

ΣΚΕΠΤΙΚΟ ΕΠΙΛΥΣΗΣ ΚΩΔΙΚΑ 4.1

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ Ι

ΕΞΑΜΗΝΟ 4^ο

ΑΝΔΡΕΑΣ ΒΑΝΙΚΙΩΤΗΣ

ΑΜ:Ε20013

Βήματα εκτέλεσης :

Συνάρτηση `int main()`:

1. Αρχικά δηλώνουμε τις βιβλιοθήκες που χρησιμοποιούνται στο πρόγραμμα μας και πέρα από αυτό δηλώνουμε και δυο σταθερές καθολικές μεταβλητές `infinity` και `max` αντίστοιχα.
2. Δηλώνουμε τις δυο συναρτήσεις που θα χρησιμοποιησούμε (`Dijkstra`, `Bellman-ford`) κανονικά με τα ορίσματα τους.
3. Στην συνέχεια μπαίνουμε στην συνάρτηση `main` όπου δηλώνουμε τις μεταβλητές που χρησιμοποιούμε και διαβάζουμε τον αριθμό των κόμβων ο οποίος πρέπει να είναι 6 διαφορετικά θα μπαίνουμε στον έλεγχο εγκυρότητας και θα εμφανίζεται μήνυμα λάθους και θα ζητάει από τον χρήστη να εισάγει ξανά τον αριθμό των κόμβων.
4. Αποθηκεύουμε την μεταβλητή `n` στην `V` για να την χρησιμοποιήσουμε στον `Bellman-Ford`.
5. Εν συνεχεία παίρνουμε τα κόστη των συνδέσμων από το αρχείο `costs.txt` (για να τρέξει σωστά το αρχείο θα πρέπει το αρχείο `.c` να βρίσκεται στο ίδιο `path` με το `.txt` file).
6. Έπειτα διαβάζουμε τον αρχικό κόμβο για τον οποίο θέλουμε να βρούμε την ελάχιστη διαδρομή. Σε περίπτωση που είναι εκτός του πεδίου τιμών ζητάει από τον χρήστη να επαναπληκτρολογήσει τον αρχικό κόμβο.
7. Αποθηκεύουμε αυτή την μεταβλητή σε μια άλλη για να την χρησιμοποιήσουμε στον `Bellman-Ford` και την μειώνουμε κατά ένα για να την χρησιμοποιήσουμε στον `Dijkstra` (ο λόγος που το κάνουμε αυτό είναι γιατί τα `loops` στην συγκεκριμένη συνάρτηση ξεκινάνε από το 0).
8. Στην συνέχεια εμφανίζουμε το μενού επιλογής του χρήστη(επιλογή 0 να εμφανίσει μόνο τον αλγόριθμο του `Dijkstra`, επιλογή 1 να εμφανίσει μόνο τον `Bellman-Ford` και επιλογή 2 να εμφανίζει και τους δυο). Αν πληκτρολογήσουμε κάποια τιμή εκτός από αυτές τις 3 εμφανίζεται μήνυμα λάθους και ξαναγίνεται καταχώρηση της τιμής.
9. Στην συνέχεια και έπειτα από τον καθαρισμό της οθόνης μπαίνουμε στην δομή επιλογής (αν μπει στην πρώτη τότε έχουμε επιλέξει μόνο τον αλγόριθμο του `Dijkstra`) από τον οποίο καλούμε

την συνάρτηση με ορίσματα τα κόστη, τους κόμβους και τον αρχικό κόμβο.

10. Εφόσον ολοκληρωθεί πατάμε κάποιο κουμπί για να κλείσουμε το πρόγραμμα.
11. Αν ο χρήστης επιλέξει το 1 θα κάνει ακριβώς την ίδια διαδικασία για τον Bellman-Ford αυτή την φορά μόνο που ελέγχουμε επιπλέον αν υπάρχουν αρνητικά κόστη στο αρχείο.
12. Αν ο χρήστης επιλέξει κάνει το βήμα 10 και αμέσως μετά το βήμα 12 και θα εμφανίσει και τους δυο αλγορίθμους.

Συνάρτηση Dijkstra():

1. Δηλώνουμε τις τοπικές μεταβλητές της συνάρτησης.
2. Στο `pred []` σημειώνουμε τον προκατόχο κάθε κόμβου, το `count` δίνει τον αριθμό του κόμβου που είναι πιο κοντά, και δημιουργούμε το `matrix` με το κόστος.
3. Όπου υπάρχει η τιμή 0 εκχωρείται η καθολική μεταβλητή `infinity` διαφορετικά το εκάστοτε στοιχείο του πίνακα `myarr`.
4. Στην συνέχεια αρχικοποιούμε τον πίνακα `distance`, `pred`, και `visited`.
5. Εν συνεχεία μέσα σε μια `while` η οποία εκτελείται όσο το `count` είναι μεγαλύτερο του 5 γίνονται τα ακόλουθα:
 - Αρχικοποίηση της `mindistance` στο άπειρο.
 - Σε μια `for` ψάχνουμε για κάθε κόμβο την ελάχιστη διαδρομή και την αποθηκεύουμε στο `nextnode`.
 - Τσεκάρουμε αν υπάρχει καλύτερη διαδρομή.
6. Στην συνέχεια εμφανίζουμε τις διαδρομές και τα κόστη και σε μια `else` εμφανίζουμε και του κόμβου αφετηρία.
7. Εν συνεχεία κάνοντας την ίδια διαδικασία από το βήμα 3 ψάχνουμε να βρούμε τον γειτονικό κόμβο `nexthop` του οποίου έχουμε αρχικοποιήσει όλες του τις τιμές στο 0 για αρχή (στην ουσία η επανάληψη θα γίνεται μια φορά λιγότερη από αυτή του βήματος 6).
8. Σε μια άλλη `for` βρίσκουμε το `nexthop` για τον `startnode` και για αυτές τιμές που είναι εκχωρισμένες ακόμα στο 0 ο `nexthop` είναι ίσος με τον αριθμό της επανάληψης συν το 1.

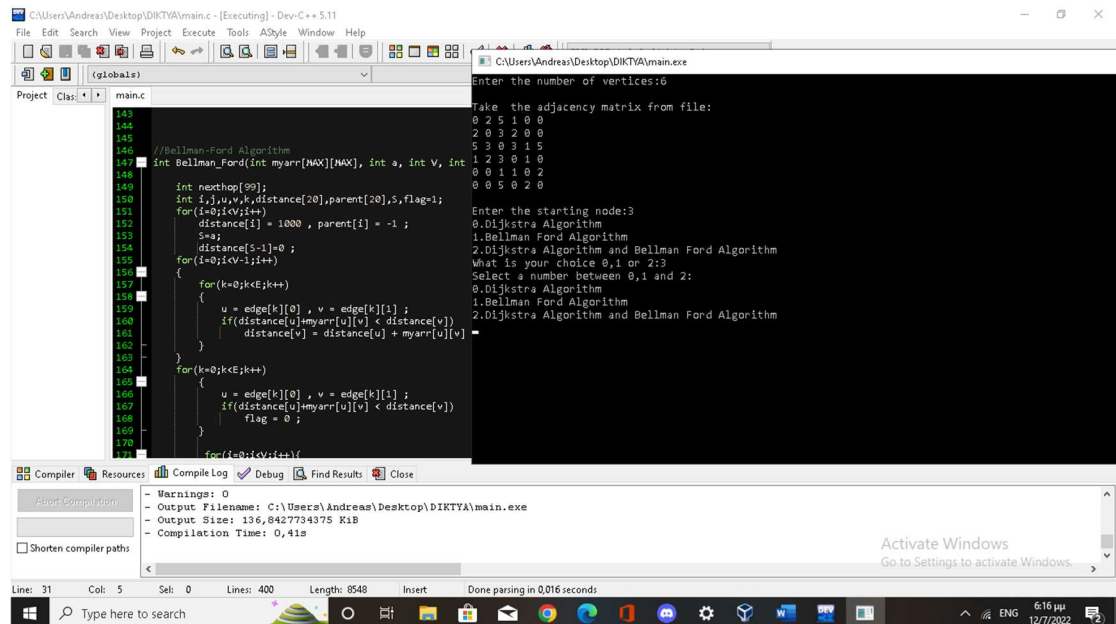
9. Τέλος τυπώνουμε τον πίνακα δρομολόγησης του επιλεγμένου κόμβου.

Συνάρτηση Bellman-Ford():

1. Δηλώνουμε τις τοπικές μεταβλητές που χρησιμοποιούμε.
2. Σε μια for loop αρχικοποιούμε την απόσταση, τον προκάτοχο ενώ βάζουμε και στην μεταβλητή S την μεταβλητή a που έχουμε καλέσει από την συνάρτηση η οποία έχει τον αριθμό του κόμβου που θέλουμε να γίνει η δρομολόγηση.
3. Στην συνέχεια στην επόμενη for loop η οποία συνοδεύεται και από άλλη μια εμφωλευμένη υπολογίζουμε την απόσταση του κάθε κόμβου και βρίσκουμε και τον προκάτοχο του κάθε κόμβου.
4. Στην επόμενη επανάληψη βρίσκουμε συγκρίνουμε τις αποστάσεις και αν ισχύει η συνθήκη μας αλλάζει η τιμή της flag την οποία στο τέλος επιστρέφουμε την συνάρτηση για να δούμε αν υπήρξε αρνητικό κόστος σε κάποιο κόμβο.
5. Εκχωρούμε όπου ο προκατόχος είναι ίσος με -1 να γίνει ίσος με 0 (εννοείται μέσα σε επαναλήψη η οποία εκτελείται όσος είναι και ο αριθμός των κόμβων).
6. Εν συνεχεία όπως και στον αλγόριθμο του Dijkstra με τον ίδιο τρόπο βρίσκω πρώτα τις ελάχιστες διαδρομές με τα κόστη τους.
7. Αρχικοποιούμε σε πίνακα nexthop τους γειτονικούς κόμβους(η αρχικοποίηση γίνεται στο 0).
8. Όπως και στον Dijkstra βρίσκω τον γειτονικό κόμβο του κόμβου που θέλουμε να βρούμε την ελάχιστη διαδρομή.
9. Τέλος τυπώνουμε τον πίνακα δρομολόγησης για τον Bellman-Ford και επιστρέφουμε την flag στην main συνάρτηση.

Ενδεικτικά Αποτελέσματα:

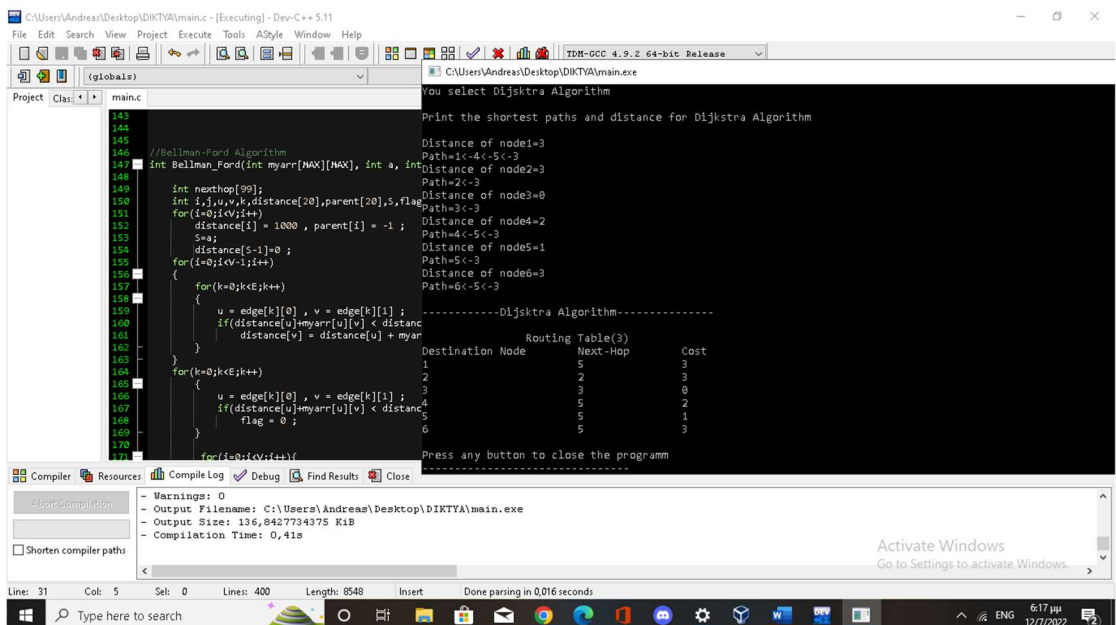
1. Εμφανίζουμε πρώτα μόνο τον αλγόριθμο του Dijkstra με αρχικό κόμβο τον 3.



The screenshot shows a C++ IDE with a project named 'main.c'. The code implements the Dijkstra algorithm. The output window displays the following text:

```
Enter the number of vertices:6
Take the adjacency matrix from file:
0 2 5 1 0 0
2 0 3 2 0 0
5 3 0 3 1 5
1 2 3 0 1 0
0 0 1 1 0 2
0 0 5 0 2 0

Enter the starting node:3
0.Dijkstra Algorithm
1.Bellman Ford Algorithm and Bellman Ford Algorithm
What is your choice 0,1 or 2:3
Select a number between 0,1 and 2:
0.Dijkstra Algorithm
1.Bellman Ford Algorithm
2.Dijkstra Algorithm and Bellman Ford Algorithm
```



The screenshot shows a C++ IDE with a project named 'main.c'. The code implements the Dijkstra algorithm. The output window displays the following text:

```
You select Dijkstra Algorithm

Print the shortest paths and distance for Dijkstra Algorithm
Distance of node1=3
Path=1<-4<-5<-3
Distance of node2=3
Path=2<-3
Distance of node3=0
Path=3<-3
Distance of node4=2
Path=4<-5<-3
Distance of node5=1
Path=5<-3
Distance of node6=3
Path=6<-5<-3

-----Dijkstra Algorithm-----
Routing Table(3)
Destination Node Next-Hop Cost
1 5 3
2 2 3
3 3 0
4 5 2
5 5 1
6 5 3

Press any button to close the program
```

2. Εμφανίζουμε τον αλγόριθμο του Bellman-Ford για αρχικό κόμβο 2

```

143 //Bellman-Ford Algorithm
144 int Bellman_Ford(int myarr[MAX][MAX],
145 int nexthop[99]);
146 int i,j,u,v,k,distance[20],parent[20];
147 for(i=0;i<V;i++)
148     distance[i] = 1000 , parent[i] = -1;
149 S=s;
150 distance[S]=0;
151 for(i=0;i<V-1;i++)
152     for(k=0;k<E;k++)
153     {
154         u = edge[k][0] , v = edge[k][1];
155         if(distance[u]+myarr[u][v]<distance[v])
156             distance[v] = distance[u]+myarr[u][v];
157         parent[v] = u;
158     }
159 for(i=0;i<V;i++)
160     for(k=0;k<E;k++)
161     {
162         u = edge[k][0] , v = edge[k][1];
163         if(distance[u]+myarr[u][v]<distance[v])
164             flag = 0;
165     }
166 if(flag == 0)
167     printf("No negative weight cycle\n");
168 else
169     printf("Negative weight cycle exists\n");
170 }

```

Output:

```

You select Bellman Ford Algorithm
Print the shortest paths and distance for Bellman-Ford Algorithm
Distance of node1=2
Path=1<-2
Distance of node2=0
Path=-2<-2
Distance of node3=3
Path=3<-2
Distance of node4=2
Path=4<-2
Distance of node5=3
Path=5<-4<-2
Distance of node6=5
Path=6<-5<-4<-2
-----Bellman-Ford Algorithm-----
Routing Table(2)
Destination Node    Next-Hop    Cost
1                    1            2
2                    2            0
3                    3            3
4                    4            2
5                    4            3
6                    4            5

```

3. Εμφανίζουμε και τους δυο αλγορίθμους με αρχικό κόμβο αυτή τη φορά το 5.

```

143 //Bellman-Ford Algorithm
144 int Bellman_Ford(int myarr[MAX][MAX],
145 int nexthop[99]);
146 int i,j,u,v,k,distance[20],parent[20];
147 for(i=0;i<V;i++)
148     distance[i] = 1000 , parent[i] = -1;
149 S=s;
150 distance[S]=0;
151 for(i=0;i<V-1;i++)
152     for(k=0;k<E;k++)
153     {
154         u = edge[k][0] , v = edge[k][1];
155         if(distance[u]+myarr[u][v]<distance[v])
156             distance[v] = distance[u]+myarr[u][v];
157         parent[v] = u;
158     }
159 for(i=0;i<V;i++)
160     for(k=0;k<E;k++)
161     {
162         u = edge[k][0] , v = edge[k][1];
163         if(distance[u]+myarr[u][v]<distance[v])
164             flag = 0;
165     }
166 if(flag == 0)
167     printf("No negative weight cycle\n");
168 else
169     printf("Negative weight cycle exists\n");
170 }

```

Output:

```

Enter the number of vertices:6
Take the adjacency matrix from file:
0 2 5 1 0 0
2 0 3 2 0 0
5 3 0 3 1 5
1 2 0 1 0 0
0 0 1 1 0 2
0 0 5 0 2 0
Enter the starting node:5
0.Dijkstra Algorithm
1.Bellman Ford Algorithm
2.Dijkstra Algorithm and Bellman Ford Algorithm
What is your choice 0,1 or 2:

```

