ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

Σχολή Θετικών Επιστημών και Τεχνολογίας

Τμήμα Επιστήμης και Τεχνολογίας Υπολογιστών

Πρόγραμμα Μεταπτυχιακών Σπουδών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Τεχνικές χρονοπρογραμματισμού σε βελτιστοποιητικούς μεταγλωττιστές

Παναγιώτης Γ. Ανδρινόπουλος

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: **Νικόλαος Κ. Καββαδίας**

ΤΡΙΠΟΛΗ

ΔΕΚΕΜΒΡΙΟΣ 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

Σχολή Θετικών Επιστημών και Τεχνολογίας
Τμήμα Επιστήμης και Τεχνολογίας Υπολογιστών
Πρόγραμμα Μεταπτυχιακών Σπουδών

Κατεύθυνση Θεωρητικής Πληροφορικής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Τεχνικές χρονοπρογραμματισμού σε βελτιστοποιητικούς μεταγλωττιστές

Παναγιώτης Γ. Ανδρινόπουλος

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: **Νικόλαος Κ. Καββαδίας**

ΤΡΙΠΟΛΗ
ΔΕΚΕΜΒΡΙΟΣ 2011

……..

Παναγιώτης Γ. Ανδρινόπουλος
Πτυχιούχος μεταπτυχιακού προγράμματος σπουδών στην Επιστήμη και Τεχνολογία των Υπολογιστών του Πανεπιστημίου Πελοποννήσου

…στους γονείς μου

Περίληψη

Οι μεταγλωττιστές διαδραματίζουν αναμφίβολα κυρίαρχο ρόλο στην εξελικτική πορεία των υπολογιστικών συστημάτων, καθώς η ποιότητα του κώδικα που παράγει ένας μεταγλωττιστής, αντανακλά την αποδοτικότητα του ίδιου του υπολογιστικού συστήματος. Ένας μεταγλωττιστής μεταφράζει ουσιαστικά ένα πρόγραμμα από μια γλώσσα υψηλού επιπέδου σε μια γλώσσα μηχανής, επιχειρώντας τη βελτιστοποίηση του κώδικα που θα προκύψει, ώστε να επιτευχθεί η καλύτερη δυνατή απόδοση μιας εφαρμογής στο δεδομένο επεξεργαστή.

*Η παρούσα διπλωματική εργασία, έχει ως θέμα τις **«Τεχνικές χρονοπρογραμματισμού σε βελτιστοποιητικούς μεταγλωττιστές»**. Ειδικότερα, η εργασία αποτελείται από την παρουσίαση και την ανάλυση μεθόδων αποδοτικού χρονοπρογραμματισμού ενός προγράμματος, κάτω από περιορισμούς χρόνου και των διαθέσιμων πόρων του επεξεργαστή. Το ενδιαφέρον μας εστιάζεται στους γράφους προγραμμάτων και στους μετασχηματισμούς στους οποίους αυτοί υπόκεινται, προκειμένου να επιτευχθεί αποδοτικός χρονοπρογραμματισμός εντολών.*

Μέσα από μια αναφορά σε γενικές έννοιες και ορισμούς στο αρχικό κεφάλαιο, επιδιώκεται μια εισαγωγή στον κόσμο των μεταγλωττιστών καθώς επίσης και στις τεχνικές χρονοπρογραμματισμού εντολών. Θέματα όπως η δέσμευση καταχωρητών, ο παραλληλισμός επιπέδου εντολών και τα διάφορα είδη εξαρτήσεων, εξετάζονται εκτενώς. Η μετάβαση στο κυρίως θέμα της διπλωματικής συντελείται στα αμέσως επόμενα κεφάλαια (2 και 3), όπου περιγράφεται αναλυτικά η διαδικασία του χρονοπρογραμματισμού μέσω μετασχηματισμών σε ένα γράφο προγράμματος, αφού πρώτα οριστεί το σύνολο των λογικών προτεραιοτήτων για το δεδομένο γράφο. Στο τρίτο κεφάλαιο παρατίθενται οι ανωτέρω μετασχηματισμοί για γράφους εξάρτησης δεδομένων, σε συνδυασμό με τους αντίστοιχους αλγορίθμους.

Στο τελευταίο κεφάλαιο η εργασία αναδεικνύει με παραδείγματα τους βασικούς αλγορίθμους χρονοπρογραμματισμού, ASAP και ALAP (για προβλήματα χωρίς περιορισμούς) καθώς και τον αλγόριθμο χρονοπρογραμματισμού λίστας (με περιορισμούς). Τέλος πραγματοποιείται μια σύντομη αναφορά στην περίπτωση χρονοπρογραμματισμού πολλαπλών γράφων σε ετερογενή συστήματα.

Λέξεις κλειδιά

Μεταγλωττιστές, Παραλληλισμός Επιπέδου Εντολών, Γράφος, Μετασχηματισμός Γράφου, Χρονοπρογραμματισμός Εντολών, ASAP, ALAP, Χρονοπρογραμματισμός Λίστας, Πολυπλοκότητα Αλγορίθμων, Βασικό Μπλοκ.

Abstract

There is no doubt that compilers play a dominant role in the evolutionary course of computing systems, as the quality of the code produced by a compiler reflects the efficiency of its own computing system. A compiler translates a program substantially from a high-level language into a machine language, attempting to optimize the resulting code, to achieve the best possible performance of an application to a given processor.

The subject of this thesis is ***"Scheduling Techniques to Compilers for Optimization"****. In particular, this thesis is constituded by the presentation and the analysis of efficient methods for program scheduling, under time constraints and available resources of the processor. Our interest is focused in program graphs and to the transformations of them, so that an efficient instruction scheduling to be achieved.*

Through a reference to general concepts and definitions in the initial chapter, an introduction to the world of compilers is sought as well as with the techniques for instruction scheduling. Issues such as engagement of registers, instruction-level parallelism and the different types of dependencies, are extensively examined. The transition to the main subject of this thesis takes place directly in the next chapters (2 and 3), where the instruction scheduling process is described analytically via transformations in a program graph, after the first set up of all rational priorities for the given graph. The third chapter sets out the above transformations, for data-dependency graphs in combination with the corresponding algorithms.

The last chapter of this thesis sets off the basic scheduling algorithms via examples, ASAP and ALAP (for problems without restrictions) and the List Scheduling algorithm (with restrictions). Finally a short reference is realised, to the case of multiple graph scheduling onto heterogeneous systems.

Key Words

Compilers, Instruction-level Parallelism, Graph, Graph Transformation, Instruction Scheduling, ASAP, ALAP, List Scheduling, Algorithm Complexity, Basic Block.

Ευχαριστίες

Η εκπόνηση της παρούσας διπλωματικής εργασίας δεν θα ήταν δυνατό να πραγματοποιηθεί, χωρίς την συνεχή καθοδήγηση και επίβλεψη από τον εισηγητή αυτής κ. Καββαδία Νικόλαο, ώστε αυτή η προσπάθεια να έρθει εις πέρας.

Θα ήθελα στο σημείο αυτό να τον ευχαριστήσω για τη δυνατότητα που μου προσέφερε να ασχοληθώ με ένα ιδιαίτερο και άκρως ενδιαφέρον θέμα, παρέχοντάς μου ταυτόχρονα χρήσιμο υλικό για την ανάπτυξη της εργασίας. Ελπίζω να ανταποκρίθηκα στις προσδοκίες του και η ευγνωμοσύνη μου να ανταποδίδει σε κάποιο βαθμό τις προσπάθειές του.

Ξεκινώντας την εκπόνηση της παρούσας εργασίας, είχα την άποψη ότι οι επιστημονικές προκλήσεις θα ήταν τα μόνα εμπόδια που θα συναντούσα. Δυστυχώς δεν ήταν έτσι. Η εκπόνηση μιας διπλωματικής εργασίας είναι μια ψυχοφθόρα διαδικασία, την οποία δεν θα είχα καταφέρει να ολοκληρώσω χωρίς την ψυχολογική κυρίως υποστήριξη, φίλων αλλά και της οικογένειάς μου. Η απομόνωση κατά διαστήματα που απαιτήθηκε, με έκανε να παραμελήσω πολλούς από τους φίλους μου. Μερικοί θα πρέπει να έχουν κάποια παράπονα, αλλά τους ευχαριστώ για την κατανόησή τους και τους ζητώ να με συγχωρήσουν.

Τέλος θα ήθελα να ευχαριστήσω και τους γονείς μου, Γεώργιο και Αικατερίνη, για την αμέριστη συμπαράστασή τους όλο αυτό το διάστημα αλλά και γενικότερα στη ζωή μου.

Παναγιώτης Γ. Ανδρινόπουλος
Τρίπολη
Δεκέμβριος 2011

Πίνακας περιεχομένων

Ευρετήριο σχημάτων

1

Μεταγλωττιστές

1 – Μεταγλωττιστές

Οι μεταγλωττιστές αποτελούν αναπόσπαστο κομμάτι της ανάπτυξης υπολογιστικών συστημάτων, είτε πρόκειται για επιτραπέζια συστήματα, είτε για εξυπηρετητές (servers), είτε για ενσωματωμένα συστήματα. Τη σημερινή εποχή, οι διάφορες υπολογιστικές εφαρμογές σε πλήθος διαφορετικών θεμάτων εκτός φυσικά από κάποιες εξαιρέσεις, περιγράφονται με χρήση γλωσσών υψηλού επιπέδου. Ωστόσο είναι γνωστό, ότι τα υπολογιστικά συστήματα όλων των ειδών και όλοι οι επεξεργαστές εκτελούν εντολές που είναι σε μορφή γλώσσας μηχανής. Ο συνδετικός κρίκος που ενώνει τις πηγαίες γλώσσες χρονοπρογραμματισμού με τη γλώσσα χαμηλού επιπέδου που αντιλαμβάνεται ένας επεξεργαστής, είναι ο μεταγλωττιστής.

1.1 – Εισαγωγή στους μεταγλωττιστές

Ο μεταγλωττιστής (compiler) ή αλλιώς μεταφραστής, είναι ένα πρόγραμμα (ή σύνολο προγραμμάτων) που μετατρέπει τον πηγαίο κώδικα (source code) μιας εφαρμογής, σε γλώσσα μηχανής (machine code). Ο κύριος λόγος για τη μετατροπή αυτή είναι η δημιουργία εκτελέσιμου προγράμματος. Ουσιαστικά λοιπόν ο μεταγλωττιστής, δέχεται τις εντολές από το υψηλό επίπεδο και δημιουργεί τις κατάλληλες δυαδικές ακολουθίες από 0 και 1 για τον συγκεκριμένο επεξεργαστή που έχουμε στη διάθεσή μας κάθε φορά, ώστε να υλοποιείται η εφαρμογή που θέλουμε. Οι πιο συνηθισμένες λειτουργίες ενός μεταγλωττιστή είναι η λεκτική ανάλυση, συντακτική ανάλυση, παραγωγή κώδικα και η βελτιστοποίηση κώδικα.

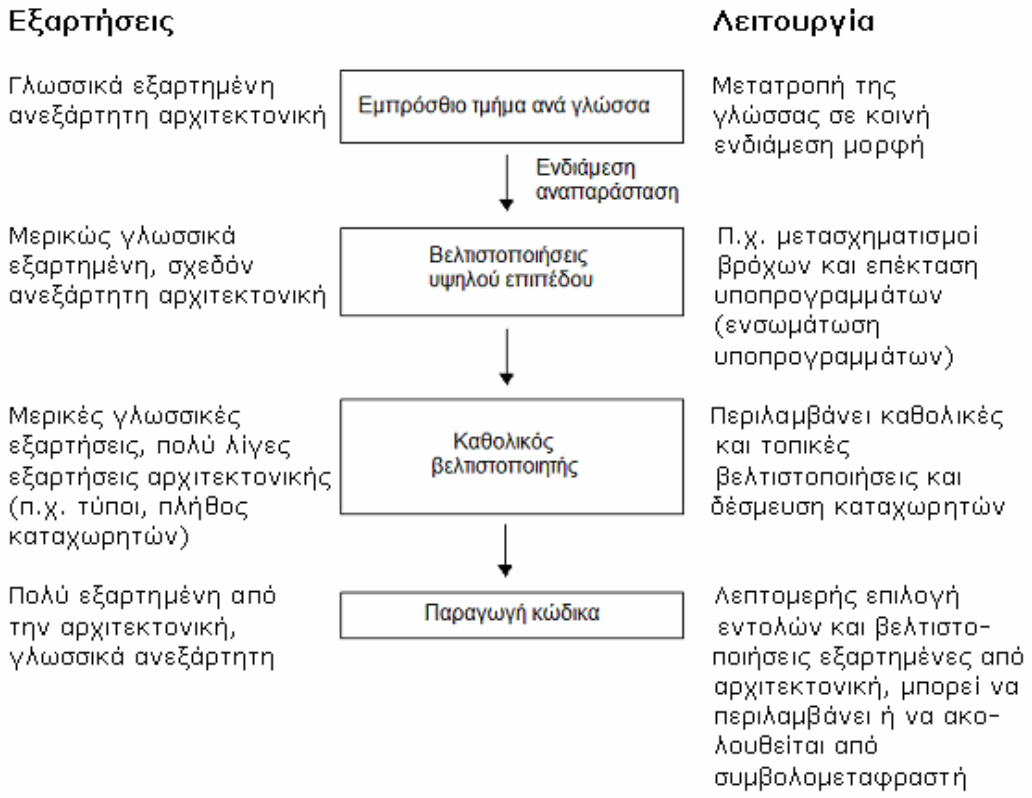
Είναι προφανές από όσα αναφέρθηκαν παραπάνω, ότι η ποιότητα του παραγόμενου κώδικα από τον μεταγλωττιστή, έχει ιδιαίτερη σημασία για την απόδοση του υπολογιστικού συστήματος και κυρίως για τα ενσωματωμένα συστήματα. Άρα η κατανόηση της τεχνολογίας και του τρόπου λειτουργίας τους αποκτά ενδιαφέρον, σε μια εποχή που η ανάγκη για υψηλού επιπέδου εφαρμογές είναι ολοένα και μεγαλύτερη.

Ο πρωτεύων στόχος για ένα μεταγλωττιστή είναι η ορθότητα, το γεγονός δηλαδή ότι όλα τα έγκυρα προγράμματα πρέπει να μεταφραστούν σωστά σε γλώσσα μηχανής. Εξίσου σημαντικός στόχος είναι η δημιουργία όσο το δυνατό πιο γρήγορα μεταφρασμένου κώδικα, δηλαδή ταχύτερου στην εκτέλεση. Σε ενσωματωμένα συστήματα σημαντικό ρόλο κατέχει και το μέγεθος του κώδικα. Ως στόχους μπορούμε επίσης να αναφέρουμε τη διαλειτουργικότητα μεταξύ των διαφόρων γλωσσών προγραμματισμού, ακόμα και την όσο το δυνατό καλύτερη μεταφορά εφαρμογών μεταξύ διαφορετικών συστημάτων. Όλα αυτά είναι σημαντικά θέματα και πρέπει να ληφθούν σοβαρά υπόψιν. Ιδιαίτερο ενδιαφέρον θα δοθεί παρακάτω στους βελτιστοποιητικούς μεταγλωττιστές και συγκεκριμένα, στις τεχνικές ως προς τη παραλληλία των εντολών.

1.2 – Η δομή ενός μοντέρνου μεταγλωττιστή

Για την καλύτερη απόδοση μιας εφαρμογής, η διαδικασία μετάφρασης του κώδικα από μια γλώσσα υψηλού επιπέδου σε γλώσσα μηχανής και η βελτιστοποίησή του από τον μεταγλωττιστή, γίνεται με διαδοχικά βήματα και όχι με ένα μεγάλο βήμα. Αυτό συμβαίνει, διότι σε διαφορετική περίπτωση θα απαιτούνταν πολύς χρόνος για να γίνει κάτι τέτοιο ενώ ο κίνδυνος εισαγωγής σφαλμάτων θα ήταν μεγάλος. Φυσικά μια τέτοια βελτιστοποίηση δεν είναι σίγουρο αν θα έδινε το καλύτερο δυνατό αποτέλεσμα σε πεπερασμένο χρονικό διάστημα.

Η πολυπλοκότητα της συγγραφής ενός μεταγλωττιστή, δυσκολεύει σημαντικά τη συντήρηση των τμημάτων του που ασχολούνται με τη βελτιστοποίηση, εφόσον αυτός δεν είναι κατάλληλα οργανωμένος. Η δομή με τα πολλαπλά βήματα βοηθά σημαντικά στη μείωση της πολυπλοκότητας που παρουσιάζει, που σημαίνει με τη σειρά του ότι ο μεταγλωττιστής πρέπει να βάλει τους βελτιστοποιητικούς μετασχηματισμούς του σε κάποια σειρά. Είναι προφανές ότι κάποιοι εκτελούνται πριν από τους υπόλοιπους.



Σχήμα 1.1: *Η δομή ενός μεταγλωττιστή τεσσάρων βημάτων*

Από το προηγούμενο σχήμα είναι εμφανές, ότι μερικές βελτιστοποιήσεις υψηλού επιπέδου εκτελούνται πολύ πριν οριστικοποιηθεί ο τελικός κώδικας

που θα προκύψει. Στο σημείο αυτό παρεμβάλλεται η ενδιάμεση αναπαράσταση, η οποία πρέπει να είναι αρκετά λεπτομερής ώστε οι διάφορες τιμές του αρχικού προγράμματος να αναπαρασταθούν στη δεδομένη αρχιτεκτονική. Αυτή παράγεται από το εμπρόσθιο τμήμα (front-end) του μεταγλωττιστή μαζί με τον πίνακα συμβόλων ανά γλώσσα.

Μετά από μια τέτοια βελτιστοποίηση, είναι ασύμφορο για το μεταγλωττιστή να επιστρέψει και να εκτελέσει όλα τα βήματα από την αρχή, αναιρώντας πιθανότατα τους προηγούμενους μετασχηματισμούς. Αυτό θα ήταν απαγορευτικό τόσο στο χρόνο μεταγλώττισης όσο και στην πολυπλοκότητα. Επομένως οι μεταγλωττιστές κάνουν υποθέσεις σχετικά με την ικανότητα μετέπειτα βημάτων να αντιμετωπίσουν ορισμένα προβλήματα. Ως παράδειγμα, οι μεταγλωττιστές συνήθως πρέπει να επιλέξουν ποιες κλήσεις υποπρογραμμάτων θα επεκτείνουν με ενσωμάτωση (inlining), πριν μάθουν το ακριβές μέγεθος του υποπρογράμματος που καλείται. Οι συγγραφείς των μεταγλωττιστών ονομάζουν αυτό το πρόβλημα ως «πρόβλημα διάταξης φάσεων» (Phase-Ordering Problem) **[1]**.

Εύλογα ερωτήματα θα μπορούσαν να σχετίζονται, με τον τρόπο αλληλεπίδρασης της διάταξης των μετασχηματισμών με την αρχιτεκτονική του συνόλου των εντολών. Τέτοιο παράδειγμα είναι αυτό της βελτιστοποίησης που ονομάζεται «καθολική απαλοιφή των κοινών υποεκφράσεων» (Common subexpression elimination). Σε αυτή τη βελτιστοποίηση ο μεταγλωττιστής βρίσκει δυο αντίγραφα κάποιας έκφρασης που υπολογίζουν τις ίδιες τιμές, και αποθηκεύει προσωρινά το αποτέλεσμα του πρώτου υπολογισμού. Κατόπιν χρησιμοποιεί την προσωρινή τιμή απαλείφοντας το δεύτερο υπολογισμό της κοινής έκφρασης.

Ωστόσο για να είναι σημαντική αυτή η βελτιστοποίηση, η προσωρινή τιμή πρέπει να τοποθετείται σε κάποιον καταχωρητή. Σε άλλη περίπτωση, το κόστος αποθήκευσης της προσωρινής τιμής στη μνήμη και αργότερα της επαναφοράς της, μπορεί να εκμηδενίσει το όφελος που έχουμε από μια επανάληψη του υπολογισμού της έκφρασης. Στην πραγματικότητα υπάρχουν περιπτώσεις στις οποίες η βελτιστοποίηση αυτή κάνει τον κώδικα πιο αργό, όταν η προσωρινή τιμή δεν τοποθετείται σε καταχωρητή. Η διάταξη των φάσεων περιπλέκει αυτό το πρόβλημα, διότι η δέσμευση καταχωρητών γίνεται συνήθως προς το τέλος του βήματος της «καθολικής βελτιστοποίησης», ακριβώς πριν από την παραγωγή του κώδικα. Έτσι ένας μεταγλωττιστής που εκτελεί αυτή τη βελτιστοποίηση, πρέπει να υποθέσει ότι η δέσμευση καταχωρητών θα τοποθετήσει την προσωρινή τιμή σε έναν καταχωρητή.

Γενικά οι βελτιστοποιήσεις που εκτελούνται από σύγχρονους μεταγλωττιστές, μπορούν να κατηγοριοποιηθούν σύμφωνα με τον τύπο του μετασχηματισμού ως εξής:

— **Βελτιστοποιήσεις υψηλού επιπέδου:** πρόκειται γι αυτές που συχνά πραγματοποιούνται στον πηγαίο κώδικα, με το αποτέλεσμά τους να τροφοδοτεί επόμενα βήματα βελτιστοποίησης.
— **Τοπικές βελτιστοποιήσεις:** αυτές βελτιώνουν τον κώδικα μόνο μέσα στα πλαίσια ενός τμήματος κώδικα εντολών σε σειρά.
— **Καθολικές βελτιστοποιήσεις:** είναι αυτές που επεκτείνουν τις τοπικές βελτιστοποιήσεις πέρα από διακλαδώσεις, και εισάγουν ένα σύνολο μετασχηματισμών για τη βελτιστοποίηση των βρόχων.
— **Δέσμευση καταχωρητών:** αντιστοιχία καταχωρητών με τελεσταίους ή μεταβλητές.
— **Εξαρτημένες από τον επεξεργαστή:** είναι αυτές που προσπαθούν να εκμεταλλευτούν συγκεκριμένα αρχιτεκτονικά χαρακτηριστικά του επεξεργαστή.

Μερικές φορές είναι δύσκολο να διαχωριστούν κάποιες από τις απλούστερες βελτιστοποιήσεις – οι τοπικές και αυτές που εξαρτώνται από τον επεξεργαστή – από τους μετασχηματισμούς που γίνονται στην παραγωγή κώδικα. Παραδείγματος χάριν οι βελτιστοποιήσεις υψηλού επιπέδου (ενσωμάτωση υποπρογράμματος), οι τοπικές βελτιστοποιήσεις (απαλοιφή κοινών υποεκφράσεων, διάδοση σταθεράς) και οι καθολικές βελτιστοποιήσεις (καθολική απαλοιφή κοινών υποεκφράσεων, διάδοση αντιγράφου, μετακίνηση κώδικα και άλλες). Τέλος, μεγάλη σημασία έχει η ανάλυση ενός βελτιστοποιημένου κώδικα πριν προταθούν βελτιώσεις σε ένα σύνολο εντολών, καθώς ο μεταγλωττιστής μπορεί να αφαιρέσει ολοκληρωτικά τις εντολές που ο αρχιτέκτονας προσπαθούσε να βελτιώσει.

1.3 – Δέσμευση καταχωρητών

Ο κεντρικός ρόλος που διαδραματίζει η δέσμευση καταχωρητών, τόσο στην επιτάχυνση εκτέλεσης του κώδικα όσο και στο να κάνει χρήσιμες κάποιες άλλες βελτιστοποιήσεις, την αναδεικνύει σε μια από τις πιο σημαντικές – αν όχι τη σημαντικότερη – βελτιστοποιήσεις. Μια αποδεκτή τεχνική με την οποία οι μεταγλωττιστές προσπαθούν να δεσμεύσουν με τον πιο κατάλληλο τρόπο τους καταχωρητές, στηρίζεται σε αλγορίθμους γράφων και η υλοποίηση της δέσμευσης ονομάζεται «χρωματισμός γράφου».

Η βασική ιδέα για την τεχνική αυτή, είναι η κατασκευή ενός γράφου που αναπαριστά τους πιθανούς υποψήφιους κόμβους για τοποθέτηση σε καταχωρητή, καθώς και η μετέπειτα χρήση του γράφου αυτού για τη δέσμευση των καταχωρητών. Ο γράφος αυτός ονομάζεται γράφος παρεμβολής (interference graph), οι κόμβοι του αναπαριστούν συμβολικές μεταβλητές ενώ οι ακμές αναπαριστούν περιορισμούς δέσμευσης στον ίδιο καταχωρητή. Σε γενικές γραμμές, το πρόβλημα του χρωματισμού γράφου είναι πώς να χρησιμοποιήσουμε ένα περιορισμένο σύνολο από χρώματα, ώστε σε ένα γράφο

εξαρτήσεων να μην υπάρχουν δυο γειτονικοί κόμβοι με το ίδιο χρώμα. Ο βασικός στόχος σε αυτή τη προσέγγιση, είναι η δέσμευση καταχωρητών για όσο το δυνατόν περισσότερες ενεργές μεταβλητές. Ο ασυμπτωτικός χρόνος επίλυσης του προβλήματος του χρωματισμού γράφου μπορεί να είναι εκθετικός, ως συνάρτηση του μεγέθους του γράφου. Ο χρωματισμός γράφου αποδίδει καλύτερα όταν ο επεξεργαστής έχει ένα ομογενές αρχείο καταχωρητών γενικού σκοπού, οι οποίοι είναι διαθέσιμοι για καθολική δέσμευση για ακέραιες μεταβλητές, όπως επίσης και επιπρόσθετοι καταχωρητές για μεταβλητές κινητής υποδιαστολής. Δυστυχώς, ο χρωματισμός γράφου δεν αποδίδει πολύ καλά όταν ο αριθμός των διαθέσιμων καταχωρητών είναι μικρός, επειδή τότε οι ευριστικοί (heuristic) αλγόριθμοι που χρωματίζουν το γράφο έχουν μεγάλη πιθανότητα να αποτύχουν.

1.4 – Παραλληλισμός επιπέδου εντολών και εξαρτήσεις

Οι επεξεργαστές ιδιαίτερα από την εισαγωγή της αρχιτεκτονικής επεξεργαστών RISC και έπειτα, σε μια προσπάθεια να επιτευχθεί η μεγαλύτερη δυνατή απόδοση των εφαρμογών, χρησιμοποιούν την τεχνική της διοχέτευσης (pipelining) για να επικαλύψουν την εκτέλεση εντολών **[2]**. Αυτή η επικάλυψη εντολών περιγράφεται ως «παραλληλισμός επιπέδου εντολών» (Instruction Level Parallelism - ILP), επειδή οι εντολές μπορούν να εκτελεστούν παράλληλα. Για παράδειγμα η πραγματοποίηση τεσσάρων (4) αθροίσεων σε κάθε κύκλο ρολογιού, προϋποθέτει την ύπαρξη τουλάχιστον τεσσάρων αθροιστών οι οποίοι μπορούν να χρησιμοποιηθούν κάθε φορά για τις ανάγκες της εφαρμογής.

Το όφελος που προκύπτει από τον παραλληλισμό είναι πάρα πολύ σημαντικό. Τα πράγματα δεν είναι όμως τόσο απλά όσο φαίνονται, αφού ακόμα και αν έχουμε τους πόρους σε υλικό για να εκμεταλλευτούμε τον παραλληλισμό, αυτό δεν είναι πάντα εφικτό. Αυτό συμβαίνει διότι υπάρχουν οι διάφορες εξαρτήσεις μεταξύ των εντολών του προγράμματος που τρέχει κάθε φορά, οι οποίες οφείλονται τόσο στα δεδομένα που είναι κοινά σε διάφορες εντολές όσο και στους ελέγχους που υπάρχουν σε κάθε πρόγραμμα, και πολλές φορές δεν μπορούμε από πριν να γνωρίζουμε τι θα μας δώσουν ως απάντηση. Στη συνέχεια λοιπόν θα αναφερθούμε πιο αναλυτικά στον ορισμό του παραλληλισμού κατά την εκτέλεση ενός προγράμματος και επιπρόσθετα, θα εξετάσουμε το θέμα των εξαρτήσεων.

1.4.1 – Ορισμός του παραλληλισμού επιπέδου εντολών

Το διαθέσιμο ποσοστό παραλληλισμού μέσα σε μια βασική ακολουθία κώδικα (χωρίς εισερχόμενους κλάδους εκτός από την αρχή της και καθόλου

εξερχόμενες διακλαδώσεις παρά μόνο στο τέλος της), είναι αρκετά μικρό. Σε ένα τυπικό πρόγραμμα η συχνότητα των διακλαδώσεων ανέρχεται σε ποσοστό 15% - 20% , που σημαίνει ότι 4 με 7 εντολές εκτελούνται ανάμεσα σε ένα ζεύγος διακλαδώσεων. Άρα αφού αυτές οι εντολές πιθανόν εξαρτώνται η μια από την άλλη, το προς εκμετάλλευση ποσοστό επικάλυψης μέσα σε μια βασική ακολουθία είναι ίσως πολύ μικρότερο από το μέγεθος του βασικού μπλοκ (Basic Block). Για να υπάρξει ουσιαστική βελτίωση της απόδοσης, θα πρέπει να εκμεταλλευτούμε ταυτόχρονα τον παραλληλισμό επιπέδου εντολών από πολλαπλές βασικές ακολουθίες.

Ο πιο απλός και κοινός τρόπος για να αυξηθεί το διαθέσιμο ποσοστό του παραλληλισμού εντολών, είναι η εκμετάλλευση του παραλληλισμού μεταξύ των επαναλήψεων ενός βρόχου. Πρόκειται για τον λεγόμενο «παραλληλισμό επιπέδου βρόχου» (Loop Level Parallelism - LLP). Στο παράδειγμα του κώδικα που ακολουθεί, φαίνεται ένας βρόχος που προσθέτει δυο σειρές 1000 στοιχείων παράλληλα.

```
for (i=0; i<1000; i++)
            x[i]=x[i]+y[i];
```

Η επανάληψη κάθε βρόχου μπορεί να επικαλυφθεί με οποιαδήποτε άλλη επανάληψη, αν και μέσα σε κάθε επανάληψη δεν υπάρχει σχεδόν καμία ευκαιρία επικάλυψης.

Η χρήση διανυσματικών εντολών, είναι μια σημαντική μέθοδος για την εκμετάλλευση του παραλληλισμού επιπέδου βρόχου. Μια διανυσματική εντολή λειτουργεί σε μια ακολουθία δεδομένων. Ο παραπάνω κώδικας για παράδειγμα, θα μπορούσε να εκτελεστεί με τέσσερις εντολές σε μερικούς διανυσματικούς επεξεργαστές: δυο εντολές για τη φόρτωση των διανυσμάτων x και y από τη μνήμη, μια εντολή για την πρόσθεση των διανυσμάτων και μια για την αποθήκευση του αποτελέσματος. Φυσικά αυτές οι εντολές δεν θα εκτελεστούν ταυτόχρονα και έχουν σχετικά μακρόχρονη καθυστέρηση, αλλά αυτές οι καθυστερήσεις μπορούν να επικαλυφθούν. Αν και η εξέλιξη των ιδεών περί διανυσμάτων προηγήθηκε πολλών τεχνικών οι οποίες εκμεταλλεύονται τον παραλληλισμό επιπέδου εντολών, οι επεξεργαστές που εκμεταλλεύονται τον παραλληλισμό τύπου SIMD (Single Instruction, Multiple Data), έχουν αντικαταστήσει σε μεγάλο βαθμό τους αποκλειστικά διανυσματικούς επεξεργαστές.

1.4.2 – Θεωρία εξαρτήσεων – Εξαρτήσεις δεδομένων

Οι διάφορες εξαρτήσεις που μπορούν να προκύψουν κατά την εκτέλεση μιας εφαρμογής, συμβαίνουν είτε λόγω εξαρτήσεων στα δεδομένα είτε λόγω

εξαρτήσεων ελέγχου. Οι εξαρτήσεις στα δεδομένα για τις οποίες θα αναφερθούμε, έχουν ως αποτέλεσμα τη μείωση του παραλληλισμού και συνεπώς της ταχύτητας εκτέλεσης της εφαρμογής.

Θεωρητικά λέμε ότι μια εντολή **i** εξαρτάται από μια άλλη **j,** όταν δεν μπορεί να εκτελεστεί:

A. γιατί περιμένει το αποτέλεσμα από την εκτέλεση της εντολής **j** ή
B. γιατί εξαρτάται από δεδομένα που παράγονται από την εκτέλεση της εντολής **k,** που με την σειρά της εξαρτάται από την εντολή **j**.

Η δεύτερη κατάσταση απλά δηλώνει ότι μια εντολή εξαρτάται από μια άλλη, όχι μόνο απευθείας όπως στην πρώτη περίπτωση, αλλά από την ύπαρξη μιας αλυσίδας εξαρτήσεων του πρώτου τύπου μεταξύ των δυο εντολών. Το χειρότερο βέβαια, είναι ότι αυτή η εξάρτηση μπορεί να είναι τόσο εκτεταμένη όσο ολόκληρο το πρόγραμμα. Στο σχήμα 1.2 φαίνεται ένα παράδειγμα ενδεικτικό της εξάρτησης εντολών των δυο τύπων που περιγράψαμε.

Πρώτος τύπος	**Δεύτερος τύπος**
`S1: x=100;`	`S1: x=100;`
`S2: y=200;`	`S2: y=x+200;`
`S3: z=x+y;`	`S3: z=y+300;`

Σχήμα 1.2: *Εξαρτήσεις εντολών*

Όπως προαναφέρθηκε, η εντολή S3 στην πρώτη περίπτωση δεν δύναται να εκτελεστεί πριν εκτελεστούν οι δυο προηγούμενες εντολές, διαφορετικά το αποτέλεσμα που θα λάβουμε δεν θα είναι το επιθυμητό. Στη δεύτερη περίπτωση είναι εμφανής η εξάρτηση του z από το x, μεσολαβώντας στο ενδιάμεσο και η μεταβλητή y. Γενικά υπάρχουν τρεις διαφορετικές κατηγορίες εξαρτήσεων οι οποίες διακρίνονται ανάλογα με τον τύπο τους σε: TRUE DEPENDENCIES, ANTI-DEPENDENCIES, OUTPUT DEPENDENCIES **[3]**. Αυτές λοιπόν τις εξαρτήσεις θα αναλύσουμε στη συνέχεια.

— **TRUE DEPENDENCIES:** Η εξάρτηση αυτή υπάρχει, όταν η πρώτη εντολή γραφεί σε μια θέση αποθήκευσης την οποία και διαβάζει μια δεύτερη εντολή. Είναι η πιο συνηθισμένη μορφή εξάρτησης την οποία συναντάμε και με το όνομα RAW (Read-After-Write). Είναι εμφανές το πρόβλημα που προκύπτει εάν δεν ικανοποιηθεί αυτή η εξάρτηση, καθώς η εντολή που ακολουθεί διαβάζει την παλιά τιμή πριν προλάβει αυτή να ανανεωθεί μέσω της προηγούμενης εντολής που δεν εκτελείται. Παράδειγμα τέτοιας εξάρτησης είναι:

```
S1: x=……;
S2: ……=x;
```

— **ANTI-DEPENDENCIES:** Η εξάρτηση αυτή υπάρχει, όταν η πρώτη εντολή διαβάζει από μια θέση στην οποία γράφει η επόμενη εντολή. Αυτή η εξάρτηση είναι γνωστή και με την ονομασία WAR (Write-After-Read). Εάν η εξάρτηση αυτή δεν τηρηθεί και γίνει πρώτα η εγγραφή, τότε η πρώτη εντολή θα λάβει τη νέα τιμή και όχι την παλιά όπως πρέπει κανονικά. Η αλήθεια είναι πάντως ότι η εξάρτηση αυτή δεν παρουσιάζεται πολύ συχνά. Παράδειγμα αυτής της εξάρτησης είναι το εξής:

```
S1: ……=x;
S2: x=……;
```

— **OUTPUT DEPENDENCIES:** Αυτή η εξάρτηση υπάρχει, όταν δυο εντολές θέλουν να γράψουν διαδοχικά στην ίδια θέση. Είναι επίσης γνωστή με το όνομα WAW (Write-After-Write) και δεν είναι τόσο συχνή όσο η RAW. Το πρόβλημα εάν δεν ικανοποιηθεί αυτή η εξάρτηση, είναι ότι καταλήγουμε τελικά στον προορισμό να μην έχουμε την τελική τιμή που πρέπει, αλλά μια προηγούμενη. Η εξάρτηση αυτή είναι της μορφής:

```
S1: x=……;
S2: x=……;
```

Πολλές φορές ένα οποιοδήποτε πρόγραμμα έχει να κάνει με εντολές που βρίσκονται μέσα σε βρόχους. Οι χαρακτηρισμοί εξαρτήσεων που αποδόθηκαν προηγουμένως αναφέρονται σε ευθύγραμμο (straight line) κώδικα, και ισχύουν επίσης για κώδικα μέσα σε βρόχο. Στην περίπτωση του βρόχου όμως προστίθενται δυο ακόμη χαρακτηρισμοί εξαρτήσεων: LOOP CARRIED και LOOP INDEPENDENT DEPENDENCIES.

Η LOOP CARRIED εξάρτηση υπάρχει, εάν οι προσβάσεις σε μετέπειτα επαναλήψεις εξαρτώνται από τιμές δεδομένων που παρήχθησαν σε προηγούμενες επαναλήψεις. Σε ότι έχει να κάνει με την εξάρτηση LOOP INDEPENDENT, αυτή δηλώνει ότι δεν υπάρχει εξάρτηση μεταξύ διαφορετικών επαναλήψεων του βρόχου, παρά μόνο μεταξύ δεδομένων της ίδιας επανάληψης του βρόχου.

1.4.3 – Θεωρία εξαρτήσεων – Εξαρτήσεις ελέγχου

Εκτός από τις εξαρτήσεις δεδομένων υπάρχουν και οι εξαρτήσεις ελέγχου. Μια τέτοια εξάρτηση καθορίζει τη διάταξη μιας εντολής έστω S1, σχετικά με μια εντολή διακλάδωσης, ώστε η εντολή S1 να εκτελείται με τη σωστή σειρά προγράμματος και μόνο όταν πρέπει **[4]**. Υπάρχουν λοιπόν στα περισσότερα προγράμματα, αρκετές εντολές που έχουν εξάρτηση ελέγχου με κάποιο σύνολο

διακλαδώσεων. Γενικά αυτές οι εξαρτήσεις ελέγχου πρέπει να διατηρούνται, για να διατηρηθεί η σειρά και η συνοχή του προγράμματος. Ένα από τα απλούστερα παραδείγματα εξάρτησης ελέγχου, είναι η εξάρτηση των εντολών στο τμήμα «then» μιας δήλωσης «if» στη διακλάδωση. Έστω το παρακάτω απόσπασμα :

```
if P1 then
{
S1 };
else if P2 then
{
S2 };
end if
```

Το S1 είναι εξαρτημένο από το p1 και ομοίως το S2 από το p2 αλλά όχι από το p1. Υπάρχουν κυρίως δυο περιορισμοί που επιβάλλονται από τις εξαρτήσεις ελέγχου:

1) Μια εντολή που έχει εξάρτηση ελέγχου από μια διακλάδωση, δεν μπορεί να μετακινηθεί σε κάποια θέση πριν από την διακλάδωση, γιατί με αυτόν τον τρόπο η εκτέλεσή της δεν ελέγχεται πλέον από τη διακλάδωση. Για παράδειγμα, δεν μπορούμε να πάρουμε μια εντολή από το τμήμα «then» ενός «if-then-else» και να την μετακινήσουμε πριν από τον έλεγχο συνθήκης «if».
2) Μια εντολή που δεν έχει εξάρτηση ελέγχου από μια διακλάδωση δεν μπορεί να μετακινηθεί μετά τη διακλάδωση, διότι η εκτέλεσή της πλέον θα ελέγχεται από τη διακλάδωση. Για παράδειγμα δεν μπορούμε να πάρουμε μια εντολή πριν από το «if» και να τη μετακινήσουμε μέσα στο τμήμα του «then».

Η διατήρηση της εξάρτησης ελέγχου αποτελεί έναν απλό και χρήσιμο τρόπο για τη διατήρηση της σειράς του προγράμματος, χωρίς όμως να είναι ταυτόχρονα βασικός περιορισμός της απόδοσης. Ίσως να είναι σχήμα οξύμωρο, εάν λέγαμε τώρα ότι η εξάρτηση ελέγχου δεν είναι η σημαντική ιδιότητα που πρέπει να διατηρηθεί. Αντί αυτής, οι δυο σημαντικές ιδιότητες για την ορθότητα του προγράμματος και οι οποίες φυσιολογικά διατηρούνται με την διατήρηση της εξάρτησης δεδομένων και ελέγχου, είναι η συμπεριφορά εξαιρέσεων και η ροή δεδομένων. Στο σημείο αυτό τονίζεται ότι οι εξαιρέσεις αφορούν το λειτουργικό σύστημα (αφού αυτό τις εξυπηρετεί) και δεν ενδιαφέρουν καθόλου το μεταγλωττιστή.

Η διατήρηση της συμπεριφοράς εξαιρέσεων, σημαίνει ότι οποιεσδήποτε αλλαγές στη διάταξη της εκτέλεσης εντολών δεν πρέπει να αλλάζουν τον τρόπο με τον οποίο οι εξαιρέσεις παρουσιάζονται στο πρόγραμμα. Η ιδιότητα της ροής δεδομένων αφορά την πραγματική ροή τιμών δεδομένων, μεταξύ εντολών οι οποίες παράγουν αποτελέσματα και εκείνων οι οποίες τα καταναλίσκουν. Οι διακλαδώσεις κάνουν τη ροή δεδομένων δυναμική, εφόσον

επιτρέπουν στην πηγή δεδομένων μιας εντολής που δίνεται, να προέλθει από πολλά σημεία. Το συμπέρασμα είναι ότι δεν αρκεί μόνο η διατήρηση των εξαρτήσεων δεδομένων, αφού μια εντολή μπορεί να εξαρτάται στα δεδομένα από πολλές προηγούμενες. Η σειρά του προγράμματος είναι αυτή που καθορίζει, ποιος προκάτοχος πραγματικά θα αποδώσει μια τιμή δεδομένων σε μια εντολή. Η σειρά του προγράμματος εξασφαλίζεται με τη διατήρηση των εξαρτήσεων ελέγχου.

1.5 – Εντοπισμός του παραλληλισμού επιπέδου εντολών

Όπως έχει προαναφερθεί, αυτό που είναι επιθυμητό σε ένα πρόγραμμα είναι η αύξηση του ποσοστού του παραλληλισμού επιπέδου εντολών που μπορεί να αξιοποιηθεί στο πρόγραμμα. Για το σκοπό αυτό πρέπει να υπάρχουν αλληλουχίες μη συσχετιζόμενων εντολών, οι οποίες μπορούν να επικαλύπτονται με κάποιο συγκεκριμένο σταθερό ρυθμό. Κατά συνέπεια θεωρούμε ότι υπάρχει ένας ακολουθιακός χρονοπρογραμματισμός, όπου πρέπει μια εξαρτώμενη εντολή να απέχει από την εντολή πηγής κατά μια συγκεκριμένη απόσταση (σε κύκλους ρολογιού). Η δυνατότητα του μεταγλωττιστή να αξιοποιήσει αυτό το χρονοπρογραμματισμό, εξαρτάται και από το ποσοστό του παραλληλισμού επιπέδου εντολών που διαθέτει το πρόγραμμα και από τις καθυστερήσεις των διαφόρων λειτουργικών μονάδων. Στη συνέχεια περιγράφονται κάποιες τεχνικές που θα βοηθήσουν στην αποκάλυψη του παραλληλισμού επιπέδου εντολών.

Α) <u>Ξετύλιγμα βρόχου (Loop unrolling)</u>

Το ξετύλιγμα βρόχου **[5]**, είναι ένας αναδομητικός μετασχηματισμός, ο οποίος μεταβάλλει τη δομή του βρόχου χωρίς να επηρεάζει τη σχετική σειρά των λειτουργιών μέσα σε αυτόν. Αποτελεί μια μέθοδο αύξησης του αριθμού των εντολών επεξεργασίας, αναλογικά με τον αριθμό των εντολών διακλάδωσης και τις επιπρόσθετες εντολές στο βρόχο. Το ξετύλιγμα απλώς αντιγράφει το σώμα του βρόχου για αριθμό φορών ίσο με τον παράγοντα ξετυλίγματος (unroll factor: u). Ο νέος βρόχος θα έχει βήμα u αντί για 1, προσαρμόζοντας τον κώδικα τερματισμού του βρόχου. Επειδή απαλείφει κάποιες εντολές διακλάδωσης, το ξετύλιγμα επιτρέπει τον χρονοπρογραμματισμό εντολών από διαφορετικές επαναλήψεις. Σε αυτή την περίπτωση μπορεί να εξαλειφθεί η ανάσχεση της χρήσης δεδομένων, χρησιμοποιώντας και άλλες ανεξάρτητες εντολές μέσα στο σώμα του βρόχου. Έτσι για το λόγο αυτό θέλουμε να γίνει χρήση διαφορετικού καταχωρητή για την ίδια μεταβλητή σε κάθε διαφορετική επανάληψη, αυξάνοντας την απαιτούμενη χρήση καταχωρητών. Τα τμήματα κώδικα που ακολουθούν παρουσιάζουν από τη μια έναν αρχικό βρόχο και από

την άλλη, τον τρόπο με τον οποίο ο αρχικός βρόχος υποβάλλεται στο μετασχηματισμό του ξετυλίγματος.

ΑΡΧΙΚΟΣ ΒΡΟΧΟΣ

```
for (i=2; i<= n-1; i++){
    a[i]=a[i]-a[i-1]*a[i+1];
}
```

ΠΑΡΑΓΟΝΤΑΣ ΞΕΤΥΛΙΓΜΑΤΟΣ:2

```
for (i=2; i<= n-2; i+=2){
    a[i]=a[i]+a[i-1]*a[i+1];
    a[i+1]=a[i+1]+a[i]*a[i+2];
}
if (mod (n-2),2)==1){
    a[n-1]=a[n-1]+a[n-2]*a[n];
}
```

Σε ορισμένες περιπτώσεις μη-στατικών βρόχων, το ανώτατο μέγεθος του βρόχου συνήθως δεν είναι γνωστό. Υποθέτουμε ότι έχουμε το σημείο n και θέλουμε να δημιουργήσουμε k αντίγραφα του σώματος του βρόχου. Αντί για έναν μόνο ξετυλιγμένο βρόχο, παράγουμε ένα ζευγάρι από διπλούς διαδοχικούς βρόχους. Το πρώτο ζευγάρι εκτελεί (n **mod**k) φορές και έχει το σώμα του αρχικού βρόχου. Το δεύτερο ζευγάρι είναι το ξετυλιγμένο σώμα το οποίο περικλείεται σε έναν εξωτερικό βρόχο που επαναλαμβάνεται (n/k) φορές. Για μεγάλες τιμές του n, ο περισσότερος χρόνος εκτέλεσης θα καταναλωθεί στο σώμα του ξετυλιγμένου βρόχου. Το όφελος που έχουμε από το μετασχηματισμό "Loop Unrolling" είναι μεγάλο, διότι αποκαλύπτει περισσότερους υπολογισμούς οι οποίοι μπορούν να χρονοπρογραμματιστούν, για να εξαλειφθούν οι ανασχέσεις για τη χρήση δεδομένων. Βέβαια αυτή η τεχνική έχει και κάποιους περιορισμούς:

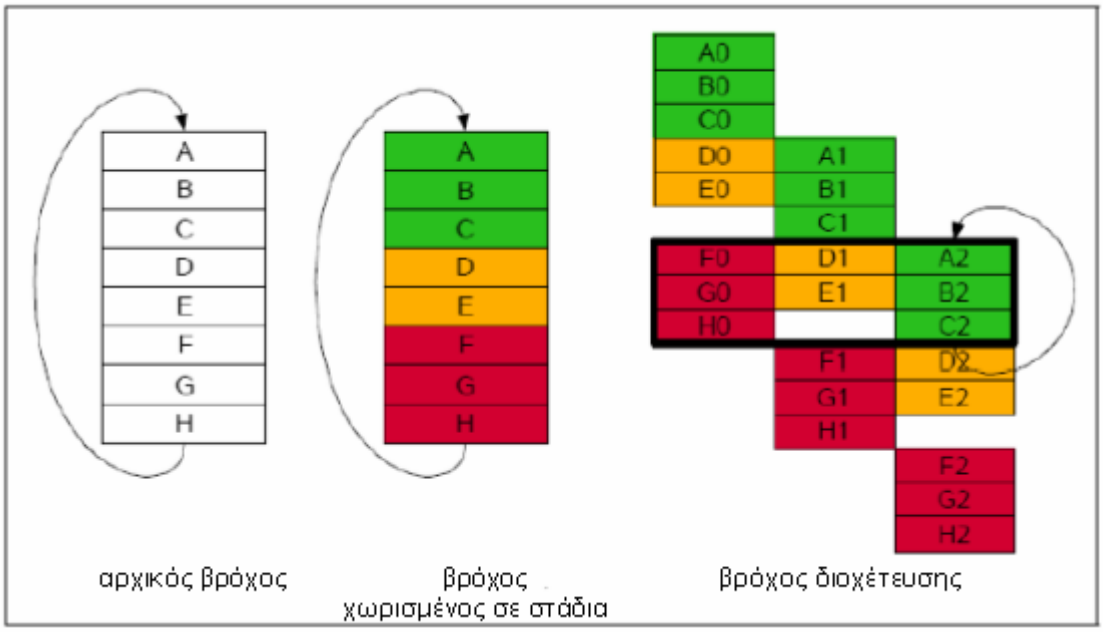
- η μείωση του κέρδους στις εντολές ανακύκλωσης
- ο περιορισμός στο μέγεθος του κώδικα, καθώς μεγαλύτεροι βρόχοι όταν ξετυλίγονται καταλαμβάνουν περισσότερο χώρο στην μνήμη
- η πιθανή έλλειψη καταχωρητών, ως ένα φυσικό επακόλουθο της διαδικασίας του χρονοπρογραμματισμού

Β) <u>Λογισμική Διοχέτευση (Software Pipelining)</u>

Πρόκειται για μια τεχνική αναδιοργάνωσης βρόχων, που περιγράφει μια οικογένεια από αλγορίθμους που εφαρμόζονται σε κυκλικές περιοχές κώδικα (αναπαράσταση στο σχήμα 1.3).

Οι αλγόριθμοι αυτοί υποδιαιρούν ένα βρόχο σε επιμέρους στάδια, τα οποία εκτελούνται περιλαμβάνοντας διαφορετικές επαναλήψεις σε παράλληλη επεξεργασία. Στην ουσία οι εντολές του βρόχου χωρίζονται σε στάδια διοχέτευσης. Πρόκειται για εντολές που επιλέγονται από διαφορετικές επαναλήψεις του αρχικού βρόχου χωρίς να υπάρχει κάποιο ξετύλιγμα (unroll) για το βρόχο. Ένας βρόχος λογισμικής διοχέτευσης θα μπορούσε να περιέχει μια εντολή **load,** μια εντολή **add** και μια εντολή **store,** καθεμία από τις οποίες μπορεί να προέρχεται από διαφορετική επανάληψη. Υπάρχει επίσης κάποιος

κώδικας εκκίνησης ο οποίος είναι αναγκαίος πριν την εκκίνηση του βρόχου, όπως και κάποιος κώδικας εκκαθάρισης ο οποίος είναι αναγκαίος μετά την ολοκλήρωση του βρόχου.

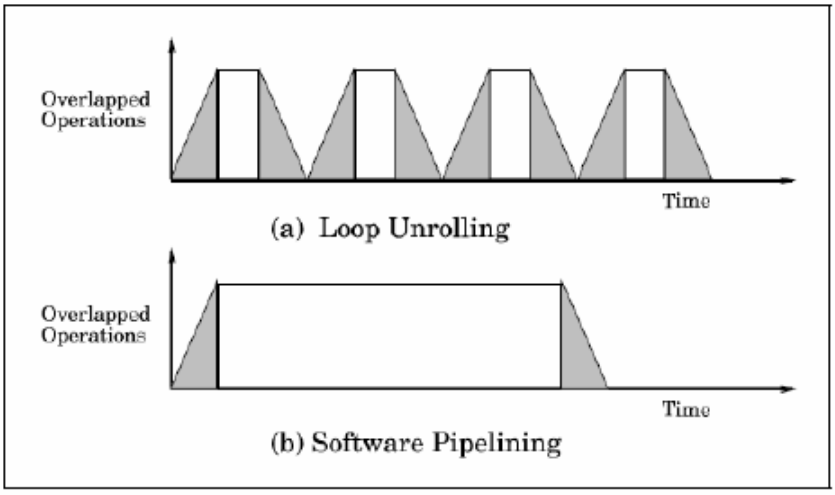


Σχήμα 1.3: *Παράδειγμα λογισμικής διοχέτευσης*

Οι εντολές ξεκινώντας από το πρώτο στάδιο, προωθούνται διαμέσου μεταγενέστερων σταδίων διάρκειας ενός κύκλου μηχανής. Μέσω αυτής της τεχνικής είναι εφικτή η ταυτόχρονη εκτέλεση πολλαπλών εντολών, με την επεξεργασία διαφορετικών μερών από διαφορετικές εντολές σε παραλληλία. Η τεχνική της διοχέτευσης μπορεί να προκαλέσει «κινδύνους δεδομένων» (Data Hazards), που έχουν ως συνέπεια την ύπαρξη κάποιων καθυστερήσεων. Αυτό συμβαίνει όταν μια εντολή εξαρτάται από το αποτέλεσμα μιας προηγούμενης εντολής, η οποία δεν είναι ακόμα διαθέσιμη. Με διάφορες αναδιατάξεις εντολών γίνεται προσπάθεια ώστε να ελαχιστοποιηθούν τα παραπάνω προβλήματα. Η κύρια μέθοδος χρονοπρογραμματισμού που παρατηρείται εδώ, είναι αυτή του ακέραιου υπολοίπου (modulo scheduling).

Πολλές φορές είναι αναγκαίος ο συνδυασμός των τεχνικών "Software Pipelining" και "Loop Unrolling". Ίσως μπορούμε να σκεφτούμε την τεχνική της λογισμικής διοχέτευσης σαν ένα συμβολικό ξετύλιγμα βρόχου. Στην πραγματικότητα κάποιοι από τους αλγόριθμους της τεχνικής της διοχέτευσης, χρησιμοποιούν αλγόριθμους ξετυλίγματος βρόχου για να εξακριβώσουν πως μπορεί να επιτευχθεί διοχέτευση λογισμικού στο βρόχο. Το κύριο πλεονέκτημα της λογισμικής διοχέτευσης είναι ότι απαιτεί μικρότερο μέγεθος κώδικα. Επίσης μειώνει το χρόνο κατά τον οποίο ο βρόχος δεν εκτελείται με τη μέγιστη ταχύτητα, σε μια μόνο φορά στην εκκίνηση και τον τερματισμό του. Στην πράξη η μεταγλώττιση που χρησιμοποιεί λογισμική διοχέτευση είναι δύσκολη

για αρκετούς λόγους: πολλοί βρόχοι απαιτούν σημαντική μετατροπή πριν χρησιμοποιηθεί η τεχνική "Software Pipelining". Το θέμα αφορά την πολυπλοκότητα που προκύπτει, καθώς ορισμένοι εμπορικοί επεξεργαστές έχουν αρκετό επιπρόσθετο υλικό για την υποστήριξη της λογισμικής διοχέτευσης. Το υλικό αυτό όμως δεν ελαττώνει την ανάγκη χρήσης ενός πολύπλοκου μεταγλωττιστή, ή την ανάγκη λήψης δύσκολων αποφάσεων για την καλύτερη μεταγλώττιση του βρόχου. Στο σχήμα 1.4 που ακολουθεί, παριστάνεται μια ενδεικτική σύγκριση ανάμεσα στις δυο τεχνικές, αυτής του ξετυλίγματος βρόχου και της λογισμικής διοχέτευσης, με επικαλυπτόμενες λειτουργίες να εκτελούνται συναρτήσει του χρόνου.



Σχήμα 1.4: *Σύγκριση "Loop Unrolling" και "Software Pipelining"*

Γ) <u>Χρονοπρογραμματισμός ίχνους (Trace Scheduling)</u>

Η τεχνική του χρονοπρογραμματισμού ίχνους είναι χρήσιμη σε επεξεργαστές, στους οποίους η εκτέλεση υπό συνθήκη είναι ακατάλληλη ή δεν υποστηρίζεται και το ξετύλιγμα βρόχου μπορεί να μην αρκεί από μόνο του, για να αποκαλύψει παραλληλισμό μεταξύ εντολών ώστε να παραμείνει απασχολημένος ο επεξεργαστής. Στον χρονοπρογραμματισμό ίχνους υπάρχουν δυο βήματα. Το πρώτο βήμα είναι αυτό της επιλογής ίχνους, κατά το οποίο γίνεται προσπάθεια εύρεσης μιας πιθανής αλληλουχίας (sequence) βασικών ακολουθιών, οι λειτουργίες των οποίων θα τοποθετηθούν μαζί σε ένα μικρότερο αριθμό εντολών. Η αλληλουχία αυτή ονομάζεται ίχνος. Το ξετύλιγμα του βρόχου χρησιμοποιείται για να παραχθούν μακριά ίχνη, αφού υπάρχει μεγάλη πιθανότητα οι διακλαδώσεις των βρόχων να είναι επιτυχείς. Επιπροσθέτως με τη χρήση της στατικής πρόβλεψης διακλάδωσης, άλλες διακλαδώσεις υπό συνθήκη επιλέγονται ως επιτυχείς ή ανεπιτυχείς, έτσι ώστε η ακολουθία ίχνους που παράγεται να είναι γραμμική από τη συνένωση πολλών βασικών ακολουθιών. Το δεύτερο βήμα αφορά τη διαδικασία συμπίεσης ίχνους (trace compaction), όπου ένα επιλεγμένο ίχνος συμπιέζεται σε

μικρότερο αριθμό εκτεταμένων εντολών. Είναι στην ουσία ο χρονοπρογραμματισμός του κώδικα.

Το πλεονέκτημα της προσέγγισης του χρονοπρογραμματισμού ίχνους, είναι ότι απλοποιεί τις αποφάσεις που σχετίζονται με την καθολική μετακίνηση κώδικα. Όταν ο κώδικας μετακινείται πέρα από τα σημεία εισόδου και εξόδου της ακολουθίας ίχνους, συχνά απαιτείται ένας επιπρόσθετος κώδικας αποκατάστασης στο σημείο εισόδου ή εξόδου. Αν και έχει εφαρμοστεί με επιτυχία σε επιστημονικούς κώδικες με τους συχνά εμφανιζόμενους βρόχους τους, ο χρονοπρογραμματισμός ίχνους είναι αμφίβολο εάν αποτελεί την πιο κατάλληλη προσέγγιση για πιο πολύπλοκα προγράμματα ή για προγράμματα, με λιγότερο πολυσύχναστους βρόχους. Σε τέτοια προγράμματα, το σημαντικό επιπρόσθετο κόστος του κώδικα αποκατάστασης μπορεί να καταστήσει τον χρονοπρογραμματισμό ίχνους μη ελκυστικό ή να κάνει την αποτελεσματική χρήση του, εξαιρετικά πολύπλοκη για τον μεταγλωττιστή.

1.6 – Βασικές έννοιες γράφου και μετασχηματισμοί

Προτού επεκταθούμε περισσότερο σε ότι έχει να κάνει με τις βελτιστοποιήσεις στους μεταγλωττιστές, θα ήταν προτιμότερο να αναφερθούμε σε κάποιες βασικές έννοιες οι οποίες θα χρησιμοποιηθούν ευρέως στη συνέχεια:

⇒ **Γράφος**

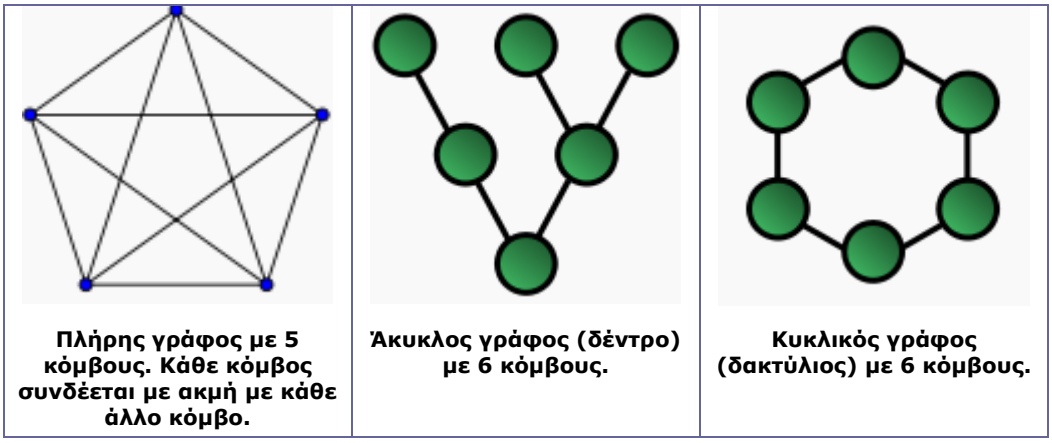
Ο γράφος (graph) στον απλούστερο ορισμό του **[6]**, είναι η οπτική αναπαράσταση των σχέσεων που αναπτύσσουν ορισμένες ποσότητες, σχεδιασμένες σε σχέση με ένα σύνολο αξόνων. Ένας άλλος ορισμός που κινείται στο ίδιο εννοιολογικό πλαίσιο της οπτικής αναπαράστασης, αναγνωρίζει το γράφο ως απεικόνιση αποτελούμενη από ένα σύνολο σημείων (κορυφών ή κόμβων), που συνδέονται με γραμμές (ακμές).

Μια άλλη εκδοχή **[7]** για το γράφο, τον αναφέρει ως ένα σύνολο από κόμβους (κορυφές) που ενώνονται μεταξύ τους με ακμές, και ορίζεται από τον τρόπο με τον οποίο συνδέονται οι κορυφές (κόμβοι). Αν οι ακμές προσανατολίζονται οριζόμενες από διατεταγμένα ζεύγη κόμβων, τότε ο γράφος αποκαλείται κατευθυνόμενος (Directed Graph ή Digraph). Αν οι ακμές δεν είναι προσανατολισμένες αλλά είναι οριζόμενες απλώς από διμελή σύνολα χωρίς διάταξη, τότε αποκαλείται μη κατευθυνόμενος (Undirected). Επιπλέον στοιχεία για τον ορισμό ενός γράφου είναι η σύνδεση των ακμών του με κάποια αξία, οπότε αποκαλείται σταθμισμένος (Weighted). Με τη σειρά του, πλήρης (Complete) αποκαλείται ο γράφος που περιέχει ακμές για κάθε ζεύγος κόμβων, αραιός (Sparse) εκείνος που περιέχει λίγες ακμές ή αντίστροφα πυκνός (Dense).

Στους κατευθυνόμενους ή προσανατολισμένους γράφους, οι ακμές απεικονίζονται διανυσματικά.

Προτού συνεχίσουμε με το σχήμα 1.5 όπου απεικονίζονται χαρακτηριστικά παραδείγματα γράφων, θα αναφερθούμε σύντομα στον ορισμό του δέντρου (tree) **[8]**. Πρόκειται στην ουσία για ένα μη κατευθυνόμενο απλό γράφο G, ο οποίος ικανοποιεί οποιαδήποτε από τις παρακάτω ισοδύναμες συνθήκες:

- ο G είναι συνδεδεμένος και δεν έχει καθόλου κύκλους.
- ο G δεν έχει καθόλου κύκλους, και ένας απλός κύκλος δημιουργείται εάν οποιαδήποτε ακμή προστίθεται στον G.
- ο G είναι συνδεδεμένος, και δεν είναι πλέον συνδεδεμένος εάν οποιαδήποτε ακμή αφαιρεθεί από αυτόν.
- ο G είναι συνδεδεμένος και ο πλήρης γράφος τριών κόμβων K_3, δεν είναι υποδεέστερος του G.
- Οποιοιδήποτε δυο κόμβοι στον G, μπορούν να είναι συνδεδεμένοι από ένα μοναδικό απλό μονοπάτι.



Πλήρης γράφος με 5 κόμβους. Κάθε κόμβος συνδέεται με ακμή με κάθε άλλο κόμβο.	Άκυκλος γράφος (δέντρο) με 6 κόμβους.	Κυκλικός γράφος (δακτύλιος) με 6 κόμβους.

Σχήμα 1.5: *Παραδείγματα οπτικής αναπαράστασης γράφου*

Εκτός από μια απλή οπτική αναπαράσταση, ένας γράφος μπορεί να εκφραστεί και με τυπολογικά παραδείγματα. Κατά συνέπεια ως μαθηματική έκφραση, ο ορισμός του μη κατευθυνόμενου γράφου έχει ως εξής:
ο γράφος G είναι ένα διατεταγμένο ζεύγος $G =< V(G), E(G) >$ όπου:

→ $V(G) = \{v1, v2, ..., v_n\}$ το σύνολο των κορυφών

→ $E(G) = \{e1, e2, ..., e_m\}$ το σύνολο των ακμών

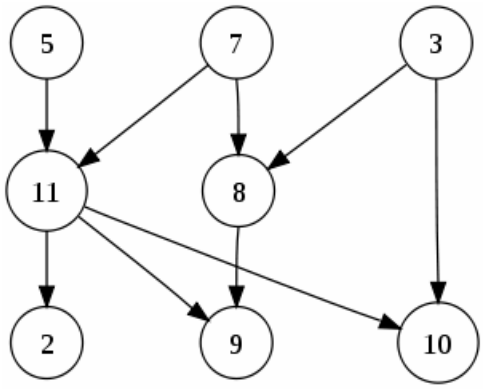
Στην προκειμένη περίπτωση κάθε ακμή είναι ένα διμελές σύνολο αποτελούμενο από δυο κορυφές, οι οποίες αποκαλούνται τερματικές κορυφές (κόμβοι) και δεν είναι απαραίτητα διαφορετικές μεταξύ τους. Ο ορισμός του κατευθυνόμενου γράφου παρουσιάζει την ίδια μαθηματική έκφραση, όμως η κύρια διαφορά ανάμεσα σε έναν μη κατευθυνόμενο και έναν κατευθυνόμενο

γράφο, είναι ότι στην πρώτη περίπτωση έχουμε διμελές σύνολο ενώ στη δεύτερη διατεταγμένο ζεύγος.

Στη Θεωρία Γράφων διακρίνουμε δύο κύρια προβλήματα που εμφανίζονται, και σχετίζονται με την εύρεση του μικρότερου (*Shortest path problem*) **[9]** ή του μεγαλύτερου (*Longest path problem*) **[10]** μονοπατιού, σε έναν δεδομένο γράφο. Στην πρώτη περίπτωση το πρόβλημα αφορά την εύρεση ενός μονοπατιού ανάμεσα σε δυο κόμβους στο γράφο, ώστε το άθροισμα των βαρών των ακμών που το αποτελούν να είναι το ελάχιστο δυνατό. Από την άλλη πλευρά, το πρόβλημα του μεγαλύτερου μονοπατιού σχετίζεται με την εύρεση ενός απλού μονοπατιού μέγιστου μήκους σε ένα δεδομένο γράφο. Το μονοπάτι καλείται «απλό» εάν δεν έχει καθόλου επαναλαμβανόμενους κόμβους. Σε αντίθεση με το πρόβλημα του μικρότερου μονοπατιού, το οποίο ψάχνει για τη μικρότερη διαδρομή ανάμεσα σε δυο κόμβους και μπορεί να λυθεί σε πολυωνυμικό χρόνο – σε γράφους χωρίς αρνητικά βάρη κύκλων – η έκδοση της απόφασης αυτού του προβλήματος είναι NP-πλήρης, πράγμα που σημαίνει ότι η βέλτιστη λύση είναι δύσκολο να βρεθεί σε πολυωνυμικό χρόνο. Η κύρια έκδοση της απόφασης, αναζητά εάν ο γράφος περιέχει ένα «απλό» μονοπάτι μήκους μεγαλύτερου ή ίσου με k, όπου το μήκος ενός μονοπατιού έχει οριστεί να είναι ο αριθμός των κόμβων σε αυτό.

Κατευθυνόμενος άκυκλος γράφος

Στα μαθηματικά και την επιστήμη των υπολογιστών, ένας κατευθυνόμενος άκυκλος γράφος (Directed Acyclic Graph – DAG) **[11]** είναι ένας κατευθυνόμενος γράφος με μη κατευθυνόμενους κύκλους. Οι κατευθυνόμενοι άκυκλοι γράφοι (DAGs) μπορούν να χρησιμοποιηθούν για τη μοντελοποίηση πολλών διαφορετικών δομών δεδομένων, όπως για παράδειγμα μια συλλογή από εργασίες που πρέπει να τακτοποιηθούν σε μια σειρά – με τον περιορισμό ότι κάποιες εκτελούνται νωρίτερα από κάποιες άλλες – θα μπορούσε να αντιπροσωπευτεί με έναν DAG, με μια κορυφή για κάθε εργασία και μια ακμή για κάθε περιορισμό. Στο σχήμα 1.6 παρουσιάζεται ένα ενδεικτικό παράδειγμα κατευθυνόμενου άκυκλου γράφου.



Σχήμα 1.6: *Κατευθυνόμενος Άκυκλος Γράφος*

Ένας γράφος μπορεί εύκολα να παρασταθεί με δυο διαφορετικούς τρόπους, οι οποίοι προϋποθέτουν ένα μονοσήμαντο σύστημα αρίθμησης των κόμβων **[7]** :

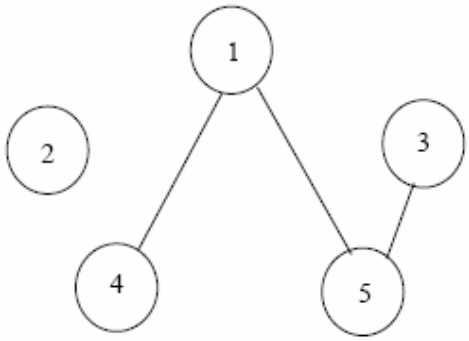
1. με πίνακα γειτνίασης
2. με λίστα γειτνίασης

Τα χαρακτηριστικά του πίνακα είναι ότι υλοποιείται και συντηρείται πιο εύκολα, με συνέπεια να προσφέρεται κυρίως για πυκνούς γράφους. Οι λίστες από την άλλη πλευρά προσφέρονται για αραιούς γράφους, όπως επίσης προσφέρονται για αλγορίθμους που η πολυπλοκότητά τους εξαρτάται από το πλήθος των ακμών.

Πίνακας γειτνίασης (Adjacency matrix)

Ένας γράφος έχει μια πολύ κομψή και εύχρηστη αναπαράσταση με ένα δισδιάστατο πίνακα, ο οποίος ονομάζεται πίνακας γειτνίασης. Σε ένα τέτοιο πίνακα η θέση (i, j) περιέχει το στοιχείο 0, εάν δεν υπάρχει σύνδεση μεταξύ των κορυφών i και j, ενώ στην περίπτωση που υπάρχει σύνδεση η θέση (i, j) περιέχει το στοιχείο 1.

Θεωρούμε λοιπόν τον γράφο του σχήματος 1.7, με τον αντίστοιχο $5x5$ πίνακα γειτνίασης. Παρατηρώντας την πρώτη γραμμή του πίνακα, βλέπουμε ότι η κορυφή 1 συνδέεται με τις κορυφές 4 και 5. Στην περίπτωση αυτή λέμε ότι ο βαθμός (degree) της κορυφής είναι ίσος με 2, καθώς συνδέεται με 2 κορυφές. Από τη δεύτερη γραμμή συμπεραίνουμε, ότι η κορυφή 2 δε συνδέεται με καμία από τις υπόλοιπες κορυφές, πρόκειται δηλαδή για απομονωμένη κορυφή (κορυφή βαθμού 0). Ένα τελευταίο συμπέρασμα είναι ότι ο πίνακας είναι συμμετρικός, δηλαδή εάν υπάρχει 1 (0) στη θέση (i, j), τότε υπάρχει 1 (0) και στη θέση (j, i) (δηλαδή, οι συνδέσεις είναι αμφίδρομες).



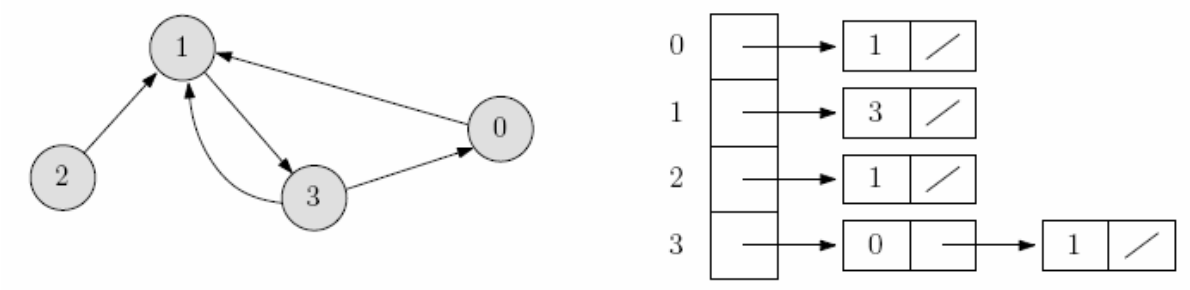
	1	2	3	4	5
1	**0**	**0**	**0**	**1**	**1**
2	**0**	**0**	**0**	**0**	**0**
3	**0**	**0**	**0**	**0**	**1**
4	**1**	**0**	**0**	**0**	**0**
5	**1**	**0**	**1**	**0**	**0**

Σχήμα 1.7: *Μη κατευθυνόμενος γράφος και αντίστοιχος πίνακας γειτνίασης*

Λίστα γειτνίασης (Adjacency list)

Μία λίστα γειτνίασης γράφου είναι ένας πίνακας από λίστες, μία για κάθε κορυφή, όπου η $j-οστή$ λίστα περιέχει μια συνδεδεμένη λίστα των κόμβων οι

οποίοι συνδέονται με την $j-$*οστή* κορυφή. Με άλλα λόγια κάθε κόμβος συσχετίζεται με μια συνδεδεμένη λίστα γειτνίασης, η οποία περιέχει τα ονόματα των άλλων κόμβων με τους οποίους συνδέεται ο συγκεκριμένος κόμβος. Έτσι σε ένα γράφο με v κόμβους, οι λίστες για όλους τους κόμβους ν μπορούν να ξεκινούν από έναν μονοδιάστατο πίνακα μήκους ν ή από μια άλλη συνδεδεμένη λίστα.
Θεωρούμε τον παρακάτω γράφο $G(V,E)$ (σχήμα 1.8), με μια αναπαράσταση που περιέχει ένα μονοδιάστατο πίνακα A, με μια θέση για κάθε κόμβο. Παρατηρούμε ότι κάθε θέση του πίνακα A, αποτελεί μια λίστα ακμών που εξέρχονται από τον αντίστοιχο κόμβο.



Σχήμα 1.8: *Κατευθυνόμενος γράφος και αντίστοιχη λίστα γειτνίασης*

⇒ **<u>Απόσταση επεξεργασίας γράφου (Graph Edit Distance – GED)</u>**
Πρόκειται στην ουσία για ένα σημαντικό τρόπο μέτρησης της ομοιότητας ανάμεσα σε ζεύγη γράφων, με ανεκτικότητα λάθους σε ανακριβή αντιστοίχιση γράφου, και έχει ευρεία εφαρμογή στην ανάλυση και την αναγνώριση προτύπων. Η απόσταση επεξεργασίας γράφου (GED) δύο γράφων G1 και G2, δίνεται από τον αριθμό βημάτων ελάχιστου κόστους που χρειάζεται να εφαρμοστούν ώστε από τον G1 να λαμβάνεται ο G2. Το κόστος καθορίζεται με βάση κάποια μετρική συνάρτηση η οποία εξαρτάται από το συγκεκριμένο πεδίο εφαρμογής. Το πρόβλημα υπολογισμού της απόστασης επεξεργασίας γράφου είναι σε γενικές γραμμές δυσεπίλυτο (NP-Hard).

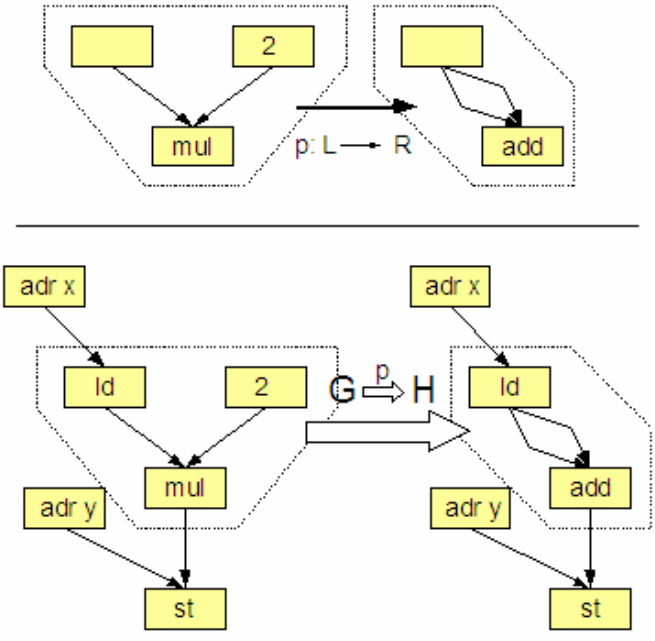
Στη βιβλιογραφία έχουν προταθεί τρεις μέθοδοι υπολογισμού των ανώτερων και κατώτερων ορίων της GED, μεταξύ δυο γράφων σε πολυωνυμικό χρόνο **[12]**. Η εφαρμογή αυτών των μεθόδων, οδηγεί δυο αλγόριθμους στην εκτέλεση διαφορετικού είδους αναζήτησης γράφου σε βάσεις δεδομένων γράφου. Τα αποτελέσματα που προκύπτουν δείχνουν, ότι αυτές οι μέθοδοι επιτυγχάνουν ικανοποιητική δυνατότητα κλιμάκωσης από την άποψη τόσο του αριθμού των γράφων όσο και του μεγέθους αυτών. Η αποτελεσματικότητα αυτών των αλγορίθμων επιβεβαιώνει επίσης τη χρησιμότητα των ορίων, στο φιλτράρισμα και την αναζήτηση των γράφων.

⇒ **Μετασχηματισμός γράφου (Graph Transformation)**

Αναφερόμενοι στην έννοια του μετασχηματισμού γράφου, εννοούμε τη διαδικασία εκείνη στην οποία υπόκειται ένας γράφος όπου ένα σύνολο κανόνων μετασχηματισμού, μπορεί να προσθέτει ή να αφαιρεί κόμβους και ακμές από αυτόν. Αυτό έχει ως συνέπεια, η επιπλέον πληροφορία που μεταφέρεται από τις ακμές και τους κόμβους να δίνει διαφορετικό νόημα στον γράφο που θα προκύψει. Μια από τις κύριες συνεισφορές του μετασχηματισμένου γράφου η οποία μας ενδιαφέρει άμεσα και θα δούμε σε επόμενο κεφάλαιο, είναι ότι η απλή εκτέλεση βασικού χρονοπρογραμματισμού σε αυτόν μπορεί να οδηγήσει σε ένα βέλτιστο πρόγραμμα.

Επανεγγραφή γράφου (Graph rewriting)

Τόσο ο μετασχηματισμός γράφου όσο και η επανεγγραφή γράφου, αφορούν την τεχνική της δημιουργίας ενός νέου γράφου από έναν ήδη υπάρχοντα, χρησιμοποιώντας ένα σύνολο κανόνων αντικατάστασης (substitution rules). Το σύστημα της επανεγγραφής γράφου **[13]** εν προκειμένω, αποτελείται από ένα σύνολο κανόνων μετασχηματισμού επανεγγραφής της μορφής $L \to R$. Το L καλείται γράφος βάσει προτύπου (ή αριστερή πλευρά του κανόνα), και το R καλείται γράφος αντικατάστασης (ή δεξιά πλευρά του κανόνα). Ένας κανόνας μετασχηματισμού εγγραφής εφαρμόζεται στο γράφο «οικοδεσπότη» (host), ψάχνοντας για μια εμφάνιση του γράφου-πρότυπο (pattern) και αντικαθιστώντας την ευρεθείσα εμφάνιση από ένα στιγμιότυπο του γράφου αντικατάστασης. Στο σχήμα 1.9 που ακολουθεί παρατηρούμε μια βελτιστοποίηση από την κατασκευή ενός μεταγλωττιστή, όπου η πράξη του πολλαπλασιασμού με το 2 αντικαθίσταται από την πράξη της πρόσθεσης.



Σχήμα 1.9: *Παράδειγμα για κανόνα μετασχηματισμού επανεγγραφής*

Μερικές φορές, η έννοια της «γραμματικής γράφου» (Graph grammar) χρησιμοποιείται ως ένα συνώνυμο του συστήματος επανεγγραφής γράφου, ειδικότερα στο πλαίσιο των επισήμων γλωσσών προγραμματισμού (formal languages). Η διαφορετική διατύπωση χρησιμοποιείται για να δώσει έμφαση στο στόχο της παράθεσης όλων των γράφων, από κάποιο γράφο εκκίνησης, π.χ. περιγράφοντας μια γλώσσα γράφου – αντί του μετασχηματισμού μιας δοθείσας κατάστασης (οικοδεσπότης γράφος), σε μια νέα κατάσταση.

1.7 – Μεταγλωττιστές και χρονοπρογραμματισμός εντολών

Ο χρονοπρογραμματισμός εντολών (Instruction Scheduling) αποτελεί μια από τις πιο σημαντικές βελτιστοποιήσεις στους μεταγλωττιστές, εξαιτίας του ρόλου του στην αύξηση της χρήσης του επεξεργαστή (Muchnick, 1997) **[14]**. Στην πραγματικότητα βελτιώνει την απόδοση του προγράμματος για τους επεξεργαστές με διοχέτευση (ενότητα 1.5). Στο σημείο αυτό εισάγεται και η έννοια του βασικού μπλοκ (Basic block), η οποία έχει ιδιαίτερη σημασία. Πρόκειται στην ουσία για ευθύγραμμα τμήματα κώδικα στα οποία συχνά εκτελείται ο χρονοπρογραμματισμός εντολών. Σε τοπικό επίπεδο το πρόβλημα έγκειται στο να βρεθεί ένας χρονοπρογραμματισμός για τις εντολές του βασικού μπλοκ, ώστε να εκτελείται σε έναν ελάχιστο αριθμό από κύκλους ρολογιού. Στην πραγματικότητα αυτό είναι ένα δισεπίλυτο, «NP-hard» πρόβλημα **[15]**.

Οι βασικότερες τεχνικές χρονοπρογραμματισμού εντολών που χρησιμοποιούνται για την επίτευξη βελτιστοποιήσεων στους μεταγλωττιστές είναι:

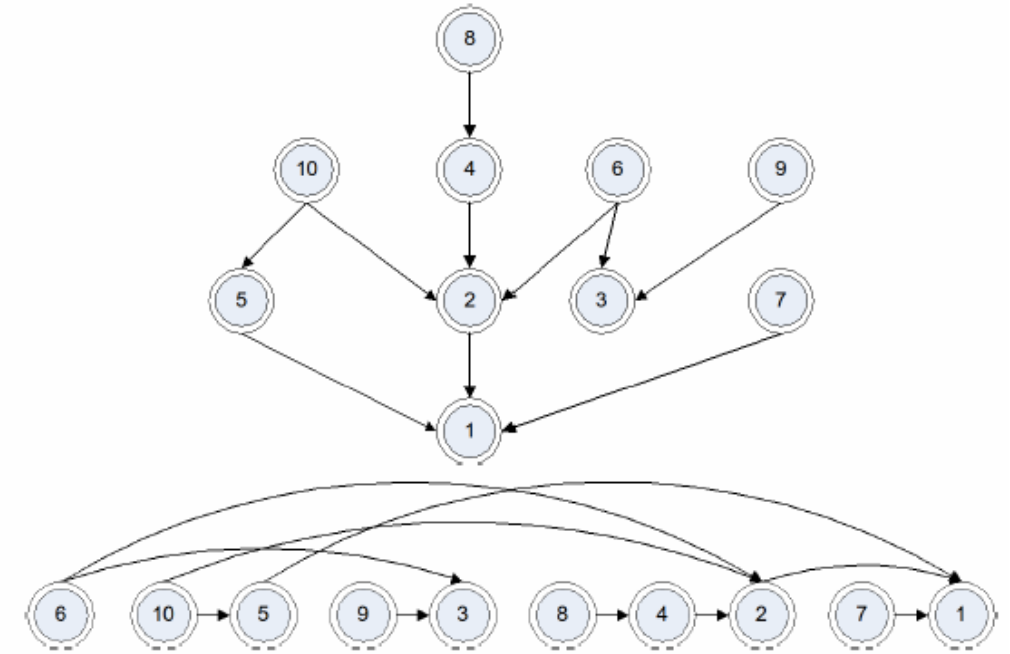
- ακολουθιακή δρομολόγηση
- χρονοπρογραμματισμός ASAP (As-Soon-As-Possible)
- χρονοπρογραμματισμός ALAP (As-Late-As-Possible)
- χρονοπρογραμματισμός λίστας (List Scheduling)

1.7.1 – Ακολουθιακή δρομολόγηση

Η τοπολογική κατάταξη (topological sorting), είναι η διαδικασία που χαρακτηρίζει την ακολουθιακή δρομολόγηση ως μια τεχνική χρονοπρογραμματισμού εντολών. Πιο συγκεκριμένα, ως τοπολογική κατάταξη ενός κατευθυνόμενου άκυκλου γράφου (DAG) ονομάζεται σύμφωνα με τη θεωρία των γράφων, η γραμμική διάταξη των κόμβων με τέτοιο τρόπο ώστε κάθε κόμβος προηγηθείς ενός κόμβου v, να προηγείται του v στην κατάταξη. Γενικά υπάρχουν περισσότερες από μια έγκυρες τοπολογικές κατατάξεις ενός κατευθυνόμενου άκυκλου γράφου.

Πιο αυστηρά μπορούμε να ορίσουμε την τοπολογική κατάταξη ενός DAG, θεωρώντας το γράφο $G(V,E)$ με V το σύνολο των κόμβων και E το σύνολο των ακμών ως μια γραμμική αλληλουχία των κόμβων V, έτσι ώστε αν ο G περιέχει ακμή $(u,v) \in E$, τότε να ισχύει $\pi(u) < \pi(v)$ όπου $\pi(x)$ είναι η θέση στην οποία βρίσκεται ο κόμβος x στην κατάταξη. Με άλλα λόγια ο u να εμφανίζεται πριν τον v στην κατάταξη **[16]**.

Το πρόβλημα της εύρεσης (ή της διαπίστωσης μη ύπαρξης) μιας τοπολογικής κατάταξης των στοιχείων του V, έτσι ώστε όλα τα διατεταγμένα ζεύγη στοιχείων του E να έχουν το πρώτο στοιχείο πριν το δεύτερο στοιχείο στη διάταξη, ονομάζεται πρόβλημα "*Τοπολογικής Κατάταξης*" των στοιχείων του V ως προς τη διμελή σχέση $E \subseteq V \times V$ **[17]**.



Σχήμα 1.10: *Τοπολογική κατάταξη*

Στο σχήμα 1.10 παρατηρούμε, ότι η προσθήκη της ακμής (1,10) δε θα συνέβαλλε στην ύπαρξη τοπολογικής κατάταξης για το γράφο, γιατί τότε αυτός θα είχε κύκλο $<10,2,1,10>$. Συνεπώς, δεν μπορεί να έχει τοπολογική κατάταξη αφού από τη μια ζητείται το 1 να έπεται του 10, και από την άλλη να προηγείται αυτού. Το πρόβλημα της τοπολογικής κατάταξης εμφανίζεται για παράδειγμα στη δρομολόγηση διεργασιών οι οποίες έχουν εξαρτήσεις προτεραιότητας εκτέλεσης, σε μια μηχανή: εάν έχουμε να εκτελέσουμε ένα σύνολο από διεργασίες, με την προϋπόθεση ότι κάποιες από αυτές προηγούνται (στην εκτέλεσή τους) κάποιων άλλων, τότε αυτοί οι περιορισμοί ορίζουν έναν κατευθυνόμενο γράφο. Οποιαδήποτε τοπολογική κατάταξη του γράφου αυτού (εφόσον υπάρχει) μας υποδεικνύει ένα τρόπο εκτέλεσης των

λειτουργιών αυτών, έτσι ώστε καμία διεργασία να μην εκτελείται πριν από εκείνες που πρέπει να προηγούνται από αυτήν.

1.7.2 – Χρονοπρογραμματισμός ASAP (As-Soon-As-Possible)

Ένας σημαντικός αλγόριθμος χρονοπρογραμματισμού είναι ο "Όσο το Δυνατόν πιο Σύντομα" (As-Soon-As-Possible – ASAP) **[4][17]**, ο οποίος χαρακτηρίζεται από πολυπλοκότητα $O(|V|+|E|)$, όπου V το σύνολο των κόμβων και E το σύνολο των ακμών για ένα δεδομένο γράφο $G(V,E)$.

Οι κόμβοι του γράφου ροής δεδομένων (DFG) υπόκεινται σε τοπολογική κατάταξη, πράγμα που διασφαλίζει ότι όταν κάθε κόμβος προσπελαύνεται όλοι οι προκάτοχοί του προσπελαύνονται επίσης. Ο χρόνος γνωστοποίησης ενός κόμβου είναι ίσος με το συντομότερο δυνατό χρόνο που μια χρονική ακολουθία κόμβων μπορεί να αρχίσει, χωρίς να παραβιαστούν οι περιορισμοί προτεραιότητας. Το κύριο χαρακτηριστικό του αλγορίθμου ASAP, είναι ότι επιλύει βέλτιστα το πρόβλημα του χρονοπρογραμματισμού χωρίς περιορισμούς. Από τη στιγμή που ο γράφος ροής δεδομένων είναι ένας DAG, μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο μεγαλύτερου μονοπατιού για την εύρεση της μεγαλύτερης διαδρομής ανάμεσα στους αρχικούς και τους τελικούς κόμβους που τελικά θα οδηγήσει σε χρονοπρογραμματισμό. Στη συνέχεια παρατίθεται κώδικας ενδεικτικός του χρονοπρογραμματισμού ASAP:

```
ASAP(G(V,E))
προγραμμάτισε u0 αρχικοποιώντας t0(S)=0;
επανέλαβε
{
επέλεξε μια κορυφή ui με προγραμματισμένους προκατόχους;
προγραμμάτισε ui αρχικοποιώντας ti(S)=max_{j:(uj,ui)∈E}(tj(S)+dj);
}
μέχρι (un να έχει χρονοπρογραμματιστεί);
επέστρεψε (ti(S) για όλα τα i);
```

Βάσει αυτού του ψευδοκώδικα (pseudocode), ο αλγόριθμος διατρέχει τη ροή δεδομένων σε τοπολογική διάταξη ξεκινώντας από την πρώτη είσοδο. Σε κάθε τέτοια είσοδο ανατίθεται η χρονική στιγμή t_0 ίση με μηδέν (0). Ένας κόμβος u_i υπόκειται σε επεξεργασία, μόνο όταν όλοι οι προκάτοχοί του έχουν προγραμματιστεί. Ο χρόνος έναρξης t_i για τον κόμβο u_i, είναι ο μέγιστος (max) ανάμεσα στους προκατόχους του.

1.7.3 – Χρονοπρογραμματισμός ALAP (As-Late-As-Possible)

Όπως ο αλγόριθμος ASAP δίνει το συντομότερο δυνατό χρόνο χρονοπρογραμματισμού μιας χρονικής ακολουθίας κόμβων, ο αλγόριθμος "Όσο το Δυνατό πιο Αργά" (As-Late-As-Possible – ALAP) **[5][18]** δίνει τον πιο καθυστερημένο αντίστοιχο χρόνο χρονοδρομολόγησης. Δίνοντας ένα μέγιστο αριθμό κύκλων εκτέλεσης, ο αλγόριθμος ALAP αρχίζει πραγματοποιώντας μια αντίστροφη τοπολογική κατάταξη των κόμβων του γράφου ροής δεδομένων (DFG). Ο χρόνος γνωστοποίησης κάθε κόμβου, καθορίζεται βρίσκοντας τον ελάχιστο χρόνο γνωστοποίησης όλων των διαδόχων κόμβων της χρονικής ακολουθίας. Υπάρχει βέβαια ένας αδρανής χρόνος (slack) μιας ακολουθίας κόμβων, που πρόκειται για τη διαφορά μεταξύ του χρόνου γνωστοποίησης για το χρονοπρογραμματισμό από τους αλγόριθμους ALAP και ASAP. Η παράμετρος "slack" είναι μια ιδέα της κινητικότητας (mobility) που έχει μια πράξη για να χρονοδρομολογηθεί. Το πρόβλημα του χρονοπρογραμματισμού γίνεται NP-πλήρες (NP-complete), όταν προστίθενται περιορισμοί στους πόρους. Ο αλγόριθμος ALAP συγκεκριμένα, επιλύει το πρόβλημα του χρονοπρογραμματισμού χωρίς περιορισμούς πόρων και υπό περιορισμό χρόνου. Χρησιμοποιεί την έννοια του ορίου της καθυστέρησης (latency bound) της μορφής $\overline{\lambda} = t_n(S) - t_0(S)$. Ένα δείγμα κώδικα για τον αλγόριθμο χρονοπρογραμματισμού ALAP είναι το εξής:

```
ALAP(G(V,E), λ̄)
προγραμμάτισε u_n αρχικοποιώντας t_n(L) = λ̄ + 1;
επανέλαβε
{
επέλεξε μια κορυφή u_i με προγραμματισμένους διαδόχους;
προγραμμάτισε u_i αρχικοποιώντας t_i(L) = min_{j:(ui,uj)∈E}(t_j(L) − d_i);
}
μέχρι (u_0 να έχει χρονοπρογραμματιστεί);
επέστρεψε (t_i(L) για όλα τα i);
```

Σε αντίθεση με τον αλγόριθμο ASAP, ο αλγόριθμος ALAP διατρέχει το γράφο από τις εξόδους προς τις εισόδους. Ένας κόμβος υπόκεινται σε επεξεργασία, μόνο όταν όλοι οι διάδοχοί του έχουν προγραμματιστεί. Εδώ παρατηρείται ένας μέγιστος χρόνος έναρξης, που αποτελεί ένα άνω όριο στο χρόνο υπολογισμού του γράφου ροής δεδομένων. Στην απλούστερη περίπτωση, αυτό το άνω όριο (λ) είναι το μήκος (χρόνος καθυστερημένης έναρξης – χρόνος πρόωρης έναρξης) του χρονοπρογράμματος, το οποίο υπολογίζεται από τον αλγόριθμο ALAP. Βάσει του παραπάνω ψευδοκώδικα, ο χρόνος έναρξης t_i για τον κόμβο u_i παίρνει την ελάχιστη τιμή (min), μεταξύ όλων των χρόνων έναρξης των προκατόχων του πλην την καθυστέρηση του u_i.

1.7.4 – Χρονοπρογραμματισμός λίστας (List Scheduling)

Μια διαδεδομένη ευρετική λύση χρονοπρογραμματισμού είναι ο χρονοπρογραμματισμός λίστας (List Scheduling) **[5][18]**. Ο αλγόριθμος χρονοπρογραμματισμού λίστας χρησιμοποιεί μια ουρά προτεραιότητας (priority queue), για να κρατήσει τους μη χρονοδρομολογημένους κόμβους. Σε κάθε βήμα μια χρονική ακολουθία κόμβων βγαίνει από την ουρά και χρονοδρομολογείται, διαδικασία που επαναλαμβάνεται μέχρι όλες οι ακολουθίες να έχουν χρονοδρομολογηθεί. Η ουρά προτεραιότητας ταξινομείται βάσει κάποιου μέτρου πίεσης, για να χρονοδρομολογηθεί μια ακολουθία κόμβων. Μια συνηθισμένη συνάρτηση ταξινόμησης, βασίζεται στην ταξινόμηση των κόμβων του DFG βάσει της απόστασης στον τελικό κόμβο. Άλλες συναρτήσεις είναι δυνατόν να χρησιμοποιηθούν, όπως για παράδειγμα ο αριθμός των άμεσων διάδοχων κόμβων ενός συγκεκριμένου κόμβου-εντολή.

Ο αλγόριθμος χρονοπρογραμματισμού λίστας είναι ένας επαναληπτικός αλγόριθμος, σε κάθε επανάληψη του οποίου επιλέγεται η καταλληλότερη λειτουργία από μια δεξαμενή αδέσμευτων λειτουργιών, ώστε αυτή να ανατεθεί στο πρώτο βήμα ελέγχου για το οποίο δεν παραβιάζονται οι περιορισμοί πόρων. Στον αλγόριθμο διατηρούνται ειδικότερα δύο λίστες:

- λίστα **ready**: διαθέσιμες εντολές που μπορούν να δρομολογηθούν κατά σειρά προτεραιότητας
- λίστα **active**: εντολές που βρίσκονται υπο εκτέλεση.

Σε κάθε βήμα, η εντολή υψηλότερης προτεραιότητας από τη λίστα ready δρομολογείται και μετακινείται στην active, όπου μένει για χρόνο εκτέλεσης ίσο με τον απαιτούμενο αριθμό κύκλων μηχανής. Ο αλγόριθμος χρονοπρογραμματισμού λίστας περιγράφεται με τον ακόλουθο κώδικα:

```
κύκλος ← 1;
ready ← {κόμβοι φύλλα του DDG};
active ← 0;

όσο (ready ∪ active ≠0)
  εάν (ready ≠0) τότε
    μετακίνησε μια λειτουργία από ready;
    προγραμμάτισε (λειτουργία) ← κύκλος;
  τέλος εάν
  κύκλος ← κύκλος + 1;
  ενημέρωσε την ουρά ready;
τέλος όσο
```

2

Χρονοπρογραμματισμός με τεχνικές ανάλυσης ή μετασχηματισμού γράφου

2 - Χρονοπρογραμματισμός με τεχνικές ανάλυσης ή μετασχηματισμού γράφου

Ο «χρονοπρογραμματισμός» (scheduling) ως έννοια, αναφέρεται στην ικανή παραλληλοποίηση ενός προγράμματος με την αντιστοίχηση της εκκίνησης λειτουργιών, από συγκεκριμένες χρονοθυρίδες (time-slots). Το χρονοπρόγραμμα καθορίζεται με βάση κάποιο εξωτερικό κριτήριο που χαρακτηρίζει το ίδιο το πρόγραμμα ή/και την αρχιτεκτονική μηχανής όπως είναι τυχόν περιορισμοί χρόνου και πόρων. Παράμετροι όπως το μέγεθος του χρονοπρογράμματος (schedule) – ως κύκλοι ρολογιού για την εκτέλεσή του – ο αριθμός των ταυτόχρονα ενεργών μεταβλητών, ο χρόνος για την εύρεσή του, το μέγεθος του κώδικα που θα χρησιμοποιηθεί και ίσως η κατανάλωση ενέργειας (ειδικά σε ενσωματωμένες εφαρμογές), επηρεάζουν την αποτελεσματικότητα του χρονοπρογράμματος. Στο συγκεκριμένο κεφάλαιο, θα αναφερθούμε στους τρόπους με τους οποίους μπορεί να επιτευχθεί χρονοπρογραμματισμός εντολών (απλών λειτουργιών) στις περιπτώσεις των μεταγλωττιστών με τη βοήθεια γράφων, που θα συμβάλλει στη βελτίωση της απόδοσης προγραμμάτων.

2.1 – Εισαγωγικά

Είναι γνωστό ότι ο χρονοπρογραμματισμός γενικά και ειδικότερα ο χρονοπρογραμματισμός εντολών, βελτιώνει την απόδοση ενός προγράμματος για τους επεξεργαστές με διοχέτευση ανάλογα με τους διαθέσιμους πόρους και άλλους περιορισμούς. Πρόκειται για την τεχνική εκείνη, όπου όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο διαιρεί τη διαδικασία της επεξεργασίας σε στάδια (βήματα ελέγχου), σε καθένα από τα οποία αντιστοιχίζεται η εκκίνηση εκτέλεσης απλών λειτουργιών/εντολών.

Όσον αφορά το χρονοπρογραμματισμό εντολών σε τοπικό επίπεδο (βασικό μπλοκ), οι εντολές και οι εξαρτήσεις δεδομένων είναι δυνατό να αναπαρασταθούν με έναν κατευθυνόμενο άκυκλο γράφο (DAG – Directed Acyclic Graph), όπου οι κόμβοι αντιπροσωπεύουν μια εντολή και μια ακμή δείχνει την εξάρτηση δεδομένων που υπάρχει μεταξύ δυο εντολών. Το βάρος που χαρακτηρίζει κάθε ακμή, αντιπροσωπεύει την αντίστοιχη καθυστέρηση που παρουσιάζει συνήθως ένας από τους δύο προσπίπτοντες κόμβους της. Σε ορισμένες περιπτώσεις στο γράφο προστίθενται κόμβοι πηγής (source) και καταβόθρας (sink), χωρίς προηγηθέντες και διαδόχους κόμβους αντίστοιχα για διευκόλυνση εκτέλεσης των βασικών αλγοριθμικών δομών στον χρονοπρογραμματισμό. Η προσθήκη επιπλέον ακμών (εξαρτήσεων) στον κατευθυνόμενο άκυκλο γράφο, εισάγει πρόσθετους περιορισμούς που μειώνουν το εύρος των λύσεων για το πρόβλημα, εξαλείφοντας είτε περιττά είτε κατώτερων επιδόσεων χρονοπρογράμματα, και συντηρώντας το βέλτιστο μήκος του προγράμματος.

Η ευρετική μέθοδος χρονοπρογραμματισμού για ένα μετασχηματισμένο πρόβλημα (π.χ. από την εισαγωγή πρόσθετων ακμών) με τη βοήθεια ενός γράφου, αποδίδει συχνά βελτιωμένα χρονοπρογράμματα για δυσεπίλυτα προβλήματα συγκριτικά με ένα μη μετασχηματισμένο πρόβλημα. Όταν τα μετασχηματισμένα προβλήματα υποβάλλονται σε χρονοπρογραμματισμό με χρήση ενός απαριθμητικού χρονοπρογραμματιστή, αυτό γίνεται γρηγορότερα και ο αριθμός των προβλημάτων που λύνονται βέλτιστα μέσα σε ένα περιορισμένο χρονικό πλαίσιο αυξάνεται. Στην πορεία του κεφαλαίου θα παρουσιαστεί μια ενιαία εικόνα μερικών χρονοπρογραμματιστικών προβλημάτων, η οποία συνεισφέρει στην κατανόησή τους και λειτουργεί καθοδηγητικά στην έρευνα για αποτελεσματικούς αλγορίθμους.

2.2 – Η γενική ιδέα του προβλήματος

Υποθέτουμε αρχικά ότι επιδιώκουμε το χρονοπρογραμματισμό ενός συνόλου εργασιών, με αυθαίρετους περιορισμούς ως προς την προτεραιότητα και με γνωστούς χρόνους εκτέλεσης, μέσω ενός συνόλου από πανομοιότυπους στοιχειώδεις «επεξεργαστές» τους οποίους εκλαμβάνουμε ως λειτουργικές μονάδες. Ο βέλτιστος χρονοπρογραμματισμός των παραπάνω εργασιών, υποδηλώνει τη χρήση ενός ελάχιστου αριθμού μονάδων ώστε να ικανοποιηθεί μια καταληκτική προθεσμία (deadline), ή η λήξη στον ελάχιστο χρόνο με χρήση σταθερού αριθμού μονάδων. Η όλη διαδικασία περιγράφεται σαν ένας μετασχηματισμός ενός αρχικού επιμέρους σχεδίου σε ένα άλλο επιμέρους σχέδιο, του οποίου τα «πρωτεία» δεν παραβιάζουν τους περιορισμούς ως προς τη βελτιστοποίηση και το οποίο έχει ένα μοναδικό βασικό χρονοπρόγραμμα. **[19]**.

Βάσει όσων προαναφέρθηκαν, θεωρούμε ένα σύνολο από εργασίες $T = \{T_1, T_2, ..., T_n\}$ ότι θα εκτελεστούν από ένα σύνολο πανομοιότυπων επεξεργαστικών ή λειτουργικών μονάδων P_i $(i = 1, 2, ..., m)$. Μια μερική διάταξη (partial order) $<$ δίνεται στο T, και ένας μη-αρνητικός ακέραιος d_j αντιπροσωπεύει τη διάρκεια εκτέλεσης για την εργασία T_j.

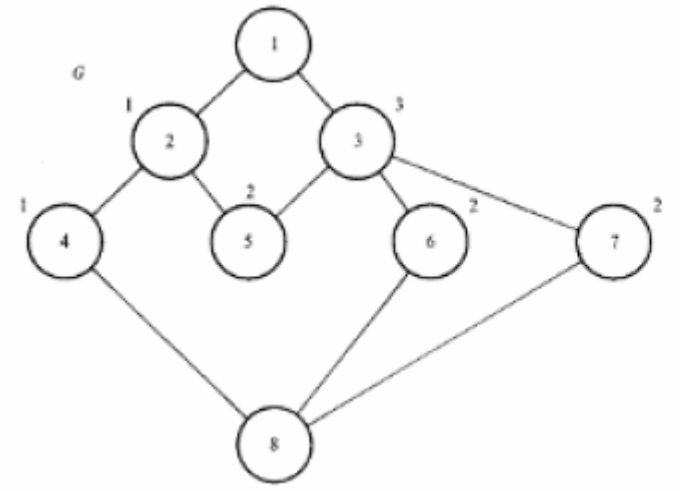
Το μερικώς διατεταγμένο σύνολο $(T, <)$ περιγράφεται από ένα πεπερασμένο, άκυκλο γράφο $G = (V, A)$, όπου V είναι ένα πεπερασμένο σύνολο κορυφών με πληθικότητα (cardinality) n, και A είναι ένα σύνολο ακμών που αντιπροσωπεύεται ως ζεύγη κορυφών. Οι εργασίες T_i αντιστοιχούν στα στοιχεία του συνόλου V και αποτελούν κορυφές του G. Οι ακμές στο σύνολο A περιγράφουν τις προτεραιότητες ανάμεσα στις εργασίες. Στο σχήμα 2.1 που ακολουθεί, παριστάνεται γράφος ως εξής:

1. οι ακμές υποθέτουμε ότι κατευθύνονται προς τα κάτω, και δεν είναι ρητά δηλωμένες ως κατευθυντικές.

2. οι αριθμοί δίπλα στις κορυφές απεικονίζουν τη χρονική διάρκεια εκτέλεσης μιας εργασίας, και εάν δεν υπάρχει αριθμός τότε η συγκεκριμένη εργασία ολοκληρώνεται σε μηδενικό χρόνο.

Από τη στιγμή που μια λειτουργική μονάδα ξεκινήσει την εκτέλεση μιας εργασίας, δεν θα πρέπει να διακοπεί εφόσον δεν έχει ολοκληρώσει. Υποτίθεται ότι οι εργασίες υπόκεινται σε χρονοπρογραμματισμό, ξεκινώντας μόνο με ακέραιες χρονικές τιμές. Το μήκος της κρίσιμης διαδρομής (συντομότερης) για το γράφο, t_{cp}, είναι ο ελάχιστος χρόνος για την εκτέλεση του συνόλου των υπολογισμών. Ως προς αυτό συγκεκριμένα, θα ήταν περισσότερο γενικό και περισσότερο χρήσιμο να οριστεί μια καταληκτική προθεσμία D, όπου οι υπολογισμοί θα πρέπει να εκτελεστούν. Σαφώς θα ισχύει $D \geq t_{cp}$. Σύμφωνα με κάποια πιθανά προσχέδια, για κάθε εργασία T_j έχουμε ένα συγκεκριμένο χρόνο ολοκλήρωσης τον οποίο χαρακτηρίζουμε ως c_j. Το C ορίζεται ως το χρονικό διάνυσμα ολοκλήρωσης, του οποίου το $j-οστό$ στοιχείο είναι το c_j.

Δυο ακραίοι χρόνοι ολοκλήρωσης μιας εργασίας παρουσιάζουν εξαιρετικό ενδιαφέρον. Ο *χρόνος πρόωρης ολοκλήρωσης*, e_{cj}, μιας διαδικασίας T_j, είναι ο ελάχιστος χρόνος κατά τον οποίο μια εργασία μπορεί να τελειώσει, δοθέντων των περιορισμών προτεραιότητας του γράφου. Ο *χρόνος καθυστερημένης ολοκλήρωσης*, l_{cj}, μιας διαδικασίας T_j, δείχνει πόσο μπορεί να καθυστερήσει η ολοκλήρωση μιας εργασίας χωρίς να υπάρχει υπέρβαση της καταληκτικής προθεσμίας. Σύμφωνα με ένα πιθανό χρονοπρόγραμμα, υπάρχει επίσης ένας χρόνος αρχικοποίησης i_j για μια δοθείσα εργασία T_j. Αναλόγως, είναι τότε δυνατό να οριστεί για μια εργασία T_j, ένας χρόνος πρόωρης αρχικοποίησης e_{ij} και ένας χρόνος καθυστερημένης αρχικοποίησης l_{ij}. Όσα αναφέρθηκαν νωρίτερα περιγράφονται στο σχήμα 2.1.



T_j	e_{ij}	l_{ij}	e_{cj}	l_{cj}
2	0	2	1	3
3	0	0	3	3
4	1	4	2	5
5	3	3	5	5
6	3	3	5	5
7	3	3	5	5

Σχήμα 2.1: *Πεπερασμένος Άκυκλος Γράφος και χρόνοι ολοκλήρωσης των διαφόρων εργασιών που περιγράφονται σε αυτόν*

Το μερικώς διατεταγμένο σύνολο αντιπροσωπεύεται από έναν κατευθυνόμενο γράφο, στον οποίο δεν εμφανίζονται οι πλεονάζουσες προτεραιότητες. Με άλλα λόγια αν ο γράφος G περιέχει την ακμή (i, j) και την ακμή (j, k), τότε μια ακμή του τύπου (i, k) δεν είναι αποδεκτή σαν ένα μέρος του συνόλου των ακμών. Ένας γράφος αυτού του τύπου λέγεται ότι βρίσκεται στην «τ-minimal» μορφή. Υπάρχουν γενικές μέθοδοι για τη μετατροπή ενός γράφου που δεν βρίσκεται σε αυτή τη μορφή, ώστε να γίνει «τ-minimal». Ένας απλός αλγόριθμος μπορεί να αποτελεί μια από αυτές τις μεθόδους.

Δυο εργασίες T_i και T_j που ανήκουν στο σύνολο T, είναι συγκρίσιμες εάν $T_i < T_j$ ή $T_j < T_i$. Ένα υποσύνολο Β του T είναι ανεξάρτητο, εάν κάθε δυο στοιχεία του Β δεν είναι συγκρίσιμα. Ένα ανεξάρτητο υποσύνολο συχνά καλείται ως «αντί-αλυσίδα» (antichain). Ένα υποσύνολο του T είναι μια αλυσίδα, εάν κάθε δυο από τα στοιχεία του είναι συγκρίσιμα.

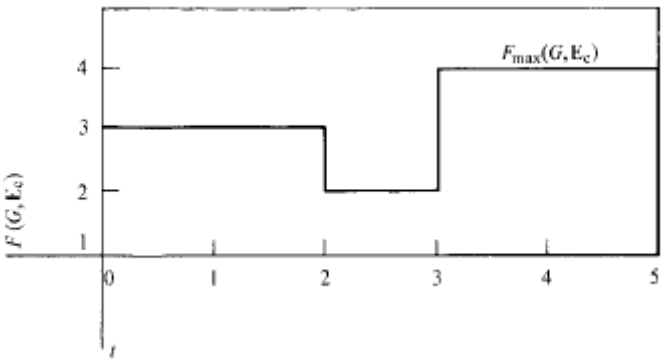
Η δραστηριότητα της εργασίας T_j στο γράφο G ορίζεται ως εξής:

$$f(G, c_J) = \begin{cases} 1, t \in [c_J - d_j, c_j] \\ 0, αλλιώς \end{cases}$$ ενώ η συνάρτηση βάρους πυκνότητας ορίζεται ως:

$$f(G, C) = \sum_{j=1}^{n} f(c_j, t)$$

Τότε, η συνάρτηση $f(G, c_J)$ δείχνει την δραστηριότητα για την εργασία T_j σε σχέση με το χρόνο, ενώ η συνάρτηση $f(G, C)$ δείχνει τη συνολική δραστηριότητα σαν συνάρτηση του χρόνου. Η μέγιστη τιμή για τη συνάρτηση $f(G, C)$, δείχνει τον αριθμό των απαιτούμενων λειτουργικών μονάδων για το πρόγραμμα που ορίζεται από αυτή τη συνάρτηση.

Επιπρόσθετα, αν ονομάσουμε ως E_c το διάνυσμα των χρόνων πρόωρης ολοκλήρωσης του γράφου G, τότε $f(G, E_c)$ είναι η συνάρτηση βάρους πυκνότητας για το χρόνο αυτό (σχήμα 2.2), δηλαδή αντιστοιχεί σε ένα πρόγραμμα όπου όλες οι εργασίες υποβάλλονται σε επεξεργασία όσο το δυνατόν πιο νωρίς.



Σχήμα 2.2: *Παράδειγμα συνάρτησης* $f(G, E_c)$ σε σχέση με το χρόνο

Όμοια, ονομάζοντας L_c το διάνυσμα των χρόνων καθυστερημένης ολοκλήρωσης l_{cj}, μπορούμε να ορίσουμε μια συνάρτηση καθυστέρησης βάρους πυκνότητας. Προφανώς ισχύει μέσω της συνάρτησης $f_{\max}(G,C)$, η μέγιστη τιμή σε σχέση με το χρόνο. Τέλος, δοθέντος ενός χρονοπρογράμματος, καλούμε ω τον συνολικό χρόνο για την επεξεργασία όλων των εργασιών του συνόλου T.

2.3 – Ο χρονοπρογραμματισμός ως μετασχηματισμός γράφου

2.3.1 – Προτεραιότητες στον πρωταρχικό γράφο

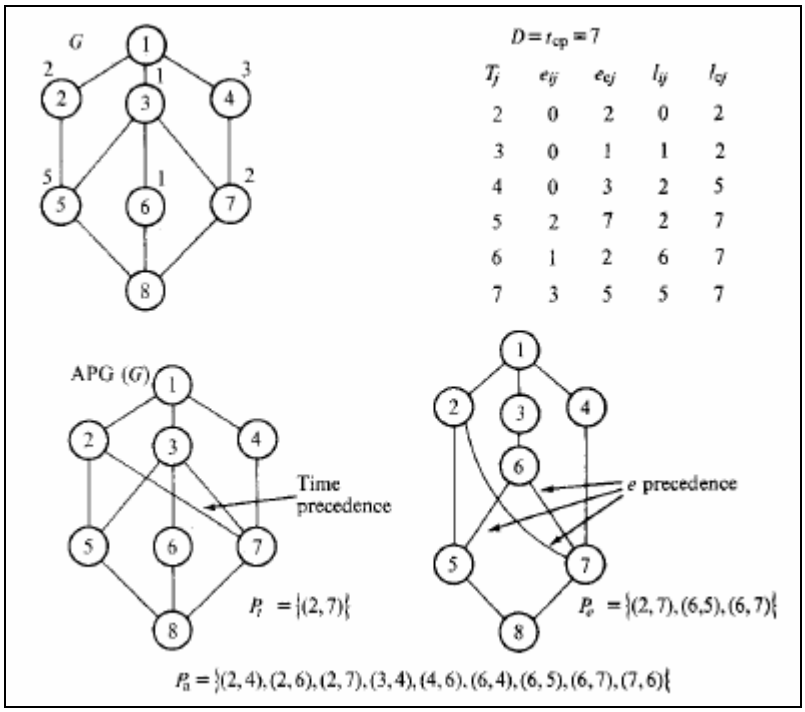
Δοθέντος ενός γράφου G, υπάρχει ένα σύνολο από προτεραιότητες οι οποίες ανταποκρίνονται στη διάταξη των στοιχείων του συνόλου V. Κάποιες από αυτές τις προτεραιότητες ορίζονται με μια ακμή στο γράφο, παραδείγματος χάριν ένα στοιχείο του συνόλου A και άλλα, υπονοούνται σύμφωνα με τη μεταβατικότητα της σχέσης διάταξης. Όταν η μερική διάταξη περιγράφει ένα σύνολο από υπολογισμούς, αυτές οι προτεραιότητες ανταποκρίνονται σε λογικές εξαρτήσεις των σημείων συνάντησης προσπιπτουσών κορυφών, και αν το G βρίσκεται σε «τ-minimal» μορφή – όπως έγινε υπόθεση νωρίτερα – δεν μπορεί καμία ακμή να μετακινηθεί, χωρίς να παραβιάσει μερικούς από τους λογικούς περιορισμούς των υπολογισμών. Κατά συνέπεια υποθέτουμε ότι ο γράφος G έχει τον ελάχιστο πιθανό αριθμό από προτεραιότητες, που είναι σύμφωνες με τους λογικούς περιορισμούς του συνόλου των υπολογισμών.

Το σύνολο των λογικών προτεραιοτήτων για το γράφο G, P_l, είναι το σύνολο όλων των προτεραιοτήτων που υπονοούνται απευθείας από τις ακμές, A, και από τη μεταβατικότητα της σχέσης διάταξης. Σύμφωνα με τους χρόνους πρόωρης και καθυστερημένης ολοκλήρωσης για κάθε εργασία στο G, υπάρχουν εργασίες οι οποίες πάντα ολοκληρώνονται πριν κάποιες άλλες ξεκινήσουν, πράγμα που υποδηλώνει την ύπαρξη χρονικών προτεραιοτήτων.

Ένας γράφος που περιγράφεται ως APG (All Precedences Graph), είναι αυτός που λαμβάνεται με την προσθήκη στον G των χρονικών προτεραιοτήτων (το ονομάζουμε P_t), και την εξάλειψη των περιττών προτεραιοτήτων. Ο γράφος APG είναι σαφώς ισοδύναμος με τον G με σεβασμό στη λογική συνάφεια των εργασιών, και το σύνολο P_t δημιουργεί μόνο συγκεκριμένες προτεραιότητες που υποδηλώνονται από τις χρονικές αλληλεξαρτήσεις του δικτύου.

Ένα διαφορετικό σύνολο προτεραιοτήτων είναι το P_e. Πρόκειται για τις λεγόμενες e-προτεραιότητες, οι οποίες υφίστανται όταν όλες οι εργασίες βρίσκονται στις πρόωρες χρονικά θέσεις τους. Με τον ίδιο τρόπο, αν όλες οι εργασίες βρίσκονται σε καθυστερημένες θέσεις ως προς το χρόνο, τότε μιλάμε

για l-προτεραιότητες (P_l). Ακόμα υπάρχουν και οι επιτρεπτές (admissible) προτεραιότητες (P_a), το σύνολο των οποίων απαρτίζεται από προτεραιότητες που θα μπορούσαν να προστεθούν στο G χωρίς η συντομότερη διαδρομή να υπερβαίνει το D.



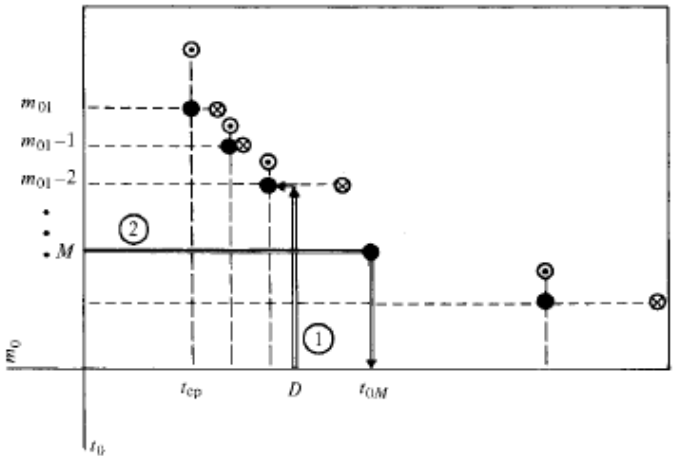
Σχήμα 2.3: *Προτεραιότητες γράφου*

Στο προηγούμενο σχήμα επεξηγούνται όσα προαναφέρθηκαν, ενώ θα πρέπει να τονιστεί ότι όσο τα σύνολα P_t και P_e περιέχουν προτεραιότητες που είναι αμοιβαία συμβατές, το σύνολο P_a περιέχει προτεραιότητες που θα μπορούσαν να είναι αμοιβαία ασύμβατες, όπως για παράδειγμα οι (4,6) και (6,4).

2.3.2 – Βασικός και βέλτιστος χρονοπρογραμματισμός

Δυο από τα θεμελιώδη προβλήματα στη θεωρία του χρονοπρογραμματισμού είναι: 1) μια καταληκτική προθεσμία για το χρόνο εκτέλεσης ενός δοθέντος συνόλου εργασιών, θα πρέπει να ικανοποιείται με τη χρήση ελάχιστου αριθμού λειτουργικών μονάδων 2) ένας σταθερός αριθμός από λειτουργικές μονάδες πρέπει να χρησιμοποιηθεί για να εκτελεστεί ένα σύνολο εργασιών στον ελάχιστο χρόνο. Υπάρχουν μερικές διττές πτυχές σε αυτά τα δυο προβλήματα, παρόλο που δεν είναι πλήρως συμμετρικά.

Μια βασική στρατηγική χρονοπρογραμματισμού, αποτελείται από την ανάθεση μιας εργασίας – της οποίας οι προκάτοχοι είναι πλήρεις – στην πρώτη διαθέσιμη μονάδα. Με άλλα λόγια τα βασικά προγράμματα δεν περιέχουν τεχνητά ανενεργές λειτουργικές μονάδες. Έτσι εάν μια λειτουργική μονάδα είναι ανενεργή, αυτό συμβαίνει επειδή καμία εργασία δεν της έχει ανατεθεί. Εμείς θεωρούμε ως $S(G)$ το σύνολο όλων των βασικών προγραμμάτων για το G. Το $S(G)$ μπορεί να περιέχει πολλά προγράμματα, επειδή περισσότερες από μια εργασίες είναι υποψήφιες προς ανάθεση σε μια λειτουργική μονάδα, όταν η τελευταία γίνει διαθέσιμη.



Σχήμα 2.4: *Σχέση ανάμεσα στο βέλτιστο m και βέλτιστο t*

Για το σύνολο $S(G)$ είναι γνωστό, ότι δεν είναι απαραίτητο να περιέχει τα βέλτιστα προγράμματα και το γράφο G που προκύπτει από αυτά. Αυτό βέβαια μπορεί να συμβεί, όταν ο γράφος έχει μετασχηματιστεί με ένα συγκεκριμένο τρόπο και έχουν επιλεγεί από αυτόν τα βασικά προγράμματα. Στο προηγούμενο σχήμα απεικονίζεται η σχέση για έναν δοθέντα γράφο G, ανάμεσα στον m_0 (ελάχιστος απαιτούμενος αριθμός λειτουργικών μονάδων, για την εκτέλεση του συνόλου Τ στα όρια της καταληκτικής προθεσμίας D) και τον t_0 (ελάχιστος χρόνος κατά τον οποίο ένα σύνολο Τ μπορεί να ολοκληρωθεί με m_0 λειτουργικές μονάδες).

Στο σχήμα 2.4, τα σημεία ⊗ αντιπροσωπεύουν τις ελάχιστες τιμές για το χρόνο των βασικών προγραμμάτων με χρήση ενός δοθέντος αριθμού από λειτουργικές μονάδες, ενώ τα σημεία ⊙ αντιπροσωπεύουν τον ελάχιστο απαιτούμενο αριθμό μονάδων ώστε να επιτευχθεί μια καταληκτική προθεσμία, με χρήση βασικού χρονοπρογραμματισμού στον G.

2.4 – Εφαρμογή σε έναν αλγόριθμο χρονοπρογραμματισμού

Η μελέτη των προτεραιοτήτων που σχετίζονται με προγράμματα, παρέχει διορατικότητα στον καθορισμό νέων αλγορίθμων χρονοπρογραμματισμού, προκειμένου να βελτιωθούν οι υπάρχοντες αλγόριθμοι ή να καθοριστούν ευρετικοί αλγόριθμοι χρονοπρογραμματισμού. Σε αυτή την υποενότητα θα αναφερθούμε σε έναν αλγόριθμο που βασίζεται στη προσθήκη προτεραιοτήτων, και θα μελετήσουμε τρόπους για τη βελτίωση της εκτέλεσής του. Ο χρονοπρογραμματισμός αυθαίρετων γράφων φαίνεται να αποτελεί ένα ουσιαστικά απαριθμητικό πρόβλημα, και είναι τότε μάταιη η αναζήτηση για αλγορίθμους που είναι αποδοτικοί υπό μια γενική έννοια. Παρόλα ταύτα, οι βελτιστοποιήσεις στους υπάρχοντες αλγορίθμους μπορούν να κάνουν τη διαφορά από πρακτική άποψη.

Μια διεργασία βέλτιστου χρονοπρογραμματισμού ως προς την επεξεργασία, ξεκινά υποθέτοντας κάποια τιμή για τον αριθμό των λειτουργικών μονάδων, m, και στη συνέχεια συνεχίζει με την κυρίως εφαρμογή του αλγόριθμου χρονοπρογραμματισμού. Εάν δε βρεθεί κάποιο επαρκές πρόγραμμα, η διεργασία επαναλαμβάνεται αφού πρώτα αυξηθεί κατά ένα ο αριθμός των μονάδων, μέχρις ότου βρεθεί ένα επιτυχές πρόγραμμα. Από τη στιγμή που η αρχική τιμή για το m είναι κατάλληλη να χρησιμοποιήσει ένα κατώτερο όριο, έκτοτε κανένα βέλτιστο πρόγραμμα δεν απαιτεί μικρότερο αριθμό λειτουργικών μονάδων από αυτή την τιμή.

Ο αλγόριθμος που εξετάζεται αποτελεί μια γενίκευση μιας ιδέας του Barskiy **[20]**. Για την περίπτωση του βέλτιστου χρονοπρογραμματισμού, αποτελείται από προστιθέμενες ακμές στις ενεργές αντί-αλυσίδες του γράφου, οι οποίες έχουν μια πληθικότητα μεγαλύτερη από τον αριθμό των λειτουργικών μονάδων που αρχικά είχε υποτεθεί. Με στόχο να επικρατήσει ένας γράφος του οποίου ο βασικός χρονοπρογραμματισμός είναι ταυτόχρονα και βέλτιστος, η πιο πρόωρη συνάρτηση βάρους πυκνότητας χρησιμοποιείται ως αναφορά σε αυτή τη διαδικασία. Ο γράφος έχει μετασχηματιστεί με τέτοιο τρόπο, ώστε η συνάρτηση βάρους πυκνότητας γι αυτόν δεν υπερβαίνει ποτέ τον εκτιμώμενο αριθμό των λειτουργικών μονάδων. Κατόπιν, αρκεί η εκτέλεση βασικού χρονοπρογραμματισμού στο μετασχηματισμένο γράφο για να προκύψει ένα βέλτιστο πρόγραμμα. Όταν δεν υπάρχει κάποιος τρόπος για την προσθήκη ακμών σε κάποιες ενεργές αντί-αλυσίδες, χωρίς να γίνει υπέρβαση στην προθεσμία "deadline", είναι απαραίτητη η οπισθοδρόμηση (backtracking) στην προηγούμενη ενεργή αντί-αλυσίδα, η προσθήκη ενός νέου συνόλου από ακμές και η συνέχιση της διαδικασίας. Στην περίπτωση που πραγματοποιηθεί οπισθοδρόμηση, εννοείται ότι αναφερόμαστε σε αλγόριθμο δυναμικού χρονοπρογραμματισμού. Ο Barskiy απέδειξε ότι εάν η πληθικότητα μιας ενεργής αντί-αλυσίδας είναι r (>m), αρκεί η προσθήκη $r - m$ ακμών σε αλυσίδες πλήθους m για να προκύψουν βέλτιστα προγράμματα.

Τα πλεονεκτήματα της προσθήκης μιας μόνο ακμής τη φορά (αντί για ένα σύνολο $r-m$ ακμών), μπορεί να επεκταθούν περισσότερο εάν παρατηρήσουμε τις ακμές που μπορούν να προστεθούν σε μια δεδομένη στιγμή. Για παράδειγμα σε μια συγκεκριμένη αντί-αλυσίδα μπορούμε να τις ταξινομήσουμε ως εξής: σε αυτή υφίστανται ένα σύνολο ακμών, ώστε η προσθήκη κάποιας από αυτές στο γράφο δεν θα κάνει το συνολικό χρόνο ολοκλήρωσης να υπερβεί την προθεσμία «deadline». Αυτό το σύνολο αποτελείται από αποδεκτές προτεραιότητες P_a. Ένα υποσύνολο του P_a, το P_b, είναι το σύνολο των ακμών ώστε η προσθήκη μιας από αυτές να μην αλλάξει την τιμή του κατώτατου ορίου. Σημειώνεται ότι το P_b ποικίλει, σύμφωνα με την έκφραση του κατώτατου ορίου που χρησιμοποιείται. Το σύνολο των ακμών που είναι μέρος βέλτιστων προγραμμάτων, P_c, είναι ένα υποσύνολο του P_b. Κατά συνέπεια ως μια μέθοδος βελτίωσης αυτού του αλγορίθμου χρονοπρογραμματισμού, προτείνεται ο υπολογισμός ξανά του κατώτερου ορίου στον αριθμό των λειτουργικών μονάδων μετά από την προσθήκη μιας ακμής. Εάν η προσθήκη της ακμής (a,b) οδηγεί σε κάποια υπέρβαση για το προηγούμενο κατώτερο όριο, τότε η (a,b) δεν προστίθεται στο γράφο και γίνεται απόπειρα με άλλη ακμή.

Στην περίπτωση της χρήσης του κατώτερου ορίου στον αριθμό των λειτουργικών μονάδων και αν υπάρχει αυξητική εκτέλεση των υπολογισμών, η υπολογιστική ταχύτητα μπορεί να βελτιωθεί σε αυτή τη δοκιμή με την εξέταση των ακόλουθων παραμέτρων:

1. Εάν η ενεργή αντί-αλυσίδα στην οποία προστίθεται η ακμή ξεκινά τη χρονική στιγμή $t=t_i$, μόνο τα διαστήματα ανάμεσα στο t_i και στη προθεσμία D θα πρέπει να εξεταστούν.
2. Από τη στιγμή που το προηγούμενο κατώτερο όριο βρεθεί σε κατάσταση υπέρβασης για κάθε διάστημα, ο υπολογισμός μπορεί να σταματήσει και η αντίστοιχη ακμή απορρίπτεται.
3. Τα κατώτερα όρια μπορούν να υπολογίζονται με έναν αυξητικό τρόπο, ξεκινώντας από τις προηγούμενες τιμές και εξετάζοντας μόνο τις αλλαγές που παράγονται με την προσθήκη μιας δεδομένης ακμής.
4. Εάν η ακμή (a,b) αποτύχει σε αυτή τη δοκιμή, αυτό σημαίνει ότι το b δεν μπορεί να είναι καθυστερημένο δοθέν από την ακμή (a,b). Αυτό ελαχιστοποιείται από την εκτίμηση άλλων ακμών που θα καθυστερούσαν το b, το ίδιο ή περισσότερο. Επίσης επιλέγοντας εκείνες τις ακμές που θα παράγουν τη μικρότερη καθυστέρηση στο b, θα αυξηθεί η πιθανότητα εύρεσης μια ακμής τόξου που θα περνά επιτυχώς τη δοκιμή του κατώτερου ορίου στον αριθμό των λειτουργικών μονάδων.
5. Έστω δυο διαδοχικές ενεργές αντί-αλυσίδες με πληθικότητα μεγαλύτερη από m τις χρονικές στιγμές t_i και t_j. Εάν η προσθήκη ακμών ώστε να μειωθεί το εύρος της αντί-αλυσίδας τη στιγμή t_i, δεν δημιουργεί

καθυστέρηση σε εργασίες μετά από τη στιγμή t_j, κατόπιν μπορεί να αποδειχθεί ότι η αντί-αλυσίδα τη χρονική στιγμή t_i μπορεί να παρακαμφθεί στην διαδικασία οπισθοδρόμησης. Επιπλέον, εάν μόνο ένα υποσύνολο των εργασιών στην αντί-αλυσίδα τη στιγμή t_i, καθυστερεί τις εργασίες μετά από το χρόνο t_j, τότε για την διαδικασία οπισθοδρόμησης αρκεί η εξέταση μόνο του υποσυνόλου των εργασιών. Όταν συμβεί αυτή η κατάσταση, υπάρχει μια μείωση του αριθμού των ακμών που θα εξεταστούν για μια πιθανή προσθήκη.

Όταν το κατώτερο όριο για ένα δοθέντα γράφο είναι γνωστό και μια ακμή προστίθεται σε αυτόν, το κατώτερο όριο του τροποποιημένου γράφου μπορεί να αξιολογηθεί αποδοτικά, ξεκινώντας από το σύνολο των τιμών που χρησιμοποιήθηκαν για τον υπολογισμό του προηγούμενου κατώτερου ορίου.

Η μελέτη της εκτέλεσης ευρετικών μεθόδων, προτάθηκε (H.O Levy) **[21]** ως η βάση ενός προσεγγιστικού αλγορίθμου για χρονικά βέλτιστο χρονοπρογραμματισμό. Σε ένα τέτοιο αλγόριθμο, οποτεδήποτε η συνάρτηση βάρους πυκνότητας υπερβαίνει τον δοθέντα αριθμό των λειτουργικών μονάδων m, ακμές προστίθενται έτσι ώστε η πυκνότητα βάρους δεν υπερβαίνει το m, χωρίς να γίνεται κάποια οπισθοδρόμηση. Εμπειρικά αποτελέσματα δείχνουν ότι πρόκειται για υποσχόμενες ευρετικές μεθόδους, αλλά απαιτούνται επιπλέον συμπερασματικές δοκιμές. Η ίδια βελτίωση προτάθηκε εδώ για τον ορθό αλγόριθμο, όπου για παράδειγμα η χρήση του κατώτερου ορίου του αριθμού λειτουργικών μονάδων για δοκιμή ακμών που είναι προς προσθήκη, μπορεί να έχει εφαρμογή σε μια ευρετική μέθοδο χρονοπρογραμματισμού.

2.5 – Συμπεράσματα

Ο χαρακτηρισμός των γράφων που ικανοποιούν τις προτεραιότητες ενός δοθέντος προγράμματος, είναι θεωρητικού ενδιαφέροντος παρόλο που δεν οδηγούν απευθείας σε καλύτερους αλγορίθμους χρονοπρογραμματισμού. Από την άλλη πλευρά, η ανάλυση των προτεραιοτήτων που είναι παρούσες στον δοθέντα υπολογιστικό γράφο, είναι εξαιρετικής σημασίας για την κατανόηση της διεργασίας του χρονοπρογραμματισμού των εργασιών του γράφου. Προσδίδει διορατικότητα με την έννοια της βελτίωσης των υπαρχόντων αλγορίθμων χρονοπρογραμματισμού, και της διατύπωσης νέων ευρετικών μεθόδων για την επίτευξή του. Βασιζόμενοι σε αυτή τη μελέτη, προτάθηκαν μερικοί τρόποι βελτίωσης ενός προγενέστερου αλγορίθμου χρονοπρογραμματισμού. Από τη στιγμή που το αποτέλεσμά τους εξαρτάται κατά ένα μεγάλο μέρος, από τη δομή του γράφου που τίθεται υπό χρονοπρογραμματισμό, μια θεωρητική ανάλυση της βελτίωσης είναι πολύ δύσκολη. Η απαριθμητική φύση της διαδικασίας χρονοπρογραμματισμού προτείνει ως υποσχόμενη αυτή την προσέγγιση, αλλά μια εμπειρική

αξιολόγηση θα ήταν αναγκαία για την εξακρίβωση της υπολογιστικής επίδρασης των προτεινόμενων ιδεών.

Η ανάλυση της διεργασίας καθορισμού κατώτερων ορίων ως μετασχηματισμών γράφου, βοηθά επίσης στη βελτίωση του υπολογισμού αυτών των κατώτερων ορίων.

3

Μετασχηματισμοί γράφου εξάρτησης δεδομένων για το χρονοπρογραμματισμό εντολών

3 - Μετασχηματισμοί γράφου εξάρτησης δεδομένων για το χρονοπρογραμματισμό εντολών

Μέσα από το παρόν κεφάλαιο, παρουσιάζεται ένα σύνολο αποδοτικών μετασχηματισμών γράφου για το χρονοπρογραμματισμό εντολών σε τοπικό επίπεδο. Αυτοί οι μετασχηματισμοί στο γράφο εξάρτησης δεδομένων, είναι γνωστό ότι περικόπτουν τα περιττά και κατώτερων επιδόσεων προγράμματα από το πεδίο λύσεων του προβλήματος. Οι μετασχηματισμοί είναι γρήγοροι και δεν αυξάνουν το ελάχιστο μήκος προγράμματος του βασικού μπλοκ. Ένα μετασχηματισμένο πρόβλημα με την εφαρμογή σε αυτό ευρετικού χρονοπρογραμματισμού, συχνά παράγει βελτιωμένα προγράμματα, συγκρινόμενα με τον ευρετικό χρονοπρογραμματισμό μη-μετασχηματισμένων προβλημάτων.

Οι μετασχηματισμοί που παρουσιάζονται στη συνέχεια, προσθέτουν ακμές σε έναν κατευθυνόμενο άκυκλο γράφο (DAG). Δεδομένου ότι κανένας περιορισμός πόρων ή λανθάνων περιορισμός δεν αφαιρείται από το πρόβλημα του χρονοπρογραμματισμού, εφικτά προγράμματα του μετασχηματισμένου γράφου είναι αναγκαία εφαρμόσιμα προγράμματα του αρχικού γράφου. Οι προτεινόμενοι μετασχηματισμοί **[15]** δεν αυξάνουν το βέλτιστο μήκος προγράμματος του γράφου, και κατά συνέπεια κάθε τέτοιος μετασχηματισμός είναι βέλτιστος.

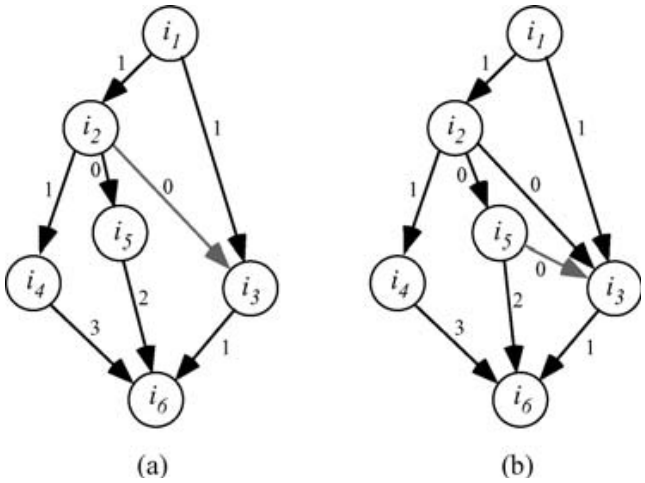
3.1 – Μετασχηματισμός κόμβου ανώτερης τάξης

Η προσθήκη μιας ακμής ανάμεσα σε ένα ζεύγος από ανεξάρτητους κόμβους σε έναν DAG, ίσως αυξήσει το βέλτιστο μήκος του προγράμματος. Γενικά ο καθορισμός του εάν μια προστιθέμενη ακμή αυξάνει το βέλτιστο μήκος ενός προγράμματος, είναι τόσο δύσκολος όσο δύσκολος είναι ο βέλτιστος χρονοπρογραμματισμός στο γράφο.

Ο μετασχηματισμός κόμβου ανώτερης τάξης, αποτελεί μια τεχνική η οποία προσθέτει ακμές μηδενικής καθυστέρησης («ανώτερες» ακμές), μεταξύ ζεύγους κόμβων οι οποίοι αποκαλούνται κόμβοι ανώτερης τάξης. Στην ουσία η τεχνική αυτή αποτελεί έναν βέλτιστο μετασχηματισμό. Μια προστιθέμενη ανώτερη ακμή (i, j), αλλάζει τις αποστάσεις για τη συντομότερη διαδρομή ανάμεσα σε κόμβους ενός DAG. Αυτό μπορεί να αποκαλύψει νέες περιπτώσεις ζευγαριών κόμβων ανώτερης τάξης, και επομένως ο μετασχηματισμός μπορεί να εφαρμοστεί επαναληπτικά.

Για παράδειγμα εξετάζουμε τον DAG στο σχήμα 3.1, όπου όλοι οι κόμβοι είναι του ιδίου τύπου. Ο κόμβος i_2 είναι ανώτερος από τον κόμβο i_3, και η «ανώτερη» ακμή (i_2, i_3) έχει ήδη προστεθεί με τον τρόπο που απεικονίζεται στο σχήμα 3.1(a). Μετά την προσθήκη της (i_2, i_3), ο κόμβος i_5 είναι ανώτερος από

τον κόμβο i_3 και η αντίστοιχη ανώτερη ακμή (i_5, i_3) προστίθεται στο σχήμα 3.1(b). Ο αριθμός των εφικτών προγραμμάτων που αντιστοιχούν στον μετασχηματισμένο DAG, είναι μειωμένος σε σύγκριση με τον αρχικό γράφο και το μήκος του βέλτιστου προγράμματος παραμένει αμετάβλητο. Κατά συνέπεια η απαρίθμηση του εύρους των λύσεων του μετασχηματισμένου προγράμματος, είναι πλέον ταχύτερη.



Σχήμα 3.1: *Παράδειγμα μετασχηματισμού κόμβου ανώτερης τάξης*

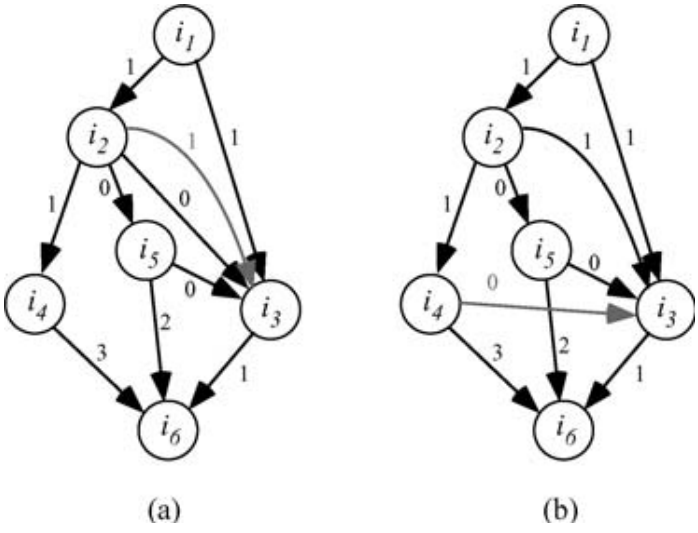
3.2 – Βελτιωμένος υπολογισμός απόστασης

Εφαρμόζοντας το μετασχηματισμό κόμβου ανώτερης τάξης, υπάρχει πιθανότητα να δημιουργηθούν μονοπάτια ακμών που χαρακτηρίζονται από μηδενική καθυστέρηση (latency-zero edges) μεταξύ των κόμβων στον άκυκλο γράφο. Η απόσταση κατά μήκος ενός τέτοιου μονοπατιού *Pij*, από έναν κόμβο i σε έναν κόμβο j είναι μηδέν. Εντούτοις, αυτό μπορεί να είναι ένα ασαφές κάτω φράγμα στο διαχωρισμό ανάμεσα στον κύκλο έκδοσης (issue cycle) του i και στον κύκλο έκδοσης του j σε ένα εφικτό πρόγραμμα, εξαιτίας περιορισμών πόρων (resource constraints) από πλευράς των λειτουργικών μονάδων του επεξεργαστή. Οι κόμβοι ανάμεσα στα i και j στο *Pij*, ίσως υπερκαλύπτουν τις παρεχόμενες θύρες διοχέτευσης που είναι διαθέσιμες σε έναν κύκλο έκδοσης, και έτσι προλαμβάνουν την έκδοση των i και j στον ίδιο κύκλο. Η συμπερίληψη των περιορισμών πόρων στον υπολογισμό της απόστασης, παράγει ένα πιο ακριβές κατώτερο όριο στο διαχωρισμό των κόμβων σε ένα εφικτό πρόγραμμα. Αυτοί οι περιορισμοί πόρων μπορούν να εκφραστούν σε έναν DAG με την προσθήκη μοναδιαίων καθυστερήσεων, και αντιστοιχίζονται σε ακμές πόρων (latency-one resource edges). Μια τέτοια ακμή προστίθεται

ανάμεσα στους κόμβους i και j, εάν υφίσταται ένα μονοπάτι Pij αποτελούμενο από ακμές μηδενικής καθυστέρησης από το i στο j, όπου ο αριθμός των κόμβων στο Pij είναι ένας παραπάνω από τον αριθμό των άμεσων γραμμών επικοινωνίας (κατευθυνόμενες ακμές) προς τον κόμβο j. Η προσθήκη μιας ακμής πόρου (i, j) είναι συμφέρουσα, διότι η απόσταση της συντομότερης διαδρομής από το i στο j επεκτείνεται. Με τον τρόπο αυτό είναι δυνατή η αποκάλυψη επιπρόσθετων ζευγαριών από κόμβους ανώτερης τάξης στον DAG.

Για παράδειγμα ο γράφος στο σχήμα 3.1(b), έχει ένα μονοπάτι από ακμές μηδενικής καθυστέρησης από τον κόμβο i_2 στον κόμβο i_3, περιέχοντας τρεις κόμβους. Για ένα μοντέλο επεξεργαστή με δυο άμεσες γραμμές επικοινωνίας για κόμβο του παραπάνω τύπου, η ακμή (i_2, i_3) που αποτελεί μια ακμή πόρου, προστίθεται με τον τρόπο που απεικονίζεται στο σχήμα 3.2(a). Η ανώτερη ακμή από τον κόμβο i_2 στον κόμβο i_3 είναι περιττή και αφαιρείται. Μετά την προσθήκη της ακμής πόρου, ο κόμβος i_4 είναι ανώτερος από τον κόμβο i_3 και η «latency-zero» ακμή (i_4, i_3) προστίθεται όπως φαίνεται στο σχήμα 3.2(b).

Οι ακμές των πόρων χρησιμοποιούνται μόνο για τον υπολογισμό ακριβέστερων ελάχιστων αποστάσεων, μεταξύ των κόμβων στον DAG. Δεν πρόκειται για επιπρόσθετους περιορισμούς στο πρόβλημα του χρονοπρογραμματισμού, και αυτό σημαίνει διαφορετικά ότι αγνοούνται κατά την εφαρμογή μετασχηματισμών γράφου.



Σχήμα 3.2: *Παράδειγμα ακμής πόρου*

3.3 – Αλγόριθμος κόμβου ανώτερης τάξης

Η υλοποίηση του μετασχηματισμού για κόμβο ανώτερης τάξης, προϋποθέτει την ύπαρξη ενός αλγορίθμου ο οποίος προσθέτει ανώτερες ακμές μεταξύ ζευγαριών κόμβων, ικανοποιώντας **θεώρημα** που δηλώνει ότι: ***ανώτερες ακμές που προστέθηκαν πρωτύτερα, δεν χρειάζεται να εξετάζονται για τον προσδιορισμό περιπτώσεων ζευγών κόμβων ανώτερης τάξης*** **[12].** Η πρακτική σημασία του θεωρήματος έγκειται στην απλοποίηση του προαναφερθέντος αλγορίθμου. Οι ακμές πόρων προστίθενται αναλόγως, οι πλεονάζουσες ακμές διαγράφονται και ο αλγόριθμος συνεχίζει, μέχρις ότου τα ζεύγη των κόμβων ανώτερης τάξης σταματήσουν να υφίστανται στον κατευθυνόμενο άκυκλο γράφο.

Ένας τέτοιος γράφος (εφεξής DAG), αποθηκεύεται ως ένας πίνακας από λίστες ακμών. Κάθε κόμβος περιλαμβάνει μια συνδεδεμένη λίστα των άμεσα προκατόχων και των άμεσα διαδόχων αυτού. Κατά συνέπεια οι ακμές μπορούν να προστεθούν σε σταθερό χρόνο, με τους προκατόχους και τους διαδόχους του κόμβου να ερευνώνται αποτελεσματικά.

Ο αλγόριθμος δημιουργεί αρχικά μια λίστα από ζευγάρια κόμβων ανώτερης τάξης στον μη μετασχηματισμένο DAG, εξετάζοντας όλα τα ζευγάρια των κόμβων και ελέγχοντας εάν ικανοποιείται το παραπάνω θεώρημα. Τότε ένα ζευγάρι από κόμβους αφαιρείται από τη λίστα και μια ανώτερη ακμή προστίθεται ανάμεσα στους κόμβους. Η προστιθέμενη ακμή ίσως αλλάξει τις σχέσεις ανωτερότητας στον DAG. Δηλαδή κάποια ζευγάρια κόμβων που παραμένουν στη λίστα, ίσως να μην ικανοποιούν πλέον τις προϋποθέσεις του θεωρήματος, και ίσως προκύψουν νέες περιπτώσεις ζευγαριών κόμβων ανώτερης τάξης. Ως εκ τούτου, το σύνολο από τέτοια ζευγάρια ανανεώνεται με κάθε ακμή που προστίθεται στον DAG.

Δύο δομές δεδομένων χρησιμοποιούνται από τον αλγόριθμο, για τον προσδιορισμό και την ανανέωση της λίστας με τα ζευγάρια των κόμβων ανώτερης τάξης. Αυτές οι δομές δημιουργούνται αρχικά με την εξέταση του μη μετασχηματισμένου DAG, και ενημερώνονται με την προσθήκη μιας ανώτερης ακμής ή μιας ακμής πόρου.

Η πρώτη δομή δεδομένων είναι ένας πίνακας αποστάσεων, για τη συντομότερη διαδρομή (critical path) ανάμεσα σε όλους τους κόμβους στον DAG. Εάν υποθέσουμε ότι ο γράφος αποτελείται από n κόμβους, τότε έχουμε έναν πίνακα με διαστάσεις $n \times n$. Ο πίνακας της απόστασης (distance table), χρησιμοποιείται για να προσδιορίσει ζεύγη από ανεξάρτητους κόμβους και να εκτιμήσει εάν ικανοποιούνται κάποιες προϋποθέσεις, για την ισχύ του θεωρήματος που προαναφέρθηκε.

Εξαιτίας του γεγονότος ότι οι αποστάσεις στον πίνακα χρησιμοποιούνται μόνο για τον προσδιορισμό της ανεξαρτησίας για έναν κόμβο, συγκρινόμενες με τις ακμές καθυστέρησης παρατηρείται το εξής: οι αποστάσεις που είναι

μεγαλύτερες από τη μέγιστη καθυστέρηση (max-latency) για μια ακμή στον DAG, δεν είναι ανάγκη να υπολογίζονται με ακρίβεια. Αυτό επιτρέπει μια απλοποίηση που μειώνει την δαπάνη για τον υπολογισμό του πίνακα απόστασης, καθώς προστίθενται νέες ακμές στο γράφο.

Η δεύτερη δομή δεδομένων που χρησιμοποιεί ο αλγόριθμος, είναι ο πίνακας SUPERIOR. Ο πίνακας μας δείχνει πόσο κοντά βρίσκεται κάθε ζευγάρι κόμβων στον DAG, όσον αφορά την ικανοποίηση μερικών προϋποθέσεων ώστε να ισχύει το αρχικό θεώρημα. Για παράδειγμα για τους κόμβους i και j, ο πίνακας SUPERIOR (i, j) ορίζεται αν και μόνο αν τα i και j είναι του ιδίου τύπου και είναι ανεξάρτητα. Οποιοσδήποτε χειρισμός δεδομένων διαφορετικού τύπου θα παραβίαζε τη δομή που χαρακτηρίζει έναν πίνακα. Ο κόμβος i είναι ανώτερος από τον κόμβο j, αν και μόνο αν ο πίνακας SUPERIOR (i, j) ισούται με το μηδέν, δηλαδή όλα τα στοιχεία του πίνακα είναι μηδενικά. Τόσο ο πίνακας της απόστασης όσο και ο πίνακας SUPERIOR, χρησιμοποιούνται για τον προσδιορισμό της αρχικής λίστας ζευγαριών ανώτερων κόμβων στον DAG. Έπειτα εφόσον έχει προστεθεί κάθε ανώτερη ακμή, αυτές οι δυο δομές δεδομένων και η λίστα με τα ζευγάρια κόμβων ανώτερης τάξης, ενημερώνονται ώστε να είναι συνεπείς με τον μετασχηματισμένο DAG. Οι ακόλουθες γραμμές αποτελούν μια περίληψη του αλγορίθμου :

ΑΛΓΟΡΙΘΜΟΣ (G) ΚΟΜΒΟΥ ΑΝΩΤΕΡΗΣ ΤΑΞΗΣ

```
1 κατασκεύασε πίνακα απόστασης και πίνακα SUPERIOR
2 δημιούργησε λίστα κόμβων ανώτερης τάξης
3 όσο η λίστα δεν είναι κενή
4   απομάκρυνε το ζευγάρι κόμβων (i,j) από την λίστα
5   πρόσθεσε ακμή «latency-zero» (i,j) στο σύνολο ακμών E(G)
6   πρόσθεσε ακμές πόρων αν είναι απαραίτητο
7   ενημέρωσε τους δυο πίνακες και απομάκρυνε περιττές ακμές
8   ενημέρωσε τη λίστα κόμβων ανώτερης τάξης
```

Η σειρά με την οποία τα ζευγάρια κόμβων απομακρύνονται από τη λίστα των κόμβων ανώτερης τάξης, δεν επηρεάζει σημαντικά τον τελικό μετασχηματισμό. Ωστόσο, για την αύξηση της χρονικής περιοχής της δομής των δεδομένων και τη μεγιστοποίηση της απόδοσης της μνήμης cache (Hennesy and Patterson, 2002) **[22]**, τα ζεύγη κόμβων προστίθενται ή αφαιρούνται από τη λίστα κόμβων ανώτερης τάξης με σειρά προτεραιότητας της μορφής LIFO (Last In First Out).

Ένα μεγάλο μέρος της πολυπλοκότητας του παραπάνω αλγορίθμου, εμπεριέχει την αποδοτική ενημέρωση του πίνακα απόστασης και του πίνακα SUPERIOR. Η προσθήκη μιας ανώτερης ακμής (i, j), οδηγεί σε μηδενική αύξηση της απόστασης (χωρίς μεταβολή) από τον κόμβο i στον κόμβο j, και ίσως αυξήσει τις αποστάσεις ανάμεσα στους προηγηθέντες κόμβους του i και

τους διαδόχους του j. Ο πίνακας απόστασης ενημερώνεται με την εκτέλεση μιας σε βάθος διερεύνησης (Depth-First Traversal) των διαδόχων του κόμβου j. Σε κάθε διάδοχο κόμβο k, εάν η απόσταση (i,k) έχει αυξηθεί μετά την προσθήκη του (i,j), το αντίστοιχο στοιχείο του πίνακα απόστασης είναι αυξημένο. Για κάθε τέτοιο k με μια ενημερωμένη απόσταση από τον κόμβο i, μια άλλη διερεύνηση εκτελείται προς την αντίθετη κατεύθυνση από τον κόμβο i, για να ενημερωθούν οι αποστάσεις από τους προηγηθέντες του κόμβου i αναφορικά με τον k.

Καθώς κάθε στοιχείο του πίνακα απόστασης ενημερώνεται, οποιεσδήποτε ακμές οι οποίες έχουν καταστεί πλεονάζουσες λόγω της αυξημένης απόστασης, αφαιρούνται. Έτσι μειώνεται ο αριθμός των ακμών στον DAG και μπορεί να βελτιωθεί η εκτέλεση του αλγορίθμου. Από τη στιγμή που ένα στοιχείο στον πίνακα απόστασης ενημερώνεται σε μέγιστη καθυστέρηση (*max-latency)*, επιπλέον ενημερώσεις για το στοιχείο δεν είναι απαραίτητες. Εάν η απόσταση ανάμεσα σε δυο κόμβους αυξάνεται σε $-\infty$ (στην πράξη σε έναν πολύ μεγάλο ακέραιο π.χ. INT max), οι δυο κόμβοι δεν είναι πλέον ανεξάρτητοι. Εάν το ζευγάρι κόμβων βρίσκεται στη λίστα ανώτερων κόμβων, τότε το ζευγάρι αφαιρείται από τη λίστα.

Ο πίνακας SUPERIOR ενημερώνεται κάθε φορά που στοιχείο του πίνακα απόστασης αυξάνεται. Για μια αύξηση της απόστασης από έναν κόμβο i σε έναν κόμβο j, τα στοιχεία «διάδοχοι» στον πίνακα SUPERIOR θα πρέπει να ενημερωθούν:

- `SUPERIOR (k,j) όπου k είναι ένας άμεσα διάδοχος του i`
- `SUPERIOR (i,k) όπου k είναι ένας άμεσα προηγηθείς του j`

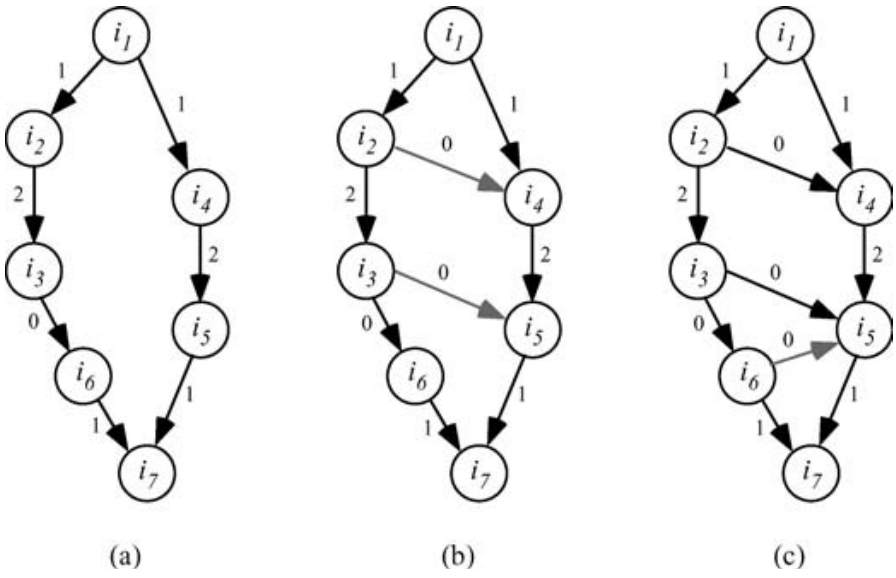
Επειδή οι αποστάσεις ανάμεσα στους κόμβους μπορούν μόνο να αυξάνονται, οι τιμές των στοιχείων του πίνακα SUPERIOR μειώνονται μονοτονικά κατά τη διάρκεια μετασχηματισμού στον κατευθυνόμενο άκυκλο γράφο. Εάν το στοιχείο του πίνακα SUPERIOR που σχετίζεται με ένα ζευγάρι κόμβων μειώνεται στο μηδέν, τότε αυτό το ζευγάρι προστίθεται στη λίστα κόμβων ανώτερης τάξης.

Η υπολογιστική πολυπλοκότητα της κατασκευής πίνακα SUPERIOR και της ενημέρωσης όλων των δομών δεδομένων, μπορεί να υπολογιστεί εξετάζοντας μόνο αναφορές ακμών, επειδή καμία άλλη λειτουργία δεν πραγματοποιείται χωρίς αναφορά σε τουλάχιστο μια ακμή. Όταν κατασκευάζεται ο πίνακας SUPERIOR κάθε ακμή αναφέρεται $O(n)$ φορές, επομένως ο χρόνος κατασκευής του πίνακα SUPERIOR είναι της τάξης $O(ne)$.

Για μια ενημέρωση της απόστασης μεταξύ δύο κόμβων i και j, μια συγκεκριμένη ακμή αναφέρεται μόνο αν αυτή τερματίζει στον κόμβο i ή έχει ως απαρχή τον κόμβο j. Δεδομένου ότι η απόσταση μεταξύ των προαναφερθέντων κόμβων ενημερώνεται το πολύ κατά έναν σταθερό αριθμό από φορές, μια ακμή αναφέρεται όχι περισσότερο από $O(n)$ φορές στο σύνολο των ενημερώσεων στον πίνακα της απόστασης. Κατά συνέπεια η συντήρηση του πίνακα της απόστασης απαιτεί $O(ne)$ χρόνο, όπως ισχύει και για τον πίνακα SUPERIOR. Άρα ο χρόνος εκτέλεσης χειρότερης περίπτωσης για τον αλγόριθμο κόμβου ανώτερης τάξης είναι $O(ne)$.

3.4 – Μετασχηματισμός υπογράφου ανώτερης τάξης

Η ανωτερότητα μπορεί να γενικευτεί από κόμβους σε έναν κατευθυνόμενο άκυκλο γράφο, σε υπογράφους ενός DAG. Υπό ορισμένες συνθήκες, ένας υπογράφος ενός DAG μπορεί να είναι ανώτερος από έναν διαμελισμένο (disjoint) υπογράφο, ο οποίος είναι ισόμορφος με τον πρώτο. Σε αυτή την περίπτωση, ανώτερες ακμές μηδενικής καθυστέρησης μπορούν να προστεθούν ανάμεσα στα αντίστοιχα ζευγάρια κόμβων στους υπογράφους. Λόγω του ότι ο μετασχηματισμός ανώτερου υπογράφου αποτελεί μια γενίκευση του μετασχηματισμού κόμβου ανώτερης τάξης, ίσως προστεθούν στον DAG σημαντικά περισσότερες ανώτερες ακμές.€



Σχήμα 3.3: *Παράδειγμα μετασχηματισμού υπογράφου ανώτερης τάξης*

Γενικά ένας υπογράφος που επάγεται από το σύνολο των κόμβων S, αποτελείται από τους κόμβους S και από όλες τις ακμές στο σύνολο E(G)

(σύνολο των ακμών), ανάμεσα στους κόμβους του S. Έστω λοιπόν ότι A είναι ο υπογράφος που επάγεται από το σύνολο των κόμβων $\{\alpha_1,\ldots,\alpha_m\}$ στον γράφο G. Επίσης θεωρούμε ότι B είναι ο υπογράφος που επάγεται από το σύνολο $\{b_1,\ldots,b_m\}$, διαμελισμένο από τους κόμβους του A. Τότε το A είναι ισόμορφο με το B εάν:

- Για κάθε $r\in\{1,\ldots,m\}$, τύπος (α_r) = τύπος (b_r).
- Για κάθε ακμή (α_r,α_s) στο E(A), υπάρχει μια ακμή (b_r,b_s) στο E(B) της ίδιας καθυστέρησης.
- Για κάθε ακμή (b_r,b_s) στο E(B), υπάρχει μια ακμή (α_r,α_s) στο E(A) της ίδιας καθυστέρησης.

Υπό αυτές τις συνθήκες, ο κόμβος α_r αντιστοιχίζεται με τον κόμβο b_r για κάθε $r\in\{1,\ldots,m\}$. Για παράδειγμα στο σχήμα 3.3(a), ο υπογράφος που επάγεται από τους δυο κόμβους i_2 και i_3, είναι ισόμορφος με τον υπογράφο που επάγεται από τους κόμβους i_4 και i_5. Ο κόμβος i_2 αντιστοιχίζεται με τον κόμβο i_4 και ο κόμβος i_3 με τον κόμβο i_5.

Για τους δυο υπογράφους A και B με τον τρόπο που ορίστηκαν, ισχύει ότι η υπό προϋποθέσεις προσθήκη μιας ακμής μηδενικής καθυστέρησης από τον κόμβο α_r στον κόμβο b_r για κάθε $r\in\{1,\ldots,m\}$, είναι ένας βέλτιστος μετασχηματισμός. Αυτές είναι οι προϋποθέσεις που δηλώνουν για παράδειγμα, εάν ο υπογράφος A είναι ανώτερης τάξης από τον υπογράφο B. Ωστόσο, τόσο ο A όσο και ο B αποτελούν μαζί ένα ζευγάρι υπογράφων ανώτερης τάξης. Όσον αφορά τις ανώτερες ακμές μηδενικής καθυστέρησης (latency-zero), αυτές προστίθενται από κάθε κόμβο του A στον αντίστοιχο κόμβο του B. Όπως συμβαίνει και με τον μετασχηματισμό κόμβου ανώτερης τάξης, η διαδικασία του μετασχηματισμού υπογράφου μπορεί να εφαρμόζεται επαναληπτικά.

Έστω ότι έχουμε ως παράδειγμα τον DAG του σχήματος 3.3(a), όπου ο υπογράφος που επάγεται από τους κόμβους $\{i_2,i_3\}$ είναι ανώτερος από αυτόν που προκύπτει από τους κόμβους $\{i_4,i_5\}$, ενώ παράλληλα γίνεται προσθήκη των ακμών (i_2,i_4) και (i_3,i_5). Στον τελικό DAG του σχήματος 3.3(b), ο κόμβος i_6 είναι ανώτερος από τον i_5 και η προσθήκη της αντίστοιχης ανώτερης ακμής απεικονίζεται στο σχήμα 3.3(c). Σημειώνεται ότι οι κόμβοι ανώτερης τάξης, είναι εκφυλισμένες περιπτώσεις υπογράφων ανώτερης τάξης.

3.5 – Αλγόριθμος υπογράφου ανώτερης τάξης

Ο αλγόριθμος που θα περιγραφεί σε αυτή την ενότητα, είναι ένας αποτελεσματικός αλγόριθμος που προσδιορίζει και μετασχηματίζει πολλούς από τους ανώτερους υπογράφους στον DAG. Πρόκειται για αλγόριθμο που

εφαρμόζεται επαναληπτικά, μέχρι να εξαντληθούν οι επιπλέον περιπτώσεις υπογράφων ανώτερης τάξης.

Η λειτουργία του αλγορίθμου διέπεται από δυο συνιστώσες. Η πρώτη προσδιορίζει τα ζευγάρια κόμβων προέλευσης, τα οποία μπορεί να είναι κόμβοι ρίζας (root nodes) και κόμβοι φύλλου (leaf nodes) των υπογράφων ανώτερης τάξης. Η δεύτερη συνιστώσα αναζητά στον DAG ένα ζευγάρι υπογράφου ανώτερης τάξης, στην εγγύτητα ενός ζεύγους κόμβων προέλευσης. Εάν βρεθεί κάποιο ζεύγος υπογράφων, τότε έχουμε μετασχηματισμό στον DAG και το σύνολο των κόμβων προέλευσης ενημερώνεται.

Οι κόμβοι προέλευσης που χρησιμοποιούνται από τον αλγόριθμο είναι ζευγάρια «ημιανώτερων» κόμβων. Ο κόμβος i είναι ημιανώτερος (semisuperior) του κόμβου j, εάν τόσο οι i και j είναι ανεξάρτητοι μεταξύ τους, είναι του ιδίου τύπου και ικανοποιούν ή τη μια ή την άλλη από τις ακόλουθες προϋποθέσεις (απόρροια του θεωρήματος που αναφέρθηκε στην ενότητα 3.3) :

- Για κάθε κόμβο k που ανήκει στους προηγηθέντες του κόμβου i, όπου (k,i) δεν αποτελεί ανώτερη ακμή, ισχύει ότι $\text{ΚΑΘΥΣΤΕΡΗΣΗ}(k,i) \le \text{ΑΠΟΣΤΑΣΗ}(k,j)$.
- Για κάθε κόμβο l που ανήκει στους διαδόχους του κόμβου j, όπου (j,l) δεν αποτελεί ανώτερη ακμή, ισχύει ότι $\text{ΚΑΘΥΣΤΕΡΗΣΗ}(j,l) \le \text{ΑΠΟΣΤΑΣΗ}(i,l)$.

Στο σημείο αυτό θα πρέπει να διαχωρίσουμε τις δυο έννοιες, καθώς η «*ΚΑΘΥΣΤΕΡΗΣΗ*» εξαρτάται από τον συντελεστή (βάρος) μιας ακμής που συνδέει δυο κόμβους στον DAG, υποδηλώνοντας τον ελάχιστο χρόνο μετάβασης από τον ένα στον άλλο. Η έννοια για την «*ΑΠΟΣΤΑΣΗ*» διαφέρει, σχετιζόμενη κυρίως με τον αριθμό των ακμών συντομότερης διαδρομής οι οποίες θα συνδέσουν δυο κόμβους στον DAG.

Κάθε ζευγάρι υπογράφων ανώτερης τάξης πρέπει να περιλαμβάνει ζευγάρια ημιανώτερων κόμβων. Εάν ο υπογράφος Α είναι ανώτερος από τον υπογράφο Β, κάθε κόμβος ρίζας του Α είναι ημιανώτερος από τον αντίστοιχο κόμβο ρίζας στον υπογράφο Β. Επίσης κάθε κόμβος φύλλου του Α είναι ημιανώτερος από τον αντίστοιχο κόμβο φύλλου στον υπογράφο Β. Για παράδειγμα, στους υπογράφους που επάγονται από τις ακμές $\{i_2, i_3\}$ και $\{i_4, i_5\}$ του σχήματος 3.3(a), ο κόμβος ρίζας i_2 του υπογράφου είναι ημιανώτερος από τον αντίστοιχο κόμβο ρίζας i_4. Κάτι ανάλογο συμβαίνει και με τον κόμβο φύλλου i_3, ο οποίος είναι ημιανώτερος από τον αντίστοιχο κόμβο φύλλου i_5 στον υπογράφο.

Τα ζεύγη ημιανώτερων κόμβων προσδιορίζονται με τη χρήση μιας ευθέως τροποποίησης του αλγορίθμου κόμβου ανώτερης τάξης, που περιγράφηκε στην

ενότητα 3.3 . Ο πίνακας SUPERIOR από τον αλγόριθμο κόμβου ανώτερης τάξης, χωρίζεται σε δυο πίνακες. Για τους κόμβους i και j, το αντίστοιχο στοιχείο στον πρώτο πίνακα ισούται με τον αριθμό των άμεσα προηγηθέντων κόμβων του i. Όμοια, το αντίστοιχο στοιχείο του δεύτερου πίνακα ισούται με τον αριθμό των άμεσα διαδόχων κόμβων του j. Στην περίπτωση που το αντίστοιχο στοιχείο ή στον ένα ή στον άλλο πίνακα είναι μηδέν, ο κόμβος i είναι ημιανώτερος του κόμβου j.

Ο αλγόριθμος υπογράφου ανώτερης τάξης, ακολουθεί τη δομή του αλγορίθμου κόμβου ανώτερης τάξης. Οι ακόλουθες γραμμές αποτελούν μια περίληψη του αλγορίθμου :

ΑΛΓΟΡΙΘΜΟΣ (G) ΥΠΟ-ΓΡΑΦΟΥ ΑΝΩΤΕΡΗΣ ΤΑΞΗΣ

```
1  κατασκεύασε πίνακα απόστασης και πίνακες ημιανωτερότητας
2  δημιούργησε λίστα κόμβων ημιανώτερης τάξης
3  όσο η λίστα δεν είναι κενή
4     απομάκρυνε το ζευγάρι κόμβων (I,j) από την πρώτη θέση
      της λίστας των κόμβων ημιανώτερης τάξης
5     αναζήτηση υπο-γράφου (∅,∅,{(i,j)})
6     εάν βρεθεί ζεύγος ανώτερου υπογράφου
7       πρόσθεσε ανώτερες ακμές ανάμεσα στους υπογράφους
8       πρόσθεσε ακμές πόρων εάν κρίνεται απαραίτητο
9       ενημέρωσε τον πίνακα απόστασης, τους πίνακες
        ημιανωτερότητας, και απομάκρυνε τις περιττές ακμές
10      ενημέρωσε τη λίστα κόμβων ημιανώτερης τάξης
```

Όπως συμβαίνει και με τον αλγόριθμο μετασχηματισμού κόμβου ανώτερης τάξης, τα ζευγάρια των κόμβων ημιανώτερης τάξης προστίθενται και απομακρύνονται από την πρώτη θέση της λίστας ημιανώτερων κόμβων, για τη μεγιστοποίηση της απόδοσης της λανθάνουσας (κρυφής) μνήμης (cache).

Η αναζήτηση ανώτερου υπογράφου στη γραμμή 5, αρχίζει με ένα ζεύγος ημιανώτερων κόμβων (i, j). Αυτοί σχηματίζουν δυο υπογράφους Α και Β, αποτελούμενους από έναν μοναδικό κόμβο ο καθένας. Ο κόμβος i βρίσκεται στον Α και ο κόμβος j βρίσκεται στον Β. Το αντικείμενο της αναζήτησης είναι η επιλογή επιπλέον κόμβων για προσθήκη στους υπογράφους Α και Β, μέχρι ο Α να είναι ανώτερος του Β. Εάν οι αρχικοί υπογράφοι Α και Β είναι ανώτερης τάξης (ο κόμβος i είναι ανώτερος από τον κόμβο j), κοινότοπα δεν είναι απαραίτητη κάποια επιπλέον αναζήτηση. Σε αντίθετη περίπτωση, υφίστανται άμεσα προηγηθέντες του i (άμεσα διάδοχοι του j), οι οποίοι πρέπει να προστίθενται στους υπογράφους για να ικανοποιούνται οι προϋποθέσεις ανωτερότητας για τους Α και Β. Συνεπώς, κάθε τέτοιος άμεσα προηγηθείς κόμβος του i (άμεσα διάδοχος του j), αντιστοιχίζεται σε έναν άμεσα προηγηθέντα κόμβο του j (άμεσα διάδοχο του i) του ιδίου τύπου, και

προστίθενται στους υπογράφους Α και Β αντίστοιχα. Για κάθε νέο ζευγάρι κόμβων (a,b) που προστίθενται στους υπογράφους, η ίδια ανάλυση για τους άμεσα προηγηθέντες και τους άμεσα διαδόχους εκτελείται για τα a και b, και περισσότεροι κόμβοι μπορεί να προστεθούν στους υπογράφους. Σε γενικές γραμμές, πολλαπλές αντιστοιχίσεις από άμεσα προηγηθέντες και άμεσα διαδόχους κόμβους είναι ίσως πιθανές, έτσι ώστε μια «προς τα πίσω» αναζήτηση (backtracking search) χρησιμοποιείται για τη διερεύνηση των πιθανών παραλλαγών. Εάν καμία αντιστοίχιση δεν είναι πιθανή για κάποια (a,b) ή έχουν εξαντληθεί όλες οι αντιστοιχίσεις, η αναζήτηση υπαναχωρεί. Οι κόμβοι a και b απομακρύνονται από τους υπογράφους και μια διαφορετική αντιστοίχιση επιδιώκεται για το ζευγάρι των κόμβων που προστέθηκε προηγουμένως. Δεδομένου ότι η αναζήτηση απαιτεί εκθετικό χρόνο στη χειρότερη περίπτωση, ο αριθμός των υπαναχωρήσεων περιορίζεται σε μια σταθερά προτού σταματήσει την αναζήτηση. Η υπορουτίνα **ΑΝΑΖΗΤΗΣΗ ΥΠΟΓΡΑΦΟΥ** περίληψη της οποίας φαίνεται παρακάτω, είναι μια αναδρομική υλοποίηση αυτού του αλγόριθμου αναζήτησης.

```
ΑΝΑΖΗΤΗΣΗ ΥΠΟΓΡΑΦΟΥ (A,B,S)
1  εάν S = ∅
2     ο υπογράφος Α είναι ανώτερος του Β, διακοπή αναζήτησης
3  επέτρεψε (a,b) να είναι ένα ζεύγος κόμβων στο S
4  για κάθε αντιστοιχία Μ των ΠΡΟΗΓΗΘΕΙΣ(a) σε ΠΡΟΗΓΗΘΕΙΣ(b)
   και ΔΙΑΔΟΧΟΣ(a) σε ΔΙΑΔΟΧΟΣ(b)
5     ΑΝΑΖΗΤΗΣΗ ΥΠΟΓΡΑΦΟΥ (A∪{a}, B∪{b}, M∪S−{(a,b)})
6  αύξησε τον αριθμό των υπαναχωρήσεων
7  εάν οι υπαναχωρήσεις είναι περισσότερες από το μέγιστο
   επιτρεπόμενο αριθμό
8     αποτυχία αναζήτησης, διακοπή αναζήτησης
9  επιστροφή
```

Η *ΑΝΑΖΗΤΗΣΗ ΥΠΟΓΡΑΦΟΥ* λαμβάνει τρεις παραμέτρους. Οι δυο πρώτες, Α και Β, είναι τα σύνολα των κόμβων που σχηματίζουν τους δυο υπογράφους. Η τρίτη παράμετρος S, αποτελεί ένα σύνολο από ζευγάρια κόμβων που πρέπει να προστεθούν στους υπογράφους, ώστε να ικανοποιούνται οι προϋποθέσεις ανωτερότητας. Κάθε επανάληψη της ΑΝΑΖΗΤΗΣΗΣ ΥΠΟΓΡΑΦΟΥ απομακρύνει ένα ζεύγος κόμβων (a,b) από το σύνολο S και προσπαθεί να αντιστοιχίσει, τους άμεσα προηγηθέντες κόμβους του a που δεν ικανοποιούν τις προϋποθέσεις ανωτερότητας, τους άμεσα προηγηθέντες κόμβους του b όπως περιγράφηκε παραπάνω. Οι άμεσα διάδοχοι κόμβοι του b έχουν αντιστοιχηθεί παρόμοια. Η *ΑΝΑΖΗΤΗΣΗ ΥΠΟΓΡΑΦΟΥ* καλείται αναδρομικά για κάθε ξεχωριστή αντιστοίχιση. Η αναζήτηση τερματίζεται εάν προσδιοριστεί ένα ζευγάρι υπογράφων ανώτερης τάξης, εάν ξεπεραστεί το όριο του μέγιστου αριθμού υπαναχωρήσεων ή εάν

διερευνηθούν όλοι οι πιθανοί υπογράφοι ανώτερης τάξης που περιέχουν τους κόμβους i και j.

Ο προσδιορισμός ενός ζεύγους υπογράφων ανώτερης τάξης, έχει ως επακόλουθο ανώτερες ακμές να προστίθενται ανάμεσα στους κόμβους που έχουν αντιστοιχηθεί στους υπογράφους. Ακμές πόρων προστίθενται εάν κριθεί απαραίτητο και οι περιττές ακμές απομακρύνονται. Οι μέθοδοι για την ενημέρωση των δομών δεδομένων που χρησιμοποιούνται από τον αλγόριθμο, είναι παρόμοιες με τις ομόλογές τους στον αλγόριθμο κόμβου ανώτερης τάξης.

Η υπολογιστική πολυπλοκότητα της κατασκευής και της συντήρησης των δομών δεδομένων για τον αλγόριθμο ανώτερου υπογράφου, είναι πανομοιότυπη με τον αλγόριθμο κόμβου ανώτερης τάξης, $O(ne)$ χρόνου. Ένα ζευγάρι κόμβων προστίθενται στη λίστα ημιανώτερων κόμβων, όταν το αντίστοιχο στοιχείο ενός από τους δυο πίνακες ημιανωτερότητας μειώνεται σε μηδέν. Έτσι, οποιοδήποτε ζευγάρι κόμβων μπορεί να προστεθεί μόνο δύο φορές και το μέγεθος της λίστας περιορίζεται σε $O(n^2)$. Κατά συνέπεια το ίδιο όριο ισχύει και για τον αριθμό αναζητήσεων στον υπογράφο. Κάθε αναζήτηση από ένα ζευγάρι κόμβων είναι της τάξης $O(e)$. Ο χρόνος εκτέλεσης χειρότερης περίπτωσης για τον αλγόριθμο υπογράφου ανώτερης τάξης είναι $O(n^2e)$.

3.6 – Μετασχηματισμός κόμβου ημιανώτερης τάξης

Δύο είναι οι μετασχηματισμοί τους οποίους θα περιγράψουμε σε αυτή την ενότητα: ο μετασχηματισμός προηγηθέντα κόμβου ημιανώτερης τάξης και ο μετασχηματισμός διαδόχου κόμβου ημιανώτερης τάξης. Το κοινό τους γνώρισμα είναι ότι προσθέτουν ακμές ανώτερης τάξης μεταξύ των ζευγαριών κόμβων ημιανώτερης τάξης, στον γράφο εξάρτησης δεδομένων (Data Dependence Graph – DDG). Παρόλα ταύτα, οι προϋποθέσεις ημιανωτερότητας για έναν κόμβο είναι πιο γενικές συγκρίσει αυτών για την ανωτερότητα κόμβου. Απαιτούνται επιπρόσθετοι περιορισμοί για τους μετασχηματισμούς κόμβου ημιανώτερης τάξης, οπότε θα λέγαμε ως συμπέρασμα ότι πρόκειται περισσότερο για συμπλήρωμα παρά για μια γενίκευση του μετασχηματισμού κόμβου ανώτερης τάξης.

Ένας κόμβος a είναι παράλληλος σε έναν κόμβο b, εάν οι εξαρτήσεις στον γράφο επιτρέπουν στους δυο κόμβους να υπόκεινται σε χρονοπρογραμματισμό στον ίδιο κύκλο. Ισοδύναμα ο κόμβος a είναι παράλληλος στον b, εάν η απόσταση της συντομότερης διαδρομής από τον a στον b και αντιστρόφως, είναι μικρότερη ή ίση με το μηδέν. Κοινότοπα, ένας κόμβος είναι παράλληλος του εαυτού του. Ο παραλληλισμός είναι λιγότερο περιοριστικός από την ανεξαρτησία, διότι ένα μονοπάτι μηδενικού μήκους μπορεί να υφίσταται μεταξύ παράλληλων κόμβων.

Ακολούθως θα αναφέρουμε ένα θεώρημα (έστω Α) και ένα πόρισμα που σχετίζονται με τον συγκεκριμένο μετασχηματισμό. Σύμφωνα με το θεώρημα Α, η προσθήκη μιας ακμής μηδενικής καθυστέρησης από έναν κόμβο a σε έναν κόμβο b σε ένα γράφο εξάρτησης δεδομένων, είναι ένας βέλτιστος μετασχηματισμός εάν ικανοποιούνται οι παρακάτω προϋποθέσεις:

1. ο a είναι προηγηθείς κόμβος ημιανώτερης τάξης σε σχέση με τον b.
2. ο αριθμός των κόμβων του τύπου $TYPE(a)$ οι οποίοι δεν είναι διάδοχοι του a και παράλληλοι στον b, είναι μικρότερος ή ίσος με τις γραμμές επικοινωνίας (pipes) $TYPE(a)$.

Το πόρισμα με τη σειρά του δηλώνει, ότι η προσθήκη μιας ακμής μηδενικής καθυστέρησης από έναν κόμβο a σε έναν κόμβο b σε ένα γράφο εξάρτησης δεδομένων, είναι ένας βέλτιστος μετασχηματισμός εάν ικανοποιούνται οι παρακάτω προϋποθέσεις:

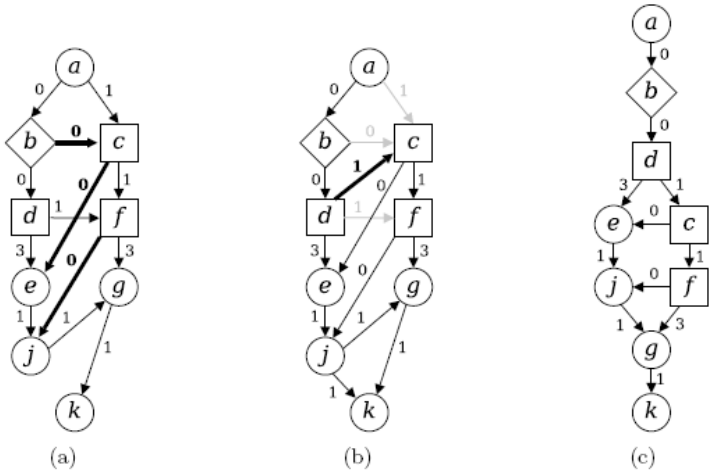
1. ο a είναι διάδοχος κόμβος ημιανώτερης τάξης σε σχέση με τον b.
2. ο αριθμός των κόμβων του τύπου $TYPE(b)$ οι οποίοι δεν είναι προηγηθέντες του b και παράλληλοι στον a, είναι μικρότερος ή ίσος με τις γραμμές επικοινωνίας $TYPE(b)$.

Οι αποδείξεις τόσο του θεωρήματος Α όσο και του πορίσματος, είναι συμμετρικές μεταξύ τους.

Ο μετασχηματισμός κόμβου ημιανώτερης τάξης, προσθέτει ακμές μεταξύ κόμβων ικανοποιώντας τις προϋποθέσεις του θεωρήματος Α και του πορίσματος που προαναφέρθηκαν. Όπως συμβαίνει και με τους μετασχηματισμούς κόμβου ανώτερης τάξης και υπογράφου ανώτερης τάξης, ο μετασχηματισμός κόμβου ημιανώτερης τάξης εφαρμόζεται επαναληπτικά σε έναν γράφο εξάρτησης δεδομένων, μέχρις ότου να μην είναι δυνατός κάποιος επιμέρους μετασχηματισμός.

Μια ακμή (a,b) που προστίθεται σε έναν μετασχηματισμό ημιανώτερης τάξης που έχει προηγηθεί, δεν δύναται να επηρεάσει έναν άπληστο (greedy) χρονοπρογραμματιστή ο οποίος προγραμματίζει με κατεύθυνση προς τα εμπρός (από τη ρίζα στα φύλλα στον DDG). Κατά τη διάρκεια του χρονοπρογραμματισμού, ο κόμβος a είναι απαραίτητα διαθέσιμος για να υποβληθεί στη διαδικασία, πριν ή κατά τη διάρκεια του κύκλου του κόμβου b, έτσι ώστε η προστιθέμενη ακμή επιβάλλει μια απόφαση χρονοπρογραμματισμού που γίνεται πάντα χωρίς την παρουσία της ίδιας της προστιθέμενης ακμής. Συμμετρικά οι ακμές που προστίθενται στον μετασχηματισμό διαδόχου κόμβου ημιανώτερης τάξης, δεν επηρεάζουν τον αντίστροφο μετασχηματισμό. Επιπλέον μια ακμή που προστίθεται στον μετασχηματισμό κόμβου ημιανώτερης τάξης, είναι απίθανο να έχει ως αποτέλεσμα την προσθήκη μιας ακμής πόρου εξαιτίας της δεύτερης

προϋπόθεσης που θέτει το προηγούμενο θεώρημα. Άρα, πολλές από τις ακμές που προστέθηκαν με τον μετασχηματισμό κόμβου ημιανώτερης τάξης, είναι απίθανο να επηρεάσουν άμεσα τα αποτελέσματα του χρονοπρογραμματισμού. Οι ακμές αυτές όμως μπορούν να εκθέσουν νέα ζευγάρια κόμβων ή υπογράφων ανώτερης τάξης, τα οποία μπορούν να βελτιώσουν τα παραπάνω αποτελέσματα. Κατά συνέπεια για τη μεγιστοποίηση της αποτελεσματικότητας, ο ανωτέρω μετασχηματισμός εφαρμόζεται με το μετασχηματισμό κόμβου ανώτερης τάξης και το μετασχηματισμό υπογράφου ανώτερης τάξης.



Σχήμα 3.4: *Παράδειγμα μετασχηματισμού κόμβου ημιανώτερης τάξης*

Θεωρούμε ότι (b,c) και (c,e), είναι τα ζευγάρια κόμβων ημιανώτερης τάξης που έχουν προηγηθεί μετά από τον μετασχηματισμό υπογράφου ανώτερης τάξης, και ότι οι δυο προϋποθέσεις του θεωρήματος ικανοποιούνται. Οι ανώτερες ακμές (b,c) και (c,e) προστίθενται όπως απεικονίζεται στο σχήμα 3.4(a). Όμοια, ο κόμβος f είναι προκάτοχος ημιανώτερος του j και οι προϋποθέσεις του πορίσματος ικανοποιούνται, ώστε η ανώτερη ακμή (f,j) να προστίθεται επίσης. Μετά την προσθήκη της ανώτερης ακμής (b,c), ο κόμβος d είναι ανώτερος από τον κόμβο c και μια ανώτερη ακμή προστίθεται όπως φαίνεται στο σχήμα 3.4(b). Το σχήμα 3.4(c) δείχνει έναν μετασχηματισμένο γράφο εξάρτησης δεδομένων, ο οποίος έχει διευθετηθεί εκ νέου για λόγους σαφήνειας. Οι μετασχηματισμοί έχουν σχεδόν διατάξει τους κόμβους αυτού του βασικού μπλοκ.

Ο μετασχηματισμός υπογράφου ανώτερης τάξης, είναι μια πιο δυναμική γενίκευση του μετασχηματισμού κόμβου ανώτερης τάξης. Ωστόσο, ο μετασχηματισμός υπογράφου ημιανώτερης τάξης ο οποίος γενικεύεται αναλογικά από τον μετασχηματισμό κόμβου ανώτερης τάξης, δεν είναι ο πλέον δυναμικός σε σχέση με τον μετασχηματισμό κόμβου ημιανώτερης τάξης. Γενικεύοντας, η προηγηθείσα ημιανωτερότητα αποδίδει κατ' αναλογία τις

ακόλουθες προϋποθέσεις για κάθε αντιστοιχισμένο ζευγάρι κόμβων a_i και b_i, σε ισόμορφους υπογράφους Α και Β:

1. ο κόμβος a_i είναι ανεξάρτητος του b_i.
2. για κάθε κόμβο p που ανήκει στους προηγηθέντες κόμβους του a_i, ισχύει ότι $\text{ΚΑΘΥΣΤΕΡΗΣΗ}(p, a_i) \leq \text{ΑΠΟΣΤΑΣΗ}(p, b_i)$.
3. ο αριθμός των κόμβων του τύπου $TYPE(a_i)$ οι οποίοι δεν είναι διάδοχοι του a_i και παράλληλοι στον b_i, είναι μικρότερος ή ίσος με τις γραμμές επικοινωνίας $TYPE(a_i)$.

Στο σημείο αυτό θα πρέπει να αναφέρουμε το θεώρημα που δηλώνει ότι: ένας υπογράφος ο οποίος μπορεί να μετασχηματίζεται με τον μετασχηματισμό προηγηθέντος υπογράφου ημιανώτερης τάξης, υπό τις υφιστάμενες προϋποθέσεις μετασχηματίζεται ισοδύναμα με τον μετασχηματισμό προηγηθέντος κόμβου ημιανώτερης τάξης.

3.7 – Αλγόριθμος κόμβου ημιανώτερης τάξης

Σε αυτή την ενότητα, περιγράφεται ο αποδοτικός αλγόριθμος για την εκτέλεση του μετασχηματισμού προηγηθέντος κόμβου ημιανώτερης τάξης. Όσον αφορά τον αλγόριθμο για τον μετασχηματισμό διαδόχου κόμβου ημιανώτερης τάξης, αυτός απαιτεί μόνο μια επιφανειακή τροποποίηση.

Το θεώρημα Α όπως είδαμε νωρίτερα, ορίζει τις δυο προϋποθέσεις που είναι αναγκαίες για την προσθήκη μιας ανώτερης ακμής μεταξύ δυο κόμβων. Καταρχήν οι δυο κόμβοι θα πρέπει να είναι προηγηθέντες και συνάμα ημιανώτερης τάξης. Τέτοιοι κόμβοι εντοπίζονται με την κατασκευή ενός πίνακα $PRED-SEMI$ – το όνομα προκύπτει από τη σχέση προτεραιότητας και ημιανωτερότητας – όπου τα προηγηθέντα ζευγάρια κόμβων ημιανώτερης τάξης, είναι εκείνα για τα οποία το αντίστοιχο στοιχείο στον πίνακα $PRED-SEMI$ είναι μηδέν.

Τα ζευγάρια των κόμβων που ικανοποιούν τη δεύτερη προϋπόθεση του θεωρήματος Α, εντοπίζονται με την κατασκευή ενός πίνακα από λίστες με το χαρακτηρισμό $PARALLEL$. Για τον κόμβο b και τον τύπο t, η λίστα $PARALLEL[b,t]$ είναι μια λίστα από κόμβους του τύπου t η οποία είναι παράλληλη στον b. Εάν το μήκος της λίστας $PARALLEL[b,t]$ είναι μικρότερο από τον αριθμό των γραμμών επικοινωνίας του τύπου t, $PIPES(t)$, η δεύτερη προϋπόθεση του θεωρήματος Α ικανοποιείται απαραίτητα για τον κόμβο b και για οποιονδήποτε κόμβο του τύπου t. Όπως συμβαίνει με τον πίνακα $PRED-SEMI$, η συστοιχία $PARALLEL$ κατασκευάζεται χρησιμοποιώντας έναν πίνακα με τις αποστάσεις για τη συντομότερη διαδρομή. Για λόγους αποτελεσματικότητας, ο αλγόριθμος εντοπίζει τα ζευγάρια κόμβων τα οποία

είναι κάπως πιο περιορισμένα από το θεώρημα Α. Για το ζευγάρι των κόμβων a και b, το θεώρημα Α απαιτεί ο αριθμός των κόμβων του τύπου $TYPE(a)$ που δεν είναι διάδοχοι του κόμβου a και είναι παράλληλοι του κόμβου b, να είναι μικρότερος ή ίσος με τις γραμμές επικοινωνίας $TYPE(a)$. Ο πίνακας $PARALLEL$, συντηρητικά λογοδοτεί για όλους τους κόμβους που είναι παράλληλοι με τον κόμβο b του τύπου $TYPE(a)$ σε αυτή την ανισότητα, συμπεριλαμβανομένων των διαδόχων του a.

Εάν ο πίνακας $PRED-SEMI[a,b]$ ισούται με το μηδέν και το μήκος της λίστας $PARALLEL[b,TYPE(a)]$ είναι μικρότερο ή ίσο με τις γραμμές επικοινωνίας $TYPE(a)$, οι προϋποθέσεις του θεωρήματος Α ικανοποιούνται για τους κόμβους a και b, και μια ανώτερη ακμή (a,b) μπορεί να προστεθεί στο γράφο εξάρτησης δεδομένων (DDG). Κατά το χρόνο κατασκευής του πίνακα $PRED-SEMI$ και των λιστών $PARALLEL$, όλα τα αρχικά ζευγάρια κόμβων εντοπίζονται και τοποθετούνται στη λίστα κόμβων ημιανώτερης τάξης. Έπειτα κατά την εκτέλεση του αλγορίθμου, τα ζευγάρια των κόμβων μετακινούνται από τη λίστα αυτή και μετασχηματίζονται ένα προς ένα, με τρόπο όμοιο με αυτόν του αλγορίθμου ανώτερης τάξης που περιγράφεται στην ενότητα 3.3 . Καθώς οι ακμές προστίθενται, οι δομές δεδομένων ενημερώνονται για να είναι σύμφωνες με τον μετασχηματισμένο γράφο DDG. Στη συνέχεια παρατίθεται μια περίληψη της δομής του αλγορίθμου κόμβου ημιανώτερης τάξης.

ΑΛΓΟΡΙΘΜΟΣ ΚΟΜΒΟΥ ΗΜΙΑΝΩΤΕΡΗΣ ΤΑΞΗΣ

```
1  κατασκεύασε πίνακα απόστασης, πίνακα PRED-SEMI και λίστες
   PARALLEL
2  δημιούργησε λίστα κόμβων ημιανώτερης τάξης
3  όσο η λίστα κόμβων ημιανώτερης τάξης δεν είναι κενή
4    απομάκρυνε το ζευγάρι κόμβων (a,b) από την πρώτη θέση της
     λίστας
5    πρόσθεσε ακμή «latency-zero» (a,b)
6    πρόσθεσε ακμές πόρων αν είναι απαραίτητο
7    ενημέρωσε τις δομές δεδομένων και απομάκρυνε περιττές
     ακμές
8    ενημέρωσε τη λίστα κόμβων ημιανώτερης τάξης
```

Μετά την προσθήκη μιας ανώτερης ακμής, ενημερώνεται πρώτα ο πίνακας απόστασης όπως συμβαίνει και με τον αλγόριθμο κόμβου ανώτερης τάξης. Έτσι με κάθε αυξητική αλλαγή στον πίνακα απόστασης, ενημερώνονται ο πίνακας $PRED-SEMI$ και οι λίστες $PARALLEL$. Οι λειτουργίες $ΕΝΗΜΕΡΩΣΗ\_ΗΜΙΑΝΩΤΕΡΟΥ$ και $ΕΝΗΜΕΡΩΣΗ\_PARALLEL$ ενημερώνουν τους πίνακες και εντοπίζουν περιπτώσεις όπου μια ανώτερη ακμή μπορεί να προστίθεται. Κάθε διεργασία καλείται, όταν η απόσταση της συντομότερης διαδρομής από έναν κόμβο a σε έναν κόμβο b αυξάνεται από d_{old} σε d, ακολουθώντας την προσθήκη μιας ανώτερης ακμής ή μιας ακμής πόρου.

ΕΝΗΜΕΡΩΣΗ _ ΗΜΙΑΝΩΤΕΡΟΥ(a, b, d_{old}, d)

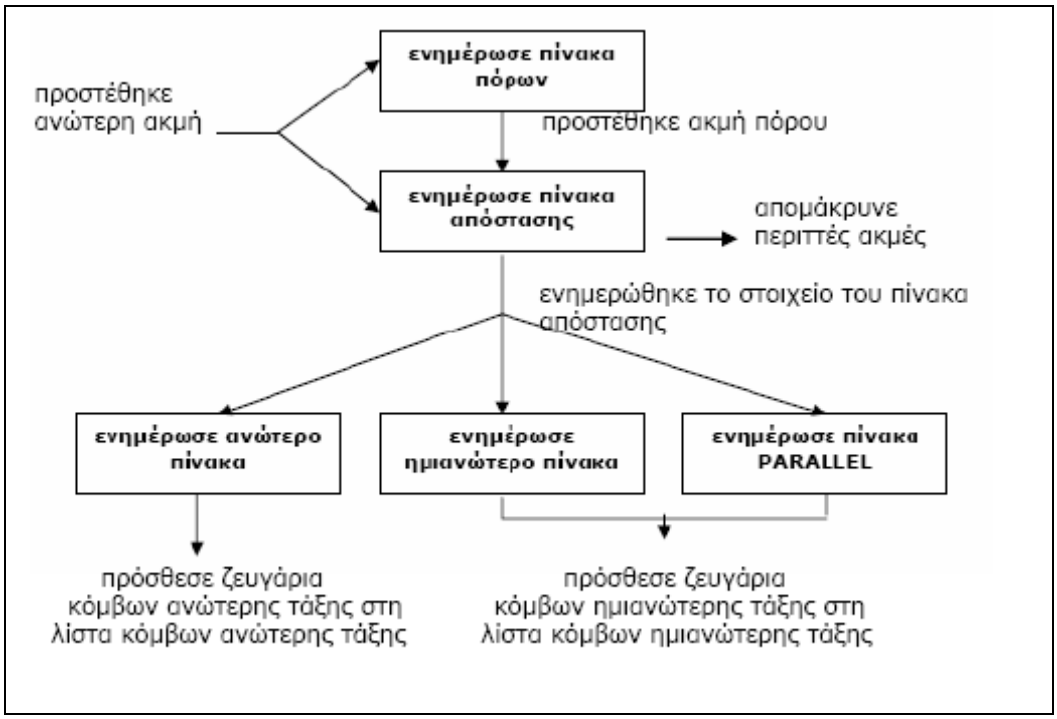
1 # απόσταση της συντομότερης διαδρομής από τον a στον b που αυξάνεται από d_{old} σε d
2 για κάθε $s \in ISSUC_a$
3 εάν $d_{old} < LATENCY_{(a,s)} \leq d$
4 # η ανισότητα της προηγηθείσας ημιανωτερότητας ικανοποιείται τώρα για την ακμή (a, s)
5 μείωση $PRED - SEMI[s, b]$
6 εάν $PRED - SEMI[s, b] = 0$
7 και $|PARALLEL[b, TYPE(s)]| \leq PIPES(TYPE(s))$
8 πρόσθεσε (s, b) στους κόμβους ημιανώτερης τάξης

ΕΝΗΜΕΡΩΣΗ _ $PARALLEL(a, b, d_{old}, d)$

1 # απόσταση της συντομότερης διαδρομής από τον a στον b που αυξάνεται από d_{old} σε d
2 εάν $d_{old} \leq 0$ και $d > 0$
3 # οι κόμβοι a και b δεν είναι πλέον παράλληλοι
4 απομάκρυνε τον κόμβο b από $PARALLEL[a, TYPE(b)]$
5 εάν $|PARALLEL[a, TYPE(b)]| = PIPES(TYPE(b))$
6 για κάθε p στον $PARALLEL[a, TYPE(b)]$
7 εάν $PRED - SEMI[p, a] = 0$
8 πρόσθεσε (p, a) στους κόμβους ημιανώτερης τάξης
9 απομάκρυνε τον κόμβο a από $PARALLEL[b, TYPE(a)]$
10 εάν $|PARALLEL[b, TYPE(a)]| = PIPES(TYPE(a))$
11 για κάθε p στον $PARALLEL[b, TYPE(a)]$
12 εάν $PRED - SEMI[p, b] = 0$
13 πρόσθεσε (p, b) στους κόμβους ημιανώτερης τάξης

Ο αλγόριθμος κόμβου ημιανώτερης τάξης και οι αλγόριθμοι ανώτερου κόμβου και υπογράφου, είναι δομικά όμοιοι και χρησιμοποιούν μερικές από τις ίδιες δομές δεδομένων. Άρα ο αλγόριθμος κόμβου ημιανώτερης τάξης, μπορεί να ενσωματωθεί αποτελεσματικά μέσα στον αλγόριθμο κόμβου (υπογράφου) ανώτερης τάξης, έτσι ώστε και οι δυο μετασχηματισμοί να εκτελούνται ταυτόχρονα. Για το σύνθετο αυτό αλγόριθμο διατηρούνται δυο λίστες από ζευγάρια κόμβων: η λίστα κόμβων ημιανώτερης τάξης και η λίστα κόμβων ανώτερης τάξης. Συγκεκριμένα ενώ εκτελείται ο βρόχος του αλγορίθμου, ένα ζευγάρι κόμβων απομακρύνεται από μια από τις λίστες και μια ανώτερη ακμή προστίθεται στον DDG (ή εκτελείται μια αναζήτηση υπογράφου). Η ενημέρωση

των δομών δεδομένων είναι κάπως περισσότερο πολύπλοκη, και απεικονίζεται στο σχήμα 3.5 για τους σύνθετους αλγόριθμους κόμβου ημιανώτερης και ανώτερης τάξης. Ο αλγόριθμος συνεχίζεται μέχρι και οι δυο λίστες να αδειάσουν.



Σχήμα 3.5: *Διάγραμμα ροής των ενημερώσεων δομών δεδομένων, στον αλγόριθμο κόμβου ανώτερης τάξης και κόμβου ημιανώτερης τάξης*

Η ταυτόχρονη εφαρμογή και των δυο μετασχηματισμών μπορεί να μετασχηματίσει το γράφο εξάρτησης δεδομένων εκτενέστερα, διότι μια ακμή που προστίθεται κατά τη διάρκεια του μετασχηματισμού ανώτερου κόμβου ή υπογράφου, μπορεί να αποκαλύψει επιπρόσθετα ζεύγη ημιανώτερων κόμβων και το αντίστροφο. Αυτή η ιδιότητα δεν είναι πλήρως αξιοποιήσιμη εάν οι μετασχηματισμοί εφαρμόζονται σε χωριστές φάσεις. Και οι δυο φάσεις θα μπορούσαν να επαναλαμβάνονται, μέχρις ότου να μην προστίθενται ακμές από τον ένα ή τον άλλο μετασχηματισμό. Ωστόσο αυτό είναι λιγότερο αποτελεσματικό από τον σύνθετο αλγόριθμο.

Το ζήτημα της υπολογιστικής πολυπλοκότητας που χαρακτηρίζει τον αλγόριθμο κόμβου ημιανώτερης τάξης, είναι ιδιαίτερης σημασίας και χρήζει αναφοράς. Ο ασυμπτωτικός χρόνος εκτέλεσης για την κατασκευή και διατήρηση τόσο του πίνακα $PRED-SEMI$ όσο και των πινάκων απόστασης, περιορίζεται σε $O(ne)$ όπως αναφέρθηκε και στην ενότητα 3.5 . Η συστοιχία που αποτελείται από τις λίστες $PARALLEL$, μπορεί να κατασκευαστεί σε χρόνο $O(n^2)$ εξετάζοντας όλα τα ζευγάρια των κόμβων και παραπέμποντας στον πίνακα απόστασης. Η διεργασία *ΕΝΗΜΕΡΩΣΗ_PARALLEL*, χειρίζεται όλες τις

ενημερώσεις για τον πίνακα *PARALLEL*. Συγκεκριμένα, δυο διαφορετικές λειτουργίες εκτελούνται στον πίνακα αυτό κατά τη διάρκεια εκτέλεσης της διεργασίας. Πρώτον ένα στοιχείο μπορεί να απομακρυνθεί από μια από τις λίστες του, όπως φαίνεται στις γραμμές 4 και 9 της *ΕΝΗΜΕΡΩΣΗΣ_PARALLEL.* Αυτό απαιτεί μόνο σταθερό χρόνο, εάν κάθε στοιχείο από κάθε λίστα *PARALLEL* σημειώνεται σε έναν πίνακα ευρετηρίου με τη μορφή ζεύγους. Το στοιχείο $[a,b]$ σε αυτόν τον πίνακα ευρετηρίου, δείχνει το a στη λίστα $PARALLEL[b,TYPE(a)]$ εάν όντως το a υφίσταται στη λίστα. Εφόσον η διεργασία *ΕΝΗΜΕΡΩΣΗ_PARALLEL* καλείται το πολύ σε χρόνο $O(n^2)$, η απομάκρυνση των στοιχείων από τις λίστες *PARALLEL* απαιτεί και αυτή το πολύ χρόνο $O(n^2)$. Η δεύτερη λειτουργία που επιτελείται στον πίνακα *PARALLEL* κατά τη διεργασία *ΕΝΗΜΕΡΩΣΗ_PARALLEL*, είναι η σάρωση μιας λίστας όπως υποδεικνύεται στις γραμμές 6 και 11. Επειδή ο αριθμός των στοιχείων στις λίστες *PARALLEL* μειώνεται μονοτονικά, κάθε λίστα είναι αποδέκτης μιας μοναδικής σάρωσης όταν ικανοποιούνται οι συνθήκες στις γραμμές 5 ή 10. Άρα, η σάρωση λιστών στην διεργασία *ΕΝΗΜΕΡΩΣΗ_PARALLEL* περιορίζεται σε $O(n^2)$, και ο ασυμπτωτικός χρόνος εκτέλεσης του αλγόριθμου κόμβου ημιανώτερης τάξης οριοθετείται σε $O(ne)$. Όταν πραγματοποιείται ταυτόχρονη εκτέλεση και των δυο μετασχηματισμών κόμβου (ανώτερου και ημιανώτερου) ισχύει η ίδια ανάλυση, και ο χρόνος κατά τον οποίο τρέχει ο σύνθετος αλγόριθμος κυριαρχείται από τη συνιστώσα του κόμβου ή του υπογράφου ανώτερης τάξης.

4

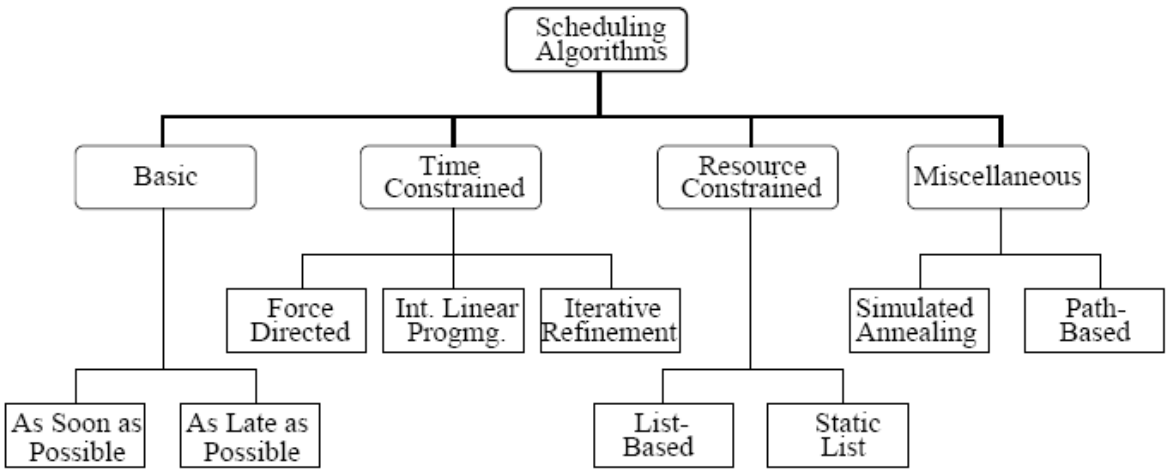
Ο χρονοπρογραμματισμός στην πράξη

4 – Ο χρονοπρογραμματισμός στην πράξη

Η έννοια του χρονοπρογραμματισμού έχει πλέον αναγνωριστεί, ως μια κύρια τεχνική βελτίωσης των επιδόσεων ενός προγράμματος. Για το λόγο αυτό υφίσταται ένα μεγάλο εύρος αλγορίθμων, οι οποίοι συμβάλλουν στην επίτευξη ικανοποιητικής χρονοδρομολόγησης. Το αντικείμενο αυτού του κεφαλαίου έχει να κάνει μια περιεκτική επισκόπηση μέσω παραδειγμάτων, τεχνικών χρονοπρογραμματισμού οι οποίες είναι ήδη γνωστές. Θα πραγματοποιηθεί αναφορά κυρίως σε παραδείγματα χρονοπρογραμματισμού ASAP και ALAP, καθώς επίσης και σε παράδειγμα χρονοπρογραμματισμού λίστας.

4.1 – Κατηγοριοποίηση αλγορίθμων

Με το πέρασμα του χρόνου, οι ερευνητές έχουν προσπαθήσει να καταλήξουν σε διάφορες λύσεις αναφορικά με το πρόβλημα του χρονοπρογραμματισμού. Αρκετοί αλγόριθμοι έχουν προταθεί, με κάθε έναν από αυτούς να εμφανίζει τα πλεονεκτήματα και τα μειονεκτήματά του. Οι αλγόριθμοι χρονοπρογραμματισμού που χρησιμοποιούνται γενικά, έχουν κατηγοριοποιηθεί σε τέσσερις διαφορετικές κατηγορίες: στους βασικούς (Basic) [εδώ ανήκουν οι ASAP και ALAP], στους αλγόριθμους περιορισμού χρόνου (Time Constrained), στους αλγόριθμους περιορισμού πόρου (Resource Constrained) και στους διάφορους ποικίλους αλγορίθμους (Miscellaneous). Εκτός όμως από αυτή, υπάρχουν επίσης και άλλες προσεγγίσεις ως προς το θέμα της κατηγοριοποίησης αλγορίθμων. Για παράδειγμα θα μπορούσαν να αναφερθούν στοχαστικές τεχνικές όπως η προσομοιωμένη ανόπτηση (Simulated Annealing).



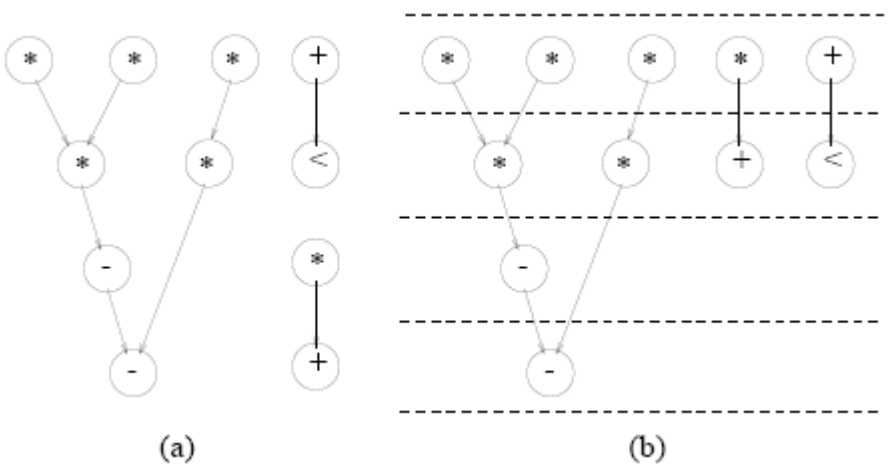
Σχήμα 4.1: *Κατηγοριοποίηση αλγορίθμων χρονοπρογραμματισμού*

4.2 – Βασικοί αλγόριθμοι χρονοπρογραμματισμού

Όπως είναι γνωστό, οι γράφοι ροής δεδομένων (DFGs) αποκαλύπτουν τον παραλληλισμό στη σχεδίαση. Ως επακόλουθο, κάθε κόμβος διαθέτει ένα εύρος από βήματα ελέγχου στο οποίο μπορεί να ανατεθεί. Οι αλγόριθμοι που θα περιγραφούν στη συνέχεια, ορίζουν χρονικά όρια μέσα στα οποία οι λειτουργίες του DFG μπορούν να χρονοπρογραμματιστούν. Προφανώς αναφερόμαστε στα πρώτα και απλούστερα σχήματα που χρησιμοποιούνται για τον προσδιορισμό των ορίων, τους γνωστούς αλγόριθμους ASAP και ALAP (ενότητες 1.8.2 και 1.8.3), τους οποίους εδώ θα τους δούμε μέσω παραδειγμάτων.

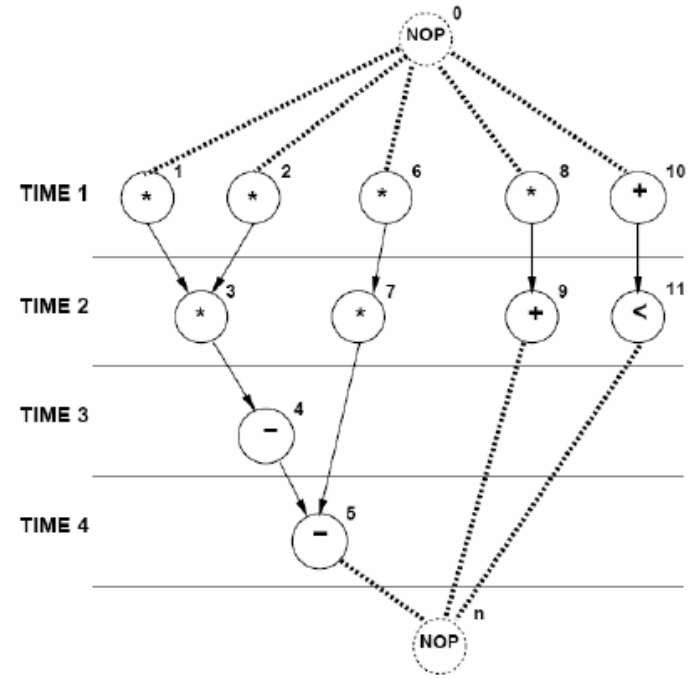
4.2.1 – Παραδείγματα ASAP (As-Soon-As-Possible)

Ο αλγόριθμος χρονοπρογραμματισμού ASAP, ξεκινά με τους κόμβους που βρίσκονται στην υψηλότερη θέση (χωρίς γονικούς κόμβους) στο γράφο ροής δεδομένων, αναθέτοντας χρονικά βήματα σε αύξουσα σειρά όσο κινείται καθοδικά (σχήμα 4.2). Ισχύει φυσικά ο απλός κανόνας, ότι ένας κόμβος μπορεί να εκτελείται μόνο μετά από την εκτέλεση του προκατόχου του (γονικός κόμβος). Ο αλγόριθμος αυτός δίνει σαφώς το ταχύτερο δυνατό χρονοπρόγραμμα. Με άλλα λόγια χρονοδρομολογεί με τον ελάχιστο αριθμό χρονικών βημάτων, χωρίς όμως να λαμβάνει υπόψιν περιορισμούς πόρου.



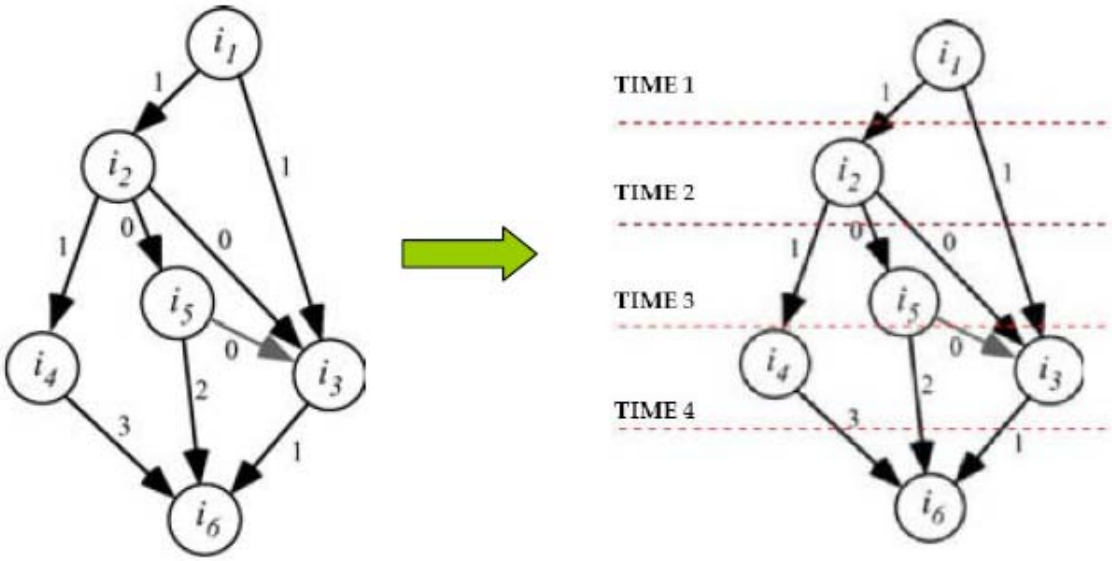
Σχήμα 4.2: *(α) Γράφος ροής δεδομένων (b) χρονοπρόγραμμα ASAP*

Στο σχήμα 4.3 που ακολουθεί, παρατίθεται ένα ακόμα παράδειγμα χρονοπρογραμματισμού ASAP με χρονική καθυστέρηση ίση με τέσσερα.



Σχήμα 4.3: *Παράδειγμα ASAP*

Το επόμενο παράδειγμα με το οποίο θα ασχοληθούμε, αναφέρεται στον χρονοπρογραμματισμό ASAP σε έναν DAG. Ο χρονοπρογραμματισμός σε αυτή την περίπτωση, επέρχεται με χρήση του αλγόριθμου μεγαλύτερου μονοπατιού ανάμεσα στον αρχικό και τον τελικό κόμβο του γράφου. Θεωρούμε λοιπόν τον μετασχηματισμένο γράφο του σχήματος 3.1(b) (σχήμα 4.4), στον οποίο θα εφαρμόσουμε την τεχνική ASAP. Παρατηρούμε ότι εδώ εμφανίζονται βάρη στις ακμές ανάμεσα στους κόμβους, τα οποία θα αναδείξουν το μεγαλύτερο μονοπάτι στο γράφο.

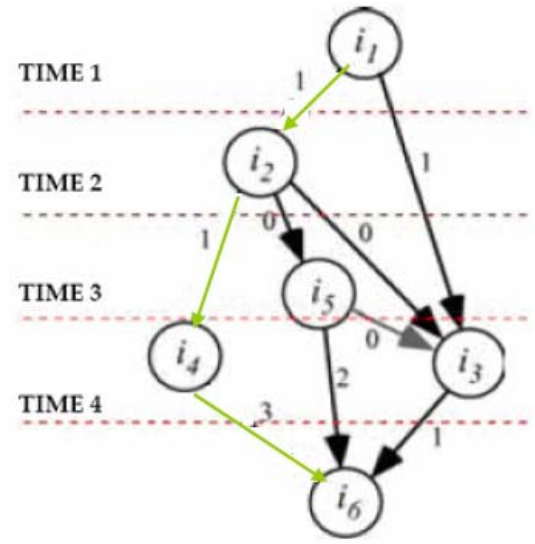


Σχήμα 4.4: *Χρονοπρογραμματισμός ASAP στο σχήμα 3.1(b)*

Η εύρεση του μεγαλύτερου μονοπατιού γίνεται με τη χρήση του παρακάτω αλγορίθμου. Θεωρούμε ότι η από πάνω προς τα κάτω σάρωση ενός γράφου G, θα δώσει ως έξοδο μια ακολουθία από κόμβους σε τοπολογική κατάταξη (αυτό μπορεί να υπολογιστεί μέσω μιας τοπολογικής κατάταξης, απαιτώντας ο γράφος εισόδου να είναι ένας DAG). Επιπλέον, έστω ότι $V(G)$ είναι το σύνολο των κόμβων του γράφου και $E(G)$ το σύνολο των ακμών στον γράφο. Εάν τα βάρη των ακμών είναι καθορισμένα, θεωρούμε (G,e) το βάρος μιας ακμής e στο γράφο G. Στην περίπτωση που ο γράφος δεν έχει βάρη για τις ακμές, εισάγουμε αυθαίρετα μια σταθερά εκτός από το μηδέν για οποιαδήποτε εμφάνιση βάρους (G,e). Ο αλγόριθμος τότε έχει ως εξής:

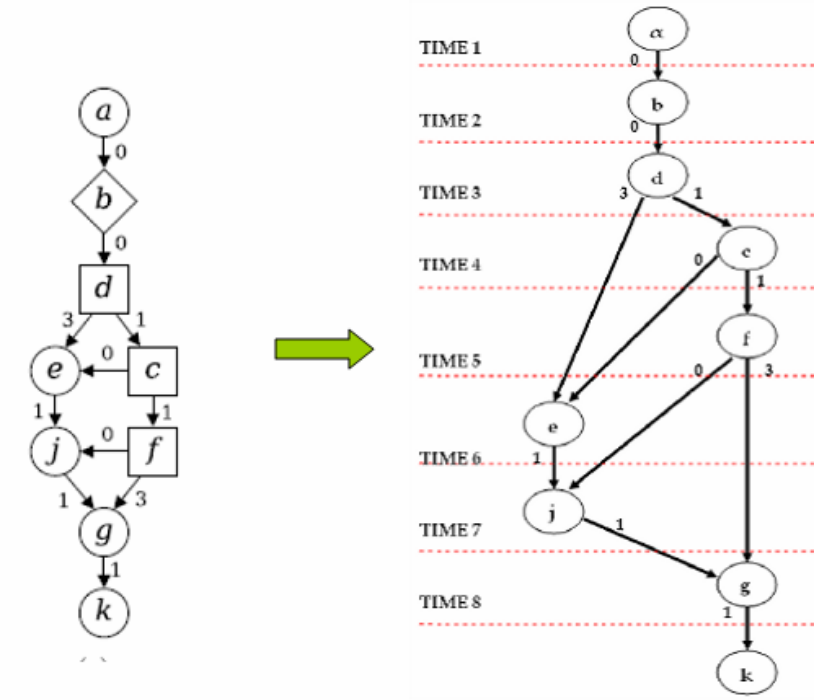
```
αλγόριθμος dag-longest-path
    είσοδος:
         Κατευθυνόμενος άκυκλος γράφος G
    έξοδος:
         Μήκος μεγαλύτερου μονοπατιού

length_to = πίνακας με |V(G)| στοιχεία τύπου ακέραιος με εξ΄
ορισμού τιμή 0
για κάθε κορυφή v σε topOrder(G) κάνε
   για κάθε ακμή v, w) στο E(G) κάνε
      εάν length_to[w] <= length_to[v] + weight(G,(v,w)) τότε
           length_to[w] = length_to[v] + weight(G,(v,w))

    επέστρεψε max(length_to[v] για v στο V(G))
```

Βάσει λοιπόν του παραπάνω αλγορίθμου, το μεγαλύτερο μονοπάτι μέσα στο γράφο του σχήματος 4.4 το οποίο υποδηλώνει ότι έχει επιτευχθεί χρονοπρογραμματισμός, απεικονίζεται στο σχήμα 4.5 με πράσινο χρώμα.

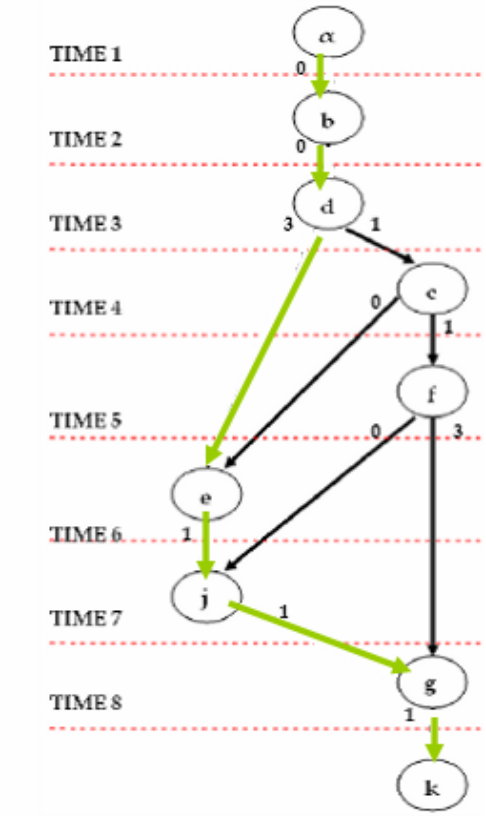


Σχήμα 4.5: *Μονοπάτι μεγαλύτερης διαδρομής (Longest path)*

Συνεχίζοντας με την παρουσίαση παραδειγμάτων χρονοπρογραμματισμού ASAP, ισχύει η ίδια διαδικασία για τον μετασχηματισμένο γράφο του σχήματος 3.4(c) (μετασχηματισμός γράφου ημιανώτερης τάξης). Παρόλο που εδώ οι κόμβοι εμφανίζονται με τη μορφή συμβόλων (τετράγωνα, ρόμβοι, κύκλοι), υποδηλώνοντας τον τύπο του κάθε κόμβου (μνήμη, σημείο συνθήκης, ακέραιος αντίστοιχα), στην ουσία αυτό δεν απασχολεί τον χρονοπρογραμματισμό. Κατά συνέπεια έχουμε τον γράφο του σχήματος 4.6(b):



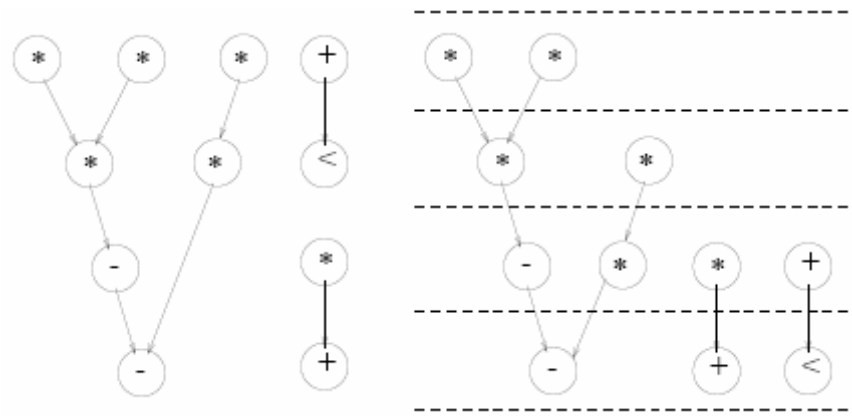
Σχήμα 4.6: *(a) Αρχικός γράφος (b) γράφος διαιρεμένος σε επίπεδα χρόνου*



Σχήμα 4.7: *Μονοπάτι μεγαλύτερης διαδρομής του σχήματος 4.6(b)*

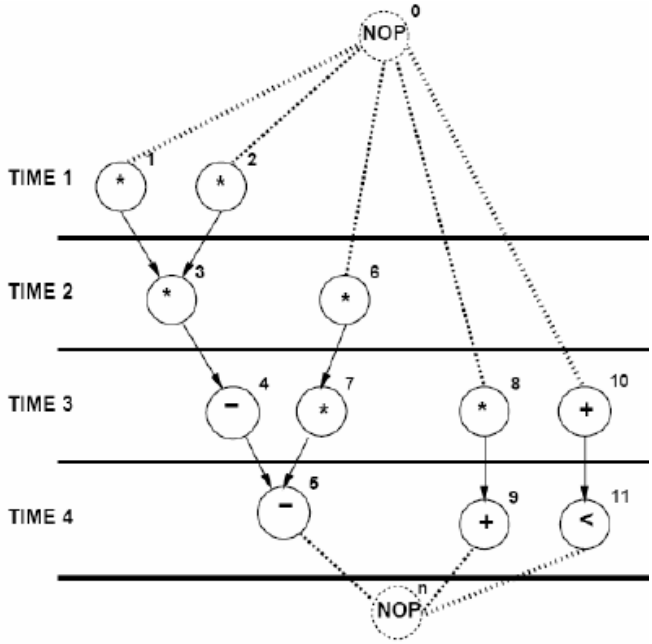
4.2.2 – Παραδείγματα ALAP (As-Late-As-Possible)

Ο αλγόριθμος χρονοπρογραμματισμού ALAP αποτελεί μια πιο εξευγενισμένη προσέγγιση του αλγορίθμου ASAP, με την υπο όρους αναβολή των λειτουργιών. Ο αλγόριθμος ουσιαστικά δουλεύει με τον ίδιο τρόπο όπως ο αλγόριθμος ASAP, με τη διαφορά ότι ξεκινά από το κατώτερο μέρος του γράφου ροής δεδομένων με κατεύθυνση προς τα πάνω (σχήμα 4.8). Ο αλγόριθμος ALAP δίνει το αργότερο δυνατό χρονοπρόγραμμα, το οποίο δέχεται το μέγιστο αριθμό από βήματα ελέγχου. Παρόλα ταύτα, αυτό δε μειώνει απαραίτητα τον αριθμό των λειτουργικών μονάδων που χρησιμοποιούνται.



Σχήμα 4.8: *(α) Γράφος ροής δεδομένων (b) χρονοπρόγραμμα ALAP*

Στο σχήμα 4.9 που ακολουθεί, παρατίθεται ένα ακόμα παράδειγμα χρονοπρογραμματισμού ALAP με χρονική καθυστέρηση ίση με τέσσερα (4).



Σχήμα 4.9: *Παράδειγμα ALAP*

4.3 – Εφαρμογή του χρονοπρογραμματισμού λίστας

Ο αλγόριθμος χρονοπρογραμματισμού λίστας (List Scheduling algorithm), ανήκει στην κατηγορία εκείνη των ευρετικών μεθόδων οι οποίες αποφασίζουν, ποία θα είναι η επόμενη εντολή που θα εκτελεστεί από μια λίστα υποψηφίων εντολών. Ο αλγόριθμος έχει τοπική εφαρμογή, με πρώτη κίνηση την ανάθεση προτεραιοτήτων στις εντολές, στατικά ή δυναμικά (οι εντολές με μεγαλύτερη προτεραιότητα εκτελούνται πρώτες). Στη συνέχεια γίνεται κατασκευή ενός κατευθυνόμενου άκυκλου γράφου (DAG) ώστε να παραχθεί το χρονοπρόγραμμα, λαμβάνοντας πάντα υπόψιν τις εξαρτήσεις των εντολών.

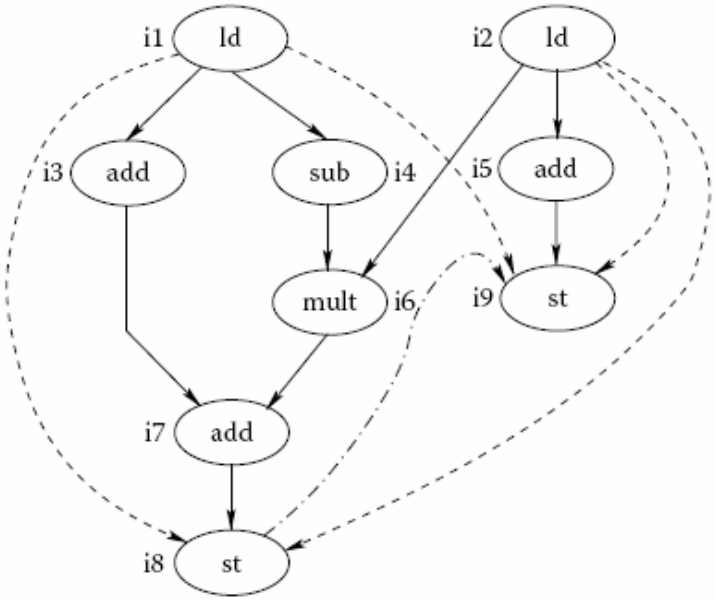
Ο DAG χαρακτηρίζεται από έναν "bottom-up" προσανατολισμό, αντιστοιχίζοντας έναν κόμβο για κάθε εντολή. Οι ακμές με τη σειρά τους εμφανίζονται όταν δυο κόμβοι χρησιμοποιούν την ίδια τιμή. Ο αλγόριθμος ως γνωστόν επαναλαμβάνεται, σταματώντας μόνο όταν εκτελεστούν όλες οι εντολές. Παρακάτω ακολουθεί ένα παράδειγμα όπου δεδομένου ενός κώδικα σε γλώσσα υψηλού επιπέδου, θα γίνει απόπειρα χρονοδρομολόγησης κάθε εντολής που τον απαρτίζει με τη βοήθεια του αλγορίθμου χρονοπρογραμματισμού λίστας.

Έστω ο κώδικας:

$$c = (a+4)+(a-2)*b;$$
$$b = b+3;$$

Μετασχηματίζουμε τον παραπάνω κώδικα σε εντολές γλώσσας χαμηλού επιπέδου (assembly) όπως φαίνεται στη συνέχεια, ενώ στο σχήμα 4.10 απεικονίζεται ο κατευθυνόμενος άκυκλος γράφος για το πρόγραμμα του παραδείγματός μας.

```
I1: LOAD T1,a;
I2: LOAD T2,b;
I3: ADD T3,T1,4;
I4: SUB T4,T1,2;
I5: ADD T5,T2,3;
I6: MUL T6,T4,T2;
I7: ADD T7,T3,T6;
I8: STR c,T7;
I9: STR b,T5;
```



Σχήμα 4.10: *Γράφος προγράμματος Assembly*

Θεωρούμε ότι υπάρχει μια αρχιτεκτονική αποτελούμενη από δυο (2) αριθμητικές λογικές μονάδες (Arithmetic Logical Units – ALUs), έναν πολλαπλασιαστή και εντολές για LOAD και STORE (STR). Το σύστημα λειτουργεί με διοχέτευση και παρουσιάζει καθυστέρηση ένα, δυο, τρία και έναν κύκλο ρολογιού αντίστοιχα. Παρατηρούμε ότι το μονοπάτι συντομότερης διαδρομής είναι το "I1, I4, I6, I7, I8".

Οι εντολές που βρίσκονται στο μονοπάτι συντομότερης διαδρομής (critical path), εκτελούνται πρώτες. Στην περίπτωση που οι δυο προσθέσεις I3 και I5 εκτελούνταν πριν από την αφαίρεση, θα υπήρχε μεγαλύτερη καθυστέρηση στο μονοπάτι συντομότερης διαδρομής και στο συνολικό χρονοπρόγραμμα. Το χρονοπρόγραμμα για το παραπάνω παράδειγμα φαίνεται στη συνέχεια.

```
TIME0: LOAD T1,a; LOAD T2,b;
TIME1:
TIME2: SUB T4,T1,2; ADD T5,T2,3;
TIME3: ADD T3,T1,4; STR b,T5;
TIME4:
TIME5:
TIME6: ADD T7,T3,T6;
TIME7: STR c,T7;
TIME8:
TIME9:
```

4.4 – Άλλα παραδείγματα χρονοπρογραμματισμού

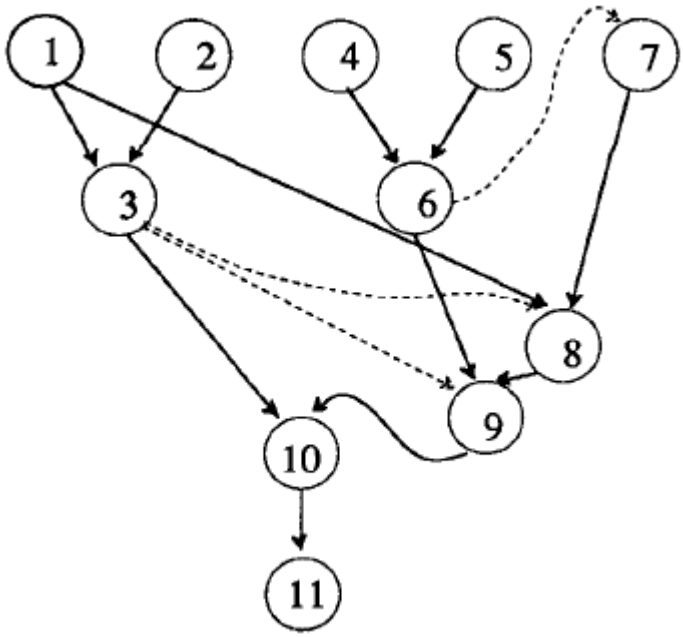
Σε συνέχεια της παράθεσης παραδειγμάτων χρονοπρογραμματισμού, δίνεται παρακάτω ο ακόλουθος κώδικας και ο αντίστοιχος DAG

Έστω ο κώδικας:

$$f = a * b;$$
$$g = (c + d) * (a + e);$$
$$h = f + g;$$

Ο παραπάνω κώδικας μετασχηματιζόμενος σε εντολές γλώσσας χαμηλού επιπέδου αναπαρίσταται ως εξής (ακολουθεί ο γράφος στο σχήμα 4.11):

```
01: LOAD R1,a;
02: LOAD R2,b;
03: MUL R3,R1,R2;
04: LOAD R4,c;
05: LOAD R5,d;
06: ADD R6,R4,R5;
07: LOAD R7,e;
08: ADD R8,R1,R7;
09: MUL R9,R6,R8;
10: ADD R10,R3,R9;
11: STR R10,h;
```

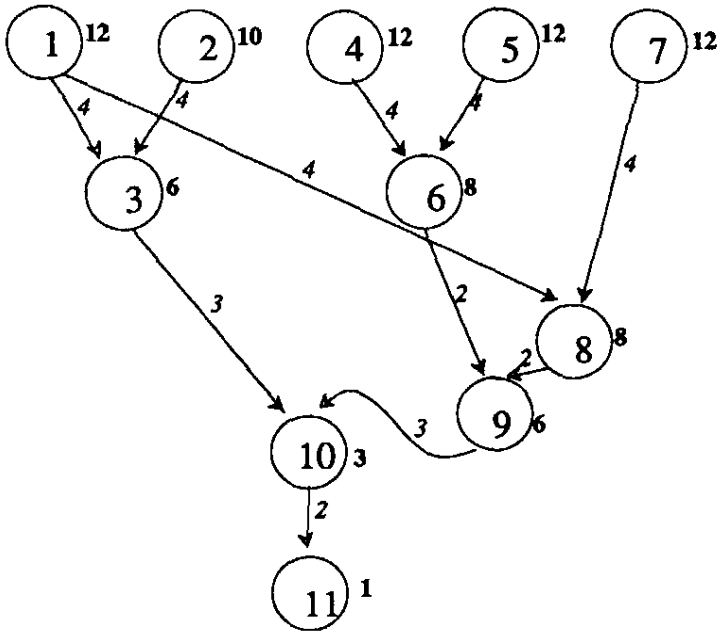


Σχήμα 4.11: *Γράφος προγράμματος*

Ο εκτιμώμενος χρόνος εκτέλεσης καθεμίας από τις παραπάνω εντολές, χρησιμοποιείται για τον υπολογισμό αθροιστικά του «κόστους» κάθε κόμβου στον DAG. Αυτό το αθροιστικό κόστος προσδιορίζει ποιοι κόμβοι ανήκουν στη συντομότερη διαδρομή (critical path) στο γράφο, κατά τη διάρκεια του χρονοπρογραμματισμού εντολών. Οι κόμβοι αυτοί και κατά συνέπεια οι εντολές που τους περιγράφουν, ως γνωστό έχουν υψηλότερη προτεραιότητα εκτέλεσης. Για το πρόγραμμα (κώδικας) που χρησιμοποιούμε στην περίπτωσή μας, υποθέτουμε ότι ισχύουν οι παρακάτω χρόνοι για τις σχετικές λειτουργίες:

Λειτουργίες	Χρόνοι (Clock periods)
LOAD	4
STORE (STR)	1
ADD	2
MULTIPLY (MUL)	3

Υποθέτουμε ότι η αρχική τιμή για τους διαθέσιμους καταχωρητές είναι 4. Η τιμή αυτή προσδιορίζεται προσεγγιστικά, από το σύνολο των καταχωρητών πλην αυτών οι οποίοι δεν έχουν ακόμα χρησιμοποιηθεί. Στη συνέχεια παρατίθεται ο παραπάνω γράφος σε συνδυασμό με τα διάφορα βάρη, τα οποία χαρακτηρίζουν τους κόμβους (αθροιστικό κόστος) και τις ακμές του στο σχήμα 4.12 .



Σχήμα 4.12: *Γράφος προγράμματος με βάρη*

Προφανώς η συντομότερη διαδρομή είναι η 1➔3➔10➔11 όπως φαίνεται από τον γράφο του σχήματος 4.12.

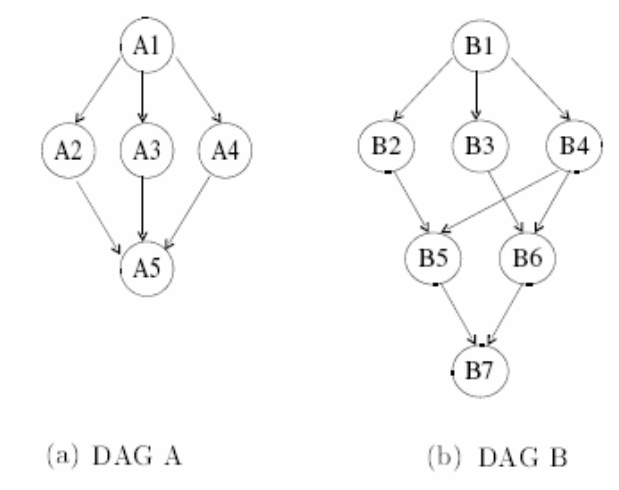
Το προηγούμενο παράδειγμα αποτελεί μια αφορμή, να αναφερθούμε στην κατάσταση που επικρατεί όταν μεγάλα τμήματα κώδικα εκτελούνται σε επεξεργαστές με διοχέτευση. Στις περιπτώσεις αυτές υπάρχει η πιθανότητα εμφάνισης καθυστερήσεων στην εκτέλεση του κώδικα, οπότε και χρησιμοποιούνται τεχνικές για την όσο το δυνατό εξάλειψη του προβλήματος αυτού. Η τεχνική του «χρονοπρογραμματισμού κώδικα» (Code scheduling) είναι μια από τις τεχνικές αυτές, η οποία αναδιατάσσει την ακολουθία του κώδικα στο χρόνο μεταγλώττισης ώστε να μειωθούν οι όποιες καθυστερήσεις.

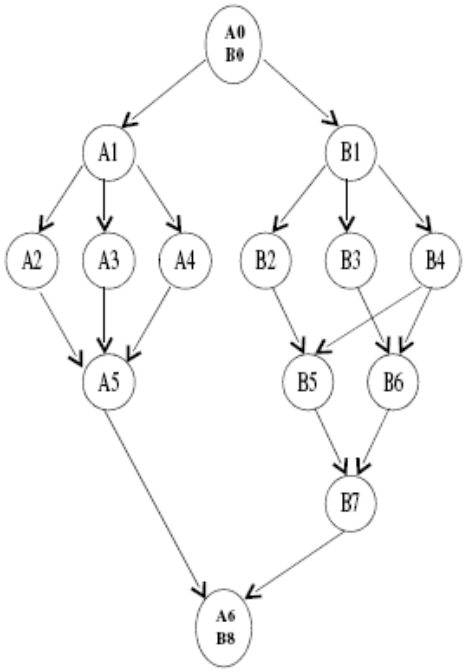
Μια άλλη τεχνική χρονοπρογραμματισμού για τις περιπτώσεις μεγάλων τμημάτων κώδικα είναι η λεγόμενη «κατανομή καταχωρητή» (Register Allocation). Χρησιμοποιείται πολλές φορές σε συνδυασμό με την τεχνική "Code scheduling", και ο ρόλος της έγκειται στην επαναχρησιμοποίηση των καταχωρητών ώστε να μειωθεί ο αριθμός τους. Με τον τρόπο αυτό επιδιώκεται ταχύτερη εκτέλεση του κώδικα, αλλά παράλληλα εισάγονται κάποιοι περιορισμοί που μειώνουν τις δυνατότητες του χρονοπρογραμματισμού κώδικα, γεγονός που δεν αξιολογείται θετικά.

<u>Χρονοπρογραμματισμός πολλαπλών DAG σε ετερογενή συστήματα</u>

Με λίγα λόγια θα μπορούσαμε να αναφερθούμε και σε αυτή την περίπτωση χρονοπρογραμματισμού, η οποία είναι αρκετά καλά μελετημένη με έναν αριθμό από ευρετικές μεθόδους να έχουν προταθεί για τη διεκπεραίωσή του.

Είναι γεγονός ότι όλες οι διεργασίες που σχετίζονται με τον χρονοπρογραμματισμό σε έναν DAG, διαπραγματεύονται με το πρόβλημα του χρονοπρογραμματισμού στον έναν και μοναδικό αυτό γράφο. Λογικό είναι όμως να αντιμετωπίζεται ως πιθανό σενάριο, η κατάσταση όπου περισσότεροι από ένας DAG να πρέπει να χρονοπρογραμματιστούν με διαθέσιμους πόρους την ίδια χρονική στιγμή. Σε αυτή την περίπτωση μια προφανής λύση είναι ο χρονοπρογραμματισμός καθενός DAG σε σειρά (ο ένας μετά τον άλλο), χρησιμοποιώντας αλγόριθμο σαν να υπήρχε ένας μοναδικός γράφος. Κατά συνέπεια λοιπόν οι δυο γράφοι του επόμενου σχήματος DAG A και DAG B συνδυάζονται σε έναν ενιαίο γράφο, όπου και θα εφαρμοστεί ο αλγόριθμος χρονοπρογραμματισμού.





Σχήμα 4.13: *Συνδυαστικός γράφος με τεχνική κοινής εισόδου και εξόδου για τους DAG A και DAG B*

Σύνοψη – Συμπεράσματα

Η έννοια του χρονοπρογραμματισμού κυριάρχησε σε όλη σχεδόν την έκταση της διπλωματικής εργασίας. Η ουσία της εργασίας αυτής, έγκειται στην κατανόηση βασικών μεθόδων για το χρονοπρογραμματισμό των εντολών ενός προγράμματος στην περίπτωση των μεταγλωττιστών, ώστε να προκύψουν βελτιωμένα ως προς την απόδοση προγράμματα, με τη βοήθεια αλγορίθμων. Το σίγουρο είναι πάντως ότι η βέλτιστη απόδοση ενός προγράμματος δεν είναι κάτι το μονοσήμαντο, αλλά εξαρτάται και από την αρχιτεκτονική του επεξεργαστή που χρησιμοποιείται για την εκτέλεσή του. Ειδικότερα η εργασία αναλώθηκε περισσότερο στους επεξεργαστές με διοχέτευση.

Σημείο αναφοράς για την εργασία ήταν οι γράφοι προγραμμάτων, οι οποίοι αποτελούν και τον κορμό στήριξης της διεργασίας του χρονοπρογραμματισμού μετά από εκτέλεση μιας σειράς μετασχηματισμών σε αυτούς. Παρατηρούμε όμως ότι οι μετασχηματισμοί δεν δίνουν πάντα το ίδιο αποτέλεσμα, αλλά προσδίδουν διαφορετικό νόημα στον κώδικα που θα προκύψει αναλόγως με την πληροφορία που μεταφέρεται από τις ακμές και τους κόμβους του. Δεν υπάρχει αμφιβολία όμως, ότι κάθε τέτοιος μετασχηματισμός είναι βέλτιστος.

Σε γενικές γραμμές για την επίτευξη της χρονοδρομολόγησης προτάθηκαν αλγόριθμοι, οι οποίοι θέτουν τα χρονικά όρια για να πραγματοποιηθεί ο χρονοπρογραμματισμός (π.χ ASAP, ALAP κ.λ.π.), όπως παρατέθηκαν στο τελευταίο κεφάλαιο με παραδείγματα. Ο αλγόριθμος ASAP φαίνεται να είναι η πιο βέλτιστη λύση στο πρόβλημα του χρονοπρογραμματισμού από πλευράς ταχύτητας εκτέλεσης, μη λαμβάνοντας υπόψιν περιορισμούς των διαθέσιμων πόρων.

Βιβλιογραφικές αναφορές

[1] Βασιλείου Μ. Χρυσάνθη, "*Χρονοδρομολόγηση μεταφορών δεδομένων και εντολών, για μεγαλύτερη ταχύτητα εφαρμογών σάρωσης πινάκων*", Διπλωματική εργασία – Πανεπιστήμιο Πατρών – Τμήμα Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Η/Υ, 2010.

[2] J. L. Henessy και D.A. Patterson, "*Computer Architecture: A Quantitive Approac*h", 4η έκδοση, Morgan Kaufmann, 2007.

[3] Σπουρλής Λ. Γεώργιος, "*Τεχνικές μεταγλωττιστών για εφαρμογές πολυμέσων*", Διπλωματική εργασία – Πανεπιστήμιο Πατρών – Τμήμα Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Η/Υ, 2009.

[4] Γκίκα Δ. Ζαχαρούλα, "*Τεχνικές μεταγλωττιστών για βελτιστοποίηση ταχύτητας ενσωματωμένων υπολογιστών*", Διπλωματική εργασία – Πανεπιστήμιο Πατρών – Τμήμα Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Η/Υ, 2010.

[5] Καββαδίας Κ. Νικόλαος, "*Προηγμένα θέματα θεωρητικής πληροφορικής*", Διαλέξεις μαθήματος – Πρόγραμμα μεταπτυχιακών σπουδών (Κατεύθυνση Θεωρητικής Πληροφορικής) – Πανεπιστήμιο Πελοποννήσου – Τμήμα Επιστήμης & Τεχνολογίας Η/Υ, 2010.

[6] Βικιπαίδεια, "*Θεωρία γράφων*", Γράφος, Τυπολογικά παραδείγματα – http://el.wikipedia.org/wiki/Θεωρία_γράφων .

[7] Σπινέλλης Διομήδης, "*Γράφοι*", Οικονομικό Πανεπιστήμιο Αθηνών – http://dmst.aueb.gr/dds/ads/graph/indexw.htm .

[8] Βικιπαίδεια, "*Θεωρία γράφων*", Δέντρα (Trees), – http://en.wikipedia.org/wiki/Tree_(graph_theory) .

[9] Βικιπαίδεια, "*Θεωρία γράφων*", "*Shortest Path Problem*" – http://en.wikipedia.org/wiki/Shortest_path_problem .

[10] Βικιπαίδεια, "*Θεωρία γράφων*", "*Longest Path Problem*" – http://en.wikipedia.org/wiki/Longest_path_problem .

[11] Βικιπαίδεια, "*Θεωρία γράφων*", "*Directed Acyclic Graph*" – http://en.wikipedia.org/wiki/Directed_acyclic_graph .

[12] Z. Zeng, A. Tung, J. Wang, J. Feng and L. Zhou, *"Comparing stars: On approximating graph edit distance"*, *Research work at University of Singapore,* VLDB Endowment, ACM 2009.

[13] Βικιπαίδεια, *"Θεωρία γράφων"*, *"Graph Rewriting System"* – http://en.wikipedia.org/wiki/Graph_rewriting .

[14] Muchnick S., *"Advanced Compiler Design and Implementation"*, Morgan Kaufmann, 1997.

[15] Mark Heffernan and Kent Wilken, "*Data-Dependency Graph Transformations for Instruction Scheduling*", Journal of Scheduling – University of California, Davis CA95616 – Department of Electrical and Computer Engineering, 8: 427-451, 2005.

[16] Βικιπαίδεια, *"Θεωρία γράφων"*, *"Τοπολογική ταξινόμηση"* – http://el.wikipedia.org/wiki/Τοπολογική_ταξινόμηση .

[17] Ντάρμος Νικόλαος, "*Σχεδίαση και ανάλυση αλγορίθμων*", Σημειώσεις Εργαστηρίου – Πανεπιστήμιο Ιωαννίνων – Τμήμα Πληροφορικής, 2011.

[18] Μιχαήλ Ε. Χαράλαμπος, "Βελτιστοποίηση επαναπροσδιοριζόμενων αρχιτεκτονικών για απόδοση και κατανάλωση ενέργειας σε κρυπτογραφικές εφαρμογές και εφαρμογές κυριαρχούμενες από δεδομένα", Διδακτορική διατριβή – Πανεπιστήμιο Πατρών – Τμήμα Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Η/Υ, 1: 31-33, 2009.

[19] E.B. Fernandez και T. Lang, "*Scheduling as a Graph Transformation*", IBM Journal of Research and Development, 20: 551-559, 1976.

[20] A. B. Barskiy, *"Minimizing the Number of Computing Devices Needed to Realize a Computational Process Within a Specified Time"*, Eng. Cybernetics (USSR) 6, 59 (1968).

[21] H. O. Levy, "Application of Graph Transformations to Scheduling", M. S. Thesis, University of California, Los Angeles, 1973.

[22] Hennesy J. , Patterson D. , *"Computer Arcitecture: A Quantitive Approach"*, 3rd ed., Morgan Kaufmann, 2002.