζουν τα σύνολα των κόμβων κυριαρχίας όταν προσθέτουμε ακμές, καθώς επίσης και διάφορους αλγόριθμους που υπολογίζουν τα νέα σύνολα μετά την αλλαγή. Στο τέλος του έκτου κεφάλαιο παρουσιάζονται μερικά πειραματικά αποτελέσματα τα οποία μας δείχνουν ποιος είναι ο καλύτερος τρόπος να αναπαρασταθεί ένα γράφημα σε ένα πρόγραμμα καθώς επίσης και ποιος αλγόριθμος υπολογισμού του προσαυξητικού προβλήματος των κόμβων κυριαρχίας είναι πιο αποδοτικός.

Περιεχόμενα

ΠΡΟΛΟΓΟΣ......3

ΕΙΣΑΓΩΓΗ......5

ΠΕΡΙΕΧΟΜΕΝΑ......7

1 ΓΡΑΦΗΜΑΤΑ **11**

1.1 Εισαγωγή **11**

1.2 Βασική Ορολογία **11**

1.3 Αναπαράσταση Γραφημάτων **19**

1.3.1 Πίνακας Ακμών (Edge Matrix)......19

1.3.2 Πίνακας Γειτνίασης (Adjacency Matrix)......21

1.3.3 Λίστες Γειτνίασης (Adjacency Lists) 23

1.3.4 Συνδεδεμένοι Πίνακες Γειτνίασης (Linked Adjacency Arrays)......26

1.3.5 Δυναμικοί Πίνακες Γειτνίασης (Dynamic Adjacency Arrays) 27

1.4 Διαπεράσεις Κατευθυνόμενων Γραφημάτων...... **28**

1.4.1 Αναζήτηση Κατά Πλάτος (Breadth First Search) 28

1.4.2 Αναζήτηση Κατά Βάθος (Depth First Search)......31

2 ΒΑΣΙΚΑ ΜΠΛΟΚ ΚΑΙ ΓΡΑΦΗΜΑΤΑ ΡΟΗΣ ΕΛΕΓΧΟΥ...... **33**

2.1 Κώδικας Τριών Διευθύνσεων (Three-address code)...... **33**

2.2 Βασικά Μπλοκ (Basic Blocks) **37**

2.3 Γραφήματα Ροής Ελέγχου (Control Flow Graph) **39**

2.4 Βελτιστοποίηση των Βασικών Μπλοκ **42**

2.4.1 Κατευθυνόμενη Άκυκλη Γραφική Αναπαράσταση των Βασικών Μπλοκ 43

2.4.2 Εύρεση Ομοίων Τοπικών Εντολών......44

2.4.3 Διαγραφή «Νεκρού» Κώδικα......46

2.4.4 Χρήση Αλγεβρικών Ταυτοτήτων 46

2.4.5 Αναπαραστάσεις σε Αναφορές Πινάκων......48

2.4.6 Αναθέσεις σε Δείκτη και Κλήσεις Διαδικασιών 50
2.4.7 Συναρμολόγηση των Βασικών Μπλοκ από το ΚΑΓ 51

3 ΚΟΜΒΟΙ ΚΥΡΙΑΡΧΙΑΣ ΚΑΙ ΟΡΙΑ ΚΥΡΙΑΡΧΙΑΣ 54

3.1 Εισαγωγή 54

3.2 Βασικοί Αλγόριθμοι 57
3.2.1 Αλγόριθμος Purdom-Moore 58
3.2.2 Επαναληπτικός Αλγόριθμος 58
3.2.3 Αλγόριθμος Lengauer-Tarjan 63
3.2.4 Αλγόριθμοι Γραμμικοί στο Χρόνο 65

3.3 Όρια Κυριαρχίας 66

4 ΔΟΜΗ ΣΤΑΤΙΚΗΣ ΜΟΝΑΔΙΚΗΣ ΑΝΑΘΕΣΗΣ (STATIC SINGLE ASSIGNMENT FORM - SSA) 69

4.1 Περιγραφή της SSA Δομής 69

4.2 Κατασκευή της SSA Δομής 72
4.2.1 Κατασκευή Πίνακα στην SSA δομή 75
4.2.2 Κατασκευή Δομών στην SSA δομή 75
4.2.3 Έμμεσες Αναφορές σε Μεταβλητές 76

4.3 Η Ελάχιστης SSA δομής 78

4.4 Κατασκευή της Ελάχιστης SSA δομή 80

4.5 Κατασκευή των Εξαρτήσεων Ελέγχου 84

4.6 Η Μετάφραση από την SSA δομή 86
4.6.1 Εξάλειψη Νεκρού Κώδικα 87
4.6.2 Κατανομή Αποθήκευσης με Χρωματισμό 90

5 ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΚΩΔΙΚΑ 92

5.1 Οι Πηγές της Βελτιστοποίησης 92
5.1.1 Πλεονασμοί 92
5.1.2 Σημασιολογική Διατήρηση 96
5.1.3 Κοινές Εντολές σε Ολόκληρο το Πρόγραμμα 97

5.1.4 Μεταφορά Αντιγράφου 100
5.1.5 Αφαίρεση «Νεκρού» Κώδικα 100
5.1.6 Έλεγχος της Κίνησης του Κώδικα 101
5.1.7 Επαγωγικές Μεταβλητές και Μείωση της Ισχύς 102

5.2 Μεταφορά Σταθεράς 106

5.3 Βελτιστοποίηση Βρόχων 108
5.3.1 Οπισθοακμές και Μειώσιμα Γραφήματα 109
5.3.2 Φυσικοί Βρόχοι 110
5.3.3 Σύγκλιση των Επαναληπτικών Αλγορίθμων 111

6 ΤΟ ΠΡΟΣΑΥΞΗΤΙΚΟ ΠΡΟΒΛΗΜΑ ΤΩΝ ΚΟΜΒΩΝ ΚΥΡΙΑΡΧΙΑΣ 113

6.1 Αναδόμηση του Δέντρου Κυριαρχίας από την Αρχή 114

6.2 Αναδόμηση του Δέντρου Κυριαρχίας με τον Επαναληπτικό Αλγόριθμο Λαμβάνοντας Υπόψη τα Προηγούμενα Σύνολα Κυριαρχίας 114

6.3 Προσαυξητικός Αλγόριθμος των Sreedhar, Gao και Lee 114

6.4 Παραλαγή του Προσαυξητικού Αλγορίθμου Sreedhar, Gao, Lee 118

6.5 Πειραματικά Αποτελέσματα 120
6.5.1 Σύγκριση των Δομών Αναπαράστασης Ενός Γραφήματος 120
6.5.2 Σύγκριση των Αλγορίθμων Υπολογισμού του Προσαυξητικού Προβλήματος των Κόμβων Κυριαρχίας 127

ΒΙΒΛΙΟΓΡΑΦΙΑ........ 120

1 ΓΡΑΦΗΜΑΤΑ

1.1 Εισαγωγή

Στα μαθηματικά και στην επιστήμη των υπολογιστών, η θεωρία των γραφημάτων είναι η μελέτη των μαθηματικών κατασκευών που χρησιμοποιούνται για να μοντελοποιήσουν μία διμελή σχέση μεταξύ των αντικειμένων ενός συνόλου. Αποτελεί ένα συναρπαστικό κλάδο των εφαρμοσμένων μαθηματικών, με ποικίλα και ενδιαφέροντα προβλήματα κυρίως στον τομέα της πληροφορικής.

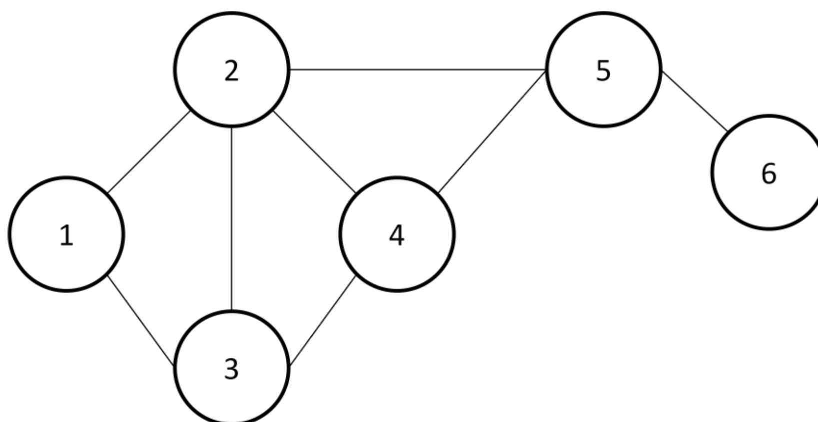
Ιστορικά, οι απαρχές της θεωρίας των γραφημάτων χρονολογείται στον 18^{ο} αιώνα. Ουσιαστικά, ωστόσο, αυτή η θεωρία αναπτύχτηκε πολύ μεταγενέστερα, και μάλιστα μεταπολεμικά, ως μία ανεξάρτητη περιοχή των Εφαρμοσμένων Μαθηματικών. Σήμερα, η θεωρία των γραφημάτων βρίσκει πολλές εφαρμογές σε ένα ευρύ φάσμα γνωστικών πεδίων, όπως η Πληροφορική, η Μηχανική, η Χημεία, η Επιχειρησιακή Έρευνα και η Κοινωνιολογία.

Στον τομέα της πληροφορικής τα γραφήματα αποτελούν ένα ισχυρότατο εργαλείο για την αντιμετώπιση πολλών πρακτικών προβλημάτων, αλγοριθμικής κυρίως φύσεως, τα οποία εμφανίζονται σε ποικίλες περιοχές όπως η Κατασκευή Αλγορίθμων, οι Βάσεις Δεδομένων, η Τεχνητή Νοημοσύνη, η Ανάκτηση Πληροφοριών, ο Παράλληλος Υπολογισμός και τα Δίκτυα Υπολογιστών όπως επίσης και σε πολλά ακόμα.

1.2 Βασική Ορολογία

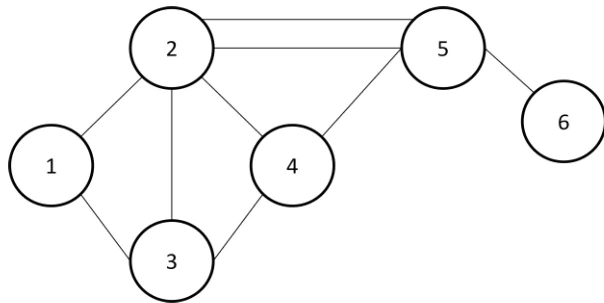
Ένα μη κατευθυνόμενο γράφημα (undirected graph) $G = (V, E)$ αποτελείται από δύο σύνολα, το V που αποτελεί το σύνολο των κόμβων ενώ το E που αποτελεί το σύνολο των ακμών. Το σύνολο E απαρτίζει μία συμμετρική διμελή σχέση επί του συνόλου V. Ένα παράδειγμα μη κατευθυνόμενου γραφήματος φαίνεται στο σχήμα 1.1. Ένας πιο επίσημος ορισμός του μη κατευθυνόμενου γραφήματος είναι ο ακόλουθος:

Ορισμός 1.1: Έστω το μη κενό και πεπερασμένο σύνολο V με n διακεκριμένα στοιχεία $V = \{u_1, \dots, u_n\}$, και E ένα σύνολο με $m \geq 0$ μη διατεταγμένα ζεύγη $e_{ij} = \{u_i, u_j\}$, $i \neq j$, στοιχείων του V. Τότε το διατεταγμένο ζεύγος $G = (V, E)$ ονομάζεται μη κατευθυνόμενο γράφημα (undirected graph) ή απλώς γράφημα.

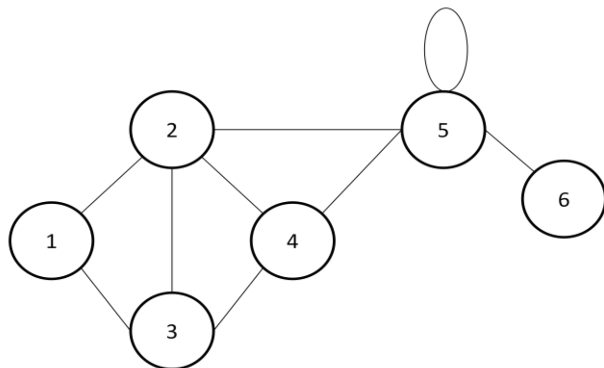


Σχήμα 1.1: Μη κατευθυνόμενο γράφημα.

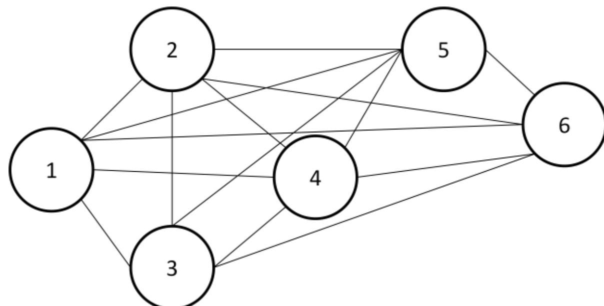
Έτσι, εάν $e_{ij} = \{u_i, u_j\}$ ανήκει στο E και τα u_i και u_j ανήκουν στο V, λέμε ότι η ακμή e προσπίπτει στα u_i και u_j ή λέμε ότι η e έχει άκρα τα u_i και u_j. Τάξη (order) του γραφήματος ονομάζεται ο αριθμός των κόμβων, ενώ μέγεθος (size) ονομάζεται το σύνολο των ακμών του. Δύο κόμβοι ονομάζονται γειτονικοί (adjacent) όταν υπάρχει μία ακμή που τους ενώνει. Σε διαφορετική περίπτωση οι κόμβοι ονομάζονται ανεξάρτητοι (independent). Ο αριθμός των ακμών που προσπίπτουν σε ένα κόμβο ορίζουν τον βαθμό του κόμβου και συμβολίζεται ως $deg(u)$. Όταν ένα γράφημα δεν περιέχει βρόχους από ένα κόμβο προς τον εαυτό του, δηλαδή ακμές οι οποίες ξεκινούν και καταλήγουν στον ίδιο κόμβο, και δεν περιέχει περισσότερες από μία ακμές από ένα συγκεκριμένο κόμβο προς ένα συγκεκριμένο κόμβο, ή αλλιώς παράλληλες ακμές, χαρακτηρίζεται ως απλό (simple) γράφημα. Ένα απλό γράφημα αποτελεί το γράφημα του σχήματος 1.1 . Τα γραφήματα στα οποία επιτρέπονται παράλληλες ακμές ονομάζονται πολυγραφήματα (multigraphs) (σχήμα 1.2 α) ενώ εκείνα στα οποία επιτρέπονται βρόχοι ονομάζονται ψευδογραφήματα (pseudographs)(σχήμα 1.2 β). Πλήρες γράφημα ονομάζεται το γράφημα στο οποίο όλοι οι κόμβοι συνδέονται μεταξύ τους και συμβολίζεται ως K_n όπου $n = |V|$ (σχήμα 1.2 γ).



α) Πολυγράφημα



β) Ψευδογράφημα

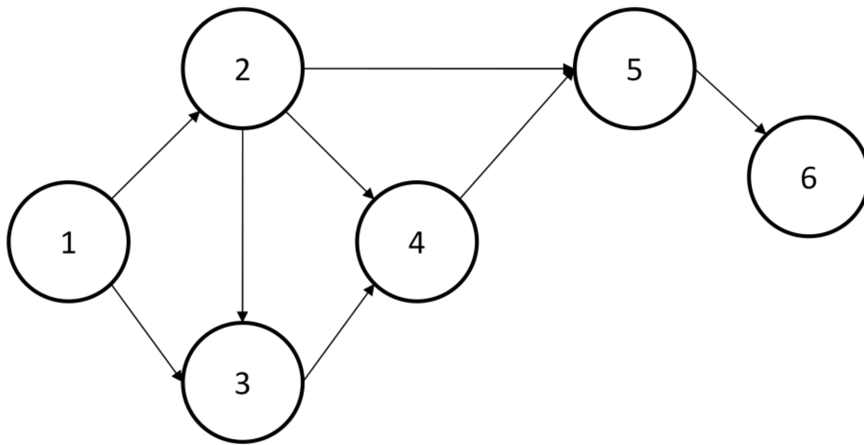


γ) Πλήρες γράφημα

Σχήμα 1.2: Γραφήματα.

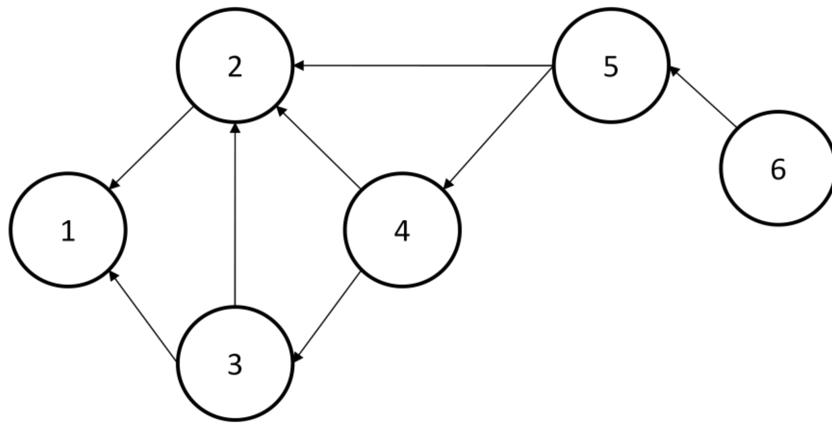
Κατευθυνόμενο γράφημα είναι εκείνη η μαθηματική κατασκευή στην οποία το E αποτελεί ένα σύνολο κατευθυνόμενων ακμών επί του V. Η μόνη διαφορά, σε σχέση με τα μη κατευθυνόμενα γραφήματα, είναι ότι το E πλέον αποτελείται από διατεταγμένα ζεύγη και όχι από απλά διμελή σύνολα. Έτσι πλέον το $e_{ij} \neq e_{ji}$, δηλαδή η ακμή από το i στο j είναι διαφορετική από την ακμή από το j στο i. Ένα κατευθυνόμενο γράφημα φαίνεται στο σχήμα 1.3 . Ο ακόλουθος ορισμός περιγράφει τα κατευθυνόμενα γραφήματα:

Ορισμός 1.2: Ένα κατευθυνόμενο γράφημα $G = (V, E)$ αποτελείται από ένα μη κενό σύνολο κόμβων V και ένα σύνολο E διατεταγμένων ζευγών (u_i, u_j) όπου u_i, u_j ανήκουν στο V, που ονομάζονται κατευθυνόμενες ακμές ή τόξα (arcs).



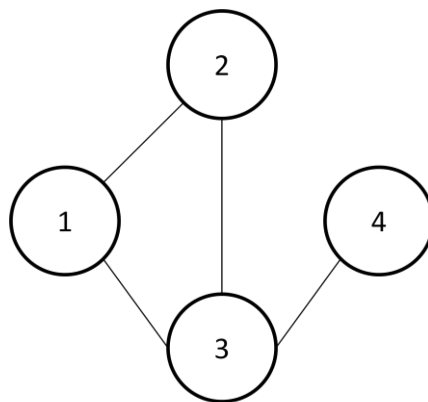
Σχήμα 1.3: Κατευθυνόμενο γράφημα.

Σε ένα τέτοιο γράφημα, ο αριθμός των ακμών που εξέρχονται από έναν κόμβο αποτελούν των βαθμό εξόδου (out-degree) του, ενώ ο αριθμός των ακμών που προσπίπτουν σε ένα κόμβο συνιστούν τον βαθμό εισόδου (in-degree). Κάθε κόμβος με βαθμό εισόδου μηδέν καλείται πηγή (source) ενώ οι κόμβοι με βαθμό εξόδου μηδέν ονομάζονται καταβόθρες (sinks). Δοθέντος ενός κατευθυνόμενου γραφήματος $G = (V, E)$ ορίζουμε το ανάστροφο γράφημα (reverse graph), και το συμβολίζουμε ως $G^R = (V, E^R)$, το γράφημα το οποίο για κάθε e_{ij} που ανήκει στο E το e_{ji} να ανήκει στο E^R. Το ανάστροφο γράφημα του σχήματος 1.3 παρουσιάζεται στο σχήμα 1.4 . Κατά κοινό τρόπο με τα μη κατευθυνόμενα γραφήματα ορίζονται και στα κατευθυνόμενα οι χαρακτηρισμοί πολυγράφημα και ψευδογράφημα ενώ τέλος πλήρες κατευθυνόμενο γράφημα ονομάζεται εκείνο το οποίο έχει ακμές προς κάθε κόμβο και προς κάθε κατεύθυνση.

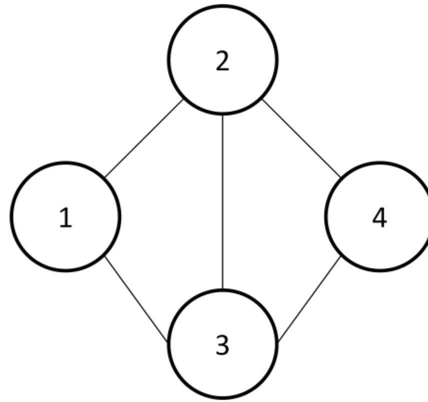


Σχήμα 1.4: Ανάστροφο γράφημα.

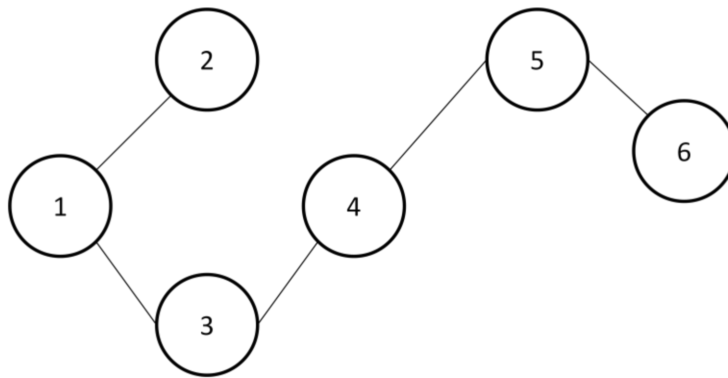
Στη συνέχεια θα δοθούν μερικοί ορισμοί. Τα παραδείγματα αυτών των ορισμών αναφέρονται στο γράφημα $G = (V, E)$ του σχήματος 1.1 . Ένα γράφημα $G' = (V', E')$ καλείται υπογράφημα (subgraph) του G, όταν το V' είναι υποσύνολο του V $(V' \subseteq V)$ και το E' είναι υποσύνολο του E $(E' \subseteq E)$ (σχήμα 1.5) . Στην ειδική περίπτωση που το E' περιέχει όλες τις ακμές του E, οι οποίες σχηματίζονται από κόμβους v για τους οποίους ισχύει ότι $v \in V'$, δηλαδή $E' = E \cap (V' \times V')$, το γράφημα χαρακτηρίζεται επαγόμενο (induced) (σχήμα 1.6), ενώ όταν το V' είναι ίσο με το V το γράφημα καλείται επικαλύπτον (spanning) (σχήμα 1.7).



Σχήμα 1.5: Υπογράφημα.



Σχήμα 1.6: Επαγόμενο.



Σχήμα 1.7: Επικαλύπτον.

Μια ακολουθία k κόμβων $(u_1, u_2, \ldots, u_k)$ με την ιδιότητα ότι $\{u_i, u_{i+1}\}$ να ανήκουν στο E για κάθε i που ανήκει στο διάστημα ακεραίων $[1, k-1]$, αποτελεί το μονοπάτι (path) μήκους $k-1$. Ένα μονοπάτι το οποίο δεν περιέχει επαναλαμβανόμενους κόμβους ονομάζεται απλό μονοπάτι (simple path), ενώ εκείνο για το οποίο ισχύει ότι $u_1 = u_k$ ονομάζεται κύκλος ή απλός κύκλος (simple cycle). Το μεγαλύτερο μήκος του συντομότερου, άρα και απλού, μονοπατιού μεταξύ δύο κόμβων ενός γραφήματος αποτελεί την διάμετρο (diameter) του γραφήματος. Παραδείγματος χάριν η διάμετρος του γραφήματος στο σχήμα 1.1 είναι τρία $(1 \rightarrow 2 \rightarrow 5 \rightarrow 6)$.

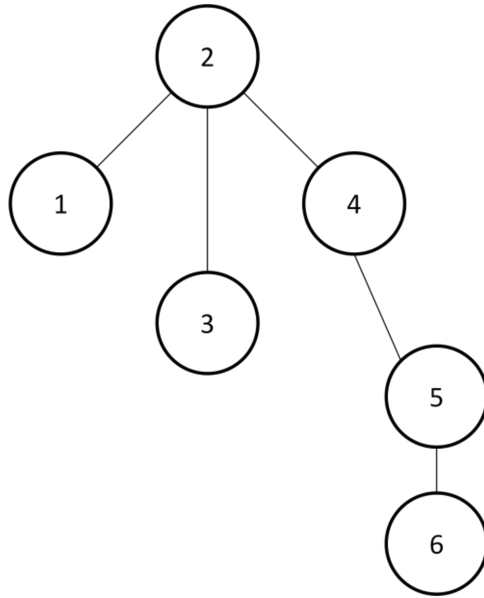
Στα κατευθυνόμενα γραφήματα οι έννοιες του μονοπατιού και της διαμέτρου είναι παρόμοιες μόνο που τώρα λαμβάνουμε υπόψη την κατεύθυνση των ακμών.

Ένα μη κατευθυνόμενο γράφημα ονομάζεται συνεκτικό (connected) όταν για οποιουσδήποτε δύο κόμβους υπάρχει ένα μονοπάτι που τους συνδέει. Σε περίπτωση που το γράφημα δεν είναι συνεκτικό τότε αποτελείται από ένα πλήθος

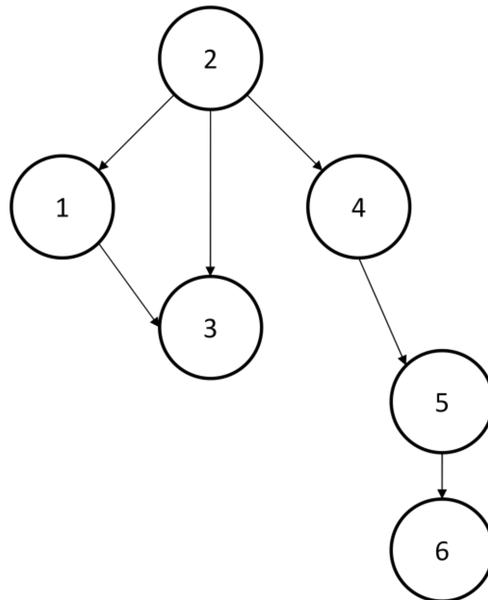
συνεκτικών συνιστωσών (connected components). Δισυνεκτικό (biconnected) ονομάζεται το γράφημα στο οποίο οποιοδήποτε ζευγάρι κόμβων συνδέεται μεταξύ τους με δύο ή περισσότερα μη τεμνόμενα μονοπάτια. Σε περίπτωση που το γράφημα δεν είναι δισυνεκτικό, τότε αποτελείται από δισυνεκτικές συνιστώσες (biconnected components), οι οποίες είναι οι μέγιστες συνιστώσες με την παραπάνω ιδιότητα. Τέλος μία ακμή ονομάζεται γέφυρα (bridge) όταν η αφαίρεση της προκαλεί την αύξηση των συνεκτικών συνιστωσών. Κατ' αναλογία, ένας κόμβος με παρόμοιες ιδιότητες ονομάζεται σημείο αρθρώσεως (articulation point) ή κόμβος αποκοπής (cut vertex).

Σε αντίθεση με τα μη κατευθυνόμενα, τα κατευθυνόμενα γραφήματα χαρακτηρίζονται ως ισχυρά συνεκτικά (strongly connected), μονόπλευρα συνεκτικά (unilaterally connected) ή ασθενή συνεκτικά (weakly connected). Ισχυρά συνεκτικά ονομάζονται τα γραφήματα στα οποία δύο οποιοιδήποτε κόμβοι συνδέονται με ένα κατευθυνόμενο μονοπάτι. Τα γραφήματα τα οποία δεν είναι ισχυρά συνεκτικά αποτελούνται από ισχυρά συνεκτικές συνιστώσες (strongly connected components), οι οποίες είναι οι μέγιστες συνιστώσες με την ιδιότητα της ισχυρής συνεκτικότητας. Μονόπλευρα συνεκτικά γραφήματα ονομάζονται τα γραφήματα στα οποία για οποιοδήποτε ζεύγος κόμβων x και y υπάρχει κατευθυνόμενο μονοπάτι είτε από το x στο y είτε από το y στο x. Ενώ τέλος ασθενή συνεκτικά ονομάζονται εκείνα για τα οποία για δύο οποιουσδήποτε κόμβους υπάρχει μονοπάτι, χωρίς να λάβουμε υπόψη την κατεύθυνση της ακμής, που τα συνδέει.

Όταν ένα γράφημα είναι άκυκλο και συνεκτικό ονομάζεται δέντρο (tree) (σχήμα 1.8), ενώ αν είναι άκυκλο και μη συνεκτικό ονομάζεται δάσος (forest). Μία ειδική κατηγορία κατευθυνόμενων γραφημάτων αποτελούν τα άκυκλα γραφήματα ή αλλιώς ΚΑΓ (κατευθυνόμενα άκυκλα γραφήματα) (σχήμα 1.9).



Σχήμα 1.8: Δέντρο.



Σχήμα 1.9: Κατευθυνόμενο Άκυκλο Γράφημα (ΚΑΓ).

Τα γραφήματα, κατευθυνόμενα ή μη, των οποίων οι ακμές συνδέονται με έναν αριθμό ονομάζονται βεβαρημένα (weighted) γραφήματα. Τα κατευθυνόμενα βεβαρημένα γραφήματα ονομάζονται επίσης και δίκτυα (networks). Βάρος ή κόστος μονοπατιού ονομάζεται το άθροισμα των βαρών των ακμών που συμμετέχουν στο μονοπάτι.

Κλείνοντας θα παραθέσουμε μερικές από τις γνωστότερες ποσοτικές συσχετίσεις.

Θεώρημα 1.1: Σε κάθε μη κατευθυνόμενο γράφημα $G = (V, E)$ ισχύει:

$$\sum_{u \in V} \deg(u) = 2|E|$$

στα κατευθυνόμενα γραφήματα $G = (V, E)$ ισχύει:

$$\sum_{u \in V} \text{in_deg}(u) = \sum_{u \in V} \text{out_deg}(u) = |E|$$

σε κάθε δέντρο $T = (V, E)$ ισχύει:

$$|E| = |V| - 1$$

και τέλος σε κάθε δάσος F το οποίο περιέχει t δέντρα ισχύει:

$$|E| = |V| - t$$

1.3 Αναπαράσταση Γραφημάτων

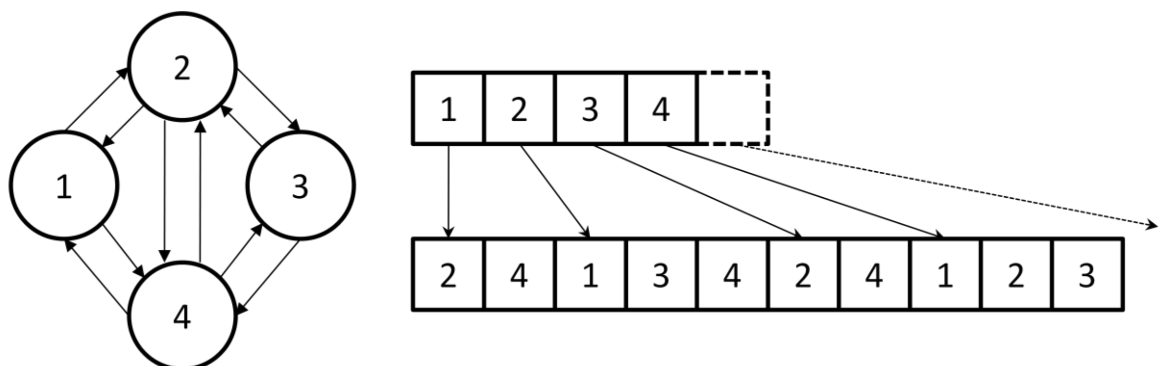
Η επεξεργασία ενός γραφήματος από έναν αλγόριθμο απαιτεί την αναπαράσταση του μέσω μίας κατάλληλης δομής δεδομένων. Επειδή ενδιαφερόμαστε αποκλειστικά για τα κατευθυνόμενα γραφήματα, γι' αυτό θα περιγράψουμε μόνο τις τεχνικές με τις οποίες αναπαριστώνται αυτά από ένα πρόγραμμα. Η ύπαρξη πολλών διαφορετικών τεχνικών για την αναπαράσταση των γραφημάτων δημιούργησε και την ανάγκη για την δημιουργία ενός μέτρου για την αποτελεσματικότητα της εκάστοτε τεχνικής. Για να περιγράψουμε πλήρως ένα γράφημα πρέπει να βρούμε ένα τρόπο να αναπαραστήσουμε το σύνολο των κόμβων V και το σύνολο των ακμών E. Έτσι, μία πιθανή τεχνική για την περιγραφή του γραφήματος δεν μπορεί να έχει πολυπλοκότητα, χρόνου ή χώρου, μικρότερη από $O(|V| + |E|)$. Στη συνέχεια, χωρίς βλάβη της γενικότητας, και κυρίως για προγραμματιστικούς λόγους, θα ακολουθήσουμε την σύμβαση πως οι κόμβοι θα φέρουν ως όνομα ακέραιους αριθμούς από το 1 έως το $|V|$.

1.3.1 Πίνακας Ακμών (Edge Matrix)

Για να εξασφαλίσουμε εύκολη πρόσβαση στις ακμές που εξέρχονται από ένα συγκεκριμένο κόμβο, μπορούμε να αποθηκεύσουμε τις εξερχόμενες ακμές ενός κόμβου σε ένα πίνακα. Αν δεν αποθηκευτούν επιπλέον πληροφορίες με τις ακμές, αυτός ο πίνακας θα περιέχει μόνο δείκτες για τους κόμβους προορισμού. Αν το

γράφημα είναι στατικό, δηλαδή δεν αλλάζει σε σχέση με τον χρόνο, τότε μπορούμε να συνδέσουμε όλους αυτούς τους μικρούς πίνακες σε έναν πίνακα ακμών E. Ένας επιπλέον πίνακας V αποθηκεύει τις θέσεις από τις οποίες ξεκινάει κάθε υποπίνακας (σχήμα 1.10). Όπως μπορεί εύκολα να καταλάβει κανένας αυτή η τεχνική αναπαράστασης του γραφήματος λειτουργεί άψογα και για κατευθυνόμενα και για μη κατευθυνόμενα γραφήματα. Αυτό συμβαίνει πολύ απλά αποθηκεύοντας για κάθε ακμή $x - y$, στα μη κατευθυνόμενα γραφήματα, την ακμή $x \rightarrow y$ και την ακμή $y \rightarrow x$, στα κατευθυνόμενα γραφήματα. Επίσης αυτή η τεχνική δουλεύει και για τα βεβαρημένα γραφήματα αποθηκεύοντας επιπλέον το βάρος της κάθε ακμής στον πίνακα E.

Αυτή η προσέγγιση απαιτεί χώρο $|V| + |E|$ για την αποθήκευση του γραφήματος, το οποίο, σύμφωνα με το μέτρο που ορίσαμε, είναι αποτελεσματικό. Όπως προαναφέραμε με αυτή την αναπαράσταση έχουμε άμεση πρόσβαση από τις ακμές στους προσκείμενους κόμβους. Η αντίστροφη πράξη όμως, δηλαδή την εύρεση των ακμών από τους προσκείμενους κόμβους, καθώς επίσης και ο έλεγχος αν δύο κόμβοι συνδέονται μεταξύ τους απαιτεί εξαντλητική αναζήτηση στον πίνακα των ακμών. Σημαντικό επίσης μειονέκτημα αυτής της τεχνικής είναι η δυσκολία στην αναπαράσταση δυναμικού γραφήματος, επειδή η πρόσθεση ή η αφαίρεση ακμών απαιτεί αναδιάταξη ολόκληρων των πινάκων.

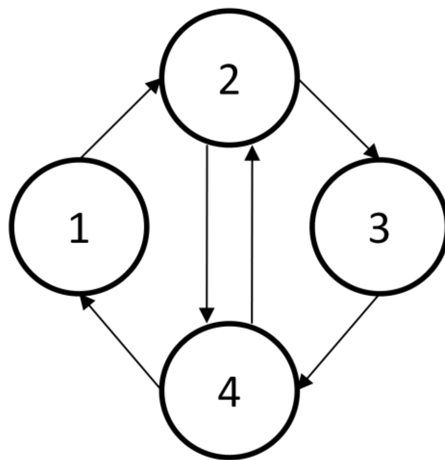


Σχήμα 1.10: Αναπαράσταση με πίνακα ακμών. Αριστερά φαίνεται το γράφημα, δεξιά πάνω ο πίνακας με τους δείκτες και κάτω ο πίνακας με τις ακμές.

1.3.2 Πίνακας Γειτνίασης (Adjacency Matrix)

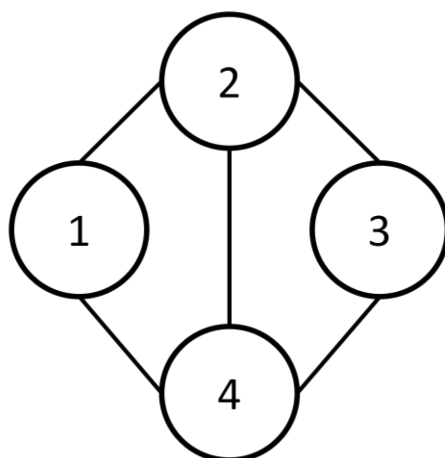
Αυτή η προσέγγιση απαιτεί την χρήση ενός δισδιάστατου πίνακα A (σχήμα 1.11), μεγέθους $V \times V$, για την αποθήκευση των ακμών του γραφήματος. Η τιμή "1" στο στοιχείο $A[x][y]$ του πίνακα δηλώνει την ύπαρξη μιας ακμής που ενώνει τους κόμβους x και y, ενώ η τιμή "0" δηλώνει την απουσία ακμής. Αυτή η προσέγγιση λειτουργεί τόσο για τα κατευθυνόμενα όσο και για τα μη κατευθυνόμενα γραφήματα, με την λεπτομέρεια ότι στα μη κατευθυνόμενα γραφήματα ο πίνακας γειτνίασης είναι συμμετρικός ως προς την διαγώνιο (σχήμα 1.12). Για τα βεβαρημένα γραφήματα αντί για δυαδικό πίνακα τιμών χρησιμοποιούμε ένα γενικό πίνακα τιμών, αποθηκεύοντας το βάρος της κάθε ακμής (σχήμα 1.13).

Με τον πίνακα γειτνίασης μπορούμε να προσθέσουμε και να αφαιρέσουμε ακμές σε χρόνο $O(1)$, όπως επίσης στον ίδιο χρόνο μπορούμε να ελέγξουμε αν υπάρχει ακμή μεταξύ δύο κόμβων. Ο πίνακας γειτνίασης αποτελεί την πιο εύκολη προσέγγιση όσον αφορά το προγραμματιστικό κομμάτι. Στα μειονεκτήματα αυτής της τεχνικής συγκαταλέγεται το γεγονός ότι απαιτεί $V \times V$ χώρο για την αποθήκευση του γραφήματος κάτι που την καθιστά ιδιαίτερα αναποτελεσματική, κυρίως στα μεγάλα και αραιά γραφήματα. Επιπλέον, σε μερικούς σημαντικούς αλγορίθμους χρειάζεται να έχουμε γρήγορη πρόσβαση στις ακμές που ξεκινούν από ένα συγκεκριμένο κόμβο. Αυτό καθιστά αυτή τη τεχνική αναπαράστασης ανάρμοστη, αφού για να εξετάσουμε ποιες ακμές εξέρχονται από ένα κόμβο χρειαζόμαστε χρόνο $O(|V|)$. Έτσι οι αλγόριθμοι της αναζήτησης κατά βάθος και της αναζήτησης κατά πλάτος χρειάζονται χρόνο $O(|V|^2)$ για να εκτελεστούν, την στιγμή που για τους ίδιους μπορούμε να επιτύχουμε χρόνο $O(|V| + |E|)$ αναπαριστώντας το γράφημα με πίνακα ακμών ή με λίστες γειτνίασης. Τελευταίο μειονέκτημα αυτής της αναπαράστασης είναι ότι η προσθήκη ή η αφαίρεση ενός κόμβου αποτελεί μία πολύ χρονοβόρο διαδικασία, αφού ολόκληρος ο πίνακας πρέπει να ξανακατασκευαστεί από την αρχή.



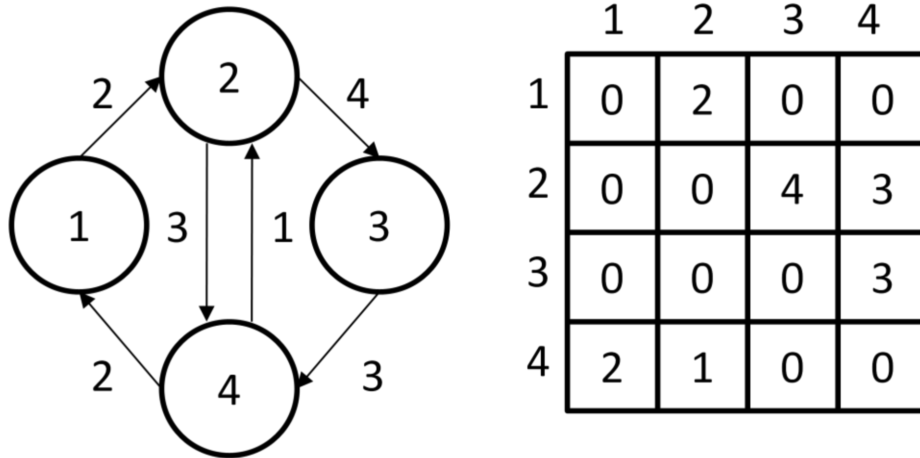
	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	0	0	0	1
4	1	1	0	0

Σχήμα 1.11: Αναπαράσταση γραφήματος με πίνακα γειτνίασης.



	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	1
4	1	1	1	0

Σχήμα 1.12: Αναπαράσταση μη κατευθυνόμενου γραφήματος με πίνακα γειτνίασης.



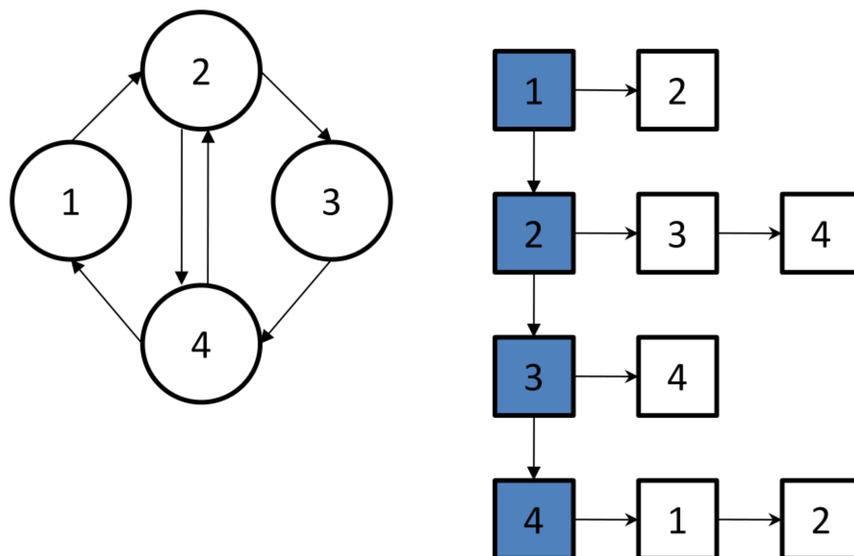
Σχήμα 1.13: Αναπαράσταση βεβαρυμμένου γραφήματος με πίνακα γειτνίασης.

1.3.3 Λίστες Γειτνίασης (Adjacency Lists)

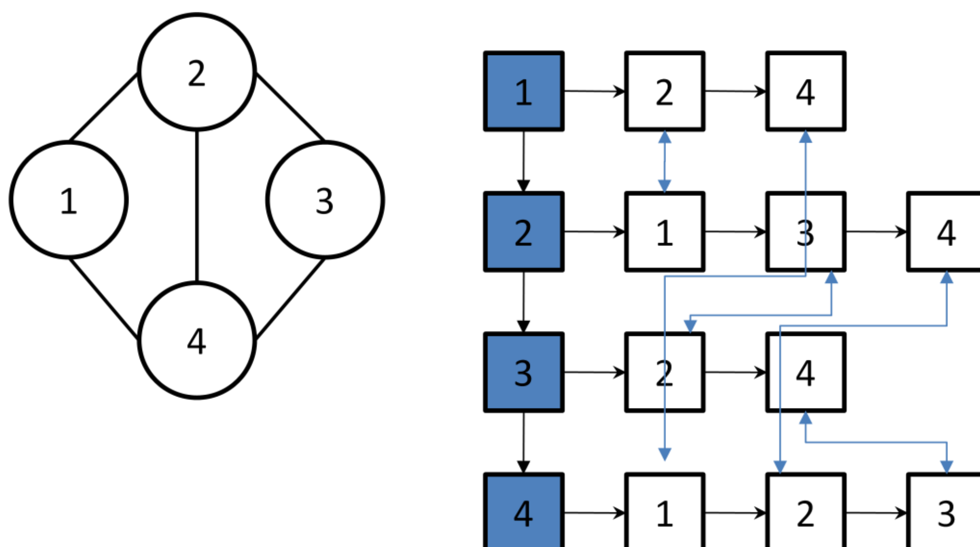
Οι λίστες γειτνίασης (σχήμα 1.14) αποτελούνται από έναν πρωτεύον πίνακα ή μία πρωτεύουσα λίστα και από δευτερεύουσες λίστες. Κάθε στοιχείο της πρωτεύουσας λίστας αντιστοιχεί σε ένα κόμβο του γραφήματος, ενώ κάθε δευτερεύουσα λίστα είναι συνδεδεμένη με ένα στοιχείο της πρωτεύουσας λίστας. Στις δευτερεύουσες λίστες αποθηκεύονται οι κόμβοι στους οποίους καταλήγουν ακμές από τον κόμβο με τον οποίο είναι συνδεδεμένη η δευτερεύουσα λίστα, δηλαδή το στοιχείο της πρωτεύουσας λίστας. Οι λίστες γειτνίασης αποτελούν μία από τις ιδανικότερες επιλογές για αραιά και δυναμικά γραφήματα. Στα μη κατευθυνόμενα γραφήματα, κάθε ακμή πρέπει να αποθηκεύεται και στις δύο δευτερεύουσες λίστες των κόμβων που συνδέουν και για λόγους απόδοσης, κυρίως κατά την διαγραφή, οι δύο εγγραφές συνηθίζεται να συνδέονται με ένα δείκτη (σχήμα 1.15). Η λίστα γειτνίασης τέλος μπορεί να χρησιμοποιηθεί εύκολα και στα βεβαρυμμένα γραφήματα αρκεί σε κάθε εγγραφή ακμής να αποθηκεύουμε και το βάρος της (σχήμα 1.16).

Από την δομή της λίστας γειτνίασης προκύπτει ότι ο χώρος που απαιτεί για την αποθήκευση είναι $|V| + |E|$. Το αποτέλεσμα είναι ότι έχουμε πιο συμπυκνωμένη αποθήκευση του γραφήματος σε σχέση με τον πίνακα γειτνίασης, αλλά η διαφορά αυτή εξαλείφεται όσο το γράφημα γίνεται πιο πυκνό. Ένα ακόμα πλεονέκτημα αυτής

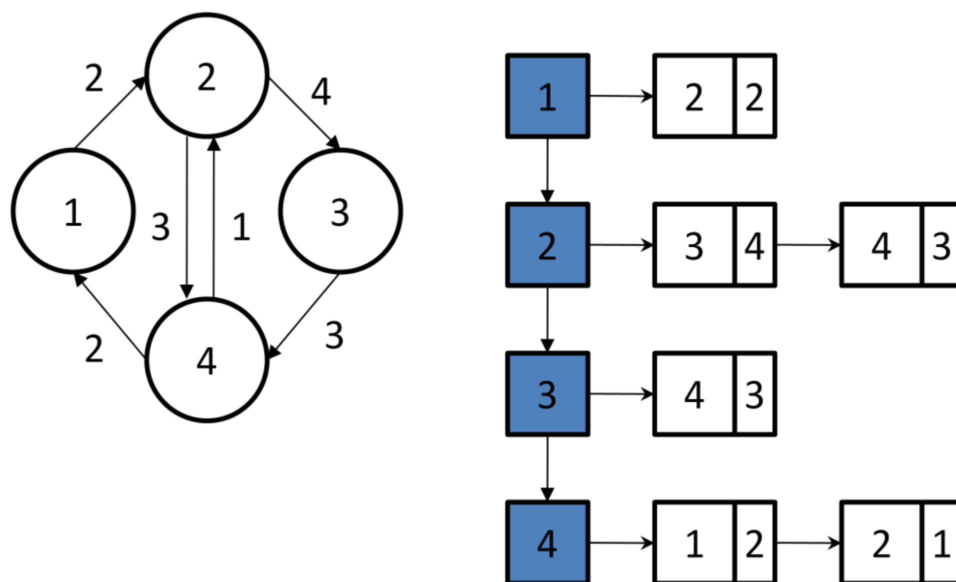
της προσέγγισης είναι ότι η εύρεσης των γειτονικών κόμβων ενός κόμβου x είναι πράξη γραμμική προς το βαθμό του κόμβου x, το οποίο αποτελεί σημαντικό πλεονέκτημα για κάποιους αλγόριθμους. Από την άλλη, η πρόσθεση και η αφαίρεση μίας ακμής είναι πιο δύσκολή σε σχέση με την τεχνική του πίνακα γειτνίασης, όπως επίσης και η αποτροπή παράλληλων ακμών και η διαπίστωση αν δύο κόμβοι διασυνδέονται ή όχι. Για την αφαίρεση μίας ακμής, είθισται να χρησιμοποιούνται βοηθητικές δομές λεξικού, όπως εκείνη ενός πίνακα κατακερματισμού, για τον εντοπισμό της ακμής σε, σχεδόν, σταθερό χρόνο.



Σχήμα 1.14: Αναπαράσταση κατευθυνόμενου γραφήματος με λίστα γειτνίασης.



Σχήμα 1.15: Αναπαράσταση μη κατευθυνόμενου γραφήματος με λίστα γειτνίασης. Τα διπλά γαλάζια βέλη είναι οι δείκτες αλληλοσυσχέτισης.

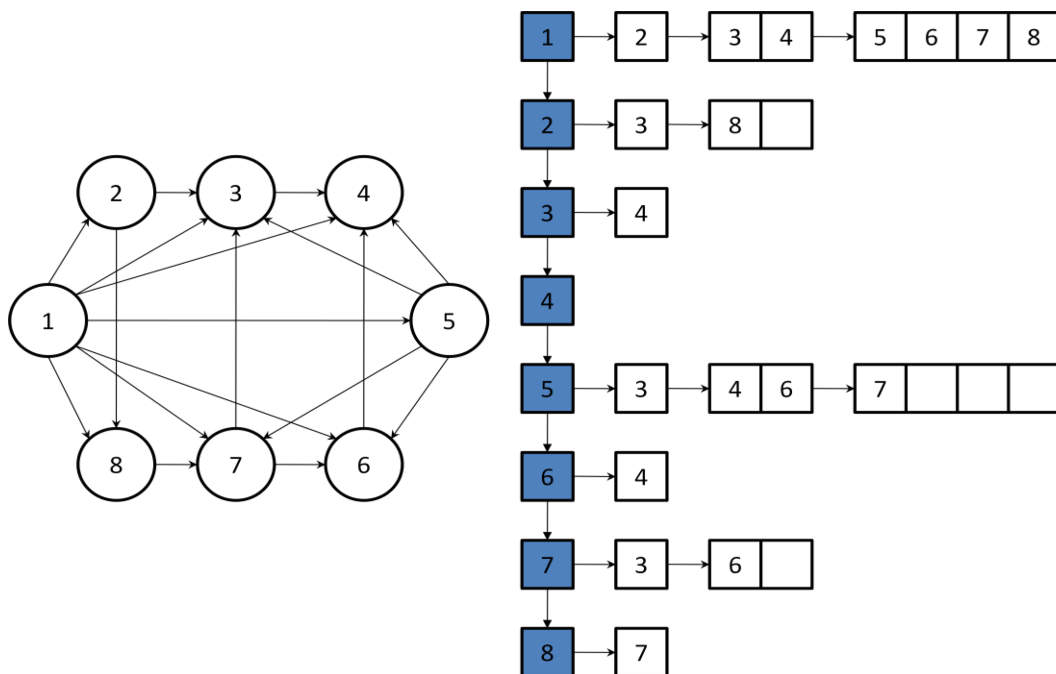


Σχήμα 1.16: Αναπαράσταση βεβαρυμμένου κατευθυνόμενου γραφήματος με λίστα γειτνίασης.

1.3.4 Συνδεδεμένοι Πίνακες Γειτνίασης (Linked Adjacency Arrays)

Οι συνδεδεμένοι πίνακες γειτνίασης (σχήμα 1.17) αποτελούν μία παραλλαγή της λίστας γειτνίασης. Αποτελούνται από μία πρωτεύουσα λίστα και από δευτερεύουσες λίστες με την μόνη διαφορά ότι οι δευτερεύουσες λίστες οργανώνονται σε πίνακες με μεταβαλλόμενο μέγεθος. Ουσιαστικά, αντί οι κόμβοι σε μία δευτερεύουσα συνδεδεμένη λίστα να έχουν χώρο για την αποθήκευση μίας ακμής μόνο, δεσμεύουν χώρο για ένα πίνακα και αποθηκεύουν εκεί τις ακμές. Όταν ο πίνακας γεμίσει, στη δευτερεύουσα λίστα προστίθεται ένας ακόμα κόμβος και ο πίνακας, αυτού του κόμβου, δεσμεύει διπλάσιο χώρο από τον πίνακα του προηγούμενου κόμβου.

Αυτή η προσέγγιση εκμεταλλεύεται όλα τα πλεονεκτήματα των λιστών γειτνίασης και επιπλέον κάνει την πράξη της προσπέλασης των ακμών πιο γρήγορη, σε σχέση με την συνδεδεμένη λίστα, αφού πλέον η προσπέλαση γίνεται και σε πίνακες εκτός από κόμβους σε συνδεδεμένη λίστα. Σαφέστατα αυτή η αναπαράσταση κάνει, σχεδόν πάντα, μία μικρή σπατάλη χώρου που είναι ανάλογη με τον αριθμό των ακμών που συνδέονται σε έναν κόμβο. Οι συνδεδεμένοι πίνακες γειτνίασης είναι κατάλληλοι για δυναμικά γραφήματα.

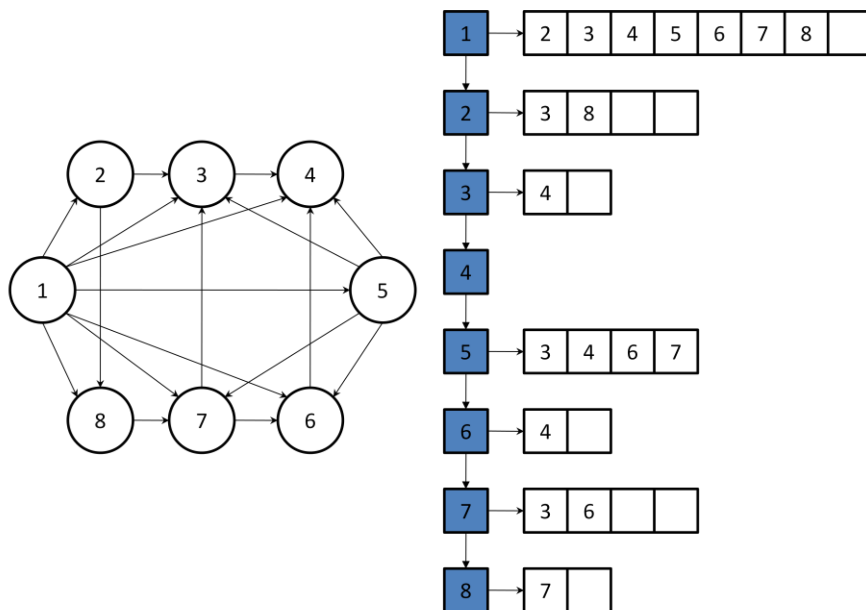


Σχήμα 1.17: Αναπαράσταση κατευθυνόμενου γραφήματος με Συνδεδεμένους Πίνακες Γειτνίασης.

1.3.5 Δυναμικοί Πίνακες Γειτνίασης (Dynamic Adjacency Arrays)

Οι δυναμικοί πίνακες γειτνίασης (σχήμα 1.18) προέκυψαν από τον συνδυασμό δύο δομών, του πίνακα ακμών και των λιστών γειτνίασης. Αποτελούνται από μία συνδεδεμένη λίστα, της οποίας κάθε στοιχείο x αντιπροσωπεύει ένα κόμβο και είναι συνδεδεμένο με έναν δυναμικό πίνακα. Σε αυτόν τον δυναμικό πίνακα αποθηκεύονται οι κόμβοι στους οποίους καταλήγουν οι ακμές που ξεκινούν από τον x. Όταν ο δυναμικός πίνακας ενός στοιχείου της λίστας γεμίσει, τότε δεσμεύεται χώρος για έναν πίνακα διπλάσιου μεγέθους, αντιγράφονται όλα τα στοιχεία από τον παλιό πίνακα στον καινούριο και τέλος το στοιχείο της λίστας συνδέεται με τον καινούριο πίνακα. Ο παλιός πίνακας δεν έχει καμία χρήση και για να εξοικονομήσουμε αποθηκευτικό χώρο μπορούμε να τον διαγράψουμε.

Η νέα τεχνική, εφόσον αποθηκεύει τις ακμές μόνο σε πίνακες, εκτελεί πιο γρήγορα τις εντολές προσπέλασης ακόμα και από την τεχνική των συνδεδεμένων πινάκων γειτνίασης. Όπως είναι φανερό, και σε αυτή τη τεχνική υπάρχει μία σπατάλη στον χώρο αποθήκευσης. Ο χρόνος εισαγωγής μίας νέας ακμής στην χειρότερη περίπτωση είναι $O(|V|)$ αλλά κατά μέσο όρο $O(1)$. Παρόλα αυτά και αυτή η τεχνική εκμεταλλεύεται όλα τα πλεονεκτήματα των λιστών γειτνίασης.



Εικόνα 1.18: Αναπαράσταση κατευθυνόμενου γραφήματος με Δυναμικούς Πίνακες Γειτνίασης.

1.4 Διαπεράσεις Κατευθυνόμενων Γραφημάτων

Με τον όρο διαπέραση ή διάβαση του γραφήματος χαρακτηρίζεται η εξαντλητική εξερεύνηση του γραφήματος, με την εξέταση όλων των κόμβων και το ακμών του. Αν και οι εξερευνήσεις ενός γραφήματος σαν μέθοδοι είναι σχετικά απλοί και κατανοητοί, παρ’ όλα αυτά είναι ισχυρές και ικανές να ανακαλύψουν ιδιότητες, όπως λ.χ. ύπαρξη συνεκτικότητας ή κύκλων στα υπό εξερεύνηση γραφήματα.

Η συνεκτικότητα μεταξύ κόμβων αποτελεί ένα πολύ βασικό αλγοριθμικό ερώτημα. Ας υποθέσουμε ότι έχουμε το γράφημα $G = (V, E)$ και δύο συγκεκριμένου κόμβους x και y. Χρειαζόμαστε έναν αποδοτικό αλγόριθμο ο οποίος θα απαντά στο ερώτημα αν υπάρχει μονοπάτι μεταξύ αυτών των δύο κόμβων. Αυτό το ερώτημα αποτελεί το πρόβλημα της συνεκτικότητας του x και του y ($x - y$ connectivity). Το πρόβλημα αυτό είναι εύκολο για μικρά γραφήματα, αλλά όσο το γράφημα μεγαλώνει, το πρόβλημα γίνεται πιο δύσκολο. Οι αλγόριθμοι που επιλύουν αυτό το πρόβλημα βασίζονται στις διαπεράσεις των γραφημάτων.

Οι μέθοδοι που περιγράφουμε είναι δύο, η Αναζήτηση Κατά Πλάτος (ΑΚΠ)(Breadth First Search - BFS) και η Αναζήτηση Κατά Βάθος (ΑΚΒ)(Depth First Search - DFS). Και οι δύο είναι κατάλληλες για την εξερεύνηση και των μη κατευθυνόμενων και των κατευθυνόμενων γραφημάτων. Εν τούτοις, στη συνέχεια εξετάσουμε τις δύο μεθόδους μόνο για κατευθυνόμενα γραφήματα, αφού η εφαρμογή τους στα μη κατευθυνόμενα γραφήματα είναι παρόμοια.

1.4.1 Αναζήτηση Κατά Πλάτος (Breadth First Search)

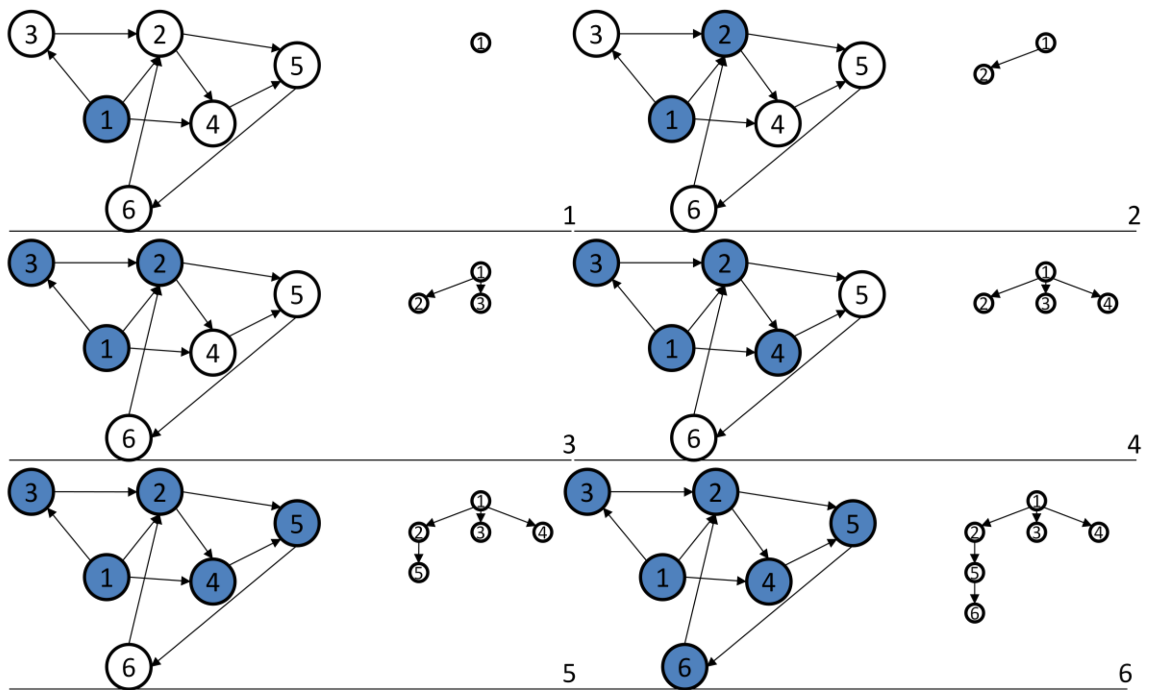
Για την έναρξη της διαπέρασης του γραφήματος χρειάζεται να επιλέξουμε ένα κόμβο ως αφετηριακό κόμβο, δηλαδή σαν την αρχή της αναζήτησης. Στην αναζήτηση κατά πλάτος ή αναζήτηση με προτεραιότητα πλάτους, η εξερεύνηση ξεκινά από τον κόμβο έναρξης προς κάθε δυνατή διεύθυνση, προσθέτοντας κόμβους κατά επίπεδα. Κάθε κόμβος που συναντάται επεξεργάζεται με FIFO(First In First Out) σειρά (σχήμα 1.19). Στο επίπεδο ένα βρίσκεται ο κόμβος έναρξης, ενώ οι κόμβοι οι οποίοι συνδέονται με αυτόν τον κόμβο, με ακμή, βρίσκονται στο επίπεδο δύο. Η αναζήτηση συνεχίζεται

στους κόμβους οι οποίοι συνδέονται με του κόμβους του δευτέρου επιπέδου. Αυτοί οι κόμβοι αποτελούν τους κόμβους του τρίτου επιπέδου. Συνεχίζοντας με τον ίδιο τρόπο καταλήγουμε στον τερματισμό της αναζήτησης, όπου πλέον δεν υπάρχει νέος κόμβος να συναντήσουμε, ή όπως λέμε συχνά να ανακαλύψουμε.

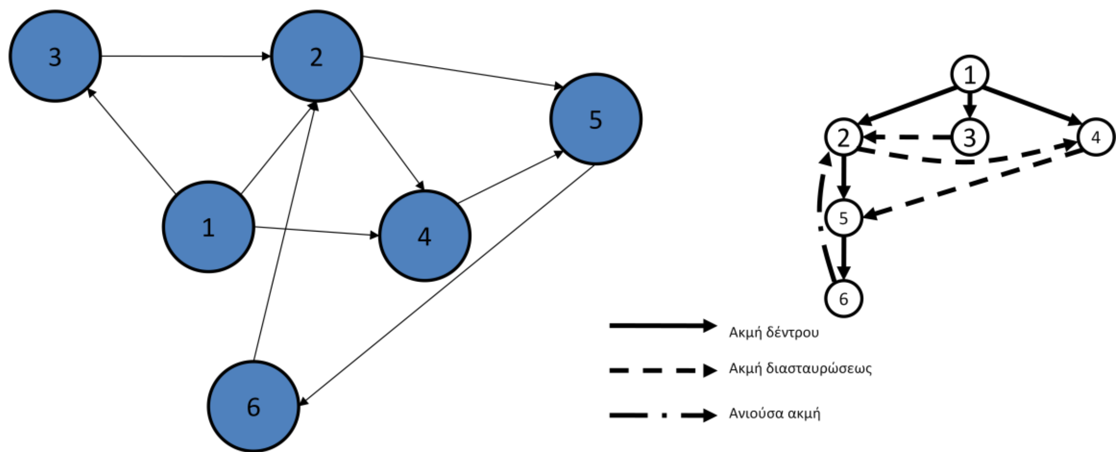
Η ΑΚΠ έχει πολυπλοκότητα χρόνου $O(|V|+|E|)$, αν η αναπαράσταση του γραφήματος είναι η κατάλληλη, όπου V είναι ο αριθμός των κόμβων και E είναι ο αριθμός των ακμών. Δηλαδή από την στιγμή που θα ξεκινήσει χρειάζεται χρόνο γραμμικό ως προς τον αριθμό των κόμβων και τον αριθμό των ακμών για να τερματίσει, αφού στην χειρότερη περίπτωση πρέπει να προσπελάσει όλες τις ακμές και όλους τους κόμβους. Ο χώρος που απαιτείται για να αποθηκευτεί το δέντρο της αναζήτησης κατά πλάτος έχει πολυπλοκότητα $O(|V|+|E|)$.

Μία βασική ιδιότητα της ΑΚΠ είναι ότι παράγει ένα δέντρο T με ρίζα τον κόμβο έναρξης (σχήμα 1.20) για το σύνολο των κόμβων που μπορούν να προσπελαστούν από τον κόμβο έναρξης. Οι ακμές του γραφήματος της ΑΚΠ αποτελούνται από ακμές δέντρου (tree edges), εάν αποτελούν ακμές του T, από ακμές διασταυρώσεως (cross edges), εάν δεν παρουσιάζουν σχέση προγόνου – απογόνου ως προς το δέντρο T, και από ανιούσες ακμές (back edges), εάν συνδέουν κόμβους με σχέση απογόνου – προγόνου.

Στο δέντρο της ΑΚΠ δεν υπάρχουν κατιούσες ακμές (forward edges), δηλαδή ακμές οι οποίες συνδέουν έναν πρόγονο με έναν απόγονο. Αυτή η ιδιότητα προκύπτει από τον τρόπο που κατασκευάζεται το δέντρο της ΑΚΠ. Η ΑΚΠ ανακαλύπτει τα συντομότερα, ως προς το πλήθος των ακμών, μονοπάτια από τον κόμβο έναρξης, τα οποία μπορούν να προσπελαστούν από αυτόν. Τέλος οι κύκλοι στο δέντρο της ΑΚΠ, ενός κατευθυνόμενου γραφήματος, δημιουργούνται από τις ανιούσες ακμές.



Σχήμα 1.19: Αναζήτηση κατά Πλάτος.



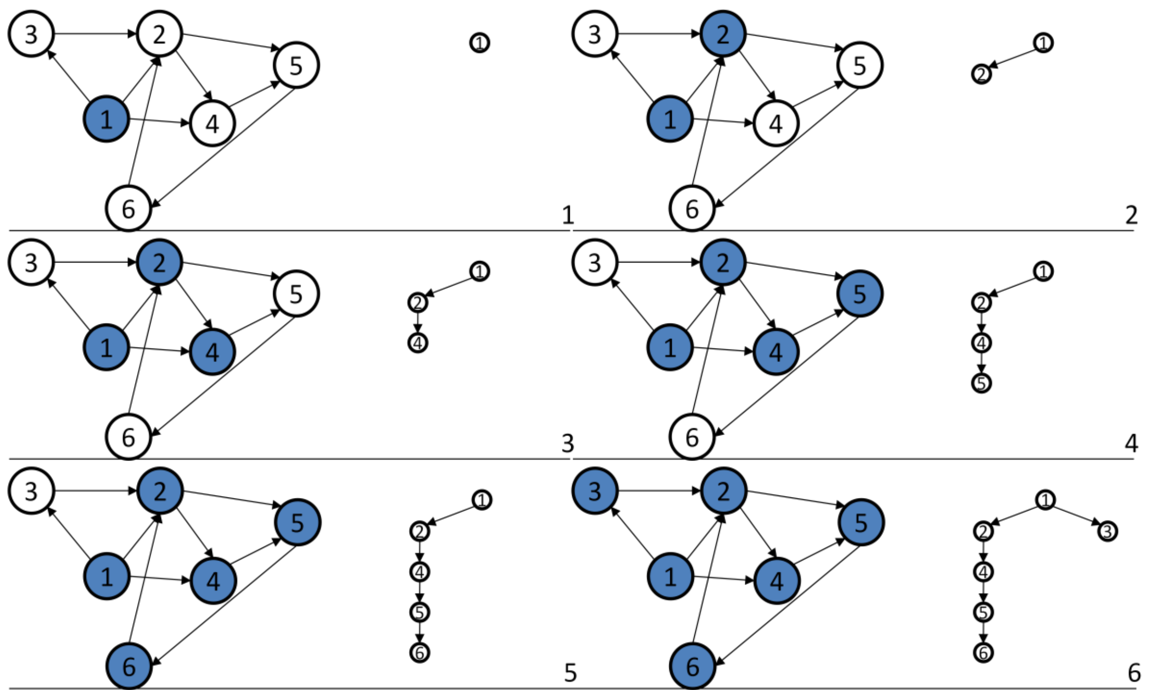
Σχήμα 1.20: Αριστερά το γράφημα που επεξεργαζόμαστε και δεξιά το δέντρο της ΑΚΠ.

1.4.2 Αναζήτηση Κατά Βάθος (Depth First Search)

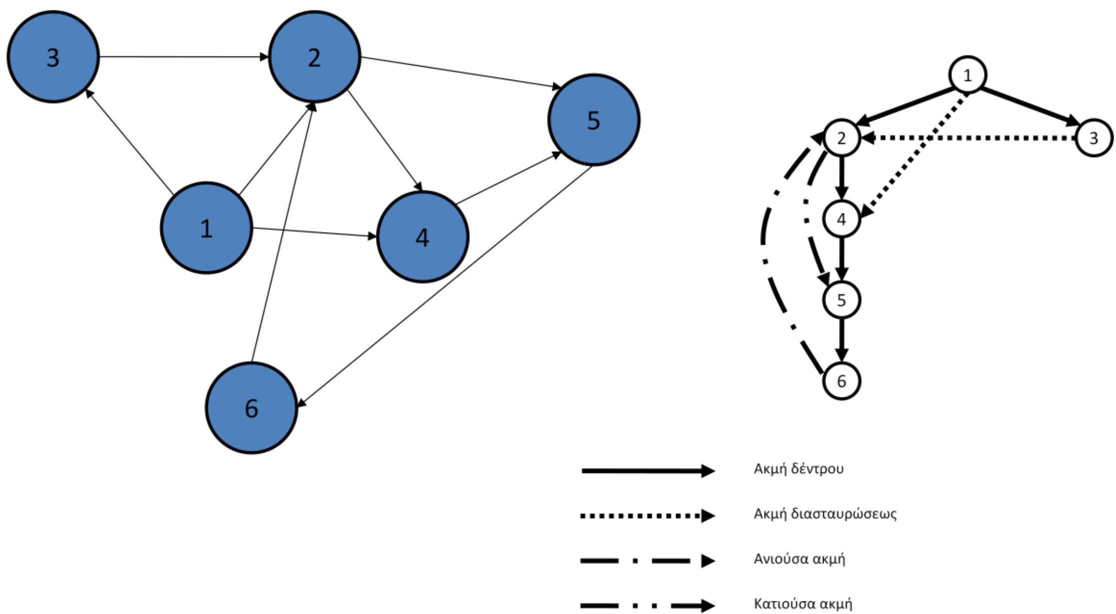
Στην αναζήτηση κατά βάθος ή αναζήτηση με προτεραιότητα στο βάθος, η εξερεύνηση ξεκινάει, όπως και στην ΑΚΠ, επιλέγοντας έναν κόμβο για αφετηρία. Ο αλγόριθμος αυτός ονομάζεται αναζήτηση κατά βάθος γιατί πηγαίνει όσο πιο βαθιά μπορεί, μέχρι να συναντήσει αδιέξοδο, πριν επιστρέψει πίσω. Κάθε κόμβος που ανακαλύπτεται επεξεργάζεται με σειρά LIFO (Last In First Out) (σχήμα 1.21). Ξεκινώντας από την αφετηρία επισκεπτόμαστε τον πρώτο κατά σειρά κόμβο. Στη συνέχεια προχωράμε παραπέρα από τον κόμβο που βρισκόμαστε και συνεχίζουμε έτσι μέχρι να συναντήσουμε αδιέξοδο. Ως αδιέξοδο ονομάζουμε την κατάσταση στην οποία ο κόμβος που βρισκόμαστε ή δεν έχει γείτονες ή όλοι του οι γείτονες έχουν ανακαλυφθεί. Αφού φτάσουμε σε μία τέτοια κατάσταση οδηγούμαστε προς τα πίσω έως ότου συναντήσουμε ένα κόμβο με ανεξερεύνητους γείτονες.

Η ΑΚΒ όπως και η ΑΚΠ έχει πολυπλοκότητα χρόνου και πολυπλοκότητα χώρου $O(|V| + |E|)$, πάντα αν εφαρμόσουμε μία κατάλληλη τεχνική αναπαράστασης του γραφήματος. Παρατηρούμε ότι οι δύο αναζητήσεις έχουν θεμελιώδεις ομοιότητες, στο γεγονός ότι και οι δύο δομούν το συνεκτικό σύνολο που περιέχει τον κόμβο έναρξης σε ποιοτικά παρόμοια επίπεδα απόδοσης. Όμως, αν και επισκέπτονται το ίδιο ακριβώς σύνολο κόμβων, η σειρά είναι, συνήθως, πολύ διαφορετική μεταξύ της μίας αναζήτησης και της άλλης. Στην ΑΚΒ ο αλγόριθμος αρχικά επισκέπτεται πολύ απομακρυσμένους κόμβους, δημιουργώντας μεγάλες διαδρομές, πριν επιστρέψει και εξερευνήσει πλησιέστερους.

Η ΑΚΒ παράγει το δέντρο T, με ρίζα τον κόμβο έναρξης, για το σύνολο των κόμβων που μπορούν να προσπελαστούν από τον κόμβο έναρξης (σχήμα 1.22). Οι ακμές του γραφήματος της ΑΚΒ αποτελούνται από ακμές δέντρου, εάν αποτελούν ακμές του T, ανιούσες ακμές, αν ενώνουν κόμβους με σχέση απογόνου – προγόνου, κατιούσες ακμές, εάν ενώνει κόμβους με σχέση προγόνου – απογόνου, και ακμές διασταυρώσεως, αν οι κόμβοι που συνδέονται δεν παρουσιάζουν σχέση προγόνου – απογόνου. Μία βασική ιδιότητα, εξ ορισμού της ΑΚΒ, είναι ότι οι ακμές διασταυρώσεως κατευθύνονται πάντα από δεξιά προς τα αριστερά. Τέλος η ανιούσες ακμές και σε αυτή τη τεχνική είναι η μοναδική αιτία δημιουργίας κύκλων.



Σχήμα 1.21: Αναζήτηση Κατά Βάθος.



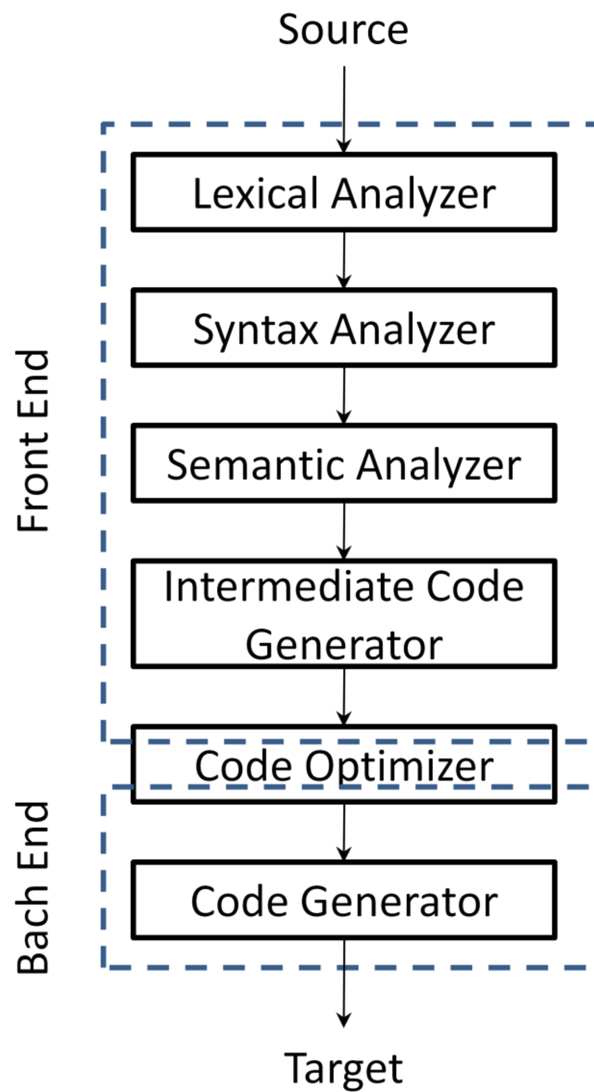
Σχήμα 1.22: Αριστερά το γράφημα που επεξεργαζόμαστε και δεξιά το δέντρο της ΑΚΒ.

2 ΒΑΣΙΚΑ ΜΠΛΟΚ ΚΑΙ ΓΡΑΦΗΜΑΤΑ ΡΟΗΣ ΕΛΕΓΧΟΥ

Αφού αναφέραμε την βασική ορολογία σε ότι αφορά τα γραφήματα, σε αυτό το κεφάλαιο θα εξετάσουμε τον τρόπο με τον οποίο μπορούμε να μετασχηματίσουμε τον κώδικα ενός προγράμματος σε γραφήματα ροής ελέγχου (Control Flow Graph). Αρχικά θα μελετήσουμε την μορφή τριών διευθύνσεων κώδικα (three-address code), μία ενδιάμεση μορφή στην οποία μεταφράζεται ο κώδικας πριν μεταγλωττιστή σε γλώσσα μηχανής. Στη συνέχεια θα δείξουμε πως χωρίζεται ο ενδιάμεσος κώδικας, ο οποίος είναι γραμμένος σε μορφή κώδικα τριών διευθύνσεων, σε βασικά μπλοκ (basic block), τα οποία αποτελούν τους κόμβους του γραφήματος ροής ελέγχου. Στο τέλος θα δείξουμε πως με τα διάφορα άλματα (υπό όρους ή χωρίς) στον ενδιάμεσο κώδικα μεταφέρεται η ροή ελέγχου μεταξύ των βασικών μπλοκ. Επιπλέον, επειδή αυτή η εργασία μελετά αλγορίθμους που χρησιμοποιούνται στη βελτιστοποίηση του κώδικα εκτός βασικών μπλοκ, κρίθηκε χρήσιμο να αναφέρουμε μερικούς βασικούς τρόπους βελτιστοποίησης εντός των βασικών μπλοκ. Το συγκεκριμένο κεφάλαιο βασίζεται στο βιβλίο των Aho, Lam, Sethi και Ullman [1].

2.1 Κώδικας Τριών Διευθύνσεων (Three-address code)

Οι περισσότεροι μεταγλωττιστές, όταν μεταγλωττίζουν τον πηγαίο κώδικα ενός προγράμματος, αρχικά τον μεταφράζουν σε μία ενδιάμεση γλώσσα, δημιουργώντας έναν ενδιάμεσο κώδικα, ο οποίος είναι κατάλληλος για ενέργειες βελτιστοποίησης. Αυτή η μετάφραση προσφέρει μία χαμηλότερου επιπέδου αναπαράσταση της εσωτερικής δομής του αρχικού προγράμματος. Η φάση της παραγωγής του ενδιάμεσου κώδικα συνδέει τις δύο βασικές λειτουργίες του μεταγλωττιστή, εκείνη της ανάλυσης (front-end) και εκείνη της σύνθεσης (back-end) (σχήμα 2.1). Η ενδιάμεση γλώσσα εκτός του ότι προσφέρει βελτιστοποίηση στον κώδικα ανεξαρτήτου αρχιτεκτονικής, οδηγεί και στην δημιουργία μεταγλωττιστών για καινούργιες αρχιτεκτονικές αλλάζοντας μόνο το τμήμα σύνθεσης. Επιπλέον, είναι απαραίτητο να αναφέρουμε ότι οι ενδιάμεσες γλώσσες πρέπει να είναι ικανές να εκφράζουν τις λειτουργίες των γλωσσών υψηλού επιπέδου όπως επίσης πρέπει να είναι ικανές να μεταφράζονται εύκολα και αποδοτικά στην εκάστοτε γλώσσα μηχανής (σχήμα 2.2). Ο κώδικας τριών διευθύνσεων αποτελεί μία από τις πιο δημοφιλής ενδιάμεσες γλώσσες. Μία βελτίωση του είναι η δομή στατικής και μοναδικής ανάθεσης (single static assignment) η οποία θα συζητηθεί εκτενέστερα στη συνέχεια.



Σχήμα 2.1: Οι λειτουργίες του μεταγλωττιστή και ο διαχωρισμός του τμήματος ανάλυσης και του τμήματος σύνθεσης.

Σχήμα 2.2: Ο κώδικας οποιασδήποτε γλώσσας μεταφράζεται στον ίδιο ενδιάμεσο κώδικα πριν μεταγλωττιστή στην εκάστοτε γλώσσα μηχανής.

Οι εντολές τριών διευθύνσεων μοιάζουν με τις εντολές assembly. Το όνομα προέρχεται από το γεγονός ότι η πλειοψηφία των εντολών αποτελούνται από τρεις διευθύνσεις. Δύο για τους τελεστές και μία για την αποθήκευση του αποτελέσματος. Οι διευθύνσεις μπορεί να είναι διευθύνσεις σταθεράς, διευθύνσεις συμβολικών ονομάτων ή διευθύνσεις προσωρινών συμβολικών ονομάτων. Τα συμβολικά ονόματα αποτελούν ονόματα του πηγαίου προγράμματος, τα οποία για ευκολία επιτρέπεται να εμφανίζονται σαν διευθύνσεις, ενώ οι προσωρινές διευθύνσεις είναι διευθύνσεις που παράγονται από τον μεταγλωττιστή. Κάθε συμβολικό όνομα μπορεί να αποθηκεύσει είτε μία τιμή είτε μία διεύθυνση στη μνήμη (r-value και l-value αντίστοιχα). Κάθε εντολή μπορεί να περιγραφεί από την τετράδα $[operator, operand\ 1, operand2, result]$ και κάθε δήλωση είναι της μορφής:

$$result\ = operand1\ operator\ operand2$$

Κάθε τέτοια τετράδα σημειώνεται με μία ετικέτα (label), που συνήθως είναι ο αριθμός της εντολής. Πιο συγκεκριμένα οι εντολές του κώδικα τριών διευθύνσεων μπορεί να έχουν τις εξής μορφές:

1. Εντολές ανάθεσης: $x\ =\ y\ op\ z$

   όπου τα x, y και z είναι διευθύνσεις από τις οποίες μόνο τα y και z μπορεί να είναι σταθερές. Το op είναι ένας δυαδικός αριθμητικός ή λογικός τελεστής. Σε περίπτωση που η πράξη έχει περισσότερους από δύο τελεστές μπορούμε να την διαμελίσουμε σε επιπλέον πράξεις. Παραδείγματος χάριν η εντολή

   $$w = 5 * (x - y + z)$$

   μπορεί να αναπαρασταθεί, με τον κώδικα τριών διευθύνσεων, ως:

   $$t_1 = y + z$$

   $$t_2 = x - t_1$$

   $$w = 5 * t_2$$

όπου τα t_1 και t_2 είναι προσωρινές μεταβλητές που παράγονται από τον μεταγλωττιστή.

2. Εντολές ανάθεσης: $x = op\, y$
   όπου τα x και y είναι διευθύνσεις και το op είναι ένας μοναδιαίος αριθμητικός ή λογικός τελεστής.
3. Εντολές αντιγραφής: $x = y$
   όπου το x είναι μία διεύθυνση ενώ το y μπορεί να είναι είτε διεύθυνση είτε σταθερά.
4. Διακλαδώσεις χωρίς συνθήκη: $goto\ L$
   όπου το L είναι μία ετικέτα με την εντολή-στόχο μετά το άλμα.
5. Διακλάδωση υπό συνθήκη: $if(True\ or\ False)\ x\ goto\ L$
   όπου το x είναι ένα συμβολικό όνομα που παίρνει την τιμή $True$ ή $False$ και το L μία ετικέτα με την εντολή-στόχο μετά το άλμα.
6. Διακλάδωση υπό συνθήκη: $if(True\ or\ False)\ x\ relop\ y\ goto\ L$
   όπου το $relop$ είναι ένας σχεσιακός τελεστής (π.χ. $=, >, <, \leq, \geq$) και το L μία ετικέτα με την εντολή-στόχο μετά το άλμα.
7. Εντολές κλήσης και επιστροφής διαδικασιών:
   Η εντολή $param\ x$ δηλώνει την παράμετρο x.
   Η εντολή $call\ p, n$ δηλώνει την κλήση της εντολής p με n παραμέτρους.
   Η εντολή $x = call\ p, n$ κάνει κλήση της εντολή p, η οποία έχει n παραμέτρους, και κάνει ανάθεση της τιμής που επιστρέφει η εντολή στο x.
   Η εντολή $return\ x$ δηλώνει την επιστροφή της τιμής x.
8. Εντολές δεικτοδοτημένης αντιγραφής: $x = y[i]$ και $x[i] = y$
   Η εντολή $x = y[i]$ αναθέτει στο x την τιμή που βρίσκεται i θέσεις μνήμης μετά το y, ενώ η εντολή $x[i] = y$ αποθηκεύει την τιμή του ονόματος y στη θέση που βρίσκεται i θέσεις μνήμης μετά το x.
9. Εντολές ανάθεσης διευθύνσεων και εντολών: $x = \&y$, x = *y και *x = y
   όπου τα x και y είναι διευθύνσεις, πιθανός προσωρινές. Η εντολή $x = \&y$ αναθέτει σαν r-value τιμή στο x την l-value τιμή του y. Η εντολή x = *y αναθέτει σαν r-value τιμή στο x το περιεχόμενο της θέσης y. Ενώ τέλος, η εντολή *x = y αναθέτει στην r-value τιμή του αντικειμένου στο οποίο δείχνει το x την r-value τιμή του y.

Ένα παράδειγμα όλων αυτών αποτελεί ο ακόλουθος κώδικας, ο οποίος αλλάζει έναν πίνακα 10×10 σε ταυτοτικό:

```
for i from 1 to 10 do
            for j from 1 to 10 do
                  a[i,j] = 0.0;
            for i from 1 to 10 do
                  a[i,i] = 0.0;
```

όταν μετασχηματίζεται σε κώδικα τριών διευθύνσεων έχει την μορφή:

```
1) i = 1
2) j = 1
3) t1 = 10 * i
4) t2 = t1 + j
5) t3 = 8 * t2
6) t4 = t3 - 88
7) a[t4] = 0.0
8) j = j + 1
9) if j <= 10 goto (3)
10)   i = i + 1
11)   if i <= 10 goto (2)
12)   i = 1
13)   t5 = i - 1
14)   t6 = 88 * t5
15)   a[t6] = 1.0
16)   i = i  + 1
17)   if i <= 10 goto (13)
```

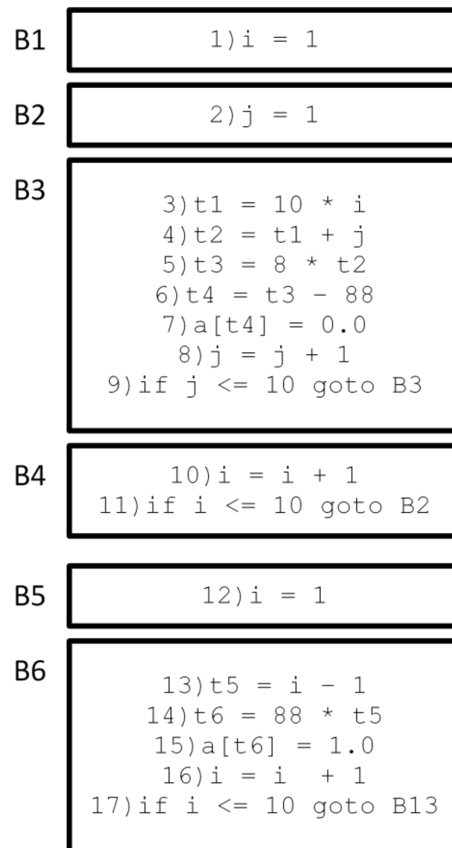
2.2 Βασικά Μπλοκ (Basic Blocks)

Αυτή η ενότητα μελετάει μία γραφική αναπαράσταση του ενδιάμεσου κώδικα. Αφού ο πηγαίος κώδικας αρχικά μεταφραστεί σε ενδιάμεσο κώδικα, εν συνεχεία χωρίζεται στα βασικά μπλοκ. Κάθε βασικό μπλοκ περιέχει ένα κομμάτι κώδικα το οποίο έχει ορισμένες βασικές ιδιότητες, οι οποίες του επιτρέπουν να είναι ιδιαίτερα δεκτικό σε ανάλυση. Ο κώδικας σε ένα βασικό μπλοκ έχει μόνο ένα σημείο εισόδου, το οποίο σημαίνει ότι καμία εντολή μέσα στο βασικό μπλοκ, εκτός πιθανόν από την αρχική εντολή, δεν είναι ο προορισμός από ένα άλμα από οπουδήποτε αλλού στον κώδικα. Εκτός αυτού το βασικό μπλοκ έχει και μοναδικό σημείο εξόδου, το οποίο με την σειρά του σημαίνει ότι μόνο η τελευταία εντολή στο βασικό μπλοκ μπορεί να προκαλέσει την εκτέλεση ενός άλλου βασικού μπλοκ. Κάτω από αυτούς τους περιορισμούς, μπορούμε εύκολα να καταλάβουμε ότι όποτε εκτελείται η πρώτη εντολή σε ένα βασικό μπλοκ, αναγκαστικά οι υπόλοιπες εντολές εκτελούνται ακριβώς μία φορά με την ακριβή σειρά που είναι διατεταγμένες μέσα στο βασικό μπλοκ.

Ο αλγόριθμος για την κατασκευή των βασικών μπλοκ είναι σχετικά απλός. Το μόνο που χρειάζεται είναι να σαρώσει ολόκληρο το πρόγραμμα και να σημειώσει τα όρια

των βασικών μπλοκ. Αυτά τα όρια είναι εντολές με τις οποίες μπορεί να ξεκινήσει ή να τελειώσει ένα βασικό μπλοκ, δηλαδή εντολές οι οποίες είτε μεταφέρουν τον έλεγχο σε άλλο σημείο, είτε εντολές οι οποίες δέχονται τον έλεγχο από άλλο σημείο. Στη συνέχεια, απλά κόβει την σειρά των εντολών τριών διευθύνσεων σε αυτά τα σημεία. Αυτές οι ομάδες εντολών αποτελούν τα βασικά μπλοκ, τα οποία πρέπει να σημειωθεί ότι δεν είναι τα μέγιστα βασικά μπλοκ που μπορούν να δημιουργηθούν, αλλά ωστόσο είναι συχνά αποδοτικά. Τα μέγιστα βασικά μπλοκ είναι εκείνα τα οποία δεν μπορούν να επεκταθούν με την συμπερίληψη άλλων παρακείμενων μπλοκ χωρίς να παραβιάζουν τον ορισμό των βασικών μπλοκ.

Ο απλός αυτός αλγόριθμος θα δέχεται ως είσοδο μία αλληλουχία με εντολές τριών διευθύνσεων και θα παράγει ως έξοδο μία λίστα με βασικά μπλοκ στα οποία θα καταχωρηθούν οι εντολές τριών διευθύνσεων. Κάθε εντολή τριών διευθύνσεων καταχωρείται αυστηρά σε ένα βασικό μπλοκ. Για να γίνει σωστά ο διαχωρισμός θα πρέπει αρχικά να βρούμε το όριο κάθε βασικού μπλοκ. Το όριο ενός βασικού μπλοκ τελειώνει όταν συναντούμε μία αρχική εντολή. Αρχική εντολή μπορεί να είναι μία εντολή όταν είναι η πρώτη εντολή στον ενδιάμεσο κώδικα, όταν είναι ο στόχος ενός, υπό όρους ή μη, άλματος ή όταν είναι αυτή που ακολουθεί ένα άλμα, υπό όρους ή μη, στον ενδιάμεσο κώδικα. Έτσι το εκάστοτε βασικό μπλοκ αποτελείται από την αρχική εντολή και όλες τις εντολές που ακολουθούν μέχρι να φτάσουμε στην επόμενη αρχική εντολή. Για παράδειγμα ο κώδικας τριών διευθύνσεων του παραδείγματος στην ενότητα 2.1 όταν χωριστεί σε βασικά μπλοκ θα έχει την μορφή που φαίνεται στο σχήμα 2.3.



Σχήμα 2.3: Βασικά Μπλοκ.

Όπως θα μπορούσε να παρατηρήσει κανείς οι ετικέτες μπροστά από κάθε εντολή μπορούν να αντικατασταθούν με ετικέτες αρχικών μπλοκ. Αρχικά, η εντολή 1 είναι αρχική λόγω του ότι είναι η πρώτη εντολή στον ενδιάμεσο κώδικα. Για να βρούμε τις άλλες αρχικές, χρειάζεται πρώτα να βρούμε τα άλματα. Στο παράδειγμα μας, υπάρχουν τρία άλματα, όλα υπό όρους, και είναι οι εντολές 9, 11 και 17. Οι στόχοι αυτών των αλμάτων είναι αρχικές εντολές και είναι οι εντολές 3, 2 και 13 αντίστοιχα. Στη συνέχεια, κάθε εντολή η οποία ακολουθεί ένα άλμα είναι αρχική. Αυτές είναι οι εντολές 10 και 12. Σημειώστε ότι καμία εντολή δεν ακολουθεί μετά την εντολή 17 στον κώδικα, αλλά αν την ακολουθούσε τότε η εντολή 18 θα ήταν αρχική.

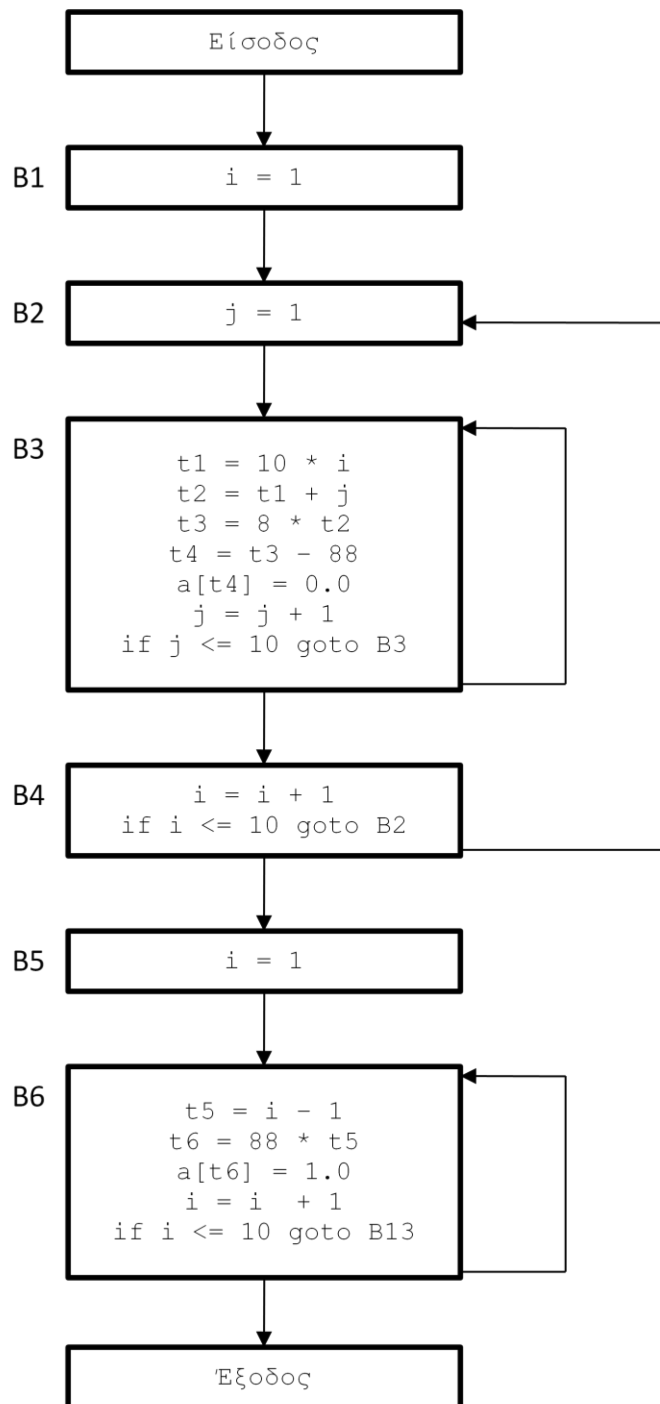
2.3 Γραφήματα Ροής Ελέγχου (Control Flow Graph)

Από την στιγμή που ο ενδιάμεσος κώδικας διαμελιστεί στα βασικά μπλοκ, αναπαριστούμε τον έλεγχο της ροής με ένα γράφημα ροής. Οι κόμβοι του γραφήματος ροής είναι τα βασικά μπλοκ. Υπάρχει ακμή από το μπλοκ X στο μπλοκ Y αν και μόνο αν είναι δυνατόν να μεταβούμε στην πρώτη εντολή του Y από την

τελευταία εντολή του X. Υπάρχουν δύο λόγοι για τους οποίους μπορούμε να εισάγουμε μία τέτοια ακμή. Ο πρώτος είναι να υπάρχει ένα άλμα, υπό όρους ή μη, από το τέλος του X στην αρχή του Y, ενώ ο δεύτερος είναι το Y να ακολουθεί απευθείας το X στον ενδιάμεσο κώδικα, και το X να τελειώνει με ένα υπό όρους άλμα. Τότε λέμε ότι το X είναι προκάτοχος του Y, και ότι το Y είναι διάδοχος του X.

Συχνά προσθέτουμε δύο κόμβους, τους οποίους ονομάζουμε Είσοδος (Entry) και Έξοδος (Exit), οι οποίοι δεν ανταποκρίνονται σε εκτελέσιμες εντολές ενδιάμεσου κώδικα. Υπάρχει μια ακμή από τον κόμβο Είσοδος στον πρώτο εκτελέσιμο κόμβο του γραφήματος ροής, δηλαδή, προς το βασικό μπλοκ το οποίο εμπεριέχει την πρώτη εντολή του ενδιάμεσου κώδικα. Επίσης υπάρχει μία ακμή προς τον κόμβο Έξοδος από κάθε βασικό μπλοκ το οποίο περιέχει μία εντολή η οποία θα μπορούσε να είναι η τελευταία εντολή του προγράμματος. Τα σύνολα των βασικών μπλοκ του παραπάνω παραδείγματος δημιουργούν το γράφημα ροής ελέγχου που φαίνεται στο σχήμα 2.4.

Ο κόμβος Είσοδος δίνει τον έλεγχο στο μπλοκ $B1$, από την στιγμή που το $B1$ περιέχει την πρώτη εντολή του προγράμματος. Ο μόνος απόγονος του $B1$ είναι ο $B2$, επειδή ο $B1$ δεν τελειώνει με άλμα και επειδή η αρχική εντολή του $B2$ ακολουθεί απευθείας μετά την τελευταία εντολή του $B1$. Το μπλοκ $B3$ έχει δύο ακμές. Η μία προς τον εαυτό του, επειδή η αρχική εντολή της $B3$ είναι ο στόχος του υπό όρους άλματος της τελευταίας εντολής του $B3$. Η άλλη ακμή καταλήγει στο $B4$, επειδή ο έλεγχος μπορεί να μεταβεί λόγο του υπό όρους άλματος από το τέλος του $B3$ στην αρχική του $B4$. Μόνο το μπλοκ $B6$ δείχνει στον κόμβο Έξοδος του γραφήματος ροής, από την στιγμή που ο μόνος τρόπος για να τερματίσει το πρόγραμμα είναι το υπό όρους άλμα σε αυτό το μπλοκ. Αυτό το τελευταίο υπό όρους άλμα ή θα ξαναδώσει τον έλεγχο στο μπλοκ $B6$ ή θα τερματίσει το πρόγραμμα.



Σχήμα 2.4: Γράφημα Ροής Ελέγχου.

Τα γραφήματα ροής, είναι σχεδόν συνηθισμένα γραφήματα, και μπορούν να αναπαρασταθούν από οποιαδήποτε δομή δεδομένων κατάλληλη για αυτά. Το περιεχόμενο των κόμβων (βασικά μπλοκ) χρειάζονται την δική τους αναπαράσταση. Μπορούμε να αναπαραστήσουμε το περιεχόμενο ενός κόμβου με ένα δείκτη στην αρχική εντολή του πίνακα με τις εντολές τριών διευθύνσεων, μαζί με ένα μετρητή με τον αριθμό των εντολών ή με ένα δεύτερο δείκτη στην τελευταία εντολή. Ωστόσο,

από την στιγμή που ο αριθμός των εντολών αλλάζει στο βασικό μπλοκ συχνά, είναι προτιμότερο και πιο αποδοτικό να δημιουργήσουμε μια συνδεδεμένη λίστα με εντολές για κάθε βασικό μπλοκ.

Κλείνοντας, πρέπει να τονίσουμε ότι δομές στις γλώσσες προγραμματισμού όπως η δήλωση while, η δήλωση do-while και η δήλωση for προκαλούν τη δημιουργία βρόχων. Από την στιγμή που κάθε πρόγραμμα στην ουσία ξοδεύει τον περισσότερο χρόνο εκτέλεσης στους βρόχους, είναι πολύ σημαντικό οι μεταγλωττιστές να παράγουν καλό κώδικα για αυτούς. Λέμε ότι ένα σύνολο με κόμβους Λ σε ένα γράφημα ροής είναι ένας βρόχος αν το Λ περιέχει έναν κόμβο ε ο οποίος λέγεται και είσοδος του βρόχου και ισχύουν οι ακόλουθες ιδιότητες:

1. Ο ε δεν είναι ο κόμβος Είσοδος, δηλαδή η αρχή του γραφήματος ροής.
2. Κανένας κόμβος x, εκτός του ε, στο Λ δεν έχει πρόγονο εκτός του Λ. Δηλαδή, κάθε μονοπάτι του γραφήματος ροής από τον κόμβο Είσοδος προς το x, περνάει υποχρεωτικά από το ε.
3. Κάθε κόμβος στο Λ έχει ένα μη κενό μονοπάτι, εξολοκλήρου μέσα στο Λ, προς το ε.

Παραδείγματος χάριν το παραπάνω γράφημα ροής ελέγχου έχει τους παρακάτω βρόχους:

1. Το $B3$ από μόνο του.
2. Το $B6$ από μόνο του.
3. Και το $B2 \rightarrow B3 \rightarrow B4 \rightarrow B2$.

2.4 Βελτιστοποίηση των Βασικών Μπλοκ

Η καθολική βελτιστοποίηση του κώδικα, η οποία ελέγχει το πώς ρέει η πληροφορία, αποτελεί ένα πολύπλοκο πρόβλημα, που χρησιμοποιεί πολλούς αλγορίθμους, για το οποίο θα μιλήσουμε στη συνέχεια. Ωστόσο μπορούμε να πετύχουμε τοπική βελτιστοποίηση στο κώδικα, εντός των βασικών μπλοκ, στο χρόνο εκτέλεσης του.

2.4.1 Κατευθυνόμενη Άκυκλη Γραφική Αναπαράσταση των Βασικών Μπλοκ

Πολλές σημαντικές τεχνικές για την τοπική βελτιστοποίηση του κώδικα ξεκινούν με την μετατροπή του βασικού μπλοκ σε κατευθυνόμενο άκυκλο γράφημα (ΚΑΓ). Ένα βασικό μπλοκ μετατρέπεται σε ΚΑΓ με τον ακόλουθο τρόπο:

I. Κάθε μία από τις αρχικές τιμές των μεταβλητών που εμφανίζονται στα βασικά μπλοκ αντιστοιχίζεται με ένα κόμβο στο ΚΑΓ.
II. Υπάρχει ένας κόμβος N ο οποίος συνδέεται με κάθε δήλωση s μέσα στο μπλοκ. Τα παιδιά του N, στο γράφημα, είναι εκείνοι οι κόμβοι οι οποίοι αντιστοιχούν στις δηλώσεις οι οποίες είναι τα ορίσματα του τελεστή που χρησιμοποιεί η δήλωση s.
III. Ο κόμβος N έχει σαν ετικέτα τον τελεστή που εφαρμόζεται στη δήλωση s, και επίσης στο N συνδέεται η λίστα με μεταβλητές, των οποίων η τελευταία αποτίμηση, συμπίπτει με το αποτέλεσμα της δήλωσης s.
IV. Συγκεκριμένοι κόμβοι καθορίζονται σαν κόμβοι εξόδου. Αυτοί είναι οι κόμβοι των οποίων οι μεταβλητές είναι ζωντανές στην έξοδο από το μπλοκ, δηλαδή η τιμή τους μπορεί να χρησιμοποιηθεί αργότερα, σε κάποιο άλλο βασικό μπλοκ στο γράφημα ροής. Ο υπολογισμός αυτών των «ζωντανών» μεταβλητών είναι ζήτημα της καθολικής ανάλυσης ροής.

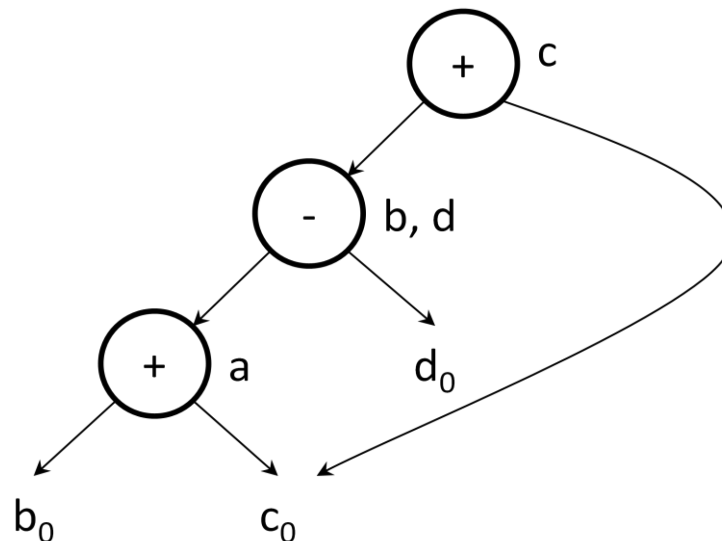
Η ΚΑΓ αναπαράσταση των βασικών μπλοκ μας επιτρέπει να εκτελέσουμε αρκετούς μετασχηματισμούς βελτίωσης κώδικα μέσα στο βασικό μπλοκ. Αναφορικά μπορούμε να εξαλείψουμε τοπικά όμοιες εντολές, μπορούμε να αφαιρέσουμε τον «νεκρό» κώδικα, δηλαδή τις εντολές που υπολογίζουν τιμές οι οποίες δεν χρησιμοποιούνται, μπορούμε να αναδιατάξουμε τις εντολές οι οποίες δεν βασίζονται η μία στην άλλη, επιτυγχάνοντας τη μείωση του χρόνου που μία προσωρινή μεταβλητή παραμένει στον καταχωρητή, και τέλος μπορούμε να εφαρμόσουμε αλγεβρικούς νόμους για να αναδιατάξουμε τους τελεστές των εντολών τριών διευθύνσεων ώστε να απλοποιήσουμε μερικούς υπολογισμούς.

2.4.2 Εύρεση Ομοίων Τοπικών Εντολών

Οι κοινές εντολές ανιχνεύονται τη στιγμή της δημιουργίας ενός κόμβου M για τη δήλωση s, ελέγχοντας αν ένας άλλος κόμβος N με το ίδιο τελεστή και τα ίδια παιδιά, στην ίδια σειρά, υπάρχει ήδη. Αν υπάρχει, τότε το N υπολογίζει την ίδια τιμή με το M και μπορεί να χρησιμοποιηθεί αντί αυτού. Παράδειγμα αποτελεί ο ακόλουθος κώδικας ο οποίος μπορεί να αποτελεί μέρος ενός βασικού μπλοκ:

$$a = b + c$$
$$b = a - d$$
$$c = b + c$$
$$d = a - d$$

η ΚΑΓ αναπαράσταση αυτού του κώδικα φαίνεται στο σχήμα 2.6.



Σχήμα 2.6: ΚΑΓ αναπαράσταση του κώδικα στο παράδειγμα.

όταν θα δημιουργήσουμε τον κόμβο για την τρίτη εντολή, $c = b + c$, θα αναγνωρίσουμε ότι η χρήση του b στο $b + c$ αναφέρεται στον κόμβο με την ετικέτα " $-$ ". Ωστόσο, οι κόμβοι που αντιστοιχούν στην τέταρτη εντολή, $d = a - d$, αποτελούνται από τον τελεστή " $-$ " και τους a και d_0 σαν παιδιά. Από την στιγμή που ο τελεστής και τα παιδιά είναι τα ίδια με εκείνα των κόμβων που δημιουργούνται από την εντολή δύο, δεν θα δημιουργηθούν νέοι κόμβοι, αλλά αντί αυτού το d θα προστεθεί στη λίστα των ορισμάτων για τον κόμβο με ετικέτα " $-$ ".

Το παράδειγμα μπορεί να δείχνει ότι οι τέσσερεις παραπάνω εντολές μπορούν να αντικατασταθούν με τις ακόλουθες τρεις:

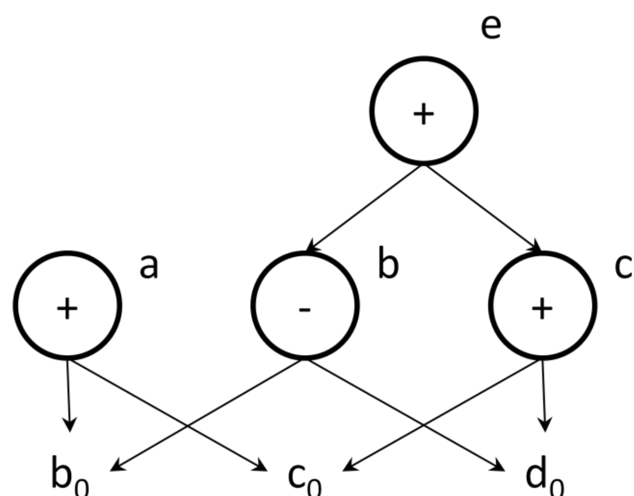
$$a = b + c$$
$$d = a - d$$
$$c = d + c$$

Ωστόσο αν τα b και d είναι «ζωντανά» στην έξοδο του μπλοκ, τότε η τέταρτη εντολή, που θα αντιγράφει τη τιμή της μίας μεταβλητής στην άλλη ($b = d$), είναι αναπόφευκτη.

Όταν ψάχνουμε για τις κοινές εντολές, στην πραγματικότητα ψάχνουμε για εκφράσεις οι οποίες είναι σίγουρο ότι υπολογίζουν την ίδια τιμή, χωρίς να έχει σημασία το πως υπολογίζεται αυτή. Έτσι, η μέθοδος ΚΑΓ θα χάσει το γεγονός ότι τα αποτελέσματα που υπολογίζονται από την πρώτη και την τέταρτη εντολή στην ακολουθία

$$a = b + c$$
$$b = b - d$$
$$c = c + d$$
$$e = b + c$$

είναι ίσα μεταξύ τους ($b_0 + c_0$). Δηλαδή, ακόμα και αν το b και το c αλλάζουν στο ενδιάμεσο της πρώτης και της τέταρτης δήλωσης, το άθροισμα τους παραμένει το ίδιο, επειδή $b + c = (b - d) + (c + d)$. Το ΚΑΓ για αυτή την ακολουθία φαίνεται στο σχήμα 2.7, αλλά δεν επιδεικνύουν καμία κοινή εντολή. Ωστόσο, αλγεβρικές ταυτότητες που εφαρμόζονται στο ΚΑΓ, οι οποίες θα συζητηθούν παρακάτω, μπορεί να φανερώσουν αυτήν την ισότητα.



Σχήμα 2.7: ΚΑΓ αναπαράσταση των εντολών του παραδείγματος.

2.4.3 Διαγραφή «Νεκρού» Κώδικα

Η διαδικασία με την οποία μπορούμε να διαγράψουμε τον «νεκρό» κώδικα από το ΚΑΓ εκτελείται ως ακολούθως. Διαγράφουμε από το ΚΑΓ κάθε πηγή (κόμβο χωρίς προγόνους) στην οποία δεν έχει συνδεθεί καμία ζωντανή μεταβλητή (μεταβλητή η οποία υπολογίζεται αλλά δεν χρησιμοποιείται). Επαναλαμβάνοντας αυτή τη διαδικασία, αφαιρούμε όλους τους κόμβους από το ΚΑΓ οι οποίοι αντιστοιχούν σε «νεκρό» κώδικα. Παραδείγματος χάριν, στο ΚΑΓ του σχήματος 2.7, αν το a και το b είναι ζωντανές μεταβλητές ενώ το c και το e είναι νεκρές μεταβλητές, τότε μπορούμε να αφαιρέσουμε απευθείας την πηγή με ετικέτα e. Στη συνέχεια, ο κόμβος με ετικέτα c γίνεται πηγή και αφαιρείται. Οι κόμβοι με ετικέτες a και b δεν αφαιρούνται, από την που τα a και b είναι ζωντανές μεταβλητές.

2.4.4 Χρήση Αλγεβρικών Ταυτοτήτων

Οι αλγεβρικές ταυτότητες αποτελούν ένα άλλο σημαντικό τμήμα της βελτιστοποίησης των βασικών μπλοκ. Για παράδειγμα, μπορούμε να προσθέσουμε αριθμητικές ταυτότητες, όπως

$$a + 0 = 0 + a = a$$
$$a \times 1 = 1 \times a = a$$
$$a - 0 = a$$
$$a/1 = a$$

για να μειώσουμε τους υπολογισμούς στα βασικά μπλοκ.

Μία άλλη ομάδα αλγεβρικών ταυτοτήτων βελτιστοποιούν τον κώδικα παρέχοντας τοπική μείωση σε ισχύ, δηλαδή αντικαθιστούν ένα δαπανηρό τελεστή με ένα λιγότερο:

Δαπανηρός		Λιγότερο Δαπανηρός
a^2	=	$a \times a$
$2 \times a$	=	$a + a$
$a/2$	=	$a \times 0.5$

Μία τρίτη ομάδα από παρόμοιες βελτιστοποιήσεις αποτελεί η αναδίπλωση (constant folding) σταθερών. Εδώ εκτιμούμε σταθερές εκφράσεις στο χρόνο μεταγλώττισης και

τις αντικαθιστούμε με την τιμή τους. Έτσι η έκφραση 2×3.14 θα αντικατασταθεί από το 6.28. Πολλές σταθερές εκφράσεις εμφανίζονται στην πράξη λόγω της συχνής χρήσης των συμβόλων σταθερών στο πρόγραμμα.

Η επεξεργασία της ΚΑΓ δομής μπορεί να μας βοηθήσει εφαρμόζοντας και άλλες πιο γενικές αλγεβρικές αλλαγές όπως η αντιμεταθετικότητα (commutativity) και η συσχέτιση (associative). Για παράδειγμα, υποθέστε μία γλώσσα η οποία έχει χαρακτηρίσει το σύμβολο " $\times$ " (πολλαπλασιασμό) ως αντιμεταθετική πράξη, δηλαδή, $x \times y = y \times x$. Πριν κατασκευάσουμε ένα καινούριο κόμβο με ετικέτα " $\times$ " με αριστερό παιδί το M και δεξιό το N, πάντα θα ελέγχουμε αν αυτός ο κόμβος υπάρχει. Ωστόσο, επειδή ο " $\times$ " είναι σύμβολο αντιμεταθετικής πράξης, θα πρέπει να ελέγξουμε και για κόμβο ο οποίος έχει επισυναπτόμενο τελεστή τον " $\times$ ", αριστερό παιδί το N και δεξιό το M.

Οι συσχετικοί τελεστές όπως οι " $<$ " και " $=$ " μερικές φορές δημιουργούν κοινές υποεκφράσεις οι οποίες δεν είναι αναμενόμενες. Για παράδειγμα, η συνθήκη $x > y$ μπορεί να ελεγχθεί και με αφαίρεση των μεταβλητών και έλεγχο του αποτελέσματος. Έτσι, μόνο ένας ΚΑΓ κόμβος θα χρειαστεί να δημιουργηθεί για τις δηλώσεις $x - y$ και $x > y$.

Οι νόμοι της συσχέτισης μπορεί επίσης να είναι κατάλληλοι για την αποκάλυψη κοινών υποεκφράσεων. Για παράδειγμα, εάν ο πηγαίος κώδικας έχει τις δηλώσεις:

$a = b + c;$

$e = c + d + b;$

τότε μπορεί να δημιουργηθεί ο ακόλουθος ενδιάμεσος κώδικας:

$a = b + c$

$t = c + d$

$e = t + b$

Εάν το t δεν χρειάζεται έξω από το μπλοκ, μπορούμε να αλλάξουμε την ακολουθία σε:

$a = b + c$

$e = a + d$

χρησιμοποιώντας ταυτόχρονα και την αντιμεταθετικότητα και την συσχετικότητα της πράξης με τελεστή το σύμβολο " $+$ ".

2.4.5 Αναπαραστάσεις σε Αναφορές Πινάκων

Με την πρώτη ματιά, μπορεί να φαίνεται ότι οι εντολές σύνταξης πινάκων μπορούν να επεξεργαστούν όπως οι τελεστές άλλων εντολών. Σκεφτείτε για μια στιγμή την ακολουθία της δήλωσης τριών διευθύνσεων

$x = a[i]$

$a[j] = y$

$z = a[i]$

Αν θεωρήσουμε τον $a[i]$ σαν ένα τελεστή ο οποίος εμπλέκει τους a και i, όπως ο $a + i$, τότε μπορεί να θεωρηθούν οι δύο χρήσεις του $a[i]$ σαν κοινές υποεκφράσεις. Σε αυτή τη περίπτωση, μπορεί να παρασυρθούμε και να το βελτιστοποιήσουμε αντικαθιστώντας την τρίτη έκφραση με την πιο απλή $z = x$. Ωστόσο, από την στιγμή που το i μπορεί να είναι ίσο με το j, η μεσαία δήλωση μπορεί να αλλάξει την τιμή της $a[i]$, καθιστώντας την αλλαγή μη ορθή.

Ο αρμόζων τρόπος για να αναπαραστήσουμε τις προσπελάσεις πίνακα σε ένα ΚΑΓ είναι ο ακόλουθος:

1. Η ανάθεση από ένα πίνακα, όπως το $x = a[i]$, αναπαρίσταται δημιουργώντας ένα κόμβο με τελεστή $=$ [] και δύο παιδιά που αναπαριστούν την αρχική τιμή του πίνακα, ας πούμε a_0, και τον δείκτη i. Η μεταβλητή x γίνεται ετικέτα στον νέο κόμβο.
2. Μία ανάθεση σε πίνακα, όπως το $a[j] = y$, αναπαρίσταται από ένα νέο κόμβο με τελεστή [] $=$ και τρία παιδιά που αναπαριστούν τα a_0, j και y. Δεν υπάρχει μεταβλητή ως ετικέτα αυτού του κόμβου. Αυτό που είναι διαφορετικό είναι ότι η δημιουργία αυτού του κόμβου καταστρέφει όλους τους κατασκευασμένους πρόσφατους κόμβους των οποίων οι τιμές εξαρτώνται από το a_0. Ένας κόμβος ο οποίος καταστρέφεται δεν μπορεί να λάβει άλλες ετικέτες, δηλαδή, δεν μπορεί να υπάρξει ως κοινή εντολή.

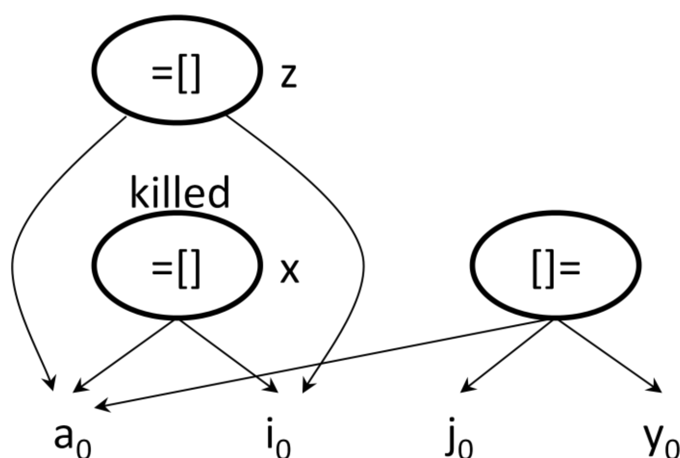
Για παράδειγμα, το ΚΑΓ για το βασικό μπλοκ

$x = a[i]$

$a[j] = y$

$z = a[i]$

εμφανίζεται στο σχήμα 2.8. Ο κόμβος με ετικέτα x δημιουργείται πρώτος, αλλά όταν δημιουργείται ο κόμβος με ετικέτα [] =, ο κόμβος του x καταστρέφεται (killed). Έτσι, όταν δημιουργείται ο κόμβος για το z, ο κόμβος του x δεν μπορεί να αναγνωριστεί σαν κοινή εντολή, και ένας νέος κόμβος με τους ίδιους τελεστές a_0 και i_0 θα πρέπει να δημιουργηθεί.



Σχήμα 2.8: ΚΑΓ αναπαράσταση των εντολών του παραδείγματος.

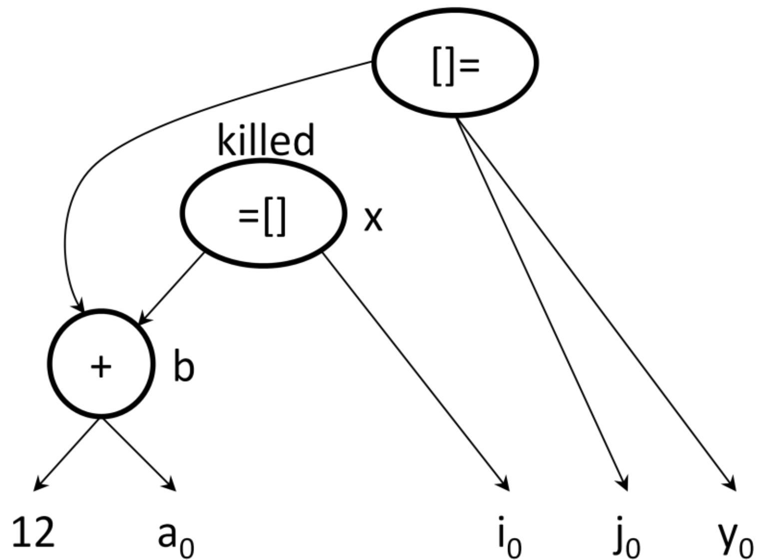
Μερικές φορές, ένας κόμβος θα πρέπει να καταστραφεί ακόμα και εάν κανένα από τα παιδιά του δεν είναι πίνακας όπως παραπάνω ο a_0. Ομοίως, ένας κόμβος μπορεί να καταστραφεί εάν έχει έναν απόγονο ο οποίος είναι πίνακας, ακόμα και αν κανένα από τα παιδιά του δεν είναι πίνακας. Για παράδειγμα, θεωρείστε των παρακάτω κώδικα τριών διευθύνσεων και την ΚΑΓ αναπαράστασή του στο σχήμα 2.9

$b = 12 + a$

$x = b[i]$

$b[j] = y$

Αυτό που συμβαίνει εδώ είναι ότι, για λόγους αποδοτικότητας, το b έχει οριστεί να είναι μία θέση στον πίνακα a. Για παράδειγμα, εάν το στοιχείο του a έχει μήκος τεσσάρων bytes, τότε το b αναπαριστά το τέταρτο στοιχείο του πίνακα a. Εάν τα i και j αναπαριστούν την ίδια τιμή, τότε τα $b[i]$ και $b[j]$ αναπαριστούν την ίδια τοποθεσία. Γι' αυτό είναι σημαντικό από την στιγμή που έχουμε την τρίτη εντολή, $b[j] = y$, να καταστρέψουμε τον κόμβο που είναι συνδεδεμένος με τη μεταβλητή x. Ωστόσο, όπως βλέπουμε στο σχήμα παρακάτω, ταυτόχρονα ο κόμβος που καταστρέφει και ο κόμβος που καταστρέφεται έχουν το a_0 σαν εγγόνι, και όχι σαν παιδί.



Σχήμα 2.9: ΚΑΓ αναπαράσταση των εντολών του παραδείγματος.

2.4.6 Αναθέσεις σε Δείκτη και Κλήσεις Διαδικασιών

Όταν κάνουμε μία έμμεση εκχώρηση μέσου ενός δείκτη, όπως στη δήλωση

x = *p

*q = y

δεν ξέρουμε που δείχνουν τα p και q. Σαν αποτέλεσμα, το x = *p είναι μία χρήση οποιασδήποτε, έως τώρα, μεταβλητής, και το *q = y είναι μία πιθανή καταχώρηση σε οποιαδήποτε μεταβλητή. Σαν αποτέλεσμα, ο τελεστής =* πρέπει να πάρει όλους τους κόμβους που ως τώρα συνδέονται με αναγνωριστικά σαν ορίσματα, το οποίο είναι συναφή με διαγραφή νεκρού κώδικα. Πιο σημαντικά, ο *= τελεστής καταστρέφει όλους τους άλλους κόμβους που δημιουργήθηκαν ως τώρα στο ΚΑΓ.

Υπάρχουν καθολικοί αναλυτές δεικτών οι οποίοι μπορούν να εκτελέσουν την πιθανή διαγραφή στο σύνολο των μεταβλητών στις οποίες μπορεί να αναφέρεται ένας δείκτης στο συγκεκριμένο σημείο του κώδικα. Ακόμα και τοπικοί αναλυτές μπορούν να περιορίσουν το εύρος ενός δείκτη. Για παράδειγμα, για την ακολουθία

p = &x

*p = y

γνωρίζουμε ότι η x, και όχι άλλη μεταβλητή, καταχωρεί την τιμή στο y, οπότε δεν χρειάζεται να καταστρέψουμε κανένα άλλο κόμβο παρά μόνο τον κόμβο στον οποίο επισυνάπτεται ο x.

Οι κλήσεις διαδικασιών συμπεριφέρονται αρκετά σαν αναθέσεις μέσου δεικτών. Εν απουσία καθολικών πληροφοριών ροής πληροφορίας, πρέπει να θεωρήσουμε ότι μία διαδικασία χρησιμοποιεί και αλλάζει κάθε πληροφορία στην οποία έχει πρόσβαση. Έτσι, εάν η διαδικασία P είναι στο εύρος της μεταβλητής x, μία κλήση της P ταυτόχρονα χρησιμοποιεί και καταστρέφει τον κόμβο με συνδεδεμένη μεταβλητή τον x.

2.4.7 Συναρμολόγηση των Βασικών Μπλοκ από το ΚΑΓ

Αφού εκτελέσουμε κάθε δυνατή βελτιστοποίηση κατά την κατασκευή και τον μετασχηματισμό του ΚΑΓ, μπορούμε να ξανασυνθέσουμε τον κώδικα τριών διευθύνσεων για το βασικό μπλοκ από το οποίο χτίσαμε το ΚΑΓ. Για κάθε κόμβο ο οποίος έχει μία ή περισσότερες επισυναπτόμενες μεταβλητές, συνθέτουμε μία δήλωση τριών διευθύνσεων η οποία υπολογίζει την τιμή μίας από αυτές τις μεταβλητές. Προτιμούμε να υπολογίζουμε το αποτέλεσμα σε μία μεταβλητή η οποία είναι ζωντανή στην έξοδο του μπλοκ. Ωστόσο, αν δεν έχουμε πληροφορία για τις καθολικά ζωντανές μεταβλητές του προγράμματος, πρέπει να θεωρήσουμε ότι κάθε μεταβλητή του προγράμματος (εκτός από τις προσωρινές οι οποίες δημιουργούνται από τον μεταγλωττιστή) είναι ζωντανή στην έξοδο του μπλοκ.

Αν ο κόμβος έχει περισσότερες από μία ζωντανές μεταβλητές συνδεδεμένες, τότε πρέπει να εισάγουμε αντίγραφα δηλώσεων για να δώσουμε τη σωστή τιμή καθεμίας από αυτές τις μεταβλητές. Μερικές φορές, η καθολική βελτιστοποίηση μπορεί να εξαλείψει αυτά τα αντίγραφα, αν καταφέρουμε να χρησιμοποιήσουμε τη μία από τις δύο αυτές μεταβλητές στη θέση της άλλης.

Ξαναθυμηθείτε το ΚΑΓ στο παραπάνω παράδειγμα (παράγραφος 2.5.2). Στη συζήτηση ακολουθώντας το παράδειγμα, αποφασίσαμε ότι αν το b δεν είναι ζωντανή μεταβλητή στην έξοδο από το μπλοκ, τότε οι τρεις δηλώσεις

$a = b + c$

$d = a - d$

$c = d + c$

επαρκούν για να ξανακατασκευάσουμε το βασικό μπλοκ. Η τρίτη εντολή, $c = d + c$, πρέπει να χρησιμοποιεί τον d σαν όρισμα αντί του b, γιατί το βελτιστοποιημένο μπλοκ δεν υπολογίζει ποτέ το b.

Αν και το b και το d είναι ζωντανά στην έξοδο, ή αν δεν είμαστε σίγουροι πότε είναι και πότε δεν είναι ζωντανά, τότε χρειάζεται να υπολογίσουμε και το b και το d. Αυτό μπορούμε να το καταφέρουμε με την ακολουθία

$a = b + c$

$d = a - d$

$b = d$

$c = d + c$

Αυτό το βασικό μπλοκ είναι ακόμα πιο αποδοτικό από το αρχικό. Παρόλο που ο αριθμός των εντολών είναι ο ίδιος, αφού πλέον έχουμε αντικαταστήσει μία εντολή πρόσθεσης με μία εντολή αντιγραφής, η οποία τείνει να είναι λιγότερο δαπανηρή στις περισσότερες μηχανές. Περεταίρω, θα μπορούσαμε κάνοντας καθολική ανάλυση, να εξαλείψουμε τον υπολογισμό του b έξω από το μπλοκ και να το αντικαταστήσουμε με το d. Σε αυτή τη περίπτωση, μπορούμε αργότερα να επιστρέψουμε στο βασικό μπλοκ και να σβήσουμε το $b = d$. Διαισθητικά, μπορούμε να σβήσουμε αυτό το αντίγραφο μόνο εάν όπου χρησιμοποιείται η τιμή του b, το d διατηρεί την ίδια τιμή. Αυτή η κατάσταση μπορεί να είναι αληθής αλλά μπορεί και να μην είναι, και εξαρτάται από το πως το πρόγραμμα επαναϋπολογίζει το d.

Όταν ξαναδημιουργούμε το βασικό μπλοκ από το ΚΑΓ, δεν πρέπει να ανησυχούμε μόνο για το ποιες μεταβλητές χρησιμοποιούνται για να κρατήσουν τις τιμές του κόμβου του ΚΑΓ, αλλά χρειάζεται επίσης να ανησυχούμε για τη σειρά με την οποία απαριθμούνται οι εντολές οι οποίες υπολογίζουν τις τιμές των διάφορων κόμβων. Οι κανόνες που πρέπει να θυμόμαστε είναι:

1. Η σειρά των εντολών πρέπει να σέβεται την σειρά των κόμβων στο ΚΑΓ, δηλαδή δεν μπορούμε να υπολογίσουμε την τιμή ενός κόμβου εάν

προηγουμένως δεν έχουμε υπολογίσει την τιμή για κάθε ένα από τα παιδιά του κόμβου.

2. Οι καταχωρήσεις σε ένα πίνακα πρέπει να ακολουθήσουν όλες τις προηγούμενες καταχωρήσεις στον ίδιο πίνακα.
3. Η αποτίμηση των στοιχείων του πίνακα πρέπει να ακολουθεί κάθε προηγούμενη καταχώρηση του ίδιου του πίνακα (σύμφωνα με το αρχικό μπλοκ). Η μόνη παραλλαγή που επιτρέπεται είναι ότι δύο αποτιμήσεις από τον ίδιο πίνακα μπορεί να γίνουν με οποιαδήποτε σειρά, από την στιγμή που καμία δεν διασταυρώνεται με μία καταχώριση στον πίνακα.
4. Κάθε χρήση μίας μεταβλητής πρέπει να ακολουθεί όλες τις προηγούμενες (σύμφωνα με το αρχικό μπλοκ) κλήσεις διαδικασίας ή έμμεσες καταχωρήσεις μέσου ενός δείκτη.
5. Όλες οι κλήσεις διαδικασίας ή οι έμμεσες καταχωρήσεις μέσου ενός δείκτη πρέπει να ακολουθούν όλες τις προηγούμενες (σύμφωνα με το αρχικό μπλοκ) αποτιμήσεις κάθε μεταβλητής.

Δηλαδή, όταν αναδιατάσσουμε τον κώδικα, καμία δήλωση δεν μπορεί να τέμνει μία κλήση διαδικασίας ή καταχώρηση μέσου ενός δείκτη, και οι χρήσεις του ίδιου πίνακα μπορούν να τέμνουν η μία την άλλη μόνο αν και οι δυο είναι προσπελάσεις πίνακα, αλλά όχι καταχωρήσεις σε στοιχεία του.

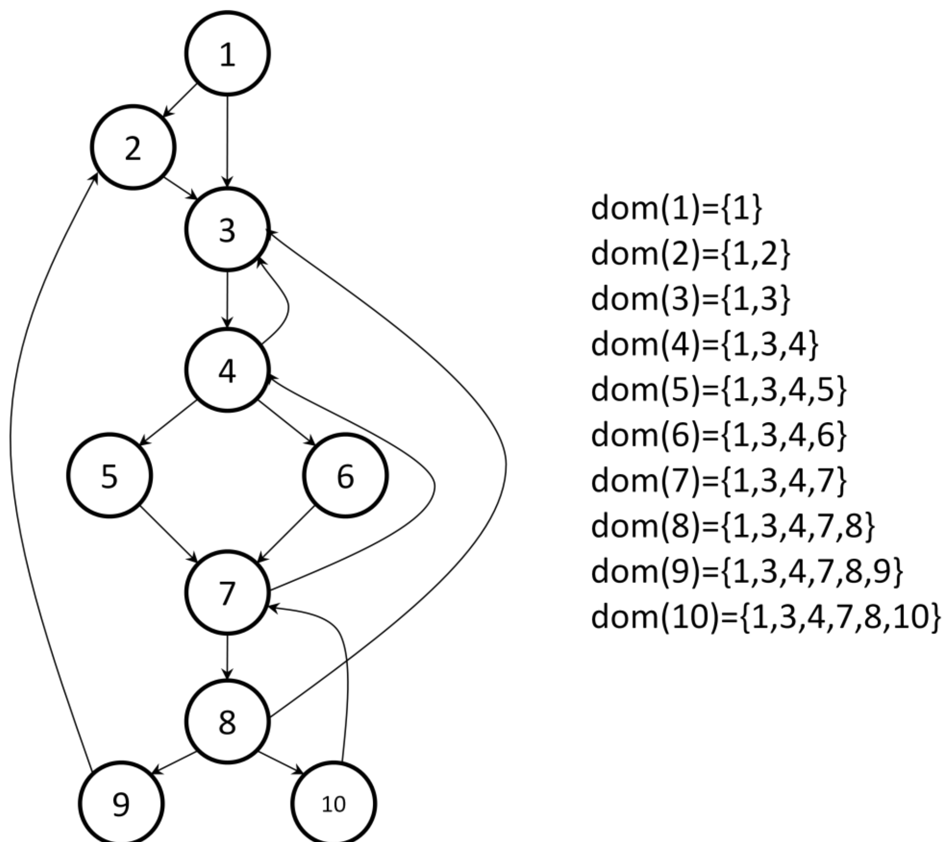
3 ΚΟΜΒΟΙ ΚΥΡΙΑΡΧΙΑΣ ΚΑΙ ΟΡΙΑ ΚΥΡΙΑΡΧΙΑΣ

Η εμφάνιση της SSA δομής (static single assignment form) ανακίνησε ξανά το ενδιαφέρον για τους κόμβους κυριαρχίας (dominators), σε ένα γράφημα, και για τεχνικές οι οποίες βασίζονται σε αυτούς. Ο λόγος είναι ότι δημιουργήθηκαν αρκετοί αλγόριθμοι για αρκετά προβλήματα στην βελτιστοποίηση και στη παραγωγή κώδικα, οι οποίοι στηρίχτηκαν σε αυτούς. Σε αυτό το κεφάλαιο θα εξετάσουμε το στατικό πρόβλημα των κόμβων κυριαρχίας, όταν δηλαδή το γράφημα παραμένει αναλλοίωτο σε σχέση με τον χρόνο. Το προσαυξητικό (incremental) πρόβλημα, όταν δηλαδή στο γράφημα προστίθενται επιπλέον ακμές, θα μελετηθεί στο κεφάλαιο 6. Σε αυτό το κεφάλαιο θα μελετήσουμε τους βασικούς αλγορίθμους, οι οποίοι υπολογίζουν τα σύνολα κυριαρχίας, και θα εξετάσουμε διεξοδικά τον επαναληπτικό αλγόριθμο, ο οποίος είναι ο βασικός αλγόριθμος που θα χρησιμοποιήσουμε για το στατικό κομμάτι του προβλήματος των κόμβων κυριαρχίας. Στη συνέχεια εξετάζουμε την έννοια των ορίων κυριαρχίας (dominance frontiers), καθώς και τρόπους για τον υπολογισμό αυτών των ορίων. Τα όρια κυριαρχίας αποτελούν τη βασική μας μέθοδο για τον υπολογισμό του προσαυξητικού τμήματος του προβλήματος των κόμβων κυριαρχίας. Επιπλέον, τα όρια κυριαρχίας αποτελούν μία αποτελεσματική μέθοδο για την τοποθέτηση των φ-συναρτήσεων που χρησιμοποιούμε στην SSA δομή, την οποία θα εξετάσουμε στο κεφάλαιο 4. Το πρόβλημα των κόμβων κυριαρχίας εμφανίζεται σε αρκετές εφαρμογές, όπως η βελτιστοποίηση και παραγωγή κώδικα, ο έλεγχο λειτουργίας κυκλωμάτων και η θεωρητική βιολογία. Οι μεταγλωττιστές κάνουν εκτενή χρήση των πληροφοριών που παίρνουμε από τους κόμβους κυριαρχίας κατά την διάρκεια της ανάλυσης και της βελτιστοποίησης ενός προγράμματος. Πιθανότατα, η καλύτερη εφαρμογή των κόμβων κυριαρχίας είναι η ανίχνευση βρόχων, η οποία με τη σειρά της ενεργοποιεί μία διαδικασία βελτιστοποίησης την οποία θα εξετάσουμε, και αυτή, παρακάτω (κεφάλαιο 5).

3.1 Εισαγωγή

Ένα γράφημα ροής $G = (V, E, r)$ είναι ένα κατευθυνόμενο γράφημα με $|V| = n$ κόμβους και $|E| = m$ κατευθυνόμενες ακμές, ή αλλιώς τόξα, τέτοια ώστε κάθε κόμβος να είναι προσπελάσιμος από τον αρχικό κόμβο r, ο οποίος ανήκει στο σύνολο V, και αποτελεί την πηγή ή αλλιώς ρίζα. Ένας κόμβος y κυριαρχεί επί ενός κόμβου x εάν κάθε μονοπάτι από το r στο x περνάει υποχρεωτικά από το y. Τότε λέμε ότι ο y ανήκει στο σύνολο με τους κόμβους οι οποίοι κυριαρχούν επί του x και

το συμβολίζουμε ως $y \in dom(x)$. Η ρίζα r καθώς και ο κόμβος x αποτελούν τετριμμένους κόμβους κυριαρχίας του x, δηλαδή εμφανίζονται πάντα στο σύνολο $dom(x)$. Ο στόχος μας είναι να βρούμε για όλους τους κόμβους x, οι οποίοι ανήκουν στο V, τα σύνολα $dom(x)$ όλων των κόμβων που κυριαρχούν επί του x. Στο σχήμα 3.1 παρουσιάζεται ένα παράδειγμα. Ο κόμβος 1 παίζει το ρόλο της ρίζας του γραφήματος (αριστερά). Τα σύνολα στο δεξιό μέρος της εικόνας είναι τα σύνολα με τους κόμβους που κυριαρχούν επί ενός κόμβου x ($dom(x)$). Το γράφημα αυτό θα μπορούσε άνετα να αποτελεί ένα γράφημα ροής ενός κώδικα, με τα βασικά μπλοκ του προγράμματος να αποτελούν τους κόμβους, αν προσθέταμε δύο επιπλέον κόμβους, ένα κόμβο Είσοδος (Entry) και ένα κόμβο Έξοδος (Exit), και ενώναμε τον κόμβο Είσοδος με τον κόμβο του προγράμματος που ξεκινάει το πρόγραμμα και το κόμβο Έξοδος με κάθε κόμβος ο οποίος μπορεί να τερματίσει το πρόγραμμα. Βέβαια τώρα ο κόμβος Είσοδος θα αποτελούσε την ρίζα του γραφήματος ροής και θα εμφανιζόταν σε όλα τα σύνολα των κόμβων κυριαρχίας.



Σχήμα 3.1: Η αναπαράσταση του γραφήματος και τα σύνολα με τους κόμβους κυριαρχίας.

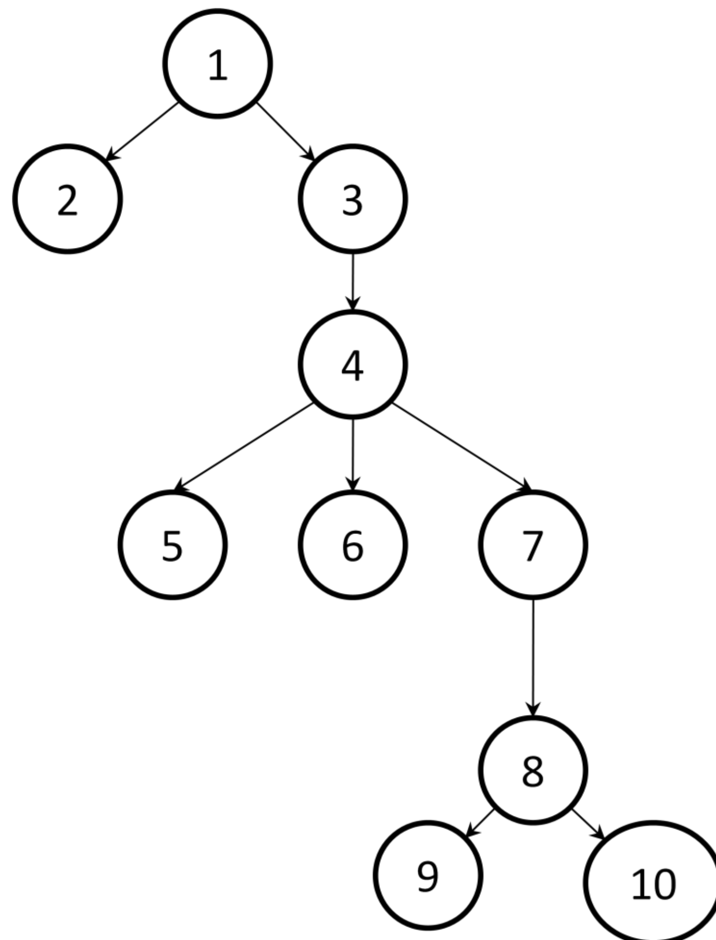
Στη συνέχεια εισάγουμε μερικούς παραπλήσιους ορισμούς οι οποίοι θα μας φανούν πολύ χρήσιμοι παρακάτω. Έτσι, λέμε ότι ένας κόμβος y είναι γνήσιος κυρίαρχος (strict dominator) του x, και συμβολίζεται ως $y\, stdom\, x$, εάν $y \in dom(x)$ και $x \neq y$. Ένας κόμβος y είναι άμεσος κυρίαρχος (immediate dominator) του x, και συμβολίζεται ως $y = idom(x)$, εάν είναι γνήσιος κυρίαρχος του x αλλά δεν είναι γνήσιος κυρίαρχος κανενός άλλου κόμβου w, ο οποίος w με τη σειρά του ανήκει στο σύνολο με τους γνήσιους κυρίαρχους του x. Πιο απλά ο άμεσος κυρίαρχος του x είναι ο πιο κοντινός, στο x, γνήσιος κυρίαρχός του. Ως παράδειγμα μπορούμε να πούμε ότι στο παραπάνω γράφημα ο άμεσος κυρίαρχος του κόμβου 7 είναι ο κόμβος 4, ή $idom(7) = 4$. Κάθε κόμβος του γραφήματος, εκτός της ρίζας r, έχει μοναδικό άμεσο κυρίαρχο κόμβο. Σε ένα γράφημα ροής $G = (V, E, r)$ λέμε ότι ο x είναι προηγηθείς ή προκάτοχος (predecessor) κόμβος του y και ότι ο y είναι διάδοχος (successor) του x εάν η ακμή (x, y) ανήκει στο E. Δηλώνουμε το σύνολο των προηγηθέντων κόμβων του y ως $pred(y)$ και το σύνολο των διαδόχων του y ως $succ(y)$. Επιπλέον, θα συμβολίζουμε $x \overset{*}{\rightarrow} y$, αν το x είναι απόγονος του y στο δέντρο T που είναι επικαλύπτον δέντρο (spanning tree) του G, και $x \overset{+}{\rightarrow} y$ εάν το x είναι γνήσιος απόγονος του y στο δέντρο T, δηλαδή αν $x \overset{*}{\rightarrow} y$ και $x \neq y$. Δοθέντος ενός δέντρου T, συμβολίζουμε ως T_x το υποδέντρο του T που έχει ρίζα το x, και ως $p_T(x)$ τον γονέα του x στο T. Αν το $T = (V, E)$ συμβολίζει το δέντρο που παίρνουμε από την αναζήτηση κατά βάθος ενός γραφήματος, τότε μία ακμή (x, y) αποτελεί ακμή του δέντρου αν $(x, y) \in E$, κατιούσα ακμή εάν $x \overset{+}{\rightarrow} y$, ανιούσα ακμή εάν $y \overset{+}{\rightarrow} x$ και ακμή διασταυρώσεως αν μεταξύ του x και του y δεν υπάρχει καμία συσχέτιση. Τέλος, μεταξύ δύο κόμβων x και y συμβολίζουμε ως $NCA(T, \{x, y\})$ τον πιο κοντινό κοινό πρόγονο (nearest common ancestor) του x και του y στο T.

Για να μειώσουμε τον χώρο στη μνήμη που χρειάζονται για να αποθηκευτούν τα σύνολα των κόμβων κυριαρχίας, χρησιμοποιούμε μία δομή δεδομένων η οποία ονομάζεται δέντρο κυριαρχίας (dominator tree) και συμβολίζεται ως I. Τα δέντρα κυριαρχίας βασίζονται στο γεγονός ότι κάθε κόμβος έχει μοναδικό άμεσο κυρίαρχο κόμβο και στη λεπτή ιδιότητα που ισχύει στους κόμβους κυριαρχίας η οποία λέει ότι:

Για όλους τους κόμβους εκτός του r ισχύει:

$$dom(x) = \{x\} \cup idom(x) \cup idom\big(idom(x)\big) \ldots . \cup \{r\}$$

Το δέντρο κυριαρχίας περιέχει όλους τους κόμβους του γραφήματος ροής και οι ακμές του αντικατοπτρίζουν τις σχέσεις κυριαρχίας μεταξύ των κόμβων σε αυτό. Στο δέντρο κυριαρχίας, κάθε κόμβος είναι παιδί του άμεσου κυρίαρχου του. Στο σχήμα 3.2 παρουσιάζεται το δέντρο κυριαρχίας του γραφήματος του σχήματος 3.1.



Σχήμα 3.2: Δέντρο κυριαρχίας.

3.2 Βασικοί Αλγόριθμοι

Το πρόβλημα της εύρεσης των κόμβων κυριαρχίας σε ένα γράφημα ροής έχει μεγάλη ιστορία. Κατά καιρούς έχουν αναπτυχθεί πολλές υλοποιήσεις και έχει γραφτεί μία αρκετά μεγάλη βιβλιογραφία που μελετά αυτό το πρόβλημα. Οι αρχικοί αλγόριθμοι, σε αυτό το πρόβλημα, παρουσίαζαν μεγάλη ασυμπτωτική πολυπλοκότητα στο χρόνο αλλά ήταν εύκολοι στην κατανόηση. Μεταγενέστερες μελέτες και εργασίες βελτίωσαν το χρονικό όριο θυσιάζοντας όμως την απλότητα και την ευκολία στην υλοποίηση. Ο

Prosser ήταν ο πρώτος ο οποίος εισήγαγε την ιδέα της κυριαρχίας σε ένα δοκίμιο το 1959.Τη χρησιμοποίησε για να αποδείξει ότι μπορεί με ασφάλεια να αναδιατάξει τις εντολές ενός κώδικα, αλλά παρόλα αυτά δεν παρουσίασε έναν αλγόριθμο που να υπολογίζει τους κόμβους κυριαρχίας από ένα πίνακα γειτνίασης.

3.2.1 Αλγόριθμος Purdom-Moore

Οι Purdom και Moore πρότειναν έναν αρκετά απλό αλγόριθμο για τον υπολογισμό των κόμβων κυριαρχίας, το 1972. Ο απλός αυτός αλγόριθμος, για να βρει τους κόμβους κυριαρχίας, αφαιρεί διαδοχικά όλους τους κόμβους x, για τους οποίους ισχύει $x \in V - r$, και στη συνέχεια εκτελεί μία αναζήτηση, κατά βάθος ή κατά πλάτος, από την ρίζα r. Εάν ένας κόμβος y γίνει μη προσπελάσιμος, μετά από αυτή την αναζήτηση, τότε ο x ανήκει στο σύνολο με τους κόμβους κυριαρχίας του y ($x \in dom(y)$). Αν και όπως φαίνεται ο αλγόριθμος αυτός είναι αρκετά απλός σαν ιδέα και εύκολα υλοποιήσιμος, ωστόσο έχει πολυπλοκότητα χρόνου $\Theta(n \times m)$, η οποία επιτυγχάνεται μόνο όταν το γράφημα αναπαρίσταται με λίστες γειτνίασης, το οποίο τον καθιστά αρκετά αργό στην πράξη.

3.2.2 Επαναληπτικός Αλγόριθμος

Έστω ότι μας δίνεται ένα γράφημα $G = (V, E)$, το οποίο παραμένει αμετάβλητο, και μας ζητείται να βρούμε τις σχέσεις κυριαρχίας μεταξύ των κόμβων από ένα ξεχωριστό κόμβο, r, ο οποίος αποτελεί τον αρχικό κόμβο του γραφήματος, ή αλλιώς τη ρίζα του γραφήματος. Στο γράφημα ροής ελέγχου ενός προγράμματος αυτός ο κόμβος θα αποτελούσε τον κόμβο Είσοδος (Entry), δηλαδή τον κόμβο με τον οποίο εισερχόμαστε στο πρόγραμμα. Οι Allen και Cocke, το 1972, ισχυρίστηκαν ότι αυτό το πρόβλημα μπορεί να λυθεί επαναληπτικά, και πρότειναν τον επαναληπτικό (iterative) αλγόριθμο. Παρουσιάζουμε μία εκδοχή αυτού του αλγορίθμου και στη συνέχεια μία βελτίωση που παρουσιάστηκε από τους Cooper, Harvey και Kennedy [9].

Ο αλγόριθμος δέχεται σαν είσοδο το γράφημα $G = (V, E)$ και τον κόμβο r, όπου $r \in V$. Στη συνέχεια εκτελεί αναζήτηση κατά βάθος, με αφετηριακό κόμβο τον r, και σημειώνει σε κάθε κόμβο έναν αριθμό, ο οποίος δείχνει την σειρά με την οποία

ανακαλύφθηκαν από την αναζήτηση κατά βάθος. Αυτή η σειρά ονομάζεται σειρά προ-διάταξης (preorder). Στη συνέχεια αρχικοποιούνται τα σύνολά κυριαρχίας για όλους τους κόμβους x ως κενά σύνολα, δηλαδή $dom(x) = \emptyset \ \forall x \in (V - r)$. Εξαίρεση αποτελεί ο αρχικός κόμβος, r, του οποίου σύνολο κυριαρχίας αρχικοποιούμε να είναι $dom(r) = r$. Τώρα τα σύνολα κυριαρχίας μπορούν να υπολογιστούν από την μοναδική μέγιστη λύση της εξίσωσης:

$$dom(x) = \left(\bigcap_{y \in pred(x)} dom(y) \right) \cup \{x\}, \qquad x \neq r$$

Καθώς ο αλγόριθμος τρέχει, αν συναντήσει ένα κόμβο x ο οποίος δεν ικανοποιεί την παραπάνω εξίσωση, αντικαθιστά το σύνολο $dom(x)$ με το δεξή σκέλος της παραπάνω εξίσωσης και συνεχίζει. Ο αλγόριθμος σταματά όταν σταματήσουν να γίνονται αλλαγές στα σύνολα κυριαρχίας των κόμβων του γραφήματος. Αυτό προϋποθέτει ότι ο αλγόριθμος θα κάνει τουλάχιστον δύο περάσματα από τους κόμβους του γραφήματος. Στο πρώτο πέρασμα θα υπολογίσει τα αρχικά σύνολα με τους κόμβους κυριαρχίας και στο δεύτερο πέρασμα θα τα ξαναϋπολογίσει για να τα συγκρίνει. Αν από τα δύο περάσματα κανένα από τα σύνολα των κόμβων κυριαρχίας δεν αλλάξει η διαδικασία σταματάει και ο αλγόριθμος έχει υπολογίσει επιτυχώς όλα τα σύνολα. Αν από τα δύο περάσματα υπάρχει έστω και ένα σύνολο το οποίο αλλάζει, τότε ο αλγόριθμος θα εκτελέσει και τρίτο πέρασμα και εν συνεχεία θα συγκρίνει τα σύνολα του δεύτερου και του τρίτου περάσματος. Αν τα σύνολα πάλι δεν είναι ίδια θα συνεχίσει τις επαναλήψεις έως ότου τα σύνολα δύο διαδοχικών περασμάτων να μην αλλάζουν.

Ο υπολογισμός καθενός ζεύγους τομών έχει πολυπλοκότητα στο χρόνο $O(n)$, αφού μπορεί να χρειαστεί, για τον υπολογισμό ενός συνόλου, να διατρέξουμε όλους τους κόμβους του γραφήματος, υπολογίζοντας τις τομές. Ο αλγόριθμος εκτελεί σε μία επανάληψη μία τομή ανά ακμή, άρα εκτελεί μία επανάληψη σε χρόνο $O(nm)$. Οι επαναλήψεις μπορεί στην χειρότερη περίπτωση να είναι $\Theta(n)$. Άρα συνολικά ο χρόνος εκτέλεσης της χειρότερης περίπτωση, ή αλλιώς η πολυπλοκότητα του αλγορίθμου στο χρόνο, είναι $O(m \times n^2)$, όταν η σειρά επεξεργασίας είναι η αντίστροφη της μετα-διάταξης (reverse postorder).

Μία απρόσεκτη υλοποίηση θα δυσκόλευε ακόμα περισσότερο τα πράγματα, αφού χρόνοι ίσοι ή κοντά στη χειρότερη περίπτωση θα συναντιόνταν συχνά. Αντίθετα μία προσεκτική υλοποίηση θα διευκόλυνε πολύ τα πράγματα κάνοντας τον αλγόριθμο πρακτικά πολύ πιο γρήγορο. Οι Cooper, Harvey και Kennedy [9] ήταν αυτοί που, στην προσπάθειά τους να αυξήσουν την αποδοτικότητα του αλγορίθμου, εισήγαγαν την ιδέα να αναπαρασταθούν τα σύνολα όλων των κόμβων κυριαρχίας στη δομή ενός δέντρου, μία ενδιάμεση αναπαράσταση που συγκλίνει στο δέντρο κυριαρχίας, και πρότειναν κάθε επαναληπτικό βήμα αυτού του αλγορίθμου να αποτελεί μια ενημέρωση αυτού του δέντρου. Πιο συγκεκριμένα, ξεκινάμε με κάθε δέντρο T , το οποίο αποτελεί υποσύνολο του G και έχει ρίζα το r, και επαναλαμβάνουμε τα ακόλουθα βήματα έως ότου να μην υπάρξουν νέες αλλαγές:

Βρίσκουμε ένα κόμβο x για τον οποίο ισχύει:

$$\mathrm{pred(x)} \cap \mathrm{T} \neq \emptyset \text{ και } \mathrm{p_T(x)} \neq \mathrm{NCA(T, pred(x))}$$

και κάνουμε αντικατάσταση του $\mathrm{p_T(x)}$ με το $\mathrm{NCA(T, pred(x))}$.

Όταν η διαδικασία τελειώσει το δέντρο T αποτελεί το δέντρο κυριαρχίας I. Η διαφορά με αυτή την υλοποίηση είναι ότι το $dom(x)$ είναι το σύνολο των προγόνων του x στο δέντρο κυριαρχίας και επιπλέον η τομή του $dom(x)$ και του $dom(y)$ είναι το σύνολο των προγόνων του $NCA(T, \{x, y\})$. Έτσι η πράξη της τομής μεταξύ δύο κόμβων απλοποιείται.

Εφόσον ο άμεσος κυρίαρχος κάθε κόμβου είναι μοναδικός, τότε ολόκληρο το δέντρο κυριαρχίας μπορεί να αποθηκευτεί απλά αποθηκεύοντας για κάθε κόμβο τον υποψήφιο άμεσο κυρίαρχό του. Με αυτή τη δομή, εκτός του ότι απλοποιείται η πράξη της τομής, η αποθήκευση όλων των συνόλων των κόμβων κυριαρχίας αποκτά πολυπλοκότητα χώρου $O(n)$, από $O(n^2)$ που θα είχε, στην χειρότερη περίπτωση, αν αποθηκεύαμε ξεχωριστά για όλους τους κόμβους όλους του τους κυρίαρχους. Και πλέον το σύνολο $dom(x)$ μπορεί να ανασυγκροτηθεί, όπως είπαμε και παραπάνω, από τον τύπο:

$$dom(x) = \{x\} \cup idom(x) \cup idom\big(idom(x)\big) \ldots . \cup \{r\}$$

Επιπλέον, η σειρά με την οποία επεξεργάζεται ο αλγόριθμος τους κόμβους, από το δέντρο της αναζήτησης κατά βάθος, δεν πρέπει να αφήνεται στην τύχη. Με την κατάλληλη σειρά επεξεργασίας μπορεί να καταφέρουμε να μειώσουμε των αριθμό των επαναλήψεων, με αποτέλεσμα ο αλγόριθμος μας να τερματίσει πιο γρήγορα. Παραπάνω αναφέραμε ότι η σειρά με την οποία ανακαλύπτονται οι κόμβοι από την αναζήτηση κατά βάθος αποτελεί την σειρά προ-διάταξης. Η σειρά μετά-διάταξης (postorder) αποτελεί την σειρά με την οποία ένας κόμβος εμφανίζεται μετά από όλους τους απογόνους του. Ουσιαστικά διατρέχουμε το δέντρο της αναζήτησης κατά βάθος από αριστερά προς τα δεξιά και από κάτω προς τα πάνω. Η καλύτερη σειρά για να επεξεργαστούν οι κόμβοι από τον επαναληπτικό αλγόριθμο είναι αντίστροφη της μετά-διάταξης (reverse postorder). Δηλαδή έχοντας σημειωμένο σε κάθε κόμβο τον αριθμό της μετά-διατακτικής σειράς, ο αλγόριθμος επεξεργάζεται τους κόμβους από το τέλος προς την αρχή.

Με αυτό τον τρόπο πριν επεξεργαστούμε ένα κόμβο είμαστε σίγουροι ότι όλοι οι πρόγονοί του έχουν επεξεργαστεί. Μοναδικό πρόβλημα σε αυτή τη σειρά αποτελούν οι ανιούσες ακμές, οι οποίες είναι οι υπαίτιες για την δημιουργία κύκλων στα δέντρα της αναζήτησης κατά βάθος. Αν το δέντρο της αναζήτησης κατά βάθος είναι άκυκλο τότε τα σύνολα των κόμβων κυριαρχίας υπολογίζονται σωστά από την πρώτη επανάληψη. Οι Kam και Ullman [23] έδειξαν ότι οι επαναλήψεις που χρειάζονται για να τερματίσει ο αλγόριθμος, και να υπολογίσει σωστά τα σύνολα των κόμβων κυριαρχίας, είναι $d(G,D)+3$, όπου το $d(G,D)$ αναπαριστά τον μέγιστο αριθμό των ανιουσών ακμών σε μονοπάτι που δεν έχει κύκλο στο γράφημα G σε σχέση με το δέντρο της αναζήτησης κατά βάθος D. Αυτή η σχέση ισχύει μόνο όταν η σειρά επεξεργασίας των κόμβων είναι η αντίστροφη της μετά-διάταξης.

Ένα παράδειγμα του τρόπου με τον οποίο εκτελείται αυτός ο αλγόριθμος παρουσιάζεται στο σχήμα 3.3. Ο κόμβος 6 αποτελεί την ρίζα του γραφήματος. Ο επαναληπτικός αλγόριθμος θα εκτελεστεί 4 φορές. Αρχικά γίνεται η αρχικοποίηση των κόμβων. Στη συνέχεια ο αλγόριθμος κάνει δύο περάσματα και συγκρίνει τα αποτελέσματα. Επειδή ο άμεσος κυρίαρχος του κόμβου 2 αλλάζει ο αλγόριθμος ξαναεκτελείται. Στην τρίτη επανάληψη πάλι έχουμε αλλαγή, στον κόμβο 3 αυτή τη φορά. Τελικά, στην τέταρτη επανάληψη οι άμεσοι κυρίαρχοι δεν αλλάζουν και ο αλγόριθμος τερματίζει.

Σχήμα 3.3: Παράδειγμα εκτέλεσης του επαναληπτικού αλγορίθμου. (Από την εργασία των Cooper, Harvey και Kennedy [9])

Ακόμα και μετά από αυτή τη τόσο προσεκτική υλοποίηση το όριο του χρόνου της χειρότερης περίπτωσης παραμένει να είναι $O(m \times n^2)$. Ωστόσο, ο αλγόριθμος πρακτικά είναι πολύ πιο γρήγορος και μόνο σε πολύ ιδιαίτερες περιπτώσεις προσεγγίζει το χρόνο της χειρότερης περίπτωσης. Στην πραγματικότητα, όσο λιγότερους κύκλους έχει το γράφημα τόσο πιο πολύ προσεγγίζει τη γραμμική πολυπλοκότητα στο χρόνο. Πειραματικές μετρήσεις διαφόρων εργασιών όπως της εργασίας [3] έδειξαν ότι σε γραφήματα με λογικά μεγέθη, δηλαδή γραφήματα που συναντάμε στην πραγματικότητα και κυρίως στα γραφήματα ροής ελέγχου ενός προγράμματος, ο επαναληπτικός αλγόριθμος υπολογίζει τα σύνολα των κόμβων κυριαρχίας πιο γρήγορα από όλους τους γνωστούς αλγορίθμους. Αυτό το γεγονός, σε συνδυασμό με το ότι ο επαναληπτικός αλγόριθμος είναι πιο απλός σαν ιδέα και σαν υλοποίηση, τον καθιστά έναν από τους πιο πολυχρησιμοποιημένους αλγόριθμους για τον υπολογισμό των συνόλων των κόμβων κυριαρχίας. Ο λόγος, για τον οποίο ο επαναληπτικός αλγόριθμος είναι πιο γρήγορος, έγκειται στο γεγονός ότι αν και οι άλλοι αλγόριθμοι έχουν καλύτερη ασυμπτοτική πολυπλοκότητα στο χρόνο, παρόλα αυτά η επεξεργασία καθενός κόμβου αποτελεί πιο πολύπλοκη πράξη σε αυτούς. Φυσικά, σε αρκετά μεγάλα γραφήματα οι αλγόριθμοι με καλύτερη ασυμπτοτική πολυπλοκότητα υπερτερούν του επαναληπτικού.

3.2.3 Αλγόριθμος Lengauer-Tarjan

Το 1974, ο Tarjan πρότεινε έναν αλγόριθμο, για τον υπολογισμό των κόμβων κυριαρχίας, ο οποίος χρησιμοποιεί την αναζήτηση κατά βάθος και έναν αλγόριθμο εύρεσης ένωσης (union - find) για να πετύχει ασυμπτοτική πολυπλοκότητα χρόνου $O(m \times \log_{2+\lfloor m/n \rfloor} n)$. Πέντε χρόνια αργότερα, οι Lengauer και Tarjan δημιούργησαν, στηριγμένοι στον προηγούμενο, έναν αλγόριθμο με σχεδόν γραμμική πολυπλοκότητα χρόνου. Για την ακρίβεια ο αλγόριθμος έχει πολυπλοκότητα $O(n \times \alpha(m,n))$, όπου η συνάρτηση $\alpha(m,n)$ αποτελεί την αντίστροφη συνάρτηση Ackermann, η οποία παρουσιάζει εξαιρετικά αργή αύξηση. Και οι δύο αλγόριθμοι βασίστηκαν στην ιδέα ότι οι κυρίαρχοι ενός κόμβου πρέπει να βρίσκονται σε υψηλότερη θέση στο επικαλύπτον δέντρο που δημιουργείται από την αναζήτηση κατά βάθος. Αυτό δίνει μία αρχική υπόθεση για του κόμβους κυριαρχίας, η οποία διορθώνεται με το δεύτερο πέρασμα, του αλγορίθμου, από τους κόμβους. Ο αλγόριθμος βασίζεται στην αποδοτικότητα της τεχνικής εύρεσης ένωσης για τον καθορισμό του χρονικού ορίου. Αυτή η ενότητα βασίζεται στην εργασία [11].

Ο αλγόριθμος Lengauer-Tarjan ξεκινάει με την αναζήτηση κατά βάθος ενός γραφήματος G από τοr, σημειώνοντας σε κάθε κόμβο τη σειρά στην οποία ανακαλύφθηκε (σειρά προ-διάταξης ή preorder). Το αποτέλεσμα της αναζήτησης κατά βάθος είναι ένα δέντρο D. Για απλότητα, αναφερόμαστε στους κόμβους του G με τον αριθμό προ-διάταξης. Έτσι, η σχέση $v \leq u$ σημαίνει ότι το v έχει μικρότερο αριθμό προ-διάταξης από το u και άρα το v ανακαλύφθηκε πριν το u από την αναζήτηση κατά βάθος. Ο αλγόριθμος βασίζεται στην ιδέα των semidominators, οι οποίοι, όπως αναφέραμε και παραπάνω, δίνουν μία αρχική προσέγγιση των άμεσων κυρίαρχων κόμβων. Ένα μονοπάτι $P = (u = v_0, v_1, \ldots, v_{k-1}, v_k = v)$, στο G, ονομάζεται μονοπάτι semidominator εάν $v_i > v$ για $1 \leq i \leq k-1$. Ο semidominator του v ορίζεται ως:

$$sdom(v) = \min\{u | \text{για το οποίο υπάρχει ένα semidominator μονοπάτι από το } u \text{ στο } v\}$$

Κάθε κόμβος $v \neq r$ συσχετίζεται με τον $idom(v)$ και με τον $sdom(v)$ με τον ακόλουθο τρόπο:

$$idom(v) \overset{*}{\rightarrow} sdom(v) \overset{+}{\rightarrow} v$$

Οι semidominators και οι άμεσοι κυρίαρχοι υπολογίζονται βρίσκοντας την ελάχιστη τιμή του $sdom$ στα μονοπάτια του δέντρου D, για κάθε κόμβο $w \neq r$, με τον ακόλουθο τρόπο:

$$sdom(w) = \min\{s_w(v) | v \in pred(w)\}$$

όπου το s_w είναι μία συνάρτηση η οποία ορίζεται, από το $pred(w)$ στο V, ως:

$$s_w(v) = \begin{cases} v, & v \leq w \\ \min\{ sdom(u) \mid NCA(D, \{v, w\}) \xrightarrow{+} u \xrightarrow{*} v \}, & v > w \end{cases}$$

Ομοίως, ο άμεσος κυρίαρχος μπορεί να υπολογιστεί αποτιμώντας την συνάρτηση e η οποία ορίζεται, από το $V - r$ στο V, ως ακολούθως:

$$e(w) = \text{argmin}\{sdom(u) | sdom(w) \xrightarrow{+} u \xrightarrow{*} w\}$$

Για κάθε $w \neq r$, το $e(w)$ συσχετίζεται με τον άμεσο κυρίαρχο του w, με τη σχέση $idom(e(w)) = idom(w)$. Επιπλέον, εάν το $sdom(e(w)) = sdom(w)$ τότε το $idom(w) = sdom(w)$.

Οι κόμβοι επεξεργάζονται με την αντίθετη σειρά της μετά-διάταξης (reverse postorder) για να διασφαλιστεί ότι όλες οι πληροφορίες που χρειάζονται είναι διαθέσιμες, κατά την επεξεργασία ενός κόμβου. Οι κύριοι υπολογισμοί εκτελούνται από μία δομή δεδομένων η οποία ονομάζεται link-eval, την οποία εισήγαγε ο Tarjan. Δοθέντος ενός δέντρου T, στο V, και των πραγματικών τιμών, $value(v)$, για κάθε $v \in V$, η δομή δεδομένων link-eval κατασκευάζει το δάσος F, το οποίο είναι υπογράφημα του T, υπό τους ακόλουθους όρους:

$link(v, x)$: Κάνει την ανάθεση $value(v) \leftarrow x$ και προσθέτει την ακμή $(p_T(v), v)$ στο F. Αυτό συνδέει το δέντρο που έχει ρίζα το v, στο F, στο δέντρο που έχει ρίζα το $p_T(v)$, στο F.

$eval(v)$: Εάν $v = root_F(v)$, τότε επιστρέφει το v. Αλλιώς, επιστρέφει οποιοδήποτε κόμβο, με μικρότερη τιμή μεταξύ των κόμβων u, που ικανοποιεί την σχέση $root_F \xrightarrow{+}_F u \xrightarrow{*}_F v$.

Αρχικά, κάθε κόμβος v στο V αποτελεί ξεχωριστό δέντρο στο δάσος F. Λέμε ότι ο κόμβος v είναι συνδεδεμένος αν η εντολή $link(v, \bullet)$ έχει εκτελεστεί. Με σκοπό να γίνει πιο αποτελεσματική η εκτέλεση του τελεστή $eval$, η δομή δεδομένων linked-eval εφοδιάζεται με μία τεχνική συμπίεσης μονοπατιού (path compression) και με άλλες τεχνικές. Έτσι, αντί να ασχολούμαστε με το F απευθείας, κατασκευάζουμε ένα εικονικό δάσος (virtual forest), το οποίο συμβολίζουμε ως VF, και διασφαλίζουμε ότι όταν εφαρμόζεται ο τελεστή $eval$ στο VF επιστρέφεται το ίδιο αποτέλεσμα με το αν εφαρμοζόταν ο ίδιος τελεστής στο F.

Στον αλγόριθμο Lengauer-Tarjan, το $T = D$ και για κάθε $v \in V$ το $value(v)$ είναι ίσο με το v αρχικά. Αφού επεξεργαστεί το v για πρώτη φορά, το $value(v)$ γίνεται ίσο με το $sdom(v)$. Κάθε κόμβος w επεξεργάζεται τρεις φορές. Την πρώτη φορά που επεξεργαζόμαστε τον w, το $sdom(w)$ υπολογίζεται εκτελώντας το $eval(u)$ για κάθε $u \in pred(w)$, και κατ' αντιστοιχία υπολογίζοντας το $s_w(u)$. Εν συνεχεία, το w εισάγεται σε μία λίστα η οποία συνδέεται με τον κόμβο $sdom(w)$ και εκτελείται ο τελεστής $link\big(w, sdom(w)\big)$. Ο αλγόριθμος επεξεργάζεται ξανά τον w αφού υπολογιστεί το $sdom(v)$, όπου $parent[v] = sdom(w)$ και $v \overset{*}{\rightarrow} w$. Αυτή τη φορά εκτελείται ο τελεστής $eval(w)$, και έτσι υπολογίζεται το $e(w)$. Τελικά, ο άμεσος κυρίαρχος κάθε κόμβου x, με $x \in V - r$, παράγεται από το $e(x)$ με ένα πέρασμα από όλους τους κόμβους στη σειρά προ-διάταξης.

Με μία απλή υλοποίηση της δομής δεδομένων linked-eval, χρησιμοποιώντας μόνο συμπίεση μονοπατιού, ο αλγόριθμος Lengauer-Tarjan, όπως αναφέραμε και παραπάνω, έχει πολυπλοκότητα χρόνου $O(m \times \log_{2+\lfloor m/n \rfloor} n)$. Με πιο περίτεχνες στρατηγικές σύνδεσης οι οποίες εξασφαλίζουν ότι το VF είναι ισορροπημένο (balanced), ο αλγόριθμος τρέχει σε χρόνο $O(m \times \alpha(m, n))$.

3.2.4 Αλγόριθμοι Γραμμικοί στο Χρόνο

Παρά το γεγονός ότι ο αλγόριθμος Lengauer-Tarjan πετυχαίνει πολύ καλό ασυμπτοτικό χρόνο ωστόσο δεν είναι γραμμικός. Έτσι ένα σημαντικό πρόβλημα που απασχόλησε αρκετούς ερευνητές ήταν η ανακάλυψη πραγματικά γραμμικών αλγορίθμων. Στη διάρκεια αυτής της προσπάθειας προτάθηκαν διάφοροι αλγόριθμοι.

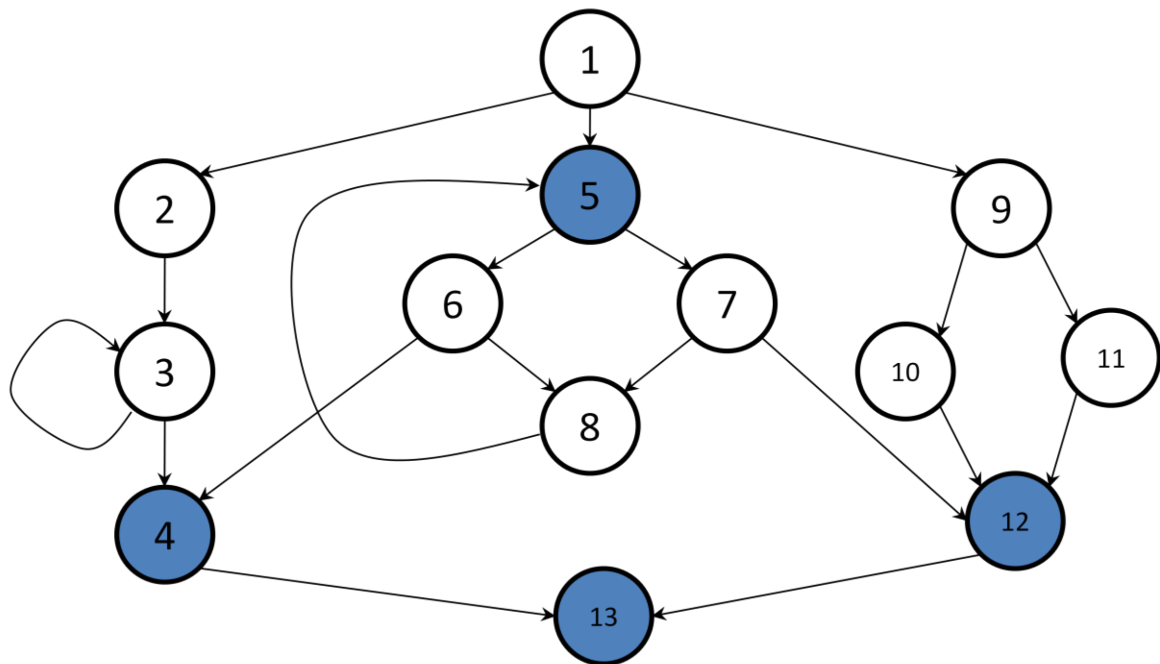
Κάποιοι από τους οποίους απεδείχθησαν ότι δεν ήταν σωστή. Για περισσότερες πληροφορίες παραπέμπουμε στο άρθρο Buchsbaum et al [20,24].

3.3 Όρια Κυριαρχίας

Το όριο κυριαρχίας (dominance frontier) ενός κόμβου x σε ένα γράφημα ροής, το οποίο συμβολίζεται ως $DF(x)$, είναι εκείνο το σύνολο των κόμβων y για το οποίο το x είναι κυρίαρχος ενός προκατόχου του y αλλά δεν είναι γνήσιος κυρίαρχος του ίδιου του y. Δηλαδή:

$$DF(x) \equiv \{y \mid (\exists\, p \in pred(y))(x \in dom(p) \kappa\alpha\iota\ x \notin stdom(y))\}$$

Ένα παράδειγμα αυτού φαίνεται στο σχήμα 3.4. Οι κόμβοι κυριαρχίας του 5 στο παρακάτω σχήμα είναι οι $dom(5) = \{5,6,7,8\}$. Τα όρια κυριαρχίας του κόμβου 5 αποτελούν οι κόμβοι $DF(5) = \{4,5,12\}$.



Σχήμα 3.4: Όρια Κυριαρχίας (οι μπλε κόμβοι ανήκουν στα όρια κυριαρχίας του 5).

Ο υπολογισμός του συνόλου $DF(x)$ από τον ορισμό προϋποθέτει εξαντλητική αναζήτηση στο δέντρο κυριαρχίας. Ο συνολικός χρόνος που απαιτείται για των υπολογισμό αυτών των συνόλων, για όλους τους κόμβους, είναι τετραγωνικός στον αριθμό των κόμβων , ακόμα και αν αυτά τα σύνολα είναι μικρά. Η επίτευξη μίας πολυπλοκότητας χρόνου γραμμική στο μέγεθος του συνόλου του ορίου κυριαρχίας του x, $|DF(x)|$, προϋποθέτει άλλη προσέγγιση από αυτή του ορισμού. Για την

καινούρια προσέγγιση χρειάζεται να ορίσουμε δύο ενδιάμεσα σύνολα για κάθε κόμβο, τα $DF(x)_{local}$ και $DF(x)_{up}$. Έτσι πλέον το $DF(x)$ μπορεί να υπολογιστεί από τον τύπο:

$$DF(x) = DF_{local}(x) \cup \bigcup_{z \in Children(x)} DF_{up}(z)$$

όπου το σύνολο $Children(x)$ αποτελείται από τους κόμβους που έχουν άμεσο κυρίαρχο τον x, ή αλλιώς αποτελείται από τους κόμβους που είναι παιδιά του x στο δέντρο κυριαρχίας. Αυτό σημαίνει ότι μερικοί κόμβοι που ανήκουν στο σύνολο $DF(z)$, όπου το z δεν είναι η ρίζα του δέντρου κυριαρχίας, μπορεί να συμμετέχουν και στο σύνολο $DF(idom(z))$.

Δοθέντος ενός κόμβου x, μερικοί διάδοχοι του x στο γράφημα μπορεί να συμμετέχουν στο σύνολο $DF(x)$. Αυτοί οι κόμβοι αποτελούν την τοπική συνεισφορά στο σύνολο $DF(x)$ και ορίζεται ως:

$$DF_{local}(x) \equiv \{ y \in succ(x) \mid y \notin stdom(x) \}$$

Το σύνολο $DF_{up}(z)$ που αποτελεί μέρος του συνόλου $DF(idom(z))$ υπολογίζεται από τον ακόλουθο τύπο, ο οποίος αποτελεί και τον ορισμό του:

$$DF_{up}(z) \equiv \{ y \in DF(z) \mid y \notin stdom(idom(z)) \}$$

Η παραπάνω προσέγγιση υπολογίζει σωστά τα σύνολα των ορίων κυριαρχίας για όλους τους κόμβους ενός γραφήματος. Μία αναλυτική απόδειξη αυτής της ορθότητας του αλγορίθμου παρατίθεται στην εργασία [12].

Ας θεωρήσουμε ένα γράφημα ροής το οποίο αποτελείται από n κόμβους και m ακμές. Ο υπολογισμός του συνόλου $DF(x)_{local}$ για κάθε x, απαιτεί να εξεταστεί κάθε διάδοχος κόμβος του x. Αυτό προϋποθέτει να εξεταστούν όλες οι ακμές του γραφήματος ροής, οπότε η πολυπλοκότητα χρόνου για τον υπολογισμού του τοπικού τμήματος των ορίων κυριαρχίας απαιτεί χρόνο $O(m)$. Για τον υπολογισμό του συνόλου $DF(x)_{up}$ για κάθε x, απαιτείται να εξεταστεί κάθε κόμβος που συμμετέχει στο σύνολο $DF(x)$. Άρα η πολυπλοκότητα χρόνου για τον υπολογισμό αυτού του συνόλου είναι $O(|DF(x)|)$. Έτσι, η πολυπλοκότητα χρόνου για τον υπολογισμό του $DF(x)$ για κάθε x προκύπτει να είναι $O(m + |DF(x)|)$, το οποίο στη χειρότερη

περίπτωση γίνεται $O(m + n^2)$. Ωστόσο, ο υπολογισμός των συνόλων των ορίων κυριαρχίας πρακτικά απαιτεί γραμμικό χρόνο.

4 ΔΟΜΗ ΣΤΑΤΙΚΗΣ ΜΟΝΑΔΙΚΗΣ ΑΝΑΘΕΣΗΣ (STATIC SINGLE ASSIGNMENT FORM - SSA)

Στην βελτίωση που προκαλείται στους μεταγλωττιστές, η επιλογή δομής δεδομένων απευθείας επηρεάζει την ισχύ και την αποτελεσματικότητα της βελτιστοποίησης των προγραμμάτων. Μία απρόσεκτη επιλογή δομής δεδομένων μπορεί να δυσχεράνει την βελτιστοποίηση ή να επιβραδύνει την μεταγλώττιση σε σημείο όπου οι προηγμένες δυνατότητες βελτιστοποίησης να γίνουν ανεπιθύμητες. Η δομή στατικής και μοναδικής ανάθεσης, ή αλλιώς δομή SSA, και τα γραφήματα εξαρτήσεων (control dependence graph) έχουν προταθεί να αναπαριστούν τη ροή πληροφορίας και τη ροή ελέγχου σε ένα πρόγραμμα. Αυτές οι δύο άσχετες, προηγουμένως , μεταξύ τους τεχνικές εφοδιάζουν με ισχύ και αποδοτικότητα ένα νέο σύνολο με βελτιστοποιήσεις κώδικα. Ωστόσο, αν και οι δύο αυτές τεχνικές είναι ελκυστικές, η δυσκολία στην κατασκευή και το πιθανό μέγεθός τους αποθάρρυναν τη χρήση τους. Το κεφάλαιο αυτό βασίζεται στην εργασία των Cytron, Ferrante, Rossen και Wegman [12]

4.1 Περιγραφή της SSA Δομής

Η SSA δομή αναπτύχθηκε την δεκαετία του ’80 από τους Cytron, Ferrante, Rossen και Wegman και αποτελεί, όπως αναφέραμε παραπάνω, μία ενδιάμεση γλώσσα η οποία βασίστηκε στον ενδιάμεσο κώδικα τριών διευθύνσεων. Η SSA δομή έχει πάρει το όνομα της από το γεγονός ότι κάθε ανάθεση γίνεται σε μοναδική μεταβλητή. Αν μία μεταβλητή V έχει περισσότερες από μία αναθέσεις τότε ένα νέο σύνολο μεταβλητών $V_1, V_2, V_3 \dots$ δημιουργείται και αντικαθιστούν την αρχική μεταβλητή. Επιπλέον, κάθε χρήση της μεταβλητής μετονομάζεται έτσι ώστε να ταιριάζει στην ανάθεση που φτάνει σε αυτή τη χρήση. Παρακάτω παρουσιάζεται ένα απλό παράδειγμα μετασχηματισμού κώδικα στην SSA δομή. Όπως φαίνεται η μεταβλητή V έχει δύο αναθέσεις και κάθε ανάθεση έχει από μία χρήση. Επειδή υπάρχουν δύο αναθέσεις στη V γι’ αυτό και δημιουργούνται δύο νέες μεταβλητές (V_1 και V_2), οι οποίες αντικαθιστούν την αρχική μεταβλητή, στην SSA δομή. Επειδή στην ανάθεση του X το V έχει την τιμή 4, χρησιμοποιείται η μεταβλητή V_1, ενώ αντίστοιχα επειδή στην ανάθεση του Y το V έχει την τιμή 6, χρησιμοποιείται η μεταβλητή V_2.

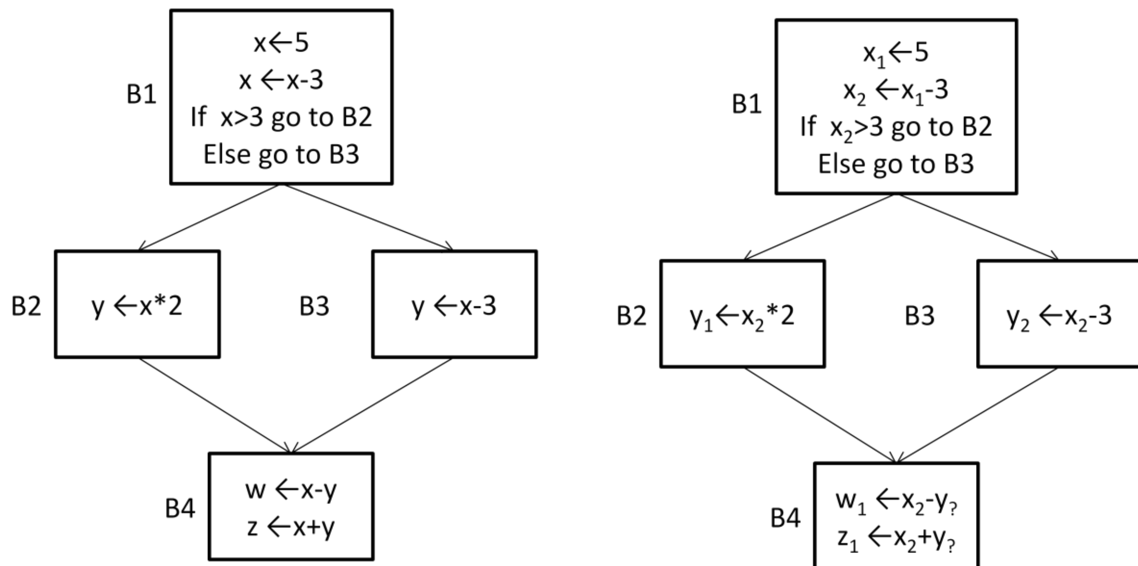
$$
\begin{array}{ll}
V \leftarrow 4 & V_1 \leftarrow 4 \\
X \leftarrow V+5 & X \leftarrow V_1+5 \\
V \leftarrow 6 & V_2 \leftarrow 6 \\
Y \leftarrow V+7 & Y \leftarrow V_2+7
\end{array}
$$

Αριστερά ο ενδιάμεσος κώδικας και δεξιά ο κώδικας σε SSA δομή.

Η SSA δομή επιτυγχάνει να αναπαραστήσει σε πιο συμπαγή μορφή την ροή της πληροφορίας, γνωστή και ως def-use αλυσίδες. Αν μία μεταβλητή έχει D ορισμούς και U χρήσεις, τότε μπορεί να δημιουργηθούν $D \times U$ def-use αλυσίδες. Η ίδια πληροφορία, στην SSA δομή, έχει το πολύ E def-use αλυσίδες, όπου το E είναι ο αριθμός των κατευθυνόμενων ακμών στο γράφημα ροής. Επιπλέον, οι def-use αλυσίδες μπορούν να ενημερωθούν πιο εύκολα στην SSA δομή, όταν εφαρμόζονται βελτιστοποιήσεις. Τέλος, οι def-use αλυσίδες, για κάθε μεταβλητή στην δομή SSA, παίρνουν τη μορφή λίστα στις οποίες αποθηκεύονται τα σημεία που γίνεται χρήση των μεταβλητών στον κώδικα, αφού κάθε χρήση της μεταβλητής είναι χρήση της τιμής της μοναδικής ανάθεσης.

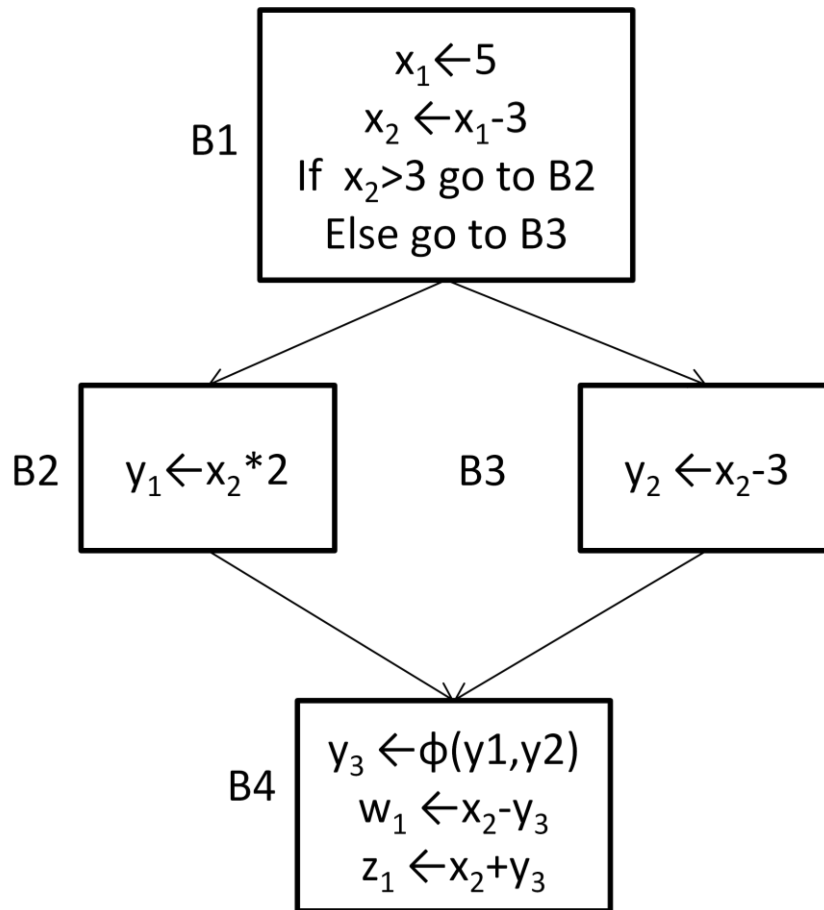
Όπως αναφέραμε στην SSA δομή υπάρχουν μοναδικές αναθέσεις σε κάθε μεταβλητή. Αυτό θα μπορούσε να δουλεύει αποτελεσματικά αν τα γραφήματα ροής τον προγραμμάτων δεν περιέχουν κόμβους διακλάδωσης (branch node) και κόμβους ένωσης (join node). Ένας κόμβος, στο γράφημα ροής, λέγεται κόμβος διακλάδωσης όταν από αυτόν εξέρχονται περισσότερες από δύο ακμές και ένας κόμβος λέγεται κόμβος ένωσης όταν σε αυτόν εισέρχονται περισσότερες από μία ακμές. Για να δούμε το πρόβλημα που δημιουργείται από αυτό στον κώδικα ας εξετάσουμε το σχήμα 4.1. Στο βασικό μπλοκ $B1$ το x έχει δύο αναθέσεις. Αυτός είναι και ο λόγος που δημιουργούνται δύο νέες μεταβλητές τα x_1 και x_2. Η εντολή if στη συνέχεια μπορεί να δώσει τον έλεγχο ή στο $B2$ ή στο $B3$. Βεβαίως, εδώ το x_2 έχει πάντα την τιμή 2, οπότε ο έλεγχος πάει πάντα στο $B3$. Ας υποθέσουμε όμως ότι ο έλεγχος μπορεί να πάει και στο $B2$. Τα μπλοκ $B2$ και $B3$ έχουν το καθένα μία ανάθεση στο y. Τέλος ο κόμβος $B4$ έχει μία ανάθεση στο w και μία στο z, όπου κάθε μία χρησιμοποιεί το y. Για τις αναθέσεις στο $B4$ για το w και το z δεν γνωρίζουμε αν η τιμή του y είναι $x * 2$ ή $x - 3$. Οπότε, στην SSA δομή, δεν γνωρίζουμε αν θα

χρησιμοποιήσουμε το y_1 ή το y_2. (Στο σχήμα ο $B1$ αποτελεί ένα κόμβο διακλάδωσης και ο $B4$ ένα κόμβο ένωσης.)



Σχήμα 4.1: Αριστερά το γράφημα ροής ενός προγράμματος και δεξιά το ίδιο γράφημα σε SSA δομή.

Για να γνωρίζει το πρόγραμμα, σε κάθε τέτοια περίπτωση, ποια μεταβλητή να χρησιμοποιήσει, εισάγουμε μία ειδική ανάθεση η οποία ονομάζεται φ-συνάρτηση. Αυτές οι συναρτήσεις εισάγονται σε κόμβους ένωσης, όπως ο $B4$, και εξετάζουν από ποια διαδρομή φτάνει ο έλεγχος σε αυτόν τον κόμβο. Στο παραπάνω παράδειγμα στο μπλοκ $B4$ θα προστεθεί μία νέα ανάθεση για το y. Πλέον, το y θα παίρνει την τιμή $x * 2$ αν ο έλεγχος έρχεται από το $B2$ και την τιμή $x - 3$ αν ο έλεγχος έρχεται από το $B3$. Το νέο γράφημα ροής θα έχει την μορφή του σχήματος 4.2. Πλέον όλες η χρήσεις της y_1 και της y_2 γίνονται χρήσεις της μεταβλητής y_3 στο μπλοκ $B4$. Η y_3 είναι η μεταβλητή στην οποία κάνει ανάθεση η ειδική φ-συνάρτηση και αναθέτει την τιμή y_1, αν ο έλεγχος έρχεται από το $B2$, ή την τιμή y_2, αν ο έλεγχος έρχεται από το $B3$. Είναι σημαντικό οι αναθέσεις των φ-συναρτήσεων να γίνονται στην αρχή του βασικού μπλοκ. Έτσι, μία παλιά μεταβλητή V μπορεί να αντικατασταθεί με ένα σύνολο μεταβλητών $V_1, V_2, V_3, \dots$ και σε κάθε χρήση της μεταβλητής V_i φτάνει μόνο μία ανάθεση του V_i. Αυτό απλοποιεί τις καταγραφές για κάθε μεταβλητή συμβάλλοντας στην αποδοτική επίτευξη των βελτιστοποιήσεων.



Σχήμα 4.2: Γράφημα ροής στην SSA δομή.

4.2 Κατασκευή της SSA Δομής

Όταν ο κώδικας μεταφράζεται σε ενδιάμεσο κώδικα κάθε δήλωση αποτιμάται σε ορισμένες εκφράσεις και τα αποτελέσματα χρησιμοποιούνται είτε για τον καθορισμό μίας διακλάδωσης είτε για να κάνουν ανάθεση σε μία μεταβλητή. Μία ανάθεση A είναι της μορφής $LHS(A) \leftarrow RHS(A)$, όπου το αριστερό μέλος αποτελεί μία πλειάδα με μεταβλητές στόχους $(V, U, \dots)$ και το δεξιό μέλος αποτελεί μία πλειάδα με εκφράσεις, με το ίδιο μέγεθος. Σε κάθε μεταβλητή-στόχο της $LHS(A)$ αντιστοιχίζεται η ανάλογη τιμή στην $RHS(A)$. Με αυτόν τον τύπο δήλωσης μπορούμε να χειριστούμε και πιο πολύπλοκες δομές, όπως κλήσεις διαδικασιών, έτσι ώστε να καταστήσουμε σαφές ποιες μεταβλητές χρησιμοποιούνται ή αλλάζουν από τις δηλώσεις στο πρόγραμμα. Το μόνο καινοτόμο στη χρήση των πλειάδων είναι ότι εφοδιαζόμαστε με ένα απλό και ομοιόμορφο τρόπο για την αναπαράσταση των αποτελεσμάτων της ανάλυσης.

Το να μεταφράσουμε τον ενδιάμεσο κώδικα σε SSA μορφή είναι μία εργασία με δύο βήματα. Στο πρώτο βήμα εισάγουμε κάποιες αναθέσεις φ-συναρτήσεων σε όσους κόμβους ένωσης χρειάζεται στο γράφημα ροής. Στο δεύτερο βήμα, παράγονται νέες μεταβλητές και αντικαθιστούν τις παλιές, έτσι ώστε σε κάθε μεταβλητή να γίνεται μία ανάθεση. Μία δήλωση ανάθεσης μπορεί να είναι είτε μία συνηθισμένη ανάθεση, την οποία προκαλεί το πρόγραμμα, είτε μία ανάθεση φ-συνάρτησης.

Η φ-συνάρτηση, η οποία εισάγεται στην αρχή ενός μπλοκ B, έχει την μορφή $V \leftarrow \varphi(R, S, \dots)$, όπου τα $V, R, S, \dots$ είναι μεταβλητές. Ο αριθμός των μεταβλητών που βρίσκονται μέσα στη φ-συνάρτηση $(R, S, \dots)$ είναι ίσος με τον αριθμό των προκατόχων του μπλοκ B στο γράφημα ροής. Οι προκάτοχοι του B μπορούν να ταξινομηθούν με αυθαίρετο τρόπο αλλά σε σταθερή σειρά. Έτσι, ο $j - οστός$ όρος στη φ-συνάρτηση αναφέρεται στον $j - οστό$ προκάτοχο του μπλοκ B στο γράφημα ροής. Αν ο έλεγχος δοθεί στο B μπλοκ από τον $j - οστό$ προκάτοχο, τότε ένας μηχανισμός υποστήριξης θυμάται το j στο χρόνο εκτέλεσης όταν εκτελείται η φ-συνάρτηση στο B. Η τιμή της φ-συνάρτησης είναι μόνο ο j τελεστής. Κάθε εκτέλεση της φ-συνάρτησης χρησιμοποιεί μόνο ένα τελεστή, αλλά ποιον εξαρτάται από την ροή του ελέγχου πριν την είσοδο στο μπλοκ B. Κάποιες παραλλαγές της φ-συνάρτησης είναι χρήσιμες για ειδικές περιπτώσεις. Για παράδειγμα, σε κάθε φ-συνάρτηση μπορεί να σημειωθεί το μπλοκ στο οποίο εμφανίζεται. Όταν το γράφημα ροής μίας γλώσσας είναι κατάλληλα περιορισμένο, τότε σε κάθε φ-συνάρτηση μπορούν να σημειωθούν πληροφορίες για βρόχους ή για υπό όρους συνθήκες του κώδικα.

Η μετάφραση στην SSA δομή έχει σαν αποτέλεσμα την αντικατάσταση του αρχικού προγράμματος με ένα καινούριο με το ίδιο γράφημα ροής. Για κάθε μεταβλητή στο αρχικό πρόγραμμα, πρέπει να ισχύουν οι ακόλουθοι όροι:

I. Αν δύο μη μηδενικά μονοπάτια $X \overset{+}{\rightarrow} Z$ και $Y \overset{+}{\rightarrow} Z$ συγκλίνουν στον κόμβο Z, και οι κόμβοι X και Y περιέχουν αναθέσεις στη μεταβλητή V, στο αρχικό πρόγραμμα, τότε μία φ-συνάρτηση, της μορφής $V \leftarrow \varphi(V, \dots, V)$, θα εισαχθεί στον κόμβο Z, στο καινούριο πρόγραμμα.
II. Κάθε αναφορά της μεταβλητής V στο αρχικό πρόγραμμα ή σε μια εισαχθείσα φ-συνάρτηση θα αντικατασταθεί με μία αναφορά σε νέα μεταβλητή V_i,

αφήνοντας το πρόγραμμα σε SSA δομή, αφού για κάθε μεταβλητή υπάρχει μία ανάθεση.

III. Κατά μήκος ενός κατευθυνόμενου μονοπατιού, μία χρήση της μεταβλητής V, στο αρχικό πρόγραμμα, και της αντίστοιχης V_i, στο νέο πρόγραμμα, θα έχουν την ίδια τιμή.

Για κάθε μεταβλητή V, οι κόμβοι, στο αρχικό πρόγραμμα, για τους οποίους χρειάζεται να εισάγουμε μία φ-συνάρτηση μπορούν να οριστούν αναδρομικά από τον όρο (1) στον παραπάνω ορισμό της SSA δομής. Ένας κόμβος Z χρειάζεται μία φ-συνάρτηση για κάθε μεταβλητή V εάν ο Z είναι το σημείο σύγκλισης για δύο μονοπάτια τα οποία ξεκινούν από δύο διαφορετικούς κόμβους στους οποίους είτε υπάρχει ανάθεση στο V είτε υπάρχει ανάγκη εισαγωγής μίας φ-συνάρτησης για το V. Μη αναδρομικά, μπορούμε να θεωρήσουμε ότι ένας κόμβος Z χρειάζεται μία φ-συνάρτηση για το V επειδή το Z είναι σημείο σύγκλισης δύο μη μηδενικών μονοπατιών $X \overset{+}{\rightarrow} Z$ και $Y \overset{+}{\rightarrow} Z$ τα οποία ξεκινούν από τα X και Y αντίστοιχα και ήδη περιέχουν αναθέσεις στη V. Εάν το Z δεν περιέχει μία ανάθεση στη V, τότε η φ-συνάρτηση που εισάγεται στο Z θα το προσθέσει στο σύνολο των κόμβων οι οποίοι περιέχουν ανάθεση στη V. Έτσι, με περισσότερους κόμβους να εξετάζονται σαν πιθανοί κόμβοι αρχής του μονοπατιού, θα πρέπει να εξετάσουμε περισσότερους κόμβους οι οποίοι θα εμφανίζονται σαν σημείο σύγκλισης μη μηδενικών μονοπατιών, τα οποία μη μηδενικά μονοπάτια ξεκινούν σε κόμβους με αναθέσεις στη V. Το σύνολο των κόμβων το οποίο χρειάζεται μία φ-συνάρτηση μπορεί να βρεθεί με επαναληπτικό τρόπο.

Αν το αρχικό πρόγραμμα υπολογίζει εκφράσεις οι οποίες χρησιμοποιούν σταθερές και βαθμωτές μεταβλητές για να κάνουν ανάθεση τιμών, τότε θα ήταν πιο απλό να παράγουμε έναν κώδικα ο οποίος είναι απευθείας σε δομή SSA. Ωστόσο, τα προγράμματα χρησιμοποιούν συχνά και άλλες δομές. Κάποιες μεταβλητές δεν είναι βαθμωτές και κάποιοι υπολογισμοί δεν αναφέρουν ρητά πια μεταβλητή χρησιμοποιούν ή αλλάζουν. Μερικά τέτοια παραδείγματα αποτελούν οι πίνακες, οι δομές και οι έμμεσες αναφορές σε μεταβλητές.

4.2.1 Κατασκευή Πίνακα στην SSA δομή

Στην παράγραφο 2.5.5 περιγράψαμε πως μπορεί να χειριστεί η ενδιάμεση γλώσσα τις προσπελάσεις και τις αναθέσεις σε ένα πίνακα. Στη συνέχεια θα περιγράψουμε πως η δομή ενός πίνακα μπορεί να μεταφραστεί σε SSA δομή. Θα είναι ικανοποιητικό να περιγράψουμε την μετάφραση σε ένα μονοδιάστατο πίνακα, αφού σε πίνακες μεγαλύτερων διαστάσεων η βασική ιδέα παραμένει η ίδια και το μόνο που χρειάζεται είναι επιπλέον σημειογραφία.

Εάν A και B είναι δύο μεταβλητές πινάκων, τότε δηλώσεις ανάθεσης όπως $A \leftarrow B$ ή $A \leftarrow 0$ μπορούν να χειρίζονται σαν αναθέσεις σε βαθμωτή μεταβλητή. Πολλές αναφορές της μεταβλητής A στο αρχικό πρόγραμμα μπορεί να είναι αναφορές του τύπου $A(i)$ για κάποιο δείκτη i, ο οποίος είναι μια μεταβλητή ακέραιου. Το να χειριστούμε το $A(i)$ σαν μεταβλητή θα είναι ανώφελο, επειδή μία ανάθεση στη μεταβλητή $A(i)$ μπορεί να αλλάξει αλλά μπορεί και να μην αλλάξει την τιμή της $A(j)$ και επειδή η τιμή στο $A(i)$ μπορεί να αλλάξει με μία ανάθεση στο i αντί για μία ανάθεση στο $A(i)$. Έτσι, ολόκληρος ο πίνακας μπορεί να μεταχειριστεί σαν μία απλή βαθμωτή μεταβλητή, στην οποία, όπως αναφέραμε και σε προηγούμενο κεφάλαιο, εφαρμόζονται μόνο τελεστές προσπέλασης (access) ή τελεστές ενημέρωσης και ανάθεσης (update). Αυτοί οι τελεστές είναι της μορφής $Access(A, i)$, η οποία αποτιμά το $i - οστό$ στοιχείο του A, και της μορφής $Update(A, j, V)$,το οποίο έχει ως αποτέλεσμα να δημιουργήσει ένα πίνακα με ίδιο μέγεθος και με ίδιες τιμές με τον A εκτός μόνο από το $j - οστό$ στοιχείο το οποίο έχει τιμή V. Η ανάθεση της βαθμωτής τιμής V στο $A(j)$ έχει το ίδιο ακριβός αποτέλεσμα με την ανάθεση ενός ολόκληρου πίνακα στον πίνακα A, όπου οι τιμές του καινούριου πίνακα εξαρτώνται από τις τιμές του παλιού πίνακα όπως επίσης από το δείκτη j και τη νέα μεταβλητή V. Η μετάφραση στην SSA δομή είναι αδιάφορη με το αν οι τιμές των μεταβλητών είναι μεγάλα αντικείμενα ή με το τι σημαίνουν οι τελεστές.

4.2.2 Κατασκευή Δομών στην SSA δομή

Οι δομές γενικά μπορούν να θεωρηθούν πίνακες, στις οποίες οι αναφορές στα πεδία της μπορούν να μεταχειριστούν όπως τα πεδία του πίνακα. Έτσι, μία ανάθεση στη δομή χρησιμοποιεί έναν τελεστή $Update$ (ενημέρωσης), και η αποτίμηση ενός πεδίου

της δομής χρησιμοποιεί ένα τελεστή $Access$ (προσπέλαση). Στην επικρατέστερη περίπτωση όπου γίνονται αναφορές σε απλά πεδία, αυτή η μεταχείριση έχει ως αποτέλεσμα πίνακες των οποίων τα στοιχεία επιλέγονται με ένα δείκτη του οποίου οι τιμές είναι ακέραιες. Η ανάλυση εξάρτησης μπορεί να καθορίσει την ανυπαρξία εξαρτήσεων κατά μήκος τέτοιων προσπελάσεων, έτσι ώστε κατά την βελτιστοποίηση να μπορεί να μεταφερθεί μία ανάθεση σε ένα πεδίο μακριά από την ανάθεση ενός άλλου πεδίου. Εάν από την ανάλυση καταλήξουμε στο συμπέρασμα ότι στη δομή, η οποία αποτελείται από n στοιχεία, όλες οι προσπελάσεις στα πεδία της γίνονται ανεξάρτητα, τότε μπορούμε να τη διαχωριστεί σε n διακριτές μεταβλητές. Τα στοιχεία μίας τέτοιας δομής ενοποιούνται στο αρχικό πρόγραμμα μόνο για λόγους οργάνωσης και στην SSA δομή γίνεται διαχωρισμός έτσι ώστε να βρίσκονται σε μία κατάλληλη δομή για επικείμενες βελτιστοποιήσεις.

4.2.3 Έμμεσες Αναφορές σε Μεταβλητές

Για να κατασκευάσουμε την SSA δομή απαιτείται να ξέρουμε ποιες μεταβλητές τροποποιούνται και ποιες χρησιμοποιούνται από μία δήλωση. Όμως ένα πρόγραμμα δεν περιέχει μόνο ρητές αναφορές σε μεταβλητές. Υπάρχουν περιπτώσεις στις οποίες μία δήλωση μπορεί να τροποποιεί ή να χρησιμοποιεί μία μεταβλητή για την οποία δεν υπάρχει ρητή αναφορά στη δήλωση. Παράδειγμα τέτοιων έμμεσων αναφορών αποτελούν οι αναφορές σε καθολικές μεταβλητές στο πρόγραμμα οι οποίες τροποποιούνται ή χρησιμοποιούνται μέσω ενός καλέσματος διαδικασίας, οι αντιστοιχίες μεταβλητών (aliased variables), δηλαδή μεταβλητές με διαφορετικό όνομα που δείχνουν στην ίδια τιμή, και επαν-αναφορά σε μία μεταβλητή δείκτη. Για να επιτύχουμε την SSA δομή, πρέπει να χειριστούμε τις έμμεσες αναφορές όπως τις ρητές, είτε με προσεγγίσεις είτε με ανάλυση. Η αποθήκευση σε σωρό μπορεί να μοντελοποιηθεί με την αναπαράσταση ολόκληρης της σωρού σαν μία μοναδική μεταβλητή η οποία μπορεί να τροποποιηθεί ή να χρησιμοποιηθεί από κάθε δήλωση, η οποία μπορεί να τροποποιήσει τη σωρό. Πιο προσεγμένα και πολύπλοκα μοντέλα είναι πιθανόν να χρησιμοποιηθούν, αλλά η προσεγγιστική τεχνική είναι ήδη αρκετή για βελτιστοποίηση κώδικα, η οποία δεν περιλαμβάνει ολόκληρο το σωρό αλλά διάσπαρτο κώδικα ο οποίος εξαρτάται από τη σωρό.

Για κάθε αναφορά S, προκαλούνται τρία είδη επιδράσεων στην SSA δομή από την μετάφραση της αναφοράς. Έτσι, όλες οι αναφορές εισάγονται στα ακόλουθα σύνολα:

I. $MustMod(S)$ είναι το σύνολο των μεταβλητών οι οποίες πρέπει να τροποποιηθούν από την εκτέλεση της αναφοράς S.
II. $MayMod(S)$ είναι το σύνολο των μεταβλητών οι οποίες μπορεί να τροποποιηθούν από την εκτέλεση της αναφοράς S.
III. $MayUse(S)$ είναι το σύνολο των μεταβλητών των οποίων οι τιμές πριν την εκτέλεση της αναφοράς S μπορεί να χρησιμοποιηθούν από την αναφορά S.

Αντικαθιστούμε κάθε έμμεση αναφορά μίας δήλωσης S με την μεταλλαγμένη δήλωση ανάθεσης A, όπου όλες οι μεταβλητές στο σύνολο $MayMode(S)$ εμφανίζονται στο $LHS(A)$ και όλες οι μεταβλητές στο σύνολο $MayUse(S) \cup (MayMod(S) - MustMod(S))$ εμφανίζονται στο $RHS(A)$.

Ένας μεταγλωττιστής που κάνει βελτιστοποίηση μπορεί να έχει αλλά μπορεί και να μην έχει πρόσβαση, έμμεση ή άμεση, στο σώμα της διαδικασίας κλήσης ή στα αποτελέσματα των επιδράσεων. Εάν καμία συγκεκριμένη πληροφορία δεν είναι διαθέσιμη, είθισται να κάνουμε προσεγγιστικές μοντελοποιήσεις. Μία κλήση μπορεί να αλλάξει κάποια καθολική μεταβλητή ή κάποια παράμετρο την οποία προσπερνά η αναφορά, αλλά είναι λογικό να θεωρήσουμε ότι οι μεταβλητές τοπικά στο σημείο κλήσης δεν θα αλλάξουν. Τέτοιες υποθέσεις περιορίζουν την έκταση των αλλαγών που μπορεί να εκτελεστούν. Υπάρχουν άλλες τεχνικές οι οποίες μπορούν να εξάγουν περισσότερες πληροφορίες.

Όταν χρησιμοποιείται μία εξελιγμένη τεχνική ανάλυσης, τα συνήθη αποτελέσματα είναι ότι υπάρχουν λίγες παράπλευρες δυσλειτουργίες και οι πλειάδες, LHS και RHS, είναι μικρές. Οι μικρές πλειάδες μπορούν να αναπαρασταθούν απευθείας. Εξελιγμένες τεχνικές ανάλυσης, ωστόσο, συχνά δεν είναι διαθέσιμες. Πολύ μεταγλωττιστές δεν κάνουν καθόλου ανάλυση στις διαδικασίες. Θεωρείστε μία κλήση σε εξωτερική διαδικασία η οποία δεν έχει αναλυθεί. Οι πλειάδες της κλήσης πρέπει να περιέχουν όλες τις καθολικές μεταβλητές. Ωστόσο, η αναπαράσταση των πλειάδων μπορεί να είναι πάλι συμπαγής. Αυτό επιτυγχάνεται με την χρήση μίας δομής, για αναπαράσταση, η οποία θα περιέχει μία σημαία, η οποία θα δείχνει ότι όλες οι καθολικές μεταβλητές χρησιμοποιούνται από την πλειάδα, και μερικές τοπικές

μεταβλητές οι οποίες είναι στην πλειάδα λόγο της μετάδοσης παραμέτρων. Η διαισθητική εξήγηση και η θεωρητική ανάλυση των τεχνικών βελτιστοποίησης, με ή χωρίς τη δομή SSA, είναι βολικά διατυπωμένη με ρητές πλειάδες. Συμπαγείς αναπαραστάσεις μπορούν επίσης να χρησιμοποιηθούν στην υλοποίηση.

4.3 Η Ελάχιστης SSA δομής

Η ελάχιστη SSA δομή είναι εκείνη η οποία περιέχει το μικρότερο δυνατό αριθμό φ-συναρτήσεων με την επιφύλαξη να ικανοποιείται ο όρος (1) του ορισμό της SSA δομής στην ενότητα 4.2. Οι βελτιστοποιήσεις, οι οποίες εξαρτώνται στην SSA δομή, είναι ακόμα έγκυρες αν σε αυτές εισαχθούν κάποιες επιπλέον φ-συναρτήσεις, πέρα από αυτές που εισάγονται στην ελάχιστή της μορφή. Ωστόσο, οι επιπλέον φ-συναρτήσεις μπορεί να προκαλέσουν απώλεια πληροφοριών και, επιπλέον, πάντα προκαλούν επιβάρυνση στην διαδικασία της βελτιστοποίησης. Έτσι, είναι σημαντικό να τοποθετούμε τις φ-συναρτήσεις μόνο εκεί που απαιτούνται. Μία παραλλαγή της SSA δομής μπορεί να παραλείψει επίτηδες την τοποθέτηση μίας φ-συνάρτησης σε ένα σημείο σύγκλισης Z για τη μεταβλητή V, από την στιγμή που δεν υπάρχουν χρήσεις της V μετά το Z. Τότε η φ-συνάρτηση μπορεί να παραληφθεί χωρίς να υπάρχει φόβος να μην ικανοποιείται ο όρος (3) στον ορισμό της SSA δομής. Αυτή η παραλλαγή λέγεται κλαδεμένη (pruned) SSA δομή και είναι μερικές φορές προτιμότερη από την παραλλαγή η οποία τοποθετεί φ-συναρτήσεις, για τις μεταβλητές, σε κάθε σημείο σύγκλισης.

Με την πρώτη ματιά, η προσεκτική τοποθέτηση φ-συναρτήσεων προϋποθέτει την τοποθέτηση μίας για κάθε ζευγάρι δηλώσεων ανάθεσης για κάθε μεταβλητή. Ο έλεγχος του πότε υπάρχουν δύο αναθέσεις σε μία μεταβλητή V οι οποίες φτάνουν σε ένα κοινό σημείο φαίνεται να αποτελεί μία μη γραμμική διαδικασία. Στην πραγματικότητα, ωστόσο, είναι αρκετό να κοιτάξουμε τα όρια κυριαρχίας (dominance frontiers) κάθε κόμβου στο γράφημα ροής.

Ξεκινώντας με τον μη αναδρομικό χαρακτηρισμό που θα τοποθετηθούν οι φ-συναρτήσεις έχουμε την ακόλουθη διαδικασία:

Δοθέντος ενός συνόλου N από κόμβους του γραφήματος ροής, το σύνολο $J(N)$ των κόμβων ένωσης ορίζεται να είναι το σύνολο όλων εκείνων των κόμβων Z για τους οποίους υπάρχουν δύο μη κενά μονοπάτια τα οποία ξεκινούν από δύο ξεχωριστούς

κόμβους στο N και συγκλίνουν στο Z. Ο επαναληπτικός υπολογισμός των κόμβων ένωσης αποτελεί το σύνολο $J^{+}(N)$ και είναι το όριο της αυξητικής ακολουθίας του συνόλου των κόμβων:

$$J_1 = J(N)$$

$$J_{i+1} = J(N \cup J_i)$$

Στην πράξη, αν το N τυγχάνει να είναι το σύνολο των κόμβων στους οποίος γίνεται ανάθεση σε μία μεταβλητή V, τότε το $J^{+}(N)$ αποτελεί το σύνολο των κόμβων οι οποίοι χρειάζονται μία φ-συνάρτηση για τη μεταβλητή V. Ουσιαστικά οι τελεστές $J(N)$ και $J^{+}(N)$ αντιστοιχούν ένα σύνολο από κόμβους στο σύνολο των κόμβων. Μπορούμε να επεκτείνουμε την διαδικασία εύρεσης των ορίων κυριαρχίας από ένα κόμβο σε ένα σύνολο από κόμβους με τον ακόλουθο μετασχηματισμό:

$$DF(N) = \bigcup_{x \in N} DF(x)$$

Η επαναληπτική μέθοδος υπολογισμού των ορίων κυριαρχίας $DF^{+}(N)$, όπως συμβαίνει και με το $J^{+}(N)$, αποτελεί το όριο της αυξητικής ακολουθίας του συνόλου των κόμβων:

$$DF_1 = DF(N)$$

$$DF_{i+1} = DF(N \cup DF_i)$$

Οι παραπάνω περιγραφές είναι βολικές για την συσχέτιση της επαναληπτικής διαδικασίας υπολογισμού των ορίων κυριαρχίας και της επαναληπτικής διαδικασίας υπολογισμού των κόμβων ένωσης. Αν το σύνολο N αποτελεί το σύνολο των κόμβων με αναθέσεις σε μία μεταβλητή V, τότε αποδεικνύεται ότι:

$$J^{+}(N) = DF^{+}(N)$$

το οποίο ισχύει αν ο κόμβος Εισόδου (Entry) ανήκει στο σύνολο N. Αν ο κόμβος Εισόδου δεν ανήκει στο σύνολο N τότε ο υπολογισμός των ορίων κυριαρχίας δεν είναι δυνατός. Μία αναλυτική απόδειξη της παραπάνω εξίσωσης παρουσιάζεται στην εργασία [12] στην οποία με μία σειρά από λήμματα, τα οποία με τη σειρά τους αποδεικνύονται, καταλήγουμε στο συμπέρασμα:

$$\left.\begin{matrix} J^{+}(N) \subseteq DF^{+}(N) \\ DF^{+}(N) \subseteq J^{+}(N) \end{matrix}\right\} \leftrightarrow J^{+}(N) = DF^{+}(N)$$

Ένα παράδειγμα θα μας βοηθήσει να καταλάβουμε διαισθητικά την παραπάνω ισότητα. Υποθέστε ότι μία μεταβλητή V έχει μόνο μία ανάθεση στο αρχικό πρόγραμμα, έτσι κάθε χρήση της μεταβλητής V θα είναι είτε χρήση της τιμής V_0, από την αρχική ανάθεση του κόμβου Εισόδου στο πρόγραμμα, είτε χρήση της τιμής V_1, από την μοναδική εντολή ανάθεση στη μεταβλητή V στο πρόγραμμα. Αν $B1$ είναι το βασικό μπλοκ κώδικα στο οποίο γίνεται η ανάθεση στη μεταβλητή V, τότε το $B1$ θα καθορίζει την τιμή της V όταν η ροή ελέγχου περνάει μόνο από μία ακμή $(B1 \rightarrow B2)$ στο βασικό μπλοκ $B2$. Έτσι, ο κώδικας στο $B1$ θα βλέπει μόνο την τιμή V_1 και θα είναι ανεπηρέαστος από την τιμή V_0. Εάν το $B2 \neq B1$, αλλά όλα τα μονοπάτια από τον κόμβο Είσοδος προς το $B2$ συναντάνε πάντα το $B1$, δηλαδή $B1 \in stdom(B2)$, τότε ο κώδικας στο βασικό μπλοκ $B2$ θα βλέπει πάντα την τιμή V_1, άσχετα από το πόσο μακριά από το μπλοκ $B1$ βρίσκεται. Ωστόσο, ο έλεγχος μπορεί να περάσει σε ένα βασικό μπλοκ $B3$ το οποίο δεν βρίσκεται υπό την κυριαρχία του βασικού μπλοκ B1, δηλαδή $B1 \notin stdom(B3)$. Υποθέστε ότι το $B3$ είναι το πρώτο, τέτοιου είδους, βασικό μπλοκ που συναντούμε στο μονοπάτι. Έτσι, ο κώδικας του $B3$ θα βλέπει την τιμή V_1 από μία εισερχόμενη ακμή αλλά θα βλέπει και την τιμή V_0 από μία άλλη εισερχόμενη ακμή. Τότε λέμε ότι το $B3$ ανήκει στα όρια κυριαρχίας του κόμβου $B1$ και είναι προφανές ότι χρειάζεται να εισάγουμε μία φ-συνάρτηση για την μεταβλητή V στο βασικό μπλοκ $B3$. Γενικότερα, χωρίς να έχει σημασία πόσες εντολές ανάθεσης στη μεταβλητή V εμφανίζονται στο αρχικό πρόγραμμα και χωρίς να έχει σημασία το πόσο πολύπλοκο είναι το γράφημα ροής, μπορούμε να τοποθετήσουμε μία φ-συνάρτηση για μία μεταβλητή V βρίσκοντας τα όρια κυριαρχίας κάθε κόμβου που κάνει ανάθεση στη V, στη συνέχεια τα όρια κυριαρχίας κάθε κόμβου ο οποίος ήδη περιέχει μία φ-συνάρτηση για τη V κοκ. Η ίδια ιδέα των κόμβων κυριαρχίας χρησιμοποιείται για τον προσδιορισμό των εξαρτήσεων ελέγχου, οι οποίες μας δείχνουν τους όρους που επηρεάζουν την εκτέλεση μίας δήλωσης. Άτυπα, μία δήλωση είναι εξαρτημένη από τον έλεγχο μίας διακλάδωσης αν η μία ακμή της διακλάδωσης προκαλεί την εκτέλεση της δήλωσης, ενώ η άλλη ακμή παραλείπει να εκτελέσει την δήλωση.

4.4 Κατασκευή της Ελάχιστης SSA δομή

Παρακάτω περιγράφουμε την διαδικασία την οποία ακολουθεί ο αλγόριθμος ο οποίος εισάγει τις φ-συναρτήσεις στην ελάχιστη SSA δομή:

1) Υπολογίζουμε τα όρια κυριαρχίας για κάθε βασικό μπλοκ.

2) Για κάθε μεταβλητή, δημιουργούμε μία λίστα με βασικά μπλοκ στα οποία γίνεται μία ανάθεση σε αυτή την μεταβλητή (αυτή η εντολή δημιουργεί την λίστα εργασίας).
3) Για κάθε μεταβλητή n

Για κάθε βασικό μπλοκ B στο οποίο γίνεται μία ανάθεση στη n

Για κάθε βασικό μπλοκ B' το οποίο ανήκει στα όρια κυριαρχίας του B, δηλαδή $B' \in DF(B)$

1. Εισάγουμε μία φ-συνάρτηση στο B' για τη n.
2. Εισάγουμε το B' στη λίστα με τα βασικά μπλοκ στα οποία γίνεται ανάθεση στη n (αυτή η εντολή ενημερώνει την λίστα εργασίας).

Για να γίνει πιο αποδοτικός ο αλγόριθμος θα πρέπει να κρατήσουμε δύο λίστες με ελέγχους. Στην πρώτη λίστα θα αποφεύγουμε να εισάγουμε ένα βασικό μπλοκ δεύτερη φορά στη λίστα εργασίας και στη δεύτερη λίστα θα αποφεύγουμε να εισάγουμε την ίδια φ-συνάρτηση δεύτερη φορά.

Έστω ότι σε κάθε βασικό μπλοκ B το σύνολο $A_{orig}(B)$ περιέχει τις αρχικές δηλώσεις ανάθεσης στις μεταβλητές, όπου κάθε δήλωση ανάθεσης είναι της μορφής $LHS \leftarrow RHS$ και συμβάλει με το μέγεθος της LHS πλειάδας του στο σύνολο $A_{orig}(B)$. Η καταμέτρηση των δηλώσεων ανάθεσης σε μεταβλητές είναι ένα από τα πολλά μέτρα για την εύρεση του μεγέθους του προγράμματος. Με αυτό το μέτρο, το μέγεθος του προγράμματος μεγαλώνει από $A_{orig} = \sum_B A_{orig}(B)$ σε $A_{tot} = \sum_B A_{tot}(B)$, όπου κάθε φ-συνάρτηση στο βασικό μπλοκ B συμβάλλει με 1 στο σύνολο $A_{tot}(B)$. Έτσι, όπως είναι προφανές έχουμε ότι $A_{tot}(B) = A_{orig}(B) + A_{\varphi}(B)$, όπου το $A_{\varphi}(B)$ αποτελεί το σύνολο με τις αναθέσεις φ-συναρτήσεων στο βασικό μπλοκ B. Υπάρχει ένα παρόμοιο μέτρο για τον αριθμό των αναφορών στις μεταβλητές, των οποίων το μέγεθος από $M_{orig} = \sum_B M_{orig}(B)$ γίνεται $M_{tot} = \sum_B M_{tot}(B)$, όπου κάθε φ-συνάρτηση που τοποθετείται στο βασικό μπλοκ B συμβάλλει με 1 στο $M_{tot}(B)$ και ως αποτέλεσμα έχουμε $M_{tot}(B) = M_{orig}(B) + M_{\varphi}(B)$. Ο υπολογισμός των ορίων κυριαρχίας για όλα τα βασικά μπλοκ έχουν πολυπλοκότητα στον χρόνο $O(E + \sum_B |DF(B)|)$, όπου το E είναι ο αριθμός των ακμών στο δέντρο κυριαρχίας του γραφήματος ροής. Η τοποθέτηση φ-συναρτήσεων σε κάθε βασικό μπλοκ B' έχει κόστος γραμμικό στο βαθμό εισόδου του B', οπότε υπάρχει μία επιβάρυνση $O(\sum_{B'} M_{\varphi}(B'))$ στον χρόνο εκτέλεσης. Η αντικατάσταση όλων των αναφορών, στις

μεταβλητές, έχει συνεισφορά τουλάχιστον $O(M_{tot})$ στον χρόνο εκτέλεσης κάθε αλγορίθμου μετάφρασης κώδικα σε SSA δομή, οπότε ο χρόνος για την αντικατάσταση των αναφορών, μετά την τοποθέτηση των φ-συναρτήσεων, μπορεί να αγνοηθεί όταν αναλύουμε την συνολική πολυπλοκότητα στο χρόνο ολόκληρης της διαδικασίας. Για τον ίδιο λόγο η συμβολή $O(N)$, για την αρχικοποίηση κάθε βασικού μπλοκ, μπορεί να αγνοηθεί επειδή συνυπολογίζεται με το κόστος του υπολογισμού των ορίων κυριαρχίας. Εκείνο το οποίο δεν μπορεί να αγνοηθεί είναι το κόστος για τη διαχείριση της λίστας εργασίας. Στο τρίτο βήμα του αλγορίθμου η δήλωση «για κάθε μπλοκ B στο οποίο γίνεται μία ανάθεση στη n» εκτελείται $A_{tot}(B)$ φορές, και κάθε εκτέλεση προκαλεί κόστος γραμμικό στο $|DF(X)|$, επειδή όλα τα βασικά μπλοκ $B' \in DF(X)$ επεξεργάζονται. Οπότε η συνολική πολυπλοκότητα στον χρόνο του παραπάνω αλγορίθμου είναι $O(\sum_B (A_{tot}(B) \times |DF(B)|) + \sum_B |DF(B)| + E)$. Μετρώντας την πολυπλοκότητα με βάση την έξοδο, με την ίδια μεθοδολογία που χρησιμοποιήσαμε στον υπολογισμό του $A_{tot}(B)$, καταλήγουμε στο $O\big((A_{tot} \times avrgDF) + \sum_B |DF(B)| + E\big)$, όπου το $avrgDF$ αποτελεί τον μέσο όρο και ορίζεται ως:

$$avrgDF \stackrel{\text{def}}{=} \frac{\sum_B (A_{tot}(B) \times |DF(B)|)}{\sum_B A_{tot}(B)}$$

Στο δοκίμιο [12] υπάρχουν πειραματικές μελέτες η οποίες δείχνουν ότι τα όρια κυριαρχίας στην πράξη είναι μικρά σύνολα και άρα η πολυπλοκότητα στο χώρο γίνεται περίπου $O(A_{tot})$, το οποίο με τη σειρά του γίνεται $O(A_{orig})$.

Τα παρακάτω βήματα, είναι τα βήματα τα οποία ακολουθεί ο αλγόριθμος για να μετονομάσει όλες τις αναφορές στις μεταβλητές:

1) Δημιουργούμε δύο πίνακες με στοίβες. Ο πρώτος πίνακας, $S(\bullet)$, περιλαμβάνει μία στοίβα για κάθε μεταβλητή. Η στοίβα κρατάει έναν ακέραιο. Ο ακέραιος i ο οποίος βρίσκεται στην κορυφή της στοίβας $S(V)$ χρησιμοποιείται για να κατασκευαστεί η μεταβλητή V_i, η οποία πρέπει να αντικαταστήσει την μεταβλητή V. Ο δεύτερος πίνακας, $C(\bullet)$, είναι ένας πίνακας με ακέραιους, ένας για κάθε μεταβλητή. Κάθε στοιχείο του πίνακα αποτελεί έναν μετρητή. Η τιμή του $C(V)$ μας λέει πόσες αναθέσεις στη μεταβλητή V έχουν επεξεργαστεί. Οι δύο πίνακες αρχικοποιούνται για κάθε μεταβλητή V ως $C(V) \leftarrow 0$ και $S(V) \leftarrow Empty$.
2) Ξεκινούμε μία διαδρομή, από πάνω προς τα κάτω, ανάμεσα στα βασικά μπλοκ στο δέντρο κυριαρχίας του γραφήματος ροής του προγράμματος.

i. Για κάθε δήλωση A στο μπλοκ B, εάν το A είναι μία συνηθισμένη ανάθεση τότε
    i. Για κάθε μεταβλητή V η οποία χρησιμοποιείται στο $RHS(A)$
        - Γίνεται αντικατάσταση της μεταβλητής με την μεταβλητή V_i, όπου το i είναι το πρώτο στοιχείο της στοίβας $S(V)$.
    ii. Για κάθε μεταβλητή V η οποία χρησιμοποιείται στο $LHS(A)$
        - Το i γίνεται ίσο με τον μετρητή $C(V)$.
        - Γίνεται αντικατάσταση της μεταβλητής V με μία νέα μεταβλητή V_i στο $LHS(A)$.
        - Τοποθετούμε το i στην αρχή της στοίβας $S(V)$.
        - Ο μετρητής στο $C(V)$ αυξάνει κατά 1.

ii. Για κάθε βασικό μπλοκ B' το οποίο ανήκει στους διαδόχους του B, δηλαδή $B' \in Succ(B)$
    i. Σε μία σταθερά j αποθηκεύεται ο αριθμό ο οποίος δείχνει ποιος προκάτοχος του B' είναι ο B.
    ii. Για κάθε φ-συνάρτηση F στο μπλοκ B'
        - Ο V_i, όπου το i αποτελεί το πρώτο στοιχείο της στοίβας $S(V)$, αντικαθιστά τη μεταβλητή στην οποία γίνεται η ανάθεση από την φ-συνάρτηση.
        - Η τιμή του V_i γίνεται ίση με την τιμή V_j. (Εδώ ουσιαστικά γίνεται η εκτέλεση της φ-συνάρτησης)

iii. Εκτελούμε αναδρομικά το (2) βήμα του αλγορίθμου για τα παιδιά του μπλοκ B στο δέντρο κυριαρχίας.

iv. Για κάθε δήλωση A στο βασικό μπλοκ B
    i. Για κάθε παλιά μεταβλητή V, η οποία αντικαταστάθηκε από μία μεταβλητή V_i
        - Αφαιρούμε το πρώτο στοιχείο από την στοίβα $S(V)$.

Στην εργασία [12] υπάρχει μία αναλυτική απόδειξη ότι οι παραπάνω δύο αλγόριθμοι υπολογίζουν σωστά και κατασκευάζουν την ελάχιστη SSA δομή ενός προγράμματος. Ουσιαστικά στην απόδειξη, λαμβάνοντας υπόψη τις εντολές των δύο αλγορίθμων, επιχειρείται να αποδειχθεί ότι και οι τρεις όροι, στον ορισμό της SSA δομής, ικανοποιούνται. Ο δεύτερος αλγόριθμος έχει πολυπλοκότητα στο χρόνο $O(N + E + M_{tot})$, όπου N είναι ο αριθμός των βασικών μπλοκ και E ο αριθμός των ακμών στο δέντρο κυριαρχίας του γραφήματος ροής του προγράμματος, αλλά επειδή ο όρος

$(E + N)$ συνυπολογίζεται στον υπολογισμό των ορίων κυριαρχίας, γι' αυτό η πολυπλοκότητα του χρόνου γίνεται $O(M_{tot})$.

Οπότε, συμπερασματικά, μπορούμε να κατασκευάσουμε την ελάχιστη SSA δομή για κάθε πρόγραμμα με τους δύο παραπάνω αλγορίθμους. Η συνολική πολυπλοκότητα στον χρόνο, για την κατασκευή, θα είναι $O(E + \sum_B |DF(B)| + (A_{tot} \times avrgDF) + M_{tot})$.

4.5 Κατασκευή των Εξαρτήσεων Ελέγχου

Σε αυτό το υποκεφάλαιο θα ασχοληθούμε με τον υπολογισμό των εξαρτήσεων ελέγχου (control dependences) και θα καταλήξουμε στο συμπέρασμα ότι τα σύνολα των εξαρτήσεων ελέγχου είναι ουσιαστικά τα όρια κυριαρχίας κάθε κόμβου στο αντίστροφο γράφημα ροής.

Έστω ότι τα $B1$ και $B2$ είναι δύο βασικά μπλοκ του γραφήματος ροής. Αν το $B2$ εμφανίζεται σε κάθε μονοπάτι από το $B1$ στον κόμβο Έξοδος (Exit), τότε θα λέμε ότι το $B2$ ανήκει στο σύνολο με τους κόμβους οι οποίοι εμφανίζουν σχέση αντίστροφης κυριαρχίας επί του $B1$ ή αλλιώς ότι $B2 \in postdominators(B1)$. Αν το $B2$ είναι αντίστροφος κυρίαρχος του $B1$ και επίσης $B2 \neq B1$, τότε το $B2$ είναι γνήσιος αντίστροφος κυρίαρχος του $B1$ ή $B2$ strictly postdominates $B1$. Ο άμεσος αντίστροφος κυρίαρχος του $B1$ είναι ο πιο κοντινός γνήσιος αντίστροφος κυρίαρχος του $B1$, στο μονοπάτι από το $B1$ προς τον κόμβο Έξοδος(Exit). Στο δέντρο της αντίστροφης κυριαρχίας, τα παιδιά του κόμβου $B2$ θα έχουν όλα άμεσο αντίστροφο κυρίαρχο τον κόμβο $B2$.

Ένας κόμβος $B1$, στο γράφημα ροής, ανήκει στο σύνολο των εξαρτήσεων ελέγχου κόμβου του $B2$ εάν ισχύουν οι δύο παρακάτω όροι:

I. Υπάρχει ένα μη μηδενικό μονοπάτι $p : B1 \xrightarrow{+} B2$, τέτοιο ώστε το $B2$ είναι αντίστροφος κυρίαρχος κάθε κόμβου μετά το $B1$ στο p.
II. Ο κόμβος $B2$ δεν είναι γνήσιος αντίστροφος κυρίαρχος του κόμβου $B1$.

Με άλλα λόγια, υπάρχει, στο γράφημα ροής μία ακμή από το $B1$ η οποία σίγουρα προκαλεί την εκτέλεση της $B2$ και υπάρχει επίσης ένα μονοπάτι από το $B1$ το οποίο αποφεύγει την εκτέλεση του $B2$. Συσχετίζουμε αυτήν την εξάρτηση ελέγχου από το $B1$ στο $B2$ μία ετικέτα στην ακμή της $B1$ η οποία προκαλεί την εκτέλεση της $B2$. Έτσι το $B2$ είναι αντίστροφος κυρίαρχος ενός διαδόχου του $B1$ αν και μόνο αν υπάρχει ένα μονοπάτι $p : B1 \overset{+}{\rightarrow} B2$ τέτοιο ώστε το $B2$ είναι αντίστροφος άμεσος κυρίαρχος κάθε κόμβου μετά το $B1$ στο μονοπάτι p. Αντίστροφα, δοθέντος ενός μονοπατιού p με αυτές τις ιδιότητες, αν ο B' είναι ο πρώτος κόμβος μετά το $B1$ στο p, τότε το B' είναι διάδοχος του $B1$ και το $B2$ είναι ο αντίστροφος κυρίαρχος του.

Το αντίστροφο γράφημα ροής έχει τους ίδιους κόμβους με το κανονικό γράφημα ροής και για κάθε ακμή $X \rightarrow Y$, στο αντίστροφο γράφημα ροής υπάρχει μία ακμή $Y \rightarrow X$. Οι ρόλοι των κόμβων Είσοδος (Entry) και Έξοδος (Exit) αντιστρέφεται, ενώ η σχέση της αντίστροφης κυριαρχίας στο γράφημα ροής γίνεται σχέση κυριαρχία στο αντίστροφο γράφημα ροής.

Το $B2$ ανήκει στο σύνολο των εξαρτήσεων ελέγχου του $B1$ αν και μόνο αν $B1 \in DF(B2)$ στο αντίστροφο γράφημα ροής. Η απόδειξη αυτής της πρότασης απορρέει εύκολα από την πρόταση ότι το $B2$ είναι αντίστροφος κυρίαρχος ενός διαδόχου του $B1$ αν και μόνο αν υπάρχει ένα μονοπάτι $p : B1 \overset{+}{\rightarrow} B2$ τέτοιο ώστε το $B2$ είναι αντίστροφος άμεσος κυρίαρχος κάθε κόμβου μετά το $B1$ στο μονοπάτι p. Αυτή η σχέση στο αντίστροφο γράφημα ροής είναι η σχέση του ορίου κυριαρχίας του $B2$.

Ο αλγόριθμος ο οποίος υπολογίζει τα σύνολα των εξαρτήσεων ελέγχου έχει τα ακόλουθα βήματα:

1) Κατασκευάζουμε το αντίστροφο γράφημα ροής.
2) Κατασκευάζουμε το δέντρο κυριαρχίας για το αντίστροφο γράφημα ροής.
3) Εφαρμόζουμε τον αλγόριθμο για την εύρεση των ορίων κυριαρχίας για κάθε κόμβο στο αντίστροφο γράφημα ροής.
4) Για κάθε κόμβο X του αντίστροφου γραφήματος ροής θέτουμε το σύνολο των εξαρτήσεων ελέγχου $CD(X) \leftarrow \emptyset$

    Για κάθε κόμβο Y που ανήκει στο $DF(X)$, για το αντίστροφο γράφημα ροής

- Θέτουμε το σύνολο των εξαρτήσεων ελέγχου ίσο με $CD(X) \leftarrow CD(X) \cup \{Y\}$.

Στον παραπάνω αλγόριθμο αφού χτίσουμε το δέντρο κυριαρχίας με έναν από τους ήδη γνωστούς αλγορίθμους σε χρόνο $O(DomTree)$, ανάλογα με τον αλγόριθμο που θα χρησιμοποιήσουμε, θα ξοδέψουμε χρόνο $O(size(reverseDF))$ για να βρούμε τα αντίστροφα όρια κυριαρχίας και για να τα αντιστρέψουμε την συνέχεια. Έτσι, η συνολική πολυπλοκότητα στο χρόνο για τον υπολογισμό των εξαρτήσεων έλεγχου είναι $O(DomTree + size(reverseDF))$.

4.6 Η Μετάφραση από την SSA δομή

Πολλές ισχυρές αναλύσεις και βελτιστοποιήσεις μπορούν να εφαρμοστούν στα προγράμματα τα οποία βρίσκονται σε μορφή SSA. Ωστόσο, μετά τις αλλαγές και τις βελτιστοποιήσεις ο κώδικας πρέπει να εκτελεστεί. Αν και οι φ-συνατήσεις έχουν ακριβή σημασία, ωστόσο, γενικά, δεν υπάρχουν μηχανές οι οποίες να τις αναπαριστούν. Αυτός είναι και ο λόγος για τον οποίο ξαναμεταφράζουμε το πρόγραμμα έξω από την SSA δομή και αντικαθιστούμε κάθε φ-συνάρτηση με κάποιες συνηθισμένες αναθέσεις. Η απλοϊκή μετάφραση μπορεί να έχει σαν αποτέλεσμα έναν μη αποδοτικό κώδικα, αλλά ένας αποδοτικός και ταυτόχρονα αποτελεσματικός κώδικας μπορεί να παραχθεί αν δύο χρήσιμες μέθοδοι βελτιστοποίησης εφαρμοστούν. Αυτές είναι η εξάλειψη του νεκρού κώδικα (dead code elimination) και καθορισμός της αποθήκευσης με χρωματισμό (storage allocation by coloring).

Με την πιο απλοϊκή μέθοδο, μία φ-συνάρτηση με k εισόδους στην είσοδο του βασικού μπλοκ B μπορεί να αντικατασταθεί με k συνηθισμένες αναθέσεις, μία στο τέλος κάθε προκατόχου του B στο γράφημα ροής. Αυτή η προσέγγιση είναι πάντα σωστή, ωστόσο προκαλεί την εκτέλεση μη ωφέλιμης εργασίας. Αν σε αυτή την απλοϊκή προσέγγιση προσθέσουμε μία τεχνική διαγραφής νεκρού κώδικα και μία τεχνική καθορισμού αποθήκευσης με χρωματισμό, τότε το αποτέλεσμα που θα έχουμε θα είναι αρκετά αποδοτικό.

4.6.1 Εξάλειψη Νεκρού Κώδικα

Το πρόγραμμα στην αρχική του μορφή μπορεί να περιέχει κομμάτια «νεκρού» κώδικα. Μερικά ενδιάμεσα βήματα στη μεταγλώττιση μπορούν επίσης να προκαλέσουν την εμφάνιση «νεκρού» κώδικα. Επιπλέον, την δημιουργία «νεκρού» κώδικα μπορεί να προκαλέσει η βελτιστοποίηση. Με τόσες διαφορετικές πηγές δημιουργίας «νεκρού» κώδικα, είναι λογικό η τεχνική εξάλειψης του να εφαρμόζεται προς το τέλος της διαδικασίας βελτιστοποίησης.

Η μετάφραση στην SSA δομή αποτελεί ένα από τα ενδιάμεσα βήματα το οποίο μπορεί να παράγει «νεκρό» κώδικα. Υποθέστε ότι γίνεται μία ανάθεση στη μεταβλητή V, στη συνέχεια υποθέστε ότι η μεταβλητή αυτή χρησιμοποιείται στη διακλάδωση μίας εντολής $if \ldots then \ldots else \ldots$, αλλά η τιμή της μεταβλητής δεν χρησιμοποιείται μετά τον κόμβο ένωσης. Οι συνηθισμένες αναθέσεις στη V είναι «ζωντανές», ωστόσο η επιπλέον ανάθεση από τη φ-συνάρτηση αποτελεί «νεκρό» κώδικα. Συχνά, τέτοιου είδους «νεκρές» φ-συναρτήσεις είναι χρήσιμες, για τον προσδιορισμό των ισοδυναμιών και στη συνέχεια για την εξάλειψη των πλεονασμών στην SSA δομή. Ένα τέτοιο παράδειγμα παρουσιάζεται στην εικόνα 4.2. Αν και συχνά αποφεύγεται να τοποθετούνται τέτοιου είδους φ-συναρτήσεις, ωστόσο οι «νεκρές» φ-συναρτήσεις αυξάνουν τις ευκαιρίες για επιπλέον βελτιστοποίηση.

Υπάρχουν πολλοί ορισμοί για το τι είναι «νεκρός» κώδικας. Αλλού ορίζεται ως ο κώδικας ο οποίος είναι απρόσιτος και αλλού ορίζεται ως κώδικας ο οποίος δεν έχει καμία επίδραση. Και στις δύο περιπτώσεις, μπορούμε να χρησιμοποιήσουμε τον ευρύτερο ορισμό, ο οποίος λέει ότι ο «νεκρός» κώδικας είναι εκείνο το τμήμα κώδικα το οποίο μπορεί να αφαιρεθεί με ασφάλεια. Για τον προσδιορισμό των τμημάτων κώδικα τα οποία αποτελούν «νεκρό» κώδικα, αρχικά σημειώνουμε ως «νεκρές» όλες της δηλώσεις. Ωστόσο, μερικές δηλώσεις σημειώνονται ως «ζωντανές» για να μπορέσει να ξεκινήσει η διαδικασία. Οι δηλώσεις που σημειώνονται ως ζωντανές είναι εκείνες οι οποίες ικανοποιούν έναν από τους όρους της παρακάτω λίστας. Έτσι, μαρκάροντας αυτές τις δηλώσεις ως «ζωντανές» μπορεί να προκληθεί η σημείωση και άλλων μεταβλητών ως ζωντανές. Όταν, πλέον, η διαδικασία σταματήσει, τότε οι δηλώσεις οι οποίες εξακολουθούν να είναι σημειωμένες ως «νεκρές» μπορούν με ασφάλεια να αφαιρεθούν από το πρόγραμμα.

```
if P1                        if P1
  then do                      then do
    Y1←1                         Y1←1
     use of Y1                    use of Y1
  end                          end
  else do                      else do
    Y2←X1                        Y2←X1
    use of Y2                    use of Y2
  end                          end
Y3←φ(Y1,Y2)                  Y3←φ(Y1,Y2)
         ...                          ...
if P1
  then Z1←1
  else Z2←X1
Z3←φ(Z1,Z2)
use of Z3                    use of Y3
```

Εικόνα 4.2: Αριστερά παρουσιάζεται ο κώδικας, σε SSA δομή, ο οποίος περιέχει μία νεκρή φ-συνάρτηση και δεξιά ο ίδιος κώδικας χωρίς αυτή.

Μία δήλωση σημειώνεται ως ζωντανή αν και μόνο αν ένας από τους παρακάτω όρους ικανοποιείται:

- Η δήλωση αποτελεί μία από τις δηλώσεις οι οποίες μπορεί να θεωρηθούν ότι έχουν επίδραση στην έξοδο του προγράμματος. Τέτοιες δηλώσεις είναι οι δηλώσεις I/O, οι δηλώσεις αναφοράς μίας παραμέτρου ή οι δηλώσεις κλήσης μίας ρουτίνας η οποία έχει παράπλευρα αποτελέσματα.
- Η δήλωση είναι μία δήλωση ανάθεσης και υπάρχουν ήδη δηλώσεις οι οποίες έχουν σημειωθεί ως «ζωντανές» οι οποίες χρησιμοποιούν μερικές από αυτές τις εξόδους.
- Η δήλωση είναι μία υπό όρους διακλάδωση και υπάρχουν ήδη δηλώσεις οι οποίες έχουν μαρκαριστεί ως ζωντανές και οι οποίες είναι εξαρτημένες από τον έλεγχο σε αυτή τη διακλάδωση.

Ο αλγόριθμος ο οποίος προσδιορίζει τον «νεκρό» κώδικα σε ένα πρόγραμμα εκτελεί τα ακόλουθα βήματα:

1) Κάθε δήλωση S σημειώνεται σαν «νεκρή», εκτός και αν ικανοποιεί τον (1) όρου του παραπάνω ορισμού. Η λίστα εργασίας, δηλαδή οι δηλώσεις οι οποίες πρόσφατα ανακαλύφθηκαν ότι είναι «ζωντανές», αρχικοποιείται με τις δηλώσεις S οι οποίες είναι ήδη σημειωμένες ως «ζωντανές».
2) Έως ότου να αδειάσει η λίστα εργασία αφαιρούμε μία-μία τις δηλώσεις S
   i. Για κάθε δήλωση D η οποία παρέχει μία τιμή που χρησιμοποιείται από τη δήλωση S, αν το D είναι σημειωμένο ως «νεκρό»
      - Σημειώνουμε την D ως «ζωντανή».
      - Προσθέτουμε την D στη λίστα εργασίας.
   ii. Για κάθε μπλοκ B το οποίο ανήκει στο σύνολο των μπλοκ τα οποία εξαρτώνται από τον έλεγχο του μπλοκ μέσα στο ο οποίο υπάρχει η δήλωση S, αν η τελευταία δήλωση L του μπλοκ B είναι σημειωμένη ως «νεκρή» τότε
      - Σημειώνουμε την L ως «ζωντανό».
      - Προσθέτουμε την L στη λίστα εργασίας.
3) Κάθε δήλωση S η οποία είναι σημειωμένη ως «νεκρή» αφαιρείται.

Αφού ανακαλυφθούν όλες οι «ζωντανές» δηλώσεις, εκείνες που παραμένουν σημειωμένες ως «νεκρές» αφαιρούνται. Άλλες βελτιστοποιήσεις μπορεί να έχουν ως αποτέλεσμα το άδειασμα, από δηλώσεις, του βασικού μπλοκ. Ένα άδειο βασικό μπλοκ μπορεί με ασφάλεια να αφαιρεθεί. Το γεγονός ότι όλες οι δηλώσεις θεωρούνται «νεκρές» μέχρι να σημειωθούν «ζωντανές» είναι κρίσιμο για τον (2) όρο. Οι δηλώσεις οι οποίες εξαρτώνται από τον εαυτό τους, δεν σημειώνονται ποτέ ως «ζωντανές» εκτός και αν χρειάζονται από μία άλλη «ζωντανή» μεταβλητή. Τέλος, ο όρος (3) ελέγχει τα μπλοκ τα οποία είναι εξαρτημένα στον έλεγχο από το μπλοκ το οποίο περιέχει την δήλωση που επεξεργαζόμαστε. Ένα βασικό μπλοκ του οποίου ο τελικός έλεγχος καταλήγει σε ένα μπλοκ με «ζωντανές» μεταβλητές έχει ως αποτέλεσμα την σημείωση της δήλωσης ως «ζωντανή».

4.6.2 Κατανομή Αποθήκευσης με Χρωματισμό

Με την πρώτη ματιά, φαίνεται ότι είναι σωστό να αντικαταστήσουμε όλες τις μεταβλητές V_i από την V και να διαγράψουμε όλες τις φ-συναρτήσεις. Ωστόσο, οι νέες μεταβλητές οι οποίες δημιουργήθηκαν από την μετάφραση στην SSA δομή δεν μπορούν πάντα να αφαιρεθούν, επειδή η βελτιστοποίηση μπορεί να παρενέβη στην ανεξαρτησία αποθήκευσης των νέων μεταβλητών. Αυτό το πρόβλημα που δημιουργείται μπορεί να περιγραφεί από το παράδειγμα της εικόνας 4.3. Το αρχικό πρόγραμμα κάνει δύο αναθέσεις στην μεταβλητή V και επίσης κάνει και δύο χρήσεις της μεταβλητής V. Μία βελτιστοποίηση του προγράμματος, στην SSA δομή, προκαλεί την μετακίνηση της αμετάβλητης ανάθεσης έξω από τον βρόχο, το οποίο έχει ως αποτέλεσμα ένα πρόγραμμα με ξεχωριστές μεταβλητές για ξεχωριστό σκοπό. Η ανάθεση στο V_3 αποτελεί «νεκρή» δήλωση και γι' αυτό θα αφαιρεθεί. Αυτές οι βελτιστοποιήσεις αφήνουν μία περιοχή στον κώδικά όπου τα V_1 και V_2 είναι ταυτόχρονα ζωντανά. Οπότε, και οι δύο μεταβλητές απαιτούνται και η αρχική μεταβλητή V δεν επιτρέπεται να τις αντικαταστήσει.

Ένας αλγόριθμος χρωματισμού γραφήματος (graph coloring algorithm) μπορεί να εφαρμοστεί για να μειώσουμε τον αριθμό των μεταβλητών που χρησιμοποιούνται και έτσι να μπορέσουμε να αφαιρέσουμε τις περισσότερες δηλώσεις ανάθεσης. Η επιλογή μίας τεχνικής χρωματισμού μπορεί να καθοδηγηθεί από μία ενδεχόμενη χρήση των αποτελεσμάτων της εξόδου. Αν έχουμε ως σκοπό να δημιουργήσουμε έναν ευανάγνωστο κώδικα, τότε είναι επιθυμητό να εξετάσουμε κάθε αρχική μεταβλητή V ξεχωριστά, χρωματίζοντας μόνο τις μεταβλητές οι οποίες παράγονται από το V. Εάν ο σκοπός μας είναι να παράγουμε έναν κώδικα μηχανής, τότε όλες οι μεταβλητές της SSA δομής μπορούν να εξεταστούν μαζί. Και στις δύο περιπτώσεις, η διαδικασία του χρωματισμού αλλάζει τις περισσότερες αναθέσεις οι οποίες εισήχθησαν για να αντικαταστήσουν τη φ-συνάρτηση σε ταυτοτικές αναθέσεις, δηλαδή, αναθέσεις της μορφής $V \leftarrow V$. Αυτές οι ταυτοτικές αναθέσεις μπορούν να αφαιρεθούν.

```
while (…) do                while (…) do                V₂←6
                              W₃←φ(W₀,W₂)              while (…) do
                              V₃←φ(V₀,V₂)                W₃←φ(W₀,W₂)
  read V                      read V₁                    V₃←φ(V₀,V₂)
  W←V + W                     W₁←V₁ + W₃                 read V₁
  V←6                         V₂←6                       W₁←V₁ + W₃
  W←V + W                     W₂←V₂ + W₁                 W₂←V₂ + W₁
end                         end                         end

        (a)                         (b)                         (c)
```

Εικόνα 4.3:a) Ο κώδικας στην αρχική μορφή b)ο κώδικας σε SSA μορφή c)ο κώδικας μετά από βελτιστοποίηση.

5 Βελτιστοποίηση Κώδικα

Στην ενότητα 2.5 εξετάσαμε την τοπική βελτιστοποίηση κώδικα, δηλαδή τη βελτιστοποίηση η οποία λαμβάνει χώρα εντός ενός βασικού μπλοκ. Σε αυτό το κεφάλαιο θα μελετήσουμε την καθολική βελτιστοποίηση κώδικα, στην οποία οι τεχνικές βελτιστοποίησης λαμβάνουν υπόψη τι συμβαίνει και έξω από τα βασικά μπλοκ. Το συγκεκριμένο κεφάλαιο βασίζεται στο βιβλίο των Aho, Lam, Sethi και Ullman [1].

5.1 Οι Πηγές της Βελτιστοποίησης

Οι τεχνικές οι οποίες προκαλούν βελτιστοποίηση στον κώδικα πρέπει να διατηρούν τη σημασιολογία του αρχικού κώδικα. Από την στιγμή που ο προγραμματιστής επιλέξει να υλοποιήσει έναν συγκεκριμένο αλγόριθμο, ο μεταγλωττιστής, εκτός από συγκεκριμένες περιπτώσεις, δεν μπορεί να καταλάβει πολλά, για τον κώδικα, έτσι ώστε να τον αντικαταστήσει με έναν ουσιαστικά διαφορετικό και πιο αποτελεσματικό αλγόριθμο. Ο μεταγλωττιστής γνωρίζει μόνο πώς να εφαρμόσει αλλαγές χαμηλού επιπέδου στη σημασιολογία, χρησιμοποιώντας γενικές τεχνικές όπως αλγεβρικές ταυτότητες λ.χ. $i + 0 = i$ ή σημασιολογικές ιδιότητες όπως το γεγονός ότι η εκτέλεση της ίδιας πράξεις με τις ίδιες τιμές προκαλούν το ίδιο αποτέλεσμα. Στη συνέχεια θα εξετάσουμε αναλυτικά όλες αυτές τις τεχνικές με τις οποίες μπορούμε να βελτιστοποιήσουμε ένα πρόγραμμα και για να κατανοήσουμε την διαισθητική τους λειτουργία θα αναφέρουμε συγκεκριμένα παραδείγματα.

5.1.1 Πλεονασμοί

Σε ένα πρόγραμμα υπάρχουν πολλές περιττές εργασίες και μερικές φορές αυτές είναι εμφανείς και στον αρχικό κώδικα. Για παράδειγμα, ο προγραμματιστής μπορεί να θεωρήσει πιο κατανοητό και πιο βολικό να ξαναϋπολογίσει κάποιο αποτέλεσμα, αφήνοντας στον μεταγλωττιστή να αναγνωρίσει αυτόν τον πλεονασμό. Αλλά συχνά, οι πλεονασμοί προκύπτουν σαν ένα παράπλευρο αποτέλεσμα του γεγονότος ότι το πρόγραμμα έχει συνταχθεί σε μία υψηλού επιπέδου γλώσσα. Στις περισσότερες γλώσσες, οι οποίες δεν υποστηρίζουν την χρήση δεικτών, οι προγραμματιστές για να αναφερθούν στο στοιχείο ενός πίνακα ή στο στοιχείο μίας δομής, δεν έχουν άλλη

επιλογή από το να χρησιμοποιήσουν μία εντολή του τύπου $A[i][j]$ ή του τύπου $X \leftarrow f1$.

Καθώς γίνεται η μεταγλώττιση του προγράμματος, καθεμία εντολή προσπέλασης, υψηλού επιπέδου, μίας δομής δεδομένων χωρίζεται σε ένα πλήθος εντολών χαμηλού επιπέδου, όπως π.χ. ο υπολογισμός της θέσης (i, j) ενός στοιχείου του πίνακα. Οι προσπελάσεις στην ίδια δομή δεδομένων έχει ως αποτέλεσμα την δημιουργία πολλών κοινών ενεργειών χαμηλού επιπέδου. Οι προγραμματιστές δεν έχουν επίγνωση για αυτούς τους πλεονασμούς και γι' αυτό δεν μπορούν να τους αφαιρέσουν μόνοι τους. Από την άλλη μεριά όμως, είναι προτιμότερο οι προγραμματιστές να χρησιμοποιούν εντολές υψηλού επιπέδου, γιατί έτσι το πρόγραμμα είναι πιο εύκολο να υλοποιηθεί και να κατανοηθεί. Αφήνοντας στον μεταγλωττιστή την αφαίρεση αυτών των πλεονασμών, έχουμε σαν αποτέλεσμα ένα πρόγραμμα εύκολο στην υλοποίηση και ταυτόχρονα αποδοτικό.

Στη συνέχεια θα παρουσιάσουμε ένα τμήμα κώδικα ταξινόμησης, του αλγορίθμου quicksort, το οποίο αποτελεί ένα καλό παράδειγμα για την επίδειξη όλων των βελτιστοποιήσεων. Ο κώδικας, σε γλώσσα C, του αλγορίθμου φαίνεται στην εικόνα 5.1 και είναι ο ίδιος με αυτόν που παρήγαγε ο Sedgewick για να συζητήσει παρόμοια θέματα βελτιστοποίησης. Στη συνέχεια θα παρουσιάσουμε μόνο τρόπους για την βελτιστοποίηση και δεν θα αναλύσουμε τον τρόπο λειτουργίας του αλγορίθμου.

```
void quicksort (int m, int n){
	int i, j;
	int v, x;
	if(n<=m) return;
	i = m-1;
	j = n;
	v = a[n];
	while(1){
		do i = i + 1; while(a[i] < v);
		do j = j - 1; while(a[j] > v);
		if(i>=j) break;
		x = a[i];
		a[i] = a[j];
		a[j] = x;
	}
	x = a[i];
	a[i] = a[n];
	a[n] = x;
	quicksort(m, j);
	quicksort(i+1, n);
}
```

Εικόνα 5.1: Ο αλγόριθμος quicksort.

Πριν ξεκινήσουμε την βελτιστοποίηση του παραπάνω κώδικα, θα πρέπει να τον μεταφέρουμε σε μορφή ενδιάμεσου κώδικα. Έτσι, οι εντολές προσπέλασης και ανάθεσης ενός πίνακα θα σπάσουν σε περισσότερες χαμηλότερου επιπέδου εντολές και οι πλεονασμοί θα είναι πλέον εμφανείς. Ο παραπάνω κώδικας σε μορφή ενδιάμεσου κώδικα τριών-διευθύνσεων φαίνεται στην εικόνα 5.2. Όπως μας δείχνει η εικόνα οι προσωρινές μεταβλητές χρησιμοποιούνται για να αποθηκευτούν τα αποτελέσματα από τις ενδιάμεσες εκφράσεις.

```
(1)  i = m-1
(2)  j = n
(3)  t1 = 4*n
(4)  v = a[t1]
(5)  i = i+1
(6)  t2 = 4*i
(7)  t3 = a[t2]
(8)  if t3<v goto (5)
(9)  j = j-1
(10) t4 = 4*j
(11) t5 = a[t4]
(12) if t5>v goto (9)
(13) if i>=j goto (23)
(14) t6 = 4*i
(15) x = a[t6]
(16) t7 = 4*i
(17) t8 = 4*j
(18) t9 = a[t8]
(19) a[t7] = t9
(20) t10 = 4*j
(21) a[t10] = x
(22) goto (5)
(23) t11 = 4*i
(24) x = a[t11]
(25) t12 = 4*i
(26) t13 = 4*n
(27) t14 = a[t13]
(28) a[t12] = t14
(29) t15 = 4*n
(30) a[15] = x
```

Εικόνα 5.2: Ο αλγόριθμος quicksort σε μορφή ενδιάμεσου κώδικα τριών-διευθύνσεων.

Θεωρώντας ότι οι ακέραιοι καταλαμβάνουν χώρο $4\ byte$, οι εντολές προσπέλασης σε πίνακα της μορφής $x = a[i]$ αντικαθίστανται από τις δύο ακόλουθες εντολές:

$$t6 = 4 * i$$

$$x = a[t6]$$

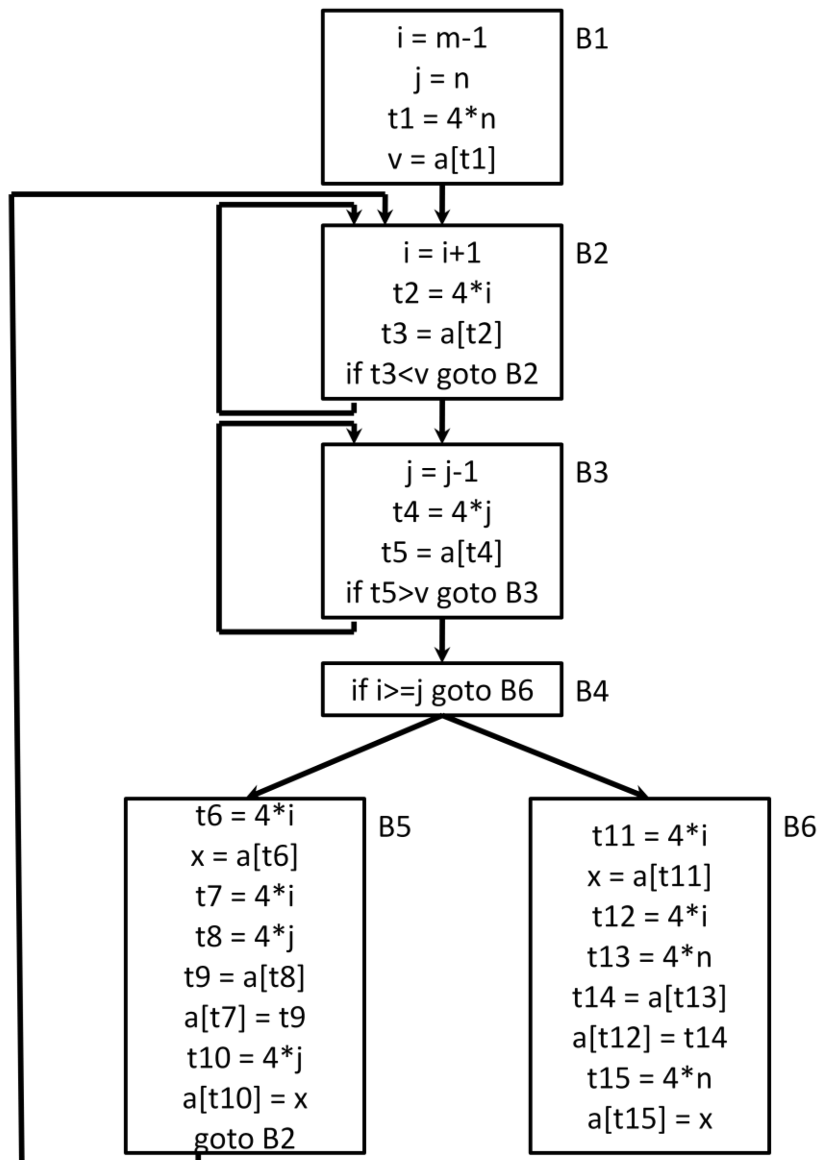
όπως φαίνεται στα βήματα (14) και (15), ενώ οι εντολές ανάθεσης σε πίνακα της μορφής $a[j] = x$ αντικαθίστανται από τις δύο ακόλουθες εντολές:

$$t10 = 4 * j$$

$$a[t10\ ] = x$$

όπως φαίνεται στα βήματα (20) και (21). Παρατηρήστε ότι κάθε εντολή προσπέλασης και ανάθεσης σε πίνακα μεταφράζεται σε ένα ζευγάρι από βήματα. Αυτό έχει σαν αποτέλεσμα το κομμάτι αυτό του κώδικα να μεταφραστεί σε μία μεγάλη ακολουθία από εντολές τριών-διευθύνσεων. Στην εικόνα 5.3 παρουσιάζεται το γράφημα ροής του αλγορίθμου quicksort. Το βασικό μπλοκ $B1$ αποτελεί το μπλοκ Είσοδος (Entry) γιατί εκεί βρίσκεται η πρώτη εντολή. Κάθε άλμα, υπό όρους ή μη, σε μία εντολή στον ενδιάμεσο κώδικα της εικόνας 5.2, γίνεται, στην εικόνα 5.3, άλμα στο βασικό μπλοκ

που περιέχει την εντολή αυτή σαν αρχική εντολή. Τα μπλοκ $B2$ και $B3$ δημιουργούν ένα βρόχο με τον εαυτό τους, ενώ τα μπλοκ $B2, B3, B4, B5$ δημιουργούν έναν τρίτο βρόχο.



Εικόνα 5.3: Το διάγραμμα ροής του αλγορίθμου quicksort.

5.1.2 Σημασιολογική Διατήρηση

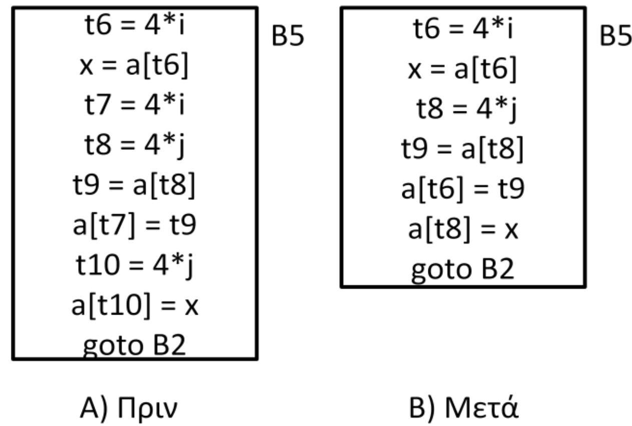
Υπάρχουν πολλές τεχνικές βελτιστοποίησης οι οποίες διατηρούν τη σημασιολογία του αρχικού προγράμματος. Μερικά παραδείγματα τέτοιων τεχνικών αποτελούν η αφαίρεση κοινών εντολών (common-subexpression elimination), η μεταφορά

αντιγράφου (copy propagation), η αφαίρεση του «νεκρού» κώδικα (dead-code elimination) και η αναδίπλωση σταθερών (constant folding). Σχεδόν πάντα, ένα πρόγραμμα περιέχει εντολές υπολογισμού της ίδιας τιμής, όπως συμβαίνει στη δομή του πίνακα. Όπως αναφέραμε και παραπάνω, τέτοιοι πλεονασμοί είναι αναπόφευκτοι επειδή βρίσκονται σε χαμηλότερο επίπεδο από εκείνο στο οποίο μπορεί να φτάσει ο προγραμματιστής, λόγο της γλώσσας προγραμματισμού που χρησιμοποιεί. Για παράδειγμα, στο βασικό μπλοκ $B5$, του γραφήματος ροής της εικόνας 5.3, οι εντολές $4 * i$ και $4 * j$ εκτελούνται δύο φορές, άσχετα από το γεγονός ότι ο προγραμματιστής δεν ζητά ρητά αυτόν τον επαναϋπολογισμό.

5.1.3 Κοινές Εντολές σε Ολόκληρο το Πρόγραμμα

Η ύπαρξη μίας εντολής E λέγεται ότι αποτελεί κοινή εντολή εάν η E έχει επαναϋπολογισθεί προηγουμένως και εάν οι τιμές των μεταβλητών της E δεν αλλάζουν από τον προηγούμενο υπολογισμό. Έτσι, αποφεύγουμε να επαναϋπολογίσουμε το E εάν μπορούμε να χρησιμοποιήσουμε την προηγούμενη τιμή του, δηλαδή την τιμή x στην οποία δεν έχει γίνει άλλη ανάθεση στο ενδιάμεσο διάστημα. Σε περίπτωση που η τιμή του x αλλάξει στο ενδιάμεσο διάστημα, υπάρχει πάλι η δυνατότητα να χρησιμοποιήσουμε την τιμή, που υπολογίστηκε, από την αρχική εκτέλεση της E εάν δημιουργήσουμε μία νέα μεταβλητή y και κάνουμε την ανάθεση $y = x$. Έτσι, μπορούμε στη συνέχεια να κάνουμε χρήση της μεταβλητής y η οποία, στο ενδιάμεσο διάστημα, δεν αλλάζει.

Ένα παράδειγμα τοπικής αφαίρεσης κοινών εντολών αποτελούν οι εντολές ανάθεσης στο $t7$ και στο $t10$ στο βασικό μπλοκ $B5$, στο γράφημα ροής της εικόνας 5.3. Η εικόνα 5.4 δείχνει την αλλαγή η οποία συμβαίνει στο μπλοκ $B5$. Οι αναθέσεις στο $t6$ και στο $t8$ αντικαθιστούν εκείνες του $t7$ και του $t10$ αντίστοιχα.



Εικόνα 5.4: Τοπική αφαίρεση κοινών εντολών στο βασικό μπλοκ B5.

Στην εικόνα 5.5 φαίνεται το αποτέλεσμα από την αφαίρεση των κοινών εντολών, και σε τοπικό και σε ευρύτερο επίπεδο, στα βασικά μπλοκ $B5$ και $B6$. Μετά την τοπική αφαίρεση των κοινών εντολών, στο βασικό μπλοκ $B5$ εξακολουθούν να υπάρχουν οι εντολές $4 * i$ και $4 * j$, οι οποίες έχουν ξαναϋπολογιστεί σε άλλο βασικό μπλοκ. Έτσι οι εντολές

$$t8 = 4 * j$$

$$t9 = a[t8]$$

$$a[t8] = x$$

θα αντικατασταθούν από τις εντολές

$$t9 = a[t4]$$

$$a[t4] = x$$

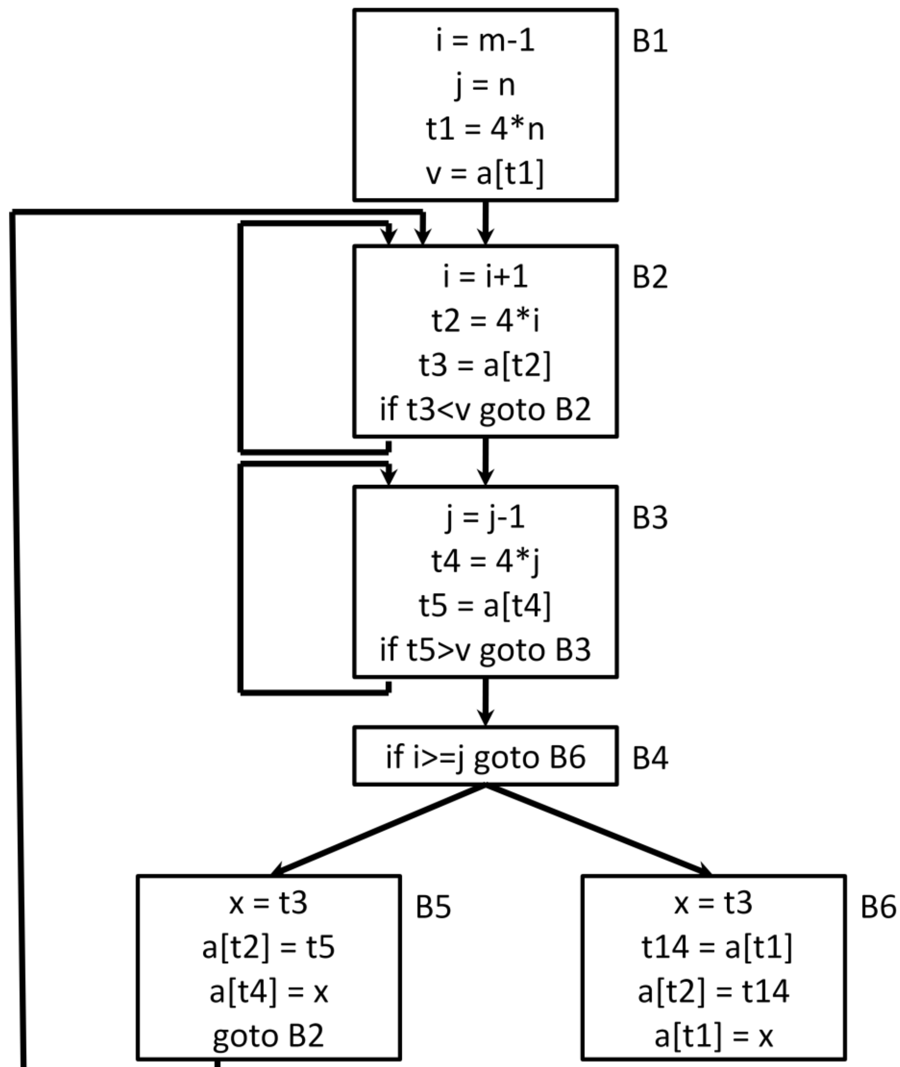
οι οποίες χρησιμοποιούν τη προσωρινή μεταβλητή $t4$ η οποία υπολογίστηκε στο βασικό μπλοκ $B3$. Στο παράδειγμα της εικόνα 5.5 ελέγχτηκε σωστά ότι δεν υπάρχει μεταβολή στο j και στο $t4$ στη διαδρομή στην οποία ο έλεγχος περνάει από το σημείο που αποτιμάται η τιμή το $4 * j$, στο $B3$, έως το $B5$. Έτσι η τιμή του $t4$ μπορεί να χρησιμοποιηθεί εάν χρειάζεται. Στη συνέχεια, ακόμα μία κοινή εντολή ανακαλύπτεται στο $B5$ από την στιγμή που το $t4$ αντικαταστήσει το $t8$. Η νέα έκφραση του $a[t4]$ αντιστοιχεί στη δήλωση $a[j]$ του αρχικού προγράμματος. Στην διαδρομή από το $B3$ στο $B5$, δεν διατηρεί μόνον το j την τιμή αλλά και το $a[j]$, του οποίου η τιμή αποθηκεύτηκε στην προσωρινή μεταβλητή $t5$. Αυτό συμβαίνει επειδή δεν υπάρχουν αναθέσεις στον πίνακα a στο ενδιάμεσο διάστημα. Έτσι, οι εντολές

$$t9 = a[t4]$$

$$a[t6] = t9$$

μπορούν να αντικατασταθούν από την εντολή

$$\alpha[t6] = t5$$



Εικόνα 5.5: Το διάγραμμα ροής μετά την αφαίρεση των κοινών εντολών στα βασικά μπλοκ B5 και B6.

Ανάλογα, η τιμή που ανατίθεται στο x στο μπλοκ $B5$ της εικόνας 5.4 φαίνεται να είναι η ίδια με την τιμή που ανατίθεται στο $t3$ στο μπλοκ $B2$. Το μπλοκ $B5$ στην εικόνα 5.5 είναι το αποτέλεσμα από την αφαίρεση των κοινών εντολών από το μπλοκ $B5$ της εικόνας 5.4(B). Μία παρόμοια σειρά με αλλαγές συμβαίνει στο βασικό μπλοκ $B6$ στην εικόνα 5.5. Σε αντίθεση με τα προηγούμενα, η έκφραση $a[t1]$ στα μπλοκ $B1$ και $B6$, στην εικόνα 5.5, δεν θεωρείται κοινή εντολή, άσχετα με το γεγονός ότι το $t1$ μπορεί να χρησιμοποιηθεί και στα δύο μπλοκ. Αυτό συμβαίνει επειδή όταν ο έλεγχος φύγει

από το $B1$ και πριν φτάσει στο $B6$, μπορεί να πάει πρώτα στο $B5$, όπου υπάρχουν αναθέσεις στον πίνακα a. Έτσι, το $a[t1]$ μπορεί να μην έχει την ίδια τιμή όταν φτάνει ο έλεγχος στο $B6$ με αυτή που είχε όταν ο έλεγχος ήταν στο $B1$ και έτσι δεν είναι σωστό να χρησιμοποιήσουμε την μεταβλητή v, στην οποία κάνουμε ανάθεση την τιμή $a[t1]$.

5.1.4 Μεταφορά Αντιγράφου

Το μπλοκ $B5$ στην εικόνα 5.5 μπορεί ακόμα να βελτιωθεί αφαιρώντας το x. Αυτό μπορούμε να το καταφέρουμε χρησιμοποιώντας δύο νέες αλλαγές. Η μία αφορά αναθέσεις της μορφής $u = v$ οι οποίες ονομάζονται αναθέσεις αντιγραφής. ή αλλιώς αντιγραφή. Η ιδέα που κρύβεται πίσω από την μεταφορά αντιγράφου είναι να χρησιμοποιήσουμε το v αντί του u, όπου είναι δυνατό, μετά την ανάθεση αντιγραφής $u = v$. Για παράδειγμα, η ανάθεση $x = t3$ στο μπλοκ $B5$ είναι μία αντιγραφή. Αν εφαρμόσουμε διάδοση αντιγράφου στο μπλοκ $B5$ θα έχουμε σαν αποτέλεσμα τον κώδικα στην εικόνα 5.6. Αν και αυτή η αλλαγή δεν φαίνεται να προκαλεί βελτίωση, παρόλα αυτά μας δίνει την ευκαιρία να διαγράψουμε την ανάθεση στο x με την παρακάτω τεχνική.

```
x = t3
a[t2] = t5
a[t4] = t3
goto B2
```

Σχήμα 5.6: Το βασικό μπλοκ B5 μετά την εφαρμογή της μεταφοράς αντιγράφου.

5.1.5 Αφαίρεση «Νεκρού» Κώδικα

Μία μεταβλητή λέμε ότι είναι «ζωντανή» σε ένα σημείο κώδικα αν η τιμή της χρησιμοποιείται στη συνέχεια. Αλλιώς, η μεταβλητή είναι «νεκρή» σε εκείνο το σημείο. Μία παρόμοια ιδέα αποτελούν οι δηλώσεις «νεκρού» ή άχρηστου κώδικα οι οποίες υπολογίζουν τιμές που δεν χρησιμοποιούνται. Αν και είναι ασυνήθιστο οι

προγραμματιστές να εισάγουν «νεκρό» κώδικα κατά λάθος, μπορεί, ωστόσο, να εμφανιστεί ως αποτέλεσμα προηγούμενων αλλαγών. Για παράδειγμα, θεωρείστε ότι η μεταβλητή x παίρνει τις τιμές $TRUE$ ή $FALSE$ σε διάφορα σημεία του προγράμματος και χρησιμοποιείται σε μία δήλωση της μορφής

$$if\ (x)\ then\ \{\dots\}$$

Ο μεταγλωττιστής μπορεί να συμπεράνει ότι κάθε φορά που το πρόγραμμα φτάνει σε αυτή τη δήλωση η τιμή του x είναι $FALSE$. Συνήθως, γιατί υπάρχει μία ανάθεση της μορφής $x = FALSE$, οι οποία πρέπει να είναι η τελευταία ανάθεση στο x πριν την εξέταση της τιμής του από την δήλωση, χωρίς να έχει σημασία πια ακολουθία από διακλαδώσεις ακολούθησε το πρόγραμμα. Εάν η τεχνική μεταφοράς αντιγράφου αντικαταστήσει το x με $FALSE$ τότε οι εντολές μέσα στις αγκύλες της if αποτελούν «νεκρό» κώδικα επειδή δεν πρόκειται να εκτελεστούν ποτέ. Έτσι, μπορούμε να αφαιρέσουμε με ασφάλεια και τον έλεγχο του x και τις εντολές στις αγκύλες. Πιο γενικά, το συμπέρασμα στον χρόνο μεταγλώττισης ότι η τιμή μίας μεταβλητής είναι σταθερά και η αντικατάσταση αυτής της σταθεράς σε κάθε χρήση της μεταβλητής λέγεται αναδίπλωση σταθεράς. Ένα πλεονέκτημα της μεταφοράς αντιγράφου είναι ότι συχνά καθιστά τις δηλώσεις αντιγραφής «νεκρό» κώδικα. Για παράδειγμα, η μεταφορά αντιγράφου η οποία ακολουθείται από αφαίρεση «νεκρού» κώδικα αφαιρεί την ανάθεση στο x και αλλάζει τον κώδικα της εικόνας 5.6 σε εκείνο της εικόνας 5.7.

```
a[t2] = t5
a[t4] = t3
goto B2
```

Εικόνα 5.7: Ο κώδικας μετά την αφαίρεση του «νεκρού» κώδικα.

5.1.6 Έλεγχος της Κίνησης του Κώδικα

Οι βρόχοι είναι πολύ σημαντικά σημεία για βελτιστοποίηση, ιδίως το εσωτερικό τους γιατί εκεί τα προγράμματα σπαταλούν τον περισσότερο χρόνο, για την εκτέλεση τους. Ο χρόνος εκτέλεσης ενός προγράμματος μπορεί να μειωθεί αν μειώσουμε τον αριθμό των εντολών στο εσωτερικό του βρόχου, ακόμη και αν αυξήσουμε των αριθμό των εντολών στο εξωτερικό αυτών. Μία τεχνική η οποία μπορεί να μειώσει τον

αριθμό των εντολών στο εσωτερικό ενός βρόχου είναι ο έλεγχος της κίνησης του κώδικα (code motion). Αυτή η τεχνική παίρνει μία εντολή η οποία αποδίδει το ίδιο αποτέλεσμα ανεξάρτητα από το πόσες επαναλήψεις θα εκτελεστούν και κάνει αποτίμηση της εντολής πριν την έναρξη του βρόχου. Να σημειώσουμε ότι η έκφραση «πριν το βρόχο» αναφέρεται στην εντολή εισόδου στο βρόχο, δηλαδή την εντολή η οποία αποτελεί στόχο κάθε άλματος εξωτερικά από το βρόχο. Ένα παράδειγμα όλων αυτών αποτελεί η παρακάτω εντολή:

$$while\ (i \leq x - 2)\{\dots\}$$

Η αποτίμηση της τιμής $x - 2$ έχει ως αποτέλεσμα την ίδια τιμή ανεξάρτητα από το πόσες φορές θα εκτελεστεί ο βρόχος while. Ο έλεγχος της κίνησης του κώδικα θα έχει ως αποτέλεσμα:

$$t = x - 2$$

$$while\ (i \leq t)\{\dots\}$$

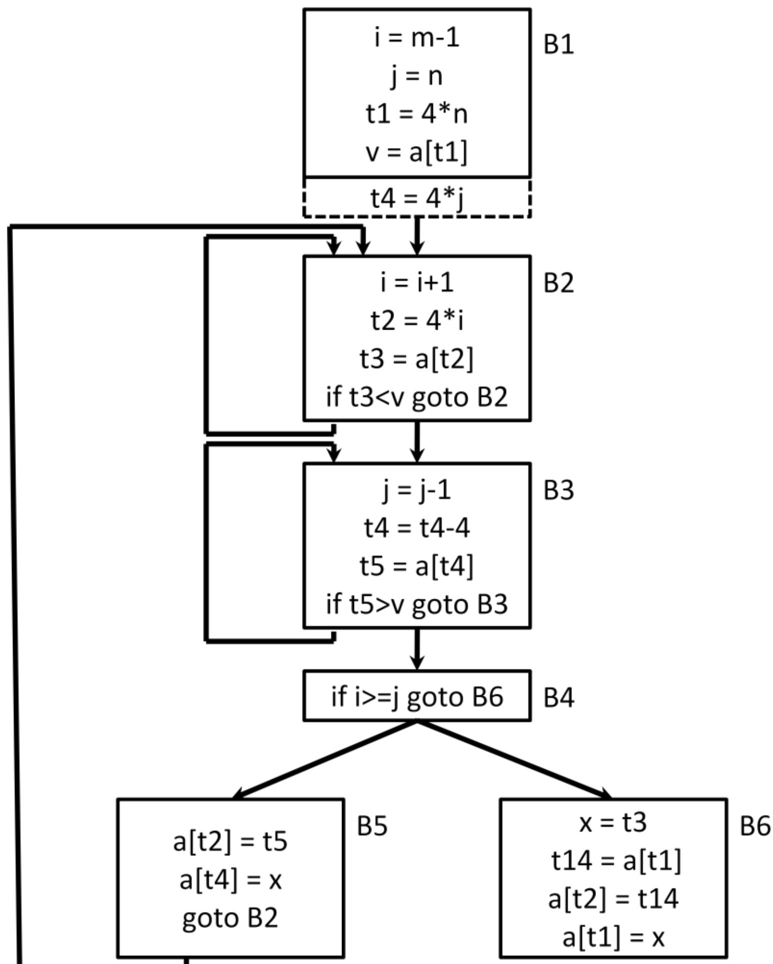
Τώρα ο υπολογισμός της εντολής $x - 2$ θα εκτελεστεί μία φορά, πριν το πρόγραμμα ξεκινήσει να εκτελεί τον βρόχο. Αντίθετα, αν δεν γινόταν αυτή η αλλαγή, η εντολή θα εκτελούνταν $n + 1$ φορές, δεδομένου ότι το σώμα του βρόχου εκτελούνταν n φορές.

5.1.7 Επαγωγικές Μεταβλητές και Μείωση της Ισχύς

Μία ακόμα σημαντική τεχνική αποτελεί η εύρεση των επαγωγικών μεταβλητών στους βρόχους και η βελτίωση του υπολογισμού τους. Μία μεταβλητή x λέγεται ότι είναι μία «επαγωγική μεταβλητή» αν υπάρχει μία θετική ή μία αρνητική σταθερά c τέτοια ώστε κάθε φορά που γίνεται ανάθεση στο x, η τιμή του αυξάνει κατά c. Για παράδειγμα, το i και το $t2$ είναι επαγωγικές μεταβλητές στον βρόχο του βασικού μπλοκ $B2$ με τον εαυτό του, στην εικόνα 5.5. Οι επαγωγικές μεταβλητές μπορούν να υπολογιστούν απλά με μία πρόσθεση ή με μία αφαίρεση σε κάθε επανάληψη του βρόχου. Η αντικατάσταση ενός ακριβού τελεστή, όπως αυτός του πολλαπλασιασμού, με έναν φθηνό, όπως εκείνου της πρόσθεσης, λέγεται μείωση στην ισχύ. Οι επαγωγικές μεταβλητές όμως δεν μας επιτρέπουν μόνο να κάνουμε μείωση στην ισχύ, συχνά μας επιτρέπει να αφαιρέσουμε όλες τις επαγωγικές μεταβλητές εκτός από μία ομάδα αυτών των οποίων οι τιμές παραμένουν κλειδωμένες στο βήμα εκτέλεσης των επαναλήψεων στο βρόχο.

Όταν επεξεργαζόμαστε τους βρόχους είναι χρήσιμο να το κάνουμε από μέσα προς τα έξω, δηλαδή θα πρέπει να ξεκινήσουμε από τον εσωτερικό βρόχο και σταδιακά να επεξεργαζόμαστε τους μεγαλύτερους εξωτερικούς βρόχους, οι οποίοι περιβάλουν τον αρχικό. Για να δούμε πως εφαρμόζεται αυτή η τεχνική στο παράδειγμα του αλγορίθμου quicksort θα ξεκινήσουμε από τον πιο εσωτερικό βρόχο, όπως τον βρόχο που σχηματίζει ο $B3$ με τον εαυτό του. Παρατηρήστε ότι η τιμές του j και του $t4$ παραμένουν σε κλειδωμένο βήμα. Κάθε φορά που η τιμή του j μειώνεται κατά 1, η τιμή του $t4$ μειώνεται κατά 4, επειδή στο $t4$ ανατίθεται η τιμή $4 * j$. Αυτές οι μεταβλητές αποτελούν ένα καλό παράδειγμα ζεύγους επαγωγικών μεταβλητών.

Όταν υπάρχουν δύο οι περισσότερες επαγωγικές μεταβλητές σε ένα βρόχο, υπάρχει πιθανότητα να μπορούμε να απαλλαγούμε από όλες εκτός από μία. Για το εσωτερικό του βρόχου $B3$, στην εικόνα 5.5, δεν μπορούμε να απαλλαγούμε ούτε από τον j ούτε από τον $t4$ εντελώς, επειδή το j χρησιμοποιείται στο $B3$ και το $t4$ χρησιμοποιείται στο $B4$. Ωστόσο, μπορούμε να εκτελέσουμε μείωση στην ισχύ και μερική αφαίρεση των επαγωγικών μεταβλητών. Τελικά, το j θα αφαιρεθεί όταν επεξεργαστεί ο εξωτερικός βρόχος ο οποίος αποτελείται από τα μπλοκ $B2, B3, B4, B5$ που κλείνει στο μπλοκ $B2$.

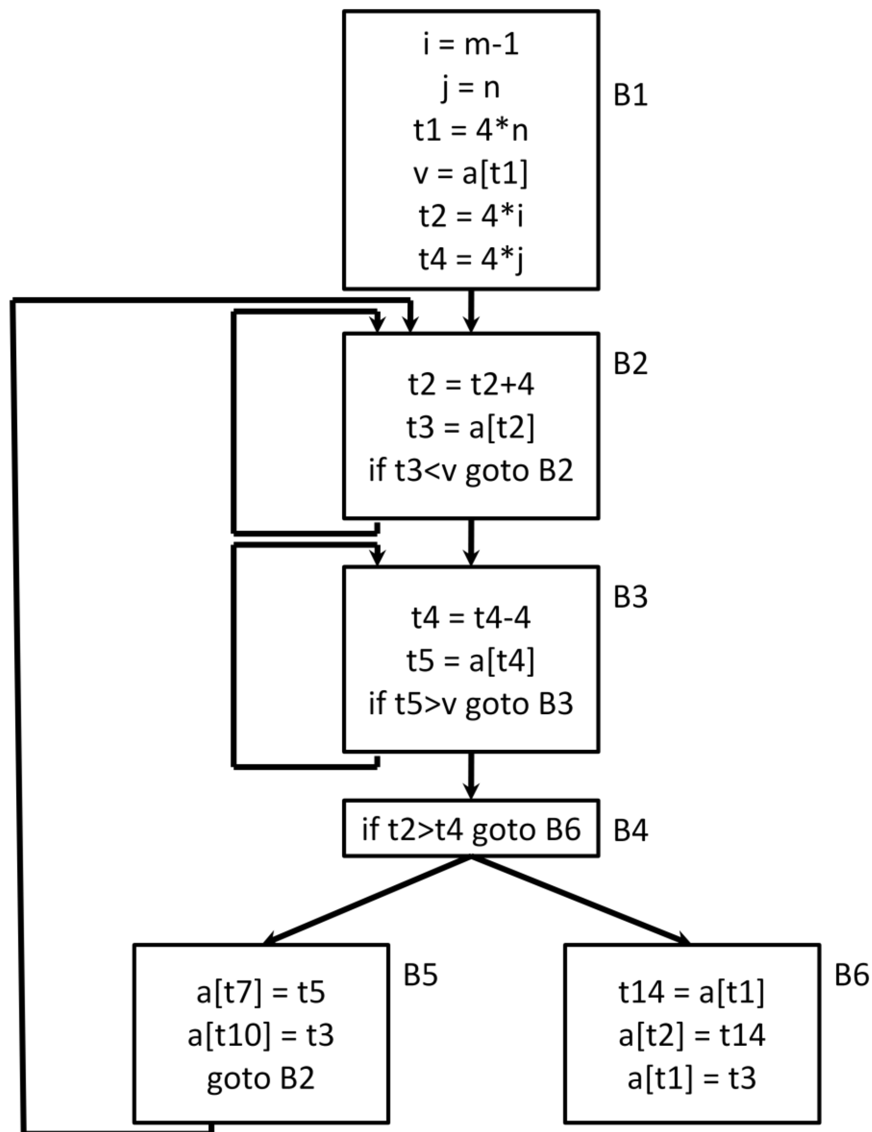


Εικόνα 5.8: Εφαρμογή της τεχνικής μείωσης ισχύος στο B3.

Καθώς η σχέση $t4 = 4 * j$ σύγουρα ισχύει μετά την ανάθεση του $t4$ στην εικόνα 5.5, και το $t4$ δεν αλλάζει πουθενά αλλού στον εσωτερικό βρόχο του $B3$, μετά την δήλωση $j = j - 1$ η σχέση $t4 = 4 * j + 4$ πρέπει να ισχύει. Έτσι, μπορούμε να αντικαταστήσουμε την ανάθεση $t4 = 4 * j$ με την $t4 = t4 - 4$. Το μόνο πρόβλημα είναι ότι η μεταβλητή $t4$ δεν έχει τιμή όταν μεταβαίνουμε στο μπλοκ $B3$ την πρώτη φορά. Από την στιγμή που πρέπει να κατασκευάσουμε τη σχέση $t4 = 4 * j$ στη είσοδο στο μπλοκ $B3$, τοποθετούμε μία αρχικοποίηση της μεταβλητής $t4$ στο τέλος του μπλοκ στο οποίο η μεταβλητή j αρχικοποιείται, όπως φαίνεται από την εντολή που προστέθηκε στο μπλοκ $B1$ στην εικόνα 5.8. Αν και έχουμε εισάγει μία εντολή στο μπλοκ $B1$, η οποία εκτελείται μόνο μία φορά, η αντικατάσταση του πολλαπλασιασμού με αφαίρεση επιταχύνει πολύ το πρόγραμμα, αν ο πολλαπλασιασμός χρειάζεται περισσότερο χρόνο για την εκτέλεση από ότι η πρόσθεση και η αφαίρεση, γεγονός το οποίο συμβαίνει στις περισσότερες μηχανές.

Τελειώνοντας, θα παρουσιάσουμε και άλλη μία εκτέλεση της αφαίρεσης μεταβατικών μεταβλητών. Σε αυτή την εκτέλεση μεταχειριζόμαστε τις μεταβλητές i και j στο ίδιο πλαίσιο του εξωτερικού βρόχου ο οποίος αποτελείται από τα μπλοκ $B2, B3, B4$ και $B5$. Αφού εφαρμοστή η τεχνική μείωσης της ισχύεις στο εσωτερικό των βρόχων $B2$ και $B3$, η μόνη χρήση του i και του j είναι να καθορίσουν το αποτέλεσμα του έλεγχου στο μπλοκ $B4$. Γνωρίζουμε ότι οι τιμές των i και $t2$ ικανοποιούν τη σχέση $t2 = 4 * i$, την ίδια στιγμή που οι μεταβλητές j και $t4$ ικανοποιούν τη σχέση $t4 = 4 * j$. Έτσι, ο έλεγχος $t2 \geq t4$ μπορεί να αντικατασταθεί από τον έλεγχο $i \geq j$. Από την στιγμή που γίνει αυτή η τοποθέτηση, το i στο βασικό μπλοκ $B2$ και το j στο βασικό μπλοκ $B3$ μετατρέπονται σε «νεκρό» κώδικα όπως επίσης και οι αναθέσεις σε αυτά και μπορούν να αφαιρεθούν με ασφάλεια. Το τελικό γράφημα ροής φαίνεται στην εικόνα 5.9.

Οι τεχνικές βελτίωσης κώδικα που συζητήσαμε, εφαρμόστηκαν στον αλγόριθμο quicksort που αποτελεί το παράδειγμα μας. Στην εικόνα 5.9 φαίνεται ότι οι εντολές στο βασικό μπλοκ $B2$ και $B3$ μειώθηκαν από 4 σε 3, σε σύγκριση με το αρχικό γράφημα ροής της εικόνας 5.3. Στο μπλοκ $B5$, ο αριθμός έχει μειωθεί από 9 σε 3, ενώ στο $B6$ από 8 σε 3. Αντίθετα, οι εντολές στο μπλοκ $B1$ αυξήθηκαν από 4 σε 6, αλλά το μπλοκ $B1$ εκτελείται μόνο μία φορά, οπότε ο συνολικός χρόνος εκτέλεσης επηρεάζεται ελάχιστα από την αύξηση των εντολών του μπλοκ $B1$.



Εικόνα 5.9: Το διάγραμμα ροής μετά την αφαίρεση των επαγωγικών μεταβλητών.

5.2 Μεταφορά Σταθεράς

Μία από τις πιο απλές, σαν ιδέα, και ταυτόχρονα χρήσιμες τεχνικές βελτιστοποίησης κώδικα είναι η μεταφορά σταθεράς (constant propagation), ή όπως αναφέρθηκε παραπάνω αναδίπλωση σταθεράς (constant folding). Σε αυτή την τεχνική εξετάζουμε για κάθε μεταβλητή ολόκληρο τον κώδικα για να βρούμε όλες τις δηλώσεις ανάθεσης σε αυτή την μεταβλητή. Αν μετά από αυτόν τον έλεγχο καταλήξουμε στο συμπέρασμα ότι η τιμή αυτής της μεταβλητής, σε όλο το πρόγραμμα, είναι σταθερή τότε αντικαθιστούμε κάθε αναφορά της, στο πρόγραμμα, με αυτή τη σταθερά. Αν και η ίδια η τεχνική δεν συμβάλει άμεσα στη βελτιστοποίηση του προγράμματος, παρόλα

αυτά άλλες τεχνικές οι οποίες μπορούν να εφαρμοστούν εν συνεχεία, όπως η διαγραφή «νεκρού» κώδικα ή η αφαίρεση πλεονασμών, μπορεί να απαλείψουν επιπλέον τμήματα του κώδικα χωρίς να αλλάζουν τη σημασιολογία του.

Σε αυτή την τεχνική, κάθε αναφορά της μεταβλητής, σε ένα σημείο του προγράμματος, σημειώνεται με έναν από τους ακόλουθους τρεις χαρακτηρισμούς:

1) Μία μεταβλητή σημειώνεται σαν σταθερά (constant) σε αυτό το σημείο αν, ακολουθώντας οποιοδήποτε μονοπάτι, έχει πάντα την ίδια τιμή.
2) Μία μεταβλητή χαρακτηρίζεται ως μη σταθερά (not a constant), ή NAC για συντομία, αν σε αυτή γίνεται ανάθεση μίας τιμής εισόδου (input value), αν παράγεται από μία NAC μεταβλητή ή αν παίρνει διαφορετικές τιμές σε αυτό το σημείο, εξαιτίας της πορείας του προγράμματος.
3) Μία μεταβλητή χαρακτηρίζεται ως απροσδιόριστη (undefined), ή $UNDEF$ για συντομία, αν δεν έχει ανακαλυφθεί κανένας ορισμός της μεταβλητής μέχρι αυτό το σημείο.

Αξίζει να σημειώσουμε ότι οι χαρακτηρισμοί NAC και $UNDEF$ δεν είναι ίδιοι. Για την ακρίβεια είναι αντίθετοι, αφού μία μεταβλητή χαρακτηρίζεται ως NAC αν έχουμε συναντήσει πολλούς ορισμούς της μέχρι αυτό το σημείο, με αποτέλεσμα η τιμή της να αλλάζει ανάλογα με την ροή του προγράμματος, ενώ μία άλλη χαρακτηρίζεται ως $UNDEF$ αν δεν έχουμε συναντήσει κανέναν ορισμό της, μέχρι αυτό το σημείο. Αν μία $UNDEF$ μεταβλητή συναντήσει μία ανάθεση σε σταθερά τότε η μεταβλητή από $UNDEF$ γίνεται σταθερά. Αν μία NAC μεταβλητή συναντήσει μία ανάθεση σε σταθερά τότε ο χαρακτηρισμός της μεταβλητής παραμένει NAC. Αν μία μεταβλητή με σταθερή τιμή x συναντήσει μία δήλωση ανάθεσης της τιμής x, τότε η μεταβλητή παραμένει χαρακτηρισμένη ως σταθερά. Αντίθετα, αν συναντήσει μία δήλωση ανάθεσης της τιμής y τότε από σταθερά η μεταβλητή γίνεται NAC. Τέλος, αν σε μία μεταβλητή x γίνει ανάθεση του αποτελέσματος μίας πράξης, με δυαδικό τελεστή όπως πρόσθεση ή αφαίρεσης, δύο άλλων μεταβλητών τότε ο χαρακτηρισμός της μεταβλητής προσδιορίζεται από τον πίνακα της εικόνας 5.10.

y	z	x
UNDEF	UNDEF	UNDEF
UNDEF	c_2	UNDEF
UNDEF	NAC	NAC
c_1	UNDEF	UNDEF
c_1	c_2	c_1 ¤ c_2
c_1	NAC	NAC
NAC	UNDEF	NAC
NAC	c_2	NAC
NAC	NAC	NAC

Εικόνα 5.10: Ο χαρακτηρισμός της μεταβλητής x μετά την ανάθεση του αποτελέσματος της πράξει y¤z, όπου το ¤ είναι οποιοσδήποτε δυαδικός τελεστής.

Ο χαρακτηρισμός $UNDEF$, εκτός του ότι χρησιμοποιείται για να προσδιορίσει μία μεταβλητή η οποία δεν έχει οριστεί, χρησιμοποιείται και για να αρχικοποιήσει όλες τις μεταβλητές στην έξοδο από τον κόμβο Είσοδος (Entry). Από τον τρόπο που ορίσαμε την αλλαγή του χαρακτηρισμού μίας μεταβλητής, αν σε ένα σημείο μία μεταβλητή παραμείνει με τον χαρακτηρισμό $UNDEF$ τότε αυτό σημαίνει ότι κανένας ορισμός αυτής της μεταβλητής δεν φτάνει σε αυτό το σημείο.

5.3 Βελτιστοποίηση Βρόχων

Μέχρι στιγμής, δεν χειριστήκαμε την ροή μέσα στους βρόχους διαφορετικά σε σχέση με τη ροή σε κάθε άλλη διάταξη. Παρόλα αυτά οι βρόχοι αποτελούν την πιο σημαντική διάταξη ενός προγράμματος, γιατί ο περισσότερος χρόνος εκτέλεσης του προγράμματος σπαταλάτε σε αυτούς. Έτσι, μία ενέργεια βελτιστοποίησης, εντός ενός βρόχου, θα επηρεάσει σημαντικά τον χρόνο εκτέλεσης του. Έτσι, είναι κρίσιμο

να εντοπίσουμε τους βρόχους και να τους μεταχειριστούμε ιδιαίτερα. (Το πώς εντοπίζονται οι βρόχοι το εξετάσαμε στο τέλος της ενότητας 2.3)

Οι βρόχοι μπορούν να επηρεάσουν και τον χρόνο ανάλυσης ενός προγράμματος. Αν το πρόγραμμα δεν περιέχει βρόχους, μπορούμε να απαντήσουμε στο πρόβλημα της ροής με ένα μόνο πέρασμα. Ένας τρόπος για την λύση αυτού του προβλήματος αποτελεί η τοπολογική ταξινόμηση.

5.3.1 Οπισθοακμές και Μειώσιμα Γραφήματα

Στην παράγραφο 1.4.2 μελετήσαμε την αναζήτηση κατά βάθος καθώς και επίσης και τα είδη των ακμών που εμφανίζονται στο επικαλύπτων δέντρο της αναζήτησης κατά βάθος. Στο κεφάλαιο 3 μελετήσαμε τις σχέσεις κυριαρχίας μεταξύ των κόμβων ενός γραφήματος. Αυτές οι δύο έννοιες είναι απαραίτητες για τον ορισμό μίας οπισθοακμής (back edge). Δοθέντος ενός επικαλύπτοντος δέντρου της αναζήτησης κατά βάθος, κάθε ανιούσα ακμή $x \rightarrow y$ σε αυτό αποτελεί μία οπισθοακμή, αν το y είναι γνήσιος κυρίαρχος του x, δηλαδή $y \in stdom(x)$. Όπως μπορούμε να συμπεράνουμε από τον ορισμό, κάθε οπισθοακμή είναι και ανιούσα ακμή. Αντίθετα, κάθε ανιούσα ακμή δεν είναι και οπισθοακμή.

Ένα γράφημα ροής λέμε ότι είναι μειώσιμο (reducible), αν κάθε ανιούσα ακμή του είναι και οπισθοακμή. Δηλαδή, μειώσιμο γράφημα είναι εκείνο το γράφημα που το σύνολο των ανιουσών ακμών είναι το ίδιο με το σύνολο των οπισθοακμών. Μη μειώσιμο (non reducible) γράφημα είναι εκείνο στο οποίο έστω και μία ανιούσα ακμή δεν είναι οπισθοακμή, ή αλλιώς είναι εκείνο στο οποίο αν αφαιρέσουμε όλες τις οπισθοακμές το αποτέλεσμα παραμένει να είναι ένα κυκλικό γράφημα. Τα γραφήματα ροής τα οποία δημιουργούνται από ένα πρόγραμμα είναι συνήθως μειώσιμα γραφήματα. Αυτό συμβαίνει γιατί δηλώσεις, στο πρόγραμμα, όπως $if \ldots then \ldots else$, $while \ldots do$, $continue$ και $break$ δημιουργούν μειώσιμα γραφήματα. Μερικές φορές και η εντολή $goto$ δημιουργεί μειώσιμο γράφημα (ωστόσο σημειώνουμε ότι κατά την διάρκεια της ανάλυσης που πραγματοποιούν κάποιοι μεταγλωττιστές μπορεί να μετατρέψουν ένα μειώσιμα γράφημα σε μη μειώσιμο).

5.3.2 Φυσικοί Βρόχοι

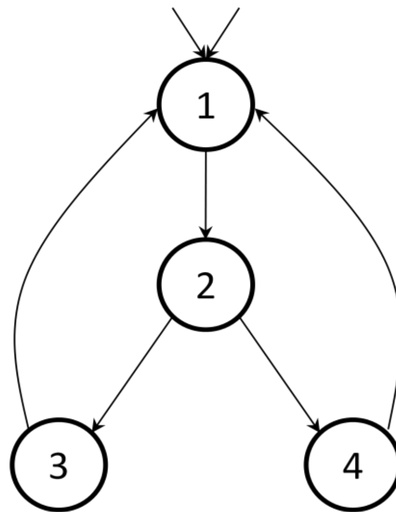
Από την μεριά της ανάλυσης ενός προγράμματος, σημασία δεν έχει να καθορίσουμε την ύπαρξη ή μη των βρόχων. Σημασία έχει να καθορίσουμε πότε ένας βρόχος τηρεί τις προϋποθέσεις για να εφαρμοστεί εύκολη βελτιστοποίηση σε αυτόν. Συγκεκριμένα, εκείνο που μας ενδιαφέρει είναι αν ένας βρόχος έχει μονό κόμβο εισόδου, έτσι ώστε να εξεταστεί, από τον μεταγλωττιστή, αν μπορεί να εφαρμόσει συγκεκριμένους αρχικούς όρους πριν την είσοδο σε αυτόν τον βρόχο. Τέτοιοι βρόχοι ονομάζονται φυσικοί βρόχοι (natural loops) και είναι εκείνοι για τους οποίους ισχύουν οι δύο ακόλουθοι όροι:

1) Ο βρόχος πρέπει να έχει μονό κόμβου εισόδου, ο οποίος ονομάζεται και αρχικός (header). Αυτός ο κόμβος κυριαρχεί επί όλων των κόμβων οι οποίοι συμμετέχουν στον βρόχο γιατί αλλιώς δεν θα ήταν ο μοναδικός αρχικός κόμβος του βρόχου.
2) Πρέπει να υπάρχει μία οπισθοακμή η οποία εισέρχεται στον αρχικό κόμβο. Αλλιώς δεν είναι δυνατόν η ροή να επιστρέψει ξανά στην αρχή του βρόχου.

Δοθέντος μίας οπισθοακμής $x \rightarrow y$, το σύνολο των κόμβων που συμμετέχουν στον βρόχο είναι το y και όλοι εκείνοι οι κόμβοι οι οποίοι μπορούν να προσπελάσουν το x χωρίς να προσπελάσουν το y. Στα μειώσιμα γραφήματα ροής, από τη στιγμή που κάθε ανιούσα ακμή είναι και οπισθοακμή, μπορούμε να συνδέσουμε ένα φυσικό βρόχο με την ανιούσα ακμή, κάτι το οποίο δεν μπορούμε να κάνουμε στα μη μειώσιμα γραφήματα ροής.

Θεωρώντας ότι έχουμε μόνο φυσικούς βρόχους, μας παρέχεται μία ιδιότητα η οποία λέει ότι εκτός και αν οι δύο φυσικοί βρόχοι έχουν κοινό αρχικό κόμβο, τότε είτε δεν έχουν κανένα κοινό κόμβο είτε ο ένας βρόχος είναι εμφωλευμένος στον άλλο. Έτσι προκύπτει ότι ο πιο εσωτερικός βρόχος να είναι αυτός ο βρόχος ο οποίος δεν περιέχει εμφωλευμένους βρόχους. Όταν δύο βρόχοι έχουν κοινό αρχικό κόμβο τότε είναι δύσκολο να προσδιορίσουμε ποιος βρόχος είναι εμφωλευμένος στον άλλο. Έτσι, όταν δύο βρόχοι έχουν κοινό αρχικό κόμβο και ο ένας δεν βρίσκεται εντός του άλλου, τότε μπορούμε να συνενώσουμε τους δύο βρόχους και να τους χειριστούμε σαν έναν. Για παράδειγμα, ας θεωρήσουμε το γράφημα ροής του σχήματος 5.11. Το γράφημα αυτό έχει δύο οπισθοακμές, τις $3 \rightarrow 1$ και $4 \rightarrow 1$, οι οποίες δημιουργούν τους βρόχους, τους {1,2,3} και {1,2,4} αντίστοιχα. Μπορούμε να ενώσουμε τους δύο βρόχους σε έναν, ο οποίος θα περιέχει τους κόμβους {1,2,3,4}. Ωστόσο, αν υπήρχε

και η οπισθοακμή $2 \rightarrow 1$, τότε ο βρόχος {1,2}, που δημιουργείται, αν και έχει αρχικό κόμβο τον 1, βρίσκεται κατάλληλα τοποθετημένος μέσα στον βρόχο {1,2,3,4}. Έτσι, δεν υπάρχει ανάγκη να συνδυάσουμε τους δύο βρόχους, αφού μπορούμε να μεταχειριστούμε τον νέο βρόχο ως εμφωλευμένο.



Εικόνα 5.11: Δύο φυσικοί βρόχοι με ίδιο αρχικό κόμβο, όπου ο ένας δεν περιέχεται στον άλλο.

5.3.3 Σύγκλιση των Επαναληπτικών Αλγορίθμων

Για πολλές τεχνικές της ανάλυσης γραφημάτων ροής, είναι δυνατόν να διατάξουμε τις αποτιμήσεις των εντολών έτσι ώστε οι αλγόριθμοι να συγκλίνουν σε λιγότερες επαναλήψεις. Η ιδιότητα που μας ενδιαφέρει είναι το πότε όλα τα γεγονότα που μας ενδιαφέρουν μπορούν να μεταφερθούν σε ένα κόμβο κατά μήκος ενός άκυκλου μονοπατιού. Όλες οι τεχνικές ανάλυσης γραφημάτων ροής που αναφέραμε, εκτός της μεταφοράς σταθεράς, έχουν αυτή την ιδιότητα. Πιο συγκεκριμένα:

- Αν ένας ορισμός d περνάει στον βασικό μπλοκ B, τότε υπάρχει κάποιο άκυκλο μονοπάτι, από το βασικό μπλοκ που περιέχει το d στο B, τέτοιο ώστε το d εισέρχεται και εξέρχεται από κάθε βασικό μπλοκ του μονοπατιού.
- Εάν μία έκφραση, όπως η έκφραση $x + y$, δεν είναι διαθέσιμη στην είσοδο του βασικού μπλοκ B, τότε υπάρχει ένα άκυκλο μονοπάτι το οποίο είτε ξεκινά από τον κόμβο Είσοδος (Entry) και δεν περιέχει δηλώσεις οι οποίες

καταστρέφουν ή παράγουν το $x+y$, είτε ξεκινάει από ένα κόμβο ο οποίος καταστρέφει την δήλωση $x+y$ και μέχρι το βασικό μπλοκ B δεν υπάρχει μεταγενέστερη δήλωση η οποία να το παράγει ξανά.

- Εάν το x είναι ζωντανό στην έξοδο από το μπλοκ B, τότε υπάρχει ένα άκυκλο μονοπάτι από το B μέχρι το σημείο που χρησιμοποιείται το x, κατά μήκος του οποίου δεν υπάρχει άλλος ορισμός για το x.

Εξετάζοντας μία από αυτές τις περιπτώσεις, τα μονοπάτια με κύκλο δεν προσθέτουν τίποτα. Για παράδειγμα, εάν μία χρήση του x προέρχεται μία ανάθεση σε ένα βασικό μπλοκ B, η οποία φτάνει σε αυτό το σημείο κατά μήκος ενός κυκλικού μονοπατιού βασικών μπλοκ, τότε μπορούμε να αφαιρέσουμε τον κύκλο για να βρούμε το μικρότερο μονοπάτι κατά μήκος του οποίου η χρήση του x προέρχεται από το B. Σε αντίθεση, η μεταφορά σταθεράς δεν έχει αυτή την ιδιότητα. Θεωρείστε ένα απλό πρόγραμμα το οποίο περιέχει ένα βρόχο και ένα βασικό μπλοκ με τις δηλώσεις:

L: $a=b$
$b=c$
$c=1$
$goto\ L$

Την πρώτη φορά που εκτελείται το μπλοκ, το c έχει τιμή ένα και τα a και b είναι απροσδιόριστα. Την δεύτερη φορά που εκτελείται το μπλοκ, η τιμή του c και του b έχουν τιμή 1. Χρειάζεται τρεις εκτελέσεις του βασικού μπλοκ για να γίνει ανάθεση της σταθεράς 1 στην μεταβλητή a.

Αν όλες οι χρήσιμες πληροφορίες μεταφέρονται κατά μήκος ενός άκυκλου μονοπατιού, τότε έχουμε την ευκαιρία να προσαρμόσουμε την σειρά με την οποία ένας επαναληπτικός αλγόριθμος επισκέπτεται τους κόμβους, έτσι ώστε μετά από σχετικά λίγες εκτελέσεις των κόμβων να μπορούμε να είμαστε σίγουροι ότι η πληροφορία έχει περάσει από όλα τα άκυκλα μονοπάτια.

6 ΤΟ ΠΡΟΣΑΥΞΗΤΙΚΟ ΠΡΟΒΛΗΜΑ ΤΩΝ ΚΟΜΒΩΝ ΚΥΡΙΑΡΧΙΑΣ

Όπως αναφέραμε και σε προηγούμενα κεφάλαια οι μεταγλωττιστές των προγραμμάτων κάνουν εκτενή χρήση των πληροφοριών που λαμβάνουν από τους κόμβους κυριαρχίας. Ωστόσο, είναι πιθανό το γράφημα ροής να αλλάξει κατά την διάρκεια βελτιστοποίησης στον κώδικα, όπως τυγχάνει να συμβαίνει με τις αλλαγές στους βρόχους και με τις διαγραφές του «νεκρού» κώδικα. Επιπλέον, σε προγραμματιστικά περιβάλλοντα που χρησιμοποιούν δέντρα κυριαρχίας για την ανάλυση του προγράμματος, οι αλλαγές του προγράμματος από τις επεξεργασίες του χρήστη μπορεί να προκαλέσουν και αλλαγές στο γράφημα ροής. Είναι σημαντικό το δέντρο κυριαρχίας να παραμένει σωστό μετά από τέτοιες αλλαγές στο γράφημα ροής.

Όπως επισήμαναν οι Carroll και Ryder το 1998, «Η έμφυτη δυσκολία στο πρόβλημα της ενημέρωσης των συνόλων κυριαρχίας έγκειται στη μη-τοπικότητα της κυριαρχίας, δηλαδή, δοθέντων δύο κόμβων x και y σε ένα γράφημα ροής, το αν ο x κυριαρχεί επί του y εξαρτάται από την παρουσία η από την απουσία μονοπατιών διαμέσου αυθαίρετων κόμβων μακριά και από το x και από το y. Προσθέτοντας ή αφαιρώντας μία ακμή στο γράφημα ροής, πράξη η οποία μπορεί να προκαλέσει την πρόσθεση ή την αφαίρεση μεγάλου αριθμού μονοπατιών, μπορεί να επηρεαστεί η κυριαρχία μεταξύ κόμβων οι οποίοι βρίσκονται αυθαίρετα μακριά από το σημείο αλλαγής».

Στη συνέχεια θα ασχοληθούμε μόνο για την περίπτωση στην οποία το γράφημα ροής αλλάζει από την προσθήκη μίας ακμής. Θα δείξουμε ποιοι κόμβοι θα επηρεαστούν από αυτή τη προσθήκη και θα αναφέρουμε τρόπους για την σωστή ενημέρωση του δέντρου κυριαρχίας. Για ευκολία, καθώς επίσης και για τον λόγο ότι αποτελεί πιο αποδοτική υλοποίηση, θα θεωρήσουμε ότι μετά τον υπολογισμό των συνόλων κυριαρχίας, σε κάθε κόμβο αποθηκεύεται μόνο ο άμεσος κυρίαρχός του και όχι ολόκληρο το σύνολο. Έτσι θα χρειαστεί να γίνουν αλλαγές μόνο στους κόμβους για τους οποίους αλλάζει ο άμεσος κυρίαρχός τους.

Στο κεφάλαιο αυτό παρουσιάζουμε επίσης συνοπτικά κάποιες ιδέες για την μετατροπή του στατικού επαναληπτικού αλγορίθμου καθώς και του αλγορίθμου

Lengauer-Tarjan σε προσαυξητικούς. Επίσης δίνουμε κάποια πρώτα πειραματικά αποτελέσματα αυτών των μεθόδων. Περισσότερες λεπτομέρειες θα παρουσιαστούν σε άρθρο που ετοιμάζεται για δημοσίευση.

6.1 Αναδόμηση του Δέντρου Κυριαρχίας από την Αρχή

Ο πιο εύκολος αλλά και λιγότερο αποδοτικός τρόπος να ανασκευαστεί το δέντρο κυριαρχίας είναι να ξαναεκτελέσουμε τον επαναληπτικό αλγόριθμο από την αρχή (ή οποιοδήποτε άλλο στατικό αλγόριθμο όπως ο Lengauer-Tarjan). Ξαναεκτελώντας τον επαναληπτικό αλγόριθμο για το καινούριο γράφημα, που προκύπτει από την προσθήκη της νέας ακμής, είμαστε σίγουροι ότι θα λάβουμε το σωστό δέντρο κυριαρχίας. Ωστόσο, δεν χρησιμοποιούμε καθόλου τις πληροφορίες που έχουμε για τους κόμβους κυριαρχίας για το γράφημα πριν την νέα προσθήκη. Αυτός είναι και ο λόγος που καθιστά αυτή τη τεχνική μη αποδοτική. Παρόλα αυτά, αυτή η τεχνική δουλεύει και για προσθήκη και για αφαίρεση ακμών.

6.2 Αναδόμηση του Δέντρου Κυριαρχίας με τον Επαναληπτικό Αλγόριθμο Λαμβάνοντας Υπόψη τα Προηγούμενα Σύνολα Κυριαρχίας

Ένας πιο αποδοτικό τρόπος να υπολογίσουμε τα νέα σύνολα κυριαρχίας είναι να εκτελέσουμε τον επαναληπτικό αλγόριθμο, αρχικοποιώντας τους άμεσους κυρίαρχους κάθε κόμβου με αυτούς που υπολογίσαμε πριν την προσθήκη της νέας ακμής. Έτσι αποκτάμε μία αρχική αίσθηση των νέων κόμβων κυριαρχίας, η οποία διορθώνεται όταν τερματίσει ο αλγόριθμος. Αυτή η τεχνική μπορεί να τερματίσει σε πολύ λιγότερες επαναλήψεις σε σχέση με την προηγούμενη, ανάλογα με το ποια θα είναι η νέα ακμή που θα εισαχθεί. Αυτός ο αλγόριθμος στην χειρότερη περίπτωση θα εκτελέσει τόσες επαναλήψεις όσες θα εκτελέσει και ο προηγούμενος. Μία ακόμα βελτίωση αυτής της ιδέας είναι να μην εκτελούμε από την αρχή καθοδική διερεύνηση σε ολόκληρο το γράφημα αλλά μόνο στα τμήματα του που είναι απαραίτητο.

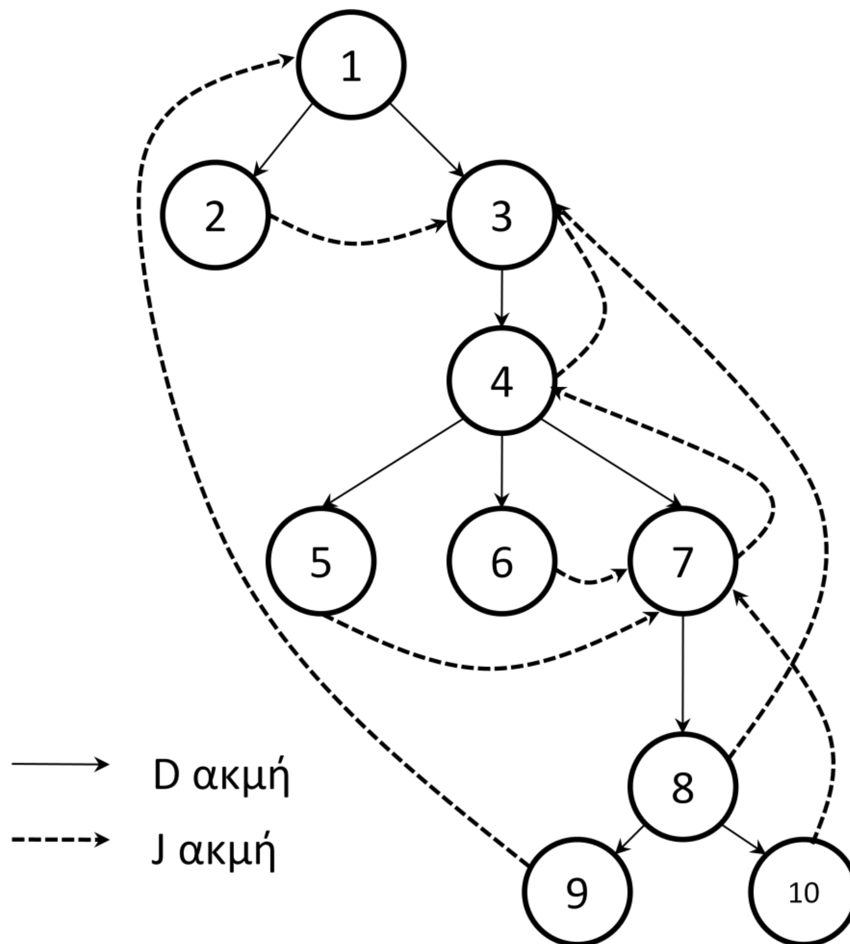
6.3 Προσαυξητικός Αλγόριθμος των Sreedhar, Gao και Lee

Στην ενότητα αυτή παρουσιάζουμε τον αλγόριθμο των Shreedhar, Gao, Lee. Αυτός ο προσαυξητικός αλγόριθμος (incremental) προσφέρει έναν αποδοτικό τρόπο για την

ενημέρωση του δέντρου κυριαρχίας μετά την εισαγωγή μίας νέας ακμής, υπολογίζοντας το ακριβές σύνολο των κόμβων που επηρεάζονται από την εισαγωγή της νέας ακμής. Ένας κόμβος λέμε ότι ανήκει στο σύνολο των κόμβων που επηρεάστηκαν, αν μετά την εισαγωγή της νέας ακμής ο άμεσος κυρίαρχος κόμβος του αλλάξει. Η κύρια αιτία για την αποδοτικότητα αυτού του αλγορίθμου είναι το γεγονός ότι μετά την εισαγωγή της ακμής $(x \rightarrow y)$ το σύνολο των κόμβων που θα επηρεαστούν είναι συσχετισμένο με το σύνολο των κόμβων που συμμετέχουν στο $IDF(y)$ (iterated dominance frontiers, που θα αναλυθεί παρακάτω). Οι Ramalingam και Reps [18] έδειξαν το 1994 ότι ο πιο κοντινός κοινός πρόγονος του x και του y στο αρχικό δέντρο κυριαρχίας είναι ο καινούριος άμεσος κυρίαρχος κόμβος κάθε κόμβου που ανήκει στο σύνολο με τους επηρεασμένους κόμβους. Στη συνέχεια θα συμβολίζουμε με z τον κοντινότερο κοινό πρόγονο των x και y στο αρχικό δέντρο κυριαρχίας.

Ο προσαυξητικός αλγόριθμος πρέπει να εξετάσει αρχικά αν μετά την αναζήτηση κατά βάθος, του αρχικού γραφήματος, παράγεται ένα δέντρο ή ένα δάσος καθώς επίσης πρέπει να προσδιορίσει ποιοι κόμβοι είναι προσπελάσιμοι από την ρίζα. Αν και ο κόμβος x και ο κόμβος y είναι προσπελάσιμοι από την ρίζα τότε εκτελεί τις ακόλουθες ενέργειες:

Αρχικά σε κάθε κόμβο, με τη βοήθεια του αρχικού δέντρου κυριαρχίας, αντιστοιχεί μία τιμή σε έναν πίνακα level. Αυτή η τιμή δείχνει το επίπεδο στο οποίο βρίσκεται κάθε κόμβος στο δέντρο κυριαρχίας. Έτσι, η ρίζα βρίσκεται στο επίπεδο ένα, οι κόμβοι που έχουν άμεσο κυρίαρχο κόμβο τη ρίζα βρίσκονται στο επίπεδο δύο κοκ. Εν συνεχεία, ο αλγόριθμος κατασκευάζει το DJ-graph (Dominator Tree Edges-Join Edges graph). Το DJ-graph είναι το δέντρο κυριαρχίας αυξημένο με τις υπόλοιπες ακμές του γραφήματος. Δηλαδή, είναι το δέντρο κυριαρχίας στο οποίο έχει προστεθεί κάθε ακμή του γραφήματος η οποία δεν ενώνει τον άμεσο κυρίαρχο ενός κόμβου με τον ίδιο τον κόμβο. Οι δεντρικές ακμές αυτού του δέντρου αποτελούν τις D-edges του γραφήματος, ενώ οι υπόλοιπες αποτελούν τις J-edges. Το DJ-graph του γραφήματος στο σχήμα 3.1 παρουσιάζεται στο σχήμα 6.1. Η βασική λειτουργία του αλγορίθμου είναι να ενημερώσει το DJ-graph μετά την εισαγωγή της καινούριας ακμής. Αν η ενημέρωση είναι σωστή τότε ενημερώνεται σωστά και το δέντρο κυριαρχίας.



Σχήμα 6.1: DJ-graph.

Αφού έχει υπολογιστεί το DJ-graph εξετάζεται η νεοεισαχθείσα ακμή $(x \rightarrow y)$. Το σύνολο των κόμβων που επηρεάζονται, $domaffected(y)$, είναι οι κόμβοι w για τους οποίους ισχύει:

$$\text{domaffected(y)} = \{w \mid w \in (\{y\} \cup IDF(y))\ \kappa\alpha\iota\ w.level > z.level + 1\}$$

όπου το z είναι ο πιο κοντινός πρόγονος των x και y στο αρχικό δέντρο κυριαρχίας, το $w.level$ είναι ο αριθμός επιπέδου του w και το $z.level$ είναι ο αριθμός επιπέδου του z. Το $IDF(S)$ (Iterated Dominance Frontier), ενός συνόλου κόμβων S, ορίζεται ως το όριο της αύξουσας συνάρτησης [12]:

$$IDF_1(S) = DF(S)$$

$$IDF_{i+1}(S) = DF(S \cup IDF_i(S))$$

όπου

$$DF(S) = \bigcup_{x \in S} DF(x)$$

εδώ απλά μπορούμε να υποθέσουμε ότι το $IDF(x)$ αποτελεί το $IDF(\{x\})$, δηλαδή ενός συνόλου με ένα μόνο κόμβο.

Στον προσαυξητικό αλγόριθμο δεν υπολογίζουμε τα σύνολα των ορίων κυριαρχίας για όλους τους κόμβου για δύο λόγους. Πρώτον, γιατί υπάρχει ένας πολύ αποτελεσματικός αλγόριθμος ο οποίος υπολογίζει το σύνολο $IDF(y)$ σε γραμμικό χρόνο όταν γνωρίζουμε ότι η νεοεισαχθείσα ακμή είναι η ακμή $(x \rightarrow y)$. Δεύτερον, γιατί η ενημέρωση όλων των συνόλων με τα όρια κυριαρχίας θα χρειαζόταν επιπλέον χώρο για την αποθήκευση και επιπλέον χρόνο για τον υπολογισμό. Η πολυπλοκότητα χρόνου που απαιτείται για την ενημέρωση των συνόλων των ορίων κυριαρχίας μπορεί να είναι πολύ χειρότερη από ότι για τον υπολογισμό του $IDF(y)$ για ένα κόμβο y.

Τέλος αφού υπολογίστηκε και το σύνολο με τους κόμβους που επηρεάζονται μένει να κάνουμε την ενημέρωση του DJ-graph. Για κάθε κόμβο w διαγράφουμε την D-ακμή του και προσθέτουμε μία J-ακμή αν η διαγραφείσα ακμή υπάρχει στο αρχικό γράφημα. Στη συνέχεια, για κάθε κόμβο w προσθέτουμε μία D-ακμή από τον κόμβο z (κοντινότερος κοινός πρόγονος του x και του y στο αρχικό δέντρο κυριαρχίας). Το ενημερωμένο δέντρο κυριαρχίας αποτελείται από το DJ-graph αν αφαιρέσουμε όλες τις J-ακμές.

Αν ο y είναι προσπελάσιμος από την ρίζα και ο x δεν είναι, τότε ο αλγόριθμος δεν εκτελεί καμιά ενέργεια αφού κανένας άμεσος κυρίαρχος κόμβος δεν αλλάζει. Τελευταία περίπτωση είναι όταν ο x είναι προσπελάσιμος από την ρίζα ενώ ο y γίνεται προσπελάσιμος μόνο μετά από την εισαγωγή της ακμής $(x \rightarrow y)$. Σε αυτή την περίπτωση δημιουργούμε ένα DJ-graph που έχει ρίζα το y. Αφού το x γίνεται άμεσος κυρίαρχος του y, προσθέτουμε μία D-ακμή από το x στο y. Αυτό συμβάλει για να συνδέσουμε το νέο DJ-graph με το ήδη υπάρχον. Τέλος, για την ενημέρωση του DJ-graph, μεταχειριζόμαστε κάθε ακμή $(u \rightarrow v)$, για την οποία το u δεν είναι προσπελάσιμο από την ρίζα του δέντρου κυριαρχίας πριν την εισαγωγή της ακμής, ενώ το v είναι προσπελάσιμο ακόμα και πριν την εισαγωγή, σαν καινούριες ακμές.

Ο προσαυξητικός αλγόριθμος έχει πολυπλοκότητα χρόνου $O(\,|E_1|)$, όπου $|E_1|$ είναι ο αριθμός τον ακμών στο υπογράφημα του DJ-graph που έχει ρίζα το z, αν η νεοεισαχθείσα ακμή είναι η $(\,x \rightarrow y\,)$ και τα x και y είναι αρχικά προσπελάσιμα από την ρίζα.

6.4 Παραλλαγή του Προσαυξητικού Αλγορίθμου Sreedhar, Gao, Lee

Ο παραπάνω αλγόριθμος μετά την εισαγωγή μίας νέας ακμής πρέπει να ενημερώσει τους αριθμούς του πίνακα level, οι οποίοι άλλαξαν, για ένα σύνολο κόμβων. Αυτό το σύνολο αποτελείται από τους κόμβους οι οποίοι επηρεάστηκαν και από τους απογόνους αυτών στο δέντρο της κυριαρχίας. Ωστόσο, μία παραλλαγή αυτού του αλγορίθμου μας απαλλάσσει από την ανάγκη για ενημέρωση των αριθμών του πίνακα level. Το μόνο που χρειάζεται είναι να δώσουμε μία αρχική τιμή για κάθε κόμβο σε αυτόν, όταν τον πρωτοεισάγουμε στο δέντρο τη κυριαρχίας. Η παραλλαγή του αλγορίθμου εκτελεί τις ακόλουθες πράξεις:

Έστω ότι εισάγεται η ακμή $x \rightarrow y$ σε ένα γράφημα G. Υπολογίζουμε το DJ-graph του αρχικού γραφήματος G και εξετάζουμε την νεοεισαχθείσα ακμή $x \rightarrow y$. Αν και ο κόμβος x και ο κόμβος y δεν είναι προσπελάσιμοι από την ρίζα του δέντρου της κυριαρχίας ή αν ο y είναι αρχικά προσπελάσιμος ενώ ο x δεν είναι, τότε δεν εκτελείται καμία ενέργεια. Αν και ο κόμβος x και ο κόμβος y είναι αρχικά προσπελάσιμοι από την ρίζα, τότε οι κόμβοι που επηρεάζονται, domaffected(y), είναι όλοι οι κόμβοι w για τους οποίους ισχύει ότι

$$\text{domaffected(y)} = \{w \mid w \in (\{y\} \cup IDF(y))\ και\ idom[w].level > z.level\}$$

όπου $idom[w]$ είναι ο άμεσος κυρίαρχος του w και z είναι ο κοντινότερος κοινός πρόγονος των x και y.

Αν ο x είναι αρχικά προσπελάσιμος από την ρίζα ενώ ο y γίνεται προσπελάσιμος μόνο μετά την εισαγωγή της ακμής $x \rightarrow y$, τότε θέτουμε το $idom(y) = x$ καθώς επίσης και το $level(y) = level(x) + 1$ (αφού κάθε κόμβος βρίσκεται ένα επίπεδο

κάτω από τον άμεσο κυρίαρχό του στο δέντρο της κυριαρχίας). Δημιουργούμε ένα DJ-graph με ρίζα τον κόμβο y και εν συνεχεία τον ενώνουμε στο αρχικό DJ-graph. Τέλος επεξεργαζόμαστε κάθε ακμή της μορφής $w \rightarrow v$, για την οποία ισχύει ότι το w αρχικά δεν ήταν προσπελάσιμο από την ρίζα του δέντρου της κυριαρχίας ενώ το v ήταν, σαν καινούρια ακμή. Αξίζει να αναφέρουμε ότι η τελευταία περίπτωση είναι και η μοναδική στην οποία κάνουμε κάποια καταχώρηση σε ένα στοιχείο του πίνακα level, αφού το μόνο που μας ενδιαφέρει να γνωρίζουμε είναι το επίπεδο ενός κόμβου όταν αυτός γίνεται για πρώτη φορά προσπελάσιμος από την κορυφή.

Το γεγονός ότι μελετάμε μόνο το προσαυξητικό πρόβλημα των κόμβων κυριαρχίας, μας δίνει την δυνατότητα να μην ενημερώνουμε τον πίνακα level μετά από κάθε εισαγωγή μίας νέας ακμής. Όταν εισάγουμε μία νέα ακμή στο γράφημα τότε είτε δεν θα συμβεί καμία αλλαγή στο δέντρο κυριαρχίας είτε κάποιοι κόμβοι θα αναρριχηθούν σε αυτό. Άρα για ένα τυχαίο κόμβο a ισχύει ότι $a.level \geq a.level_{true}$, όπου $a.level$ είναι η τιμή επιπέδου που έχει ο κόμβος a ενώ $a.level_{true}$ είναι η πραγματική τιμή επιπέδου που θα είχε αν ενημερώναμε τον πίνακα level μετά από κάθε νέα εισαγωγή.

Εφόσον ένας κόμβος μπορεί μόνο να αναρριχηθεί στο δέντρο κυριαρχίας τότε πάντα θα ισχύει ότι $idom[a].level < a.level$. Αυτό αποδεικνύει ότι μπορούμε να υπολογίσουμε σωστά τον κοντινότερο κοινό πρόγονο χρησιμοποιώντας τον πίνακα level. Αυτό οφείλεται και στο ότι θέτουμε το $a.level = idom[a].level + 1$, όταν ο κόμβος a γίνεται αρχικά προσπελάσιμος από την κορυφή. Αυτή η ιδιότητα μας εφοδιάζει επίσης με ένα κριτήριο για να ελέγξουμε αν ένας κόμβος επηρεάζεται ή δεν επηρεάζεται από την εισαγωγή της νέας ακμής.

Ένας κόμβος w για να ανήκει στο $IDF(y)$ θα πρέπει να συνδέεται με μία j – ακμή με ένα κόμβο του υποδέντρου του y στο δέντρο της κυριαρχίας. Αν η $v \rightarrow w$ αποτελεί μία τέτοια j – ακμή, από τον τρόπο κατασκευής του DJ – graph, θα πρέπει να ισχύει ότι ο $idom[w]$ είναι ο κοντινότερος κοινός πρόγονος των w και v στο δέντρο της κυριαρχίας. Κατ’ επέκταση αν $w \in IDF(y)$ τότε ο $idom[w]$ είναι ο κοντινότερος κοινός πρόγονος των w και y στο δέντρο της κυριαρχίας. Από αυτό προκύπτει ότι ο κοντινότερος κοινός πρόγονος, z, των x και y, όταν εισάγουμε την ακμή $x \rightarrow y$, και ο $idom[w]$, όπου $w \in IDF(y)$, είτε είναι ο ίδιος κόμβος, είτε ο z ανήκει στο υποδέντρο

του $idom[w]$, είτε ο $idom[w]$ ανήκει στο υποδέντρο του z. Άρα όταν ελέγχουμε αν το $idom[w].level > z.level$ είμαστε σίγουροι ότι υπολογίζουμε το σωστό σύνολο των κόμβων οι οποίοι επηρεάζονται από την προσθήκη της ακμής $x \rightarrow y$, ασχέτος με το πόσες ακμές προσθέσαμε πριν (δηλαδή ασχέτως με το γεγονός ότι οι τιμές του πίνακα level δεν είναι οι πραγματικές).

6.5 Πειραματικά Αποτελέσματα

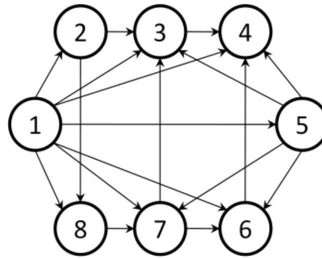
Τα πειραματικά αποτελέσματα, που συμπεριλαμβάνονται στην παρούσα διπλωματική, αποτελούνται από μετρήσεις σε προγράμματα τα οποία έχουν όλα υλοποιηθεί σε γλώσσα C++. Τα πειράματα που εκτελούνται είναι δύο. Στο πρώτο πείραμα συγκρίνεται η αποδοτικότητα των δομών δεδομένων οι οποίες αναπαριστούν ένα γράφημα, ενώ στο δεύτερο πείραμα συγκρίνεται η αποδοτικότητα των αλγορίθμων οι οποίοι υπολογίζουν το προσαυξητικό πρόβλημα των κόμβων κυριαρχίας. Έχει γίνει προσπάθεια ώστε τα αποτελέσματα και των δύο πειραμάτων να είναι όσο το δυνατόν πιο αντικειμενικά και να εξαρτώνται από την εκάστοτε δομή δεδομένων και τον εκάστοτε αλγόριθμο και όχι από τις ιδιομορφίες των γραφημάτων, τα οποία αποτελούν το δείγμα υπολογισμού των πειραμάτων. Τα γραφήματα με την σειρά τους έχουν κατασκευαστεί με εντελώς τυχαίο τρόπο, προσδιορίζοντας μόνο τον αριθμό των κόμβων και τον αριθμό των ακμών. Στη συνέχεια αυτής της ενότητας παρουσιάζονται τα αποτελέσματα από τα πειράματα καθώς επίσης και τα συμπεράσματα που προέκυψαν.

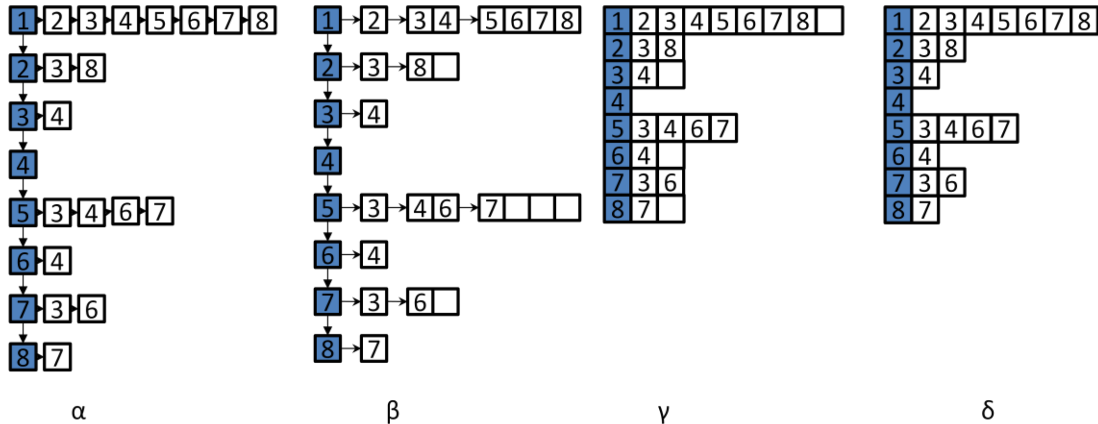
6.5.1 Σύγκριση των Δομών Αναπαράστασης Ενός Γραφήματος

Εφόσον όλες οι τεχνικές βελτιστοποίησης προγραμμάτων βασίζονται στο γράφημα ροής του εκάστοτε προγράμματος, γι' αυτό κρίθηκε σκόπιμο να εξεταστεί πια από τις δομές δεδομένων που παρουσιάζονται στην ενότητα 1.3 είναι πιο αποδοτική. Το πρώτο πείραμα συγκρίνει αυτές τις δομές δεδομένων, έτσι ώστε να επιλέξουμε την πιο αποδοτική με σκοπό να την χρησιμοποιήσουμε στο δεύτερο πείραμα για να αναπαραστήσουμε το γράφημα. Εκείνο που μας ενδιαφέρει περισσότερο να ελέγξουμε είναι ο χρόνος ο οποίος απαιτείται για να διαπεράσει η δομή δεδομένων όλες τις ακμές από όλους τους κόμβους. Ωστόσο, για να είναι τα αποτελέσματα πιο αξιοκρατικά συγκρίνουμε επίσης και τον χρόνο που χρειάστηκε η εκάστοτε δομή δεδομένων για να δημιουργήσει το γράφημα. Τέλος επειδή μερικές δομές

δεδομένων, όπως αναφέραμε και στην αντίστοιχη ενότητα, σπαταλούν κάποιο μέρος της μνήμης, δεσμεύοντας χώρο χωρίς να χρειάζεται πρακτικά, στις μετρήσεις μας προσδιορίζουμε το μέγεθος αυτής της σπατάλης.

Οι δομές δεδομένων που συγκρίνονται είναι οι λίστες γειτνίασης, οι συνδεδεμένοι πίνακες γειτνίασης και δύο υλοποιήσεις των δυναμικών πινάκων γειτνίασης. Ο λόγος που εξετάζουμε μόνο αυτές τις δομές δεδομένων και όχι τις υπόλοιπες (πίνακας ακμών και πίνακας γειτνίασης) είναι ότι οι άλλες δύο δεν είναι κατάλληλες για την αναπαράσταση ενός δυναμικού γραφήματος. Ένας ακόμη λόγος για τον οποίο δεν συμπεριλαμβάνουμε τον πίνακα γειτνίασης στο πρώτο πείραμα είναι ότι χρειάζεται υπερβολικά μεγάλο χώρο στην μνήμη. Πιο συγκεκριμένα, χρειαζόμαστε τετραγωνικό στον αριθμό των κόμβων χώρο στην μνήμη. Όσον αφορά τις δύο υλοποιήσεις των δυναμικών πινάκων γειτνίασης, η πρώτη δεν χρησιμοποιεί καμία έτοιμη συνάρτηση. Όλες οι δομές (structures) και όλες οι συναρτήσεις που χρησιμοποιούνται ορίζονται μέσα στο πρόγραμμα. Εν αντιθέσει, η δεύτερη κάνει χρήση μίας έτοιμης δομής δεδομένων, της vector, η οποία δημιουργεί ένα δυναμικό πίνακα στον οποίο μπορούμε να εισάγουμε τιμές κατά βούληση όταν εκτελείται το πρόγραμμα. Μία δισδιάστατη μεταβλητή της δομής vector είναι κατάλληλη για την αναπαράσταση ενός δυναμικού, ως προς τους κόμβους και ως προς τις ακμές, γραφήματος. Αυτή η συνάρτηση βρίσκεται στην βιβλιοθήκη <vector.h>.





Σχήμα 6.2: Αναπαράσταση του γραφήματος (επάνω) α) με λίστες γειτνίασης, β) συνδεδεμένους πίνακες γειτνίασης, γ) με δυναμικούς πίνακες γειτνίασης και γ) με δυναμικούς πίνακες γειτνίασης (υλοποίηση με vector).

Στο σχήμα 6.2 υπάρχει μία εικονική αναπαράσταση των τεσσάρων δομών δεδομένων, για την αναπαράσταση ενός γραφήματος, τις οποίες συγκρίνουμε. Το σχήμα αυτό βοηθάει τον αναγνώστη ώστε να πάρει μία οπτική αίσθηση του πως αποθηκεύεται η πληροφορία στην εκάστοτε δομή δεδομένων. Από το σχήμα φαίνεται ότι μόνο οι δύο πρώτες δομές δεδομένων χρησιμοποιούν μεταβάσεις μέσω δεικτών ενώ οι υπόλοιπες χρησιμοποιούν προσπελάσεις σε πίνακες. Επίσης από το σχήμα φαίνεται η μνήμη που σπαταλάτε από την δομή των συνδεδεμένων πινάκων και από την δομή των δυναμικών πινάκων στην πρώτη υλοποίηση. Αξίζει να σημειώσουμε ότι η δεύτερη υλοποίηση των δυναμικών πινάκων, δηλαδή η υλοποίηση με vector, δεν σπαταλά επιπλέον μνήμη.

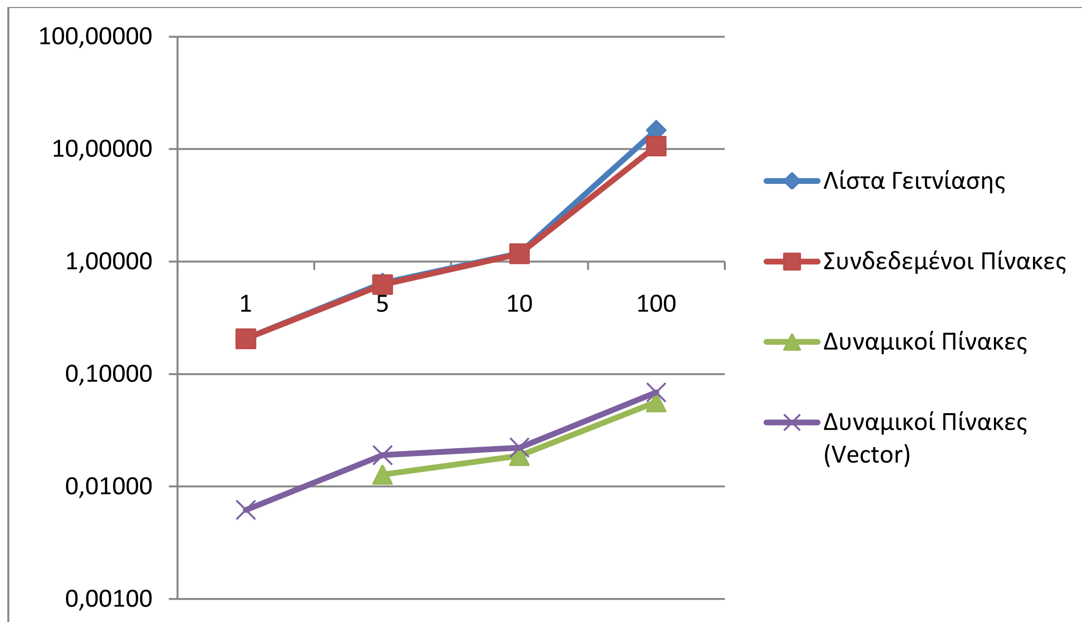
Το δείγμα από το οποίο λάβαμε τις μετρήσεις αποτελείται από είκοσι γραφήματα των 10.000 κόμβων και μεταβλητό αριθμό, τυχαίων, ακμών. Οι μετρήσεις έγιναν για 10.000, για 50.000, για 100.000 και για 1.000.000 ακμές. Για να πετύχουμε την πολυπόθητη αντικειμενικότητα, η οποία είναι ο στόχος μας όπως προείπαμε, σε κάθε

περίπτωση πήραμε τις μετρήσεις από πέντε διαφορετικά γραφήματα και στο πίνακα παρουσιάζουμε τον μέσο όρο αυτών των μετρήσεων.

		Χρόνοι Κατασκευής(sec)			
ΚΟΜΒΟΙ	**ΑΚΜΕΣ**	Λίστα Γειτνίασης	Συνδεδεμένοι Πίνακες	Δυναμικοί πίνακες	Δυναμικοί πίνακες (vector)
10000	10000	0,20620	0,20580	0,00000	0,00620
10000	50000	0,64375	0,62320	0,01280	0,01900
10000	100000	1,18240	1,17320	0,01880	0,02220
10000	1000000	14,72340	10,62640	0,05640	0,06860

Σχήμα 6.3: Σύγκριση των δομών δεδομένων στον χρόνο δημιουργίας του γραφήματος.

Όπως φαίνεται από τις μετρήσεις, το γράφημα δημιουργείται πιο γρήγορα στους δυναμικούς πίνακες, έχοντας ελαφρός πιο γρήγορη εκτέλεση η πρώτη υλοποίηση, η οποία δεν χρησιμοποιεί έτοιμες συναρτήσεις, για όλες τις περιπτώσεις των γραφημάτων. Αν και, όπως είπαμε στο αντίστοιχο κεφάλαιο, στη χειρότερη περίπτωση, στην πρώτη υλοποίηση των δυναμικών πινάκων, η εισαγωγή μίας νέας ακμής έχει πολυπλοκότητα στο χρόνο $O(n)$ παρόλα αυτά σε κάθε άλλη περίπτωση η νέα ακμή εισάγεται σε χρόνο $O(1)$, οπότε υπάρχει μία αντιστάθμιση. Οι συνδεδεμένοι πίνακες δημιουργούν το γράφημα πιο γρήγορα σε σχέση με τη συνδεδεμένη λίστα επειδή οι εισαγωγές γίνονται και σε πίνακα, εκτός από κόμβους συνδεδεμένης λίστας. Αυτό το γεγονός καθιστά αυτή τη δομή δεδομένων μέχρι και τέσσερα δευτερόλεπτα πιο γρήγορη, σε σχέση με τη λίστα γειτνίαση, όταν έχουμε ένα γράφημα με 1.000.000 ακμές. Τέλος στη λίστα γειτνίασης η εισαγωγή κάθε ακμής έχει σταθερή πολυπλοκότητα στο χρόνο, επειδή για κάθε εισαγωγή εκτελείται ο ίδιος αριθμός πράξεων. Ωστόσο, οι πράξει που εκτελούνται για κάθε εισερχόμενη ακμή είναι τόσες πολλές ώστε σαν αποτέλεσμα η λίστα γειτνίασης να είναι πολύ αργή σε σύγκριση με της άλλες δομές δεδομένων.



Σχήμα 6.4: Γραφική παράσταση των χρόνων δημιουργίας ενός γραφήματος (η κλίμακα του άξονας y είναι λογαριθμική)

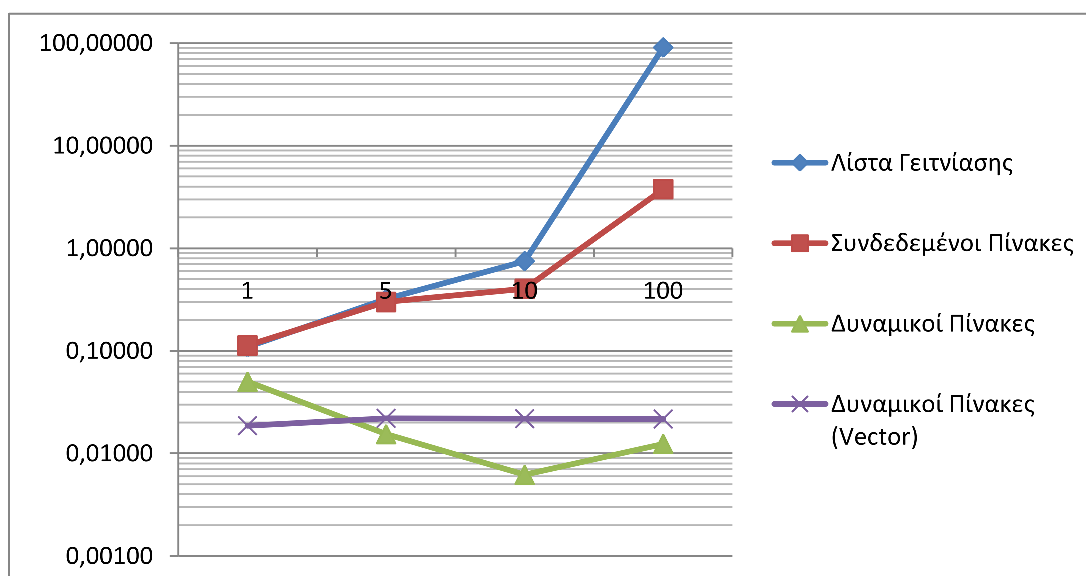
Το σχήμα 6.4 παρουσιάζει με τη μορφή γραφικής παράστασης τα αποτελέσματα του σχήματος 6.3. Όπως φαίνεται από το σχήμα, η ψαλίδα στον χρόνο δημιουργίας του γραφήματος, μεταξύ της λίστας γειτνίασης και του συνδεδεμένου πίνακα ανοίγει όσο αυξάνονται οι ακμές στο γράφημα. Αντίθετα, η ψαλίδα μεταξύ των δύο υλοποιήσεων των δυναμικών πινάκων κλείνει καθώς οι ακμές του γραφήματος αυξάνονται.

		Χρόνοι Διαπεράσεις(10.000 επαναλήψεις)(sec)			
ΚΟΜΒΟΙ	**ΑΚΜΕΣ**	Λίστα Γειτνίασης	Συνδεδεμένοι Πίνακες	Δυναμικοί πίνακες	Δυναμικοί πίνακες (vector)
10000	10000	0,10900	0,11260	0,05000	0,01860
10000	50000	0,32100	0,29940	0,01540	0,02200
10000	100000	0,74880	0,40240	0,00620	0,02180
10000	1000000	90,78920	3,77220	0,01240	0,02160

Σχήμα 6.5: Σύγκριση των δομών δεδομένων στον χρόνο προσπέλασης των ακμών ενός γραφήματος.

Στο δεύτερο σκέλος του πειράματος μετρήσαμε τον χρόνο για την προσπέλαση όλων των ακμών, του γραφήματος, για κάθε δομή δεδομένων. Για να έχουν νόημα οι χρόνοι που μετρήσαμε η κάθε δομή δεδομένων προσπέλασε όλες τις ακμές 10.000 φορές.

Από το σχήμα 6.5 φαίνεται πάλι ότι οι δυναμικοί πίνακες εκτελούν την πράξη της προσπέλασης πιο γρήγορα, με την πρώτη υλοποίηση να είναι πάλι ελαφρώς πιο γρήγορη. Αυτή η ταχύτητα στην προσπέλαση οφείλεται στο ότι μετακινούμαστε μόνο μέσα σε πίνακα σε αντίθεση με τη λίστα γειτνίασης που μετακινούμαστε μέσα σε κόμβους, οι οποίοι είναι συνδεδεμένοι με δείκτες, και τους συνδεδεμένους πίνακες όπου μετακινούμαστε και μέσα σε κόμβους, οι οποίοι είναι συνδεδεμένοι με δείκτες, και μέσα σε πίνακες.



Σχήμα 6.6: Γραφική παράσταση των χρόνων προσπέλασης των ακμών (η κλίμακα του άξονας y είναι λογαριθμική)

Στη γραφική αναπαράσταση φαίνεται ξανά ότι όσο αυξάνουν οι ακμές τόσο αυξάνεται και η ψαλίδα μεταξύ των χρόνων της λίστας γειτνίασης και των συνδεδεμένων πινάκων. Από την άλλη, η ψαλίδα, στους χρόνους προσπέλασης, μεταξύ των δύο υλοποιήσεων των δυναμικών πινάκων κλείνει καθώς αυξάνονται οι ακμές του γραφήματος. Αξίζει να εξηγήσουμε το παράδοξο που εμφανίζεται στους χρόνους

προσπέλασης της πρώτης υλοποίησης των δυναμικών πινάκων. Στη γραφική παράσταση φαίνεται ότι ο χρόνος προσπέλασης των ακμών, του γραφήματος, πέφτει όσο αυτές αυξάνονται. Αυτό οφείλεται στο ότι ο μεταγλωττιστής στον οποίο εκτελέσαμε τα πειράματα υποστήριζε τις τεχνικές βελτιστοποίησης που περιγράφουμε σε αυτή τη διπλωματική. Έτσι, παρουσιάζεται ένα πρακτικό παράδειγμα του πόσο χρήσιμες είναι οι τεχνικές που παρουσιάζει αυτή η εργασία.

		Σπατάλη Μνήμης (byte)	
ΚΟΜΒΟΙ	**ΑΚΜΕΣ**	Συνδεδεμένοι Πίνακες	Δυναμικοί πίνακες
10000	10000	9502,40	17555,20
10000	50000	77102,40	61422,40
10000	100000	162380,00	145827,20
10000	1000000	1512844,80	1509376,00

Σχήμα 6.5: Σύγκριση των δομών δεδομένων στην σπατάλη μνήμης.

Στο τελευταίο σκέλος του πρώτου πειράματος παρουσιάζονται τα αποτελέσματα από την σπατάλη μνήμης στις δομές δεδομένων (σχήμα 6.5). Από τις τιμές του πίνακα προκύπτει ότι η σπατάλη μνήμης είναι σχεδόν ίδια στους συνδεδεμένους πίνακες και στην πρώτη υλοποίηση των δυναμικών πινάκων και είναι ανάλογη στο πλήθος των ακμών του γραφήματος.

Μετά την εκτέλεση του πρώτου πειράματος καταλήξαμε στο συμπέρασμα ότι οι δυναμικοί πίνακες αποτελούν μακράν την καλύτερη δομή δεδομένων για την αναπαράσταση ενός γραφήματος. Ο λόγος που θα χρησιμοποιήσουμε την δεύτερη υλοποίηση των δυναμικών πινάκων στο δεύτερο πείραμα, δηλαδή εκείνη των vector, οφείλεται στο ότι είναι πιο απλή στην χρήση. Το γεγονός ότι οι χρόνοι της πρώτης υλοποίησης είναι καλύτεροι σε σχέση με αυτούς της δεύτερης μπορεί να θεωρηθεί αμελητέο.

6.5.2 Σύγκριση των Αλγορίθμων Υπολογισμού του Προσαυξητικού Προβλήματος των Κόμβων Κυριαρχίας

Το δεύτερο πείραμα χωρίζεται σε δύο σκέλη. Στο πρώτο σκέλος μετράμε τις πράξεις που εκτελούν οι προσαυξητικοί αλγόριθμοι για την εύρεση των κόμβων κυριαρχίας σε ένα στατικό γράφημα. Εν συνεχεία, στο δεύτερο σκέλος προσθέτουμε μία ακμή στο αρχικό γράφημα και προσδιορίζουμε τις πράξεις που εκτελούν για να υπολογίσουν τους νέους κόμβους κυριαρχίας. Για να είναι τα αποτελέσματα αντικειμενικά οι αλγόριθμοι δοκιμάζονται σε ίδια γραφήματα, στο πρώτο σκέλος, ενώ στο δεύτερο σκέλος εισάγουμε σε αυτά την ίδια ακμή.

Οι αλγόριθμοι που εξετάζονται είναι οι ακόλουθοι:

1. **Inciter1:** Αποτελεί μία προσαυξητική έκδοση του επαναληπτικού αλγορίθμου, ο οποίος εκτελείται από την αρχή όταν εισάγεται μία καινούρια ακμή $x \rightarrow y$. Το μόνο που ελέγχει ο αλγόριθμος είναι αν το x ήταν αρχικά προσπελάσιμο από τη ρίζα του γραφήματος.
2. **Inciter2:** Αποτελεί μία επέκταση του παραπάνω αλγορίθμου, στον οποίο όταν εισάγεται μία νέα ακμή εκτελούνται τροποποιήσεις στο ήδη υπάρχον δέντρο της κυριαρχίας.
3. **Inciter3:** Αποτελεί επέκταση του incriter2, στον οποίο μετά την εισαγωγή μίας νέας ακμής εκτελείται αναζήτηση κατά βάθος μόνο για τους κόμβους οι οποίοι ήταν αρχικά μη προσπελάσιμοι από την ρίζα του δέντρου κυριαρχίας. Έτσι κατασκευάζει μία ψευδό – reverse – postororder αρίθμηση με την οποία επεξεργάζονται οι κόμβοι.
4. **Inciter4:** Εκτελεί τις ίδιες ενέργειες με τον inciter3 μόνο που αυτός κατασκευάζει το δέντρο της κυριαρχίας με ρίζα το y, όταν εισέρχεται η ακμή $x \rightarrow y$, και εν συνεχεία εκτελεί τον επαναληπτικό αλγόριθμο.
5. **Incslt1:** Αποτελεί μία προσαυξητική έκδοση του αλγορίθμου Lengauer-Tarjan, ο οποίος εκτελείται αν το x είναι προσπελάσιμο από την ρίζα του δέντρου της κυριαρχίας, όταν εισάγουμε την ακμή $x \rightarrow y$.
6. **Incslt2:** Αποτελεί επέκταση του αλγορίθμου incslt1, στον οποίο η αναζήτηση κατά βάθος εκτελείται μόνο αν η νέα ακμή αναιρεί το προηγούμενο δέντρο της αναζήτησης κατά βάθος. Το δέντρο της αναζήτησης κατά βάθος αναιρείται όταν εισέρχεται μία ακμή διασταυρώσεως, η οποία έχει φορά από τα αριστερά προς τα δεξιά.

7. **Incslt3:** Αποτελεί επέκταση του αλγορίθμου incslt2, στον οποίο μετά την εισαγωγή της νέας ακμής, αν διατηρείται το προηγούμενο δέντρο της αναζήτησης κατά βάθος, εξετάζεται για ποιους κόμβους χρειάζεται να ξανά-υπολογίσουμε τα semi-dominators. Αυτοί οι κόμβοι είναι οι κόμβοι οι οποίοι έχουν αριθμό προ-διάταξης μικρότερο από εκείνο του y.
8. **Incnca:** Υπολογίζει όπως και ο incslt3 τα semi-dominator και εκτελεί ένα πέρασμα του επαναληπτικού αλγορίθμου ως εξής: το αρχικό δέντρο της κυριαρχίας είναι το δέντρο της καθοδικής διερεύνησης και επεξεργάζονται μόνο οι ακμές από $sdom(v) \rightarrow v$ για κάθε κορυφή v, όπου $sdom(v)$ είναι ο semi-dominator του v (αυτές οι ακμές μπορεί να ανήκουν ή να μην ανήκουν στο γράφημα).
9. **SGL_algorithm1:** Αποτελεί τον προσαυξητικό αλγόριθμο των Sreedhar, Gao, Lee που μελετήσαμε αναλυτικά στην παράγραφο 6.3.
10. **SGL_algorithm2**: Αποτελεί παραλλαγή του προσαυξητικού αλγορίθμου των Sreedhar, Gao, Lee που μελετήσαμε αναλυτικά στην παράγραφο 6.4.

Η αναλυτική περιγραφή των αλγορίθμων θα υπάρχει σε ένα άρθρο το οποίο πρόκειται να δημοσιευθεί.

Graph		inciter1		inciter2		inciter3		inciter4	
Κόμβοι	Ακμές	comparisons	iterations	comparisons	iterations	comparisons	iterations	comparisons	iterations
1000	2000	15124747	2079	6149551	873	6174243	963	5892670	997
1000	3000	60886725	5338	24274698	2093	23000134	2226	22196885	2237
1000	4000	119769178	8254	50170466	3228	45864536	3369	44881504	3374
1000	5000	196766727	11227	82003745	4276	733782206	4417	72342513	4420
1000	6000	291406130	14224	120821439	5305	106825961	5448	105730569	5450
1000	7000	402576922	17224	166422453	6321	145958836	6464	144863459	6466
1000	8000	529992185	202214	218631048	7327	190716676	7470	189621299	7472
2000	4000	50528787	3012	20286063	1271	20126970	1379	18824790	1380
2000	5000	123823882	6037	50384855	2473	48307960	2624	46025380	2613
2000	6000	218044002	8970	91026598	3671	85510874	3858	82162967	3830
2000	7000	325165676	11802	137922623	4793	127161487	4992	123376691	4958
2000	8000	455094479	14709	191961080	5883	174352421	6085	170313497	6046
2000	9000	604377234	17658	2531199601	6959	227653825	7165	223370394	7121
2000	10000	769799516	20601	320334492	8000	285700054	8208	281306397	8164

Σχήμα 6.6: Αποτελέσματα του πρώτου μέρους του πειράματος για τους επαναληπτικούς αλγορίθμους.

Graph		inciter1		inciter2		inciter3		inciter4	
Κόμβοι	**Ακμές**	**comparisons**	**iterations**	**comparisons**	**iterations**	**comparisons**	**iterations**	**comparisons**	**iterations**
1000	2000	40901	4	20873	2	19525	2	19525	2
1000	3000	51210	3	19551	1	16845	1	16845	1
1000	4000	69028	3	50182	2	46427	2	46427	2
1000	5000	86091	3	63538	2	58744	2	58744	2
1000	6000	102983	3	77034	2	71112	2	71112	2
1000	7000	119209	3	90007	2	83023	2	83023	2
1000	8000	135695	3	55462	1	47494	1	47494	1
2000	4000	86041	4	41032	2	38266	2	38266	2
2000	5000	111476	4	83372	3	79722	3	79722	3
2000	6000	103669	3	72522	2	67178	2	67178	2
2000	7000	123303	3	86656	2	80066	2	80066	2
2000	8000	145529	3	100272	2	92601	2	92601	2
2000	9000	159835	3	113899	2	104994	2	104994	2
2000	10000	177614	3	127471	2	117478	2	117478	2

Σχήμα 6.7: Αποτελέσματα του δεύτερου μέρους του πειράματος για τους επαναληπτικούς αλγορίθμους.

Στο σχήμα 6.6 παρουσιάζονται τα αποτελέσματα του πρώτου μέρους του πειράματος, ενώ στο σχήμα 6.7 παρουσιάζονται τα αποτελέσματα του δεύτερου μέρους του πειράματος για τους επαναληπτικούς αλγορίθμους. Οι τιμές στα κελιά comparisons αποτελούν τον αριθμό συγκρίσεων που εκτέλεσε ο αλγόριθμος, ενώ οι τιμές των κελιών iterations αποτελούν τον αριθμό των επαναλήψεων για να τερματίσει ο αλγόριθμος σε κάθε περίπτωση. Τα αποτελέσματα δείχνουν ότι ο inciter4 εκτελεί τις λιγότερες συγκρίσεις και τις λιγότερες επαναλήψεις σε σχέση με τους υπόλοιπους.

Graph		incslt1	incslt2	incslt3	incsnca
Κόμβοι	Ακμές	comparisons	comparisons	comparisons	comparisons
1000	2000	4916991	4633211	3728647	3130912
1000	3000	22648061	20953642	15294625	12655020
1000	4000	51122285	46753277	32313702	26708703
1000	5000	87915115	79573526	52760784	43405915
1000	6000	132338524	118972535	76228010	62909071
1000	7000	183869683	164397341	103160171	85514879
1000	8000	242125705	215503383	133176847	110720322
2000	4000	16802813	15943717	12801395	10514232
2000	5000	45587365	42728827	32334089	26672475
2000	6000	88624659	82239466	59635752	49285070
2000	7000	141900079	130738526	90702468	74676654
2000	8000	205688557	188589867	127884616	104946111
2000	9000	278598538	254392757	169052509	138832281
2000	10000	359396367	326922790	213098506	174898160

Σχήμα 6.8: Αποτελέσματα του πρώτου μέρους του πειράματος για τους αλγορίθμους Lengauer-Tarjan.

Graph		incslt1	incslt2	incslt3	incsnca
Κόμβοι	Ακμές	comparisons	comparisons	comparisons	comparisons
1000	2000	14760	14760	14763	12696
1000	3000	24687	21858	13078	11138
1000	4000	33679	29761	27846	23098
1000	5000	40539	40539	40542	38469
1000	6000	48172	42174	40296	35913
1000	7000	55048	55048	55051	53471
1000	8000	61571	53570	48500	41471
2000	4000	29246	26060	22079	16020
2000	5000	40299	35888	27943	17298
2000	6000	51496	45852	23702	12221
2000	7000	61326	61326	61329	56140
2000	8000	69594	61805	41802	38625
2000	9000	78206	69348	63599	53767
2000	10000	86531	86531	86534	81800

Σχήμα 6.9: Αποτελέσματα του δεύτερου μέρους του πειράματος για τους αλγορίθμους Lengauer-Tarjan.

Στα σχήματα 6.8 και 6.9 παρουσιάζονται τα αποτελέσματα των δύο μερών του πειράματος για τους αλγορίθμους Lengauer-Tarjan καθώς επίσης και για τον αλγόριθμο incsnca. Τα αποτελέσματα δείχνουν ότι ο incsnca εκτελεί τις λιγότερες συγκρίσεις σε κάθε περίπτωση, ακόμα και από τον incriter4.

Graph		SGL_algorithm1			SGL_algorithm2		
Κόμβοι	Ακμές	time elapsed	affected	processed	time elapsed	affected	processed
1000	2000	0,436	526	509171	0,515	524	509183
1000	3000	8,002	822	3846194	8,564	820	3846205
1000	4000	26,972	964	9677213	29,375	962	9677224
1000	5000	53,304	1010	17114502	56,705	1008	17114513
1000	6000	92,851	1034	27274125	98,546	1032	27274136
1000	7000	139,526	1049	39474384	150,791	1047	39474395
1000	8000	189,822	1055	52971212	205,5	1053	52971223
2000	4000	1,279	3295	1955863	1,731	3295	1955863
2000	5000	3,026	3728	3631677	3,323	3731	3612581
2000	6000	26,551	3994	8823757	28,002	3995	8804311
2000	7000	63,757	4163	15726111	67,798	4164	15706665
2000	8000	115,44	4263	24431835	121,79	4264	24412389
2000	9000	173,91	4342	34239957	183,472	4343	34220511
2000	10000	272,158	4393	47017174	285,123	4394	46997728

Σχήμα 6.10: Αποτελέσματα του πρώτου μέρους του πειράματος για τους αλγορίθμους Shreedhar, Gao, Lee.

Graph		SGL_algorithm1			SGL_algorithm2		
Κόμβοι	Ακμές	time elapsed	affected	processed	time elapsed	affected	processed
1000	2000	0	1	1	0	1	1
1000	3000	0,016	0	6080	0,016	0	6080
1000	4000	0	1	7070	0,031	1	7070
1000	5000	0,062	1	9064	0,032	1	9064
1000	6000	0,063	1	11092	0,047	1	11092
1000	7000	0,031	1	13013	0,031	1	13013
1000	8000	0,047	0	13989	0,078	0	13989
2000	4000	0	1	1	0	1	1
2000	5000	0,016	1	5429	0,016	1	5419
2000	6000	0,047	1	8315	0,031	1	8315
2000	7000	0,047	1	9196	0,063	1	9196
2000	8000	0,062	1	10145	0,047	1	10145
2000	9000	0,047	1	11092	0,063	1	11092
2000	10000	0,125	1	14033	0,109	1	14033

Σχήμα 6.11: Αποτελέσματα του δεύτερου μέρους του πειράματος για τους αλγορίθμους Shreedhar, Gao, Lee.

Τέλος στα σχήματα 6.10 και 6.11 παρουσιάζονται τα αποτελέσματα από τα δύο μέρη του πειράματος για τους αλγορίθμους των Shreedhar, Gao, Lee. Εκείνο που μετρήσαμε σε αυτούς τους αλγορίθμους είναι ο χρόνος εκτέλεσης (time elapsed), οι κόμβοι οι οποίοι επηρεάστηκαν (affected) και οι κόμβοι οι οποίοι προσπελάστηκαν μετά την εισαγωγή μίας ακμής (processed). Ο λόγος που μετράμε τον χρόνο εκτέλεσης είναι για να συγκρίνουμε τους δύο αλγορίθμους μεταξύ τους. Οι μετρήσεις δείχνουν ότι ο χρόνος εκτέλεσης του αλγορίθμου SGL_algorithm2 είναι χειρότερος σε σχέση με τον αλγόριθμο SGL_algorithm1 άσχετα από γεγονός ότι δεν ενημερώνει τα επίπεδα των κόμβων μετά την εισαγωγή μίας νέας ακμής. Αυτό οφείλεται στο ότι το μόνο που κερδίζουμε με τον αλγόριθμο SGL_algorithm2 είναι μία πράξη πρόσθεσης για κάθε κόμβο που ανήκει στο σύνολο των κόμβων που προσπελάστηκαν (processed). Ωστόσο η συνθήκη που χρησιμοποιεί ο αλγόριθμος για τον έλεγχο αν ένας κόμβος ανήκει στο σύνολο με τους κόμβους που επηρεάστηκαν (affected) αποτελεί πιο ακριβή, σε χρόνο, πράξη σε σχέση με εκείνη του αλγορίθμου SGL_algorithm1. Οι δύο αλγόριθμοι επεξεργάζονται σχεδόν τον ίδιο αριθμό κόμβων

και είναι πολύ λίγοι σε σχέση με τους κόμβους που επεξεργάζονται οι άλλοι αλγόριθμοι.

Σημειώνουμε ότι το πείραμα που εκτελείται στα πλαίσια αυτής της διπλωματική είναι ένα προκαταρκτικό πείραμα. Το βασικό πείραμα θα εκτελεστεί και θα δημοσιευθεί σε εργασία η οποία προετοιμάζεται.

Βιβλιογραφία

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman, «Compilers. Principles, techniques and Tools», Prentice Hall, Second Edittion, 2007.

[2] Παναγιώτης Δ. Μποζάνης, «Αλγόριθμοι», Εκδόσεις Τζιόλα, 2005.

[3] Jon Kleinberg, Eva Tardos, «Σχεδιασμός Αλγορίθμων», Εκδόσεις Κλειδάριθμος, 2008.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, «Εισαγωγή στους Αλγορίθμους», Τόμος 1, Πανεπιστημιακές εκδόσεις Κρήτης, 2006.

[5] Robert Sedgewick, «Αλγόριθμοι σε C», Μέρη 1 – 4, Εκδόσεις Κλειδάριθμος, 2005.

[6] Robert Sedgewick, «Algorithm in C», Part 5, Third Edittion, Addison – Wesley Professional, 2001.

[7] Μάριος Μαυρονικόλας, «Θεωρία Γράφων» Τόμος Β', Ελληνικό Ανοικτό Πανεπιστήμιο, 2000.

[8] Guang R. Gao, Yong – Fong Lee and Vugranam C. Shreedhar, «Incremental Computation of Dominator Trees», ACM Transactions on Programming Language and System, Vol. 19, No. 2, Martch 2007, Pages 239 - 252.

[9] Keith D. Cooper, Timothy J. Harvey and Ken Kennedy, «A Simple, Fast Dominiance Algorithm», Software Practice and Experience, 2001, Vol. 4, Pages 1 – 10.

[10] Thomas Lengauer and Robert E. Tarjan, «A Fast Algorithm for Finding Dominators in a Flowgraph», ACM Transactions on Programming Language and System, Vol. 1, No. 1, July 1979, Pages 121 - 141 .

[11] Loukas Georgiadis, Robert E. Tarjan and Renato F. Werneck, «Finding Dominators in Practice», Journal of Graph Algorithm and Applications, Vol. 10, No. 1, 2006, Pages 69 - 94.

[12] Ron Cytron, Jeanne Ferrante, Barry K. Rosen and Mark N. Wegman, «Efficiently Computing Static Single Assignment Form and the Control

Dependence Graph», ACM Transactions on Programming Language and System, Vol. 13, No. 4, October 1991, Pages 451 - 490.

[13] Johann Blieberger, «Average case analysis of DJ graphs», Journal of Discrete Algorithms, Elsevier Science Publishers B. V., Vol. 4, Issue 4, December 2006, Pages 649 – 675.

[14] Guang R. Gao, Yong-Fong Lee and Vugranam C. Sreedhar, « Identifying loops using DJ graphs», ACM Trans. Program. Lang. Syst., Vol. 18, Issue 6, November 1996, Pages 649 – 658.

[15] Andrew W. Appel, «Modern compiler implementation in Java», Cambridge University Press, 1998.

[16] Guang R. Gao and Vugranam C. Sreedhar, «A linear time algorithm for placing φ-nodes», Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, Series POPL '95, 1995, 62 - 73.

[17] G. Ramalingam, «On loops, dominators, and dominance frontiers», ACM Trans. Program. Lang. Syst., Vol. 24, Issue 5, September 2002, Pages 455 – 490.

[18] G. Ramalingam and Thomas Reps, « An incremental algorithm for maintaining the dominator tree of a reducible flowgraph», Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '94), ACM, 1994, Pages 287 – 296.

[19] Loukas Georgiadis and Robert E. Tarjan, « Finding dominators revisited», Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '04), Society for Industrial and Applied Mathematics, 2004, Pages 869 – 878.

[20] Adam L. Buchsbaum, Loukas Georgiadis, Haim Kaplan, Anne Rogers, Robert E. Tarjan, and Jeffery Westbrook, «Linear-Time Algorithms for Dominators and Other Path-Evaluation Problems», SIAM Journal on Computing (SICOMP), volume 38, No. 4, 2008, pages 1533-1573.

[21] http://www.mpi-inf.mpg.de/~mehlhorn/ftp/Toolbox/GraphRep.pdf, κείμενο «Graph Representation» .

[22] http://www.algolist.net/Data_structures/Graph/Internal_representation, κείμενο «Undirected graphs representation».

[23] J. B. Kam and J. D. Ullman, «Global Data Flow Analysis and Iterative Algorithms», Journal of the ACM, Vol. 23, 1976, Pages 158 – 171.

[24] A. L. Buchsbaum, H. Kaplan, A. Rogers and J. R. Westbrook, « Linear-Time Pointer-Machine Algorithms for Least Common Ancestors, mst

verification and dominators », ACM Transactions on Programming Languages and Systems, Vol. 20, November 1998, Pages 1265 – 1296.

[25] E. Lowry and C. Medloc, «Object Code Optimization», Communication of the ACM, January 1969, Pages 13 – 22.

[26] S. S. Muchnick, «Advanced Compiler Design and Implementation», Morgan – Kaufmann Publishers, San Francisco, CA, Ch. 14, 1997.