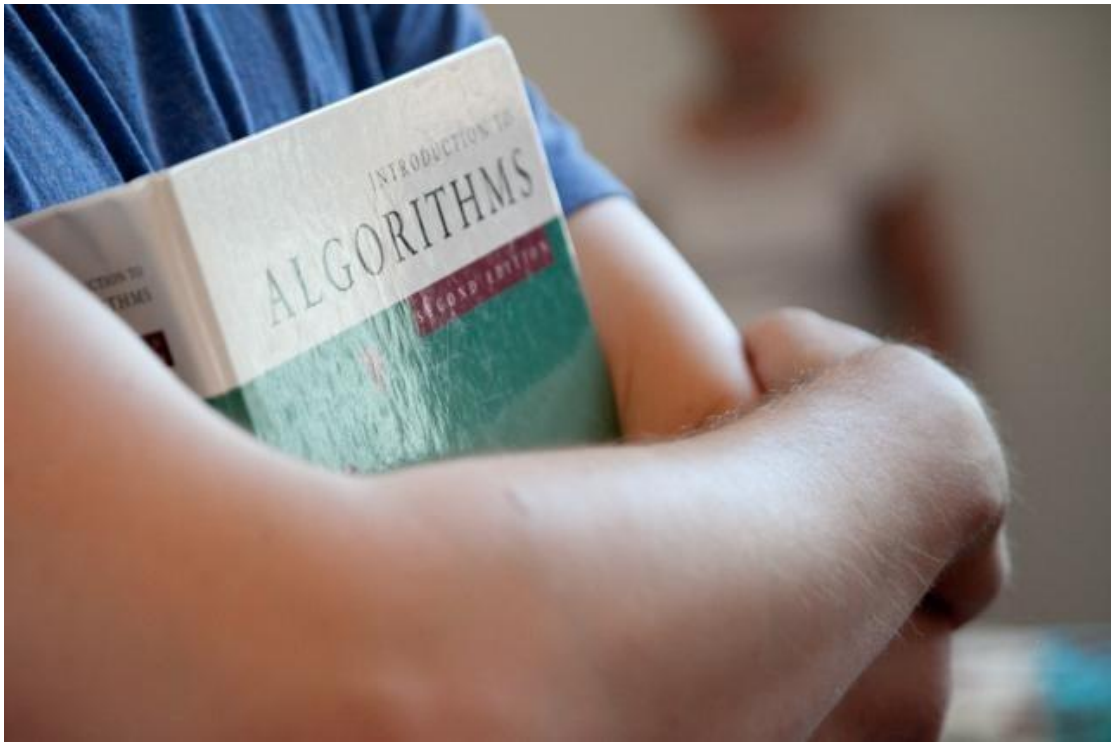
Practical Exercise 10

Ομάδα : 3

Συμμετέχοντες : Μιχάλης Μιχαήλ
Σώτος Βασιλείου
Πασιουρτίδης Κώστας



Άσκηση 1

```
int main(){
    int n1,n2,w1;
    FILE *readfile;

    readfile=fopen("ECE325_graph.txt","r");

    int V,E,c;
    fscanf(readfile,"%d",&V);
    fscanf(readfile,"%d",&E);

    Graph *graph=createGraph(V);
    for (c=0;c<E;c++){
        fscanf(readfile,"%d",&n1);
        fscanf(readfile,"%d",&n2);
        fscanf(readfile,"%d",&w1);
        addEdge(graph,n1,n2,w1);
    }
    fclose(readfile);
```

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

```
    int start=0,endd=0;
    while (start!=-1 && endd!=-1){
        printf("Give starting point and destination for shortest path:");
        scanf("%d",&start);
        scanf("%d",&endd);
        if (!(start==-1 || endd==-1))
            Dijkstra(graph,start,endd);
```

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

```
65  void Dijkstra(Graph *graph,int start,int endd){
66      int v=graph->V,c,finishcounter,k,min;
67      int distance[v],predecessors[v],visited[v];
68      for(c=0;c<v;c++){
69          distance[c]=INT_MAX;
70          predecessors[c]=-1;
71          visited[c]=0;
72      }
73      countryNode *temp,*temp1;
74      k=start;
75      distance[k]=0;
76      visited[k]=1;
77      finishcounter=0;
78      temp=graph->array[k].head;
```

Αρχικά ο αλγόριθμος πραγματοποιεί αρχικοποιήσεις στις μεταβλητές μας

```
    while (finishcounter<v){
        templ=temp;
        while (templ!=NULL){
            if (distance[templ->value]>templ->weight+distance[k] && visited[templ->value]==0){

                predecessors[templ->value]=k;

                distance[templ->value]=distance[predecessors[templ->value]]+templ->weight;
            }
            templ=templ->next;
        }

        min=INT_MAX;
        visited[k]=1;
        for(c=0;c<v;c++){
            if (distance[c]<min && visited[c]==0){
                min=distance[c];
                k=c;
            }
        }
        finishcounter++;
        temp=graph->array[k].head;
    }
```

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

```
102         printf("\nNodes:\t\t");
103         for(c=0;c<v;c++){
104             printf("%d\t",c);
105         }
106         printf("\nCosts:\t\t");
107         for(c=0;c<v;c++){
108             printf("%d\t",distance[c]);
109         }
110         printf("\nPredecessors:\t");
111         for(c=0;c<v;c++){
112             if (predecessors[c]==-1)
113                 printf("Null\t");
114             else
115                 printf("%d\t",predecessors[c]);
116         }
117
118     if (distance[endd]==INT_MAX){
119         printf("\nThe is no path\n");
120         return;
121     }
122     printf("\nShortest Path from %d to %d: ",start,endd);
123     k=endd;
124     int z=0;
125     int path[v];\
126     while (k!=start){
127         path[z]=k;
128         z++;
129         k=predecessors[k];
130    }
131    path[z]=start;
132     for(c=z;c>=0;c--){
133         printf("%d ",path[c]);
134     }
135     printf("\n");
136
```

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

Άσκηση 2: shortest path

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

$$\left[D_{ij}^{(0)}\right] = \begin{cases} w_{ij} & \text{Εάν υπάρχει ακμή από τον } i \text{ στον } j \\ \infty & \text{Εάν δεν υπάρχει ακμή} \end{cases}$$

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]  ;
                        }
                }
        }
```

```
diavazw 5 xwres
diavazw 12 diadromes
-->0 1 3 <--
-->0 2 2 <--
-->1 0 5 <--
-->1 2 5 <--
-->1 3 4 <--
-->1 4 1 <--
-->2 0 4 <--
-->2 1 3 <--
-->3 1 4 <--
-->3 4 5 <--
-->4 3 5 <--
-->4 1 3 <--

      0  1  2  3  4
--------------------------------
  0|  0  3  2  0  0
  1|  5  0  5  4  1
  2|  4  3  0  0  0
  3|  0  4  0  0  5
  4|  0  3  0  5  0



D0:
  0|  ▌  3  2  ▌  ▌
  1|  5  ▌  5  4  1
  2|  4  3  ▌  ▌  ▌
  3|  ▌  4  ▌  ▌  5
  4|  ▌  3  ▌  5  ▌
D0:
  0|  0  3  2  ▌  ▌
  1|  5  0  5  4  1
  2|  4  3  0  ▌  ▌
  3|  ▌  4  ▌  0  5
  4|  ▌  3  ▌  5  0
D1:
  0|  0  3  2  7  4
  1|  5  0  5  4  1
  2|  4  3  0  7  4
  3|  9  4  9  0  5
  4|  8  3  8  5  0
D2:
  0|  0  3  2  7  4
  1|  5  0  5  4  1
  2|  4  3  0  7  4
  3|  9  4  9  0  5
  4|  8  3  8  5  0
D3:
  0|  0  3  2  7  4
  1|  5  0  5  4  1
  2|  4  3  0  7  4
  3|  9  4  9  0  5
  4|  8  3  8  5  0
D4:
  0|  0  3  2  7  4
  1|  5  0  5  4  1
  2|  4  3  0  7  4
  3|  9  4  9  0  5
  4|  8  3  8  5  0

--------------------------------
```

Transitive Closure

Αρχικα ανοιγω το αρχειο μου το διαβαζω και παιρνω από μεσα του, τον αριθμο κομβων και τις ακμες δηλαδη την $1^{η}$ και $2^{η}$ 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)
         }}}
```