Project | Δομές Δεδομένων

Συντάκτες

Θάνος Ιωαννίδης – 4618

Θεοχάρης Σαββίδης – 4555

Δομές που υλοποιήθηκαν

- Σωρός Ελαχίστων
- Σωρός Μεγίστων
- Δέντρο AVL
- Γράφος
- Πίνακας Κατακερματισμού

Κάποια γενικά σχόλια

Η υλοποίηση και η δοκιμή όλων των δομών και συνολικά η εργασία ,έχουν γίνει με το πρόγραμμα CodeBlocks.

Οι ενδεικτικοί χρόνοι εκτέλεσης των εντολών που ενεργοποιούν την αντίστοιχη συνάρτηση κάθε δομής και αναφέρονται στους παρακάτω πίνακες, προέκυψαν με δοκιμή αρχείου 2.000.000 στοιχείων.

Για την κάθε δομή έχει υλοποιηθεί ένα αρχείο header και ένα αρχείο implementation για καλύτερη οργάνωση του κώδικα.

Project | Δομές Δεδομένων

Η συνάρτηση Main

Η συνάρτηση main που χρησιμοποιείται λειτουργεί με σχετικά απλό τρόπο . Διαβάζεται αρχικά μία γραμμή από το αρχείο. Έπειτα διαβάζονται μία μία οι λέξεις της γραμμής και αναλόγως με την λέξη που διαβάζεται, πρώτον , γίνεται γνωστό αν υπάρχει άλλη λέξη και ποιες είναι οι πιθανές λέξεις που μπορεί να ακολουθούν (αφού οι εντολές που δίνονται έχουν σαφώς ορισμένη δομή) και διαβάζονται και αυτές. Έτσι μετά από διαδοχικές συνθήκες εκτελείται η αντίστοιχη λειτουργία που υποδεικνύει η πρόταση. Κάθε λειτουργία περικλείεται από συναρτήσεις κατάλληλες για την μέτρηση του χρόνου που χρειάζεται για να εκτελεστεί.

Η διαδικασία επαναλαμβάνεται έως ότου αδειάσει το αρχείο εισόδου.

Σωρός Ελαχίστων

Εντολή	Χρόνος Εκτέλεσης
BUILD MINHEAP	0.95 seconds
GETSIZE MINHEAP	Αμελητέος (O(1))
FINDMIN MINHEAP	Αμελητέος (O(1))
INSERT MINHEAP	7e-07 seconds
DELETEMIN MINHEAP	0.0055 seconds

Ο Σωρός Ελαχίστων υλοποιείται μέσω της συνάρτησης **buildMinHeap** με διαδοχικές εισαγωγές (Insert). Μετά από κάθε εισαγωγή στοιχείου καλείται η συνάρτηση **minHeapify**, η οποία αναδιατάσσει τον σωρό (με προσέγγιση bottom-up) ώστε να διατηρείται η ιδιότητα MinHeap. Για την εισαγωγή των δεδομένων έχει πρώτα χρησιμοποιηθεί η βοηθητική συνάρτηση **readData**, η οποία αποθηκεύει τα στοιχεία που διαβάζονται από το αρχείο στον δυναμικό πίνακα **heap**. Σε περίπτωση που χρησιμοποιηθούν όλες οι διαθέσιμες (αρχικά έχουν οριστεί 10.000) θέσεις του πίνακα, το μέγεθος αυξάνεται δυναμικά κατά 10.000 επιπλέον θέσεις.

Η συνάρτηση **getSize** επιστρέφει το τρέχον πλήθος των στοιχείων του σωρού ελαχίστων.

Η συνάρτηση **findMin** επιστρέφει το ελάχιστο στοιχείο του σωρού ελαχίστων (το στοιχείο δηλαδή που βρίσκεται στην πρώτη θέση του πίνακα).

Η συνάρτηση **insertNode** εισάγει ένα νέο στοιχείο στον σωρό ελαχίστων. Αρχικά, τοποθετεί το νέο στοιχείο στην τελευταία θέση του πίνακα σωρού. Στη συνέχεια, ακολουθεί μια διαδικασία που είναι παρόμοια με τη heapify με προσέγγιση bottom-up, προσαρμόζοντας το σωρό έτσι ώστε να διατηρείται η ιδιότητα min-heap. Αυτό σημαίνει ότι το νέο στοιχείο συγκρίνεται με τον γονέα του και, αν είναι μικρότερο, ανταλλάσσονται οι θέσεις τους. Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου το νέο στοιχείο βρει τη σωστή του θέση στον σωρό, διατηρώντας έτσι την ιδιότητα min-heap για τον σωρό ελαχίστων.

Η συνάρτηση **deleteMin** διαγράφει το ελάχιστο (πρώτο) στοιχείο του σωρού ελαχίστων και έπειτα καλεί τη συνάρτηση heapify και αναδιατάσσει ανάλογα το σωρό ώστε να διατηρηθεί η ιδιότητα minheap.

Σωρός Μεγίστων

Εντολή	Χρόνος Εκτέλεσης
BUILD MAXHEAP	0.925 seconds
GETSIZE MAXHEAP	Αμελητέος (O(1))
FINDMAX MAXHEAP	Αμελητέος (O(1))
INSERT MAXHEAP	4e-07 seconds
DELETEMAX MAXHEAP	0.0048 seconds

Ο Σωρός Μεγίστων υλοποιείται μέσω της συνάρτησης **buildMaxHeap** με διαδοχικές εισαγωγές (Insert), όπου μετά από κάθε εισαγωγή στοιχείου καλείται η συνάρτηση **maxHeapify**. Η maxHeapify αναδιατάσσει τον Σωρό με προσέγγιση Bottom-up ώστε να διατηρείται η ιδιότητα MaxHeap. Για την εισαγωγή των δεδομένων χρησιμοποιείται πρώτα η βοηθητική συνάρτηση **readData**, η οποία αποθηκεύει τα στοιχεία που διαβάζονται από το αρχείο στον δυναμικό πίνακα heap. Σε περίπτωση που χρησιμοποιηθούν όλες οι διαθέσιμες θέσεις του πίνακα (αρχικά έχουν οριστεί 10.000), το μέγεθος αυξάνεται δυναμικά κατά 10.000 επιπλέον θέσεις.

Η συνάρτηση **getSize** επιστρέφει το τρέχον πλήθος των στοιχείων του σωρού μεγίστων.

Η συνάρτηση **findMax** επιστρέφει το μέγιστο στοιχείο του σωρού μεγίστων (το στοιχείο δηλαδή που βρίσκεται στην πρώτη θέση του πίνακα).

Η συνάρτηση **insertNode** εισάγει νέο στοιχείο στον σωρό μεγίστων, τοποθετώντας το αρχικά στην τελευταία θέση του πίνακα, και έπειτα ακολουθώντας μια όμοια διαδικασία με τη heapify (προσέγγιση bottom-up) αναδιατάσσει ανάλογα το σωρό ώστε να διατηρηθεί η ιδιότητα maxheap.

Η συνάρτηση **deleteMax** διαγράφει το μέγιστο (πρώτο) στοιχείο του σωρού μεγίστων και έπειτα καλεί τη συνάρτηση heapify και αναδιατάσσει ανάλογα το σωρό ώστε να διατηρηθεί η ιδιότητα maxheap.

Project | Δομές Δεδομένων

Δένδρο AVL

Εντολή	Χρόνος Εκτέλεσης
BUILD AVLTREE	3.1 seconds
GETSIZE AVLTREE	Αμελητέος (O(1))
FINDMIN AVLTREE	6e-07 seconds
SEARCH AVLTREE	5e-07 seconds
INSERT AVLTREE	1.5e-06 seconds
DELETE AVLTREE	6e-07 seconds

Για την δομή AVL Tree έχει δημιουργηθεί μία βοηθητική κλάση **TreeNode** που αναπαριστά τον κάθε κόμβο του δένδρου AVL και ουσιαστικά η μεταβλητή root αναπαριστά όλο το δένρδο. Η κλάση TreeNode αποτελείται από τα πεδία data , left , right και height που αποθηκεύουν το δεδομένο , τη διεύθυνση του αριστερού παιδιού, την διεύθυνση του δεξιού παιδιού και το ύψος του κάθε κόμβου αντίστοιχα. Το ύψος εξασφαλίζει ότι το δένδρο διατηρεί την ιδιότητα AVL, σε περίπτωση δηλαδή που το δένδρο δεν είναι ισοζυγισμένο πραγματοποιούνται κατάλληλες περιστροφές.

Η συνάρτηση **buildAVLTree** καλώντας κάθε φορά την Insert δημιουργεί το δένδρο AVL αναδρομικά και εξασφαλίζει την ιδιότητα AVL όπως προαναφέρθηκε.

Η συνάρτηση **getSize** επιστρέφει το τρέχον πλήθος των κόμβων του δένδρου AVL.

Η συνάρτηση **findMin** επιστρέφει το ελάχιστο στοιχείο του δένδρου AVL το οποίο πάντα θα είναι το αριστερότερο φύλλο του δένδρου.

Η συνάρτηση **search** ελέγχει εάν υπάρχει στο δένδρο AVL το αναζητούμενο στοιχείο επιστρέφοντας Success ή Failure ανάλογα με το αποτέλεσμα της αναζήτησης. Ένα στοιχείο δεν βρίσκεται στο δένδρο AVL αν με τις κατάλληλες συγκρίσεις φτάσει σε φύλλο που δεν περιέχει το αναζητούμενο στοιχειο διαφορετικά μία από αυτές τις συγκρίσεις οδηγεί στο στοιχείο.

Η συνάρτηση **Insert** είναι υπεύθυνη για την εισαγωγή ενός νέου στοιχείου στο AVL δέντρο. Η διαδικασία αυτή γίνεται αναδρομικά, και κατά την εισαγωγή κάθε νέου στοιχείου, ελέγχεται αν το δέντρο παραμένει ισοζυγισμένο μέσω της συνάρτησης **getBalanceFactor**. Αν διαπιστωθεί ότι το δέντρο δεν είναι ισοζυγισμένο, πραγματοποιούνται οι κατάλληλες περιστροφές για να επαναφερθεί το δένδρο σε μορφή AVL. Οι

περιστροφές που μπορεί να γίνουν είναι οι εξής: Μονή δεξιά περιστροφή, Μονή αριστερή περιστροφή, Διπλή δεξιά περιστροφή, Διπλή αριστερή περιστροφή. Κάθε μία από αυτές τις περιστροφές εξασφαλίζει ότι το δέντρο παραμένει ισοζυγισμένο μετά την εισαγωγή του νέου στοιχείου.

Η συνάρτηση **deleteNode** είναι υπεύθυνη για την αφαίρεση ενός στοιχείου από το AVL δέντρο. Κατά την αφαίρεση ενός στοιχείου, η διαδικασία ελέγχει αν το δέντρο παραμένει ισοζυγισμένο και, εάν δεν είναι, πραγματοποιεί τις κατάλληλες περιστροφές για να επαναφέρει την ισορροπία. Αν το στοιχείο που πρόκειται να αφαιρεθεί είναι φύλλο, απλά αφαιρείται. Αν έχει ένα παιδί, το στοιχείο αντικαθίσταται από το παιδί του. Αν έχει δύο παιδιά, το στοιχείο αντικαθίσταται από τον διάδοχό του (successor) και στη συνέχεια αφαιρείται ο διάδοχος από την αρχική του θέση, διατηρώντας το δέντρο ισοζυγισμένο. Οι περιστροφές που χρησιμοποιούνται για την επαναφορά της ισορροπίας είναι οι ίδιες με αυτές που χρησιμοποιούνται στη συνάρτηση insert.

Γράφος

Εντολή	Χρόνος Εκτέλεσης
BUILD GRAPH	1.5 seconds
GETSIZE GRAPH	Αμελητέος (O(1))
COMPUTESHORTESTPATH GRAPH	6.2 seconds
COMPUTESPANNINGTREE GRAPH	0.27 seconds
FINDCONNECTEDCOMPONENTS GRAPH	Δεν υλοποιήθηκε
INSERT GRAPH	1.2e-05 seconds
DELETE GRAPH	4e-07 seconds

Η εκτέλεση που επιλέχθηκε για την δομή του γράφου έκανε την συνολική υλοποίηση πολύπλοκη προκαλώντας ιδιαίτερη δυσκολία στην ολοκλήρωση ορισμένων λειτουργιών. Για αυτόν τον λόγο δεν έχει ολοκληρωθεί η συνάρτηση **findConnectedComponents** η οποία επιστρέφει -1 απλώς για να μπορεί να τρέξει απρόσκοπτα το πρόγραμμα. Εκτός αυτού μόνο για την συνάρτηση **getShortestPathCost** έχουν χρησιμοποιηθεί οι έτοιμες βιβλιοθήκες <queue> και <unordered_map> διότι παρά τις προσπάθειες , η λειτουργία της ήταν προηγουμένως ιδιαίτερα αργή.

Στην δομή του Γράφου έχει κατασκευαστεί μία βοηθητική κλάση **HashMap** η οποία χρησιμοποιείται για την γρηγορότερη αναζήτηση των στοιχείων του γράφου. Η HashMap στην τελική υλοποίηση χρησιμοποιήθηκε μόνο στην συνάρτηση DFS.

Η συνάρτηση **Insert** φροντίζει για τον έλεγχο της διαθεσιμότητας τόσο τον κορυφών του γραφήματος όσο και των ακμών και των βαρών. Σε κάθε περίπτωση πληρότητας , χρησιμοποιώντας την κατάλληλη συνάρτηση resize (**resizeGraph**, **resizeVertexEdges**, **resizeVertexWeights**) , προσθέτει διαθεσιμότητα(διπλασιάζει την υπάρχουσα).

Με την συνάρτηση **BuildGraph** κατασκευάζεται ο γράφος , με κλήση της Insert για κάθε κόμβο.

Χρησιμοποιούνται επίσης δύο συναρτήσεις για την επιστροφή του τρέχοντος πλήθους των κορυφών και των ακμών του γράφου αντίστοιχα: η **getSizeOfVerteces** και η **getSizeOfEdges**.

Η συνάρτηση **getShortestPathCost** επιστρέφει το συντομότερο μονοπάτι από έναν κόμβο σε έναν άλλο. Αυτό το επιτυγχάνει χρησιμοποιώντας έναν αλγόριθμο εύρεσης συντομότερης διαδρομής. Η συνάρτηση υλοποιείται ως εξής: Χρησιμοποιεί μια ουρά (queue) για την αποθήκευση των κορυφών που θα επισκεφτεί. Δημιουργεί έναν πίνακα (unordered_map) που αποθηκεύει τις αποστάσεις των κορυφών από την αρχική κορυφή, με όλες τις αποστάσεις αρχικά ρυθμισμένες στο INT_MAX (που συμβολίζει το άπειρο). Προσθέτει την αρχική κορυφή (Start) στην ουρά και ρυθμίζει την απόσταση της από τον εαυτό της στο μηδέν. Όσο η ουρά δεν είναι άδεια, αφαιρεί την κορυφή από το μπροστινό μέρος της ουράς. Για κάθε γείτονα της τρέχουσας κορυφής, ελέγχει αν μπορεί να μειώσει την απόσταση του γείτονα μέσω της τρέχουσας κορυφής (χαλάρωση - relaxation). Αν η νέα υπολογισμένη απόσταση είναι μικρότερη από την αποθηκευμένη απόσταση, ενημερώνει την απόσταση και προσθέτει τον γείτονα στην ουρά για περαιτέρω επεξεργασία. Μετά την επεξεργασία όλων των κορυφών, επιστρέφει την απόσταση από την αρχική κορυφή (Start) στην τελική κορυφή (End). Αν η απόσταση είναι ακόμη INT_MAX, σημαίνει ότι η τελική κορυφή δεν είναι προσβάσιμη από την αρχική, και επιστρέφει -1.

Η συνάρτηση **getMinimumSpanningTreeCost** επιστρέφει το κόστος του ελάχιστου εκτεινόμενου δέντρου (MST) με χρήση ουσιαστικά του αλγορίθμου Kruskal. Αυτό πραγματοποιείται ως εξής: Δημιουργεί έναν πίνακα με όλες τις μοναδικές ακμές του γράφου. Ταξινομεί τις ακμές κατά αύξουσα σειρά βάρους. Δημιουργεί και αρχικοποιεί τις δομές parent και rank για την παρακολούθηση των συνόλων των κορυφών. Στη συνέχεια διασχίζει τις ακμές με αύξουσα σειρά βάρους και προσθέτει κάθε ακμή στο MST εφόσον δεν δημιουργεί κύκλο, χρησιμοποιώντας τις λειτουργίες find και unite των δομών Union-Find. Προσθέτει το βάρος κάθε επιλεγμένης ακμής στο συνολικό κόστος του MST. Τέλος, επιστρέφει το συνολικό κόστος του MST.

Η συνάρτηση **deleteEdge** διαγράφει την αντίστοιχη ακμή που δίνεται ως όρισμα εφόσον υπάρχει .

Project | Δομές Δεδομένων

Πίνακας Κατακερματισμού

Εντολή	Χρόνος Εκτέλεσης
BUILD HASHTABLE	2.15 seconds
GETSIZE HASHTABLE	Αμελητέος (O(1))
SEARCH HASHTABLE	4e-07 seconds
INSERT HASHTABLE	2e-06 seconds

Η δόμη hashtable υλοποιείται μέσω της **buildHashtable** η οποία χρησιμοποιώντας την Insert εισάγει το κάθε δεδομένο του αρχείου. Το hashtable χρησιμοποιεί μια κλάση node που αντιπροσωπεύει μια συνδεδεμένη λίστα με σκοπό να αποφευχυούν οι συγκρούσεις για δεδομένα με ίδιο key το οποίο επιστρέφεται από τη συνάρτηση κατακερματισμού **hashFunction**. Σε περίπτωση που η πληρότητα της δεσμευμένης μνήμης του πίνακα κατακερματισμού φτάσει το 70% χρησιμοποείται η συνάρτησει resize ώστε να δεσμευτεί περισσότερος χώρος για των τη δομή hashtable.

Η **getSize** επίστρέφει το τρέχον πλήθος των στοιχείων του hashtable.

Η συνάρτηση **search** ελέγχει εάν ένα στοιχείο υπάρχει στον Πίνακα Κατακερματισμού με την βοήθεια και πάλι της συνάρτησης κατακερματισμού, επιστρέφοντας Success ή Failure ανάλογα με το αποτέλεσμα της αναζήτησης.