ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών

Δομές Δεδομένων και Τεχνικές Προγραμματισμού

Ενότητα 7: ΑΤΔ Γράφημα

Ιωάννης Κοτρώνης

Σχολή Θετικών Επιστημών

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Σκοποί ενότητας

- Να ορίσει τον ΑΤΔ Γράφημα.
- Να σχεδιάσει και υλοποιήσει με εναλλακτικούς τρόπους (πίνακα, λίστα γειτονικών κορυφών, λίστα ακμών)
- Να ορίσει τους δυο τρόπους αναζήτησης κατά βάθος και κατά πλάτος.
- Να ορίσει τα δένδρα επικάλυψης
- Να παρουσιάσει εφαρμογές (πίνακες μετάβασης και τον Αλγόριθμο του Dijkstra)

Περιεχόμενα ενότητας

- Γράφημα και νέες έννοιες
- Κατευθυνόμενο και μη κατεθυνόμενο
- Ορισμός (δεδομένα και Πράξεις)
- Υλοποίηση με Πίνακα, λίστα γειτονικών κορυφών, λίστα ακμών
- Αναζήτηση κατά βάθος και κατά πλάτος
- Δένδρα επικάλυψης
- Πίνακες Μετάβασης
- Αλγόριθμος Dijkstra

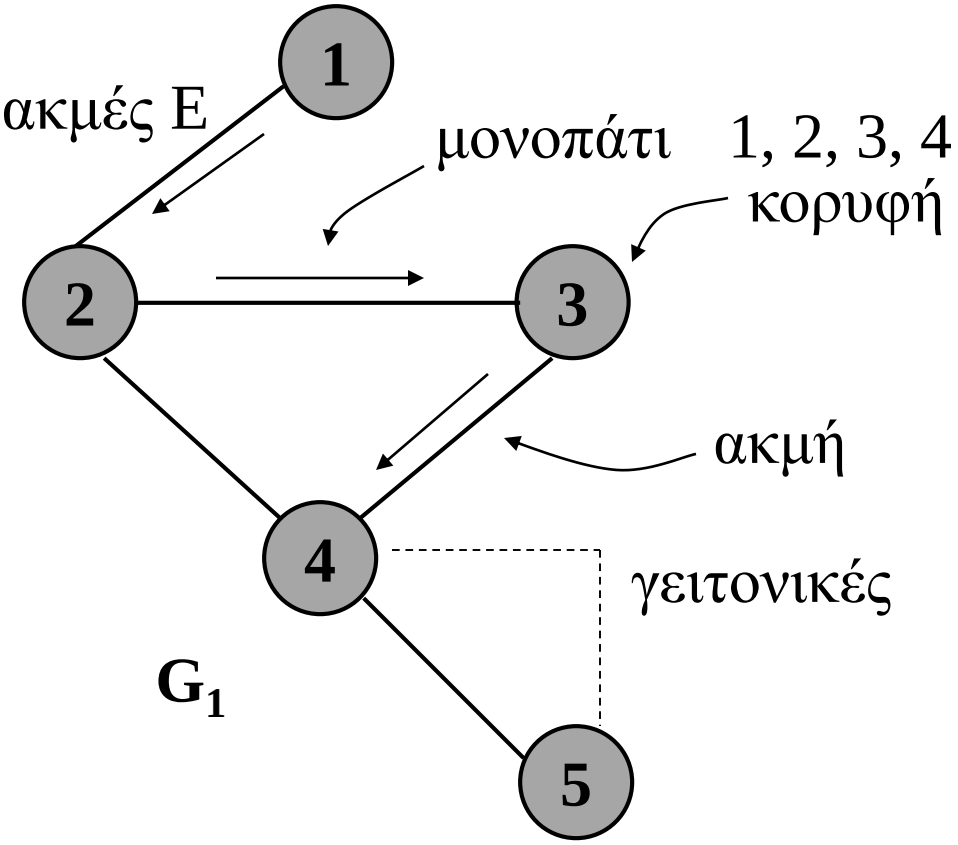
Ενότητα 7

ΑΤΔ Γράφημα

Γραφήματα (Graphs)

G = (V, E)

Κορυφές V και ακμές E



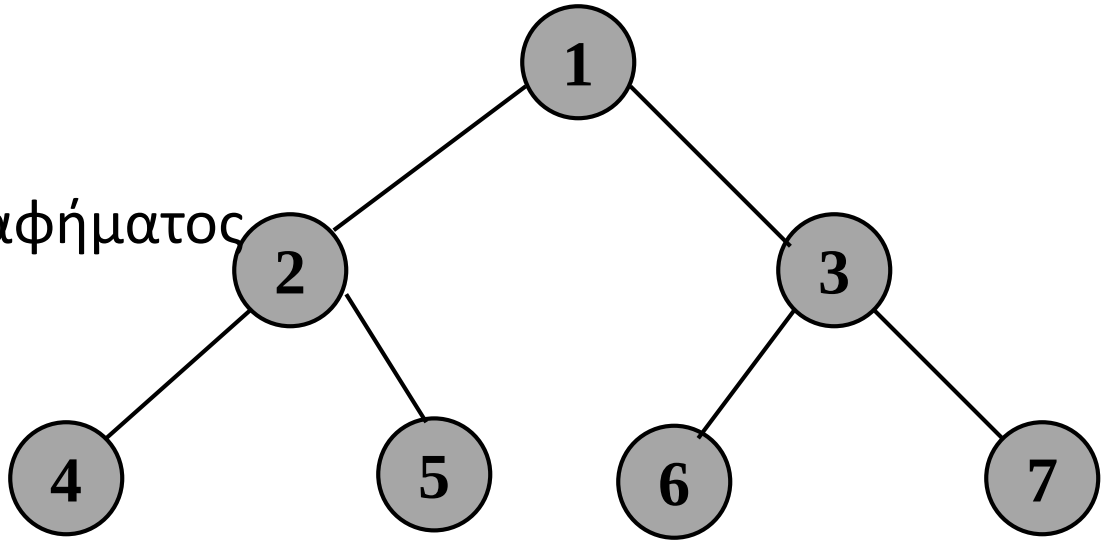
G_1

$V(G_1) = \{1,2,3,4,5\}$

$E(G_1) = \{(1,2), (2,3), (2,4), (3,4), (4,5)\}$

Το Δένδρο,
ειδική περίπτωση γραφήματος

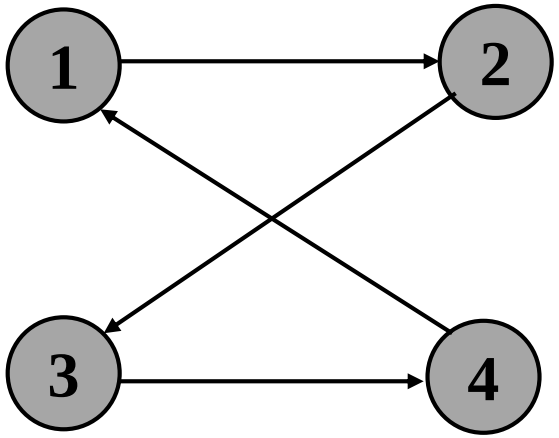
G_2



- μήκος μονοπατιού (διαδρομή από κόμβο – έως κόμβο)
- κύκλος (μονοπάτι από έναν κόμβο στον εαυτό του)
- αν δεν υπάρχει κύκλος τότε ακυκλικό (πχ. Δένδρο)
- συνδεδεμένο (αν για ζεύγη κόμβων υπάρχει μονοπάτι)
- βαθμός κόμβου (ο αριθμός των ακμών)

Κατευθυνόμενο γράφημα

ακμές με κατεύθυνση (μονόδρομος)



G_3

$V(G_3) = \{1,2,3,4\}$

$E(G_3) = \{<1,2>, <2,3>, <3,4>, <4,1>\}$

Ισχυρά συνδεδεμένο αν για κάθε ζεύγος κόμβων Vi, Vj υπάρχουν κατευθυνόμενα μονοπάτια από Vi σε Vj και απόVj σε Vi.

Το G3 είναι ισχυρά συνδεδεμένο κατευθυνόμενο γράφημα

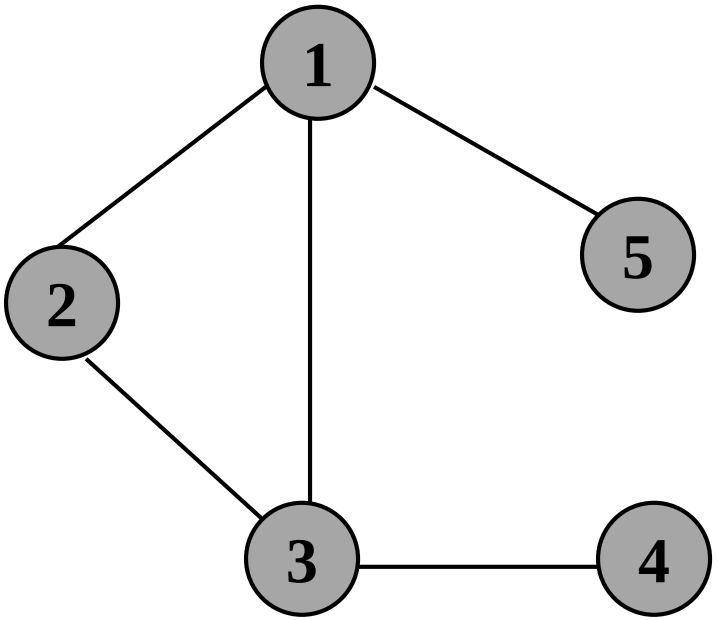
Ο ΑΤΔ Γράφημα και οι υλοποιήσεις του

Ο ΑΤΔ γράφημα G = (V, E) αποτελείται από ένα σύνολο V (κορυφές), μια σχέση E (ακμές) και πράξεις που επιδρούν πάνω στα δύο αυτά σύνολα.

Βασικές πράξεις

1. Δημιουργία γραφήματος: Δημιουργεί ένα κενό γράφημα.
2. Γειτονική κορυφή: Ελέγχει αν υπάρχει ακμή μεταξύ δύο κορυφών.
3. Ενημέρωση: Τροποποιεί το περιεχόμενο μιας κορυφής.
4. Ανάκτηση: Επιστρέφει το περιεχόμενο μιας κορυφής.
5. Διαγραφή κορυφής: Διαγράφει μια κορυφή από το γράφημα μαζί με τις ακμές που περιέχουν την κορυφή αυτή.
6. Διαγραφή ακμής: Διαγράφει μια ακμή από το γράφημα.
7. Εισαγωγή κορυφής: Εισάγει μια κορυφή στο γράφημα, χωρίς τη σύνδεσή της με άλλες κορυφές.
8. Εισαγωγή ακμής: Εισάγει μια ακμή στο γράφημα.

Υλοποίηση του ΑΤΔ Γράφημα με πίνακα



G_4

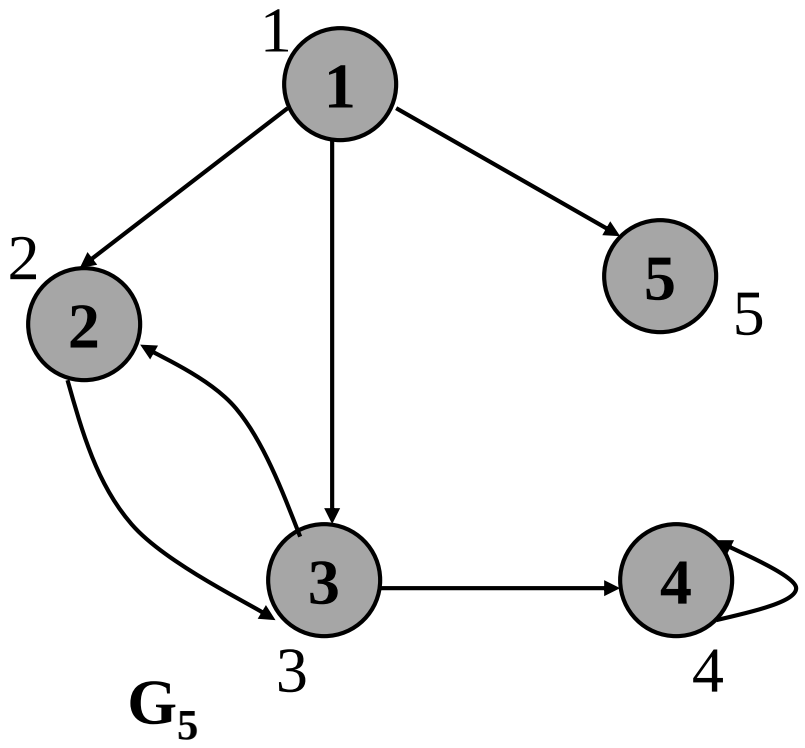
$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

κατασκευή γειτονικού πίνακα (πίνακας γειτνίασης):

αν κορυφές i,j γειτονικές A[i,j]=1 αλλιώς 0

(για μη-κατευθυνόμενα οι πίνακες είναι συμμετρικοί)

Ο γειτονικός πίνακας ενός γραφήματος δεν είναι μονοσήμαντα ορισμένος, αφού η κατασκευή του εξαρτάται από την αρίθμηση των κορυφών του γραφήματος.



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Γειτονικός πίνακας κατευθυνόμενου γραφήματος

Υλοποίηση με Πίνακα

```
#define plithos 100

typedef struct {
        int geit_pin[plithos][plithos];
        int arkrfon ;    // ο πραγματικός αριθμός κορυφών
} graphima;

graphima G;
```

```
/*Δημιουργία κενού γραφήματος (μερική απόκρυψη)*/

void dimiourgia(graphima *G){
      G->arkrfon = 0;
}
```

```
int geitoniki(graphima G, korifi p, korifi q){
/* Ελέγχει αν οι (p,q) είναι γειτονικές στο μη
      κατευθυνόμενο γράφημα G
*/

    if(      (p < 0)||(p > G.arkrfon-1)
       ||    (q < 0)||(q > G.arkrfon-1))
             return -1; /* Λάθος κορυφή */
    else
             return (G.geit_pin[p][q]);
}
```

```
int diagrafi_akmis(graphima *G, korifi p, korifi q){
/* Προ: Η (p,q) είναι ακμή του μη κατευθυνόμενου
      γραφήματος *G.
   Μετά: Αν οι κορυφές p,q δεν ανήκουν στο σύνολο των
κορυφών του γραφήματος *G τότε επιστρέφει-1, αλλιώς αν η
(p,q) είναι ακμή  του γραφήματος *G τότε διαγράφεται η
(p,q) από το *G και η συνάρτηση επιστρέφει 1, αλλιώς η
συνάρτηση   επιστρέφει 0. */

      if (   (p<0) || (p > G->arkrfon-1)
          || (q<0) || (q > G->arkrfon-1) )
            return -1; /* Λάθος κορυφές */
      else if (!geitoniki(*G,p,q,))
            return 0;
      else{
            G->geit_pin[p][q] = 0;
            G->geit_pin[q][p] = 0;
            return 1;
      }
}
```

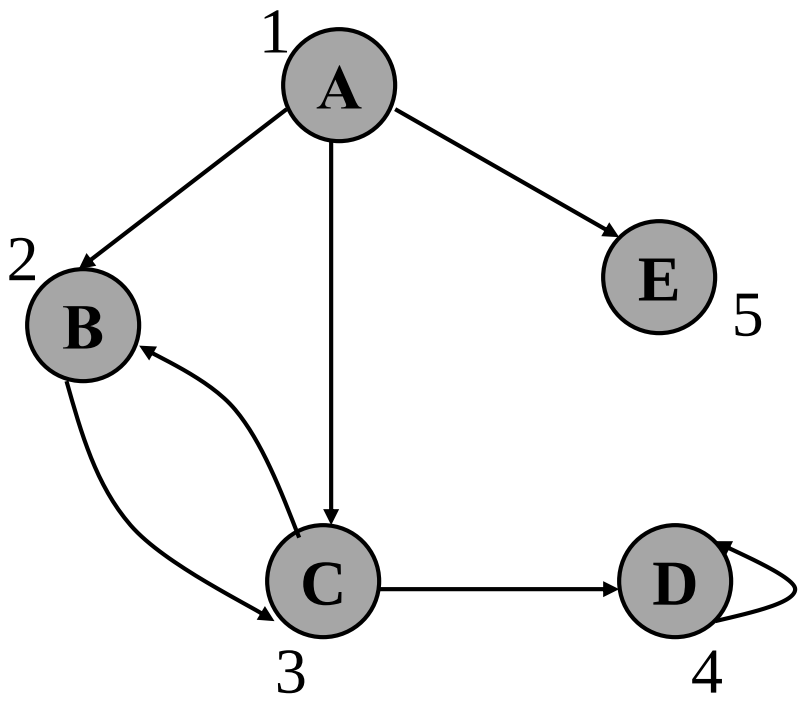
```
int eisagogi_akmis(graphima *G, korifi p,korifi q){
/* Προ: Οι κορυφές p,q ανήκουν στο σύνολο κορυφών του *G
και η ακμή (p,q) δεν ανήκει στο μη κατευθυνόμενο γράφημα.
   Μετά: Αν οι κορυφές p,q δεν ανήκουν στο σύνολο των
κορυφών του *G τότε η συνάρτηση επιστρέφει -1, αλλιώς αν
η ακμή (p,q) δεν ανήκει στο σύνολο ακμών του *G τότε
εισάγεται και η συνάρτηση επιστρέφει 1, αλλιώς η
συνάρτηση επιστρέφει 0
*/
      if    (p<0) || (p > G->arkrfon-1)
         || (q<0) || (q > G->arkrfon-1) )
            return -1; /* Λάθος κορυφές */
      else if (geitoniki(*G, p, q))
            return 0;
      else{
            G->geit_pin[p][q] = 1;
            G->geit_pin[q][p] = 1;
            return 1;
      }
}
```

```
int eisagogi_korifis(graphima *G, korifi *p){
/* Προ:Το γράφημα *G δεν έχει το μέγιστο πλήθος κορυφών.
   Μετά: Εισάγεται στο γράφημα *G η κορυφή G->arkrfon
χωρίς αυτή να συνδεθεί με άλλες κορυφές, το *p γίνεται
ίσο με αυτήν και η συνάρτηση επιστρέφει 1. Αν το *G είχε
το μέγιστο πλήθος κορυφών τότε η συνάρτηση επιστρέφει 0.
*/
      korifi i;
      if (G->arkrfon == plithos)
            return 0;
      else{
            G->arkrfon++;
            *p = (G->arkrfon)-1; /* Τελευταία κορυφή */
            for (i = 0; i <= *p; i++){
                  G->geit_pin[*p][i] = 0; /*Νέα γραμμή*/
                  G->geit_pin[i][*p] = 0; /*Νέα στήλη*/
            }
            return 1;
      }
}
```

```
int diagrafi_korifis(graphima *G, korifi p){

/* Προ : Η κορυφή p ανήκει στο σύνολο κορυφών του γραφήματος *G.

   Μετά: Αν η κορυφή p ανήκει στο σύνολο κορυφών του γραφήματος
*G τότε αυτή διαγράφεται και διαγράφονται και όλες οι ακμές που
περιέχουν την κορυφή p και η συνάρτηση επιστρέφει 1, αλλιώς η
συνάρτηση επιστρέφει 0. Μετά από αυτές τις διαγραφές το γράφημα
έχει ως κορυφές του τις 0,1,.., (G->arkrfon)-2.

*/
```

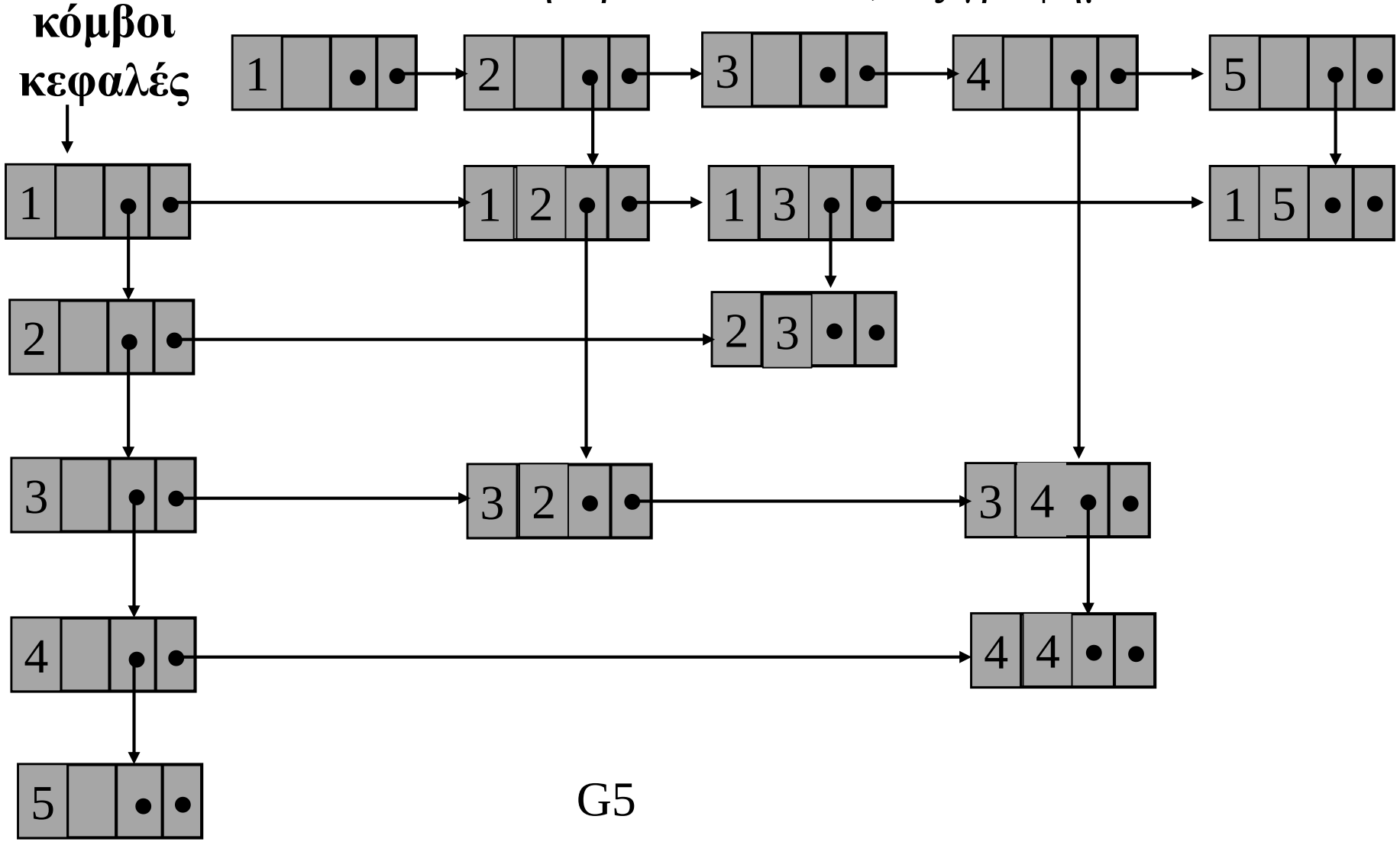
```
int diagrafi_korifis(graphima *G, korifi p) {
      korifi i,j;
      if ( (p<0) || (p > (G->arkrfon)-1) )
            return 0; /* Λάθος κορυφή */
      else {
            /* μετακίνηση i+1 γραμμής προς τα πάνω */
            for ( i = p; i <= (G->arkrfon)-2; i++)
            for ( j = 0; j <= (G->arkrfon)-1; j++)
                  G->geit_pin[i][j]=G->geit_pin[i+1][j];

            /* μετακίνηση i+1 στήλης προς τα αριστερά */
            for ( i = p; i <= (G->arkrfon)-2; i++)
            for ( j = 0; j <= (G->arkrfon)-2; j++)
                 G->geit_pin[j][i] =G->geit_pin[j][i+1];

            G->arkrfon--;
            return 1;
      }
}
```

Ορθογώνια Λίστα (αραιοί πίνακες 5-8% !=0)



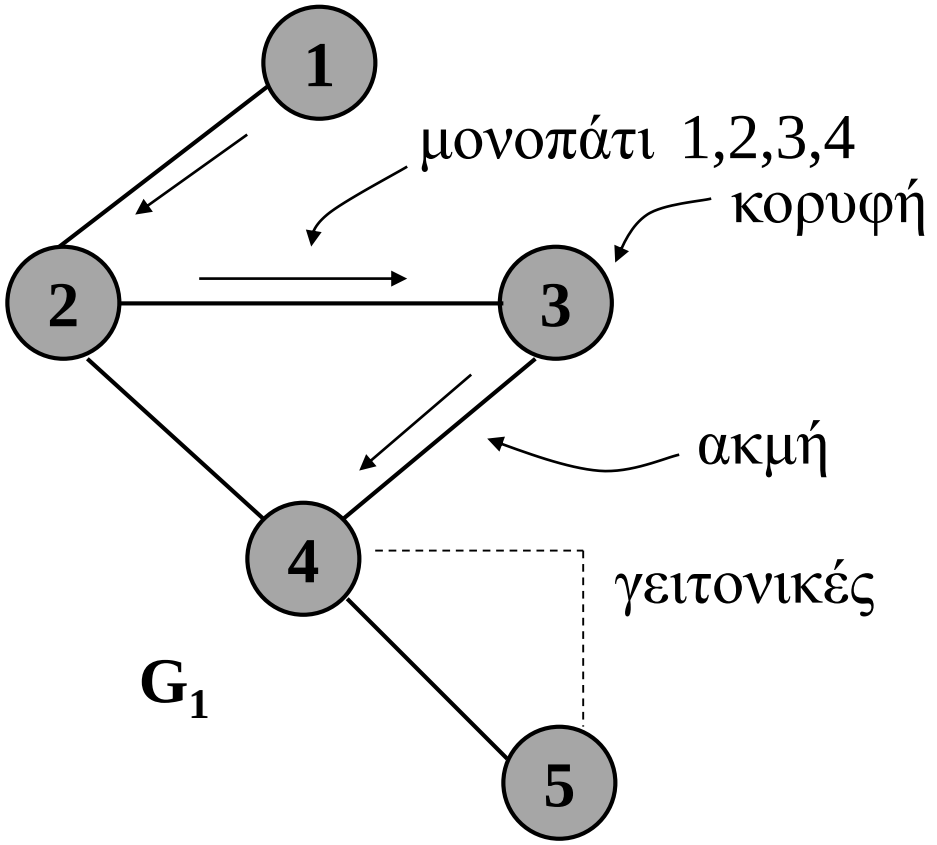
	1	2	3	4	5
1	0	1	1	0	1
2	0	0	1	0	0
3	0	1	0	1	0
4	0	0	0	1	0
5	0	0	0	0	0

Απεικόνιση Αραιού Πίνακα, ως γράφημα

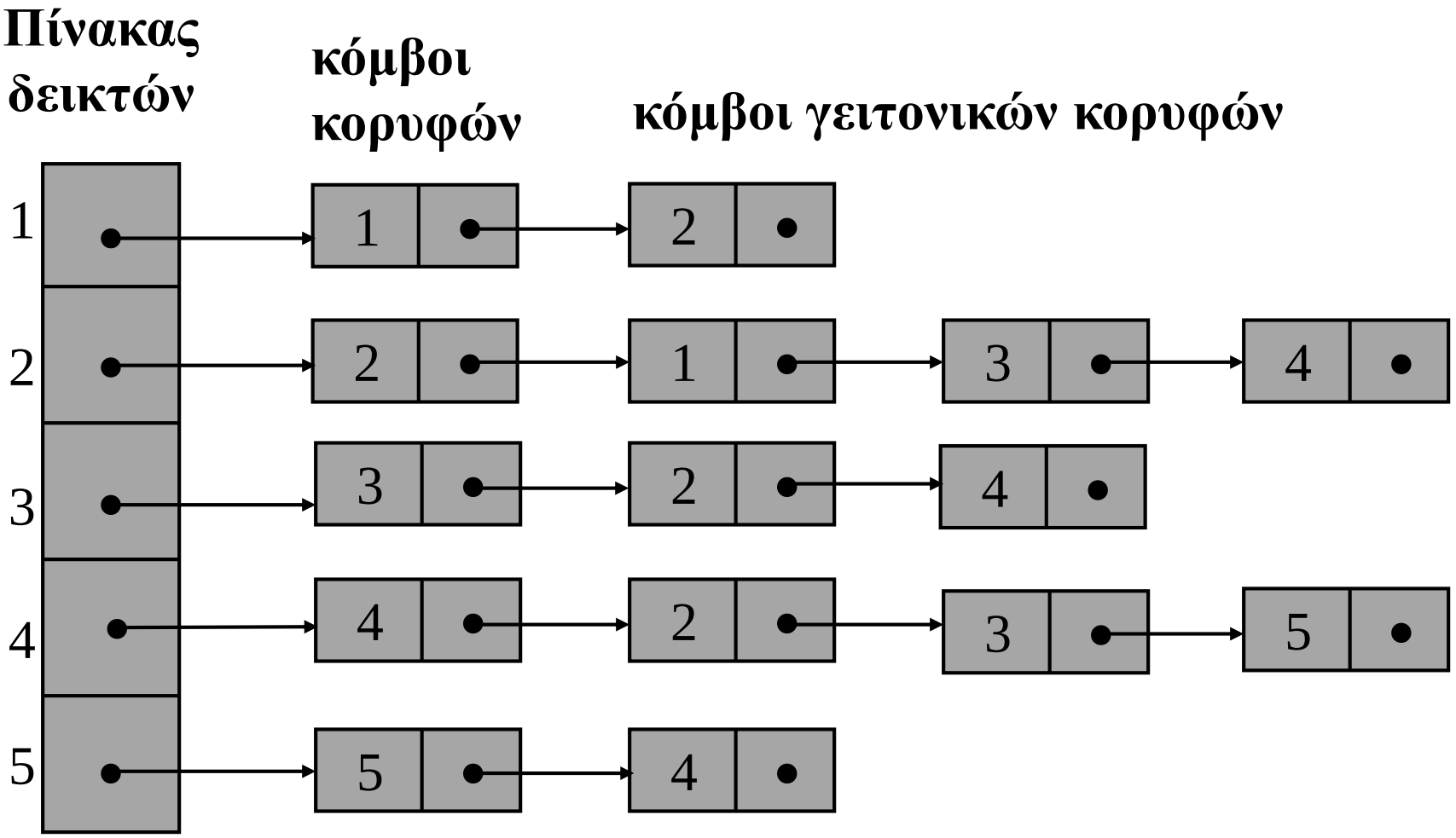


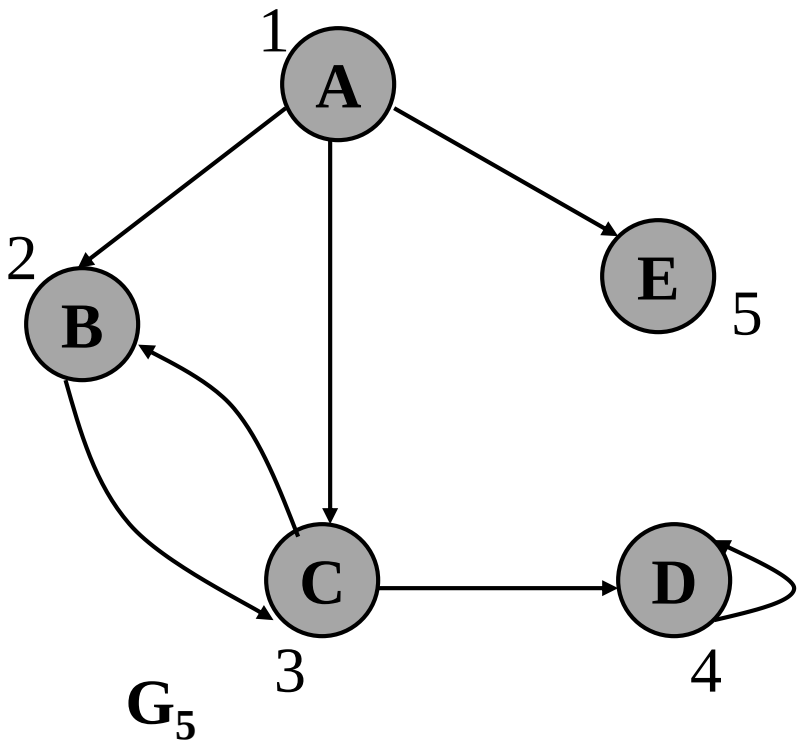
G5

Υλοποίηση του ΑΤΔ Γράφημα με συνδεδεμένες λίστες (κόμβοι και δείκτες)

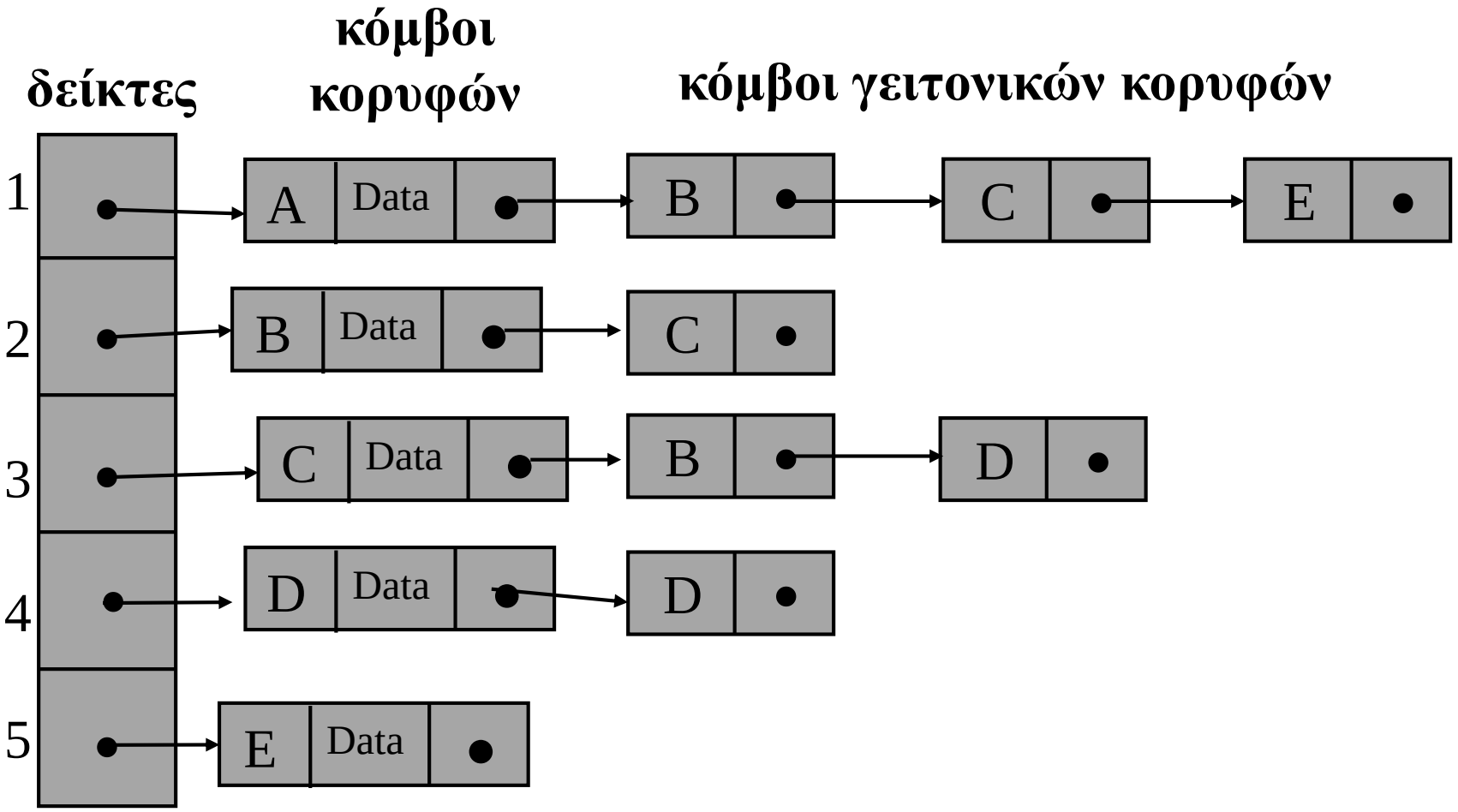


Α. Λίστα γειτονικών κορυφών του G_1




1
A
2
B
E
5
C
D
3
4
G_5

Λίστα γειτονικών κορυφών του G_5

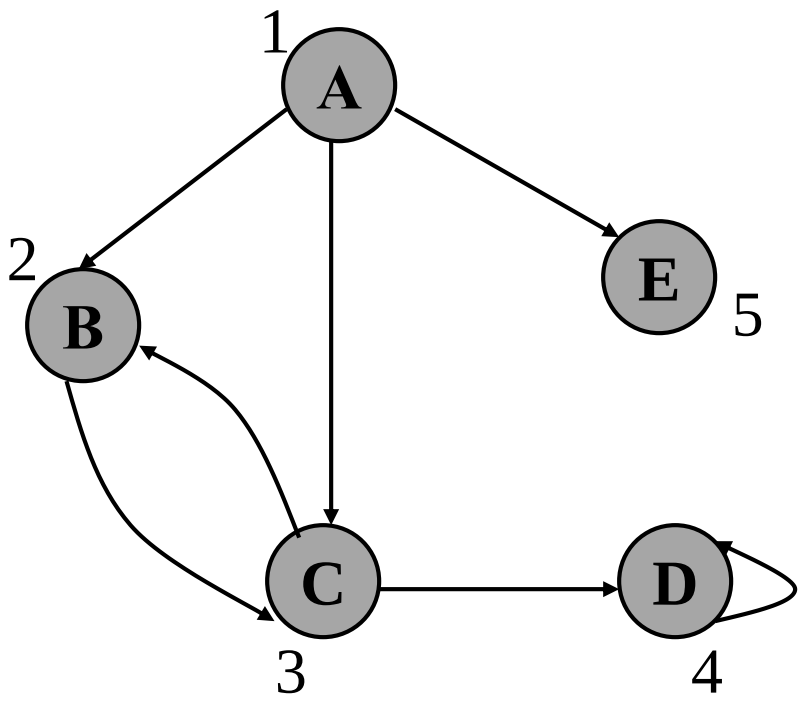


Οι δηλώσεις στην C για την παράσταση ενός γραφήματος με λίστα γειτονικών κορυφών είναι οι ακόλουθες :

```
#define plithos_korifon ...

typedef ... typos_dedomenon; /* τ.δ.κορυφών */
typedef int arithmos_korifis;

typedef struct korifi *kdktis;

typedef struct korifi{      /* κόμβος κορυφή */
     arithmos_korifis arkorifis;
     kdktis epomenos ;
};
```

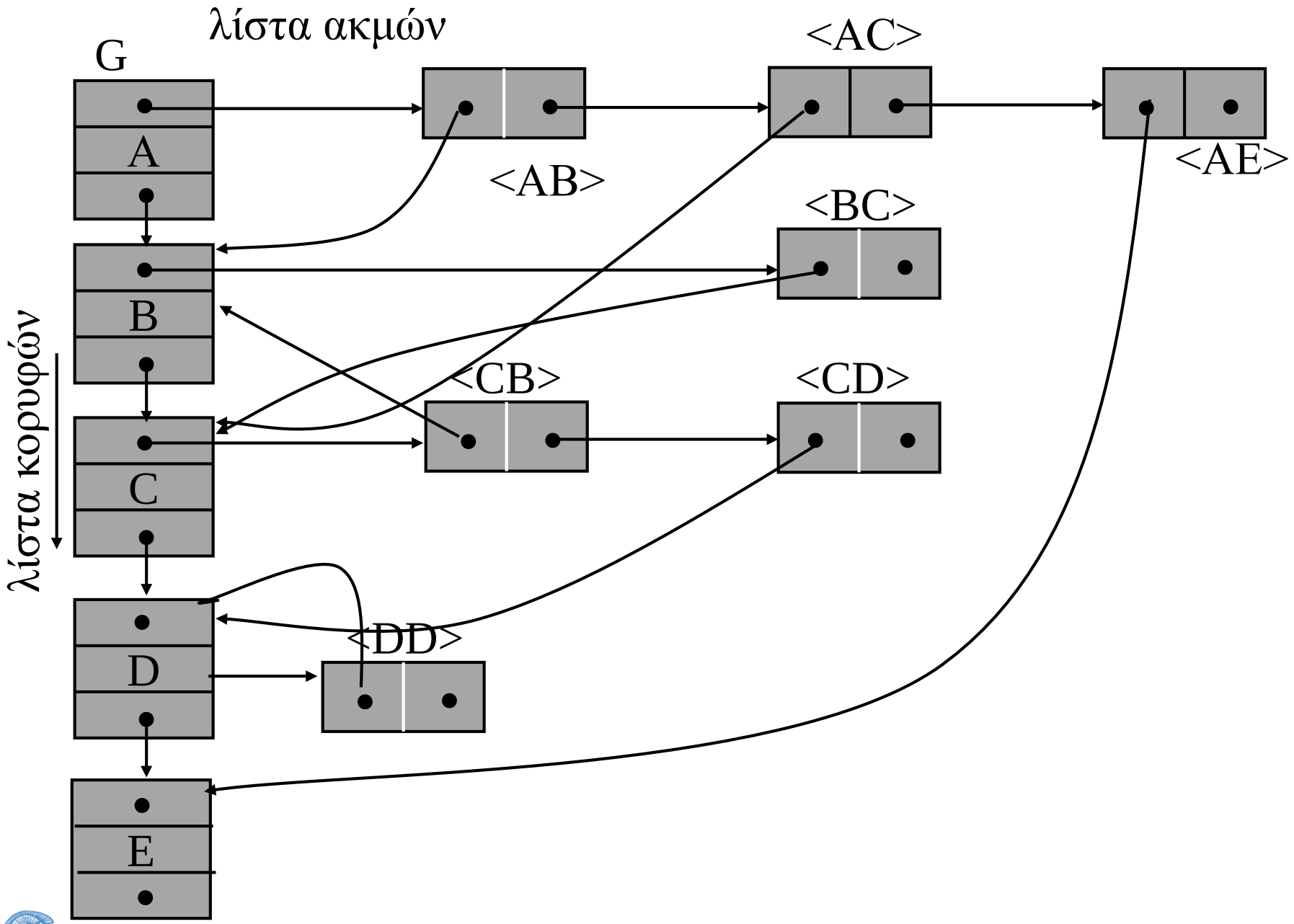
```
typedef struct kefali *kefdktis;

typedef struct kefali{
      typos_dedomenon dedomena;
      kdktis epomenos;
}; /* κόμβος – ένας για κάθε κορυφή*/

typedef struct{
      kefdktis pinakas[plithos_korifon];
      int k;     /* αριθμός κορυφών */
} graphima;

graphima G;
```

Υλοποίηση Γραφήματος με Λίστα Ακμών



	1	2	3	4	5
1	0	1	1	0	1
2	0	0	1	0	0
3	0	1	0	1	0
4	0	0	0	1	0
5	0	0	0	0	0

G5


λίστα ακμών
G
A
B
C
D
E
λίστα κορυφών
<AB>
<AC>
<AE>
<BC>
<CB>
<CD>
<DD>

```
typedef ... typos_stoixeiou; /* δεδομένα κορυφής */
typedef struct korifi *kdktis;
typedef struct akmi *adktis;

typedef struct korifi {
      typos_stoixeiou dedomena;
      kdktis epomenos;  /* επόμενος λίστας κορυφών*/
      adktis kefali;    /* δείκτης λίστας ακμών */
};
typedef struct akmi {
      typos_stoixeiou dedomena;
            /* δεδομένα ακμής π.χ βάρος, απόσταση*/
      kdktis akro;      /* άκρο ακμής */
      adktis epomenos;  /* επόμενος λίστας ακμών */
};

typedef kdktis graphima;
graphima G;
```

```
/*Δημιουργία κενού γραφήματος*/
void dimiourgia(graphima *G){
      *G = NULL;
}
```

```
void enimerosi(typos_stoixeiou stoixeio, kdktis p){
/* Μετά:Εισάγει το stoixeio στον κόμβο που δείχνει ο p.
*/
     p->dedomena = stoixeio;
}

typos_stoixeiou anaktisi(kdktis p){
/* Μετά: Η συνάρτηση επιστρέφει το περιέχομενο του
                    κόμβου που δείχνει δείκτης p. */
     return (p->dedomena);
}
```

```
int geitoniki(kdktis p, kdktis q){
      int telos;
      adktis trexon; /*για τη σάρωση της λίστας ακμών */
      int geit=0;

      trexon = p->kefali; /* αφετηρία από την κεφαλή */

      telos = 0;
      while (!telos){
            if (trexon == NULL) /* τέλος λίστας ακμών */
                  telos = 1;
            else if ( trexon->akro == q ){ /* βρέθηκε */
                        telos = 1;
                        geit = 1;
            } else /*επόμενος κόμβος λίστας ακμών */
                        trexon = trexon->epomenos;
      }
      return geit;
}
```

```
void diagrafi_korifis(kdktis p,graphima *G){
/* Μετά: Η κορυφή που δείχνει ο p διαγράφεται */
      kdktis trexon;
      trexon = *G; /* διατρέχει τη λίστα κορυφών */
      while (!keni (trexon)){
            /* δεν εξετάζεται η λίστα του p */
            if (trexon! = p) /* διαγραφή της (trexon,p)*/
                  diagrafi_komvou(p,&(trexon->kefali));
            proxorise (&trexon)/* στην επόμενη κορυφή*/
      }

      /* διαγραφή της λίστας ακμών της p */
       diagrafi_listas(&(p->kefali)); /* βοηθητική */
       diag_kor(p,G);   /*διαγραφή κορυφής p */
}
```

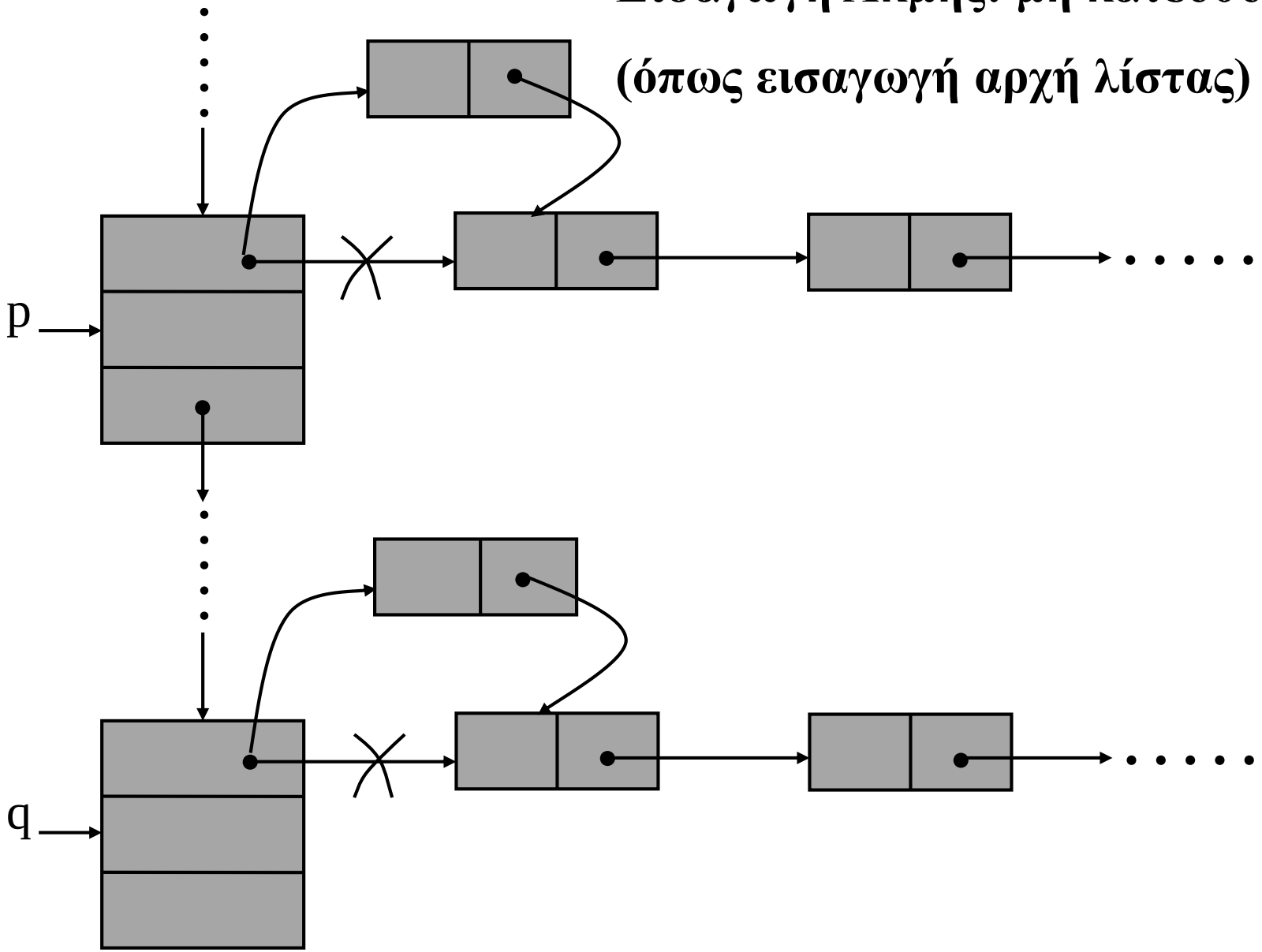
```
void diagrafi_komvou(kdktis v, adktis *kefali)}
/* Μετά: Εντοπίζει και διαγράφει τον κόμβο της λίστας
         ακμών που ξεκινά απο τον *kefali του οποίου
         το πεδίο akro είναι ίσο με v. */

      adktis pros;
      if (*kefali != NULL){
            if (((*kefali)->akro) == v)
            /* Διαγραφή του κόμβου */
            {     pros = *kefali;
                  *kefali = (*kefali)->epomenos;
                  free(pros);
            } else
            diagrafi_komvou(v, &((*kefali)->epomenos));
      }
}
```

```
void diagrafi_listas(adktis* kefali){
/* Μετά: Διαγράφει όλους τους κόμβους μιας λίστας
            ακμών ξεκινώντας από την κεφαλή. */

      if (*kefali!=NULL){
            diagrafi_listas(&((*kefali)->epomenos));
            free (*kefali);
            *kefali = NULL;
      }
}
```

```
void eisagogi_korifis(graphima *G,
                      typos_stoixeiou stoixeio)
{/* Μετά: Εισάγεται ένας νέος κόμβος στην αρχή της
        λίστας κορυφών με δεδομένα stoixeio */

      kdktis p;
      p = malloc(sizeof(struct korifi));
      /* Συνδέει το νέο κόμβο με την κεφαλή της λίστας
      κορυφών του *G */
      p->dedomena = stoixeio;
      p->epomenos = *G;
      p->kefali = NULL; /* Λίστα ακμών του p είναι κενή*/
      /* Ο κόμβος που δείχνει ο p είναι η νέα κεφαλή της
      λίστας κορυφών του *G */
      *G = p;
}
```

Εισαγωγή Ακμής: μη κατευθυνόμενο

(όπως εισαγωγή αρχή λίστας)

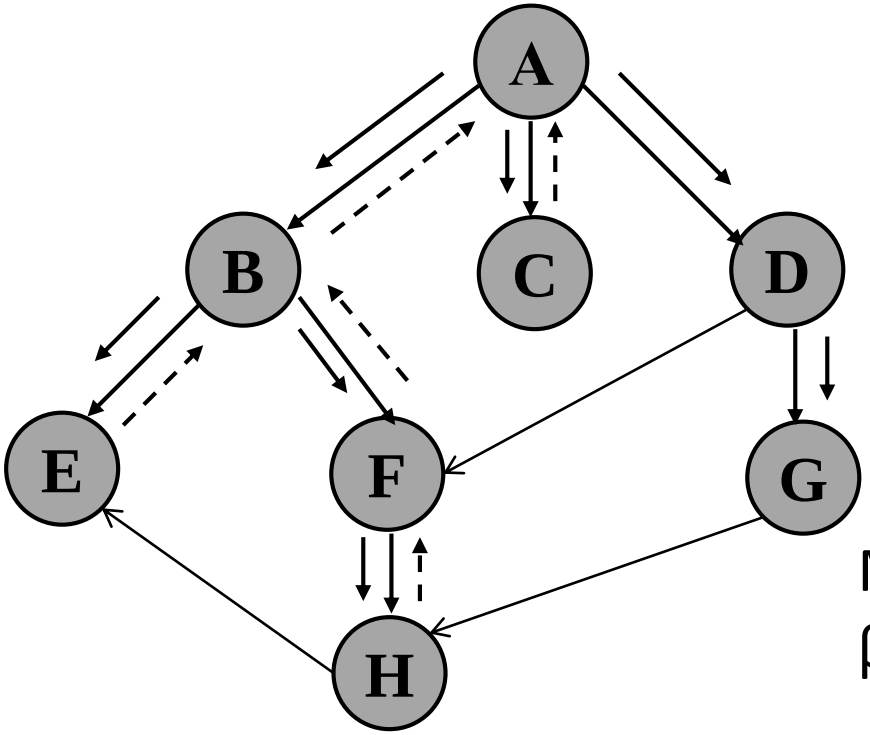


```
void eisagogi_akmis(kdktis p,kdktis q){
/* Μετά: Εισάγεται μια ακμή μεταξύ των κορυφών που
 δείχνουν οι p και q, σε ένα μη κατευθυνόμενο γράφημα.
*/
      adktis pros;
      if (!geitoniki(p,q)){
            pros = p->kefali;
            p->kefali = malloc(sizeof(struct akmi));
            /* Εισαγωγή στην αρχή & σύνδεση με q */
            p->kefali->akro = q;
            p->kefali->epomenos = pros;

            /* Επαναλαμβάνονται τα παραπάνω για το q */
            pros = q->kefali;
            q->kefali = malloc(sizeof(struct akmi));
            q->kefali->akro = p;
            q->kefali->epomenos = pros;
      }
}
```

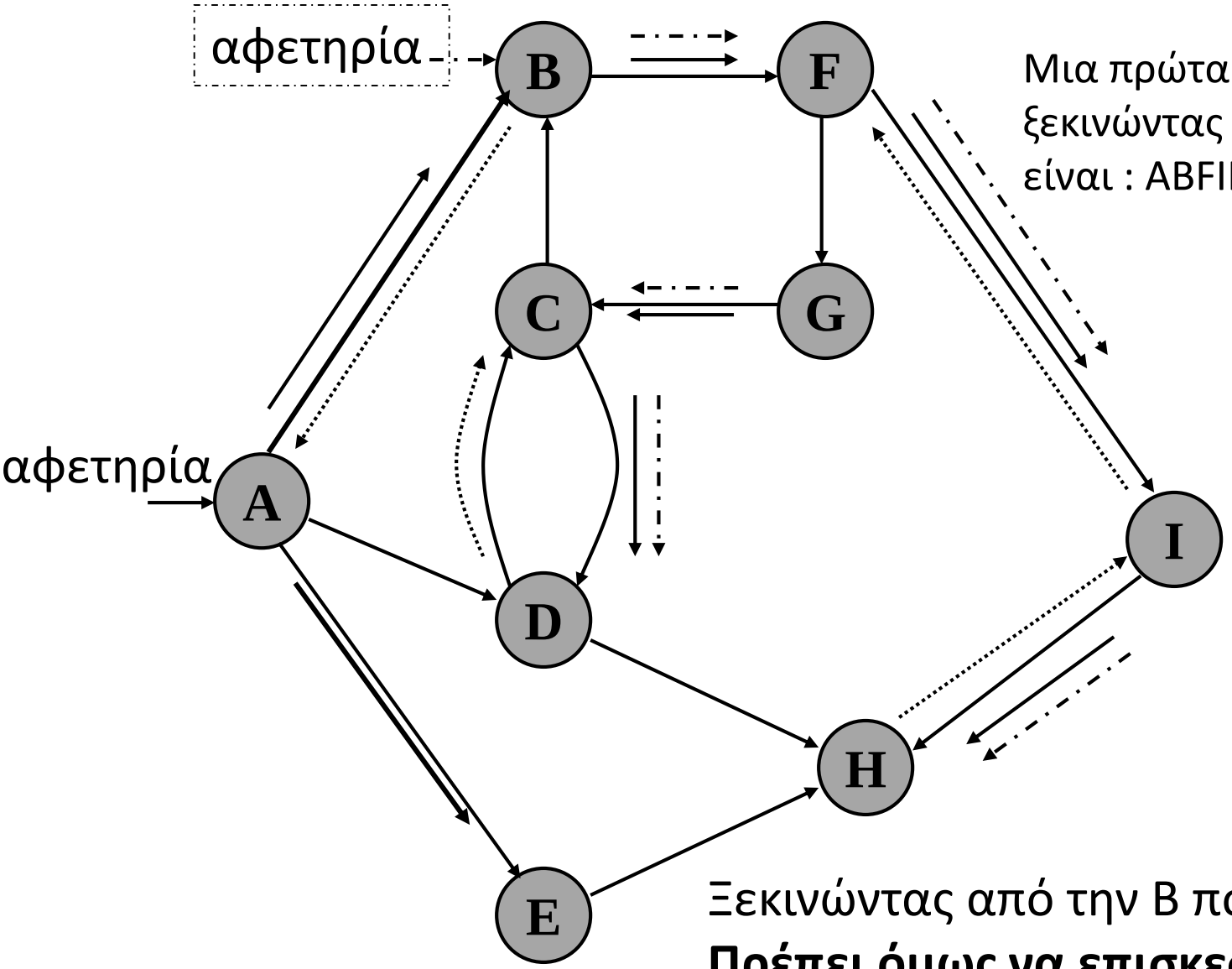
Διαδρομή (ή Αναζήτηση) Γραφημάτων

- Διαδρομή πρώτα κατά βάθος
- Διαδρομή πρώτα κατά πλάτος

Διαδρομή **πρώτα κατά βάθος**



Μια διαδρομή πρώτα κατά βάθος είναι ABEFHCDG



Μια πρώτα κατά βάθος σειρά, ξεκινώντας από την κορυφή Α, είναι : ABFIHGCDE

Ξεκινώντας από την Β ποτέ σε Α και Ε.
Πρέπει όμως να επισκεφτούμε όλες.

Αλγόριθμος Διαδρομής πρώτα κατά βάθος (ισχυρά συνδεδεμένου)

Επισκέπτεται όλες τις κορυφές ενός γραφήματος που είναι συνδεδεμένες με την κορυφή v από την οποία ξεκινά, με την πρώτα κατά βάθος αναζήτηση

1. Επίσκεψη της αρχικής κορυφής v.

2. Για κάθε γειτονική κορυφή w της v γίνονται τα εξής:
Αν δεν έχουμε επισκεφτεί την w, εφαρμόζεται ο αλγόριθμος της αναζήτησης πρώτα κατά βάθος με αρχική κορυφή την w.

2 προβλήματα

Π1. Ενα γράφημα μπορεί να περιέχει κύκλους, πράγμα που σημαίνει ότι είναι δυνατόν να επισκεφθούμε μία κορυφή περισσότερες από μία φορές. Για αυτό μαρκάρουμε τις κορυφές που επισκεπτόμαστε και ελέγχουμε αν τις έχουμε ξαναεπισκεφθεί.

Π2. Το άλλο φαινόμενο που μπορεί να παρουσιαστεί είναι εκείνο του μη συνδεδεμένου (για μη κατευθυνόμενο) ή μη ισχυρά συνδεδεμένου (για κατευθυνόμενο) γραφήματος. Για αυτό ξεκινάμε Διαδρομή από κάθε κορυφή του γραφήματος και αν δεν την έχουμε ξαναεπισκεφθεί ξεκινάμε νέα Διδρομή.

Η Διαδρομή πρώτα κατά βάθος περιγράφεται από τα παρακάτω δυο υποπρογράμματα σε ψευδογλώσσα για να καλύπτει αφαιρετικά και τις τρεις υλοποιήσεις των γραφημάτων. Στην συνέχεια τα εξειδικεύουμε για κάθε υλοποίηση.

```
/* ψευδοκώδικας κατά βάθος για όλες τις υλοποιήσεις */

int episkeftike[plithos]; /* Πίνακας Επισκέψεων */

void vathos(graphima G){
      int i;
      for (i=0;i<plithos;i++)do episkeftike[k] = 0;

      Για κάθε κορυφή στο G          /* ψευδοκώδικας */
            if (!episkeftike[k])
                  diadromi_vathos(k);
}
```

Προσοχή! Ξεκινάμε κατά βάθος από κάθε κορυφή. Αν την έχουμε επισκεφτεί έχοντας ξεκινήσει από άλλη κορυφή, δεν την επισκεπτόμαστε. Επισκεπτόμαστε ΟΛΟΥΣ τους κόμβους.

/*Προ: k είναι κορυφή του γραφήματος G.
Μετά: Εχει εκτελεστεί η πρώτα κατά βάθος διαδρομή, με τη χρήση του υποπρογράμματος episkepsi για την k και για όλες τις γειτονικές της κορυφές.*/

```
/* Αναδρομικός
      ψευδοκώδικας για όλες τις υλοποιήσεις*/

void diadromi_vathos(korifi k){
      korifi w;
      episkepsi(k);
      episkeftike[k] = 1;

      Για κάθε w γειτονικό του k  /* ψευδοκώδικας */
            if (!episkeftike[w])
                  diadromi_vathos(w);
}
```

Εξειδίκευση:
η υλοποίηση της Διαδρομής πρώτα κατά βάθος στην περίπτωση όπου το γράφημα παριστάνεται με γειτονικό πίνακα.

```
int episkeftike[plithos];
/*Προ: Εχει δημιουργηθεί ο γειτονικός πίνακας.
  Μετά: Επίσκεψη όλων των κορυφών του γραφήματος με τη
       μέθοδο της αναζήτησης πρώτα κατά βάθος*/

void vathosP(graphima G){
      korifi i;
      for (i=0; i<=(G.arkrfon-1); i++)
            episkeftike[i] = 0;

      for (i=0; i<=(G.arkrfon-1); i++)
            if (!episkeftike[i])
                  diadromi_vathosP(G,i);
}
```

```
/*Προ: i είναι μια κορυφή του γραφήματος.
Μετά: Η επίσκεψη της κορυφής i και των γειτονικών της
έχειτελειώσει με την αναζήτηση πρώτα κατά βάθος*/

void diadromi_vathosP(graphima G, korifi i){
      int j;

      episkepsi(i); /*επίσκεψη της i κορυφής*/
      episkeftike[i] = 1;

      for (j=0; j<= G.arkrfon-1; j++)
            if (geitoniki(G,i,j) && (!episkeftike[j]))
                  diadromi_vathosP(G,j);
}
```

Εξειδίκευση: το γράφημα παριστάνεται με τη λίστα ακμών

Κάθε κόμβος περιέχει ένα boolean πεδίο episkeftike, το οποίο αρχικά έχει τεθεί ίσο με false για κάθε κορυφή.

Η diadromi τροποποιείται ως εξής:

```
void diadromi_vathosLA(kdktis p){
/* Ο δείκτης p δείχνει την κορυφή αφετηρίας */

      adktis trexon;
      kdktis q;
      episkepsi(p);
      p->episkeftike = 1;

      trexon=p->kefali;
      while (trexon!=NULL){
            q=trexon->akro; /* η q γειτονική της p */
            if (!(q->episkeftike))
                  diadromi_vathosLA(q);
            trexon = trexon->epomenos;
      }
}
```

Πολυπλοκότητα Διαδρομής κατά βάθος

- Γειτονικός Πίνακας : $O(n^2)$, επειδή $O(n)$ για γειτονικές x n
- Λίστα γειτονικών κορυφών : $O(e)$ (το πολύ 2e κόμβοι)
- Λίστα ακμών : $O(n+e)$
  Συνήθως $e << n^2$

Οι υλοποιήσεις με λίστες έχουν πιο αποδοτικές αναζητήσεις.

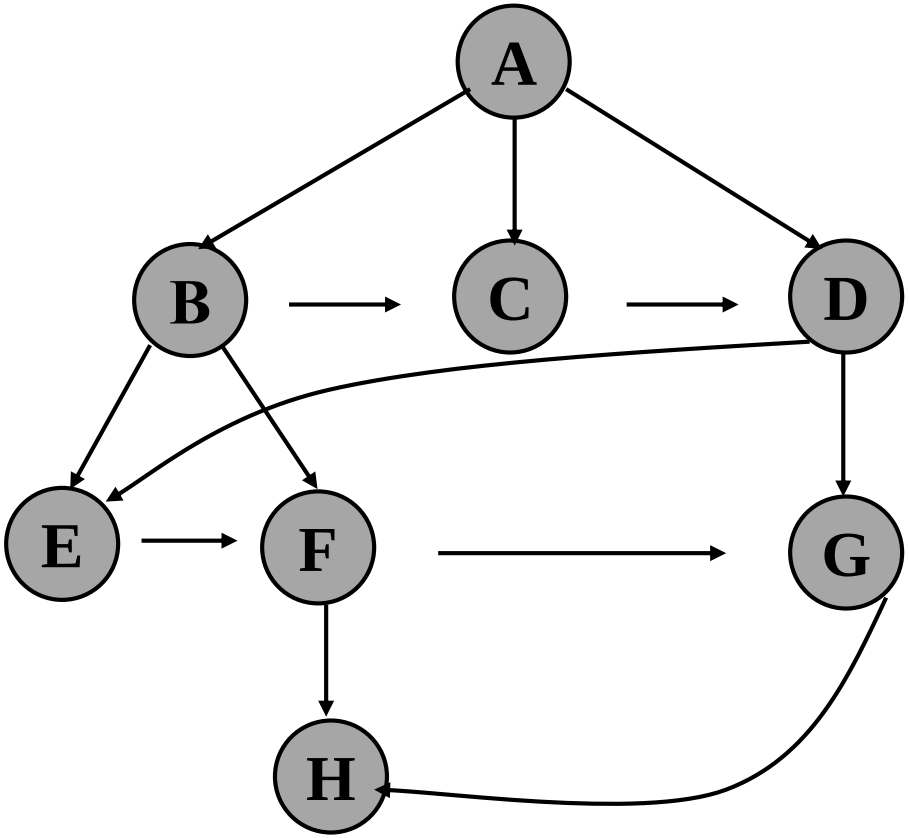
Η υλοποίηση με πίνακα χρησιμοποιείται σε αλγεβρικούς υπολογισμούς με πίνακες (μετάβασης).

Αναζήτηση πρώτα κατά πλάτος

Η αναζήτηση πρώτα κατά πλάτος ενός γραφήματος είναι ανάλογη με την επίσκεψη των κόμβων ενός δυαδικού δέντρου κατά επίπεδα ή βαθμό απομάκρυνσης από τον αρχικό κόμβο.

1. Τοποθετούμε αρχική κορυφή σε ουρά
2. Εξάγουμε την πρώτη κορυφή από την ουρά και την επισκεπτόμαστε
3. Τοποθετούμε κάθε γειτονική της κορυφή, αν δεν την έχουμε επισκεφτεί ήδη, στην ουρά.
4. και πάλι το 2.

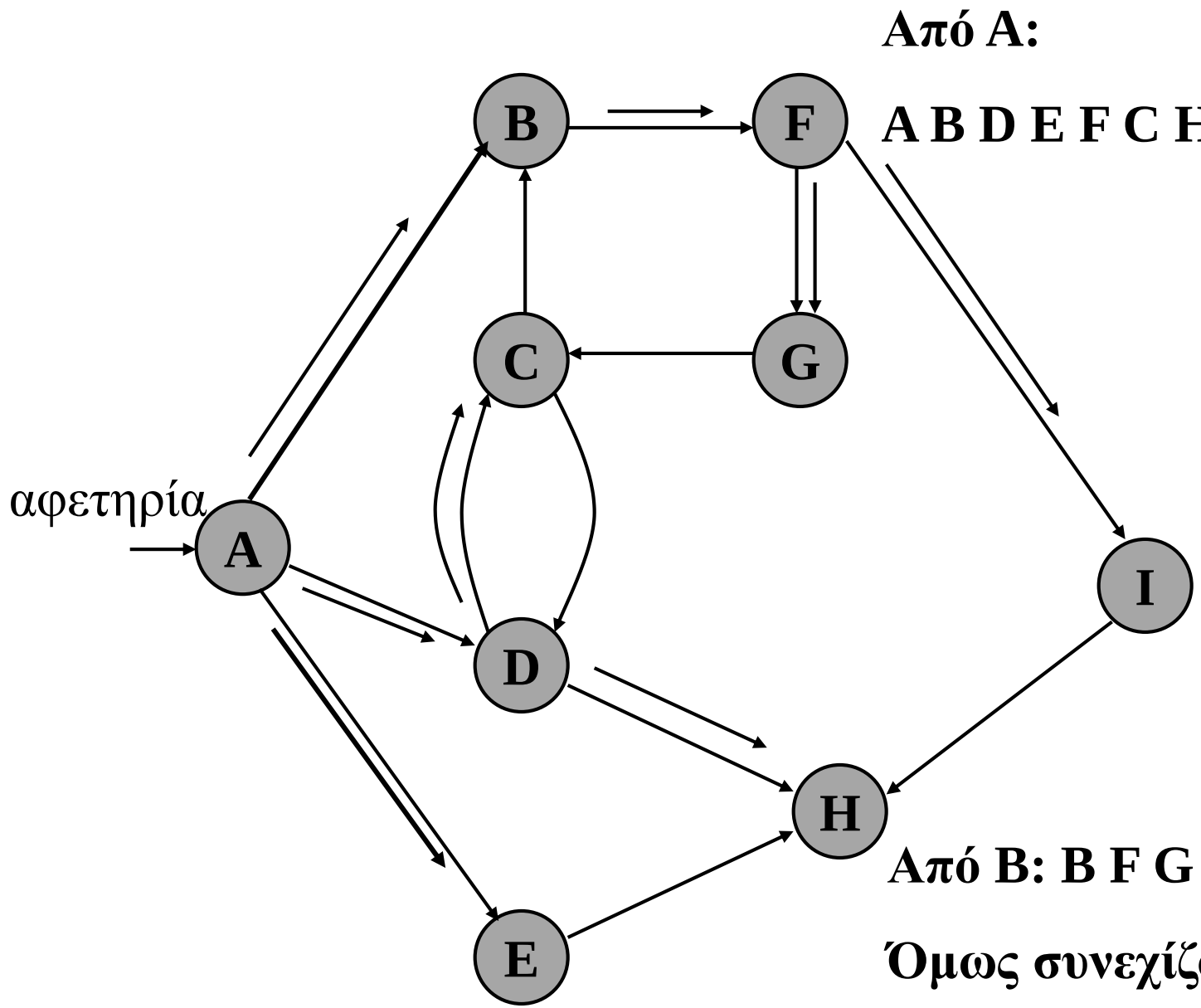
Αναζήτηση πρώτα κατά πλάτος



A B C D E F G H

Ουρά

	A
A	B-C-D
B	C-D-E-F
C	D-E-F
D	E-F-G
E	F-G
F	G-H
G	H
H	


Από Α:
Α Β D Ε F C H G I (όλες)
αφετηρία
A
B
C
D
E
F
G
H
I
Από Β: B F G I H C D
Όμως συνεχίζουμε από Α Ε

Ο αλγόριθμος της αναζήτησης πρώτα κατά πλάτος είναι παρόμοιος με εκείνον της αναζήτησης πρώτα κατά βάθος με τη διαφορά ότι τώρα προστίθεται η δημιουργία της ουράς των κορυφών και καλεί diadromi_platos .

```
typedef korifi typos_stoixeiou;
typos_oura oura_korifon;
int episkeftike[plithos];

void platos(graphima G){
      korifi k;
      for all k in G do
            episkeftike[k] = 0;

      dimiourgia(&oura_korifon);

      for all k in G do
            if (!episkeftike[k])
                  diadromi_platos(k);
}
```

Όλες οι αλλαγές βρίσκονται στον αλγόριθμο του τρόπου επίσκεψης των κορυφών. Πιο συγκεκριμένα ο αλγόριθμος της διαδρομής των κορυφών του γραφήματος τροποποιείται στον παρακάτω:

Επαναληπτικός αλγόριθμος

```
void diadromi_platos(korifi k){
/*Προ: k είναι η κορυφή του γραφήματος.
  Μετά:  Έχει εκτελεστεί η κατά πλάτος διαδρομή */

      korifi w,i;
      episkeftike[k] = 1;
      prosthesi(&oura_korifon,k);

      while (!keni(oura_korifon)){
            apomakrynsi(&oura_korifon,&i);
            episkepsi(i);
            for all w adjacent to i do /* ψευδοκώδικας */
                  if (!episkeftike[w]){
                        episkeftike[w] = 1;
                        prosthesi(&oura_korifon,w);
                  }
      }
}
```

Εξειδίκευση: το γράφημα παριστάνεται με τη λίστα ακμών, τότε υποθέτουμε ότι κάθε κόμβος περιέχει ένα boolean πεδίο episkeftike, το οποίο αρχικά έχει τεθεί ίσο με false για κάθε κορυφή. Η δε διαδρομή τροποποιείται ως εξής:

/*Ο δείκτης p δείχνει την κορυφή-αφετηρία.
Προ: Το γράφημα G έχει δημιουργηθεί.
Μετά: Εχει εκτελεστεί η κατά πλάτος διαδρομή όλων των κορυφών του γραφήματος.
Χρήση: Οι βασικές πράξεις της ουράς.*/

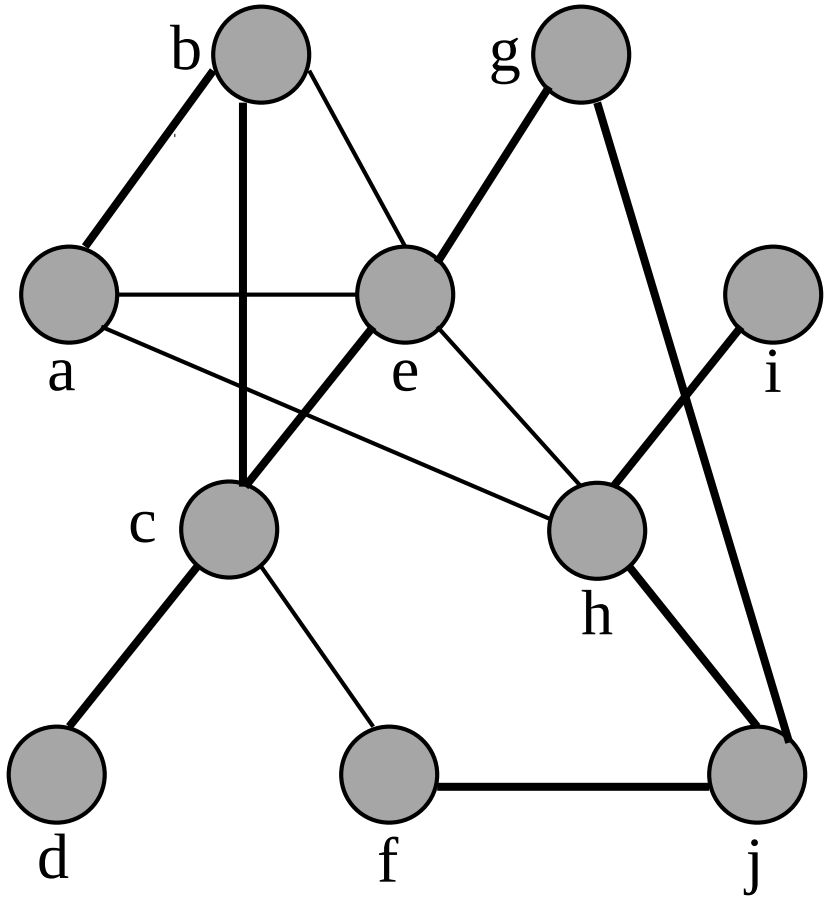
```
void diadromi_platosLA(kdktis p){
      adktis trexon;
      kdktis q,i;
      p->episkeftike = 1;
      prosthesi(&oura_korifon, p);

      while (!(keni(oura_korifon))){
            apomakrynsi(&oura_korifon, &i);
            episkepsi(i);
            trexon=i->kefali;
            while (trexon!=NULL){
                  q=trexon->akro;/* q γειτονική p*/
                  if (!(q->episkeftike)){
                        q->episkeftike = 1;
                        prosthesi(&oura_korifon,q);
                  }
                  trexon=trexon->epomenos;
            }
      }
}
```

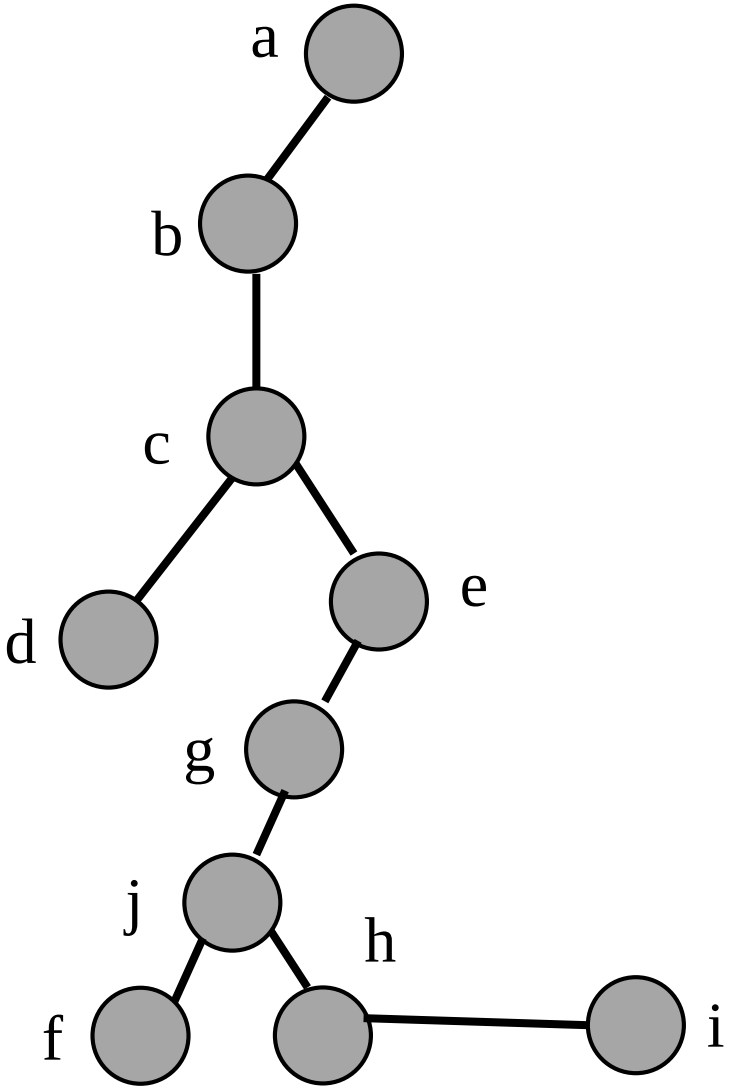
Δέντρα Επικάλυψης (Spanning Trees)

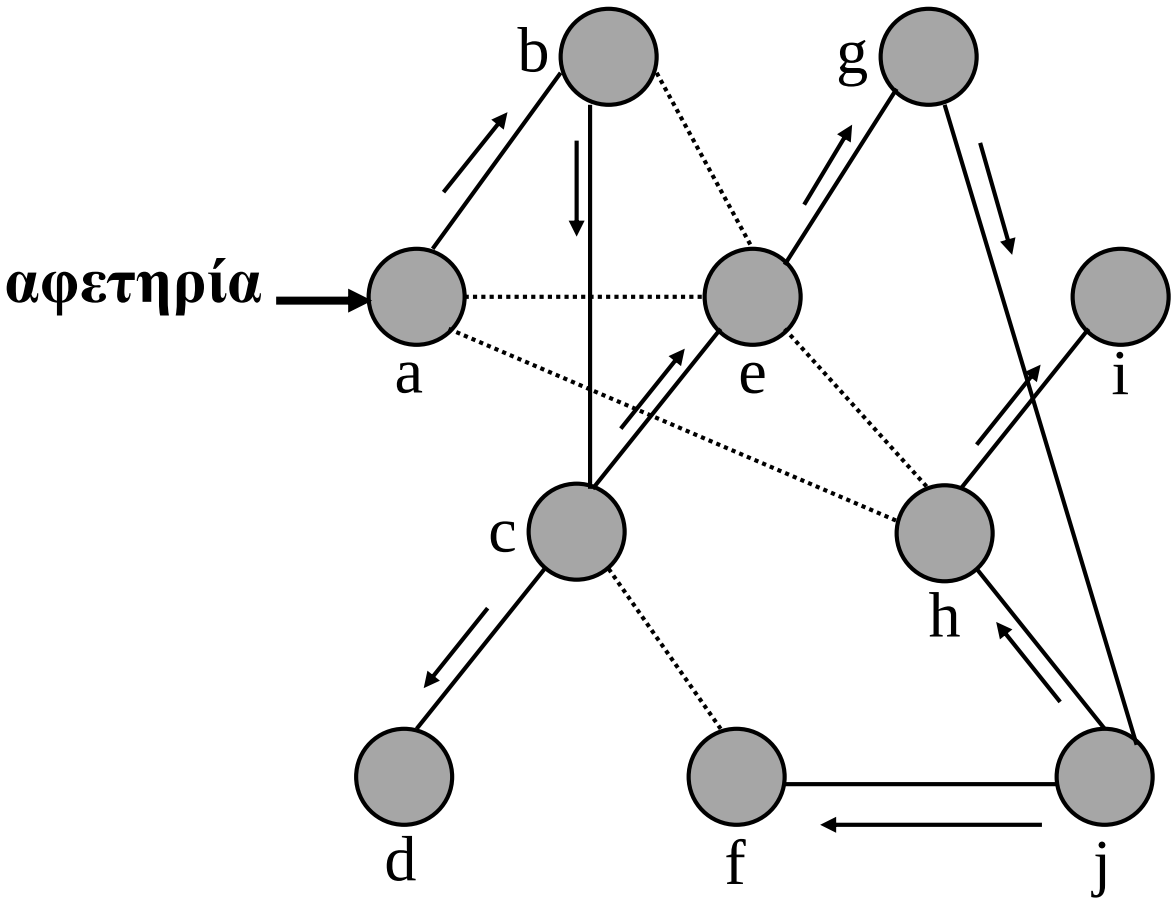
Ενα δέντρο επικάλυψης για ένα γράφημα G, είναι ένα δέντρο που έχει δημιουργηθεί από μερικές ακμές του G έτσι ώστε να περιέχει όλες τις κορυφές του.

Με άλλα λόγια ένα δέντρο επικάλυψης είναι ένα γράφημα με το ελάχιστο πλήθος ακμών που επιτρέπει την επικοινωνία μεταξύ οποιονδήποτε κορυφών του γραφήματος.

Ενα δέντρο επικάλυψης με πρώτα κατά βάθος επίσκεψη των κορυφών του. Αφετηρία η κορυφή a.






αφετηρία
a
b
c
d
e
f
g
h
i
j

Στη συνέχεια η vathos τροποποιείται κατάλληλα για τον εντοπισμό των ακμών ενός **πρώτα κατά βάθος δέντρου επικάλυψης**.

```
/*Δημιουργεί ένα κατά βάθος δέντρο επικάλυψης*/
void vathos_spantree(graphima G){
      kdktis p;
      adktis e;
      p  = G;
      /*Αρχικές τιμές – δίνονται αρχικές τιμές σε όλα τα
σημάδια επισκέπτονται όλες οι κορυφές του G και
σημαδεύεται η κάθε ακμή ώστε αρχικά να μην ανήκειστο
δέντρο επικάλυψης*/
```

συνέχεια

```
        while (p!=NULL){
                /*αρχικά οι κορυφές δεν έχουμε επισκεφτεί
                p->episkeftike = 0;
                e = p->kefali;
                /*επισκέπτεται τη λίστα ακμών για κάθε κορυφή
                καισημαδεύεται η κάθε ακμή, ώστε αρχικά να
                μην ανήκει στο δέντρο επικάλυψης*/

                while (e!=NULL){
                        e->simadi = 0;
                        e = e->epomenos;
                }
                p = p->epomenos;
        }
        p = G;
        vathos_dimiourgia(p);
}
```

```
void vathos_dimiourgia(kdktis p){
      adktis trexon;
      kdktis q;
      p->episkeftike = 1;
      trexon = p->kefali;

      while (trexon!=NULL){
            q= trexon->akro;
            if (!(q->episkeftike)){
                  /* σημαδεύεται η ακμή του δέντρου*/
                  trexon->simadi = 1;
                  /*θα μπορούσαμε να τυπώσουμε την ακμή
                  του δέντρου στο σημείο*/
                  vathos_dimiourgia(q);
            }
            trexon = trexon->epomenos;
      }
}
```

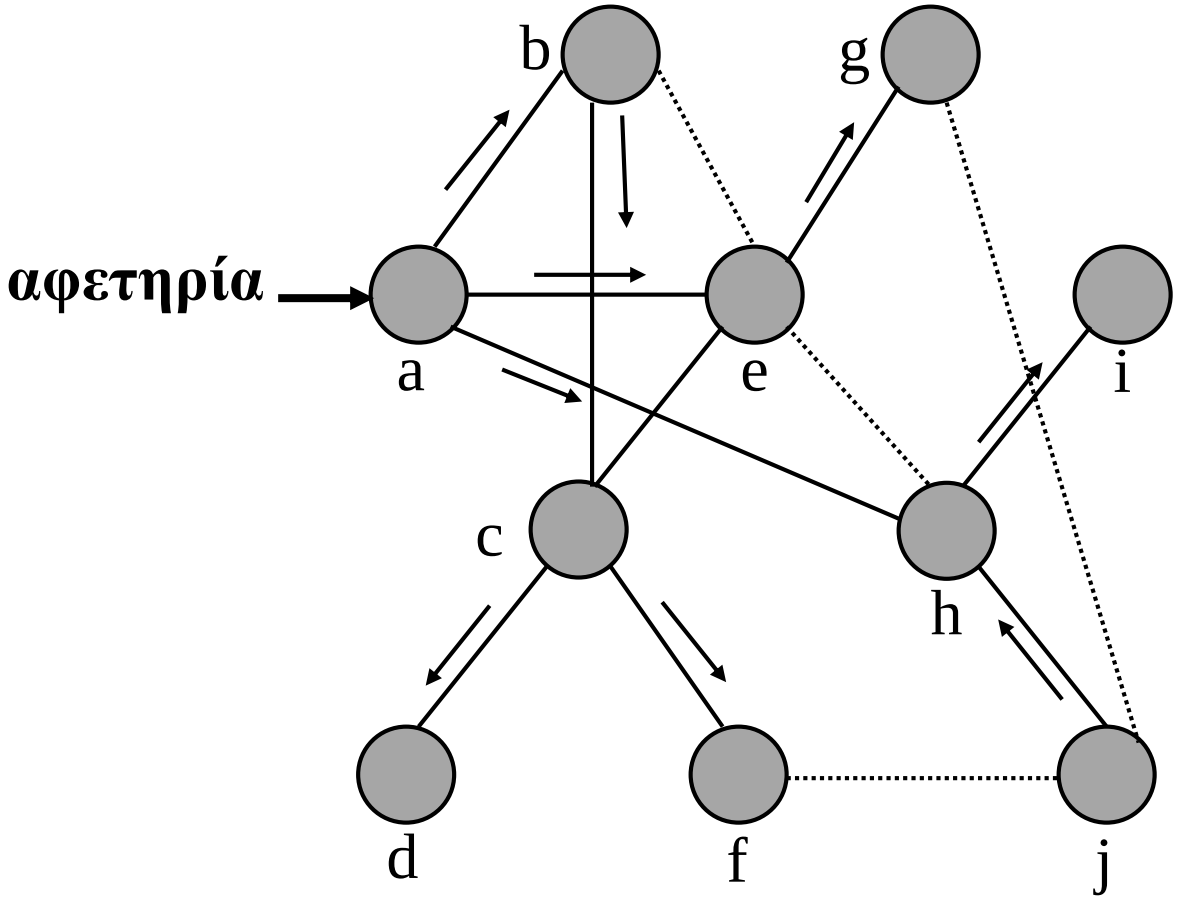
Με όμοιο τρόπο τροποποιείται η platos και κατασκευάζεται ένα πρώτα κατά πλάτος δέντρο επικάλυψης ενός γραφήματος G.

```
void platos_spantree(graphima G){
/*Δημιουργεί ένα κατά πλάτος δέντρο επικάλυψης}. */
      typos_ouras oura;
      kdktis n,m;
      adktis trexon;
      n = G;
      while (n!=NULL){
            n->episkeftike = 0;
            trexon = n->kefali;
            while (trexon!=NULL){
                  trexon->simadi = 0;
                  trexon = trexon->epomenos;
            }
            n = n->epomenos;
      }
```

συνέχεια

```
        dimiourgia(&oura);
        /*Το ξεκίνημα από την κεφαλή της λίστας κορυφών*/
        n = G;
        prosthesi(&oura,n);
        n->episkeftike = 1;
        while (!keni(oura)){
              apomakrynsi(&oura,&n);
              /* επισκέπτεται η λιστά ακμών */
              trexon = n->kefali;
              while (trexon!=NULL){
                    m = trexon->akro;
                    if (!(m->episkeftike)){
                          prosthesi(&oura,m);
                          m->episkeftike = 1;
                          trexon->simadi = 1;
                    }
                    trexon = trexon->epomenos;
              }
        }
}
```

Ενα πρώτα κατά πλάτος δέντρο επικάλυψης του γραφήματος. Αφετηρία η κορυφή a.



Εφαρμογές Γραφημάτων

Μεταβατικό γράφημα :

$$\text{Αν } A[i,j] = \text{true} \Rightarrow \exists \langle v_i, v_j \rangle \in E(G)$$

$$\text{Αν } A[i,k] \text{ and } A[k,j] = \text{true} \Rightarrow \exists \text{ μονοπάτι μήκους } 2 \quad v_i \rightarrow v_k \rightarrow v_j$$

Γενικά :

$$(A[i,1] \text{ and } A[1,j]) \text{ or } (A[i,2] \text{ and } A[2,j]) \text{ or}$$

$$\ldots\ldots \text{or } (A[i,n] \text{ and } A[n,j]) = \text{true} \Leftrightarrow \exists \text{ μονοπάτι μήκους } 2$$

$$v_i \rightarrow v_1 \rightarrow v_j \text{ ή } v_i \rightarrow v_2 \rightarrow v_j \text{ ή } \ldots\ldots$$

$$\text{ή } A^{(2)} = \bigcup_{k=1}^{n} A[i,k] \cap A[k,j]$$

$$A^{l} = A^{(1)} \cap A^{l-1},\ l = 2, 3, 4, \ldots.$$

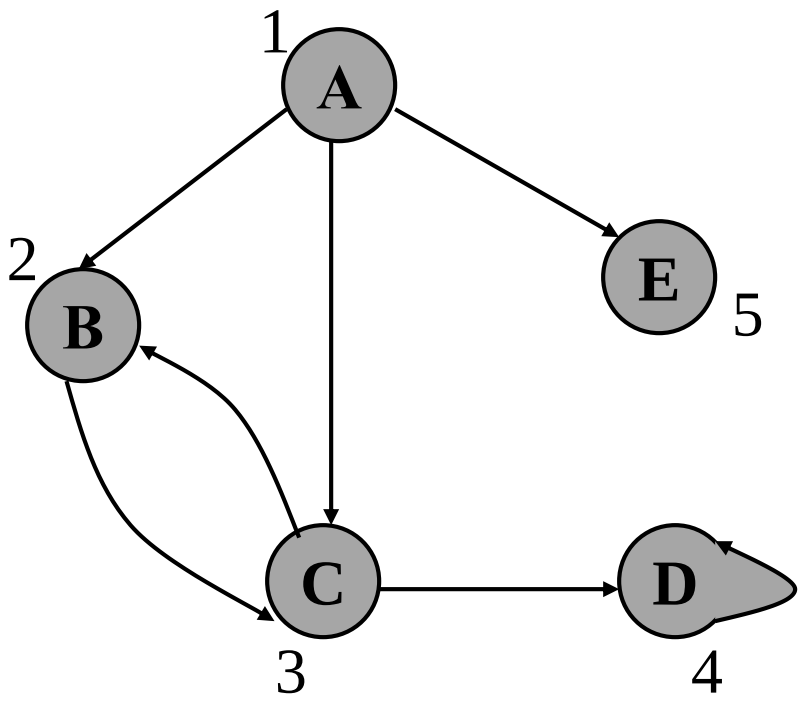
$$A = A^{(1)}$$

Υπάρχει μονοπάτι μήκους <= 3 ;

$$P = A^{(1)} \cup A^{(2)} \cup A^{(3)}$$

Υπαρξη μονοπατιού

$$P = A^{(1)} \cup A^{(2)} \cup A^{(3)} \cup \ldots\ldots \cup A^{(n)}$$



	1	2	3	4	5
1	0	1	1	0	1
2	0	0	1	0	0
3	0	1	0	1	0
4	0	0	0	1	0
5	0	0	0	0	0

Γειτονικός πίνακας

$$A^{(2)} = A^{(1)} \bullet A^{(1)} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

ή

$$A^2 = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Μεταβατικός πίνακας

2 βημάτων

Μεταβατικός Πίνακας / Γράφημα

Γράφημα που συνδέει κορυφές που γειτνιάζουν με οποιοδήποτε αριθμό μεταβάσεων (1..n). Και ο πίνακας γειτνίασής του λέγεται μεταβατικός πίνακας. Π.χ. για το παραπάνω γράφημα ο μεταβατικός πίνακας είναι:

$$P=A \vee A^2 \vee A^3 \vee A^4 \vee A^5 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$O(n^4)$

Αλγόριθμος Warshall Μεταβατικού Πίνακα

$O(n^3)$

$P_k[i, j] = 1$ /* TRUE */

Αν υπάρχει μονοπάτι από την i στην j το οποίο να μην διέρχεται από καμία κορυφή μεγαλύτερη του k.

$P_{k+1}[i, j] = 1$ /*true*/ ΑΝ

1. $P_k[i, j] = 1$ /*true*/

ή

2. $P_k[i, k+1] = 1$ και $P_k[k+1, j] = 1$

Τα παραπάνω ισοδυναμούν με

$$P_{k+1}[i, j] = (P_k[i, j] \,||\, (P_k[i, k+1] \;\&\&\; P_k[k+1, j]));$$

ή αναλυτικότερα

```
for (i=1;i<=n;i++)
    for (j=1;j<=n;j++)
        P_k[i, j] = P_{k-1}[i, j]  || (P_{k-1}[i, k] && P_{k-1}[k, j]);
```

Απλοποιώντας τα παραπάνω λαμβάνουμε

```
P_k = P_k-1;
for (i=1;i<=n;i++)
   if  (P_k-1[i, k])
      for (j=1; j<=n; j++)
         P_k[i, j] := P_k-1[i, j] || P_k-1[k, j];
```

και τελικά :

$P_k = P_{k-1;}$
for (i=1;i<=n;i++)
  if (P_{k-1}[i, k])
    for (j=1;j<=n;j++)
      if (P_{k-1}[k, j])
        P_k[i, j] = 1;

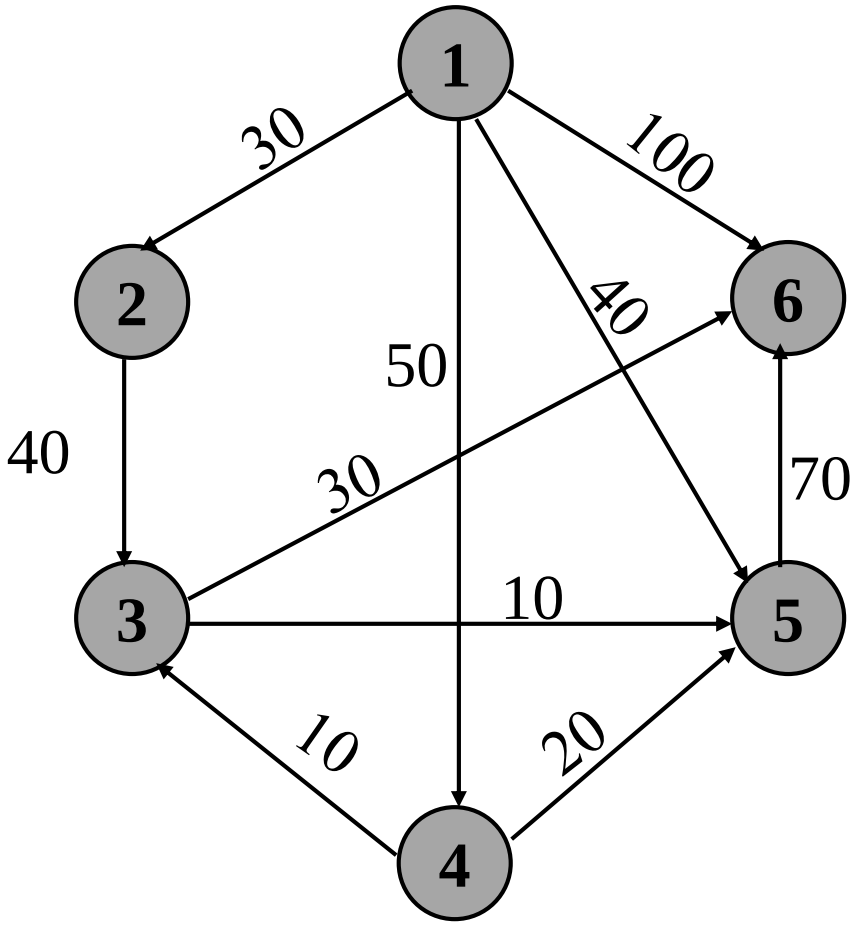
Ισχύει P_0[i,j] = A[i,j]

```
void metavatikos_pinakas(geit_pin A, geit_pin P){
      korifi i,j,k;
      geit_pin PP;
      for (i=0; i <= plithos-1; i++)
            for (j = 0; j <= plithos-1; j++)
                  P[i][j]=A[i][j];

      for (k=0; k <= plithos-1; k++){
            for (i =0 ; i <= plithos-1; i++)
                  for (j = 0; j <= plithos-1; j++)
                        PP[i][j]=P[i][j];

            for (i=0; i <= plithos-1; i++)
                  if (PP[i][k])
                        for (j=0; j <= plithos-1; j++)
                              if (PP[k][j])
                                    P[i][j] = 1;
      }
 }
```

Το συντομότερο μονοπάτι σε ένα κατευθυνόμενο γράφημα: Ο Αλγόριθμος του Dijkstra

Μπορούμε να φανταστούμε το G σαν μια απεικόνιση αεροπορικών γραμμών, στην οποία κάθε κορυφή παριστάνει μια πόλη και το βάρος σε κάθε ακμή το κόστος της πτήσης από μια πόλη σε μιαν άλλη. Το πρόβλημά μας είναι η εύρεση μιας διαδρομής από την πόλη v στην πόλη w έτσι ώστε το συνολικό κόστος (απόσταση) να είναι ελάχιστο.

Ένα κατευθυνόμενο γράφημα με βάρη



• Ας θεωρήσουμε το κατευθυνόμενο γράφημα του παραπάνω σχήματος. Αν υποτεθεί ότι η κορυφή 1 είναι η πηγή, τότε το πρόβλημά μας είναι η εύρεση του συντομότερου μονοπατιού από την κορυφή 1 προς όλες τις άλλες κορυφές του γραφήματος.

• Παράσταση με πίνακα A όπου A[i][j] είναι το κόστος μετάβασης από την i στην j. Αν δεν υπάρχει ακμή, A[i][j]=∞.

• Διατηρεί σύνολο S με κορυφές των οποίων η συντομότερη διαδρομή από την κορυφή-πηγή είναι γνωστή. Σε κάθε βήμα προσθέτει μια κορυφή στο S.

•Πίνακας D (1Δ) περιέχει κόστος (μήκος) για κάθε κορυφή του S.

•Πίνακας P (1Δ) περιέχει το μονοπάτι - προηγούμενη κάθε κορυφής του S.

•Άπληστος αλγόριθμος.

Αλγόριθμος του Dijkstra

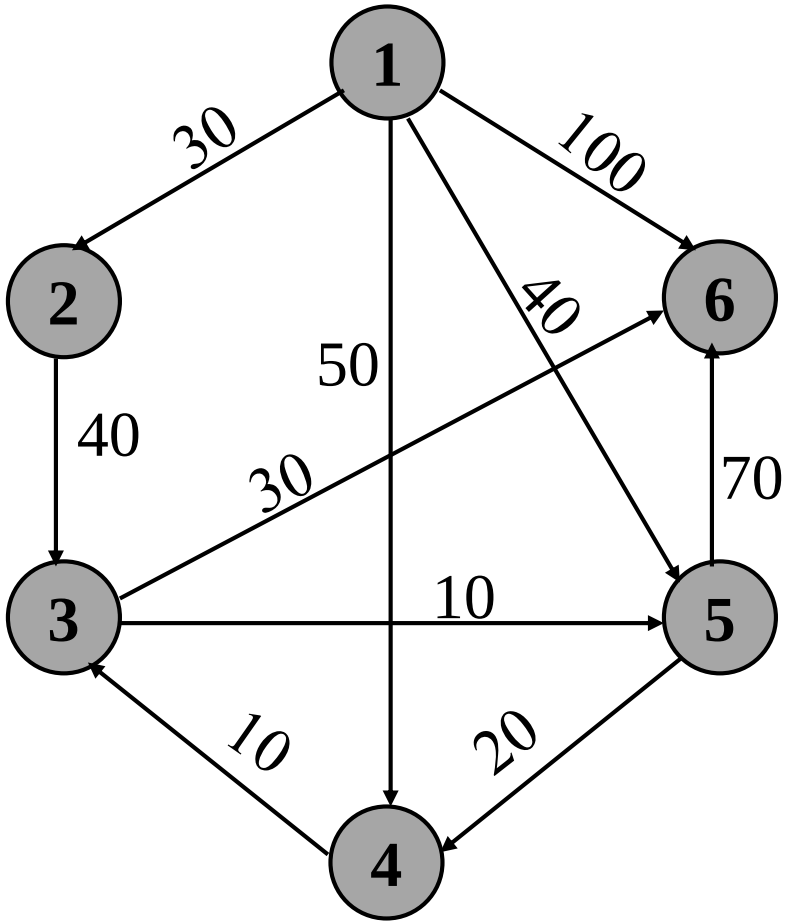
/* Υπολογίζει το κόστος του συντομότερου μονοπατιού από την κορυφή 1 μέχρι μια οποιαδήποτε κορυφή ενός κατευθυνόμενου γραφήματος */

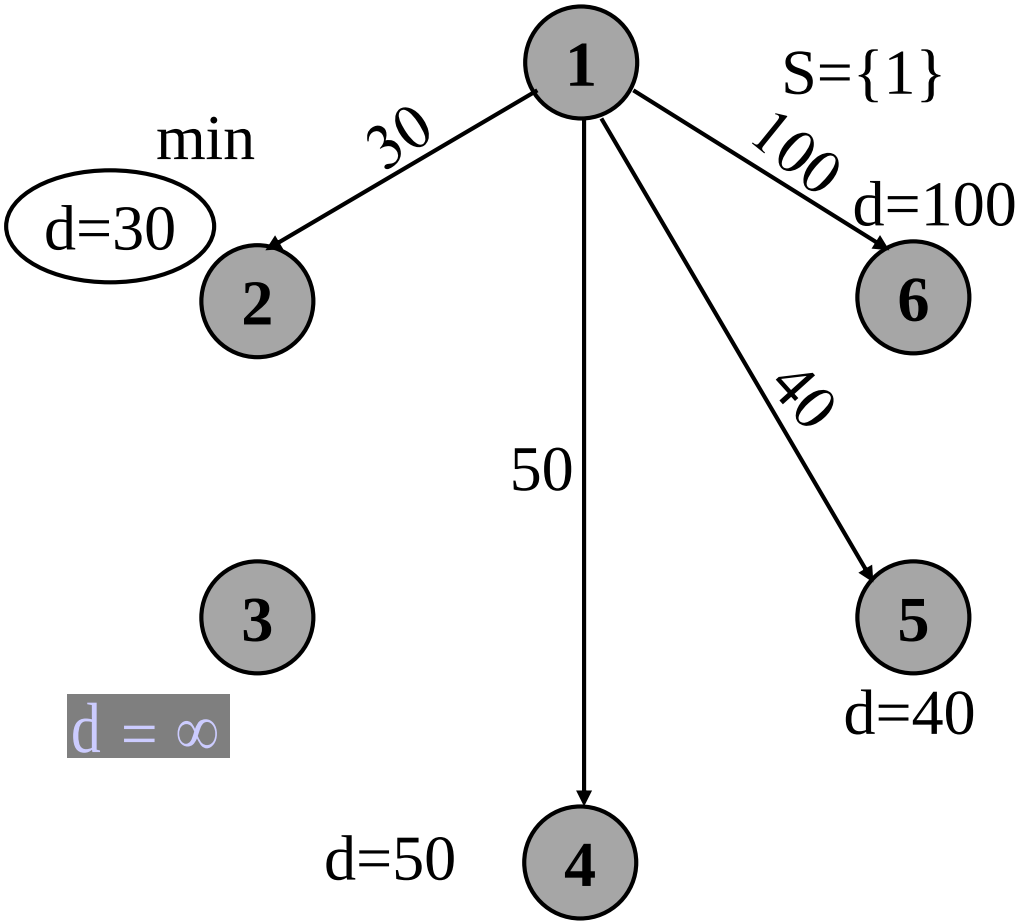
(1) S = {1}; /*Το σύνολο S περιέχει την κορυφή 1*/

(2) for i=2,n do D[i] = A[1,i]; {Αρχικές τιμές του D}

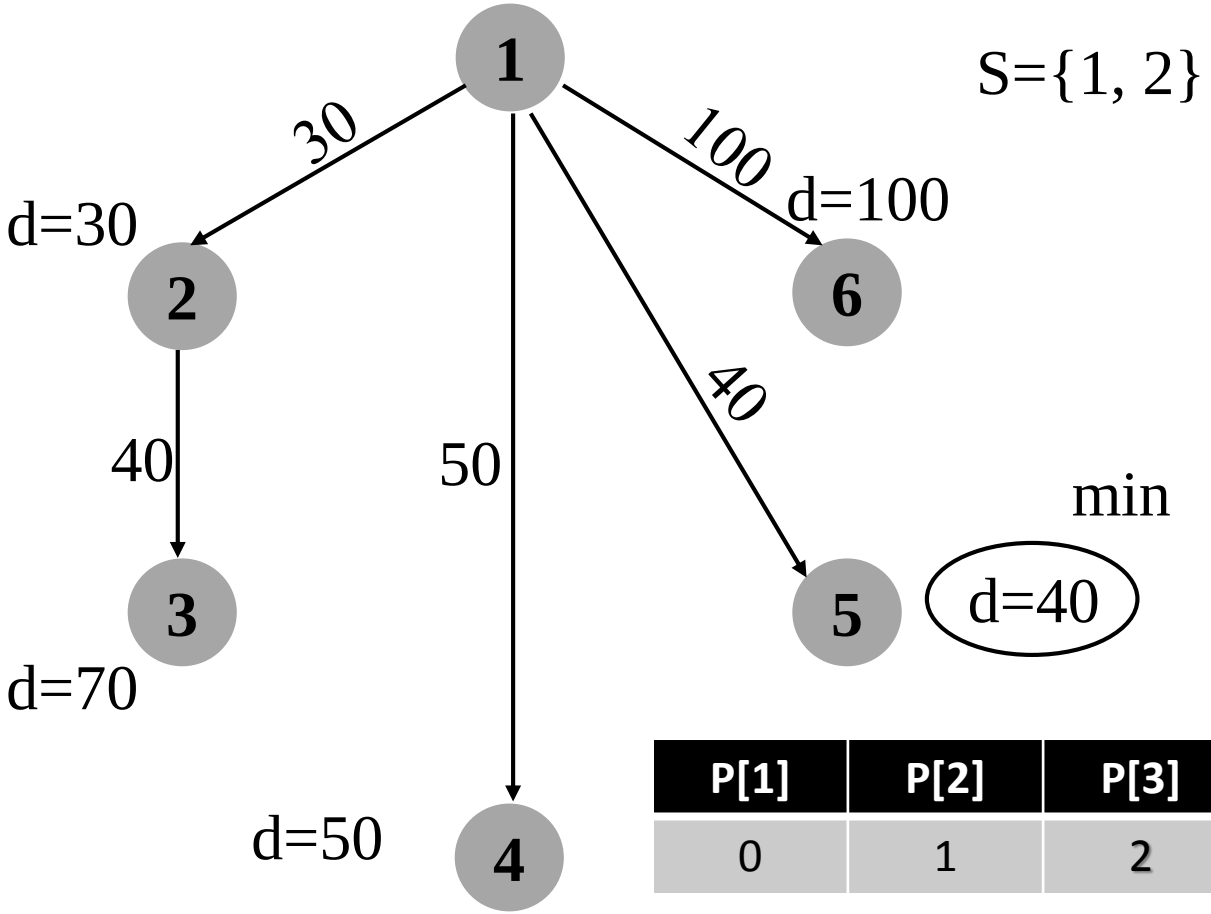
(3) for i=1,n do
P[i] = 0 αν δεν υπάρχει ακμή από την 1 προς την i
P[i] = 1 αν υπάρχει ακμή από την 1 προς την i

(4) for i = 1..n-1 do {
(5) Διάλεξε μια κορυφή w εκτός S τέτοια ώστε η D[w] να είναι ελάχιστη
(6) Πρόσθεσε την w στο S
(7) Για κάθε κορυφή v εκτός S να ελεγχθεί αν η απόσταση απο την 1 στην v μπορεί να συντομευθεί διαμέσου της w. Αν ναι, να ενημερωθεί η αντίστοιχη τιμή του πίνακα D.
(1) D[v] = min (D[v], D[w] + A[w,v])
(2) Αν D[w]+A[w,v] < D[v] τότε P[v]=w



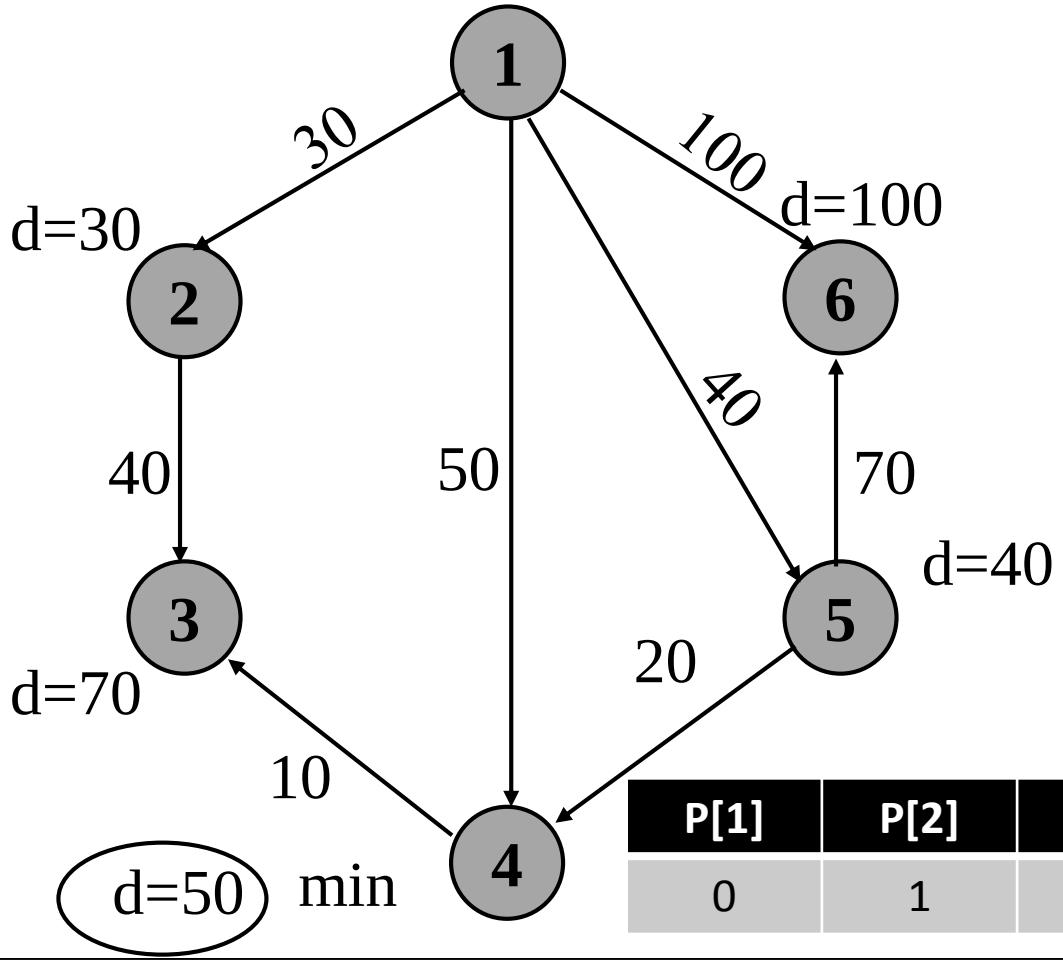


Επανάληψη	s	w	D[2]	D[3]	D[4]	D[5]	D[6]
αρχική	{1}	-	30	∞	50	40	100
	{1,2}	2	30	70	50	40	100



P[1]	P[2]	P[3]	P[4]	P[5]	P[6]
0	1	2	1	1	1

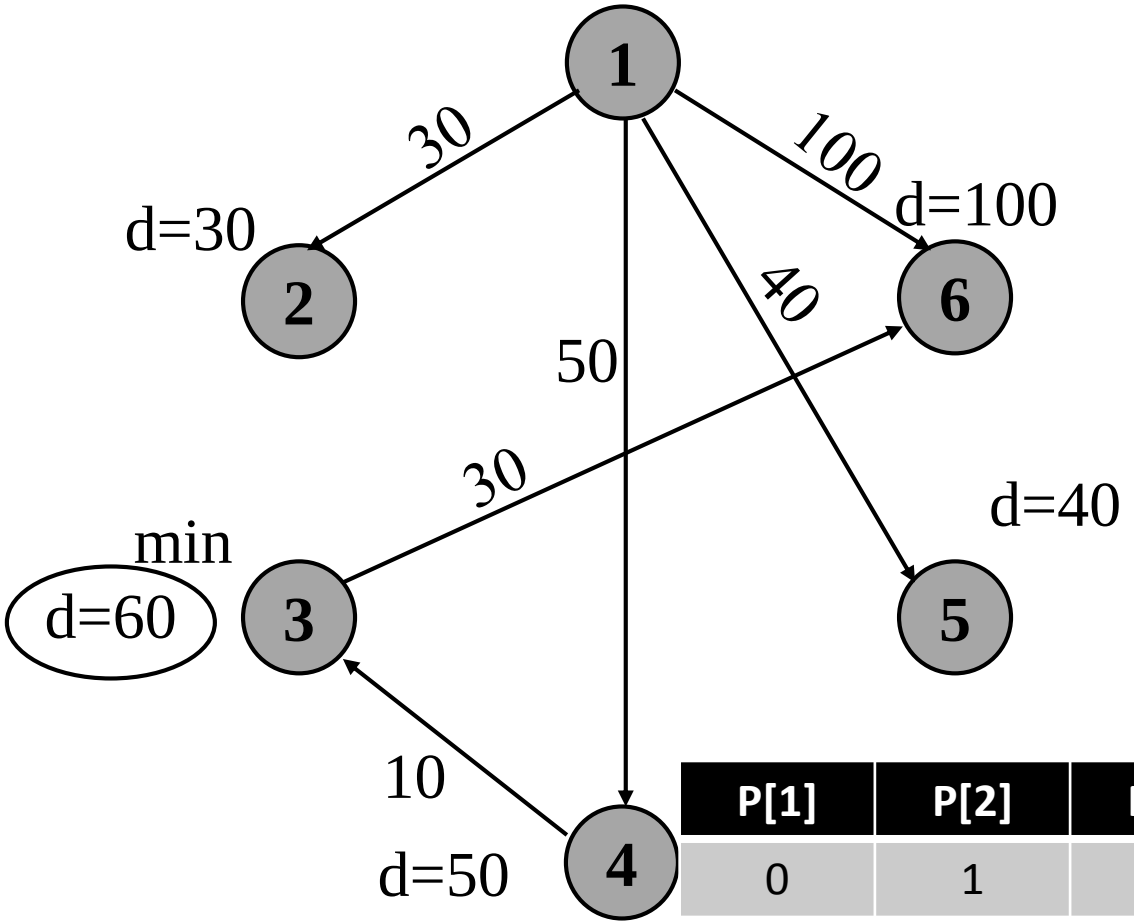
Επανάληψη	s	w	D[2]	D[3]	D[4]	D[5]	D[6]
1	{1,2}	2	30	70	50	40	100
2	{1,2,5}	5	30	70	50	40	100



S={1, 2, 5}

P[1]	P[2]	P[3]	P[4]	P[5]	P[6]
0	1	**4**	1	1	1

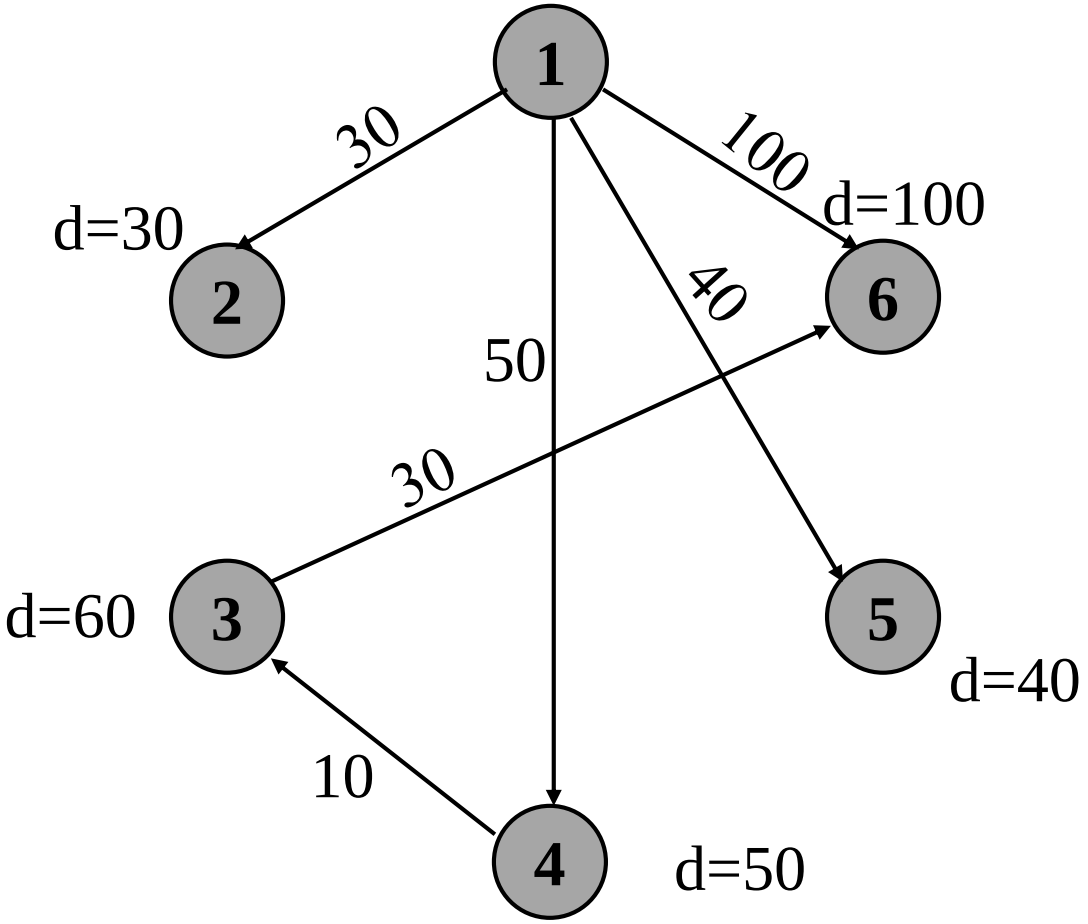
Επανάληψη	s	w	D[2]	D[3]	D[4]	D[5]	D[6]
2	{1,2,5}	5	30	70	50	40	100
3	{1,2,5,4}	4	30	60	50	40	100



S={1, 2, 5, 4}

P[1]	P[2]	P[3]	P[4]	P[5]	P[6]
0	1	4	1	1	**3**

Επανάληψη	s	w	D[2]	D[3]	D[4]	D[5]	D[6]
3	{1,2,5,4}	4	30	60	50	40	100
4	{1,2,5,4,3}	3	30	60	50	40	90



S={1, 2, 5, 4, 3, 6}

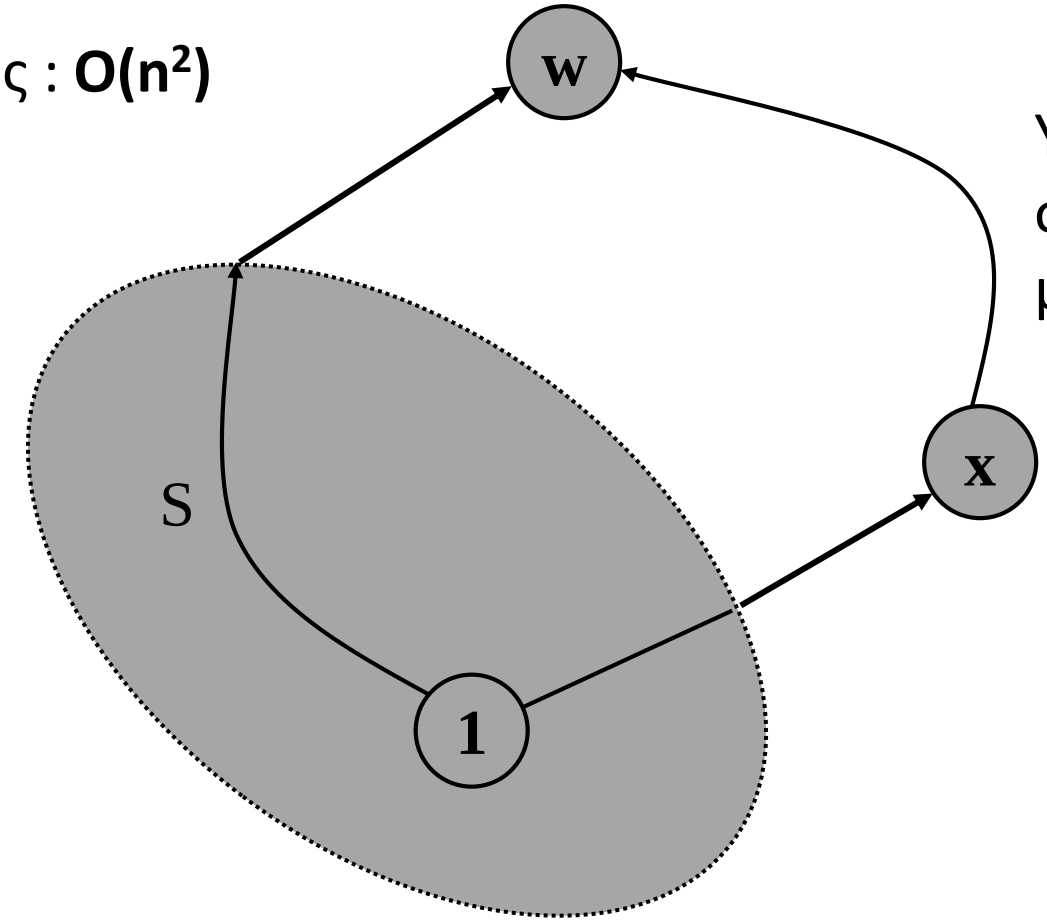
Επανάληψη	s	w	D[2]	D[3]	D[4]	D[5]	D[6]
4	{1,2,5,4,3}	3	30	60	50	40	90
5	{1,2,5,4,3,6}	6	30	60	50	40	90

	s	w	D[2]	D[3]	D[4]	D[5]	D[6]
αρχική	{1}	-	30	∞	50	40	100
1	{1,2}	2	30	70	50	40	100
2	{1,2,5}	5	30	70	50	40	100
3	{1,2,5,4}	4	30	60	50	40	100
4	{1,2,5,4,3}	3	30	60	50	40	90
5	{1,2,5,4,3,6}	6	30	60	50	40	90

Αν αλλάξει D[v] λόγω w P[v]=w. Εδώ Αλλαγές για w=2,4,3 P[3]=2, P[3]=4,P[6]=3

P[1]	P[2]	P[3]	P[4]	P[5]	P[6]
0	1	0	1	1	1
0	1	**2**	1	1	1
0	1	**4**	1	1	1
0	1	4	1	1	**3**

Πολυπλοκότης : **O(n^2)**



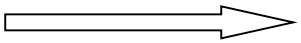
Υποθετικά συντομότερο μονοπάτι

Ο αλγόριθμος του Dijkstra υλοποιείται στο παρακάτω πρόγραμμα όπου χρησιμοποιείται ο πίνακας γειτονικών κορυφών για την παράσταση του γραφήματος.

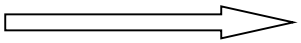
```
#include <stdio.h>
#define plithos 50
#define infinity 20000 /* τιμή όπου δεν υπάρχει ακμή */
#define min_stoixeio  0
#define max_stoixeio plithos-1
#define megisto_plithos max_stoixeio-min_stoixeio+1
typedef int typos_synolou[megisto_plithos];

typedef int korifi;
typedef int typos_pinaka[plithos][plithos];
typedef struct {
      typos_pinaka geit_pin;
      int arkrfon;
 } graphima;
```

συνέχεια

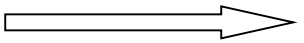
```
void Dimiourgia_graphimatos(graphima *G);
void Diavasma_graphimatos(graphima *G);
void Ektyposi_graphimatos(graphima G);

void Dijkstra(graphima G);
void Synolo_dimiourgia(typos_synolou synolo);
int Synolo_eisagogi(int stoixeio, typos_synolou synolo);
int Synolo_melos(int stoixeio, typos_synolou synolo);

main(){
      graphima G;
      printf("\n Δώστε το πλήθος των κορυφών: ");
      scanf("%d",&(G.arkrfon));
      Dimiourgia_graphimatos(&G);
      Diavasma_graphimatos(&G);
      Ektyposi_graphimatos(G);
      Dijkstra(G);
}
```



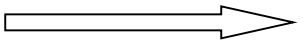
```
void Dimiourgia_graphimatos(graphima *G){
/* Όλα τα στοιχεία του geit_pin έχουν τιμή infinity */

      int i,j;

      for (i=0; i<=(G->arkrfon)-1; i++)
            for (j=0; j<=(G->arkrfon)-1; j++)
                  (G->geit_pin)[i][j] = infinity;
}
```



```
void Diavasma_graphimatos(graphima *G){
/* Προ: έχει δημιουργηθεί ο geit_pin χωρίς ακμές.
  Μετά: Έχει δημιουργηθεί ο geit_pin με ακμές.*/
      korifi i,j;
      int kostos;
      printf("\nΔώστε τα στοιχεία του γραφήματος");
      printf("\nΔώστε i,j και kostos.\n\n");
      scanf ("%d %d %d",&i,&j,&kostos);

      while (i != plithos+1){
            if (  (i>=0) && (i<(G->arkrfon))
                &&(j>=0) &&(j<(G->arkrfon))
                &&(kostos>=0)&&(kostos < infinity))
                  (G->geit_pin)[i][j] = kostos;
            else{
               printf("\nΛάθος δεδομένα. Ξαναδώστε τα");
            }
            scanf ("%d %d %d", &i, &j, &kostos);
      }
}
```



```
void Ektyposi_graphimatos(graphima G){
/*Προ : Έχει δημιουργηθεί ο γειτονικός πίνακας.
Μετά: Για κάθε κορυφή έχει τυπωθεί μια γραμμή στην οποία
εμφανίζονται οι κορυφές με τις οποίες είναι συνδεδεμένη*/

      korifi i,j;

      for (i=0; i <= G.arkrfon-1; i++){
            printf("Γραμμή %2d ", i);
            for (j=0; j <= G.arkrfon-1; j++)
                  printf ("%8d",G.geit_pin[i][j]);
            printf ("\n");
      }
      printf ("\n");
}
```



```
/* Προ : Ένα κατευθυνόμενο γράφημα με βάρη
Μετά: Εύρεση του συντομότερου μονοπατιού και επιστροφή
του αποτελέσματος στον πίνακα D. */
void Dijkstra(graphima G){
      typos_synolou S; /* το σύνολο ειδικών κορυφών*/
      korifi i,j,w;
      int elaxisti_apostasi;
      int apostasi[plithos];
      int monopati[plithos];
      dimiourgia(S);
      eisagogi(0,S);
      /* Αρχικές τιμές για τα συντομότερα μονοπάτια
            εφόσον υπάρχουν ακμές από την κορυφή 0*/
      for (j=1; j <= G.arkrfon-1; j++){
            apostasi[j] = G.geit_pin[0][j];
            if (G.geit_pin[0][j] == infinity)
                  monopati[j] = -1;
            else  monopati[j] = 0;
      }
```

συνέχεια

```
for ( i = 1; i <= G.arkrfon-1; i++){
   /* Επιλογή μιας κορυφής w στο V-S : apostasi[w]=min */
      elaxisti_apostasi = infinity;
      for (j = 1; j <= G.arkrfon-1; j++)
            if (  (!melos(j,S))
                &&(apostasi[j]<elaxisti_apostasi)){
                   elaxisti_apostasi = apostasi[j];
                   w = j;}
      eisagogi(w,S); /* Πρόσθεση της w στο S */
      /*Ενημέρωση των τιμών για τον πίνακα apostasi */
      for (j = 1; j <= G.arkrfon-1; j++)
            if (  (!melos(j,S))
               &&(   apostasi[w]+(G.geit_pin)[w][j])
                   < (apostasi[j])){
                   apostasi[j]=
                       apostasi[w]+(G.geit_pin)[w][j];
                   monopati[j] = w;
            }
}
for (i=1; i <= G.arkrfon-1; i++)
      printf("%d %d %d\n",i,apostasi[i],monopati[i]);
```

Τέλος Ενότητας

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σημειώματα

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.