C ΑΛΓΟΡΙΘΜΟΙ

ΜΑΘΗΜΑ 6

ΣΥΝΤΟΜΟΤΕΡΑ ΜΟΝΟΠΑΤΙΑ

ΠΕΡΙΕΧΟΜΕΝΑ:

1. Αλγόριθμος του Dijkstra
    1. Περιγραφή
    2. Υλοποίηση σε C
    3. Ασκήσεις
2. Αλγόριθμος Bellman-Ford
    1. Περιγραφή
    2. Υλοποίηση σε C
    3. Ασκήσεις

Γιώργος Μ.

Σμαραγδένιος Χορηγός Μαθήματος

Πάνος Γ.

Ασημένιος Χορηγός Μαθήματος

- **<u>Συντομότερο Μονοπάτι:</u>**
  - Δίνεται απλός, μη κατευθυνόμενος γράφος με βάρη G=(V,E,W), αφετηρία s, προορισμός t.
  - Ζητείται το συντομότερο μονοπάτι από s στο t
  - (Αναλυτικά: Βλέπε Θεωρία Γράφων: Μάθημα 5.4)

<u>Αλγόριθμος του Dijkstra</u>

<u>Αρχικοποίηση:</u>

- Θέτουμε όλες τις ετικέτες L[v]=+∞ εκτός της αφετηρίας που έχει L[s]=0

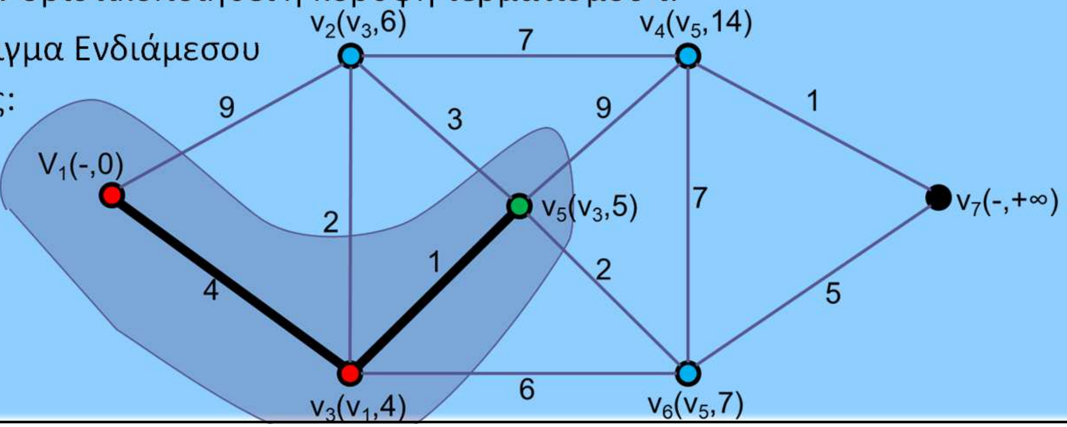
<u>Σε κάθε βήμα:</u>

- Οριστικοποιείται η κορυφή με το μικρότερο κόστος από τις μη οριστικοποιημένες
- Διορθώνονται οι ετικέτες των γειτονικών μη οριστικοποιημένων κορυφών (σε περίπτωση που βρεθεί καλύτερο μονοπάτι από την κορυφή που οριστικοποιήθηκε)

<u>Τερματισμός:</u>

- Όταν *οριστικοποιηθεί* η κορυφή τερματισμού t.

Παράδειγμα Ενδιάμεσου Βήματος:



<u>Παρατηρήσεις για την υλοποίηση:</u>

- Επιλέγουμε την αποτύπωση του γράφου με λίστες γειτνίασης (λόγω πιο γρήγορης πρόσβασης στους επόμενους μιας κορυφής)

<u>Αλγόριθμος σε Ψευδογλώσσα (~ από Wikipedia):</u>

```
procedure Dijkstra(G=(V,E,W), αφετηρία, προορισμός):
   Q: σύνολο όλων των κορυφών
   Αρχικοποίησε: L[v]=+∞, P[v]=-1 για κάθε κορυφή v
   Θέσε L[αφετηρία]=0, P[αφετηρία]=αφετηρία

   while (Q δεν είναι άδειο)
      v = κορυφή του Q με ελάχιστο L
      Αφαίρεσε τη v από το Q
      Για κάθε γείτονα x της v (που ανήκει στο Q):
            Αν L[v]+W[v,x] < L[x]:
                  L[x] = L[v]+W[v,x]
                  P[x] = v
   Υπολόγισε το συντομότερο μονοπάτι από τους πίνακες
         L,P και επέστρεψέ το.
```

<u>Πολυπλοκότητα:</u>

- Η πολυπλοκότητα του αλγορίθμου είναι O(n*A+m*B), A: χρόνος εντοπισμού κορυφής και B: διόρθωση στο Q
- Αν Q είναι απλός πίνακας: $O(n^2+m) = O(n^2)$
- Αν Q είναι σωρός Fibonacci: O(m+nlogn)

Υλοποίηση σε C (βλέπε project Dijkstra):

```
void Dijkstra(GRAPH g, int start, int target, int **shortest_path,
              int *path_size, int *path_length)
{
   int *Q, *L, *P;
   int i, x, v, cnt, vertices, neighbors_length;
   int infinity, min;
   elem *neighbors;

   vertices = GR_vertices_count(g);

   /* L[*]: Mikos elaxistis diadromis gia na ftasoume stin korufi */
   L = (int *)malloc(sizeof(int)*vertices);
   if (!L)
   {
      printf("Error Allocating Memory!");
      exit(0);
   }

   infinity = 0;
   for (v=0; v<vertices; v++)
   {
      GR_neighbors(g, v, &neighbors_length, &neighbors);
      for (i=0; i<neighbors_length; i++)
         infinity += neighbors[i].weight;
      free(neighbors);
   }
```

```
   for (i=0; i<vertices; i++)
      L[i] = infinity;
   L[start] = 0;

   /* P[*]: proigoymenos tis korifis sti veltisti diadromi */
   P = (int *)malloc(sizeof(int)*vertices);
   if (!P)
   {
      printf("Error Allocating Memory!");
      exit(0);
   }
   for (i=0; i<vertices; i++)
      P[i] = -1;
   P[start] = start;

   /* Q[i]=0 an i korufi den exei oristikopoiithei, alliws 0 */
   Q = (int *)malloc(sizeof(int)*vertices);
   if (!Q)
   {
      printf("Error Allocating Memory!");
      exit(0);
   }
   for (i=0; i<vertices; i++)
      Q[i] = 0;
```

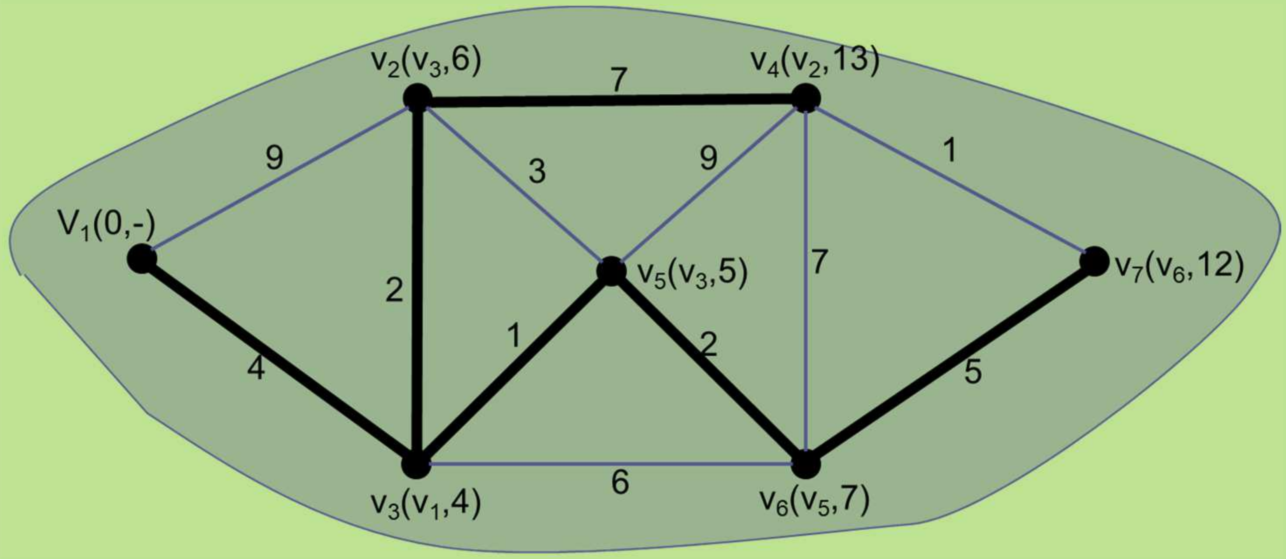
```
cnt=0;
   while(cnt<vertices)
   {
      min=infinity;
      for (i=0; i<vertices; i++)
         if (L[i]<min && Q[i]==0)
         {
            min = L[i];
            v=i;
         }
      cnt++;
      Q[v]=1;
      if (v==target) break;
      GR_neighbors(g, v, &neighbors_length, &neighbors);
      for (x=0; x<neighbors_length; x++)
      {
         if (Q[neighbors[x].id]==0 &&
             L[v]+neighbors[x].weight<L[neighbors[x].id])
         {
            L[neighbors[x].id] = L[v] + neighbors[x].weight;
            P[neighbors[x].id] = v;
         }
      }
      free(neighbors);
   }
```

```
   *path_size = 2;
   v = target;
   while(P[v]!=start)
   {
      v = P[v];
      (*path_size)++;
   }
   (*shortest_path) = (int *)malloc(sizeof(int)*(*path_size));
   if (!(*shortest_path))
   {
      printf("Error Allocating Memory!");
      exit(0);
   }
   v = target;
   for (i=*path_size-1; i>=0; i--)
   {
      (*shortest_path)[i] = v;
      v = P[v];
   }
   *path_length = L[target];

   free(L);
   free(Q);
   free(P);
}
```

Παρατήρηση:

- Αν αφήσουμε τον αλγόριθμο του Dijkstra να τρέξει για όλες τις κορυφές του γραφήματος, τότε υπολογίζει το δένδρο συντομότερων μονοπατιών από την αφετηρία.



Άσκηση 1: Συντομότερα Μονοπάτια από δεδομένη αφετηρία

Κατασκευάστε τη συνάρτηση BFSShortestPathTree η οποία με ορίσματα έναν απλό μη κατευθυνόμενο γράφο και μία αφετηρία:

- Τυπώνει όλα τα μονοπάτια από την αφετηρία προς κάθε προορισμό.
- Επιστρέφει το δένδρο συντομότερων μονοπατιών

Άσκηση 2: Με ουρά προτεραιότητας

Θα υλοποιήσουμε το Q ως μια ουρά προτεραιότητας.

- Η προτεραιότητα θα είναι η ετικέτα μήκους L (όσο μικρότερο το L, τόσο μεγαλύτερη η προτεραιότητα)
- Θα πρέπει ο εντοπισμός της κορυφής με το μικρότερο L, να γίνεται σε σταθερό χρόνο
- Ενώ όποτε γίνεται διόρθωση του L κάποιας κορυφής, θα πρέπει να γίνεται διόρθωση της ουράς.

Ενσωματώστε την ουρά προτεραιότητας στον αλγόριθμο.

[Η υλοποίηση ουράς προτεραιότητας έχει γίνει στο μάθημα Δομές Δεδομένων σε C – Μάθημα 3: Ουρά – Εφαρμογή 2]

Παρατήρηση:

- Η χρήση μίας πιο αποδοτικής ουράς προτεραιότητας (σωρός Fibonacci) ρίχνει την πολυπλοκότητα έως και O((m+n)logn)

- **Συντομότερο Μονοπάτι:**
  - Δίνεται απλός, μη κατευθυνόμενος γράφος με βάρη G=(V,E,W), αφετηρία s,
  - Ζητείται το συντομότερο μονοπάτι από το s προς όλες τις κορυφές
  - (Αναλυτικά: Βλέπε Αλγόριθμοι και Πολυπλοκότητα: Μάθημα 2.2)

Βασική ιδέα:

- Υπολόγισε διαδοχικά τα συντομότερα μονοπάτια μήκους 1, μήκους 2, …, έως και μήκος n-1.

Αλγόριθμος Bellman – Ford σε Ψευδογλώσσα:

Bellman-Ford(G=(V,E,W) και αφετηρία s)
Αρχικοποίησε για όλες τις κορυφές i: B[i] σε +∞
B[s]=0, P[s]=s

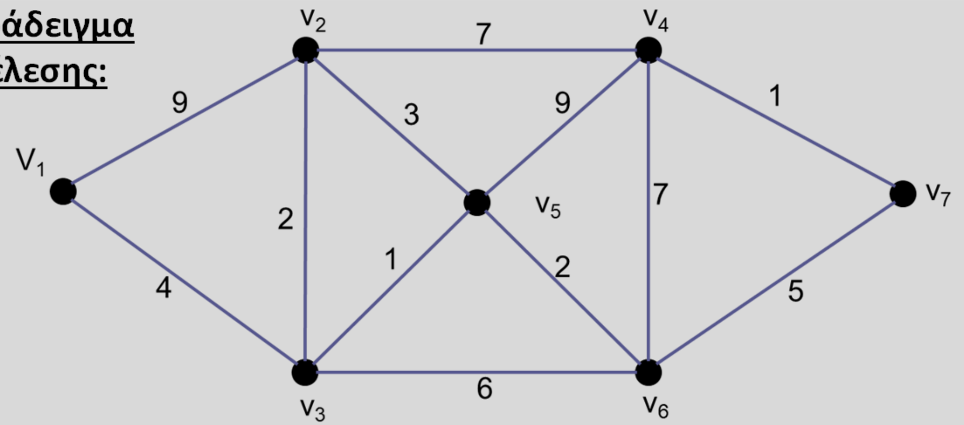
Bnew = B
Για k=1 έως n-1:
  Για κάθε κορυφή i:
    Για κάθε κορυφή j γειτονική της i:
      Αν $B[j]+W[j][i]<Bnew[i]$:
        $Bnew[i]=B[j]+W[j][i]$
        $P[i]=j$
  $B=Bnew$

Επέστρεψε το δένδρο συντομότερων μονοπατιών

Παρατηρήσεις για την υλοποίηση:

- Επιλέγουμε την αποτύπωση του γράφου με λίστες γειτνίασης (λόγω πιο γρήγορης πρόσβασης στους επόμενους μιας κορυφής)

Παράδειγμα Εκτέλεσης:



	V_1	V_2	V_3	V_4	V_5	V_6	V_7
0	$0, v_1$	$\infty, -$	$\infty, -$	$\infty, -$	$\infty, -$	$\infty, -$	$\infty, -$
1	$0, v_1$	$9, v_1$	$4, v_1$	$\infty, -$	$\infty, -$	$\infty, -$	$\infty, -$
2	$0, v_1$	$6, v_3$	$4, v_1$	$16, v_2$	$5, v_3$	$10, v_3$	$\infty, -$
3	$0, v_1$	$6, v_3$	$4, v_1$	$13, v_2$	$5, v_3$	$7, v_5$	$15, v_6$
4	$0, v_1$	$6, v_3$	$4, v_1$	$13, v_2$	$5, v_3$	$7, v_5$	$12, v_6$
5	$0, v_1$	$6, v_3$	$4, v_1$	$13, v_2$	$5, v_3$	$7, v_5$	$12, v_6$

Πολυπλοκότητα:

- Η πολυπλοκότητα του αλγορίθμου είναι O(nm) εφόσον χρησιμοποιήσουμε λίστες γειτνίασης.

Υλοποίηση σε C (βλέπε project Bellman-Ford):

```
GRAPH BellmanShortestPathTree(GRAPH g, int start)
{
   int *B, *Bnew, *P;
   int i, x, v, cnt, vertices, neighbors_length, infinity;
   elem *neighbors;
   GRAPH st;

   vertices = GR_vertices_count(g);
   GR_init(&st, vertices);

   /* B[*]: Mikos elaxistis diadromis gia na ftasoume stin korufi */
   B = (int *)malloc(sizeof(int)*vertices);
   if (!B)
   {
      printf("Error Allocating Memory!");
      exit(0);
   }
   infinity = 0;
   for (v=0; v<vertices; v++)
   {
      GR_neighbors(g, v, &neighbors_length, &neighbors);
      for (i=0; i<neighbors_length; i++)
         infinity += neighbors[i].weight;
      free(neighbors);
   }
   for (i=0; i<vertices; i++)
      B[i] = infinity;
   B[start] = 0;

   /* Bnew[i]=0 an i korufi den exei oristikopoiithei, alliws 1 */
   Bnew = (int *)malloc(sizeof(int)*vertices);
   if (!Bnew)
   {
      printf("Error Allocating Memory!");
      exit(0);
   }
   /* P[*]: proigoymenos tis korifis sti veltisti diadromi */
   P = (int *)malloc(sizeof(int)*vertices);
   if (!P)
   {
      printf("Error Allocating Memory!");
      exit(0);
   }
   for (i=0; i<vertices; i++)
      P[i] = -1;
   P[start] = start;
```

```
/* Bellman-Ford */
cnt=1;
for (i=0; i<vertices; i++) Bnew[i]=B[i];
while(cnt<vertices)
{
   for (i=0; i<vertices; i++)
   {
      GR_neighbors(g, i, &neighbors_length, &neighbors);
      for (x=0; x<neighbors_length; x++)
      {
         if (B[neighbors[x].id] + neighbors[x].weight < Bnew[i])
         {
            Bnew[i] = B[neighbors[x].id] + neighbors[x].weight;
            P[i] = neighbors[x].id;
         }
      }
      free(neighbors);
   }
   for (i=0; i<vertices; i++) B[i]=Bnew[i];
   cnt++;
```

```
      /* Just for fun - some printing */
      printf("\nStep %d", cnt);
      printf("\nB: ");
      for (i=0; i<vertices; i++)
         printf("%3d", B[i]);

      printf("\nP: ");
      for (i=0; i<vertices; i++)
         printf("%3d", P[i]);
      printf("\n");

   }
   /* construct tree */
   for (i=1; i<vertices; i++)
      GR_add_edge(st, i, P[i], GR_edge_weight(g,i,P[i]));


   free(B);
   free(Bnew);
   free(P);

   return st;
}
```

<u>Άσκηση 3: Εντοπισμός κύκλων αρνητικού βάρους</u>

Αν σε έναν γράφο υπάρχουν κύκλοι αρνητικού μήκους, τότε δεν νοούνται συντομότερα μονοπάτια. Ο αλγόριθμος Bellman-Ford μπορεί να χρησιμοποιηθεί για τον εντοπισμό κύκλων αρνητικού βάρους. Συγκεκριμένα αν:

- Υπάρχει πρόοδος (μείωση στο κόστος) τουλάχιστον ενός μονοπατιού από το βήμα n-1 στο βήμα n, τότε υπάρχει κύκλος αρνητικού βάρους.

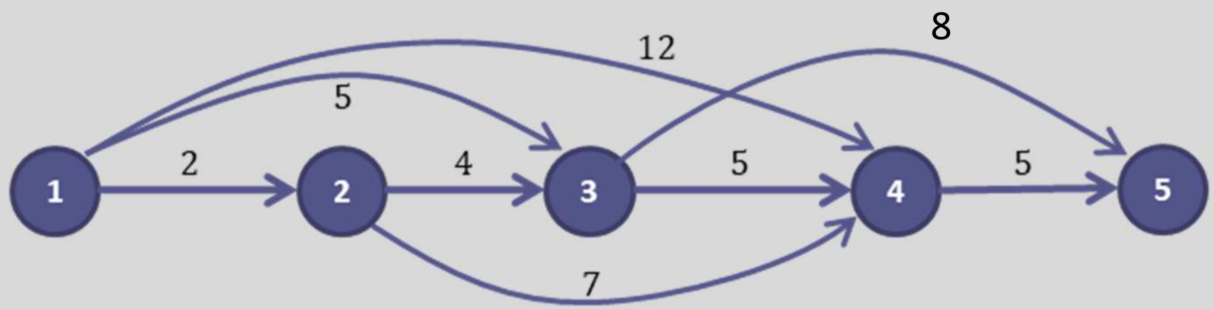
Υλοποιήστε μια συνάρτηση με όνομα negativeCycle η οποια να επιστρέφει T/F ανάλογα με το αν υπάρχει κύκλος αρνητικού βάρους στο γράφημα.

<u>Άσκηση 4: Συντομότερα Μονοπάτια σε Άκυκλα Κατευθυνόμενα Γραφήματα.</u>

Ο αλγόριθμος Bellman-Ford μπορεί να τροποποιηθεί ώστε να τρέχει σε χρόνο O(m) σε άκυκλα κατευθυνόμενα γραφήματα:

- Δεδομένης της τοπολογικής ταξινόμησης των κορυφών (μιας διάταξης, τέτοιας ώστε για κάθε ακμή (u,v) η u να προηγείται στη διάταξη της v – στο παράδειγμα η ταξινόμηση είναι 1,2,3,4,5)
- Για κάθε κορυφή v, σύμφωνα με την τοπολογική ταξινόμηση, το κόστος του συντομότερου μονοπατιού δίνεται από τη σχέση OPT[u] = OPT[v]+W[v,u] όπου v είναι η κορυφή-προηγούμενος της v με το μικρότερο OPT[v]+W[v,u].

Κατασκευάστε τη συνάρτηση DAGShortestPaths που επιστρέφει το δένδρο συντομότερων μονοπατιών από μία δεδομένη αφετηρία.



	<u>1: **0** + 2 = 2</u>	<u>1: **0** + 5 = 5</u> 2: **2** + 4 = 6	1: **0** + 12 = 12 <u>2: **2** + 7 = 9</u> 3: **5** + 5 = 10	<u>3: **5** + 8 = 13</u> 4: **9** + 5 = 14
OPT[1]=**0**	OPT[2]=**2**	OPT[3]=**5**	OPT[4]=**9**	OPT[5]=**13**