ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Τεχνολογίες Υλοποίησης Αλγορίθμων

Χρήστος Ζαρολιάγκης

Καθηγητής
Τμήμα Μηχ/κων Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών
email: zaro@ceid.upatras.gr

Ενότητα 5

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

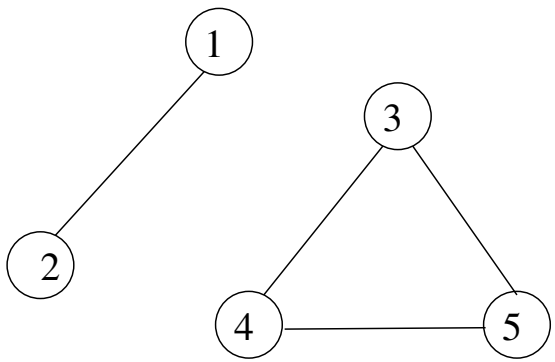
- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο **«Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών»** έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση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

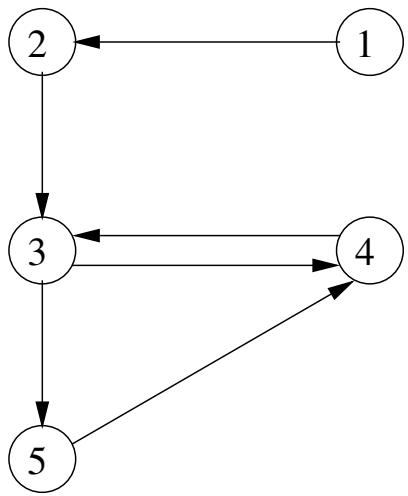
Περιεχόμενα

Γραφήματα – βασικές έννοιες

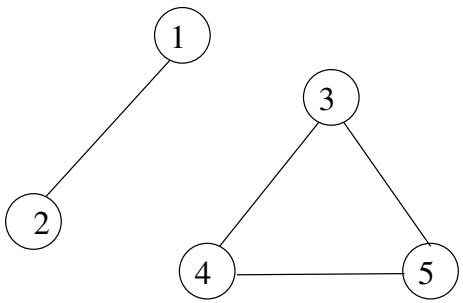
- **Γράφημα** $G = (V, E)$:
  - σύνολο **κόμβων/κορυφών (nodes/vertices)** V
  - σύνολο **πλευρών/ακμών (edges/arcs)** $E \subseteq V \times V$

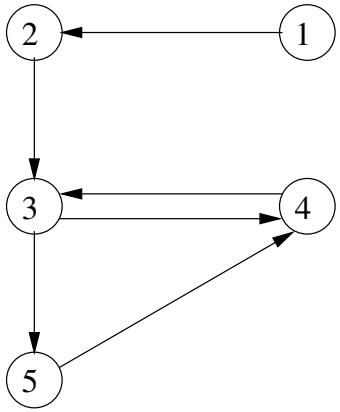
    σχέσεις μεταξύ κόμβων
- Μια ακμή $e = (i, j) \in E$ «συσχετίζει» ή «συνδέει» τις κορυφές i και j





Γραφήματα – βασικές έννοιες



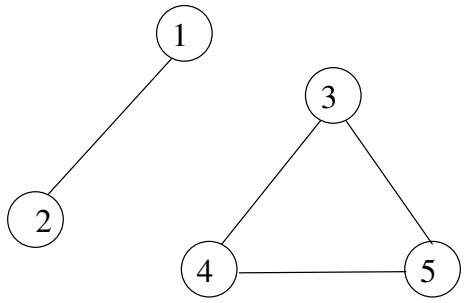


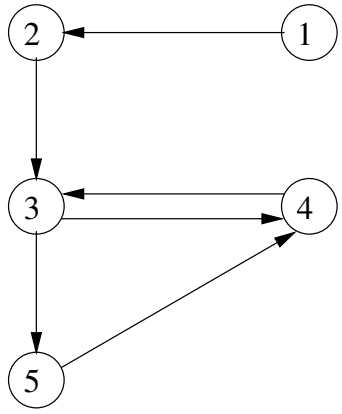
- Κατευθυνόμενες ακμές $\Rightarrow$ **κατευθυνόμενο γράφημα**
  Μη-κατευθυνόμενες ακμές $\Rightarrow$ **μη κατευθυνόμενο γράφημα**

- Κατευθυνόμενη ακμή $e = (i, j)$
  - i: αρχική κορυφή (source/tail)
  - j: τελική κορυφή (target/head)

- Ακμή $e = (i, j) \in E$
  - Η i είναι γειτονική (adjacent) της j
  - Η e προσπίπτει (incident) στις i και j

Γραφήματα – βασικές έννοιες





Συμβολισμός

- $n = |V|, m = |E|$
- Μη κατευθυνόμενα γραφήματα, **βαθμός (degree)** κορυφής i:
  d_i = # προσπιπτουσών ακμών της κορυφής i
- Κατευθυνόμενα γραφήματα
  βαθμός-εισόδου (in-degree) κορυφής i:
  d_i^{in} = # προσπιπτουσών ακμών που έχουν την i ως τελική
  βαθμός-εξόδου (out-degree) κορυφής i:
  d_i^{out} = # προσπιπτουσών ακμών που έχουν την i ως αρχική

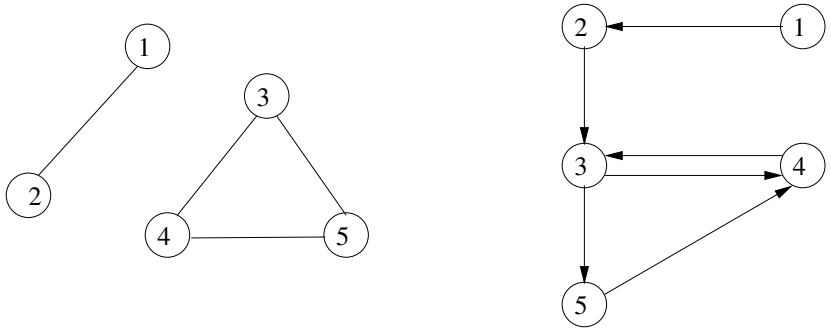
Θεώρημα Euler

(I) $\sum_{i=1}^{n} d_i = 2m$ (II) $\sum_{i=1}^{n} d_i^{in} = \sum_{i=1}^{n} d_i^{out} = m$

Γραφήματα – Αναπαράσταση

- Μητρώο Γειτνίασης (adjacency matrix)
- Λίστες Γειτνίασης (adjacency lists)
  - Με χρήση πινάκων (Packed-Adjacency Lists)
  - Με χρήση γραμμικών λιστών (Linked-Adjacency Lists)
- Έμμεση αναπαράσταση (LEDA)

Γραφήματα – Αναπαράσταση Μητρώου Γειτνίασης



- Μητρώο A διαστάσεων $n \times n$:

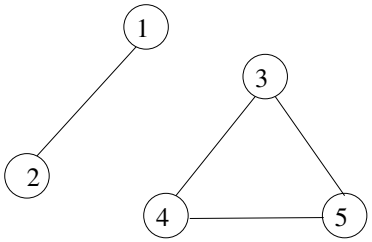
$$A(i,j) = \begin{cases} 1 & \textbf{αν } (i,j) \in E \\ 0 & \textbf{αλλιώς} \end{cases}$$

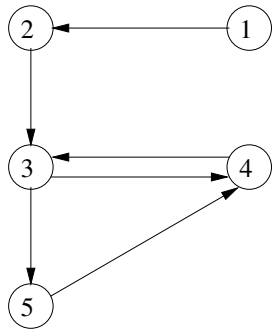
	1	2	3	4	5
1	0	1	0	0	0
2	1	0	0	0	0
3	0	0	0	1	1
4	0	0	1	0	1
5	0	0	1	1	0

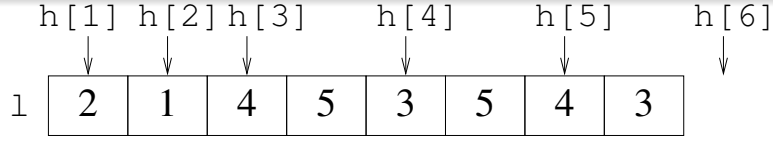
	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	0	1	0

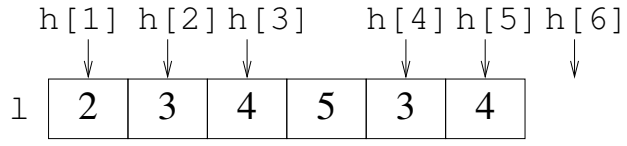
- Χώρος: $O(n^2)$ bytes ή bits
- Χρόνος προσδιορισμού βαθμού κορυφής: $\Theta(n)$
- Χρόνος εισαγωγής/διαγραφής ακμής: $O(1)$

Γραφήματα – Αναπαράσταση με χρήση πινάκων







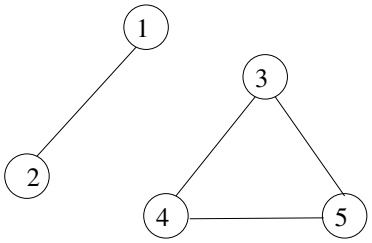


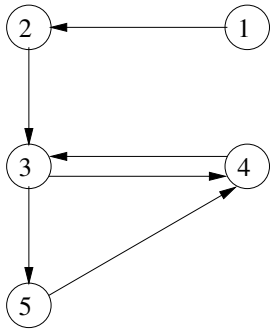
- Μονοδιάστατοι πίνακες $h[0..n+1]$ και $l[0..M]$, όπου $M = m - 1$ ($M = 2m - 1$) αν G είναι κατευθυνόμενο (μη κατευθυνόμενο)

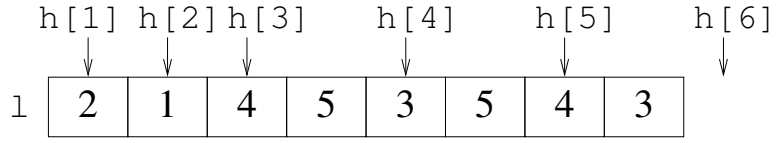
- Οργάνωση πίνακα $l[0..M]$:
  οι πρώτες d_1^{out} (d_1) θέσεις περιέχουν όλες τις κορυφές που είναι γειτονικές στην κορυφή 1,
  οι επόμενες d_2^{out} (d_2) θέσεις περιέχουν όλες τις κορυφές που είναι γειτονικές στην κορυφή 2, κοκ

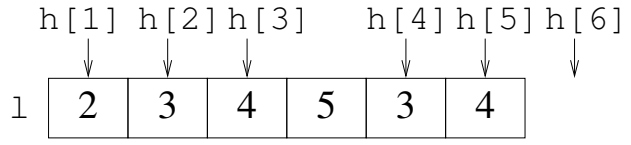
- Οργάνωση πίνακα $h[0..n+1]$: οι γειτονικές κορυφές της κορυφής i βρίσκονται στις θέσεις $l[h[i]], l[h[i]+1], \ldots, l[h[i+1]-1]$, όπου $h[i] < h[i+1]$

Γραφήματα – Αναπαράσταση με χρήση πινάκων









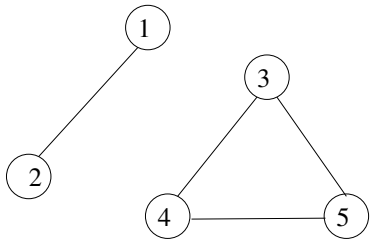
- Χώρος: $\leq (n+1)\lceil \log(2m+1) \rceil + 2m\lceil \log n \rceil$
  $= O((n+m)\log n)$ bits $= O(n+m)$ bytes
- Χρόνος προσδιορισμού βαθμού κορυφής: $O(1)$
- Χρόνος εισαγωγής/διαγραφής ακμής: $O(n+m)$

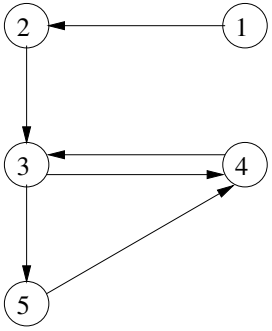
Γραφήματα – Αναπαράσταση με χρήση γραμμικών λιστών

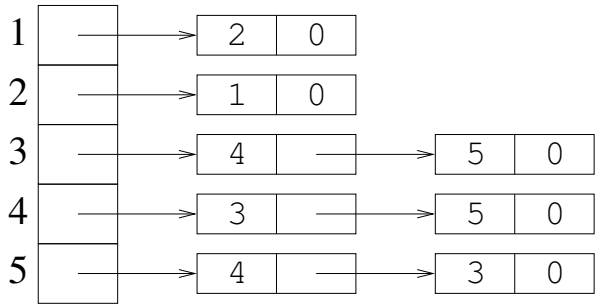
- Κάθε σύνολο γειτονικών κορυφών $\leftrightarrow$ γραμμική λίστα

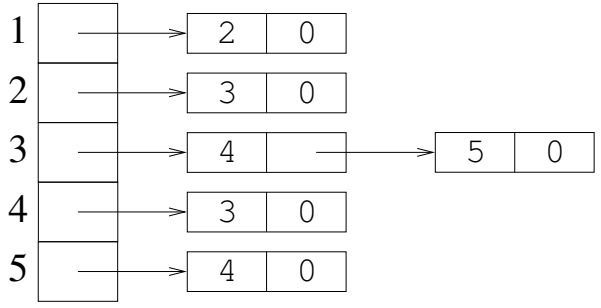
- Κορυφή i αποθηκεύεται στη θέση $h[i]$ ενός πίνακα h
  - $h[i] \rightarrow$ γραμμική λίστα κορυφής i

Γραφήματα – Αναπαράσταση με χρήση γραμμικών λιστών









- Χώρος: $\leq (n+1)\texttt{sizeof(pointer)} + 2m\lceil \log n \rceil$
  $= O((n+m)\log n)$ bits $= O(n+m)$ bytes
- Χρόνος προσδιορισμού βαθμού κορυφής: $O(d_i^{out})$
- Χρόνος εισαγωγής/διαγραφής ακμής: $O(1)$ / $O(d_i^{out})$

Αναπαράσταση Δικτύων

- *Δίκτυο:* γράφημα του οποίου οι κορυφές/ακμές συσχετίζονται με κάποια τιμή

- Μητρώο Γειτνίασης: επιπρόσθετο μητρώο W έτσι ώστε

$$W(i,j) = \begin{cases} \text{weight}(i,j) & \text{αν } (i,j) \in E \\ \texttt{NoEdge} & \text{αλλιώς} \end{cases}$$

  όπου `NoEdge` ισούται με κάποια προεπιλεγμένη τιμή (π.χ. 0 ή ∞)

- Λίστες Γειτνίασης - χρήση πινάκων:
  κάθε στοιχείο του πίνακα I περιέχει ένα ζεύγος (κορυφή, τιμή)

- Λίστες Γειτνίασης - χρήση γραμμικών λιστών:
  κάθε στοιχείο μιας λίστας περιέχει ένα επιπρόσθετο πεδίο το οποίο αποθηκεύει την τιμή της ακμής

Γραφήματα στην LEDA

- Κατευθυνόμενα γραφήματα: `graph, GRAPH<vtype,etype>`
- Μη κατευθυνόμενα γραφήματα: `ugraph, UGRAPH<vtype,etype>`
- Υλοποίηση των τύπων γραφημάτων στην LEDA: αναπαράσταση λιστών γειτνίασης με χρήση γραμμικών λιστών (χρησιμοποιούνται διπλά διασυνδεδεμένες λίστες κορυφών και ακμών)
- Απαιτήσεις χώρου για τον τύπο `graph`: $O(1) + 44m + 52n$ bytes
- Απαιτήσεις χώρου για τον τύπο `GRAPH<T1,T2>`: $O(1) + 44m + 52n +$ `sizeof(T1)`$n +$ `sizeof(T2)`m bytes

Ορισμός γραφημάτων, κορυφών και ακμών

```
graph G;     // G is initialized to the empty graph

node v,w;    // initial values are unspecified for
edge e,f;    // nodes and edges, however...

node u=nil; // we can initialize a node or edge to the
            // special value nil.
            // nil is not a node or edge of any graph.
```

- **Επαναλήπτες Γραφημάτων**

```
forall_nodes(v,G) { }  // iterates over all nodes of G
forall_edges(e,G) { }  // iterates over all edges of G

// iterates over all edges e out of v,
// i.e., source(e)=v
forall_out_edges(e,v) { }
forall_adj_edges(e,v) { }

// iterates over all edges e incident to v,
// i.e., target(e)=v
forall_in_edges(e,v) { }

// iterates over all edges e incident to and from v
forall_inout_edges(e,v) { }
```

- ∃ επίσης κλάσεις τύπου *Iterator*, με χρήση του γνωστού σχεδιαστικού προτύπου Iterator· δείτε το εγχειρίδιο της LEDA

Χρήση Επαναληπτών

```
int m = 0;           // counts the number of edges in G
forall_edges(e,G)    // this number is also available as
   m++;              // G.number_of_edges()
```

Πίνακες Κορυφών και Ακμών (Node and Edge arrays)

- Επιτρέπουν την συσχέτιση πληροφορίας (π.χ. κόστος, id) με κορυφές και ακμές

- ```
  // Name is indexed by the nodes of G; each entry
  // of Name is initialized to the empty string.
  node_array<string> Name(G);

  // Length is indexed by the edges of G; each entry
  // of Length is initialized with the value 1.
  edge_array<int> Length(G,1);

  // ...

  Name[v]   = "Patra";
  Length[e] = 37;
  ```
  ```

Γραφήματα στην LEDA – Συσχετισμός Πληροφορίας με Κορυφές/Ακμές

- Παράδειγμα: ανάθεση αριθμών από 0 μέχρι $n-1$ στις κορυφές του G

- ```
  node_array<int> Number(G);
  int count = 0;
  forall_nodes(v,G)
     Number[v] = count++;
  ```
  ```

Παραμετρικά Γραφήματα

- Εναλλακτική μέθοδος στους πίνακες κορυφών και ακμών

- ```
  GRAPH<string,int> H;

  H[v] = "Patra";    // v must be a node of H
  H[e] = 37;          // e must be an edge of H
  ```
  ```

Πίνακες Κορυφών/Ακμών και Παραμετρικά Γραφήματα

- Οι πίνακες κορυφών και ακμών χρησιμοποιούνται μόνο για στατικά γραφήματα, δηλ. μία νέα κορυφή (ακμή) δεν θα έχει ένα αντίστοιχο στοιχείο στον πίνακα κορυφών (ακμών)
- Τα παραμετρικά γραφήματα είναι πλήρως δυναμικά
- Ένας οποιοσδήποτε αριθμός πινάκων κορυφών και ακμών μπορούν να ορισθούν για ένα γράφημα

Γραφήματα στην LEDA – Δημιουργία

- Γράφημα με δύο κορυφές και καθόλου ακμές

```
graph G;
G.new_node();
G.new_node();
node v;
forall_nodes(v,G)
   // ... do something
```

Γραφήματα στην LEDA – Δημιουργία

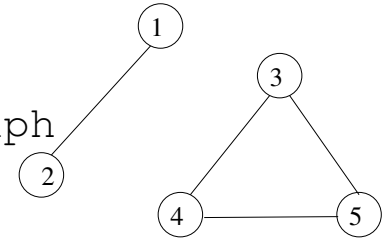
- Δημιουργία ακμής απαιτεί προσδιορισμό αρχικής και τελικής κορυφής

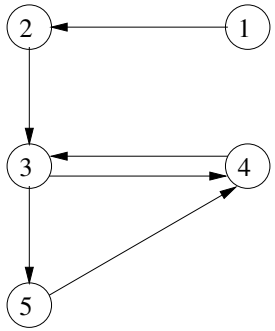
```
node w = G.first_node();
G.new_edge(w, G.succ_node(w));
```

- ***Παρατήρηση:*** Η LEDA διατάσσει εσωτερικά τις κορυφές (ακμές) ενός γραφήματος G σύμφωνα με την σειρά που αυτές προστέθηκαν στο G.

Ένας καλύτερος τρόπος δημιουργίας ακμών

- Η μέθοδος `G.new_node()` όχι μόνο προσθέτει μια νέα κορυφή στο G, αλλά και την επιστρέφει





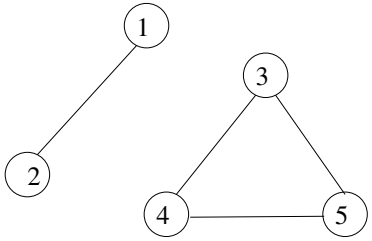

```
// creation of our example digraph
graph G;
node v1 = G.new_node();
node v2 = G.new_node();
node v3 = G.new_node();
node v4 = G.new_node();
node v5 = G.new_node();
G.new_edge(v1,v2); G.new_edge(v2,v3);
G.new_edge(v3,v4); G.new_edge(v4,v3);
G.new_edge(v3,v5); G.new_edge(v5,v4);
```

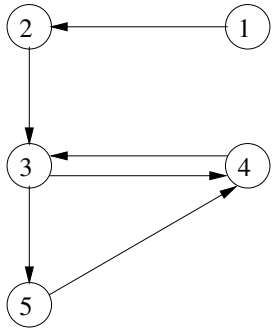
- Αποθήκευση γραφήματος σε αρχείο

```
G.write("graph.gw");
```

Ένας καλύτερος τρόπος δημιουργίας ακμών

- Αν το G είναι παραμετρικό, τότε μπορούμε να συσχετίσουμε πληροφορία με τις κορυφές και τις ακμές κατά τη διάρκεια της δημιουργίας του



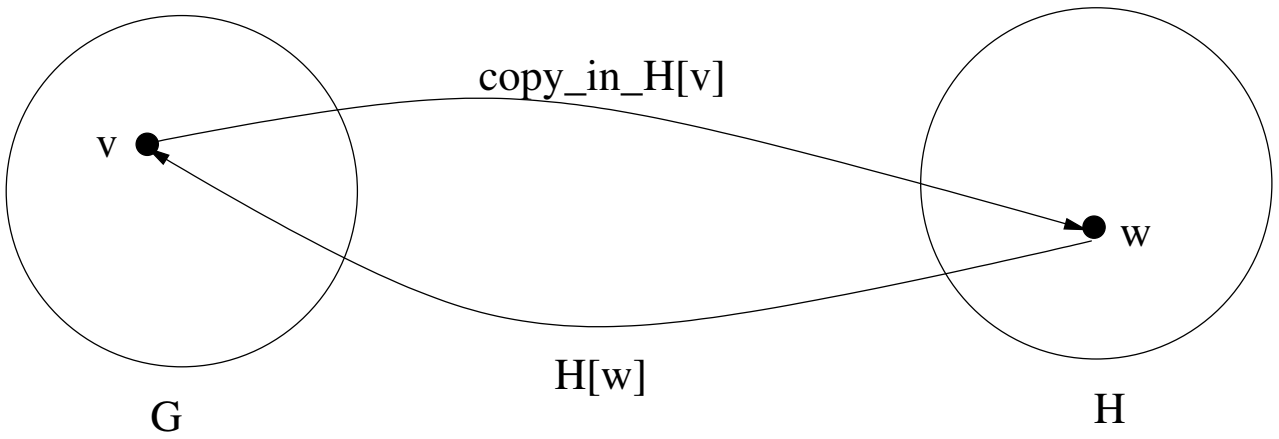



```
// creation of our example digraph
// with info on nodes and edges

GRAPH<int,int> G;
node v1 = G.new_node(1);
node v2 = G.new_node(2);
node v3 = G.new_node(3);
node v4 = G.new_node(4);
node v5 = G.new_node(5);
G.new_edge(v1,v2,10); G.new_edge(v2,v3,15);
G.new_edge(v3,v4,25); G.new_edge(v4,v3,7);
G.new_edge(v3,v5,0); G.new_edge(v5,v4,37);
```

- Δημιουργία ενός ισομορφικού αντιγράφου H ενός γραφήματος G, έτσι ώστε κάθε κορυφή και ακμή του H να γνωρίζει την αυθεντική της στο G

Γραφήματα στην LEDA – δημιουργία ισομορφικού αντίγραφου



- Χρήση παραμετρικού γραφήματος για το H και πίνακα κορυφών για το G
- $\forall v \in G$: `H[copy_in_H[v]]=v`

  $\forall w \in H$: `copy_in_H[H[w]]=w`

Γραφήματα στην LEDA – δημιουργία ισομορφικού αντίγραφου

```
#include <iostream>
#include <LEDA/graph.h>

void CopyGraph(GRAPH<node,edge>& H, const graph& G)
{
 node v;
 edge e;
 H.clear();  // reset H to the empty graph

 node_array<node> copy_in_H(G);
           // associates nodes of G with
           // their copies in H

 forall_nodes(v,G)
   copy_in_H[v] = H.new_node(v);

 forall_edges(e,G)
   H.new_edge(copy_in_H[source(e)],copy_in_H[target(e)],e);
}
```

```
int main()
{
    graph G;
    G.read("graph.gw");
    GRAPH<node,edge> H;

    CopyGraph(H,G);
    cout << endl;
    cout << "n_G = " << G.number_of_nodes() << endl;
    cout << "n_H = " << H.number_of_nodes() << endl;
    cout << "m_G = " << G.number_of_edges() << endl;
    cout << "m_G = " << H.number_of_edges() << endl;
    return 0;
}
```

Γραφήματα στην LEDA – Πίνακες Κορυφών/Ακμών

- Υλοποιούνται με τους συνήθεις πίνακες
  χρόνος αρχικοποίησης $\sim$ # κορυφών/ακμών

- Κορυφές/Ακμές ενός γραφήματος αριθμούνται σύμφωνα με τη σειρά δημιουργίας τους, αρχίζοντας από το 0 (αριθμός $=$ *index*)

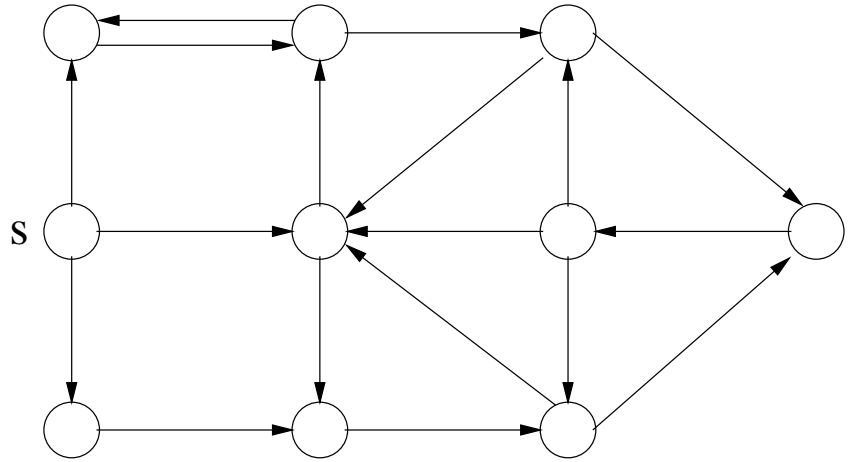
- Η αρίθμηση μιας κορυφής (ακμής) v (e) είναι διαθέσιμη ως $\textbf{\textit{index}}(v)$ ($\textbf{\textit{index}}(e)$)

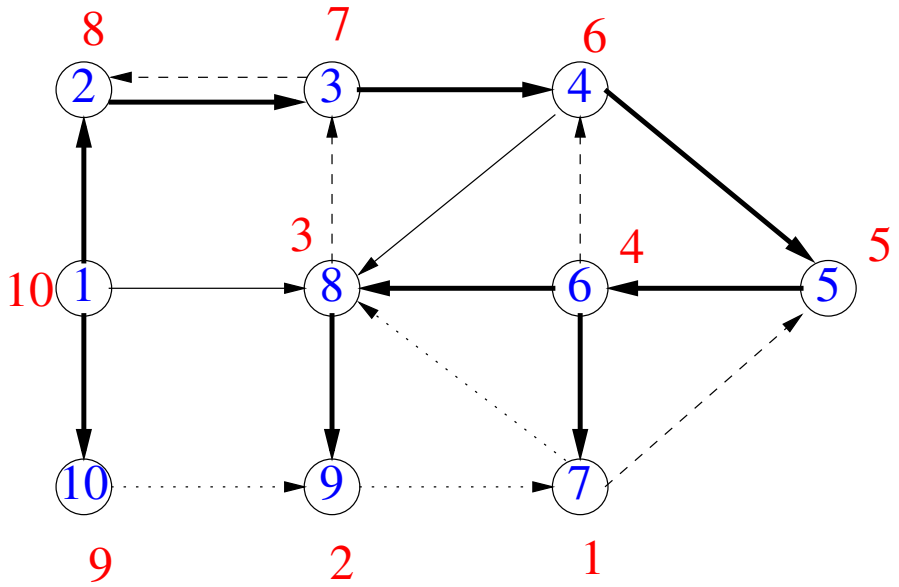
- Προσπέλαση στοιχείου $A[v] \Longrightarrow$
  - Προσπέλαση της δομής αναπαράστασης της κορυφής v για τον προσδιορισμό του $\textbf{\textit{index}}(v)$
  - Προσπέλαση του στοιχείου $A[\textbf{\textit{index}}(v)]$

Γραφήματα στην LEDA – Node/Edge maps

- Παρόμοια με node/edge arrays
- Υλοποιούνται με χρήση κατακερματισμού
- Αρχικοποίηση σε $O(1)$ χρόνο

Γραφήματα στην LEDA – Αναζήτηση Πρώτα κατά Βάθος





Γραφήματα στην LEDA – Αναζήτηση Πρώτα κατά Βάθος

```
void dfs(node s, node_array<bool>& reached, list<node>& L)
{
   L.append(s);
   reached[s] = true;
   node v;
   forall_adj_nodes(v,s)
      if ( !reached[v] ) dfs(v,reached,L);
 }

list<node> DFS(const graph&, node v,
               node_array<bool>& reached)
{
   list<node>    L;
   dfs(v,reached,L);
   return L;
}
```

Κατηγοριοποίηση ακμών

- Διατήρηση δύο αριθμών για κάθε κορυφή v:
  - $dfsnum[v]$: χρόνος (πρώτης) ανακάλυψης
  - $compnum[v]$: χρόνος εγκατάλειψης

- $e = (v, w)$: καλείται *ακμή δένδρου* αν η `dfs(w,...)` καλείται όταν η e εξετάζεται στην `dfs(v,...)`

- Μια ακμή (v, w) είναι
  - *ακμή δένδρου* ή *εμπρός ακμή* αν $dfsnum[v] < dfsnum[w]$ και $compnum[v] > compnum[w]$
  - *πίσω ακμή* αν $dfsnum[v] > dfsnum[w]$ και $compnum[v] < compnum[w]$
  - *διασυνδετική ακμή* αν $dfsnum[v] > dfsnum[w]$ και $compnum[v] > compnum[w]$

Γραφήματα στην LEDA – Αναζήτηση Πρώτα κατά Βάθος

```
list<edge> DFS_NUM(const graph& G,
                   node_array<int>& dfsnum,
                   node_array<int>& compnum)
{
  list<edge> T;
  dfsnum_counter = compnum_counter = 0;

  dfsnum.init(G,-1);  // declares all nodes unreached

  node v;
  forall_nodes(v,G)
    if (dfsnum[v] == -1) dfs(v,dfsnum,compnum,T);

  return T;
}
```

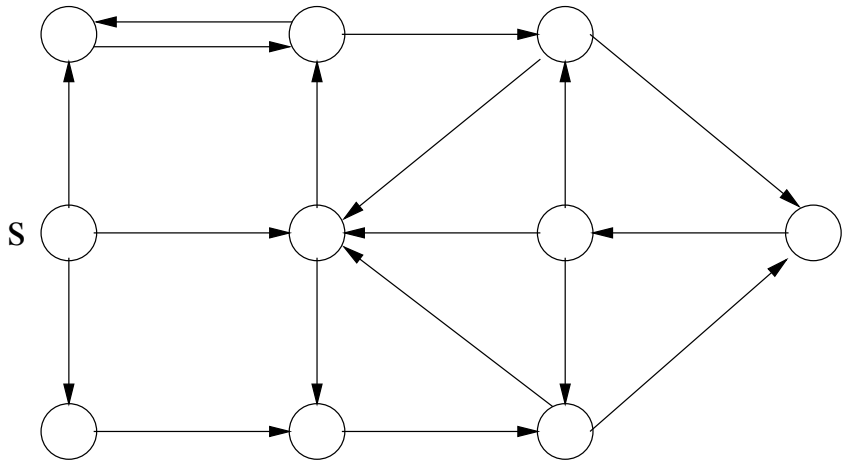
Γραφήματα στην LEDA – Αναζήτηση Πρώτα κατά Βάθος

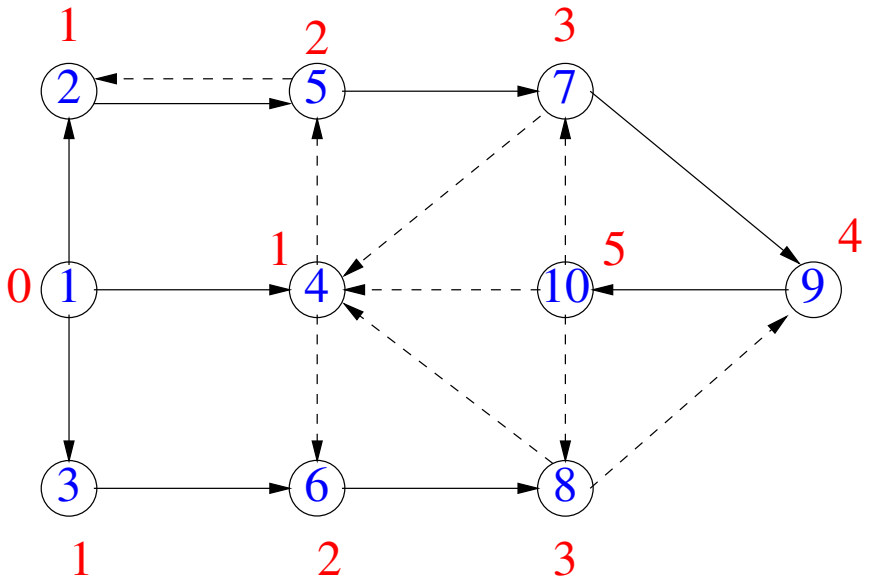
```
static int dfsnum_counter;
static int compnum_counter;

static void dfs(node v, node_array<int>& dfsnum,
                node_array<int>& compnum, list<edge>& T)
{
  dfsnum[v] = ++dfsnum_counter;
  edge e;
  forall_adj_edges(e,v)
  {
      node w = target(e);
      if (dfsnum[w] == -1)
      {
         T.append(e);
         dfs(w,dfsnum,compnum,T);
      }
  }

  compnum[v] = ++compnum_counter;
}
```

Γραφήματα στην LEDA – Αναζήτηση Πρώτα κατά Πλάτος





Γραφήματα στην LEDA – Αναζήτηση Πρώτα κατά Πλάτος

```
void BFS(const graph&, node s, node_array<int>& dist)
{
  queue<node> Q;
  node v,w;
  forall_nodes(w,G)
     dist[w] = -1;

  dist[s] = 0;
  Q.append(s);

  while ( !Q.empty() )
  {
    v = Q.pop;
    forall_adj_nodes(w,v)
       if (dist[w] < 0)
       { Q.append(w);
         dist[w] = dist[v]+1;
       }
  }
}
```

- Έλεχγος εαν ένα δεδομένο κατευθυνόμενο γράφημα $G = (V, E)$ είναι άκυκλο

- Αν ναι, υπολογισμός της *τοπολογικής διάταξης ή ταξινόμησης*: εύρεση αρίθμησης $\ell : V \to \mathbb{N}$ τέτοια ώστε $\forall (x, y) \in E, \ell(x) < \ell(y)$

- **Αλγόριθμος:** επαναληπτική διαδικασία
  - Σε κάθε επανάληψη, επιλογή κορυφής v με μηδενικό βαθμό-εισόδου
  - Ανάθεση στην v της επόμενης τιμής στην αρίθμηση, και διαγραφή της μαζί με τις εξερχόμενες ακμές της
  - Αν σε κάποια επανάληψη δεν υπάρχουν κορυφές μηδενικού βαθμού-εισόδου, τότε η επαναληπτική διαδικασία τερματίζει· αλλιώς, συνεχίζει
  - Αν στον τερματισμό το G είναι το κενό γράφημα, τότε είναι άκυκλο· αλλιώς, περιέχει κύκλο

- Χρόνος εκτέλεσης: $O(n+m)$ - *γιατί ;*

Γραφήματα στην LEDA – Τοπολογική Διάταξη

- Ζητείται συνάρτηση

```
bool TOPSORT(const graph& G, node_array<int>& ord);
```

η οποία επιστρέφει `true` αν το G είναι άκυκλο, αλλιώς `false`. Στην πρώτη περίπτωση επιστρέφει επίσης την τοπολογική διάταξη των κορυφών του G στον πίνακα `ord`

- *Πώς υλοποιείται ο αλγόριθμος χωρίς διαγραφή του γραφήματος ;*

Γραφήματα στην LEDA – Τοπολογική Διάταξη

```
bool TOPSORT(const graph& G,node_array<int>& ord)
{
  // initialization
   node_array<int> INDEG(G);
   queue<node>     ZEROINDEG;
   node v,w;
   forall_nodes(v,G)
   if ( (INDEG[v] = G.indeg(v)) == 0 )
     ZEROINDEG.append(v);

  // removing nodes of indegree zero
   int count = 0;
   ord.init(G);

   while (!ZEROINDEG.empty())
   {
      v = ZEROINDEG.pop();
      ord[v] = ++count;
      forall_out_edges(e,v)
      { node w = G.target(e);
        if ( --INDEG[w] == 0 ) ZEROINDEG.append(w);
      }
   }
   return (count == G.number_of_nodes());
}
```

Γραφήματα στην LEDA – Γεννήτριες Γραφημάτων

- Δημιουργία γραφημάτων με ακολουθίες λειτουργιών `new_node` και `new_edge` είναι αρκετά κουραστική (και βαρετή)

- Η LEDA παρέχει μια αρκετά μεγάλη συλλογή από γεννήτριες γραφημάτων

  - Δημιουργία πλήρους γραφήματος G με n κορυφές: $\forall v, w \in G$, η ακμή (v, w) ανήκει στο G

    ```
    complete_graph(graph& G, int n);
    ```

  - Δημιουγία τυχαίου γραφήματος με n κορυφές και m ακμές

    ```
    random_graph(graph& G, int n, int m);
    ```

  - $\exists$ διάφορες άλλες γεννήτριες οι οποίες κατασκευάζουν ειδικά γραφήματα, π.χ. επίπεδα (planar), πλέγματα (grids), διμερή (bipartite), κλπ

Γραφήματα στην LEDA – δημιουργία τυχαίων γραφημάτων

- Η κλήση

```
random_graph(graph& G, int n, int m, bool no_anti_parallel_edges,
             bool loopfree, bool no_parallel_edges);
```

δημιουργεί ένα τυχαίο γράφημα G με n κορυφές και m ακμές σύμφωνα με το μοντέλο $G_{n,m}$

- Μια τυχαία ακμή επιλέγεται από ένα σύνολο C που ορίζεται ως εξής
  - Το C αρχικοποιείται με το σύνολο όλων των n^2 ζευγών των κορυφών, αν `loopfree = false`, ή με το σύνολο όλων των $n(n-1)$ ζευγών διαφορετικών κορυφών, αν `loopfree = true`
  - Η επιλογή ενός ζεύγους (v, w) από το C συνεπάγεται τη διαγραφή του από το C αν `no_parallel_edges = true`, και το αντίθετο ζεύγος (w, v) διαγράφεται από το C αν `no_anti_parallel_edges = true`

Ειδικές περιπτώσεις της `random_graph`

```
// The following pairs of calls are equivalent

random_graph(G,n,m);
random_graph(G,n,m,false,false,false);

random_simple_graph(G,n,m);
random_graph(G,n,m,false,false,true);

random_simple_loopfree_graph(G,n,m);
random_graph(G,n,m,false,true,true);

random_simple_undirected_graph(G,n,m);
random_graph(G,n,m,true,true,true);
```

Γραφήματα στην LEDA – δημιουργία γραφημάτων

- Η συνάρτηση

  ```
  void test_graph(graph& G)
  ```

  δημιουργεί διαλογικά ένα γράφημα που ορίζεται από τον χρήστη

Τέλος Ενότητας

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση **1.0**.

Σημείωμα Ιστορικού Εκδόσεων Έργου

Σημείωμα Ιστορικού Εκδόσεων Έργου

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