Practical Exercise 10

Ομάδα : 3

Συμμετέχοντες : Μιχάλης Μιχαήλ

Σώτος Βασιλείου

Πασιουρτίδης Κώστας

A person holding a card

Description automatically generated with low confidence

Άσκηση 1

Text

Description automatically generated

Αρχικά το πρόγραμμα διαβάζει από το αρχείο και δημιουργεί τους κόμβους του γράφου μας.

Text

Description automatically generated

Ακολούθως ζητά τα σημεία από το χρήστη και τρέχει τον αλγόριθμο Dijkstra.

Text

Description automatically generated

Αρχικά ο αλγόριθμος πραγματοποιεί αρχικοποιήσεις στις μεταβλητές μας

Graphical user interface, text, application, email

Description automatically generated

Στην συνέχεια τρέχει ο αλγόριθμος όσο δεν έχει ελέγξει όλο το Γράφο While (1). Ακολούθως για όλο το temp ελέγχω εάν το distance του temp1->value είναι μεγαλύτερο από το temp1->weight+distance[k] και εάν δεν έχω επισκεφθεί τον κόμβο αυτό. Αν ισχύει τότε θα ορίσω την τιμή του predecessors μου = με του k δηλαδή του start. Το distance του κόμβου αυτού θα είναι όσο του predecessor + το βάρος του κόμβου. Και στην συνέχεια μετακινούμε στον επόμενο κόμβο.

Ακολούθως υπολογίζω για την διαδρομή αυτή την απόσταση της, και μετά ξανά από την αρχή για την επόμενη πιθανή διαδρομή και αν είναι καλύτερη τότε την αποθηκεύω στις μεταβλητές μου για εκτύπωση

A picture containing timeline

Description automatically generated

Στην συνέχεια εκτυπώνω της μεταβλητές μου και υπολογίζω το συντομότερο μονοπάτι για εκτύπωση.

Άσκηση 2: shortest path

A picture containing text, computer

Description automatically generatedΓια κάθε k ο πίνακας D καταχωρεί το ελάχιστο μονοπάτι από τον κόμβο I στον κόμβο j εάν επιστρέψουμε μονοπάτια μέχρι και κ κόμβους .

Text

Description automatically generated

graph\_init // το ίδιο με τις προηγούμενες ασκήσεις

D=πίνακας γειτνίασης του γράφου

for(k=0;k<SIZE+1;k++){

for(i=0;i<SIZE+1;i++){

for(j=0;j<SIZE+1;j++){

if( (D[i][j]) > (D[i][k] + D[k][j]) ){

D[i][j]= D[i][k] + D[k][j] ;

}

}

}

Transitive Closure

Αρχικα ανοιγω το αρχειο μου το διαβαζω και παιρνω από μεσα του, τον αριθμο κομβων και τις ακμες δηλαδη την 1η και 2η κορυφη της ακμης. Επισης παιρνω και το βαρος.

Δημιουργω τον γραφο μου (Constactor) με τον εξης αριθμο κομβων που παιρνω από το αρχειο και στη συνεχεια με ένα while loop παιρνω ότι ειπα πιο πανω από το αρχειο προσθετωντας τις ακμες με την εξης συναρτηση addEdge προσθετωντας την κάθε ακμη στη λιστα μου.

Μετα ξεκινα η main function transitiveClosure().

ifstream fin

fin.open("ECE325\_graph.txt")

int nodes,akmes,kor1,kor2,weight

fin>>nodes

fin>>akmes

Graph graphos(nodes)

while (!fin.eof()) {

fin>>kor1>>kor2>>weight;

graphos.addEdge(kor1, kor2);

}

cout << Transitive closure matrix:

graphos.transitiveClosure()

fin.close()

Main function transitiveClosure()

// dfs ksekinontas apo oles tis korifes 1 pros 1

for (int i = 0; i < V; i++)

DFS(i, i)

for (int i=0; i<V; i++)

cout<<setw(2)<<i

cout << endl

for (int i=0; i<V; i++){

cout<< i << " "

for (int j=0; j<V; j++)

cout << tc[i][j] << " "

cout << endl

}

Η ιδεα είναι ότι:

Δημιουργω έναν πίνακα tc[V][V] που θα είναι ο τελικος πινακας transitiveClosure του γραφου. Αρχικοποιω όλες τις καταχωρήσεις του tc ως 0.

Καλώ το DFS για κάθε κόμβο του γραφου μου για να επισημάνω τις προσβάσιμες κορυφές στο tc. Όταν κανω αναδρομικό DFS στο function, δεν καλω το DFS για μια γειτονική κορυφή, εάν έχει ήδη επισημανθεί ως προσβάσιμη στο tc[][].

DFS

if(s==v){

if(find(adj[v].begin(),adj[v].end(),v)!=adj[v].end()) //vrisko to v mesa sti lista (found diladi) apo tin arxi mexri to telos me to find

tc[s][v] = true

}

else

tc[s][v] = true //not found

// vrisko oles tis korifes apo v

list<int>::iterator i

for (i = adj[v].begin(); i != adj[v].end(); ++i)

{

if (tc[s][\*i] == false)

{

if(\*i==s)

{

tc[s][\*i]=1; //to monopati pou diasxizo tou vazo 1

}

else

{

DFS(s, \*i)

}}}