Τεχνολογικο Εκπαιδευτικο Ιδρυμα Πελοποννησου
Σχολη Τεχνολογικων Εφαρμογων
Τμημα Μηχανικων Πληροφορικης Τ.Ε.

Υλοποίηση του αλγορίθμου Dijkstra με Fibonacci Σωρό

Πτυχιακη Εργασια

του

Γεωργίου Ελευθερίου Βενιέρη

Επιβλέπων: Γρηγόρης Καραγιώργος
Αναπληρωτής Καθηγητής

Σπάρτη, Ιανουάριος 2017

Τεχνολογικο Εκπαιδευτικο Ιδρυμα Πελοποννησου
Σχολη Τεχνολογικων Εφαρμογων
Τμημα Μηχανικων Πληροφορικης Τ.Ε.

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

Όνομα και Επώνυμο Συγγραφέα (Με Κεφαλαία):

ΓΕΩΡΓΙΟΣ ΕΛΕΥΘΕΡΙΟΣ ΒΕΝΙΕΡΗΣ

Υπογραφή (Ολογράφως, χωρίς μονογραφή):

. . . .

Ημερομηνία (Ημέρα – Μήνας – Έτος):

19 – 1 – 2017

Περίληψη

Σκοπός της παρούσας πτυχιακής εργασίας είναι να υλοποιηθεί στη γλώσσα προγραμματισμού C, ο αλγόριθμος του Dijkstra για τον υπολογισμό των συντομότερων διαδρομών σε ένα γράφο καθώς και η δομή δεδομένων που θα χρησιμοποιεί ως ουρά προτεραιότητας, ο σωρός Fibonacci. Επίσης θα γίνει ανάλυση της πολυπλοκότητας του αλγορίθμου αυτού και θα συγκριθεί με αυτή άλλων υλοποιήσεων και ειδικότερα με την περίπτωση που χρησιμοποιείται μια άλλη δομή δεδομένων ως ουρά προτεραιότητας. Αρχικά θα επεξηγηθούν βασικές έννοιες που θα είναι απαραίτητες μετέπειτα. Από εκεί και πέρα θα γίνει επικέντρωση στις κύριες δομές δεδομένων που θα χρειαστούν και στον αλγόριθμο του Dijkstra. Ο αλγόριθμος καθώς και ο σωρός Fibonacci και οι πράξεις του θα συνοδεύονται από σχήματα και ψευδοκώδικα για την καλύτερη κατανόησή τους. Τέλος, θα υλοποιηθούν στη γλώσσα προγραμματισμού C.

Abstract

The purpose of this thesis is to implement in the programming language C, Dijkstra's algorithm for the computation of the shortest paths in a graph as well as the data structure that will be used as priority queue, the Fibonacci heap. Also the complexity of the algorithm will be analyzed and compared to that of other implementations and particularly in the case that another data structure is used as priority queue. Initially basic concepts that will be necessary later on will be explained. From then on the focus will be set on the main data structures that will be needed and Dijkstra's algorithm. The algorithm as well as the Fibonacci heap and it's operations will be accompanied by figures and pseudocode for better comprehension. Finally, they will be implemented in the programming language C.

Περιεχόμενα

1 Εισαγωγή **1**

1.1 Αλγόριθμοι και πολυπλοκότητα . . . 1

1.2 Δομές δεδομένων . . . 2

1.2.1 Πίνακες . . . 2

1.2.2 Συνδεδεμένες λίστες . . . 3

1.2.3 Σωρός . . . 4

1.3 Αφηρημένος τύπος δεδομένων . . . 6

1.3.1 Ουρά προτεραιότητας . . . 6

2 Γράφος **7**

2.1 Βασικοί τύποι γράφων και ορισμοί . . . 7

2.2 Αναπαραστάσεις . . . 9

3 Σωρός Fibonacci **11**

3.1 Δομή των σωρών Fibonacci . . . 11

3.2 Πράξεις των σωρών Fibonacci . . . 13

3.2.1 Δημιουργία . . . 13

3.2.2 Εισαγωγή . . . 14

3.2.3 Συγχώνευση . . . 15

3.2.4 Ελάχιστο . . . 17

3.2.5 Εξαγωγή ελαχίστου . . . 18

3.2.6 Μείωση κλειδιού . . . 23

3.2.7 Διαγραφή . . . 26

3.2.8 Κατασκευή . . . 27

3.3 Απόδειξη του μέγιστου βαθμού . . . 28

4 Ο Αλγόριθμος του Dijkstra **29**

4.1 Περιγραφή . . . 29

4.2 Λειτουργία . . . 30

4.3 Ανάλυση πολυπλοκότητας . . . 33

4.3.1 Υλοποίηση με σωρό Fibonacci . . . 33

4.3.2 Υλοποίηση με πίνακα και με λίστα . . . 33

4.3.3 Υλοποίηση με δυαδικό σωρό ελαχίστου . . . 34

5 Υλοποίηση στη γλώσσα C 35
5.1 Γράφος 35
5.1.1 Πίνακας γειτνίασης 35
5.1.2 Λίστες γειτνίασης 38
5.2 Σωρός Fibonacci 42
5.3 Αλγόριθμος Dijkstra 51

6 Συμπεράσματα 57

Βιβλιογραφία 58

Κατάλογος Σχημάτων

1.2.1.1 Δομές πινάκων . . . 3
1.2.2.1 Δομές συνδεδεμένων λιστών . . . 4
1.2.3.1 Δομή δυαδικού σωρού . . . 5

2.1.1 Βασικοί τύποι γράφων . . . 8
2.2.1 Αναπαραστάσεις γράφων . . . 10

3.1.1 Δομή ενός σωρού Fibonacci . . . 12
3.2.2.1 Εισαγωγή σε σωρό Fibonacci . . . 15
3.2.3.1 Συγχώνευση σωρών Fibonacci . . . 17
3.2.5.1 Εξαγωγή ελαχίστου από σωρό Fibonacci . . . 22
3.2.6.1 Μείωση κλειδιού σε σωρό Fibonacci . . . 25

4.2.1 Εκτέλεση του αλγορίθμου του Dijkstra . . . 32

1 Εισαγωγή

1.1 Αλγόριθμοι και πολυπλοκότητα

Υπάρχουν πολλοί ορισμοί για την έννοια του αλγορίθμου. Αλγόριθμος είναι μια καλά ορισμένη υπολογιστική διαδικασία που δέχεται με κάποιο τρόπο δεδομένα ως είσοδο και ακολουθώντας μια σειρά βημάτων, επεξεργάζεται τα δεδομένα αυτά και παράγει ένα αποτέλεσμα το οποίο αποτελεί την έξοδο [1].

Ένας αλγόριθμος ορίζεται από τα υπολογιστικά βήματα που ακολουθεί για την επίλυση ενός προβλήματος και μετατρέπει την είσοδο που του δόθηκε σε έξοδο [1]. Οι αλγόριθμοι χρησιμοποιούνται ως εργαλεία για την επίλυση υπολογιστικών προβλημάτων [1]. Ένας αλγόριθμος είναι ορθός αν για κάθε είσοδο τερματίζει και δίνει τη σωστή έξοδο [1].

Πολυπλοκότητα ή ανάλυση ενός αλγορίθμου είναι η εκτίμηση της επίδοσής του [7]. Η μονάδα μέτρησης της επίδοσης είναι συνήθως ο χρόνος που απαιτείται για την εκτέλεσή του σε σχέση με το μέγεθος της εισόδου [7]. Το μέγεθος ενός προβλήματος εκφράζεται ως το πλήθος των στοιχείων εισόδου το οποίο γενικεύεται και αναπαριστάνεται με τη μεταβλητή n [7]. Για να υπολογιστεί η πολυπλοκότητα ενός αλγορίθμου πρέπει να βρεθεί μια συνάρτηση του n (έστω $g(n)$) που να υπολογίζει τον αριθμό των βασικών πράξεων που εκτελεί ο αλγόριθμος [7]. Έπειτα προσδιορίζεται μια συνάρτηση $f(n)$ που αποτελεί το φράγμα της $g(n)$ [7]. Με τον τρόπο αυτό περιορίζονται οι λεπτομέρειες της $g(n)$ και εξετάζεται συνήθως ο μεγαλύτερος όρος της [7].

Ο συμβολισμός O χρησιμοποιείται για τον ορισμό ενός άνω φράγματος μιας συνάρτησης $g(n)$ και ορίζεται ως εξής: [1, 7]

$O(g(n)) = f(n)$ αν υπάρχουν δύο θετικές σταθερές c_0 και n_0 τέτοιες ώστε για κάθε $n > n_0$ να ισχύει ότι $0 \leq f(n) \leq cg(n)$. Όταν φράσσεται μέσω του O ο χρόνος εκτέλεσης ενός αλγορίθμου στη χειρότερη περίπτωση εισόδου, δηλαδή στην είσοδο για την οποία ο αλγόριθμος εκτελεί τον μέγιστο αριθμό πράξεων, φράσσεται και ο χρόνος εκτέλεσης για οποιαδήποτε είσοδο.

Όπως ο συμβολισμός O ορίζει ένα άνω φράγμα, ο συμβολισμός Ω ορίζει ένα κάτω φράγμα: [1, 7]

$\Omega(g(n)) = f(n)$ αν υπάρχουν δύο θετικές σταθερές c_0 και n_0 τέτοιες ώστε για κάθε $n > n_0$ να ισχύει ότι $0 \leq cg(n) \leq f(n)$. Όταν φράσσεται μέσω του Ω ο χρόνος εκτέλεσης ενός αλγορίθμου στην καλύτερη περίπτωση εισόδου, δηλαδή σε εκείνη που

θα εκτελεστεί ο ελάχιστος αριθμός πράξεων, φράσσεται επίσης και για οποιαδήποτε είσοδο.

Ο συμβολισμός Θ ορίζει το φραγμό της συνάρτησης από πάνω και από κάτω: [1, 7]

$\Theta(g(n)) = f(n)$ αν υπάρχουν τρεις θετικές σταθερές c_1, c_2 και n_0 τέτοιες ώστε για κάθε $n > n_0$ να ισχύει ότι $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$. Για οποιοδήποτε ζεύγος $f(n)$ και $g(n)$, για να ισχύει ότι $f(n) = \Theta(n)$ θα πρέπει επίσης να ισχύει ότι $f(n) = O(g(n))$ και $f(n) = \Omega(g(n))$.

Η αντισταθμιστική ανάλυση είναι μια μέθοδος που χρησιμοποιείται για ανάλυση πολυπλοκότητας υπολογίζοντας το μέσο χρόνο εκτέλεσης ενός αλγορίθμου και συνήθως χρησιμοποιείται για την ανάλυση των πράξεων δομών δεδομένων [1]. Σε πολλές περιπτώσεις, κάποιες πράξεις χρειάζονται αρκετό χρόνο ενώ κάποιες άλλες όχι οπότε στην αντισταθμιστική ανάλυση λαμβάνονται υπόψη και οι δύο παράλληλα για μια ακολουθία πράξεων και μπορεί να δειχθεί ότι παρόλο που στη χειρότερη περίπτωση το κόστος σε χρόνο μπορεί να είναι μεγάλο, στη μέση περίπτωση είναι μικρό [1].

1.2 Δομές δεδομένων

Καθώς τα δεδομένα μπορεί να είναι σε οποιαδήποτε μορφή και σκοπός είναι η να πραγματοποιούνται λειτουργίες σε αυτά, θα πρέπει η διαδικασία οργάνωσής τους να είναι αποτελεσματική ώστε η επεξεργασία τους να γίνεται με τον πιο αποδοτικό τρόπο.

Μια δομή δεδομένων είναι ο τρόπος με τον οποίο γίνεται η οργάνωση και η αποθήκευση των δεδομένων για την αποτελεσματικότερη επεξεργασία τους [1].

1.2.1 Πίνακες

Ο πίνακας είναι μια διαδοχική συλλογή στοιχείων γραμμικής διάταξης όπου το καθένα από αυτά είναι προσβάσιμο χρησιμοποιώντας το όνομα του πίνακα και μέσα σε τετραγωνικές αγκύλες τον αύξοντα αριθμό που συμβολίζει τη θέση του στοιχείου μέσα σε αυτόν [1].

Η αρίθμηση των θέσεων ενός πίνακα σε πολλές γλώσσες προγραμματισμού ξεκινάει συνήθως από την τιμή 0 ή την τιμή 1. Οι πίνακες στην παρούσα πτυχιακή θα ξεκινάνε από τη θέση 0.

Έστω n θέσεων πίνακας A, ο τρόπος με τον οποίο θα αποκτάται η πρόσβαση στο i-οστό στοιχείο είναι $A[i]$ [1]. Στο σχήμα 1.2.1.1 (α′) απεικονίζεται ένας απλός πίνακας 5 θέσεων, στα κελιά του περιέχονται τα στοιχεία που έχουν αποθηκευτεί και πάνω από αυτά οι αριθμοδείκτες τους στον πίνακα.

Οι πίνακες μπορούν επίσης να έχουν παραπάνω από μια διάσταση, σε αυτή την περίπτωση θα ορίζεται ο αριθμός των θέσεων της κάθε διάστασης και η πρόσβαση στα στοιχεία του θα αποκτάται με παρόμοιο τρόπο. Έστω δισδιάστατος $m \times n$ θέσεων πίνα-

	0	1	2	3	4
A→	2	5	7	9	1

(α′) Δομή ενός μονοδιάστατου πίνακα

A→	*0*	*1*	*2*	*3*
0	6	1	7	5
1	4	8	1	2
2	5	2	3	1

(β′) Δομή ενός δισδιάστατου πίνακα

Σχήμα 1.2.1.1: Δομές πινάκων

κας A όπου m είναι οι γραμμές του και n οι στήλες του. Η πρόσβαση στο στοιχείο της θέσης i, j θα γράφεται ως $A[i, j]$. Στο σχήμα 1.2.1.1 (β′) απεικονίζεται ένας πίνακας δύο διαστάσεων με 3 γραμμές και 4 στήλες, σε αυτή την περίπτωση χρησιμοποιούνται δύο αριθμοδείκτες για κάθε στοιχείο, ένας για κάθε διάσταση.

Η δημιουργία ενός πίνακα γίνεται σε χρόνο $O(1)$, η προσπέλαση κάποιου στοιχείου γίνεται επίσης σε χρόνο $O(1)$. Αν ο πίνακας είναι δυναμικός, δηλαδή το μέγεθός του μπορεί να αλλάζει τότε η εισαγωγή ή η διαγραφή ενός στοιχείου στο τέλος χρειάζεται λογιστικό χρόνο $O(1)$, στην αρχή και στα ενδιάμεσα $O(n)$ [1].

1.2.2 Συνδεδεμένες λίστες

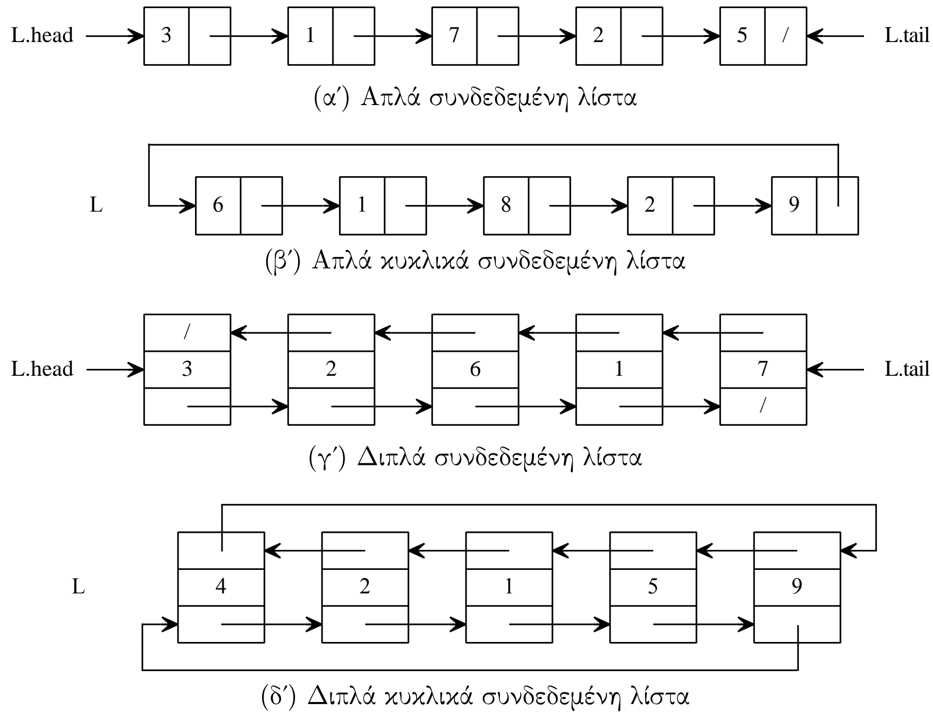
Οι συνδεδεμένες λίστες είναι μια επίσης γραμμικής διάταξης συλλογή στοιχείων που ονομάζονται κόμβοι και κάθε ένας από αυτούς αποθηκεύει πέρα από το στοιχείο του, δείκτες προς άλλους κόμβους [1]. Το πρώτο στοιχείο μιας λίστας L ονομάζεται κεφαλή (head) και το τελευταίο ουρά (tail) [1].

Σε απλά συνδεδεμένες λίστες, κάθε κόμβος αποθηκεύει ένα δείκτη προς τον επόμενο κόμβο από αυτόν και αρκεί να αποθηκευτεί η κεφαλή για να υπάρχει πρόσβαση σε ολόκληρη τη λίστα [1]. Ο δείκτης της ουράς προς τον επόμενο κόμβο έχει την τιμή ΚΕΝΟ [1]. Για πρόσβαση σε ολόκληρη τη λίστα αρκεί να αποθηκευτεί η κεφαλή της.

Σε απλά κυκλικά συνδεδεμένες λίστες, η μοναδική διαφορά είναι ότι ο δείκτης του επόμενου κόμβου της ουράς αντί να δείχνει στο ΚΕΝΟ, δείχνει στην κεφαλή [1]. Αρκεί να αποθηκευτεί οποιοσδήποτε κόμβος για να υπάρχει πρόσβαση σε ολόκληρη τη λίστα.

Σε διπλά συνδεδεμένες λίστες, κάθε στοιχείο αποθηκεύει όχι μόνο ένα δείκτη προς τον επόμενο κόμβο αλλά και έναν προς τον προηγούμενο [1]. Η πρόσβαση στη λίστα αποκτάται όπως και σε απλά συνδεδεμένες λίστες, μπορεί όμως να προσπελαστεί προς δύο κατευθύνσεις και λόγω αυτής της δυνατότητας ο χειρισμός τους είναι συχνά πιο απλός. Ο δείκτης της ουράς προς τον επόμενο κόμβο δείχνει στο ΚΕΝΟ και ο δείκτης της κεφαλής προς τον προηγούμενο κόμβο δείχνει επίσης στο ΚΕΝΟ [1].

Σε διπλά κυκλικά συνδεδεμένες λίστες, η πρόσβαση αποκτάται όπως και στις απλά κυκλικά συνδεδεμένες λίστες. Οι δείκτες που αποθηκεύονται είναι ίδιοι με αυτούς των διπλά συνδεδεμένων λιστών με τη μόνη διαφορά ότι ο δείκτης της ουράς προς τον επόμενο κόμβο δείχνει στην κεφαλή και ο δείκτης της κεφαλής προς τον προηγούμενο



(αʹ) Απλά συνδεδεμένη λίστα

(βʹ) Απλά κυκλικά συνδεδεμένη λίστα

(γʹ) Διπλά συνδεδεμένη λίστα

(δʹ) Διπλά κυκλικά συνδεδεμένη λίστα

Σχήμα 1.2.2.1: Δομές συνδεδεμένων λιστών

κόμβο δείχνει στην ουρά [1].

Για κάθε δομή λίστας, η δημιουργία, η εισαγωγή και η διαγραφή ενός στοιχείου χρειάζεται χρόνο $O(1)$ και η αναζήτηση ενός στοιχείου $O(n)$ [1]. Για την εισαγωγή και τη διαγραφή ενός στοιχείου στα ενδιάμεσα και το τέλος, αν δεν υπάρχει δείκτης στον κόμβο έπειτα από τον οποίο θα εισαχθεί ο νέος ή στον κατάλληλο κόμβο για την πραγματοποίηση της διαγραφής, θα πρέπει να προηγηθεί αναζήτηση οπότε ο χρόνος εκτέλεσης στη χειρότερη περίπτωση είναι $O(n)$ [1]. Διαφορετικά αν οι κατάλληλοι δείκτες είναι διαθέσιμοι ο χρόνος θα είναι $O(1)$.

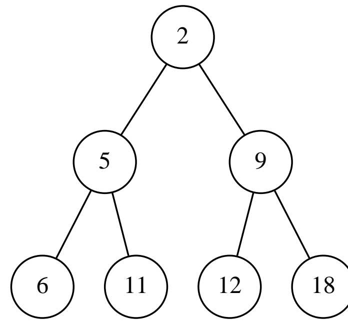
1.2.3 Σωρός

Ο σωρός είναι μια δενδρικής μορφής δομή δεδομένων στην οποία ικανοποιείται η ιδιότητα του σωρού και διακρίνεται σε δύο τύπους, το σωρό ελαχίστου και το σωρό μεγίστου [1].

Η ιδιότητα σωρού ελαχίστου είναι ότι κάθε κόμβος στο σωρό έχει τιμή μικρότερη ή ίση από αυτή των παιδιών του και ο ελάχιστος κόμβος βρίσκεται στη ρίζα [1]. Αντίθετα για την ιδιότητα σωρού μεγίστου κάθε κόμβος έχει τιμή μεγαλύτερη ή ίση από αυτή των παιδιών του και στη ρίζα βρίσκεται ο κόμβος με τη μέγιστη τιμή [1].

Η τιμή ενός κόμβου ονομάζεται κλειδί και ένας συνηθισμένος τρόπος υλοποίησης της δομής αυτής είναι ο δυαδικός σωρός.

Η μορφή ενός δυαδικού σωρού είναι αυτή που έχει ένα πλήρες δυαδικό δένδρο, δηλαδή



(α′)

	0	1	2	3	4
A→	2	5	7	9	1

(β′)

Σχήμα 1.2.3.1: Δομή δυαδικού σωρού

όλα τα επίπεδα του δένδρου εκτός ίσως του τελευταίου είναι συμπληρωμένα [1]. Αν το τελευταίο επίπεδο δεν είναι εντελώς συμπληρωμένο τότε θα είναι συμπληρωμένο μέχρι κάποιο σημείο από τα αριστερά προς τα δεξιά [1].

Οι κόμβοι δε χρειάζεται να αποθηκεύουν δείκτες προς άλλους κόμβους καθώς ο δυαδικός σωρός υλοποιείται με τη χρήση ενός πίνακα [1]. Αν η αρίθμηση του πίνακα ξεκινάει από τη θέση 1 τότε η ρίζα του σωρού θα βρίσκεται στη θέση 1 και για κάθε κόμβο i η θέση του γονέα του θα υπολογίζεται με την πράξη $\lfloor \iota/2 \rfloor$, του αριστερού του παιδιού με την $2i$ και του δεξιού του παιδιού με την $2\iota + 1$ [1]. Αν ο πίνακας ξεκινάει από τη θέση 0 τότε η ρίζα θα είναι στη θέση αυτή, ο γονέας στη θέση $\lfloor (i-1)/2 \rfloor$ και το αριστερό και δεξί παιδί αντίστοιχα στις $2i + 1$ και $2i + 2$.

Στο σχήμα 1.2.3.1 απεικονίζεται ένας δυαδικός σωρός ελαχίστου. Στο (α′) ως δυαδικό δένδρο και στο (β′) ως πίνακας.

Το ύψος ενός κόμβου σε ένα δυαδικού σωρό ορίζεται ως το πλήθος των ακμών της μεγαλύτερη απλής καθοδικής διαδρομής από αυτόν μέχρι κάποιον από τους καταληκτικούς κόμβους [1]. Το ύψος του ίδιου του σωρού είναι το ύψος της ρίζας του [1]. Για παράδειγμα στο σχήμα 1.2.3.1 το ύψος του κόμβου με την τιμή 5 είναι 1 ενώ το ύψος του σωρού είναι 2.

Καθώς ένας δυαδικός σωρός n στοιχείων βασίζεται σε ένα πλήρες δυαδικό δένδρο, το ύψος του είναι $lg(n)$ και για το λόγο αυτό πολλές από τις βασικές του πράξεις χρειάζονται χρόνο $O(lg(n))$ [1].

Ένας δυαδικός σωρός υποστηρίζει όλες τις πράξεις του αφηρημένου τύπου δεδομένων ουρά προτεραιότητας σε χρόνο $O(lg(n))$ ενώ για να κατασκευαστεί από έναν πίνακα χρειάζεται χρόνος $O(n)$ [1].

1.3 Αφηρημένος τύπος δεδομένων

Σε έναν αφηρημένο τύπο δεδομένων, ένας τύπος δεδομένων ορίζεται από τις πράξεις που μπορούν να πραγματοποιηθούν σε αυτόν και τη συμπεριφορά τους και όχι οι λεπτομέρειες υλοποίησής τους, δηλαδή το πως οργανώνονται τα δεδομένα ή το πως πραγματοποιούνται οι πράξεις σε αυτά [6]. Συνεπώς ένας αφηρημένος τύπος δεδομένων ορίζεται από την οπτική του χρήστη και όχι του υλοποιητή. Οι αφηρημένοι τύποι δεδομένων υλοποιούνται χρησιμοποιώντας δομές δεδομένων και τις πράξεις τους και κάθε ένας από αυτούς μπορεί να έχει πολλούς τρόπους υλοποίησης [7]. Μια δομή δεδομένων μπορεί να υλοποιήσει έναν αφηρημένο τύπο δεδομένων αν μπορεί να υποστηρίξει τις πράξεις που ορίζονται από αυτόν.

1.3.1 Ουρά προτεραιότητας

Ο αφηρημένος τύπος δεδομένων ουρά προτεραιότητας ορίζεται ως μια συλλογή στοιχείων όπου το κάθε ένα από αυτά έχει μία προτεραιότητα που ονομάζεται κλειδί [7]. Οι βασικότερες πράξεις που υποστηρίζονται είναι η εισαγωγή και εξαγωγή ενός στοιχείου όμως στην εξαγωγή θα εξάγεται το στοιχείο που έχει τη μεγαλύτερη προτεραιότητα στην ουρά [7].

Η ουρά προτεραιότητας μπορεί να είναι είτε μεγίστου είτε ελαχίστου [1]. Στην πρώτη περίπτωση ένα στοιχείο έχει μεγαλύτερη προτεραιότητα από ένα άλλο αν η τιμή του κλειδιού του είναι μεγαλύτερη ενώ στη δεύτερη αν είναι μικρότερη [1].

Συχνά υποστηρίζονται και περισσότερες πράξεις. Σε μια ουρά προτεραιότητας ελαχίστου αυτές είναι η εύρεση ελαχίστου που επιστρέφει το στοιχείο με τη μικρότερη τιμή κλειδιού και η μείωση κλειδιού που αντικαθιστά το κλειδί του στοιχείου που δόθηκε με ένα μικρότερο ή ίσο [1]. Αντίστοιχα σε μια ουρά προτεραιότητας μεγίστου υποστηρίζεται η εύρεση μεγίστου και η αύξηση κλειδιού [1].

2 Γράφος

2.1 Βασικοί τύποι γράφων και ορισμοί

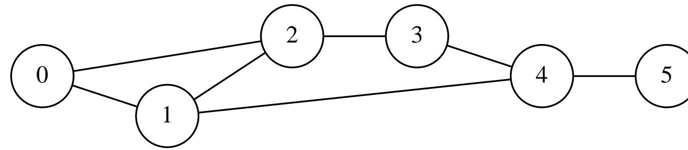
Ένας γράφος G ορίζεται ως ένα ζεύγος αντικειμένων (V, E) το οποίο αποτελείται από ένα σύνολο κορυφών V (vertices) μαζί με ένα σύνολο ακμών E (edges) όπου η καθεμία από αυτές είναι υποσύνολο δύο στοιχείων του συνόλου των κορυφών V, δηλαδή κάθε ακμή συσχετίζεται με δύο κορυφές [7]. Ο συσχετισμός παίρνει τη μορφή μη διατεταγμένων ζευγών (u, v) όπου το u και το v ανήκουν στο σύνολο των κορυφών V [7].

Οι κορυφές ενός γράφου ονομάζονται και σημεία ή κόμβοι [7]. Ο συμβολισμός που θα χρησιμοποιείται για αναφορά στο πλήθος των κορυφών ενός γράφου είναι $|V|$ και για το πλήθος των ακμών $|E|$ [1].

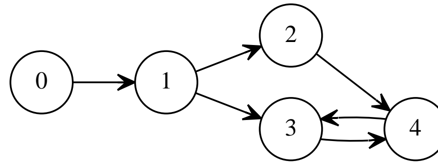
Ένας μη κατευθυνόμενος γράφος είναι ένας γράφος στον οποίο οι ακμές δεν έχουν προσανατολισμό [7]. Σε αυτήν την περίπτωση ένα ζεύγος ακμών (u, v) είναι όμοιο με το ζεύγος (v, u), αυτό σημαίνει ότι επιτρέπεται η μετάβαση και από την κορυφή u προς την v αλλά και από την v προς την u [7]. Στο Σχήμα 2.1.1 (α΄) απεικονίζεται ένας μη κατευθυνόμενος γράφος με 6 κορυφές και 7 ακμές. Επιτρέπονται οι απευθείας μεταβάσεις μόνο στις κορυφές που ενώνονται μεταξύ τους με μια γραμμή η οποία συμβολίζει την ακμή.

Ένας κατευθυνόμενος γράφος είναι ένας γράφος στον οποίο οι ακμές έχουν προσανατολισμό [7]. Σε αυτήν την περίπτωση ένα ζεύγος ακμών (u, v) δεν είναι όμοιο με το ζεύγος (v, u), οπότε αν δεν υπάρχει και το δεύτερο ζεύγος τότε επιτρέπεται μόνο η μετάβαση από την κορυφή u προς την v. Στο Σχήμα 2.1.1 (β΄) απεικονίζεται ένας κατευθυνόμενος γράφος με 5 κορυφές και 6 ακμές. Εδώ οι απευθείας μεταβάσεις επιτρέπονται μόνο από τον κόμβο που ξεκινάει το βέλος (ακμή) προς τον κόμβο που δείχνει. Για παράδειγμα επιτρέπεται μόνο η μετάβαση από την κορυφή 0 προς την 1 και όχι από την 1 προς την 0, αυτό σημαίνει πως υπάρχει μόνο η ακμή $(0, 1)$ χωρίς την $(1, 0)$ ενώ στην περίπτωση των κορυφών 3 και 4 επιτρέπεται η μετάβαση μεταξύ τους και από τις δύο κατευθύνσεις.

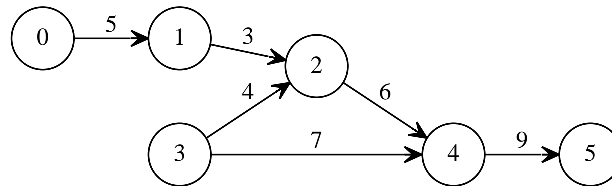
Δύο κορυφές σε έναν γράφο ονομάζονται γειτονικές αν υπάρχει μια ακμή που να τις ενώνει [7]. Δηλαδή μία κορυφή u είναι γειτονική με μία κορυφή v αν υπάρχει μια ακμή από την u προς την v ή από την v προς την u. Για παράδειγμα στο Σχήμα 2.1.1 α΄ οι κορυφές 0 και 1 είναι γειτονικές. Μια κορυφή v γειτνιάζει με μια κορυφή u όταν στο



(α′) Μη κατευθυνόμενος γράφος



(β′) Κατευθυνόμενος γράφος



(γ′) Εμβαρής κατευθυνόμενος γράφος

Σχήμα 2.1.1: Βασικοί τύποι γράφων

γράφο υπάρχει μια ακμή (u, v) [1].

Ένας γράφος ονομάζεται απλός αν δεν υπάρχουν ακμές που ξεκινούν και καταλήγουν στην ίδια κορυφή και αν δεν υπάρχει παραπάνω από μια ίδια ακμή για την σύνδεση δύο κορυφών [7]. Όλοι οι γράφοι που απεικονίζονται στο Σχήμα 2.1.1 είναι απλοί.

Ένας γράφος ονομάζεται πλήρης αν υπάρχει μία ακμή μεταξύ κάθε ζεύγος των κορυφών του [7]. Αν ο γράφος είναι απλός και μη κατευθυνόμενος τότε ο μέγιστος αριθμός των ακμών του είναι $\frac{|V|(|V|-1)}{2}$ ενώ αν είναι απλός και κατευθυνόμενος $|V|(|V|-1)$ [7].

Ως διαδρομή ή μονοπάτι ενός γράφου ονομάζεται μια ακολουθία γειτονικών κορυφών που ξεκινάει από μια κορυφή και καταλήγει σε μια άλλη [7]. Ενδεικτικά ένα μονοπάτι στο Σχήμα 2.1.1 β′ είναι το $1 \to 3 \to 4$.

Πυκνός γράφος ονομάζεται ένας γράφος στον οποίο ο αριθμός των ακμών του είναι κοντά στο μέγιστο αριθμό ακμών [2, 7]. Αν ο αριθμός των ακμών του είναι κοντά στον αριθμό των κορυφών του τότε ο γράφος ονομάζεται αραιός [2].

Ένας γράφος ονομάζεται εμβαρής όταν κάθε ακμή του φέρει ένα βάρος [1]. Το βάρος δηλώνει το κόστος μιας μετάβασης από μια κορυφή του γράφου σε μια άλλη, σε έναν γράφο χωρίς βάρη το κόστος μιας μετάβασης θεωρείται ίσο με το 1, αυτό σημαίνει ότι το κόστος μίας διαδρομής ισούται με τον αριθμό των ακμών από τον οποίο αποτελείται [7]. Για παράδειγμα στο Σχήμα 2.1.1 (α′) το κόστος της διαδρομής $0 \to 2 \to 1 \to 4$ είναι 3 και στο (γ′) η διαδρομή $1 \to 2 \to 4 \to 5$ έχει κόστος 18.

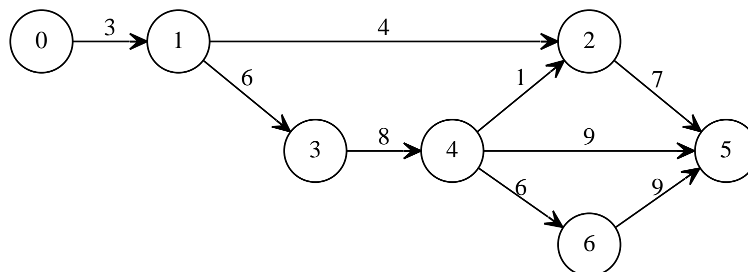
2.2 Αναπαραστάσεις

Οι γράφοι μπορούν να αναπαρασταθούν με διάφορους τρόπους. Οι δύο κυριότεροι είναι ο πίνακας γειτνίασης και οι λίστες γειτνίασης [1, 2].

Στην αναπαράσταση μέσω πίνακα γειτνίασης χρησιμοποιείται ένας δισδιάστατος πίνακας $|V| \times |V|$ [1, 2]. Για κάθε θέση $[i, j]$ του πίνακα, αν υπάρχει ακμή από την κορυφή i προς την j τότε θα τοποθετείται στη θέση αυτή η τιμή 1 αλλιώς η τιμή 0 [1, 2]. Αν ο γράφος είναι εμβαρής τότε αντί να τοποθετείται η τιμή 1 θα τοποθετείται η τιμή του βάρους της ακμής [1]. Ο χώρος που χρειάζεται για να αποθηκευτεί ένας πίνακας γειτνίασης ανεξάρτητα από το πλήθος των ακμών του γράφου είναι $O(|V|^2)$ [1, 2]. Καθώς αυτός ο τρόπος αναπαράστασης καταλαμβάνει το συγκεκριμένο χώρο ακόμα και όταν οι ακμές είναι λίγες προτιμάται όταν ο γράφος είναι πυκνός [1, 2]. Επίσης προτιμάται όταν είναι απαραίτητο να γίνεται γρήγορα έλεγχος σύνδεσης μεταξύ δύο κορυφών καθώς ο χρόνος που χρειάζεται για να πραγματοποιηθεί αυτό είναι $O(1)$ [1, 2].

Στην αναπαράσταση με λίστες γειτνίασης χρησιμοποιείται ένας μονοδιάστατος πίνακας μεγέθους $|V|$ του οποίου τα στοιχεία είναι λίστες που θα συμβολίζουν τις κορυφές του γράφου [1]. Στους κόμβους τις κάθε λίστας θα αποθηκεύονται όλες οι κορυφές με τις οποίες υπάρχει ακμή που ξεκινάει με την κορυφή που αντιπροσωπεύει η κάθε λίστα [1]. Για παράδειγμα αν u μια κορυφή στο γράφο και Adj ο πίνακας λιστών τότε στη λίστα $Adj[u]$ θα αποθηκεύονται οι κορυφές στις οποίες μπορεί να γίνει μετάβαση από την u, δηλαδή όλες τις κορυφές που γειτνιάζουν με την u [1]. Εναλλακτικά στους κόμβους των λιστών θα μπορούσαν να αποθηκεύονται δείκτες προς τις κορυφές του γράφου [1]. Αν ο γράφος είναι εμβαρής τότε θα αποθηκεύεται και το βάρος σε κάθε κόμβο των λιστών [1]. Ο χώρος που χρειάζεται την αναπαράσταση με λίστες γειτνίασης είναι $O(|V| + |E|)$ [1]. Ο συγκεκριμένος τρόπος αναπαράστασης προτιμάται όταν ο γράφος είναι πολύ αραιός καθώς ο χώρος που καταλαμβάνει εξαρτάται και από το πλήθος των ακμών του [1].

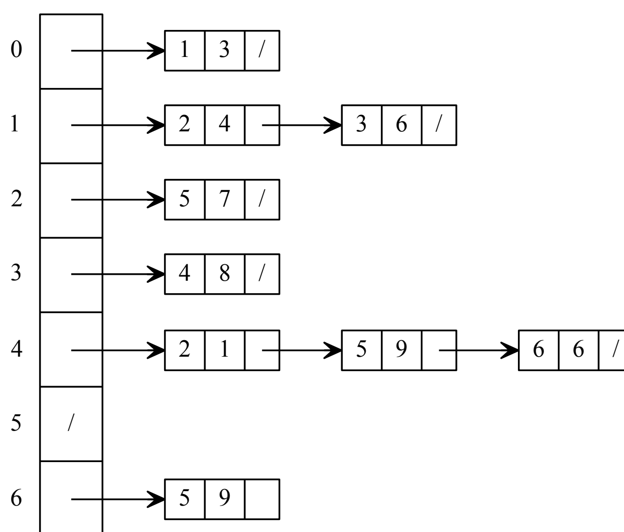
Στο Σχήμα 2.2.1 (α΄) απεικονίζεται ένας κατευθυνόμενος εμβαρής γράφος και στα (β΄) και (γ΄) αντίστοιχα οι αναπαραστάσεις του με πίνακα και λίστες γειτνίασης. Στο (γ΄) κάθε κόμβος στις λίστες γειτνίασης αποτελείται από τρία πεδία, το πρώτο είναι η κορυφή, το δεύτερο το βάρος και το τρίτο ο δείκτης προς τον επόμενο κόμβο. Ο συγκεκριμένος γράφος είναι πολύ αραιός, συνεπώς σε αυτήν την περίπτωση η αναπαράσταση με λίστες γειτνίασης είναι προτιμότερη όσον αφορά το χώρο αποθήκευσης καθώς δε χρειάζεται να αποθηκεύονται και οι ελλείψεις ακμών όπως στον πίνακα γειτνίασης.



(α΄) Απλός κατευθυνόμενος εμβαρής γράφος

	0	*1*	*2*	3	4	5	6
0	0	3	0	0	0	0	0
1	0	0	4	6	0	0	0
2	0	0	0	0	0	7	0
3	0	0	0	0	8	0	0
4	0	0	1	0	0	9	6
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	9

(β΄) Πίνακας γειτνίασης



(γ΄) Λίστες γειτνίασης

Σχήμα 2.2.1: Αναπαραστάσεις γράφων

3 Σωρός Fibonacci

3.1 Δομή των σωρών Fibonacci

Ο σωρός Fibonacci είναι μια δομή δεδομένων που μπορεί να υλοποιήσει τον αφηρημένο τύπο δεδομένων ουρά προτεραιότητας .

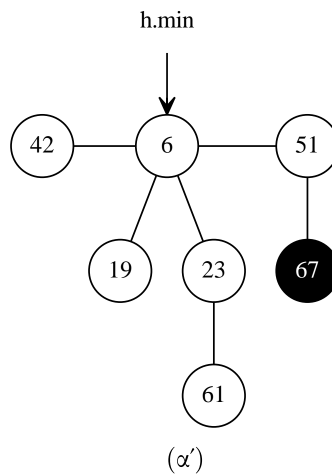
Ένας σωρός Fibonacci αποτελείται από μια συλλογή δένδρων όπου κάθε ένα από αυτά ικανοποιεί την ιδιότητα σωρού ελαχίστου, δηλαδή κάθε γονέας έχει τιμή μικρότερη ή ίση από την τιμή των παιδιών του [1, 4].

Κάθε κόμβος x σε έναν σωρό Fibonacci περιέχει ένα δείκτη προς το γονέα του x.γονέας, ένα δείκτη προς ένα από τα παιδιά του x.παιδί, ένα δείκτη προς τον αριστερό του αδελφό x.αριστερός , ένα δείκτη προς το δεξιό του αδελφό x.δεξιός, έναν ακέραιο αριθμό x.βαθμός στον οποίο θα αποθηκεύεται ο αριθμός των παιδιών του, ένα δυφίο (bit) x.σήμανση και το πεδίο x.κλειδί [1, 4]. Η χρήση του πεδίου της σήμανσης θα αναφερθεί στην πράξη της μείωσης κλειδιού. Οι κόμβοι στους οποίους το πεδίο της σήμανσης έχει την τιμή ΑΛΗΘΕΣ θα ονομάζονται επισημασμένοι [1, 4].

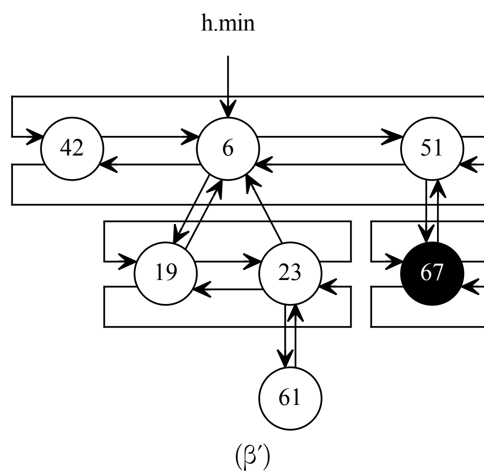
Κάθε κόμβος σε ένα σωρό Fibonacci συνδέεται με τα αδέλφια του σε μια διπλά κυκλικά συνδεδεμένη λίστα, αν ένας κόμβος είναι ο μοναδικός σε μια τέτοια λίστα τότε οι δείκτες προς τα αδέλφια του θα δείχνουν προς τον εαυτό του [1, 4].

Σε ουρές προτεραιοτήτων εισάγονται αντικείμενα όπου το κάθε ένα περιέχει μια προτεραιότητα, στους σωρούς Fibonacci το πεδίο x.κλειδί είναι η προτεραιότητα, για να αποθηκεύονται αντικείμενα αρκεί να προστεθεί και το πεδίο x.αντικείμενο στους κόμβους. Τα αντικείμενα είναι ουσιαστικά τα στοιχεία που θα αποθηκεύονται με τη δεδομένη προτεραιότητα και δεν επηρεάζουν τη λειτουργία των πράξεων του σωρού καθώς το πεδίο αυτό δε θα προσπελαστεί από καμία πράξη του. Οι τιμές τους καθώς και των κλειδιών μπορούν να έχουν οποιαδήποτε μορφή όμως για λόγους απλότητας και χωρίς κάποιο περιορισμό τα κλειδιά θα αποθηκεύουν ακέραιους αριθμούς και τα αντικείμενα θα είναι τα ίδια τα κλειδιά.

Ένας σωρός Fibonacci h περιέχει ένα δείκτη $h.min$ προς το ριζικό κόμβο που έχει τη μικρότερη τιμή κλειδιού, με αυτό το δείκτη θα αποκτάται η πρόσβαση στο σωρό και στο ριζικό του επίπεδο [1, 4]. Αν ο δείκτης αυτός έχει την τιμή ΚΕΝΟ σημαίνει ότι ο σωρός είναι κενός, δηλαδή δεν περιέχει κανέναν κόμβο [1, 4]. Επίσης σε κάθε σωρό Fibonacci θα υπάρχει και το πεδίο $h.n$ στο οποίο θα αποθηκεύεται ο αριθμός των κόμβων που βρίσκονται στο σωρό [1]. Αυτό το πεδίο μπορεί επίσης να χρησιμοποιηθεί



(α′)



(β′)

Σχήμα 3.1.1: Δομή ενός σωρού Fibonacci

για να ελεγχθεί αν ένας σωρός Fibonacci είναι κενός.

Στο σχήμα 3.1.1 απεικονίζεται η δομή ενός σωρού Fibonacci με τρία δένδρα και επτά κόμβους. Οι κόμβοι με τις τιμές 42, 6 και 51 είναι οι ρίζες των δένδρων. Στο (α′) η απεικόνιση γίνεται με βασικό τρόπο. Στο (β′) φαίνονται αναλυτικά οι συνδέσεις μεταξύ των κόμβων, οι αδελφικοί κόμβοι συνδέονται με οριζόντια βέλη, οι γονείς με ανηφορικά και τα παιδιά τους με κατηφορικά. Και στα δύο σχήματα ένας κόμβος είναι μαύρος, ένας κόμβος θα έχει μαύρο χρώμα αν είναι επισημασμένος. Σε όλα τα επόμενα σχήματα η απεικόνιση θα γίνεται με το βασικό τρόπο καθώς οι πληροφορίες των συνδέσεων που φαίνονται στο (β′) μπορούν να αποκτηθούν από το (α′).

3.2 Πράξεις των σωρών Fibonacci

Ένας σωρός Fibonacci υποστηρίζει τις παρακάτω πράξεις: [1, 4]

Δημιουργία Σωρού*()* δημιουργεί και επιστρέφει έναν άδειο σωρό.

Εισαγωγή(h, x) εισάγει στο σωρό h τον κόμβο x.

Ελάχιστο(h) επιστρέφει τον κόμβο του σωρού h του οποίου το κλειδί είναι το ελάχιστο.

Εξαγωγή Ελαχίστου(h) εξάγει και επιστρέφει τον κόμβο της σωρού h του οποίου το κλειδί είναι το ελάχιστο, διαγράφοντάς τον από τη σωρό.

Συγχώνευση$(h1, h2)$ δημιουργεί και επιστρέφει ένα νέο σωρό στον οποίο περιέχονται τα στοιχεία των σωρών $h1$ και $h2$. Η πράξη αυτή χρησιμοποιεί τα ίδια τα στοιχεία των σωρών και όχι αντίγραφα τους συνεπώς οι δύο αυτοί σωροί καταστρέφονται.

Μείωση Κλειδιού(h, x, k) αντικαθιστά στο σωρό h το κλειδί του κόμβου x με το κλειδί k το οποίο θα πρέπει να έχει τιμή μικρότερη ή ίση με αυτή του κλειδιού του κόμβου x.

Διαγραφή(h, x) διαγράφει από τον σωρό h τον κόμβο x.

Επίσης θα προστεθεί και η πράξη Κατασκευή Σωρού(A) που θα δημιουργεί και θα επιστρέφει ένα σωρό Fibonacci από τον πίνακα A ο οποίος θα περιέχει δείκτες προς κόμβους με συμπληρωμένα κλειδιά.

Για τον υπολογισμό της λογιστικής πολυπλοκότητας των πράξεων των σωρών Fibonacci θα χρησιμοποιηθεί αντισταθμιστική ανάλυση με την ενεργειακή μέθοδο [1, 4].

Για ένα σωρό Fibonacci h ο αριθμός των δένδρων του θα έχει τον συμβολισμό $t(h)$ και ο αριθμός των επισημασμένων κόμβων του τον συμβολισμό $m(h)$ [1]. Το δυναμικό του θα ορίζεται από τη σχέση $\Phi(h) = t(h) + 2m(h)$ [1]. Για παράδειγμα το δυναμικό του σωρού Fibonacci στο Σχήμα 3.1.1 είναι $3 + 2 \cdot 1 = 5$. Το δυναμικό μιας συλλογής σωρών Fibonacci είναι το άθροισμα του δυναμικού κάθε σωρού στη συλλογή [1].

3.2.1 Δημιουργία

Για να δημιουργηθεί ένας νέος σωρός Fibonacci θα δεσμεύεται και θα επιστρέφεται ένα αντικείμενο σωρού Fibonacci h με το πεδίο $h.min$ να έχει την τιμή KENO και το πεδίο $h.n$ την τιμή 0 [1].

Δημιουργία Σωρού Fibonacci*(h,x)*
1. Δέσμευση νέου Σωρού Fibonacci h
2. $h.min \leftarrow$ KENO
3. $h.n = 0$
4. επιστροφή h

Ο πραγματικός χρόνος αυτής της πράξης είναι $O(1)$ [1, 4].

Καθώς ο σωρός είναι κενός, δηλαδή δεν έχει κανέναν κόμβο το δυναμικό του είναι $\Phi(h) = 0$ άρα ο λογιστικός χρόνος είναι επίσης $O(1)$ [1].

3.2.2 Εισαγωγή

Για να εισαχθεί ένας κόμβος σε ένα σωρό Fibonacci θα προστίθεται στο ριζικό του επίπεδο [1, 4].

Εισαγωγή Σε Σωρό Fibonacci(h, x)
1. Αρχικοποίηση Κόμβου(x)
2. Αν $h.min$ = ΚΕΝΟ
3. $\quad h.min \leftarrow x$.αριστερός $\leftarrow x$.δεξιός $\leftarrow x$
4. Αλλιώς
5. $\quad$ Προσθήκη Κόμβου$(h.min, x)$
6. $\quad$ Αν x.κλειδί $<$ $h.min$.κλειδί
7. $\quad\quad h.min \leftarrow x$
8. $h.n \leftarrow h.n + 1$

Αρχικοποίηση Κόμβου(x)
1. x.βαθμός $\leftarrow 0$
2. x.γονέας $\leftarrow$ ΚΕΝΟ
3. x.παιδί $\leftarrow$ ΚΕΝΟ
4. x.σήμανση $\leftarrow$ ΨΕΥΔΕΣ

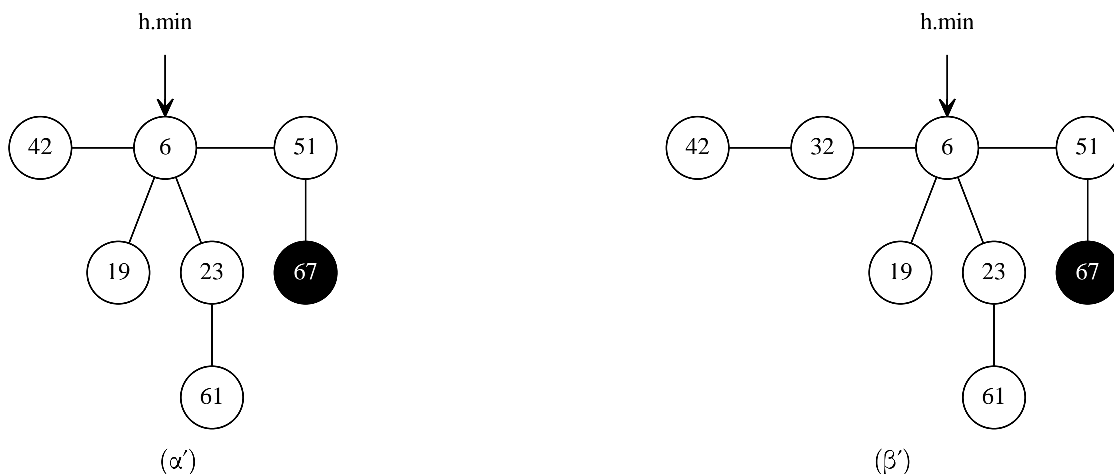
Στη γραμμή 1 αρχικοποιούνται τα πεδία του κόμβου [1]. Στη γραμμή 2 ελέγχεται αν ο σωρός h είναι κενός, αν ναι τότε ο κόμβος θα έχει ως αριστερό και δεξί αδελφό τον εαυτό του και ο δείκτης $h.min$ θα δείχνει προς τον κόμβο αυτό (γραμμή 3). Οι γραμμές 4-7 είναι για την περίπτωση που ο σωρός δεν είναι κενός. Ο κόμβος θα προστίθεται στο ριζικό επίπεδο με την πράξη Προσθήκη Κόμβου. Αν το κλειδί του έχει μικρότερη τιμή από αυτή του $h.min$ τότε ο δείκτης $h.min$ θα δείχνει προς τον κόμβο αυτό [1]. Αφού ο κόμβος εισαχθεί στο ριζικό επίπεδο θα αυξάνεται ο αριθμός των κόμβων του σωρού $h.n$ κατά ένα στη γραμμή 8 [1].

Η πράξη Προσθήκη Κόμβου(x, y) προσθέτει τον κόμβο y αριστερά από τον κόμβο x στη διπλά κυκλικά συνδεδεμένη λίστα που βρίσκεται ο x.

Προσθήκη Κόμβου(x, y)
1. x.αριστερός.δεξιός $\leftarrow y$
2. y.αριστερός $\leftarrow x$.αριστερός
3. x.αριστερός $\leftarrow y$
4. y.δεξιός $\leftarrow x$

Ο κόμβος που βρίσκεται αριστερά από τον x θα έχει ως δεξιό αδελφό τον y και ο y θα τον έχει ως αριστερό αδελφό. Ο x θα έχει ως αριστερό αδελφό τον y και ο y ως δεξιό αδελφό τον x. Η πράξη αυτή θα λειτουργεί και για την περίπτωση που ο x είναι ο μοναδικός κόμβος στη λίστα του καθώς αν δεν υπάρχει άλλος κόμβος θα έχει ως αδέλφια τον εαυτό του οπότε x.αριστερός και x.δεξιός θα είναι ο ίδιος ο x.

Στο Σχήμα 3.2.2.1 απεικονίζεται ένας σωρός Fibonacci (α′) πριν και (β′) μετά την εισαγωγή του κόμβου με το κλειδί 32.



Σχήμα 3.2.2.1: Εισαγωγή σε σωρό Fibonacci

Ο πραγματικός χρόνος της πράξης Εισαγωγή Σε Σωρό Fibonacci είναι $O(1)$ [1]. Έστω h ο σωρός πριν την εισαγωγή του νέου κόμβου και h' ο τελικός σωρός, $t(h') = t(h) + 1$ καθώς ο κόμβος προστέθηκε ως ρίζα στο σωρό και επειδή οι κόμβοι που εισάγονται δεν είναι επισημασμένοι $m(h') = m(h)$ [1]. Άρα η αύξηση του δυναμικού είναι $(t(h) + 1) + 2m(h)) - (t(h) + 2m(h)) = 1$ [1]. Ο λογιστικός χρόνος της πράξης είναι $O(1) + 1 = O(1)$, δηλαδή ίδιος με τον πραγματικό [1].

3.2.3 Συγχώνευση

Για να πραγματοποιηθεί συγχώνευση δύο σωρών Fibonacci θα δημιουργείται και θα επιστρέφεται ένας νέος σωρός Fibonacci ο οποίος θα αποτελείται από την ένωση των ριζικών κόμβων των δύο σωρών και θα προσδιορίζεται ο νέος ελάχιστος κόμβος [1, 4].

Συγχώνευση Σωρών Fibonacci*($h1, h2$)*
1. $h \leftarrow$Δημιουργία Σωρού Fibonacci()
2. $h.min \leftarrow h1.min$
3. αν $h1.min =$ KENO
4. $h.min \leftarrow h2.min$
5. αλλιώς αν $h2.min \neq$ KENO
6.   Συνένωση Λιστών*($h1.min, y$)*
7.   αν $h2.min$.κλειδί $< h1.min$.κλειδί
8. $h.min \leftarrow h2.min$
9. $h.n \leftarrow h1.n + h2.n$
10. αποδέσμευση των αντικειμένων $h1$ και $h2$
11. επιστροφή h

Στις γραμμές 1-2 δημιουργείται ένας νέος σωρός Fibonacci και αρχικά θεωρείται ότι ο $h1$ περιέχει το ελάχιστο στοιχείο [1]. Στις γραμμές 3-4 ελέγχεται αν ο σωρός είναι κενός και αν ναι τότε ο $h2$ περιέχει το ελάχιστο. Οι γραμμές 4-8 είναι για την

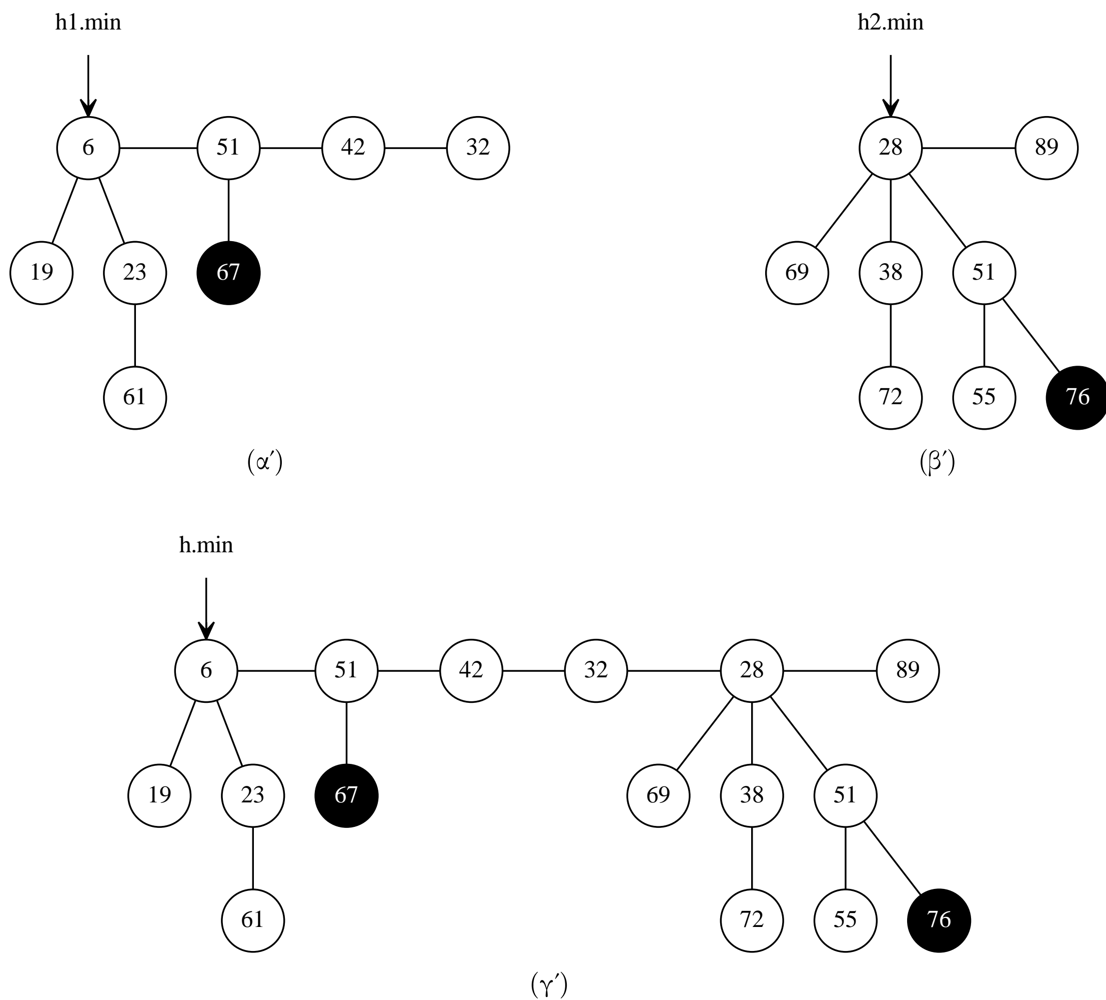
περίπτωση που ούτε ο $h1$ ούτε ο $h2$ είναι κενοί. Καλείται η πράξη Συνένωση λιστών για να ενωθούν οι λίστες που βρίσκονται οι $h1.min$ και $h2.min$, δηλαδή οι ριζικοί τους κόμβοι. Από το βήμα 2 το $h.min$ είναι ίδιο με το $h1.min$ οπότε αρκεί να ελεγχθεί αν το $h2.min$ έχει μικρότερο κλειδί από αυτό και αν ναι τότε έχει και το ελάχιστο των δύο σωρών. Στις γραμμές 9-11 υπολογίζεται ο αριθμός κόμβων του νέου σωρού, αποδεσμεύονται ο $h1$ και $h2$ και επιστρέφεται ο νέος σωρός h [1]. Αν και οι δύο σωροί ήταν κενοί σε αυτό το σημείο θα επιστραφεί ένας κενός σωρός με αριθμό κόμβων 0.

Για να πραγματοποιηθεί η συνένωση των ριζικών κόμβων θα χρησιμοποιηθεί η πράξη Συνένωση Λιστών(χ, y) η οποία θα ενώνει τις λίστες που βρίσκονται δύο κόμβοι σωρών Fibonacci x και y.

Συνένωση Λιστών(χ, y)
1. $t1 \leftarrow x$.αριστερός
2. $t2 \leftarrow y$.αριστερός
3. $t1$.δεξιός $\leftarrow y$
4. y.αριστερός $\leftarrow t1$
5. $t2$.δεξιός $\leftarrow x$
6. x.αριστερός $\leftarrow t2$

Χρησιμοποιούνται δύο προσωρινοί δείκτες $t1$ και $t2$ που αντίστοιχα θα δείχνουν στον αριστερό αδελφό του x και του y. Καθώς οι λίστες των κόμβων είναι διπλά κυκλικά συνδεδεμένες ο x και ο $t1$ είναι τα δύο άκρα της λίστας του x και αρχή της θα θεωρείται ο x ενώ το τέλος της ο $t1$. Αντίστοιχα για τον y και $t2$. Οπότε για να ενωθούν οι δύο λίστες αρκεί να συνδεθούν και από τις δύο κατευθύνσεις μέσω του αριστερού και δεξιού αδελφού το τέλος της πρώτης λίστας με την αρχή της δεύτερης και το τέλος της δεύτερης με την αρχή της πρώτης. Ανεξάρτητα από το πόσο μεγάλες είναι οι λίστες ο χρόνος της πράξης είναι $O(1)$.

Στο Σχήμα 3.2.3.1 απεικονίζονται στο (α΄) και (β΄) ο πρώτος και ο δεύτερος σωρός Fibonacci και στο (γ΄) ο νέος σωρός που δημιουργείται μετά τη συγχώνευσή τους.



Σχήμα 3.2.3.1: Συγχώνευση σωρών Fibonacci

Ο πραγματικός χρόνος της πράξης Συγχώνευση Σωρών Fibonacci είναι $O(1)$ [1].

Ο αριθμός δένδρων στο σωρό h είναι το άθροισμα του αριθμού δένδρων των σωρών $h1$ και $h2$. Ομοίως για τους επισημασμένους κόμβους.

Άρα $t(h) = t(h1) + t(h2)$ και $m(h) = m(h1) + m(h2)$ [1].

Η μεταβολή του δυναμικού είναι: [1]

$\Phi(h) - \Phi(h1) + \Phi(h2) = (t(h) + 2m(h)) - ((t(h1) + 2m(h1)) + (t(h2) + 2m(h))) = 0.$

Οπότε ο λογιστικός χρόνος είναι $O(1)$, ίσος με τον πραγματικό [1].

3.2.4 Ελάχιστο

Ένας σωρός Fibonacci περιέχει ήδη έναν δείκτη προς τον ελάχιστο κόμβο στο πεδίο $h.min$ [1]. Συνεπώς αρκεί η πράξη Ελάχιστο Σε Σωρό Fibonacci να επιστρέφει τον δείκτη $h.min$.

Ελάχιστο Σε Σωρό Fibonacci*(h)*
1. επιστροφή $h.min$

Ο πραγματικός χρόνος της πράξης είναι $O(1)$ και επειδή δε γίνεται κάποια μεταβολή στο δυναμικό του h ο λογιστικός χρόνος είναι επίσης $O(1)$ [1].

3.2.5 Εξαγωγή ελαχίστου

Για να εξαχθεί ο ελάχιστος κόμβος από ένα σωρό Fibonacci θα αφαιρείται από το ριζικό επίπεδο και σε αυτό θα προστεθούν οι κόμβοι της λίστας των παιδιών του [1, 4]. Ύστερα θα συνδέονται οι κόμβοι του ριζικού επιπέδου που έχουν τον ίδιο βαθμό μέχρι κάθε ένας που θα απομείνει να έχει διαφορετικό βαθμό από τους άλλους και θα βρίσκεται ο νέος ελάχιστος κόμβος [1, 4].

Εξαγωγή Ελαχίστου Από Σωρό Fibonacci*(h)*
1. αν $h.min$ = ΚΕΝΟ
2.   επιστροφή ΚΕΝΟ
3. $z \leftarrow h.min$
4. αν z.παιδί $\neq$ ΚΕΝΟ
5.   για κάθε παιδί x του z
6. x.γονέας $\leftarrow$ ΚΕΝΟ
7.   Συνένωση Λιστών$(h.min, x)$
8. Αφαίρεση Κόμβου(z)
9. αν $z = z$.δεξιός
10. $h.min \leftarrow$ ΚΕΝΟ
11. αλλιώς
12. $h.min \leftarrow z$.δεξιός
12.   Ενοποίηση(h)
13. $h.n \leftarrow h.n - 1$
14. επιστροφή z

Αφαίρεση Κόμβου *(x)*
1. x.αριστερός.δεξιός $\leftarrow x$.δεξιός
2. x.δεξιός.αριστερός $\leftarrow x$.αριστερός

Στις γραμμές 1-2 ελέγχεται η περίπτωση που ο σωρός είναι κενός και αν ναι επιστρέφεται ΚΕΝΟ και η πράξη ολοκληρώνεται [1]. Στις γραμμές 3-8 αποθηκεύεται ένας δείκτης z προς τον ελάχιστο κόμβο, όλα τα παιδιά του θα προστεθούν στο ριζικό επίπεδο και καθώς κανένα τους δε θα έχει πλέον γονέα το πεδίο αυτό θα έχει την τιμή ΚΕΝΟ και ο κόμβος z θα αφαιρεθεί [1]. Για να μην προσθέτονται οι κόμβοι της λίστας παιδιών του z ένας ένας στο ριζικό επίπεδο χρησιμοποιείται η πράξη Συνένωση Λιστών και ο κόμβος z αφαιρείται με την πράξη Αφαίρεση Κόμβου. Η πράξη αφαίρεση κόμβου ενώνει τον αριστερό αδελφό του z με τον δεξιό, δεν είναι πλέον συνδεδεμένοι με αυτόν όμως οι δικοί του δείκτες παραμένουν προς τα αδέλφια του. Στις γραμμές 9-12 ελέγχεται αν ο δεξιός αδελφός του z είναι ο ίδιος ο z και αν ναι τότε είναι ήταν ο μοναδικός κόμβος στο σωρό (δεν υπήρχαν άλλοι κόμβοι και δεν είχε παιδιά) και ο σωρός είναι πλέον άδειος. [1]. Αλλιώς ο δείκτης $h.min$ θα δείχνει στο δεξιό αδελφό του z χωρίς να

σημαίνει ότι αυτός θα είναι ο ελάχιστος και καλείται η πράξη Ενοποίηση για να μειώσει τον αριθμό δένδρων στο ριζικό επίπεδο και να βρεθεί ο νέος ελάχιστος κόμβος [1]. Στις γραμμές 13-14 μειώνεται ο αριθμός κόμβων του h κατά 1 και επιστρέφεται ο z [1].

Για την πράξη Ενοποίηση θα είναι απαραίτητη και η πράξη Σύνδεση(h, y, x) [1]. Η πράξη αυτή αφαιρεί τον y από το ριζικό επίπεδο του h και τον κάνει παιδί του x αυξάνοντας το βαθμό του και στο πεδίο της σήμανσης του y εκχωρείται η τιμή ΨΕΥΔΕΣ [1]. Καθώς η πράξη αυτή θα χρησιμοποιηθεί μόνο στην πράξη της ενοποίησης, η οποία στο τέλος θα επαναδημιουργήσει το ριζικό επίπεδο από έναν πίνακα στον οποίο ο κάθε y δε θα υπάρχει, η αφαίρεση του κόμβου y στην πράξη Σύνδεση θα παραλειφθεί επειδή δεν είναι απαραίτητο να ενημερωθούν οι δείκτες δεξιός και αριστερός των άλλων ριζικών κόμβων για την απουσία του από το ριζικό επίπεδο. Για λόγους απλότητας όμως στα σχήματα θα φαίνεται συνδεδεμένος μόνο με τον x και τη λίστα παιδιών του.

Σύνδεση(y, x)
1. αν x.παιδί $\neq$ ΚΕΝΟ
2. Προσθήκη Κόμβου$(x$.παιδί$, y)$
3. αλλιώς
4. x.παιδί $\leftarrow$ y.αριστερός $\leftarrow$ y.δεξιός $\leftarrow$ y
5. y.γονέας $\leftarrow$ x
6. x.βαθμός $\leftarrow$ x.βαθμός $+1$
7. y.σήμανση $\leftarrow$ ΨΕΥΔΕΣ

Στις γραμμές 1-5 ελέγχεται αν ο x έχει ήδη παιδιά και αν ναι χρησιμοποιείται η πράξη Προσθήκη κόμβου για να προσθέσει τον y στη λίστα τους, αλλιώς ο y θα είναι το μοναδικό παιδί του x. Για να ολοκληρωθεί η σύνδεση ο y θα έχει πλέον ως γονέα τον x. Ο βαθμός του x αυξάνεται κατά ένα και αφαιρείται η επισήμανση από τον y (γραμμές 6-7) [1].

Η πράξη Ενοποίηση βρίσκει κόμβους στο ριζικό επίπεδο που έχουν τον ίδιο βαθμό και τους συνδέει επανειλημμένα κάνοντας κάθε έναν με το μικρότερο κλειδί γονέα αυτού με το μεγαλύτερο [1, 4]. Μετά το τέλος της πράξης κάθε κόμβος στο ριζικό επίπεδο θα έχει μοναδικό βαθμό [1, 4].

Στη πράξη Ενοποίηση θα χρησιμοποιηθεί ένας πίνακας στον οποίο θα αποθηκεύονται δείκτες προς τους ριζικούς κόμβους [1, 4]. Στις θέσεις από το 0 έως και το $D(h.n)$ θα μπαίνουν ριζικοί κόμβοι που θα έχουν τον ίδιο βαθμό με τη θέση του πίνακα έτσι ώστε αν $A[i] = y$ τότε εκείνη τη στιγμή ο κόμβος y θα έχει βαθμό ίσο με το i [1]. Η επιπρόσθετη θέση μετά την $D(h.n)$ είναι απαραίτητη μόνο για να μη βγει εκτός πίνακα η επανάληψη των γραμμών 8-15 οπότε δε θα εμφανίζεται στα σχήματα. Ο μέγιστος βαθμός που μπορεί να έχει οποιοσδήποτε κόμβος σε ένα σωρό Fibonacci με n κόμβους είναι ίσος με $lg(n)$ [1, 4]. Αυτό θα αποδειχθεί στην τελευταία ενότητα του κεφαλαίου 3. Ο συμβολισμός του θα είναι $D(h.n)$ [1].

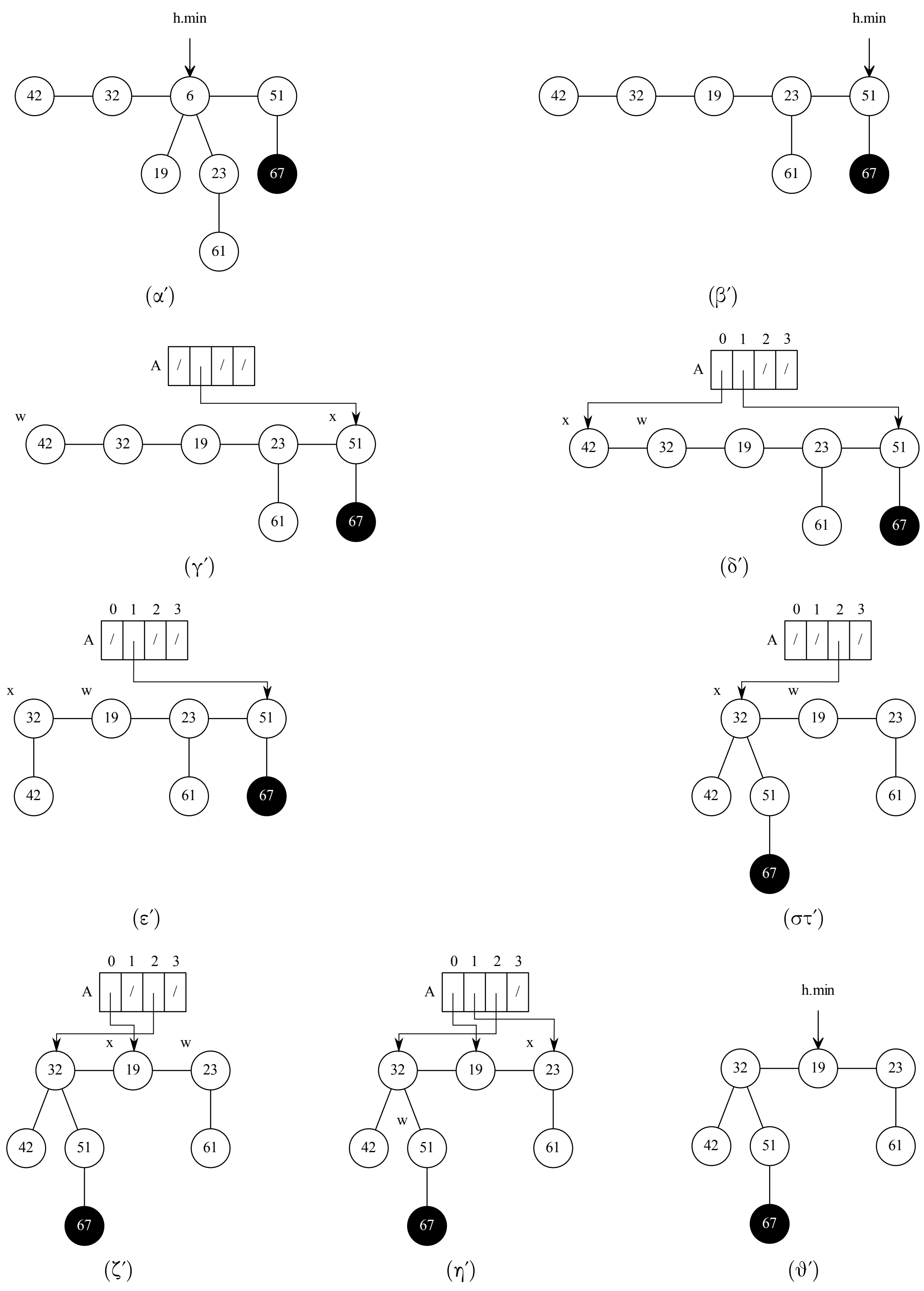
Στις γραμμές 1-3 αρχικοποιούνται τα στοιχεία του πίνακα A στην τιμή ΚΕΝΟ και

```
Ενοποίηση(h)
 1.  για i ← 0 έως D(h.n) + 1
 2.       A[i] ← ΚΕΝΟ
 3.  w ← h.min
 4.  επανάληψη:
 5.       x ← w
 6.       d ← x.βαθμός
 7.       w ← w.δεξιός
 8.       όσο A[d] ≠ΚΕΝΟ
 9.            y ← A[d]
10.            αν x.κλειδί > y.κλειδί
11.                 y ← x
12.                 x ← A[d]
13.            Σύνδεση(y, x)
14.            A[d] ← ΚΕΝΟ
15.            d ← d + 1
16.       A[d] ← x
17.  όσο w ≠ h.min
18.  h.min ← ΚΕΝΟ
19.  για i ← 0 έως D(h.n)
20.       αν A[i] ≠ ΚΕΝΟ
21.            αν h.min = ΚΕΝΟ
22.                 h.min ← A[i].αριστερός ← A[i].δεξιός ← A[i]
23.            αλλιώς
24.                 Προσθήκη Κόμβου(h.min, A[i])
25.                 αν A[i].κλειδί < h.min.κλειδί
26.                      h.min ← A[i]
```

αποθηκεύεται στο δείκτη w ο δείκτης $h.min$ χωρίς να σημαίνει απαραίτητα ότι είναι ο ελάχιστος [1]. Η επανάληψη των γραμμών 4-17 θα επεξεργαστεί όλους τους ριζικούς κόμβους [1]. Ο w θα χρησιμοποιείται για την προσπέλαση της λίστας, σε κάθε στάδιο της επανάληψης στον x θα αποθηκεύεται ο τρέχων κόμβος προς επεξεργασία w και στο d ο βαθμός του (γραμμές 5-6) [1]. Όταν ο w θα δείχνει ξανά προς τον $h.min$ λόγω του κύκλου στη λίστα η επανάληψη θα τερματίζει. Καθώς στα επόμενα βήματα ο τρέχων κόμβος μπορεί να γίνει παιδί κάποιου άλλου ο w θα πηγαίνει από τώρα στην επόμενη ρίζα ώστε να παραμείνει στο ριζικό επίπεδο (γραμμή 7). Η επανάληψη των γραμμών 8-15 συνδέει τον x με έναν άλλο ριζικό κόμβο ίδιου βαθμού μέχρι να μην υπάρχει άλλος με το συγκεκριμένο βαθμό [1]. Δε θα πραγματοποιηθεί αν $A[d]$ =ΚΕΝΟ, δηλαδή αν δεν υπάρχει στον πίνακα άλλος κόμβος με τον ίδιο βαθμό. Στον πίνακα μπορούν να υπάρχουν μόνο κόμβοι που κάποια στιγμή προσπελάστηκαν από τον w σε προηγούμενη επανάληψη. Αν στον πίνακα υπάρχει κόμβος με τον ίδιο βαθμό που έχει ο x τότε εκχωρείται στον y ο οποίος θα συνδεθεί με τον x και όποιος από τους δύο έχει το μικρότερο κλειδί θα γίνει ο γονέας του άλλου [1]. Η πράξη Σύνδεση(y, x) συνδέει τον y στον x, δηλαδή ο y θα γίνει παιδί του x, για αυτό και ο x εναλλάσσεται με τον

y αν αυτό χρειαστεί [1]. Αυτό σημαίνει πως ο x θα δείχνει πάντα σε ριζικό κόμβο μετά τη σύνδεση. Καθώς ο x απέκτησε ένα παραπάνω παιδί αυξάνεται ο βαθμός του κατά 1 και το πεδίο της σήμανσης του y γίνεται ΨΕΥΔΕΣ [1]. Επειδή ο y δε θα είναι πλέον στο ριζικό επίπεδο αφαιρείται ο δείκτης του από τον πίνακα A και αφού ο x θα έχει ένα παραπάνω παιδί ο d θα αυξηθεί επίσης κατά ένα [1]. Έτσι στη γραμμή 8 θα ελεγχθεί στον πίνακα αν υπάρχει άλλος κόμβος με το νέο βαθμό του x και αν ναι η διαδικασία αυτή θα επαναλαμβάνεται. Μετά την ολοκλήρωση της επανάληψης ο δείκτης $A[d]$ θα δείχνει προς τον x καθώς θα είναι ο μοναδικός με αυτό το βαθμό (γραμμή 16) [1]. Στη γραμμή 17 ελέγχεται αν ο w γύρισε στον $h.min$ για να σταματήσει η εξωτερική επανάληψη. Αφού ολοκληρωθεί ο πίνακας A θα περιέχει δείκτες προς όλους τους ριζικούς κόμβους του σωρού που έχουν απομείνει [1]. Στις γραμμές 18-26 επαναδημιουργείται το ριζικό επίπεδο του σωρού χρησιμοποιώντας τον πίνακα A και βρίσκεται ο νέος ελάχιστος κόμβος [1]. Ο τρόπος εισαγωγής κάθε κόμβου (γραμμές 21-26) είναι ίδιος με της πράξης Εισαγωγή Σε Σωρό Fibonacci με τη διαφορά ότι δεν αυξάνεται ο αριθμός κόμβων.

Στο Σχήμα 3.2.5.1 απεικονίζονται τα βήματα της πράξης Εξαγωγή Ελαχίστου Από Σωρό Fibonacci. Στο (α′) ο αρχικός σωρός Fibonacci και στο (β′) ο σωρός αφού κληθεί η πράξη Εξαγωγή Ελαχίστου Από Σωρό Fibonacci πριν ξεκινήσει η πράξη Ενοποίηση. Στα (γ′)-(θ′) η κατάσταση του σωρού, του πίνακα A και των δεικτών x και w κατά την πράξη Ενοποίηση. Στο (γ′) και (δ′) αντίστοιχα η πρώτη και δεύτερη εξωτερική επανάληψη των γραμμών 4-17, δε θα γίνει κάποια εσωτερική επανάληψη (γραμμές 8-15) καθώς οι κόμβοι με τα κλειδιά 51 και 42 έχουν μέχρι στιγμής μοναδικό βαθμό οπότε απλώς θα προστεθούν στις θέσεις του βαθμού τους στον πίνακα A. Στο (ε′) και (στ′) αντίστοιχα, οι δύο εσωτερικές επαναλήψεις που θα πραγματοποιηθούν κατά την τρίτη εξωτερική επανάληψη. Στο (ε′) ο κόμβος με το κλειδί 32 έχει γίνει γονέας του κόμβου με το κλειδί 42. Η εσωτερική επανάληψη πραγματοποιήθηκε επειδή στον έλεγχο της γραμμής 8 η θέση του πίνακα με το βαθμό του ($A[0]$) δεν ήταν κενή (υπήρχε ο δείκτης προς τον κόμβο με το κλειδί 42) και αφού είχε μικρότερο κλειδί από αυτόν έγινε ο γονέας του και αυξήθηκε ο βαθμός του κατά ένα. Στο (στ′) έγινε γονέας του κόμβου με το κλειδί 51 για τον ίδιο λόγο. Στο (ζ′) και (η′) οι δύο επόμενες εξωτερικές επαναλήψεις. Και στις δύο κάθε κόμβος είχε μοναδικό βαθμό οπότε απλώς προστέθηκαν στις θέσεις του βαθμού τους στον πίνακα A. Στο (θ′) η τελική μορφή του σωρού αφού επαναδημιουργηθεί το ριζικό του επίπεδο με τη χρήση του πίνακα A και βρεθεί ο νέος ελάχιστος κόμβος (γραμμές 18-26).



Σχήμα 3.2.5.1: Εξαγωγή ελαχίστου από σωρό Fibonacci

Έστω h η κατάσταση του σωρού πριν κληθεί η πράξη Εξαγωγή Ελαχίστου Από Σωρό Fibonacci [1].

Η επανάληψη των γραμμών 5-6 της εξαγωγής ελαχίστου θα πραγματοποιηθεί για κάθε παιδί του ελάχιστου κόμβου και σε αυτήν θα επεξεργαστούν το πολύ $D(n)$ κόμβοι και στη γραμμή 7 θα προστεθούν στο ριζικό επίπεδο [1].

Πριν κληθεί η πράξη της ενοποίησης ο αριθμός κόμβων στο ριζικό επίπεδο θα είναι το πολύ $D(n) + t(h) - 1$ [1]. Αρχικά το ριζικό επίπεδο είχε $t(n)$ κόμβους, σε αυτούς προστέθηκαν κατά την εξαγωγή του ελάχιστου κόμβου τα παιδιά του $(D(n))$ και ο ίδιος αφαιρέθηκε (-1) [1]. Στην εσωτερική επανάληψη (γραμμές 8-13) γίνεται σύνδεση ενός ριζικού κόμβου σε έναν άλλο οπότε ο συνολικός χρόνος της εξωτερικής επανάληψης των γραμμών 4-17 θα είναι το πολύ $O(D(n) + (t(h))$ [1]. Καθώς $D(n) = lg(n)$ ο πραγματικός χρόνος της πράξης είναι $O(lg(n) + t(h))$ [1].

Πριν την πράξη Εξαγωγή Ελαχίστου Από Σωρό Fibonacci το δυναμικό είναι $t(h) + 2m(h)$ και μετά τη πράξη μικρότερο ή ίσο με $(D(n) + 1) + 2m(h)$ καθώς στο ριζικό επίπεδο θα έχουν απομείνει το πολύ $D(n+1)$ κόμβοι και επειδή κανένας κόμβος δε θα επισημανθεί [1]. Άρα ο λογιστικός χρόνος της πράξης είναι: [1]

$$O(D(n) + t(h)) + ((D(n) + 1) + 2m(h)) - (t(h) + 2m(h))$$
$$= O(D(n)) + O(t(h)) - t(h)$$
$$= O(D(n)) = O(lg(n)).$$

Επειδή για κάθε κόμβο του ριζικού επιπέδου που συνδέεται με έναν άλλο ο αριθμός των ριζικών κόμβων μειώνεται κατά ένα, η μείωση δυναμικού που προκαλείται καλύπτει το χρόνο που χρειάζεται η κάθε σύνδεση [1].

3.2.6 Μείωση κλειδιού

Για να μειωθεί το κλειδί ενός κόμβου σε ένα σωρό Fibonacci αρχικά ανατίθεται στον κόμβο αυτό το νέο κλειδί το οποίο θα πρέπει να είναι μικρότερο ή ίσο του υπάρχοντος κλειδιού [1]. Αυτό ενδέχεται να παραβιάσει την ιδιότητα σωρού ελαχίστου και αν συμβεί τότε αποκόπτεται η σύνδεση του κόμβου με το γονέα του μειώνοντας το βαθμό του, προστίθεται στο ριζικό επίπεδο και αν έχει μικρότερο κλειδί από του τρέχων ελάχιστου κόμβου τότε θα είναι ο νέος ελάχιστος κόμβος [1, 4]. Όταν ένας ριζικός κόμβος γίνεται παιδί ενός άλλου, με το που χάσει δύο από τα παιδιά του μέσω αποκοπών τότε θα αποκόπτεται και ο ίδιος από το γονέα του [1, 4]. Η διαδικασία αυτή ονομάζεται κλιμακωτή αποκοπή [1, 4].

Το πεδίο της σήμανσης χρησιμοποιείται για να ελέγχεται που πρέπει να γίνουν κλιμακωτές αποκοπές, όταν ένας ριζικός κόμβος γίνει παιδί ενός άλλου τότε η επισήμανσή του αφαιρείται, όταν αποκοπεί από το γονέα του και ο γονέας του δεν είναι ριζικός τότε ο γονέας του θα επισημαίνεται ενώ αν έχει ήδη επισημανθεί τότε θα πραγματοποιείται κλιμακωτή αποκοπή και θα αποκόπτεται και αυτός από το γονέα του [1, 4].

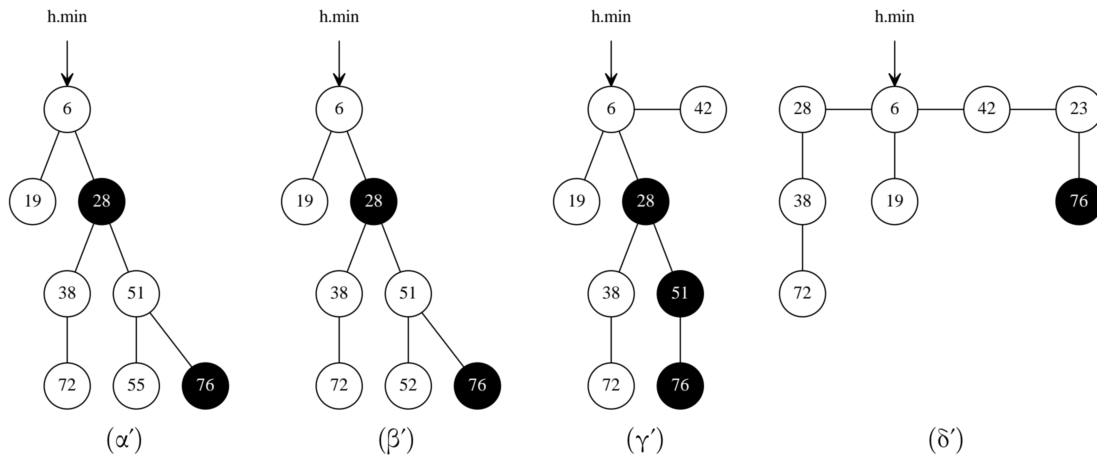
Μείωση Κλειδιού Σε Σωρό Fibonacci(h, x, k)
1. αν $k > x$.κλειδί
2. Σφάλμα ≪Το νέο κλειδί είναι μεγαλύτερο από το τρέχων≫
3. x.κλειδί $\leftarrow k$
4. $y \leftarrow x$.γονέας
5. αν $y \neq$ ΚΕΝΟ και x.κλειδί $< y$.κλειδί
6. Αποκοπή(h, x, y)
7. Κλιμακωτή Αποκοπή(h, y)
8. αν x.κλειδί $< h.min$.κλειδί
9. $h.min \leftarrow x$

Αποκοπή(h, x, y)
1. αν x.δεξιός $= x$
2. y.παιδί $\leftarrow$ ΚΕΝΟ
3. αλλιώς
4. y.παιδί $\leftarrow x$.δεξιός
5. Αφαίρεση Κόμβου(x)
6. y.βαθμός $\leftarrow y$.βαθμός -1
7. Προσθήκη Κόμβου$(h.min, x)$
8. x.γονέας $\leftarrow$ ΚΕΝΟ
9. x.σήμανση $\leftarrow$ ΨΕΥΔΕΣ

Κλιμακωτή Αποκοπή(h, y)
1. $z \leftarrow y$.γονέας
2. αν $z \neq$ ΚΕΝΟ
3. αν y.σήμανση = ΨΕΥΔΕΣ
4. y.σήμανση $\leftarrow$ ΑΛΗΘΕΣ
5. αλλιώς
6. Αποκοπή(h, y, z)
7. Κλιμακωτή Αποκοπή(h, z)

Στις γραμμές 1-3 της μείωση κλειδιού διασφαλίζεται ότι το νέο κλειδί είναι μικρότερο ή ίσο του παλιού και ανατίθεται στο κλειδί του x [1]. Αν ο x βρίσκεται στο ριζικό επίπεδο ή αν το κλειδί του είναι μεγαλύτερο ή ίσο από του γονέα του τότε δε θα χρειαστεί να γίνει κάποια αλλαγή στη δομή του σωρού καθώς δεν παραβιάζεται η ιδιότητα σωρού ελαχίστου (έλεγχος στις γραμμές 4-5) [1]. Αν όμως παραβιασθεί αυτή η ιδιότητα τότε στις γραμμές 6-7 θα γίνεται αποκοπή του κόμβου από το γονέα του και θα επιχειρείται να πραγματοποιηθεί κλιμακωτή αποκοπή [1]. Αφού ολοκληρωθούν τα παραπάνω τότε στις γραμμές 8-9 ελέγχεται αν το νέο κλειδί του x είναι μικρότερο από του ελάχιστου κόμβου ώστε αν χρειαστεί να γίνει ο νέος ελάχιστος [1].

Η πράξη αποκοπή αποσυνδέει τον κόμβο x από το γονέα του (y) κάνοντάς τον ριζικό κόμβο [1]. Στις γραμμές 1-4 ελέγχεται αν ο x είναι ο μοναδικός κόμβος στη λίστα παιδιών του y και αν ναι τότε ο y δε θα έχει πλέον κανένα παιδί αλλιώς ενημερώνεται ώστε να δείχνει σε κάποιο άλλο παιδί του αντί του x. Στις γραμμές 5-9 ο x αφαιρείται από τη λίστα παιδιών του y, ο βαθμός του y μειώνεται κατά ένα, ο x προστίθεται



Σχήμα 3.2.6.1: Μείωση κλειδιού σε σωρό Fibonacci

στο ριζικό επίπεδο οπότε ο δείκτης προς το γονέα του θα δείχνει προς το κενό και αφαιρείται η επισήμανση από αυτόν [1]. Η αφαίρεση του κόμβου και η προσθήκη του στο ριζικό επίπεδο γίνεται αντίστοιχα με τις πράξεις Αφαίρεση Κόμβου και Προσθήκη Κόμβου.

Στις γραμμές 1-2 της κλιμακωτής αποκοπής ελέγχεται αν ο y είναι ριζικός κόμβος και αν ναι τότε η διαδικασία απλά θα επιστρέφει [1]. Στις γραμμές 3-4 ελέγχεται αν ο κόμβος δεν έχει επισήμανση ώστε να επισημανθεί [1]. Αυτό σημαίνει πως έχασε ένα από τα παιδιά του [1]. Αλλιώς αν έχει ήδη επισήμανση τότε αποκόπτεται από το γονέα του μέσω της αποκοπής και η πράξη καλεί αναδρομικά τον εαυτό της (γραμμές 5-7) [1]. Με τον τρόπο αυτό ανέρχεται στο δένδρο μέχρι να συναντηθεί ένας ριζικός κόμβος ή ένας κόμβος που δε θα έχει επισήμανση [1].

Στο σχήμα 3.2.6.1 απεικονίζονται τρία παραδείγματα της μείωσης κλειδιού σε ένα σωρό Fibonacci. Στο (α′) ο αρχικός σωρός Fibonacci. Στο (β′) πραγματοποιήθηκε μείωση κλειδιού στον κόμβο με το κλειδί 55 και πλέον η τιμή του είναι 52. Καθώς δεν παραβιάστηκε η ιδιότητα σωρού ελαχίστου δε χρειάστηκε να γίνει κάποια δομική αλλαγή στο σωρό. Στο (γ′) μειώθηκε το κλειδί του ίδιου κόμβου στην τιμή 42, καθώς το κλειδί του είναι πλέον μικρότερο από του γονέα του ο κόμβος αποκόπτεται από αυτόν, η επισήμανσή του αφαιρείται και προστίθεται στο ριζικό επίπεδο. Καθώς ο γονέας του δεν είχε επισήμανση θα επισημανθεί αφού έχασε ένα από τα παιδιά του και δεν είναι ριζικός κόμβος. Στο (δ′) πραγματοποιήθηκε μείωση κλειδιού στον κόμβο με το κλειδί 51 και έχει πλέον το κλειδί με την τιμή 23. Επειδή το κλειδί του είναι μικρότερο από του γονέα του θα αποκοπεί από αυτόν και θα προστεθεί στο ριζικό επίπεδο χωρίς επισήμανση. Επειδή ο γονέας του είχε επισήμανση αποκόπηκε και εκείνος από το γονέα του, αφαιρέθηκε η επισήμανση του και προστέθηκε επίσης στο ριζικό επίπεδο. Ο δικός του γονέας ήταν ο κόμβος με την τιμή 6 και επειδή ήταν ριζικός δεν επισημάνθηκε.

Ο χρόνος που χρειάζεται για να πραγματοποιηθεί η πράξη της μείωσης κλειδιού σε

σωρό Fibonacci είναι $O(1)$ συν το χρόνο που χρειάζονται για να πραγματοποιηθούν οι κλιμακωτές αποκοπές [1]. Έστω ότι σε μια πράξη μείωσης κλειδιού η πράξη της κλιμακωτής αποκοπής εκτελείται αναδρομικά c φορές, αφού κάθε κλήση της απαιτεί χρόνο $O(1)$ τότε ο συνολικός χρόνος των αναδρομικών κλήσεων και συνεπώς της πράξης μείωσης κλειδιού είναι $O(c)$ [1].

Έστω h η κατάσταση του σωρού πριν κληθεί η πράξη Μείωση Κλειδιού Σε Σωρό Fibonacci [1]. Κάθε φορά που καλείται αναδρομικά η πράξη της κλιμακωτής αποκοπής εκτός από την τελευταία, αποκόπτεται ένας κόμβος με επισήμανση η οποία θα αφαιρεθεί από αυτόν, οπότε ο αριθμός των δένδρων αφού ολοκληρωθούν οι κλιμακωτές αποκοπές θα είναι $t(h) + c$ [1]. Ο αρχικός αριθμός δένδρων ήταν $t(h)$, οι κλιμακωτές αποκοπές πρόσθεσαν $c-1$ δένδρα ενώ ένα ακόμη (ο ριζικός κόμβος x) προστέθηκε από την πράξη της μείωσης κλειδιού όταν κλήθηκε η πράξη Αποκοπή [1]. Ο αριθμός των κόμβων με επισήμανση είναι το πολύ $m(h) - c + 2$ καθώς λόγω των κλιμακωτών αποκοπών αφαιρέθηκε η επισήμανση από $c-1$ κόμβους ενώ στην τελευταία της αναδρομική κλήση υπάρχει περίπτωση ένας να επισημάνθηκε [1]. Οπότε η μεταβολή του δυναμικού είναι το πολύ $((t(h)+c)+2(m(h)-c+2))-(t(h)+2m(h)) = 4-c$ [1]. Άρα ο λογιστικός χρόνος της πράξης είναι $O(c) + 4 - c = O(1)$ [1].

Επειδή όταν ένας κόμβος αποκόπτεται μέσω της κλιμακωτής αποκοπής αφαιρείται η επισήμανση από αυτόν, το δυναμικό μειώνεται κατά 2 [1]. Μια μονάδα δυναμικού χρησιμοποιείται για την κάλυψη της αποκοπής και αφαίρεσης της επισήμανσης του κόμβου και η άλλη επειδή όταν ο κόμβος γίνει ριζικός το δυναμικό θα αυξηθεί κατά 1 [1].

3.2.7 Διαγραφή

Για να διαγραφεί ένας κόμβος από ένα σωρό Fibonacci, θα μειώνεται το κλειδί του στην τιμή $-\infty$ χρησιμοποιώντας την πράξη της μείωσης κλειδιού και ο κόμβος θα αφαιρείται από το σωρό χρησιμοποιώντας την πράξη της εξαγωγής ελαχίστου [1]. Έπειτα ο κόμβος x μπορεί να αποδεσμευτεί. Η τιμή $-\infty$ θα πρέπει να είναι μοναδική στο σωρό [1].

Διαγραφή Από Σωρό Fibonacci*(h, x)*
1. Μείωση Κλειδιού Σε Σωρό Fibonacci*($h, x, -\infty$)*
2. Εξαγωγή Ελαχίστου Από Σωρό Fibonacci*(h)*

Ο λογιστικός χρόνος της πράξης είναι ίσος με το άθροισμα του λογιστικού χρόνου των πράξεων της μείωσης κλειδιού και εξαγωγής ελαχίστου, δηλαδή $O(1)+O(lg(n)) = O(lg(n))$ [1].

3.2.8 Κατασκευή

Για να κατασκευαστεί ένας σωρός Fibonacci από έναν πίνακα που περιέχει δείκτες κόμβων αρκεί να προστεθούν όλοι οι κόμβοι στο ριζικό επίπεδο ενός νέου σωρού Fibonacci και να βρεθεί ο ελάχιστος κόμβος.

Ο πίνακας και οι δείκτες που βρίσκονται στις θέσεις του δε θα τροποποιούνται, θα αλλάζουν μόνο τα πεδία των κόμβων στους οποίους δείχνουν.

Κατασκευή Σωρού Fibonacci*(A)*
1. $h \leftarrow$ Δημιουργία Σωρού Fibonacci()
2. αν A.μέγεθος $= 0$
3.   επιστροφή h
4. $h.min \leftarrow A[0]$
5. $h.n \leftarrow A$.μέγεθος
6. Αρχικοποίηση Κόμβου*(A[0])*
7. Για $i \leftarrow 1$ έως A.μέγεθος -1
8.   Αρχικοποίηση Κόμβου*(A[i])*
9. $A[i]$.αριστερός $\leftarrow A[i-1]$
10. $A[i-1]$.δεξιός $\leftarrow A[i]$
11.   αν $A[i]$.κλειδί $< h.min$.κλειδί
12. $h.min \leftarrow A[i]$
13. $A[i]$.δεξιός $\leftarrow A[0]$
14. $A[0]$.αριστερός $\leftarrow A[i]$
15. επιστροφή h

Στις γραμμές 1-3 δημιουργείται ένας νέος σωρός Fibonacci και ελέγχεται αν ο πίνακας είναι άδειος. Αν ναι τότε επιστρέφεται ένας κενός σωρός Fibonacci και η πράξη ολοκληρώνεται. Στις γραμμές 4-6 θεωρείται ότι το πρώτο στοιχείο του πίνακα είναι ο ελάχιστος κόμβος, εκχωρείται στον αριθμό κόμβων του σωρού το αριθμός των στοιχείων του πίνακα και αρχικοποιούνται τα πεδία του κόμβου. Η επανάληψη των γραμμών 7-12 ξεκινάει από τη δεύτερη θέση του πίνακα και τελειώνει αφού επεξεργαστεί και το στοιχείο που βρίσκεται στην τελευταία. Σε κάθε επανάληψη αρχικοποιούνται τα πεδία του τρέχων κόμβου (θέση i του πίνακα), ο τρέχων θα έχει ως αριστερό αδελφό τον κόμβο της προηγούμενης θέσης και εκείνος δεξιό αδελφό τον τρέχων. Έπειτα ελέγχεται αν το κλειδί του τρέχων κόμβου είναι μικρότερο από του ελάχιστου και αν ναι τότε είναι ο νέος ελάχιστος. Στις γραμμές 13-14 ολοκληρώνεται ο κύκλος της λίστας ριζικών κόμβων, ο τελευταίος έχει ως δεξιό αδελφό τον πρώτο και ο πρώτος αριστερό αδελφό τον τελευταίο. Στη γραμμή 15 επιστρέφεται ο σωρός Fibonacci.

Ο χρόνος της πράξης είναι ίδιος με το χρόνο που χρειάζεται για να κληθεί η πράξη Εισαγωγή Σε Σωρό Fibonacci n φορές καθώς όλοι οι κόμβοι του πίνακα απλώς προστίθονται στο ριζικό επίπεδο. Άρα ο πραγματικός και ο λογιστικός χρόνος της πράξης είναι $O(n)$.

3.3 Απόδειξη του μέγιστου βαθμού

Σε ένα σωρό Fibonacci με n κόμβους, ο μέγιστος βαθμός που μπορεί να έχει οποιοσδήποτε κόμβος είναι $D(n) = O(lg(n))$ [1].

Ο k-οστός αριθμός Fibonacci ορίζεται ως $F_0 = 0, F_1 = 1, F_k = F_{k-2} + F_{k-1}$ για $k \geq 2$ [1, 4]. Η χρυσή τομή ορίζεται ως $\varphi = (1 + \sqrt{5})/2$ και ισχύει ότι $F_{k+2} \geq \varphi^k$ [1, 4].

Λήμμα 1

Έστω x οποιοσδήποτε κόμβος σε ένα σωρό Fibonacci, και έστω $s(x)$ το μέγεθός του (ο αριθμός των κόμβων συμπεριλαμβανομένου του x στο υπόδενδρο με ρίζα τον x) [1]. Αν ο βαθμός του x είναι k τότε $s(x) \geq F_{k+2} \geq \varphi^k$.

Απόδειξη

Έστω $y_1, y_2, \ldots, y_k$ τα παιδιά του x με τη σειρά που συνδέθηκαν σε αυτόν από το πιο παλιό μέχρι το πιο πρόσφατο [1]. Τότε $\text{βαθμός}_{y1} \geq 0$ και για $i = 2, \ldots, k$ $\text{βαθμός}_{yi} \geq i - 2$ [1]. Αυτό ισχύει γιατί ακριβώς πριν συνδεθεί ο κόμβος y_i στον x, ο x είχε τουλάχιστον $i - 1$ παιδιά και για να συνδέθηκε στον x θα είχαν τον ίδιο βαθμό οπότε και ο y_i θα είχε τουλάχιστον $i - 1$ παιδιά [1, 4]. Μετά τη σύνδεση ο βαθμός του y_i θα μπορούσε να μειωθεί κατά το πολύ ένα χωρίς να αποκοπεί από τον x [1, 4]. Έστω s_k το ελάχιστο δυνατό μέγεθος ενός κόμβου x με βαθμό k, υποθέτοντας ότι $s_k = s(x)$ και ότι έχει παιδιά $y_1, \ldots y_k$ ισχύει όπως αποδείχθηκε ότι $_{y1} \geq 0$ και $_{yi} \geq i - 2$ [1]. Μέσω επαγωγής ισχύει ότι $s_k \geq F_{k+2}$ αφού $s_0 = 1, s_1 = 2$ [1, 4]. Άρα $s(x) = s_k = 1 + \sum_{i=1}^{k} s(y_i) \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i = F_{k+2}$ [1].

Πόρισμα 1

Ο μέγιστος βαθμός $D(n)$ για οποιονδήποτε κόμβο σε ένα σωρό Fibonacci με n κόμβους είναι $O(lg(n))$ [1].

Απόδειξη

Με βάση το λήμμα 1, για οποιονδήποτε κόμβο x με βαθμό k σε ένα σωρό Fibonacci με n κόμβους, $n \geq s(x) \geq \varphi^k$ [1]. Παίρνοντας το λογάριθμο ως προς βάση φ των δύο άκρων, $k \leq log_{\Phi} n$ και επειδή ο k είναι ακέραιος αριθμός $k \leq \lfloor log_{\Phi} n \rfloor$ [1]. Άρα ο μέγιστος βαθμός για οποιονδήποτε κόμβο είναι $O(lg(n))$ [1].

4 Ο Αλγόριθμος του Dijkstra

4.1 Περιγραφή

Υπάρχουν διάφορες παραλλαγές του αλγορίθμου, ο αυθεντικός αλγόριθμος βρίσκει τη συντομότερη διαδρομή μεταξύ δύο κόμβων σε ένα γράφο [3]. Συνήθως όμως είναι απαραίτητο να βρεθούν όλες οι συντομότερες διαδρομές σε ένα γράφο από έναν κόμβο αφετηρία προς όλους τους άλλους κόμβους [1, 2]. Ο αλγόριθμος του Dijkstra μπορεί να χρησιμοποιηθεί σε ένα γράφο υπό την προϋπόθεση ότι δεν υπάρχουν ακμές με αρνητικά βάρη [1, 2].

Στη συγκεκριμένη παραλλαγή του αλγορίθμου χρησιμοποιούνται δύο πίνακες, ο πίνακας αποστάσεων dist και ο πίνακας προκατόχων prev [2]. Για κάθε κόμβο u στο γράφο, στον πίνακα αποστάσεων αποθηκεύεται η απόσταση από τον κόμβο αφετηρία s προς τον κόμβο u και στον πίνακα προκατόχων ο κόμβος που βρίσκεται ακριβώς πριν από τον u στη συντομότερη διαδρομή από τον κόμβο s προς τον κόμβο u [2]. Επίσης χρησιμοποιείται μια ουρά προτεραιότητας που υποστηρίζει τις πράξεις Κατασκευή Ουράς Προτεραιότητας, Εξαγωγή Ελαχίστου και Μείωση Κλειδιού [2]. Ακολουθεί ο αλγόριθμος σε μορφή ψευδοκώδικα [2].

```
Dijkstra(G, w, s)
1.  για κάθε κόμβο u ϵ V
2.      dist[u] ← ∞
3.      prev[u] ← ΚΕΝΟ
4.  dist[s] ← 0
5.  Q ← Κατασκευή Ουράς Προτεραιότητας(V) (με κλειδιά τις τιμές του dist)
6.  όσο η Q δεν είναι κενή
7.      u ← Εξαγωγή Ελαχίστου(Q)
8.      για κάθε ακμή (u, v) ϵ E
9.      αν dist[v] > dist[u] + w(u, v)
10.         dist[v] ← dist[u] + w(u, v)
11.         prev[v] ← u
12.         Μείωση Κλειδιού(Q, v,dist[v])
```

Στις γραμμές 1-4 πραγματοποιείται αρχικοποίηση των τιμών στους πίνακες dist και prev. Καθώς δεν έχουν υπολογιστεί οι αποστάσεις ούτε οι προκάτοχοι κάθε τιμή στον πίνακα dist θα έχει την τιμή ∞ και κάθε τιμή στον πίνακα *prev* τιμή ΚΕΝΟ. Καθώς το σημείο εκκίνησης είναι ο κόμβος αφετηρία s, στη θέση του στον πίνακα dist απο-

θηκεύεται η απόσταση 0. Στη γραμμή 5 δημιουργείται μια ουρά προτεραιότητας Q η οποία θα περιέχει όλους τους κόμβους του γράφου ως αντικείμενα και ως κλειδιά (προτεραιότητα) για κάθε έναν από τους κόμβους τις τιμές του πίνακα αποστάσεων. Στην επανάληψη των γραμμών 6-12 εξάγεται κάθε φορά ο κόμβος u που έχει τη μικρότερη απόσταση. Για κάθε κόμβο v στο γράφο που υπάρχει ακμή από τον u προς τον v ελέγχεται αν η τιμή του v στον πίνακα αποστάσεων είναι μικρότερη από αυτής του u συν το βάρος w της ακμής που τους ενώνει. Αν ναι τότε έχει μικρότερη απόσταση και ενημερώνεται η τιμή στη θέση του v στον πίνακα αποστάσεων με τη νέα τιμή και στην ίδια θέση του πίνακα προκατόχων ανατίθεται ο δείκτης προς τον κόμβο u. Επίσης ενημερώνεται για τη νέα απόσταση και η ουρά προτεραιότητας μέσω της πράξης της μείωσης κλειδιού.

Μετά την ολοκλήρωση του αλγορίθμου μπορούν να ακολουθηθούν οι δείκτες του πίνακα prev προς τα πίσω για να ανακατασκευαστεί η συντομότερη διαδρομή [2]. Για να εκτυπωθεί η διαδρομή από τον κόμβο s προς έναν κόμβο v μπορεί να χρησιμοποιηθεί η παρακάτω διαδικασία [1].

Εκτύπωση Διαδρομής*(*prev, s, v*)*
1. αν $s = v$
2.   Εκτύπωση s
3. αλλιώς
4.   αν prev[v] = KENO
5.   Εκτύπωση «Δεν υπάρχει διαδρομή από» s «προς» v
6.   αλλιώς
7.   Εκτύπωση Διαδρομής(prev , s, prev[v]
8.   Εκτύπωση v

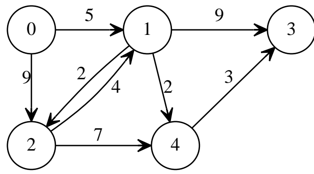
Αν οι κόμβοι του γράφου είναι αποθηκευμένοι σε έναν πίνακα τότε στον πίνακα prev μπορούν να αποθηκεύονται οι θέσεις που βρίσκεται κάθε κόμβος στον πίνακα κόμβων αντί για δείκτες προς τους ίδιους τους κόμβους. Σε αυτήν την περίπτωση αντί να ανατίθεται η τιμή KENO μπορεί να ανατεθεί η τιμή ∞ όταν κάποιος κόμβος δεν έχει προκάτοχο.

4.2 Λειτουργία

Στο σχήμα 4.2.1 απεικονίζεται μια πλήρη εκτέλεση του αλγορίθμου του Dijkstra σε ένα γράφο με τον κόμβο 0 ως αφετηρία. Σε κάθε σχήμα απεικονίζονται οι καταστάσεις των πινάκων dist και prev και της ουράς προτεραιότητας. Η ουρά προτεραιότητας θα εμφανίζεται με τη μορφή συνόλου και κάθε στοιχείο θα είναι ένα ζεύγος της μορφής *(*αντικείμενο, κλειδί*)*. Στο (α′) έχουν αρχικοποιηθεί οι πίνακες και έχει κατασκευαστεί η ουρά προτεραιότητας. Στα (β′)-(ε′) οι καταστάσεις των πινάκων dist και prev και της ουράς μετά το τέλος κάθε επανάληψης των γραμμών 6-12 του αλγορίθμου. Ο

κόμβος με το χρώμα γκρι είναι αυτός που εξάχθηκε από την ουρά και οι ακμές με έντονες γραμμές δείχνουν τους κόμβους των οποίων η απόσταση θα ελεγχθεί. Στο (α΄) εξάχθηκε ο κόμβος 0 καθώς είχε το μικρότερο κλειδί (0). Οι κόμβοι που γειτνιάζουν με αυτόν είναι ο 1 και 2. Επειδή οι τιμές τους στον πίνακα dist είναι ∞ ο έλεγχος της γραμμής 9 θα είναι αληθής και θα εκχωρηθεί στις θέσεις αυτές η απόσταση τους από τον 0. Για τον 1 είναι $0+5=5$ και για τον 2 είναι $0+9=9$. Οι κόμβοι αυτοί θα έχουν πλέον ως προκάτοχο τον 0 στις αντίστοιχες θέσεις του πίνακα prev και τα κλειδιά τους στην ουρά προτεραιότητας θα ενημερωθούν στις νέες τιμές 5 και 9 αντίστοιχα. Στο (γ΄) εξάχθηκε ο κόμβος 1. Η περίπτωση αυτή είναι ίδια με την προηγούμενη με τους κόμβους 3 και 4. Για τον 3 η τιμή στον dist μειώνεται από ∞ σε $5+9=14$ και για τον 4 επίσης από ∞ σε $5+2=7$ και οι κόμβοι θα έχουν ως προκάτοχο τον 1. Στο (δ΄) υπήρχαν δύο κόμβοι που είχαν τη μικρότερη τιμή (7) στην ουρά προτεραιότητας, ο 2 και ο 4. Δεν έχει σημασία ποιος από τους δύο θα εξαχθεί πρώτος, οι διαδρομές που θα βρεθούν θα εξακολουθούν να είναι οι συντομότερες μετά την ολοκλήρωση του αλγορίθμου. Έστω ότι εξάχθηκε πρώτα ο κόμβος 2. Οι κόμβοι που γειτνιάζουν με αυτόν είναι ο 1 και ο 4. Η απόσταση που έχει ήδη υπολογιστεί για τον 1 (5) δεν είναι μεγαλύτερη από αυτήν που θα είχε με τον 2 $(9+4=13)$ οπότε δε θα γίνει κάποια αλλαγή. Ομοίως και για τον 4 η απόσταση 7 δεν είναι μεγαλύτερη της $9+7=16$. Στο (ε΄) εξάχθηκε ο κόμβος 4. Ο μοναδικός κόμβος που γειτνιάζει με αυτόν είναι ο 3 και η ήδη υπολογισμένη απόσταση είναι μεγαλύτερη $(14>7+3)$ οπότε θα μειωθεί σε 10 και θα έχει ως προκάτοχο τον 4. Στην επόμενη επανάληψη θα εξαχθεί ο κόμβος 3, επειδή όμως δεν υπάρχει κάποιος κόμβος που να γειτνιάζει με αυτόν δε θα πραγματοποιηθεί κάποια αλλαγή, οπότε η απεικόνιση παραλείπεται.

Καλώντας τη διαδικασία Εκτύπωση κόμβου στον πίνακα προκατόχων του (ε΄) με κόμβο αφετηρία τον 0 και προορισμό οποιονδήποτε άλλο κόμβο μπορεί να εκτυπωθεί η συντομότερη διαδρομή προς αυτόν. Έστω ότι καλείται για τον κόμβο προορισμού 4. Ο 4 έχει ως προκάτοχο τον 1 οπότε η διαδικασία καλείται αναδρομικά σε αυτόν (γραμμή 7), ο 1 έχει ως προκάτοχο τον 0 οπότε πραγματοποιείται δεύτερη αναδρομική κλήση της. Ο 0 είναι ο κόμβος αφετηρία οπότε θα εκτυπωθεί ολοκληρώνοντας τη δεύτερη αναδρομή. Επιστρέφοντας στην πρώτη εκτυπώνεται ο κόμβος 1 και επιστρέφοντας στην αρχική κλήση εκτυπώνεται ο κόμβος 4, οπότε το αποτέλεσμα θα είναι 0 1 4, η συντομότερη διαδρομή από τον κόμβο 0 προς τον 4.



dist→

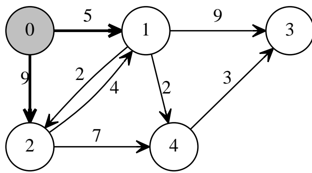
0	1	2	3	4
0	∞	∞	∞	∞

$Q = \{(0,0),(1,\infty),(2,\infty),(3,\infty),(4,\infty)\}$

prev→

0	1	2	3	4
K	K	K	K	K

(α′)



dist→

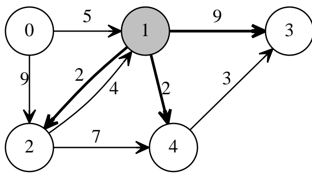
0	1	2	3	4
0	5	9	∞	∞

$Q = \{(1,5),(2,9),(3,\infty),(4,\infty)\}$

prev→

0	1	2	3	4
K	0	0	K	K

(β′)



dist→

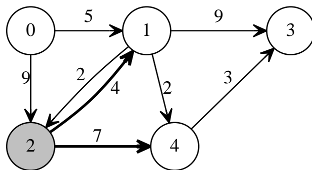
0	1	2	3	4
0	5	7	14	7

$Q = \{(2,7),(3,14),(4,7)\}$

prev→

0	1	2	3	4
K	0	1	1	1

(γ′)



dist→

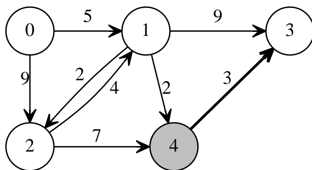
0	1	2	3	4
0	5	7	14	7

$Q = \{(3,14),(4,7)\}$

prev→

0	1	2	3	4
K	0	1	1	1

(δ′)



dist→

0	1	2	3	4
0	5	7	10	7

$Q = \{(3,10)\}$

prev→

0	1	2	3	4
K	0	1	4	1

(ε′)

Σχήμα 4.2.1: Εκτέλεση του αλγορίθμου του Dijkstra

4.3 Ανάλυση πολυπλοκότητας

Ο χρόνος εκτέλεσης του αλγορίθμου του Dijkstra εξαρτάται από την ουρά προτεραιότητας που θα χρησιμοποιηθεί [1, 2]. Εφόσον θα πραγματοποιηθούν $|V|$ εισαγωγές στην ουρά προτεραιότητας, χρειάζονται $|V|$ εξαγωγές και το πολύ $V + E$ πράξεις μείωσης κλειδιού [2]. Οπότε σε αφαιρετικό επίπεδο ο χρόνος εκτέλεσης του αλγορίθμου του Dijkstra είναι $O(|V| \times$ Εξαγωγή Ελαχίστου $+(|V|+|E|)\times$ Μείωση Κλειδιού) [2].

4.3.1 Υλοποίηση με σωρό Fibonacci

Για να χρησιμοποιηθεί ως ουρά προτεραιότητας ο σωρός Fibonacci θα πρέπει να είναι υπάρχει ένας δείκτης προς οποιονδήποτε κόμβο του σωρού ώστε να μπορεί να κληθεί σε αυτόν η πράξη της μείωσης κλειδιού. Αρκεί να χρησιμοποιηθεί ένας πίνακας δεικτών A για κόμβους σωρών Fibonacci και να αποθηκεύεται σε αυτόν κάθε κόμβος του σωρού κατά τη δημιουργία του. Οι κόμβοι μπορούν να δημιουργούνται και να εκχωρείται ένας δείκτης προς κάθε έναν από αυτούς κατά την επανάληψη που αρχικοποιεί τις τιμές των πινάκων dist και prev στον αλγόριθμο του Dijkstra (γραμμές 1-3). Για κάθε κόμβο του σωρού x θα αποθηκεύεται στο πεδίο x.αντικείμενο κάθε κόμβος του γράφου u και στο πεδίο x.κλειδί η τιμή ∞. Στη γραμμή 4 θα εκχωρείται και στο κλειδί του κόμβου της θέσης s του πίνακα A η τιμή 0. Στη γραμμή 5 αρκεί να κληθεί η πράξη Κατασκευή Σωρού Fibonacci με τον πίνακα A. Στη γραμμή 7 θα αποθηκεύεται στον u το αντικείμενο του κόμβου του σωρού που εξάχθηκε με την πράξη Εξαγωγή Ελαχίστου Από Σωρό Fibonacci. Στη γραμμή 12 αρκεί να κληθεί η πράξη Μείωση κλειδιού Σε Σωρό Fibonacci για τον κόμβο $A[v]$ με το κλειδί dist$[v]$.

Ο χρόνος που χρειάζεται για να κατασκευαστεί ένας Σωρός Fibonacci από τον πίνακα δεικτών A είναι $O(|V|)$. Έπειτα οποιοσδήποτε κόμβος στο σωρό θα είναι προσβάσιμος σε χρόνο $O(1)$. Αντικαθιστώντας το χρόνο εκτέλεσης των πράξεων Εξαγωγή Ελαχίστου Από Σωρό Fibonacci και Μείωση Κλειδιού Σε Σωρό Fibonacci στο χρόνο εκτέλεσης του αλγορίθμου του Dijkstra με ουρά προτεραιότητας, προκύπτει ο χρόνος εκτέλεσης του αλγορίθμου. Κάθε εξαγωγή ελαχίστου χρειάζεται λογιστικό χρόνο $O(lg(|V|))$ και κάθε μείωση κλειδιού λογιστικό χρόνο $O(1)$ άρα ο χρόνος εκτέλεσης του αλγορίθμου του Dijkstra με σωρούς Fibonacci είναι $O(|V|\, lg(|V|)+|E|)$ [1, 2, 4].

4.3.2 Υλοποίηση με πίνακα και με λίστα

Οι προσθήκη όλων των κόμβων του γράφου σε έναν πίνακα χρειάζεται χρόνο $O(|V|)$, για να πραγματοποιηθεί η πράξη της μείωσης κλειδιού αρκεί απλώς να εκχωρηθεί η νέα τιμή κλειδιού οπότε ο χρόνος που χρειάζεται η πράξη είναι $O(1)$ [1, 2]. Για να βρεθεί ο ελάχιστος κόμβος ώστε να εξαχθεί θα πρέπει να ελεγχθούν όλα τα στοιχεία του πίνακα, οπότε ο χρόνος της πράξης αυτής είναι $O(n)$ [1, 2]. Συνεπώς ο χρόνος εκτέλεσης του

αλγορίθμου του Dijkstra χρησιμοποιώντας ως ουρά προτεραιότητας έναν πίνακα είναι $O(|V|^2)$ [1, 2]. Τον ίδιο χρόνο εκτέλεσης θα έχει αν χρησιμοποιηθεί μια λίστα ως ουρά προτεραιότητας καθώς η μεθοδολογία για την πραγματοποίηση των πράξεων αυτών είναι ίδια [2].

4.3.3 Υλοποίηση με δυαδικό σωρό ελαχίστου

Για να πραγματοποιηθεί η πράξη Εξαγωγή Ελαχίστου.σε ένα δυαδικό σωρό, αντιμετατίθενται ο τελευταίος του κόμβος του σωρού με τον ριζικό ο οποίος θα επιστραφεί και ελέγχεται αν παραβιάστηκε η ιδιότητα σωρού ελαχίστου στη νέα ρίζα [1, 2]. Η διαδικασία αυτή ονομάζεται Αποκατάσταση Σωρού Ελαχίστου [1]. Αρχικά ελέγχονται οι τιμές των παιδιών του, αν κάποιο έχει μικρότερη τιμή από αυτόν τότε η ιδιότητα έχει παραβιαστεί και ο κόμβος θα αντιμετατεθεί με το παιδί του το οποίο έχει τη μικρότερη τιμή [1, 2]. Η διαδικασία αυτή θα επαναλαμβάνεται σε αυτόν τον κόμβο μέχρι να μην παραβιάζει την ιδιότητα σωρού ελαχίστου [1, 2]. Καθώς το ύψος ενός δυαδικού σωρού είναι $lg(n)$ ο χρόνος της πράξης της εξαγωγής ελαχίστου είναι $O(lg(n))$ [1, 2]. Ένας δυαδικός σωρός ελαχίστου μπορεί να κατασκευαστεί από έναν πίνακα σε χρόνο $O(n)$, για να πραγματοποιηθεί αυτό αρκεί να κληθεί η διαδικασία της αποκατάστασης από το τέλος του σωρού προς την αρχή ξεκινώντας από το προ τελευταίο επίπεδο [1]. Για να πραγματοποιηθεί η πράξη της μείωσης κλειδιού θα ανατίθεται στο κόμβο η νέα τιμή και θα καλείται η διαδικασία της αποκατάστασης κάθε φορά στο γονέα του [1, 2]. Ο χρόνος της πράξης είναι $O(lg(n))$ [1, 2].

Ο χρόνος εκτέλεσης του αλγορίθμου του Dijkstra με δυαδικό σωρό ελαχίστου είναι $O((|V| + |E|)lg(|V|))$ [1, 2].

5 Υλοποίηση στη γλώσσα C

5.1 Γράφος

5.1.1 Πίνακας γειτνίασης

Στο αρχείο adj_matrix.h δηλώνεται ο τύπος του πίνακα γειτνίασης: adj_matrix ο ο- ποίος αποτελείται από έναν δισδιάστατο πίνακα ακεραίων: adj που θα δημιουργείται δυναμικά και τον αριθμό των γραμμών και στηλών του: n ο οποίος είναι και ο αριθμός των κορυφών του γράφου. Επίσης δηλώνεται η συνάρτηση φόρτωσης πίνακα γειτνίασης: adj_matrix_load από αρχείο του οποίου το όνομα θα περνιέται ως παράμετρος. Η συνάρτηση αυτή δημιουργεί έναν πίνακα γειτνίασης διαβάζοντας τα περιεχόμενα του αρχείου και επιστρέφει ένα δείκτη σε αυτόν. Σε οποιοδήποτε σφάλμα θα εμφανίζεται κατάλληλο μήνυμα και το πρόγραμμα θα τερματίζει. Η συνάρτηση adj_matrix_destroy δέχεται ως παράμετρο έναν πίνακα γειτνίασης και απελευθερώνει τη μνήμη που δεσμεύεται από αυτόν και τα μέλη του.

Το πρώτο στοιχείο του αρχείου εισόδου θα είναι ο αριθμός των κορυφών του γράφου και έπειτα θα γράφεται ο πίνακας γειτνίασης. Κάθε στοιχείο των στηλών μπορεί να χωρίζεται με ένα ή περισσότερα κενά και κάθε γραμμή του με νέα γραμμή.

adj_matrix.h

```
#ifndef ADJ_MATRIX_H_
#define ADJ_MATRIX_H_

typedef struct adj_matrix
{
    int **adj;
    int n;
}adj_matrix;

adj_matrix *adj_matrix_load(char *filename);
void adj_matrix_destroy(adj_matrix *graph);

#endif
```

Στο αρχείο adj_matrix.c υλοποιούνται οι συναρτήσεις φόρτωσης και καταστροφής πίνακα γειτνίασης που δηλώθηκαν στο αρχείο adj_matrix.h.

adj_matrix.c

```
#include "adj_matrix.h"
#include <stdio.h>
#include <stdlib.h>

adj_matrix *adj_matrix_load(char *filename)
{
    adj_matrix *graph;
    FILE *fp;
    int u, v;
    graph = malloc(sizeof(adj_matrix));
    if (graph == NULL)
    {
        perror("Memory allocation error");
        exit(EXIT_FAILURE);
    }
    fp = fopen(filename, "r");
    if (fp == NULL)
    {
        perror("Error opening input file");
        exit(EXIT_FAILURE);
    }
    if (fscanf(fp, "%d", &graph->n) != 1)
    {
        fprintf(stderr, "Input file error\n");
        fprintf(stderr, "Expected number of vertices\n");
        exit(EXIT_FAILURE);
    }
    if (graph->n <= 0)
    {
        fprintf(stderr, "Input file error\n");
        fprintf(stderr, "Number of vertices cannot be 0 or less");
        exit(EXIT_FAILURE);
    }
    graph->adj = malloc(graph->n * sizeof(int *));
    if (graph->adj == NULL)
```

```
    {
        perror("Memory allocation error");
        exit(EXIT_FAILURE);
    }
    for (u = 0; u < graph->n; u++)
    {
        graph->adj[u] = malloc(graph->n * sizeof(int));
        if (graph->adj[u] == NULL)
        {
            perror("Memory allocation error");
            exit(EXIT_FAILURE);
        }
    }
    for (u = 0; u < graph->n; u++)
        for (v = 0; v < graph->n; v++)
            if (fscanf(fp, "%d", &graph->adj[u][v]) != 1)
            {
                fprintf(stderr, "Input file error\n");
                fprintf(stderr, "Expected entries: %d\n", graph->n *
                    graph->n);
                fprintf(stderr, "Entries missing: %d\n", graph->n *
                    graph->n - (u * graph->n + v));
                exit(EXIT_FAILURE);
            }
            else if (graph->adj[u][v] < 0)
            {
                fprintf(stderr, "Input file error\n");
                fprintf(stderr, "Negative edge lengths are not
                    allowed\n");
                exit(EXIT_FAILURE);
            }
    return(graph);
}

void adj_matrix_destroy(adj_matrix *graph)
{
    int u;
    for (u = 0; u < graph->n; u++)
        free(graph->adj[u]);
```

```
    free(graph->adj);
    free(graph);
}
```

5.1.2 Λίστες γειτνίασης

Στο αρχείο adj_lists.h δηλώνεται ο τύπος κόμβου για λίστα γειτνίασης: adj_list_node και ο τύπος των λιστών γειτνίασης: adj_lists ο οποίος αποτελείται από έναν πίνακα δεικτών για κόμβους λίστας γειτνίασης: adj που θα δημιουργείται δυναμικά και τον αριθμό των γραμμών του: n ο οποίος είναι και ο αριθμός των κορυφών του γράφου. Κάθε γραμμή του πίνακα adj θα αντιστοιχεί σε μια κορυφή του γράφου και για κάθε μία από αυτές θα αποθηκεύεται μια λίστα κόμβων λίστας γειτνίασης με τις κορυφές που γειτνιάζουν με αυτήν. Κάθε κόμβος λίστας γειτνίασης: adj_list_node αποτελείται από μια κορυφή του γράφου, το βάρος της ακμής και έναν δείκτη next ώστε να αποθηκεύεται ο επόμενος κόμβος της λίστας. Επίσης δηλώνεται η συνάρτηση φόρτωσης λιστών γειτνίασης: adj_lists_load από αρχείο του οποίου το όνομα θα περνιέται ως παράμετρος και η συνάρτηση adj_matrix_destroy που δέχεται ένα γράφο σε μορφή λιστών γειτνίασης και απελευθερώνει τη μνήμη που δεσμεύεται από αυτές και τα μέλη τους. Σε οποιοδήποτε σφάλμα κατά τη φόρτωση του αρχείου θα εμφανίζεται κατάλληλο μήνυμα και το πρόγραμμα θα τερματίζει.

Το πρώτο στοιχείο του αρχείου εισόδου θα είναι ο αριθμός των κορυφών του γράφου, έπειτα για κάθε κορυφή του θα γράφεται η κορυφή αυτή και ζεύγη της μορφής κορυφή,βάρος για τις κορυφές που γειτνιάζουν με αυτήν. Οι κορυφές και τα ζεύγη μπορούν να χωρίζονται με ένα ή περισσότερα κενά αλλά και με νέες γραμμές

adj_lists.h

```
#ifndef ADJ_LISTS_H_
#define ADJ_LISTS_H_

typedef struct adj_list_node
{
    int vertex;
    int weight;
    struct adj_list_node *next;
}adj_list_node;

typedef struct adj_lists
{
    struct adj_list_node **adj;
```

```
    int n;
}adj_lists;

adj_lists *adj_lists_load(char *filename);
void adj_lists_destroy(adj_lists *graph);

#endif
```

Στο αρχείο adj_lists.c υλοποιούνται οι συναρτήσεις φόρτωσης και καταστροφής λιστών γειτνίασης που δηλώθηκαν στο αρχείο adj_lists.h.

adj_lists.c

```
#include "adj_lists.h"
#include <stdio.h>
#include <stdlib.h>

adj_lists *adj_lists_load(char *filename)
{
    adj_lists *graph;
    adj_list_node *tmp;
    FILE *fp;
    int u, v, l, n;
    graph = malloc(sizeof(adj_lists));
    if (graph == NULL)
    {
        perror("Memory allocation error");
        exit(EXIT_FAILURE);
    }
    fp = fopen(filename, "r");
    if (fp == NULL)
    {
        perror("Error opening input file");
        exit(EXIT_FAILURE);
    }
    if (fscanf(fp, "%d", &graph->n) != 1)
    {
        fprintf(stderr, "Input file error\n");
        fprintf(stderr, "Expected the number of vertices\n");
        exit(EXIT_FAILURE);
    }
```

```
if (graph->n <= 0)
{
    fprintf(stderr, "Input file error\n");
    fprintf(stderr, "Number of vertices cannot be 0 or less\n");
    exit(EXIT_FAILURE);
}
graph->adj = malloc(graph->n * sizeof(adj_list_node));
if (graph->adj == NULL)
{
    perror("Memory allocation error");
    exit(EXIT_FAILURE);
}
for (u = 0; u < graph->n; u++)
    graph->adj[u] = NULL;
u = 0;
while ((n = fscanf(fp, "%d , %d", &v, &l)) != EOF)
{
    if (n == 0)
    {
        fprintf(stderr, "Input file error\n");
        fprintf(stderr, "Invalid file format\n");
        exit(EXIT_FAILURE);
    }
    if (v < 0 || v > graph->n)
    {
        fprintf(stderr, "Input file error\n");
        fprintf(stderr, "Vertex %d does not exist in this
           graph\n", v);
        fprintf(stderr, "Range of existing vertices: %d - %d\n",
           0, graph->n - 1);
        exit(EXIT_FAILURE);
    }
    if (n == 1)
        u = v;
    else
    {
        if (l < 0)
        {
            fprintf(stderr, "Input file error\n");
```

```
            fprintf(stderr, "Negative edge lengths are not
               allowed\n");
            exit(EXIT_FAILURE);
         }
         tmp = malloc(sizeof(adj_list_node));
         if (tmp == NULL)
         {
            perror("Memory allocation error");
            exit(EXIT_FAILURE);
         }
         tmp->next = graph->adj[u];
         tmp->vertex = v;
         tmp->weight = l;
         graph->adj[u] = tmp;
      }
   }
   return(graph);
}

void adj_lists_destroy(adj_lists *graph)
{
   adj_list_node *v, *tmp;
   int u;
   for(u = 0; u < graph->n; u++)
   {
      v = graph->adj[u];
      while(v != NULL)
      {
         tmp = v->next;
         free(v);
         v = tmp;
      }
   }
   free(graph->adj);
   free(graph);
}
```

5.2 Σωρός Fibonacci

Ο σωρός Fibonacci υλοποιείται στη γλώσσα C με βάση το κεφάλαιο 3.

Στο αρχείο fheap.h δηλώνεται ο τύπος κόμβου για σωρούς Fibonacci: fheap_node, ο τύπος των ίδιων των σωρών: fheap και οι πράξεις που υποστηρίζονται από αυτούς.

Η δομή fheap_node αποτελείται από τα εξής πεδία: ένα δείκτη προς τον κόμβο-γονέα του: parent, ένα δείκτη προς κάποιον από τους κόμβους-παιδιά του: child, ένα δείκτη προς τον κόμβο που βρίσκεται αριστερά από αυτόν: left, ένα δείκτη προς τον κόμβο που βρίσκεται δεξιά από αυτόν: right, έναν ακέραιο αριθμό: key ο οποίος θα δηλώνει την προτεραιότητά του, έναν ακέραιο αριθμό: item που είναι το αντικείμενο που θα αποθηκεύεται από τον κόμβο, έναν θετικό ακέραιο αριθμό 7 bit που δηλώνει το βαθμό του: degree και έναν θετικό ακέραιο αριθμό 1 bit που δηλώνει αν ο κόμβος είναι επισημασμένος ή όχι: mark. Ο λόγος που τα μέλη κλειδί: key και αντικείμενο: item ορίζονται ως ακέραιοι αριθμοί είναι ότι θα χρησιμοποιηθούν στον αλγόριθμο του Dijkstra για να αποθηκεύσουν τις τιμές του πίνακα αποστάσεων ως κλειδιά και τις κορυφές του γράφου ως αντικείμενα οι οποίες θα αναπαριστώνται επίσης ως ακέραιοι αριθμοί.

Η δομή fheap αποτελείται από έναν δείκτη στον κόμβο με το μικρότερο κλειδί: min και έναν ακέραιο αριθμό: n που δηλώνει τον αριθμό των κόμβων που βρίσκονται στο σωρό.

Οι συναρτήσεις σωρού έχουν τη συμπεριφορά που αναφέρεται στο κεφάλαιο 3.

Η συνάρτηση δημιουργίας κόμβου: fheap_node_create_node δε δέχεται παραμέτρους, δεσμεύει μνήμη για έναν κόμβο και επιστρέφει ένας δείκτης προς αυτόν. Στην συνάρτηση αυτή καθώς και στην fheap_make_heap, αν υπάρξει σφάλμα δέσμευσης μνήμης θα εμφανίζεται κατάλληλο μήνυμα και το πρόγραμμα θα τερματίζει.

Αν στη πράξη μείωσης κλειδιού fheap_decrease_key δοθεί κλειδί μεγαλύτερο από το ήδη υπάρχων θα εμφανίζεται μήνυμα σφάλματος και η διαδικασία θα επιστρέφει χωρίς να κάνει τίποτα.

fheap.h

```
#ifndef FHEAP_H_
#define FHEAP_H_

typedef struct fheap_node
{
    struct fheap_node *parent;
    struct fheap_node *child;
    struct fheap_node *left;
    struct fheap_node *right;
    int key;
```

```
    int item;
    unsigned int degree : 7;
    unsigned int mark : 1;
}fheap_node;

typedef struct fheap
{
    struct fheap_node *min;
    int n;
}fheap;

fheap_node *fheap_node_create_node(void);

fheap *fheap_make_heap(void);
fheap *fheap_build_heap(fheap_node **A, int n);
void fheap_insert(fheap *h, fheap_node *x);
fheap_node *fheap_minimum(fheap *h);
fheap_node *fheap_extract_min(fheap *h);
fheap *fheap_meld(fheap *h1, fheap *h2);
void fheap_decrease_key(fheap *h, fheap_node *x, int k);
void fheap_delete(fheap *h, fheap_node *x);

#endif
```

Στο αρχείο fheap.c υλοποιούνται οι συναρτήσεις που δηλώθηκαν στο αρχείο fheap.h.

Επίσης δηλώνονται και υλοποιούνται μόνο για χρήση από το συγκεκριμένο αρχείο οι παρακάτω συναρτήσεις:

fheap_node_init, fheap_node_add, fheap_node_remove, fheap_list_concat, fheap_consolidate, fheap_link, fheap_cut και fheap_cascading_cut.

Οι παραπάνω συναρτήσεις είναι βοηθητικές για τις πράξεις των σωρών Fibonacci, για το λόγο αυτό περιορίζονται στο αρχείο υλοποίησης. Η λειτουργία τους αναφέρεται στο κεφάλαιο 3.

Στην πράξη της διαγραφής: fheap_delete για να μη δεσμευτεί κάποια τιμή ως $-\infty$ καλούνται απευθείας οι απαραίτητες πράξεις που θα πραγματοποιούσε η πράξη της μείωσης κλειδιού.

fheap.c

```
#include "fheap.h"
#include <math.h>
#include <stdio.h>
```

```
#include <stdlib.h>

static void fheap_node_init(fheap_node *x);
static void fheap_node_add(fheap_node *x, fheap_node *y);
static void fheap_node_remove(fheap_node *x);
static void fheap_list_concat(fheap_node *x, fheap_node *y);
static void fheap_consolidate(fheap *h);
static void fheap_link(fheap_node *y, fheap_node *x);
static void fheap_cut(fheap *h, fheap_node *x, fheap_node *y);
static void fheap_cascading_cut(fheap *h, fheap_node *y);

fheap_node *fheap_node_create_node(void)
{
    fheap_node *x;
    x = malloc(sizeof(fheap_node));
    if (x == NULL)
    {
        perror("Memory allocation error");
        exit(EXIT_FAILURE);
    }
    return(x);
}

static void fheap_node_init(fheap_node *x)
{
    x->degree = 0;
    x->parent = NULL;
    x->child = NULL;
    x->mark = 0;
}

static void fheap_node_add(fheap_node *x, fheap_node *y)
{
    x->left->right = y;
    y->left = x->left;
    x->left = y;
    y->right = x;
}
```

```
static void fheap_node_remove(fheap_node *x)
{
    x->left->right = x->right;
    x->right->left = x->left;
}

static void fheap_list_concat(fheap_node *x, fheap_node *y)
{
    fheap_node *t1, *t2;
    t1 = x->left;
    t2 = y->left;
    t1->right = y;
    y->left = t1;
    t2->right = x;
    x->left = t2;
}

fheap *fheap_make_heap(void)
{
    fheap *h;
    h = malloc(sizeof(fheap));
    if (h == NULL)
    {
        perror("Memory allocation error");
        exit(EXIT_FAILURE);
    }
    h->min = NULL;
    h->n = 0;
    return(h);
}

fheap *fheap_build_heap(fheap_node **A, int n)
{
    fheap *h;
    int i;
    h = fheap_make_heap();
    if (n == 0)
        return(h);
    h->min = A[0];
```

```
    h->n = n;
    fheap_node_init(A[0]);
    for (i=1; i<n; i++)
    {
        fheap_node_init(A[i]);
        A[i]->left = A[i-1];
        A[i-1]->right = A[i];
        if (A[i]->key < h->min->key)
            h->min = A[i];
    }
    i--;
    A[i]->right = A[0];
    A[0]->left = A[i];
    return(h);
}

void fheap_insert(fheap *h, fheap_node *x)
{
    fheap_node_init(x);
    if (h->min == NULL)
        h->min = x->left = x->right = x;
    else
    {
        fheap_node_add(h->min, x);
        if (x->key < h->min->key)
            h->min = x;
    }
    h->n++;
}

fheap_node *fheap_minimum(fheap *h)
{
    return(h->min);
}

fheap_node *fheap_extract_min(fheap *h)
{
    fheap_node *x, *z;
    if (h->min == NULL)
```

```
        return(NULL);
    z = h->min;
    if (z->child != NULL)
    {
        for(x = z->child; x->right != z->child; x = x->right)
            x->parent = NULL;
        x->parent = NULL;
        fheap_list_concat(h->min, x);
    }
    fheap_node_remove(z);
    if (z == z->right)
        h->min = NULL;
    else
    {
        h->min = z->right;
        fheap_consolidate(h);
    }
    h->n--;
    return(z);
}

static void fheap_consolidate(fheap *h)
{
    fheap_node **A, *w, *x, *y;
    int i, d, D; /* d:degree, D:maximum degree */
    D = (int)(log(h->n) / log(2) + 1);
    A = malloc(D * sizeof(fheap_node *));
    if (A == NULL)
    {
        perror("Memory allocation error");
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < D; i++)
        A[i] = NULL;
    w = h->min;
    do
    {
        x = w;
        d = x->degree;
```

```
        w = w->right;
        while (A[d] != NULL)
        {
            y = A[d];
            if (x->key > y->key)
            {
                y = x;
                x = A[d];
            }
            fheap_link(y, x);
            A[d] = NULL;
            d++;
        }
        A[d] = x;
    } while (w != h->min);
    h->min = NULL;
    for (i = 0; i < D; i++)
        if (A[i] != NULL)
            if (h->min == NULL)
                h->min = A[i]->left = A[i]->right = A[i];
            else
            {
                fheap_node_add(h->min, A[i]);
                if (A[i]->key < h->min->key)
                    h->min = A[i];
            }
    free(A);
}

static void fheap_link(fheap_node *y, fheap_node *x)
{
    if (x->child != NULL)
        fheap_node_add(x->child, y);
    else
        x->child = y->left = y->right = y;
    y->parent = x;
    x->degree++;
    y->mark = 0;
}
```

```c
fheap *fheap_meld(fheap *h1, fheap *h2)
{
    fheap *h;
    h = fheap_make_heap();
    h->min = h1->min;
    if (h1->min == NULL)
        h->min = h2->min;
    else if (h2->min != NULL)
    {
        fheap_list_concat(h1->min,h2->min);
        if (h2->min->key < h1->min->key)
            h->min = h2->min;
    }
    h->n = h1->n + h2->n;
    free(h1);
    free(h2);
    return(h);
}

void fheap_decrease_key(fheap *h, fheap_node *x, int k)
{
    fheap_node *y;
    if (k > x->key)
    {
        fprintf(stderr, "Error: New key is greater than current
           key\n");
        return;
    }
    x->key = k;
    y = x->parent;
    if (y != NULL && x->key < y->key)
    {
        fheap_cut(h, x, y);
        fheap_cascading_cut(h, y);
    }
    if (x->key < h->min->key)
        h->min = x;
}
```

```
static void fheap_cut(fheap *h, fheap_node *x, fheap_node *y)
{
    if (x->right == x)
        y->child = NULL;
    else
        y->child = x->right;
    fheap_node_remove(x);
    y->degree--;
    fheap_node_add(h->min, x);
    x->parent = NULL;
    x->mark = 0;
}

static void fheap_cascading_cut(fheap *h, fheap_node *y)
{
    fheap_node *z;
    z = y->parent;
    if (z != NULL)
        if (y->mark == 0)
            y->mark = 1;
        else
        {
            fheap_cut(h, y, z);
            fheap_cascading_cut(h, z);
        }
}

void fheap_delete(fheap *h, fheap_node *x)
{
    fheap_node *y;
    y = x->parent;
    if (y != NULL)
    {
        fheap_cut(h, x, y);
        fheap_cascading_cut(h, y);
    }
    h->min = x;
    free(fheap_extract_min(h));
```

```
}
```

5.3 Αλγόριθμος Dijkstra

Στο αρχείο dijkstra.c ορίζεται η συνάρτηση main και δηλώνονται και υλοποιούνται οι συναρτήσεις dijkstra_adj_matrix, dijkstra_adj_lists, save_results και save_path.

Το πρόγραμμα δέχεται ως παραμέτρους το χαρακτήρα m για τη φόρτωση πίνακα γειτνίασης ή τον χαρακτήρα l για τη φόρτωση λιστών γειτνίασης, το όνομα του αρχείου εισόδου, το όνομα του αρχείου εξόδου στο οποίο θα αποθηκευτούν τα αποτελέσματα και τον κόμβο αφετηρίας από τον οποίο θα υπολογιστούν μέσω του αλγορίθμου του Dijkstra οι συντομότερες διαδρομές προς όλους τους άλλους κόμβους.

Η συναρτήσεις dijkstra δέχονται ως παραμέτρους το γράφο, τους πίνακες αποστάσεων και προκατόχων και τον κόμβο αφετηρία και υλοποιούν τον αλγόριθμο του Dijkstra με βάση το κεφάλαιο 4. Η συνάρτηση dijkstra_adj_matrix θα χρησιμοποιείται για πίνακα γειτνίασης ενώ η dijkstra_adj_lists για λίστες γειτνίασης. Μια εγγραφή στον πίνακα προκατόχων θα θεωρείται κενή αν έχει την τιμή που ορίζεται ως άπειρο.

Η συνάρτηση save_results δέχεται ως παραμέτρους το όνομα του αρχείου εξόδου, τους πίνακες αποστάσεων και προκατόχων, τον κόμβο αφετηρία και το πλήθος των κορυφών του γράφου. Δημιουργεί το αρχείο εξόδου και καλεί τη συνάρτηση save_path για κάθε κόμβο προορισμού.

Η συνάρτηση save_path βασίζεται στη πράξη Εκτύπωση Διαδρομής του κεφαλαίου 4, είναι μια αναδρομική συνάρτηση που δέχεται ως παραμέτρους ένα δείκτη σε αρχείο για εγγραφή, τον πίνακα προκατόχων, τον κόμβο αφετηρία και τον κόμβο προορισμού. Αποθηκεύει στο αρχείο χρησιμοποιώντας τον πίνακα προκατόχων τη διαδρομή από τον κόμβο αφετηρία προς τον κόμβο προορισμού.

Η συνάρτηση main αρχικά ελέγχει τον αριθμό των παραμέτρων, αν δε δόθηκαν παράμετροι ή δόθηκε λάθος αριθμός παραμέτρων θα εμφανίζεται μήνυμα που εξηγεί πως χρησιμοποιείται το πρόγραμμα και η τιμή που ορίζεται για την άπειρη απόσταση. Έπειτα το πρόγραμμα θα τερματίζει με κατάσταση επιτυχίας. Καμία διαδρομή στο γράφο δεν πρέπει να έχει συνολικό βάρος μεγαλύτερο της τιμής που ορίζεται ως άπειρο. Γίνεται φόρτωση του γράφου καλώντας τη συνάρτηση adj_matrix_load, ελέγχεται αν ο κόμβος αφετηρίας υπάρχει στο γράφο, αν δεν υπάρχει θα εμφανίζεται κατάλληλο μήνυμα και το πρόγραμμα θα τερματίζει με κατάσταση αποτυχίας. Δεσμεύεται η μνήμη για τον πίνακα αποστάσεων: dist και τον πίνακα προκατόχων: prev και καλείται η συνάρτηση dijkstra για τον υπολογισμό των συντομότερων διαδρομών. Με συμπληρωμένους πλέον τους πίνακες καλείται η συνάρτηση save_results για να αποθηκεύσει τα αποτελέσματα στο αρχείο εξόδου.

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include "adj_matrix.h"
#include "adj_lists.h"
#include "fheap.h"

#define INF (INT_MAX / 2 + 1)

void dijkstra_adj_matrix(adj_matrix *graph, int *dist, int *prev,
   int s);
void dijkstra_adj_lists(adj_lists *graph, int *dist, int *prev, int
   s);
void save_results(char *filename, int *dist, int *prev, int s, int
   n);
void save_path(FILE *fp, int *prev, int s, int v);

main(int argc, char *argv[])
{
    adj_matrix *graph_m;
    adj_lists *graph_l;
    int *dist, *prev, source, n;
    if (argc != 5)
    {
        printf("Usage: %s m|l input_file output_file
           source_vertex\n\n", argv[0]);
        printf("m: Load an adjacency matrix file\n");
        printf("l: Load an adjacency lists file\n\n");
        printf("INF = %d\n", INF);
        exit(EXIT_SUCCESS);
    }
    switch (argv[1][0])
    {
        case 'm':
            graph_m = adj_matrix_load(argv[2]);
            n = graph_m->n;
            break;
        case 'l':
```

```
        graph_l = adj_lists_load(argv[2]);
        n = graph_l->n;
        break;
    default:
        fprintf(stderr, "Error: Invalid graph representation\n");
        exit(EXIT_FAILURE);
}
source = atoi(argv[4]);
if (source < 0 || source >= n)
{
    fprintf(stderr, "Error: The selected source vertex does not
       exist in this graph\n");
    fprintf(stderr, "Range of existing vertices: %d - %d\n", 0,
       n - 1);
    exit(EXIT_FAILURE);
}
dist = malloc(n * sizeof(int));
if (dist == NULL)
{
    perror("Memory allocation error");
    exit(EXIT_FAILURE);
}
prev = malloc(n * sizeof(int));
if (prev == NULL)
{
    perror("Memory allocation error");
    exit(EXIT_FAILURE);
}
if (argv[1][0] == 'm')
{
    dijkstra_adj_matrix(graph_m, dist, prev, source);
    adj_matrix_destroy(graph_m);
}
else
{
    dijkstra_adj_lists(graph_l, dist, prev, source);
    adj_lists_destroy(graph_l);
}
save_results(argv[3], dist, prev, source, n);
```

```
    free(dist);
    free(prev);
}

void dijkstra_adj_matrix(adj_matrix *graph, int *dist, int *prev,
    int s)
{
    fheap *h;
    fheap_node **A;
    int u, v;
    A = malloc(graph->n * sizeof(fheap_node *));
    if (A == NULL)
    {
        perror("Memory allocation error");
        exit(EXIT_FAILURE);
    }
    for (u = 0; u < graph->n; u++)
    {
        A[u] = fheap_node_create_node();
        A[u]->item = u;
        dist[u] = prev[u] = A[u]->key = INF;
    }
    dist[s] = A[s]->key = 0;
    h = fheap_build_heap(A, graph->n);
    while (h->n)
    {
        u = fheap_extract_min(h)->item;
        free(A[u]);
        for (v = 0; v < graph->n; v++)
            if (graph->adj[u][v] && dist[v] > dist[u] +
                graph->adj[u][v])
            {
                dist[v] = dist[u] + graph->adj[u][v];
                prev[v] = u;
                fheap_decrease_key(h, A[v], dist[v]);
            }
    }
    free(A);
    free(h);
```

```
}

void dijkstra_adj_lists(adj_lists *graph, int *dist, int *prev, int
    s)
{
    fheap *h;
    fheap_node **A;
    adj_list_node *v;
    int u;
    A = malloc(graph->n * sizeof(fheap_node *));
    if (A == NULL)
    {
        perror("Memory allocation error");
        exit(EXIT_FAILURE);
    }
    for (u = 0; u < graph->n; u++)
    {
        A[u] = fheap_node_create_node();
        A[u]->item = u;
        dist[u] = prev[u] = A[u]->key = INF;
    }
    dist[s] = A[s]->key = 0;
    h = fheap_build_heap(A, graph->n);
    while (h->n)
    {
        u = fheap_extract_min(h)->item;
        free(A[u]);
        for (v = graph->adj[u]; v != NULL; v = v->next)
            if (dist[v->vertex] > dist[u] + v->weight)
            {
                dist[v->vertex] = dist[u] + v->weight;
                prev[v->vertex] = u;
                fheap_decrease_key(h, A[v->vertex], dist[v->vertex]);
            }
    }
    free(A);
    free(h);
}
```

```
void save_results(char *filename, int *dist, int *prev, int s, int
   n)
{
    FILE *fp;
    int v;
    fp = fopen(filename, "w");
    if (fp == NULL)
    {
        perror("Error opening output file");
        exit(EXIT_FAILURE);
    }
    fprintf(fp, "Source Vertex: %d\n", s);
    for (v = 0; v < n; v++)
    {
        fprintf(fp, "\n\n");
        fprintf(fp, "Target Vertex: %d\n", v);
        if (dist[v] != INF)
            fprintf(fp, "Distance: %d\n", dist[v]);
        else
            fprintf(fp, "Distance: INF\n");
        fprintf(fp, "Path: ");
        save_path(fp, prev, s, v);
    }
    fclose(fp);
}

void save_path(FILE *fp, int *prev, int s, int v)
{
    if (s == v)
        fprintf(fp, "%d", s);
    else
        if (prev[v] == INF)
            fprintf(fp, "No Path From %d --> %d", s, v);
        else
        {
            save_path(fp, prev, s, prev[v]);
            fprintf(fp, " --> %d", v);
        }
}
```

6 Συμπεράσματα

Ανεξάρτητα από το πλήθος των ακμών σε ένα γράφο, για την υλοποίηση του αλγορίθμου του Dijkstra με πίνακα ή με λίστα ο χρόνος που απαιτείται θα είναι πάντα της τάξης του $O(|V|^2)$ και για το λόγο αυτό οι συγκεκριμένες υλοποιήσεις είναι προτιμότερες μόνο όταν ο γράφος είναι πολύ πυκνός [2].

Η υλοποίηση με δυαδικό σωρό ελαχίστου βελτιώνει το χρόνο εκτέλεσης του αλγορίθμου καθώς εξαρτάται από το πλήθος των ακμών στο γράφο και είναι προτιμότερη όταν το $|E|$ είναι μικρότερο του $|V|^2 / lg(|V|)$ [1, 2]. Συγκεκριμένα ο χρόνος εκτέλεσης μειώνεται από $O(|V|^2)$ σε $O((|V| + |E|)lg(|V|))$ [1, 2].

Η υλοποίηση με σωρό Fibonacci εξαρτάται επίσης από το πλήθος των ακμών στο γράφο και βελτιώνει το χρόνο εκτέλεσης του αλγορίθμου σε σχέση με την υλοποίηση του δυαδικού σωρού [1, 2]. Ο χρόνος εκτέλεσής του με σωρό Fibonacci είναι $O(|V|\, lg(|V|) + |E|)$ [1, 2, 4]. Όπως φαίνεται ο αριθμός των ακμών αυξάνει το χρόνο εκτέλεσης σε σημαντικά μικρότερο βαθμό από την υλοποίηση του δυαδικού σωρού συνεπώς η υλοποίηση αυτή είναι η ασυμπτωτικά ταχύτερη.

Στην παρούσα πτυχιακή εργασία έχουν προστεθεί όλες οι απαραίτητες λεπτομέρειες και διάφορες τροποποιήσεις ώστε να είναι εύκολη, αποτελεσματική και αποδοτική η υλοποίηση των σωρών Fibonacci και του αλγορίθμου του Dijkstra και παρέχεται μια ολοκληρωμένη υλοποίησή τους στη γλώσσα προγραμματισμού C.

Βιβλιογραφία

[1] T. H. Cormen, C. Stein, R. L. Rivest, C. E. Leiserson. *Εισαγωγή στους αλγορίθμους*, Τόμος Ι. Πανεπιστημιακές Εκδόσεις Κρήτης, 2006.

[2] S. Dasgupta, C. Papadimitriou, U. Vazirani. *Αλγόριθμοι*. Εκδόσεις Κλειδάριθμος, 2009.

[3] E. W. Dijkstra. *A note on two problems in connexion with graphs*. Numer. Math., vol. 1, no. 1, pp. 269-271, 1959.

[4] M. L. Fredman, R. E. Tarjan. *Fibonacci heaps and their uses in improved network optimization algorithms*. Journal of the ACM, vol. 34, no. 3, pp. 596-615, 1987.

[5] B. W. Kerninghan, D. M. Ritchie. *Η γλώσσα προγραμματισμού C*. Εκδόσεις Κλειδάριθμος, 2008.

[6] B. Liskov, S. Zilles. *Programming with abstract data types*. SIGPLAN Not., vol. 9, no. 4, pp. 50-59, 1974.

[7] Ι. Παπουτσής. *Εισαγωγή στις Δομές Δεδομένων και στους Αλγόριθμους Υλοποίηση σϵ C*. Εκδόσεις ΑΘ. Σταμούλης, 2010.