ΠΕΡΙΕΧΟΜΕΝΑ:

Σπύρος Παπαγιάκουμος
Σμαραγδένιος Χορηγός Μαθήματος

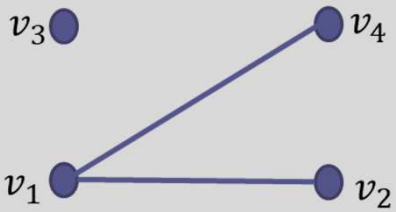
Πάνος Γ.
Ασημένιος Χορηγός Μαθήματος

- **Μη Κατευθυνόμενος Γράφος G=(V,E):**
  - V: Σύνολο Κορυφών $V = \{v_1, v_2, \dots, v_n\}$
  - E: Σύνολο Ακμών $E = \{e_1, e_2, \dots, e_m\}$
    - Κάθε ακμή συνδέει δύο κορυφές, δηλαδή $e_k = [v_i, v_j]$ ή $e_k = \{v_i, v_j\}$ με $v_i, v_j \in V$ για κάθε $k = 1, \dots, m$
    - Οι ακμές είναι μη διατεταγμένες (άρα η ακμή $[v_i, v_j]$ είναι ίδια με την ακμή $[v_j, v_i]$)

Παραδείγματα:

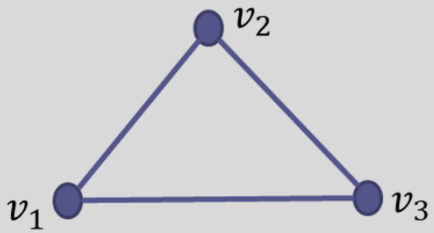
$G = (V, E)$ όπου:

$V = \{v_1, v_2, v_3, v_4\}$
$E = \{[v_1, v_2], [v_1, v_4]\}$



$G = (V, E)$ όπου:

$V = \{v_1, v_2, v_3\}$
$E = \{[v_1, v_2], [v_1, v_3], [v_3, v_2]\}$



Παρατήρηση:

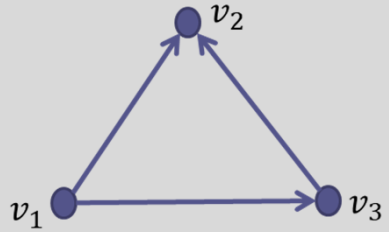
- Βλέπε και playlist Μαθηματικά Πληροφορικής => Θεωρία Γράφων

- **Κατευθυνόμενος Γράφος G=(V,E)**
  - Ομοίως αλλά οι ακμές θεωρούνται διατεταγμένες
  - και συνήθως τις αναπαριστούμε ως $e_k = (v_i, v_j)$

Παραδείγματα:

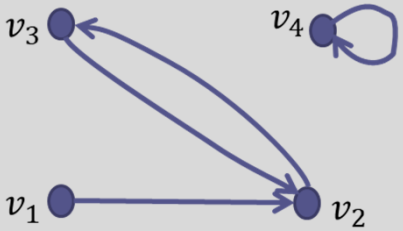
$G = (V, E)$ όπου:

$V = \{v_1, v_2, v_3\}$
$E = \{(v_1, v_2), (v_1, v_3), (v_3, v_2)\}$



$G = (V, E)$ όπου:

$V = \{v_1, v_2, v_3, v_4\}$
$E = \{(v_1, v_2), (v_2, v_3), (v_3, v_2), (v_4, v_4)\}$



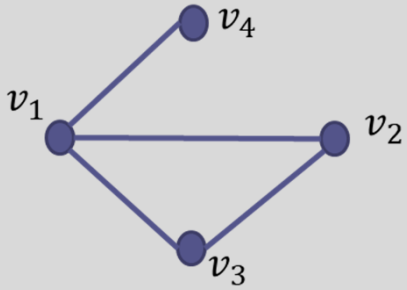
- **Ανακύκλωση:** Ακμή από κορυφή στον εαυτό της
- **Παράλληλες ακμές:** Ακμές με κοινά άκρα και κοινή φορά
- **Αντιπαράλληλες ακμές:** Ακμές με κοινά άκρα και αντίθετη φορά
- **Απλός Γράφος:** Χωρίς ανακυκλώσεις και παράλληλες ακμές.
- **Μονοπάτι:** Ακολουθία «διαδοχικών» ακμών που σέβεται την κατεύθυνση
- **Κύκλος:** Μονοπάτι που ξεκινά και τελειώνει στην ίδια κορυφή
- **Βαθμός Κορυφής:** Πλήθος Προσπιπτουσών Ακμών στην κορυφή (σε ΚΓ=> **έσω βαθμός**: πλήθος εισερχόμενων ακμών, **έξω βαθμός**: πλήθος εξερχόμενων ακμών)
- **Γράφος με βάρη:** Οι ακμές έχουν βάρος (π.χ. κόστος διέλευσης)

Αναπαράσταση με Πίνακα Γειτνίασης

- Ένας απλός μη κατευθυνόμενος γράφος αναπαρίσταται με τον πίνακα:

$$A_{n\times n} = (a_{i,j}) = \begin{cases} 1, & \alpha\nu\ [v_i, v_j] \in E \\ 0, & \alpha\nu\ [v_i, v_j] \notin E \end{cases}$$

Παράδειγμα:



$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Παρατηρήσεις:

- Οι παράλληλες ακμές εκφράζονται θέτοντας τιμές >1 στο αντίστοιχο κελί του πίνακα γειτνίασης.
- Οι ανακυκλώσεις εκφράζονται θέτοντας 1, στο αντίστοιχο κελί της κύριας διαγωνίου του πίνακα.
- Ο πίνακας γειτνίασης για μη κατευθυνόμενα γραφήματα, μπορεί να αναπαρασταθεί με κατάλληλη αποθήκευση μόνο του άνω τριγωνικού μέρους του πίνακα.
- Για κατευθυνόμενα γραφήματα, η διαχείριση είναι εντελώς αντίστοιχη.

Δηλώσεις στη C

- Η δομή περιέχει: Τη διάσταση N και τον NxN πίνακα

```
/* graph.h : Dilwseis grafou (pin.geitniasis) */

struct graph       /* Typos komvou listas */
{
  int **array;     /* pin.geitniasis */
  int N;           /* Plithos komvwn */
};

typedef struct graph GRAPH;
```

Και θα ορίσουμε τις βασικές πράξεις:

- **GR_init(graph, N):** Δεσμεύει χώρο για τον γράφο και αρχικοποιεί τον πίνακα με μηδενικά.
- **GR_print(graph):** Τυπώνει το γράφο.
- **GR_add_edge(graph, vertex1, vertex2):** Προσθέτει την ακμή (vertex1, vertex2) στο γράφο.
- **GR_destroy(graph):** Αποδεσμεύει το χώρο μνήμης του γράφου.

[βλ. project: Graph.Adjacency.Matrix]

Παρατηρήσεις:

- Ορίζουμε και πολλές δευτερεύουσες πράξεις (αλλά θα τις προσθέτουμε κατά περίπτωση, ανάλογα με τον αλγόριθμο που θα μελετάμε)

GR_init(graph, N): Δεσμεύει χώρο για τον γράφο και αρχικοποιεί τον πίνακα με μηδενικά.

```
/* GR_init(): arxikopoiei to grafo */
void GR_init(GRAPH *g, int N)
{
  int i,j;
  g->N = N;
  g->array = (int **)malloc(sizeof(int*)*N);
  if(!g->array)
  {
     printf("Error Allocating Memory");
     exit(0);
  }
  for (i=0; i<N; i++)
  {
     g->array[i] = (int *)malloc(sizeof(int)*N);
     if (!g->array[i])
     {
          printf("Error Allocating Memory");
          exit(0);
      }
      for (j=0; j<N; j++)
         g->array[i][j] = 0;
   }
}
```

GR_print(graph): Εκτυπώνει τον πίνακα γειτνίασης.

GR_add_edge(graph, vertex1, vertex2): Προσθέτει την ακμή (vertex1, vertex2) στο γράφο.

```
void GR_add_edge(GRAPH g, int vertex1, int vertex2)
{
   if (vertex1<0 || vertex1>g.N || vertex2<0 || vertex2>g.N)
   {
      printf("Error: index out of bounds");
      exit(0);
    }
    if (g.array[vertex1][vertex2]==1)
       printf("Error: H akmh (%d,%d) yparxei hdh", vertex1, vertex2);
    else
    {
       g.array[vertex1][vertex2]=1;
       g.array[vertex2][vertex1]=1;
    }
}
```

Παρατηρήσεις:

- Βλέπε και τη δευτερεύουσα πράξη GR_init_from_file(graph, filename) η οποία αρχικοποιεί τον πίνακα από ένα αρχείο κειμένου.

Άσκηση 1: Κατευθυνόμενοι Γράφοι και Γράφοι με Βάρη

Δημιουργήστε παραλλαγές της δομής δεδομένων «Γράφος – Πίνακας Γειτνίασης» ώστε:

- Να αποτυπώνει έναν κατευθυνόμενο γράφο
- Να αποτυπώνει έναν μη κατευθυνόμενο γράφο με ακέραια βάρη στις ακμές

Άσκηση 2: Πλήθος ακμών

Δεδομένου του γράφου με βάρη που κατασκευάσαμε στην προηγούμενη άσκηση:

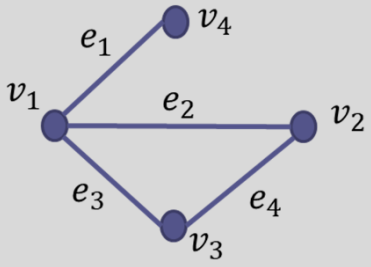
- Κατασκευάστε τη συνάρτηση edges() η οποία να επιστρέφει το πλήθος των ακμών του γράφου.

Αναπαράσταση με Πίνακα Προσπτώσεως

- Ένας μη κατευθυνόμενος γράφος αναπαρίσταται με τον πίνακα:

$$A_{n\times m} = (a_{i,j}) = \begin{cases} 1, & \text{αν η ακμη } e_j \text{ προσπιπτει στη } v_i \\ 0, & \text{αν η ακμη } e_j \text{ δεν προσπιπτει στη } v_i \end{cases}$$

Παράδειγμα:



$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Παρατηρήσεις:

- Ο πίνακας γειτνίασης για κατευθυνόμενα γραφήματα, αναπαρίσταται θέτοντας 1 στην κορυφή-αφετηρία και -1 στην κορυφή-πέρας της ακμής.

Δηλώσεις στη C

- Η δομή περιέχει: Τη διάσταση N, το πλήθος των ακμών M και τον NxM πίνακα

```
/* graph.h : Dilwseis grafou (pin.geitniasis) */

struct graph       /* Typos komvou listas */
{
  int **array;     /* pin.geitniasis */
  int N;           /* Plithos komvwn */
};

typedef struct graph GRAPH;
```

Και θα ορίσουμε τις βασικές πράξεις:

- **GR_init(graph, N, M):** Δεσμεύει χώρο για τον γράφο και αρχικοποιεί τον πίνακα με μηδενικά.
- **GR_print(graph):** Τυπώνει το γράφο.
- **GR_destroy(graph):** Αποδεσμεύει το χώρο μνήμης του γράφου.

[βλ. project: Graph.Adjacency.Matrix]

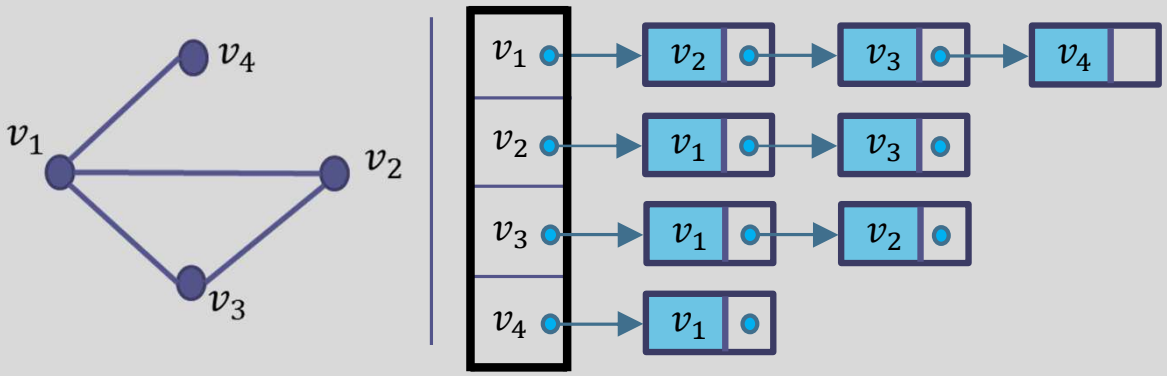
Παρατηρήσεις:

- Αυτή η δομή είναι η πιο σπάνια χρησιμοποιούμενη αναπαράσταση ενός γραφήματος

Αναπαράσταση με Λίστες Γειτνίασης

- Ένας μη κατευθυνόμενος γράφος αναπαρίσταται ως εξής:
  - Ένας πίνακας που περιέχει τους κόμβους του γράφου
  - Κάθε κόμβος περιέχει μία λίστα με τους γειτονικούς της κόμβους.

Παράδειγμα:



Παρατηρήσεις:

- Σε μη κατευθυνόμενο γράφο, κάθε ακμή εμφανίζεται σε δύο κορυφές στον πίνακα κορυφών.
- Σε κατευθυνόμενο γράφο, κάθε ακμή εμφανίζεται σε μία κορυφή (μόνο στην κορυφή που είναι η αφετηρία της ακμής)
- Ένας γράφος με βάρη αναπαρίσταται επεκτείνοντας κάθε επόμενο ώστε να περιλαμβάνει (εκτός το όνομα της κορυφής που είναι επόμενη) και το βάρος αυτής της ακμής.

Δηλώσεις στη C

- Η δομή περιέχει: Τη διάσταση N και τον NxN πίνακα

```
/* graph.h : Dilwseis grafou (pin.geitniasis) */

struct graph              /* Typos komvou listas */
{
   LIST_PTR *array;       /* pinakas listwn geitniasis */
   int N, M;              /* Plithos komvwn, akmwn */
};

typedef struct graph GRAPH;
```

Και θα ορίσουμε τις βασικές πράξεις:

- **GR_init(graph, N):** Δεσμεύει χώρο για τον γράφο και αρχικοποιεί τον πίνακα με μηδενικά.
- **GR_print(graph):** Τυπώνει το γράφο.
- **GR_add_edge(graph, vertex1, vertex2):** Προσθέτει την ακμή (vertex1, vertex2) στο γράφο.
- **GR_destroy(graph):** Αποδεσμεύει το χώρο μνήμης του γράφου.

[βλ. project: Graph.Adjacency.Lists]

Παρατηρήσεις:

- Η λίστα γειτνίασης προτιμάται όταν ο γράφος είναι «αραιός» (έχει σχετικά λίγες ακμές: m < nlogn)

GR_init(graph, N): Δεσμεύει χώρο για τον γράφο και αρχικοποιεί τις (απλά) συνδεδεμένες λίστες

```
void GR_init(GRAPH *g, int N)
{
   int i;
   g->N = N;

   g->array = (LIST_PTR *)malloc(sizeof(int)*N);
   if(!g->array)
   {
      printf("Error Allocating Memory");
      exit(0);
   }
   for (i=0; i<N; i++)
      LL_init(&(g->array[i]));
}
```

GR_print(graph): Εκτυπώνει τη δομή.

GR_add_edge(graph, vertex1, vertex2): Προσθέτει την ακμή (vertex1, vertex2) στο γράφο.

```
void GR_add_edge(GRAPH g, int vertex1, int vertex2)
{
   if (vertex1<0 || vertex1>g.N || vertex2<0 || vertex2>g.N)
   {
      printf("Error: index out of bounds");
      exit(0);
   }

   LL_insert(&g.array[vertex1], vertex2);
   LL_insert(&g.array[vertex2], vertex1);
}
```

Παρατηρήσεις:

- Βλέπε και τη δευτερεύουσα πράξη GR_init_from_file(graph, filename) η οποία αρχικοποιεί τον πίνακα από ένα αρχείο κειμένου.
- Έχει υλοποιηθεί και μία δευτερεύουσα πράξη στη δομή της απλά συνδεδεμένης λίστας, η οποία κάνει την εισαγωγή του στοιχείου διατηρώντας μια διάταξη (αύξουσα με βάση τον αριθμό της κορυφής)

Άσκηση 3: Επόμενοι κορυφής

Επεκτείνετε τη δομή των λιστών γειτνίασης:

- Με τη συνάρτηση neighbors: θα παίρνει ως όρισμα μία κορυφή και θα επιστρέφει το πλήθος των γειτόνων της και έναν πίνακα με αυτούς τους γείτονες.

Άσκηση 4: Γράφος με Βάρη

Επεκτείνετε κατάλληλα τη δομή των λιστών γειτνίασης, ώστε κάθε ακμή να σχετίζεται με ένα βάρος (ακέραιος αριθμός)