Εργασία στις Δομές Δεδομένων 2ο μέρος
Υπεύθυνος Καθηγητής: κ. Παπαδόπουλος

Φοιτητές
Μηνάς Αδαμάντιος ΑΕΜ:2373
Μουστάκας Αριστείδης ΑΕΜ:2380

Το πρόγραμμα που δημιουργήσαμε αποτελεί προέκταση του πρώτου μέρους της εργασίας. Η βασική διαφορά είναι ότι αντί για ένα μεγάλο AVL δέντρο(ο κατάλογος της προηγούμενης εργασίας) πλέον έχουμε ένα πίνακα κατακερματισμού, του οποίου κάθε στοιχείο περιέχει και έναν δείκτη σε AVL δένδρο, που περιέχει τους συνδέσμους του και λειτουργεί σύμφωνα με τις συναρτήσεις που υλοποιήσαμε στην προηγούμενη εργασία.
Η υλοποίηση αυτού έγινε ως εξής:

Ο πίνακας κατακερματισμού περιέχει τις εξής συναρτήσεις:

1) Η συνάρτηση insertBoth(int,int,int) η οποία δέχεται σαν ορίσματα τις τιμές 2 κόμβων και το βάρος της ακμής τους. Η συνάρτηση αυτή υλοποιεί την εντολή INSERT_LINK. Αρχικά εισάγει στον πίνακα κατακερματισμού τις 2 τιμές χρησιμοποιώντας μια συνάρτηση Add η οποία σύμφωνα με την συνάρτηση κατακερματισμού hash ((2*τιμή+19) MOD μέγεθος_πίνακα) βάζει το στοιχείο στην κατάλληλη θέση(αν υπάρχει ήδη στοιχείο στην θέση αυτή λειτουργεί με την μέθοδο κατακερματισμού ανοικτής διεύθυνσης και αυξάνει κατά ένα τον αριθμό της θέσης μέχρι να βρει κενή θέση, όπου και τοποθετεί τον κόμβο). Έπειτα πηγαίνει στο δένδρο με τους γείτονες του κόμβου(το οποίο αρχικά είναι άδειο) και προσθέτει τον άλλο κόμβο που δώσαμε ως όρισμα, καθώς και το βάρος της σύνδεσής τους. Παρομοίως, όταν προσθέτει στον πίνακα τον 2ο κόμβο προσθέτει και στο δένδρο των γειτόνων του το πρώτο στοιχείο, και το βάρος της σύνδεσής τους.

2) Η συνάρτηση deleteBoth(int,int) η οποία δέχεται σαν ορίσματα τις τιμές 2 κόμβων, και υλοποιεί την εντολή DELETE_LINK. Μέσω της συνάρτησης find (που επιστρέφει τη θέση ενός κόμβου στον πίνακα) αφαιρεί από την λίστα γειτόνων του ενός κόμβου τον άλλο, και αντιστοίχως για τον 2ο κόμβο. Σε περίπτωση που αδειάσει η λίστα γειτόνων, διαγράφεται και ο κόμβος από τον πίνακα και μέσω της deleteTest ελέγχονται οι επόμενοι κόμβοι και όπου χρειάζεται γίνετε επανατοποθέτηση στον πίνακα(στην περίπτωση που είχε γίνει σύγκρουση με τον κόμβο που διαγράψαμε).

3) Η συνάρτηση check, η οποία καλείται κάθε φορά που καλείται και η insertBoth. Η συνάρτηση αυτή ελέγχει αν το πηλίκο των στοιχείων που υπάρχουν στον πίνακα διά του μεγέθους του (παράγοντας φόρτωσης πίνακα) είναι >=0,5. Στην περίπτωση αυτή δεσμεύει χώρο για έναν δεύτερο πίνακα διπλάσιου μεγέθους και εισάγει όλα τα

στοιχεία εκεί με την ενημερωμένη συνάρτηση hash που πλέον χρησιμοποιεί το καινούριο μέγεθος του πίνακα, διαγράφοντας τον παλιό. Αυτό γίνεται ώστε να έχουμε όσο το δυνατό λιγότερες συγκρούσεις στην εισαγωγή στοιχείων.

Για την εντολή CN που εντοπίζει τους κοινούς γείτονες 2 κόμβων, υλοποιήσαμε έναν αλγόριθμο ο οποίος χρησιμοποιεί την μη αναδρομική προσέγγιση της inorder για δένδρα AVL. Για τον αλγόριθμο αυτό χρησιμοποιήσαμε:

1) Την συνάρτηση samefriends η οποία δέχεται ως ορίσματα τις τιμές 2 κόμβων και με την χρήση της συνάρτησης find εντοπίζει τη θέση τους στον πίνακα κατακερματισμού, και στέλνει τους δείκτες των ριζών των δένδρων που περιέχουν τους γείτονές τους στην συνάρτηση aresame.

2) Η συνάρτηση aresame δέχεται ως ορίσματα τους δείκτες των ριζών που αναφέρθηκαν πιο πάνω. Χρησιμοποιώντας την συνάρτηση getnext παίρνει τα 2 μικρότερα στοιχεία από κάθε δένδρο AVL και τα συγκρίνει μεταξύ τους. Αν είναι ίδια εκτυπώνει στο αρχείο εξόδου και καλεί 2 φορές την getnext για να πάρει τα επόμενα στοιχεία, μία για κάθε δένδρο. Αν είναι άνισα τότε καλεί την getnext μόνο για το δένδρο που της είχε δώσει το μικρότερο στοιχείο. Η διαδικασία αυτή συνεχίζεται μέχρις ότου η getnext να επιστρέψει για κάποιο από τα 2 δένδρα το μεγαλύτερο στοιχείο του. Αν γίνει αυτό με το που ξανακληθεί η getnext για το συγκεκριμένο δένδρο η aresame τερματίζει.

3)Η συνάρτηση getnext δέχεται ως ορίσματα μία δυναμική στοίβα που έχει δημιουργηθεί στην aresame και έναν δείκτη node* ο οποίος χρησιμοποιείται για να ξέρουμε σε ποιο σημείο έχει σταματήσει η στοίβα(θα εξηγηθεί παρακάτω). Με την σειρά της η getnext καλεί την elaborofstack η οποία δέχεται ως ορίσματα την στοίβα (που είχαμε στείλει στην getnext) με αναφορά και μία μεταβλήτη int με αναφορά την οποία την στέλνουμε για να της δοθεί η τιμή που επιστρέφει η getnext.

4)Η συνάρτηση elaborofstack δέχεται ως ορίσματα μία δυναμική στοίβα, έναν δείκτη (node* &current) που της υποδεικνύει σε ποιο σημείο «βρίσκεται» η στοίβα. Αρχικά με current = root(η ρίζα του δένδρου) μια επαναληπτική διαδικασία κάνουμε push στην στοίβα τον δείκτη current και δίνουμε στο current ως τιμή τον δείκτη που δείχνει στο αριστερό παιδί του (current->left) μέχρις ότου ο δείκτης current πάρει την τιμή NULL. Αυτό σημαίνει ότι έχουμε φτάσει στο πιο αριστερά φύλλο του δένδρου. Έπειτα κάνουμε pop το πρώτο στοιχείο της στοίβας. Έτσι επιστρέφεται η πρώτη τιμή στην getnext η οποία είναι και η μικρότερη του δένδρου. Έπειτα το current γίνεται ίσο με δεξί παιδί του δείκτη που κάναμε pop (t-όπως είναι στον κώδικα), δηλαδή current=t->right. Όταν ξανακαλείται η elaborforstack επαναλαμβάνεται η ίδια διαδικασία για το t->right το οποίο είναι το νέο current.Όταν το current=NULL η elaborofstack απλά κάνει push το επόμενο στοιχείο της στοίβας και δίνει στο current επόμενη τιμή του. Έτσι καταφέρνουμε να έχουμε μια διάσχιση Αριστερά-Ρίζα-Δεξία η αλλιώς inorder, χωρίς αναδρομή, ώστε να παίρνουμε το επόμενο σε αύξουσα σειρά στοιχείο κάθε δένδρου.

Για περισσότερα στοιχεία σχετικά με την υλοποίηση της στοίβας μας μπορείτε να βρείτε στην διεύθυνση:
http://www.geeksforgeeks.org/inorder-tree-traversal-without-recursion/

Για τους αλγορίθμους Prim και Dijkstra χρησιμοποιείται σωρός ελαχίστων, η οποία θα περιγραφεί πιο κάτω.

Για την εντολή SP(shortest-path, αλγόριθμος dijkstra) χρησιμοποιούμε τους εξής αλγορίθμους:

1)InitForDijkstra, ο οποίος αρχικοποιεί 3 πίνακες:

-Τον πίνακα indices, ο οποίος έχει μέγεθος size, και για κάθε στοιχείο του πίνακα κατακερματισμού περιέχει, στον ίδιο αριθμό θέσης, την θέση του μέσα στην σωρό ελαχίστων. Π.Χ. Αν το στοιχείο 3 βρίσκεται στην θέση 5 του πίνακα κατακερματισμού, τότε στην θέση 5 του πίνακα indices θα βρίσκεται και η θέση του 3 στη σωρό, και στην θέση 5 του πίνακα output η απόστασή του από τον κόμβο που δώθηκε για τον αλγόριθμο dijkstra. Ομοίως και για τον πίνακα color.

-Τον πίνακα output, ο οποίος έχει μέγεθος size, έχει ίδια οργάνωση με τον πίνακα indices, και περιέχει τα κόστη της κάθε διαδρομής από τον αρχικό κόμβο προς οποιονδήποτε άλλο.

-Τον πίνακα color, που έχει ίδια οργάνωση με τους προηγούμενους και διακρίνει αν έχουμε επισκεφθεί ή όχι έναν κόμβο.

Οι πίνακες indices και output αρχικοποιούνται σε -1, ενώ ο πίνακας color se 0.

2)inorderadd, μια αναδρομική συνάρτηση η οποία δέχεται σαν ορίσματα έναν δείκτη σε κόμβο δένδρου node, τον σωρό ελαχίστων, την τιμή ενός κόμβου, και τους 3 παραπάνω πίνακες και προσθέτει στην σωρό μέσω της συνάρτησης insert τους γείτονες του δοθέντως κόμβου μαζί με το βάρος της μέχρι τότε διαδρομής + το βάρος της ακμής από τον κόμβο στον γείτονα, ενημερώνοντας ταυτόχρονα τον πίνακα color. Στην περίπτωση που υπάρχει ήδη στην σωρό κάποιο στοιχείο, τότε ξεχωρίζει αν το βάρος που έχει ήδη το στοιχείο αυτό είναι μεγαλύτερο από το βάρος της διαδρομής + το βάρος του συνδέσμου του κόμβου που πήραμε ως όρισμα με το στοιχείο αυτό, και αν είναι όντως μεγαλύτερο τότε το ενημερώνει. Αυτό γίνεται μέσω της συνάρτησης relax της σωρού. Η inorderadd καλείται για κάθε κόμβο του δένδρου γειτόνων με διάσχιση inorder.

3) Dijkstra, η οποία δέχεται ένα όρισμα int x και είναι η διαδικασία που υπολογίζει τα ελάχιστα μονοπάτια. Αυτό γίνεται ως εξής: Αρχικά καλεί την InitForDijkstra για

τους 3 πίνακες και έπειτα καλεί την inorderadd για το x που δώθηκε σαν όρισμα, και αφού προσθέσει τους γείτονες του κόμβου x στη σωρό αρχίζει μια επαναληπτική διαδικασία η οποία καλεί την inorderadd για κάθε κόμβο που λαμβάνουμε από την extractMin της σωρού. Η διαδικασία τελειώνει όταν αδειάσει η σωρός.

Τα αποτελέσματα εκτυπώνονται με την σειρά που υπάρχουν τα στοιχεία στον πίνακα κατακερματισμού.

Για τον αλγόριθμο αυτό αν δωθεί μη συνεκτικός γράφος πάλι θα υπολογίσει τις ελάχιστες αποστάσεις από τον κόμβο που δώθηκε προς όλους με τους οποίους συνδέεται. Για τους κόμβους με τους οποίους δεν συνδέεται δεν εκτυπώνει τίποτα.

Για την εντολή MST(minimum spanning tree, αλγόριθμος Prim):

Ο αλγόριθμος Prim ξεκινάει με την συνάρτηση runprim(), όπου ενεργοποιείται ένα ρολόι ώστε να υπολογίσουμε και τον χρόνο εκτέλεσης του αλγορίθμου. Στη συνέχεια καλείται η συνάρτηση startprim όπου και αρχικοποιείται ένας σωρός ελαχίστων Q μεγέθους ίσο με τον αριθμό των στοιχείων που έχει εκείνη την στιγμή το hashTable.

Επίσης δεσμέυουμε έναν δυναμικό πίνακα (primtable) ίδιου μεγέθους που περιλαμβάνει αντικείμενα της κλάσης primnode. Ένα τέτοιο αντικείμενο περιέχει 3 μεταβλητές value, key και prev (int) που αναπαριστούν την τιμή, το κόστος σύνδεσης με το υπόλοιπο δένδρο και το προηγούμενο του κάθε στοιχείου. Επίσης δεσμεύουμε ακόμα μερικές μεταβλητές και δείκτες που μας βοηθάνε να ξέρουμε τη συσχέτιση των θέσεων του κάθε στοιχείους στο hashTable, στο primtable και στη σωρό. Έπειτα αρχικοποιούμε στο primtable για κάθε στοιχείο τις τιμές color=0, key=-1 και prev=-1.

Στη συνέχεια κάνουμε insert στη σωρό τον αριθμό της θέσης ενός τυχαίου στοιχείου του πίνακα primtable και το κόστος σύνδεσής του με το υπόλοιπο δένδρο (key) όπου για το στοιχείο αυτό είναι 0. Έπειτα ξεκινάει μια επαναληπτική διαδικασία. Κάνουμε extractMin ένα στοιχείο u. Μετά παίρνουμε κάθε γείτονα του u. Όσοι έχουν color=0 δηλαδή δεν τους έχουμε επισκεφθεί τους κάνουμε (συγκεκριμένα κάνουμε την θέση τους στο primtable και όχι την ίδια την τιμή) insert στη σωρό μαζί με το κόστος σύνδεσής τους με το υπόλοιπο δένδρο και κάνουμε το color τους 1, το key ίσον με το βάρος της ακμής τους που τους συνδέει με το u και prev το u.

Όσοι έχουν color 1, δηλαδή τους έχουμε ξαναεπισκεφθεί συγκρίνουμε το key τους με το κόστος σύνδεσης του στοιχείου αυτού με το u. Αν το key είναι μεγαλύτερο το κάνουμε ίσο με το κόστος σύνδεσής τους με το u, prev το u και καλούμε την συνάρτηση της σωρού decreaseKey ώστε να ενημερώσει το minheap ώστε να γίνουν οι κατάλληλες αντιμεταθέσεις για να διατηρηθεί η ιδιότητα του σωρού ελαχίστων.

Μόλις ελεγχθούν όλοι οι φίλοι του u, το color του γίνεται 2. Αυτή η διαδικασία συνεχίζεται μέχρι να αδειάσει η σωρός. Τέλος αθροίζουμε τα key όλων των

στοιχείων του primtable και αυτό είναι το κόστος του ελαχίστου δένδρου. Αν υπάρχει στοιχείο στο primtable που να έχει color και prev ίσα με -1 σημαίνει πως ο γράφος είναι μη συνεκτικός και δεν υπάρχει ελάχιστο δένδρο.

Η κλάση BinaryHeap:

Η κλάση αυτή υλοποιεί μια σωρό ελαχίστων. Τα αντικείμενα της σωρού είναι αντικείμενα της κλάσης heapNode. Η κλάση αυτή ουσιαστικά αποτελεί ένα struct με μεταβλητές int value, int weight, ένα δείκτη deiktis_8esis (primnode *) για τον αλγόριθμο prim, int index και int hashTablePos.
Για τον αλγόριθμο Dijkstra χρησιμοποεί τις 4 μεταβλητές int. Το weight και το value είναι το βάρος σύνδεσης και η τιμή του κόμβου αντίστοιχα. Το index είναι η θέση του αντικειμένου μέσα στη σωρό, ενώ το hashTablePos είναι η θέση της τιμής value του αντικειμένου στον πίνακα κατακερματισμού. Έτσι κάθε φορά που εισάγουμε ένα στοιχείο στην σωρό η το επανατοποθετούμε ενημερώνουμε και τον indices χρησιμοποιώντας τις παραπάνω μεταβλητές έτσι ώστε να έχουμε στην ίδια θέση που βρίσκεται ένα στοιχείο στον πίνακα κατακερματισμού και την θέση του στη σωρό.

Για τον αλγόριθμο Prim χρησιμοποιούμε τις int μεταβλητές value και weight και έναν δείκτη σε primnode. Το weight είναι το βάρος σύνδεσης και το value είναι η θέση του στοιχείου στον primtable. Ο δείκτης deiktis_8esis δείχνει στη θέση του συγκεκριμένου στοιχείου στο primtable ώστε να μπορούμε έυκολα να αλλάζουμε σε αυτό την μεταβλητη που δείχνει τη θέση του στοιχείου στη σωρό μετά από κάθε decreaseKey.

Γενικά η BinaryHeap περιέχει τις συναρτήσεις για προσθήκη(insert), αφαίρεση ελαχίστου(extractMin) στη σωρό καθώς και τη συνάρτηση Heapify η οποία δέχεται σαν όρισμα ένα στοιχείο και το τοποθετεί στην σωστή θέση στη στοίβα. Σε κάθε insert καλείται και η heapify για την διατήρηση της ιδιότητας του minheap, όπως και στην extractMin, στην relax και στην decreaseKey.

Για τον αλγόριθμο Prim η decreaseKey καλείται για να ενημερώσει το βάρος ενός στοιχείου στη σωρό.

Επίσης για τον αλγόριθμο Dijkstra υπάρχουν οι συναρτήσεις relax και updateIndices.
Η πρώτη ενημερώνει τις τιμές του output κάθε φορά που πηγαίνουμε σε άλλον κόμβο και συναντάμε έναν γείτονά του με color=1, όπου και ελέγχουμε αν το κόστος που είχε μέχρι τώρα είναι μικρότερο απο το κόστος της καινούριας διαδρομής.Η updateIndices ενημερώνει τον πίνακα που περιέχει την θέση κάθε στοιχείου στη σωρό.

Κλείνοντας, πρέπει να τονίσουμε ότι επειδή αναλάβαμε να υλοποιήσουμε από έναν αλγόριθμο ο καθένας(prim και dijkstra) οι συναρτήσεις insert, heapify και extractMin είναι υπερφωρτομένες (λόγω διαφορών στον τρόπο αποθήκευσης των θέσεων του κάθε στοιχείου στη σωρό).