Ενότητα 10
Γράφοι (ή γραφήματα)

Γράφοι (ή Γραφήματα)

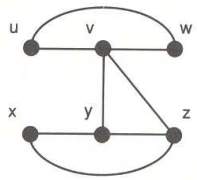
- Ένας γράφος αποτελείται από ένα σύνολο από σημεία (που λέγονται **κόμβοι**) και ένα σύνολο από γραμμές (που λέγονται **ακμές**) οι οποίες συνδέουν ζεύγη κόμβων.

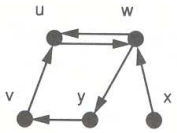
Εφαρμογές

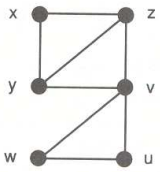
Μοντελοποίηση πολλών προβλημάτων:

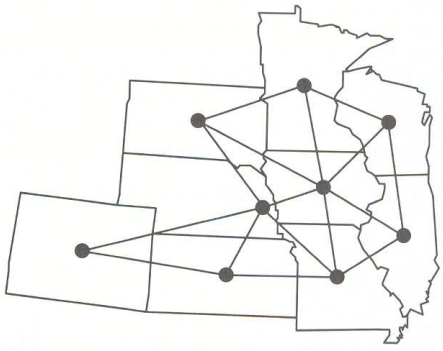
- Αεροπορικές πτήσεις μεταξύ κάποιων πόλεων.
- Συνηθίζεται στη δημιουργία χαρτών να ζωγραφίζονται γειτονικές χώρες (νομοί) με διαφορετικό χρώμα. Το πρόβλημα αυτό μπορεί να μοντελοποιηθεί σαν ένα πρόβλημα γράφων.
- Πολλά παιχνίδια μπορούν να μοντελοποιηθούν με χρήση γράφων.
- Traveling Salesman Problem (Πρόβλημα περιπλανώμενου πωλητή): Δεδομένου ενός συνόλου από πόλεις και της απόστασης μεταξύ κάθε ζεύγους πόλεων, βρείτε τη συντομότερη διαδρομή που επισκέπτεται κάθε πόλη (δηλαδή εκείνη στην οποία διανύεται η μικρότερη απόσταση).

Παραδείγματα Γράφων

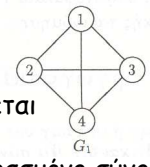




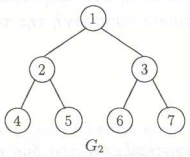




Χρήσιμη Ορολογία



G_1



G_2

G_3

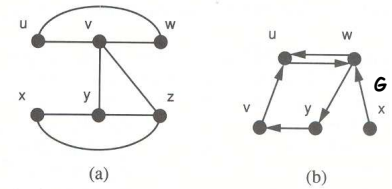
- Ένας **γράφος** G χαρακτηρίζεται από δύο σύνολα V και E.
  Το σύνολο V είναι ένα πεπερασμένο σύνολο, που περιέχει ως στοιχεία τις **κορυφές** (vertices) ή **κόμβους** (nodes) ή **σημεία** (points) του γράφου. Το σύνολο E περιέχει τα ζεύγη κορυφών του γράφου, τα οποία ορίζουν τις **ακμές** (edges) ή **τόξα** (arcs) ή **συνδέσμους** (links) του.
- Οι κόμβοι ή οι ακμές ενός γράφου χαρακτηρίζονται από ένα μοναδικό όνομα που ονομάζεται **ετικέτα** (label).

- **V(G) ή V:** το σύνολο των κόμβων ενός γράφου G
- **E(G) ή E:** το σύνολο των ακμών ενός γράφου G
- **G(V,E):** γράφος με σύνολο κόμβων V και σύνολο ακμών E

Παραδείγματα
$V(G_1) = \{1,2,3,4\}$, $E(G_1) = \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}$
$V(G_2) = \{1,2,3,4,5,6,7\}$, $E(G_2) = \{(1,2), (1,3), (2,4), (2,5), (3,6), (3,7)\}$

- **Γράφος με βάρη στις ακμές** (weighted graph) λέγεται ένας γράφος, όπου με κάθε ακμή του έχει συσχετισθεί ένας αριθμός που ονομάζεται **βάρος** (weight).

Χρήσιμη Ορολογία

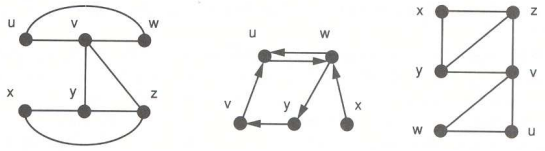


- Ένας γράφος είναι **μη κατευθυνόμενος** αν τα ζεύγη των κορυφών που ορίζουν τις ακμές του στερούνται διάταξης, π.χ., τα ζεύγη (v, u) και (u, v) αναφέρονται στην ίδια ακμή.
- Στους **κατευθυνόμενους γράφους** κάθε ακμή συμβολίζεται με το κατευθυνόμενο ζεύγος <v, u>, όπου v είναι η **ουρά** (tail) και u είναι η **κεφαλή** (head) της ακμής (έτσι, οι ακμές <v, u> και <u, v> είναι δυο διαφορετικές ακμές).
- Ένας μη κατευθυνόμενος γράφος μπορεί να θεωρηθεί ως ένας συμμετρικός κατευθυνόμενος γράφος.

Παράδειγμα

- Οι γράφοι G και G3 είναι κατευθυνόμενοι ενώ ο γράφος του σχήματος (a) είναι μη-κατευθυνόμενος.
- $V(G_3) = \{1,2,3\}$, $E(G_3) = \{<1,2>,<2,1>,<2,3>\}$
- $V(G) = \{u,v,w,x,y\}$, $E(G) = \{<u,w>, <w,u>, <w,y>, <x,w>, <y,v>,<v,u>\}$.

Χρήσιμη Ορολογία



- Αν $(v, u) \in E(G)$, τότε οι κορυφές v και u λέγονται **διπλανές** (adjacent) ή **γειτονικές** (neighboring) και η ακμή (v, u) ονομάζεται **προσκείμενη** στις κορυφές v και u.
- Αν δύο κορυφές v και u δεν συνδέονται μεταξύ τους με ακμή λέγονται **ανεξάρτητες** (independent).
- Αν (v, u) είναι μια ακμή τότε η κορυφή v λέγεται **διπλανή** (adjacent) της u. Επίσης, οι κορυφές v και u λέγονται **γειτονικές**.
- Αν <v,u> είναι μια ακμή ενός κατευθυνόμενου γράφου, τότε ο κόμβος v είναι **γειτονικός** του κόμβου u, αλλά το αντίστροφο ισχύει μόνο αν και η ακμή <u, v> υπάρχει επίσης στον κατευθυνόμενο γράφο.

- Ένας γράφος με πολλές ακμές λέγεται **πυκνός** (συνήθως με $\Theta(n \log n)$ ή περισσότερες ακμές).
- Ένας γράφος με λίγες ακμές (συνήθως λιγότερες από $O(n)$) λέγεται **αραιός**.

Θεμιτές Λειτουργίες σε Γράφους

- **MakeGraph(V):** επιστρέφει έναν γράφο με σύνολο κορυφών V και καμία ακμή.
- **Vertices(G):** επιστρέφει V(G), το σύνολο των κορυφών του G.
- **Edges(G):** επιστρέφει E(G), το σύνολο των ακμών του G.
- **Neighbors(G,v):** επιστρέφει το σύνολο των κορυφών που είναι γειτονικές του κόμβου v στον G.
- **AddVertex(G,v):** Προσθέτει ένα νέο κόμβο με ετικέτα v στον G.
- **AddDirectedEdge(G,u,v):** προσθέτει μια νέα (κατευθυνόμενη) ακμή <u,v> που συνδέει τους κόμβους u και v στον G.
- **AddUndirectedEdge(G,u,v):** προσθέτει τη νέα ακμή (u,v) που συνδέει τους κόμβους u και v στον G.
- **DeleteVertex(G,v):** διαγράφει τον κόμβο v από τον G, μαζί με όλες τις ακμές που πρόσκεινται σε αυτόν.
- **DeleteEdge(G,u,v):** διαγράφει την ακμή που συνδέει τους κόμβους u και v στον G.

Αναπαράσταση Γράφων

Έστω ένας γράφος G του οποίου οι κόμβοι έχουν αριθμηθεί από το 1 ως το |V| κατά αυθαίρετο τρόπο.

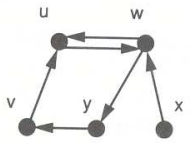
Ο πίνακας γειτνίασης A του G είναι ένας |V|x|V| πίνακας με στοιχεία:

$$A[i,j] = \begin{cases} 1 & \text{αν (i,j)} \in E \\ 0 & \text{διαφορετικά} \end{cases}$$

Ορίζουμε ως ανάστροφο ενός πίνακα γειτνίασης A τον πίνακα $A^T = \{a_{ij}^T\}$, όπου $a_{ij}^T = a_{ji}$, για κάθε i,j ∈ {1,..,|V|}.

- Ο πίνακας γειτνίασης ενός μη-κατευθυνόμενου γράφου ισούται με τον ανάστροφό του.
  - Για κάθε εγγραφή A[i,j] του πίνακα που είναι ίση με 1, η εγγραφή A[j,i] ισούται επίσης με 1.

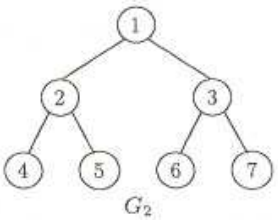
Παραδείγματα



Κατευθυνόμενος γράφος G_1

	u	v	w	x	y
u	0	0	1	0	0
v	1	0	0	0	0
w	1	0	0	0	1
x	0	0	1	0	0
y	0	1	0	0	0

Πίνακας γειτνίασης του G_1



G_2

Θετικά & Αρνητικά Πινάκων Γειτνίασης

Θετικά

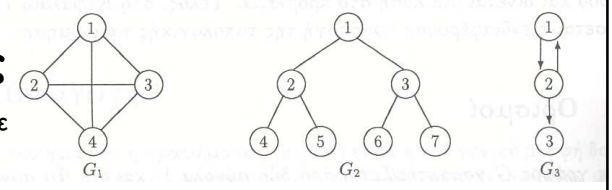
- Αν δεν χρειάζεται να αποθηκευτούν επιπρόσθετες πληροφορίες για κάθε κόμβο του γράφου, η μέθοδος είναι πολύ ελκυστική. Κάθε κόμβος χαρακτηρίζεται από έναν ακέραιο και οι ακέραιοι αυτοί χρησιμοποιούνται για διευθυνσιοδότηση του πίνακα γειτνίασης.
- Πολλές λειτουργίες υλοποιούνται απλά και αποτελεσματικά.

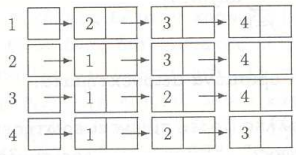
Αρνητικά

- Κάθε διαγραφή ή εισαγωγή κόμβου στο γράφο προκαλεί αλλαγή στο μέγεθος του πίνακα.
- Η μέθοδος πίνακα γειτνίασης είναι απλή, αλλά δεν υποστηρίζει αποτελεσματικά όλες τις λειτουργίες.
  - Ποια η πολυπλοκότητα της λειτουργίας εύρεσης των γειτονικών κόμβων ενός κόμβου υ?
  - Θα μπορούσαμε να υλοποιήσουμε τη λειτουργία αυτή πιο αποτελεσματικά αν γνωρίζαμε πως ο κόμβος δεν έχει καθόλου γειτονικούς κόμβους ή έχει πολύ λίγους?

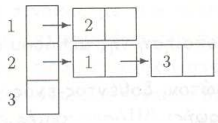
Λίστες Γειτνίασης

- Οι κόμβοι αποθηκεύονται σε μια (στατική ή δυναμική) λίστα. Σε κάθε κόμβο v της λίστας αποθηκεύεται δείκτης στο πρώτο στοιχείο μιας λίστας που περιέχει τους γειτονικούς κόμβους του v στο γράφο.
- Η λίστα γειτνίασης είναι μια λίστα από λίστες!





Λίστα Γειτνίασης G_1



Λίστα Γειτνίασης G_3

Θετικά

Ένας κόμβος μπορεί να εισαχθεί ή να διαγραφεί με την ίδια ευκολία όπως μια ακμή.
Οδηγεί σε καλή χωρική πολυπλοκότητα για την αναπαράσταση αραιών γράφων.

Αρνητικά

Η χρονική πολυπλοκότητα της λειτουργίας «Εύρεση αν δύο κόμβοι είναι γειτονικοί» είναι μεγαλύτερη από όταν χρησιμοποιείται πίνακας γειτνίασης.
Ακριβή μέθοδος ως προς τη χωρική πολυπλοκότητα για πυκνούς γράφους.

Πολυπλοκότητα Αλγορίθμων Γράφων

Πρόταση: Το μέγιστο πλήθος ακμών για κάθε μη κατευθυνόμενο γράφο με n κορυφές είναι $E_{max} = n(n-1)/2$. Γιατί;

Τι ισχύει αν ένας γράφος *G* είναι κατευθυνόμενος?
Ο G μπορεί να έχει το πολύ διπλάσιο πλήθος ακμών από τον αντίστοιχο μη κατευθυνόμενο γράφο.

Τι θα ήταν καλύτερο, ένας αλγόριθμος που τρέχει σε χρόνο $\Theta(n^2)$ ή σε $\Theta(m)$?

- Αλγόριθμοι με πολυπλοκότητα $\Theta(m)$ συνήθως δεν μπορούν να σχεδιαστούν, αφού αν το $|E|$ είναι μικρό, δεν αρκεί ο χρόνος ούτε για να εξεταστεί κάθε κόμβος.
- Πολλές φορές η γνώση του αν ο γράφος είναι πυκνός ή αραιός βοηθάει στο σχεδιασμό αποτελεσματικών αλγόριθμων.
- Η χρονική πολυπλοκότητα γράφων είναι συνήθως συνάρτηση τόσο του αριθμού των κόμβων, όσο και του αριθμού των ακμών, π.χ. $\Theta(n+m)$.
- Άλλοι παράγοντες που επηρεάζουν σημαντικά την πολυπλοκότητα είναι η υλοποίηση (δηλαδή η μέθοδος αναπαράστασης που χρησιμοποιείται).
  - Πως θα μπορούσε να υλοποιηθεί μια εντολή ανακύκλωσης του τύπου "για κάθε ακμή e στον *G*" δεδομένου ότι ο *G* υλοποιείται με:
    - ✓ Πίνακα γειτνίασης;
    - ✓ Λίστες γειτνίασης;

Διάσχιση «Κατά Πλάτος»(Breath First Search ή BFS)

Δεδομένων ενός γραφήματος *G*=(V,E) και ενός κόμβου $s \in V$ (**κόμβος ρίζα** ή **κόμβος εκκίνησης**), η διάσχιση κατά πλάτος συνίσταται στον εντοπισμό (δηλαδή τη διάσχιση) όλων των κόμβων που είναι προσπελάσιμοι από τον *s*.

Κατά την εκτέλεση μιας διάσχισης κατά πλάτος στον *G*:

- ❑ δημιουργείται ένα δένδρο με ρίζα τον κόμβο *s* που περιέχει όλους τους κόμβους του *G* που είναι προσπελάσιμοι από τον *s* (το δένδρο αυτό ονομάζεται δένδρο «κατά πλάτος»),
- ❑ υπολογίζεται η απόσταση, δηλαδή το μήκος του συντομότερου μονοπατιού, από τον *s* προς οποιονδήποτε κόμβο *v* (την οποία θα συμβολίζουμε με $\delta(s,v)$).

Η διάσχιση των κόμβων γίνεται «κατά πλάτος»:

«Η διάσχιση οποιουδήποτε κόμβου σε απόσταση k+1 από τον *s* πραγματοποιείται μόνο όταν όλοι οι κόμβοι σε απόσταση k από τον *s* έχουν διασχισθεί».

Διάσχιση «Κατά Πλάτος»

Σημαντικότερα σημεία του αλγορίθμου

- Ο αλγόριθμος χρωματίζει κάθε κόμβο του γράφου λευκό, γκρίζο ή μαύρο.
  - **Λευκοί κόμβοι:** είναι αυτοί που δεν έχουν ακόμη εξερευνηθεί. Ένας κόμβος θεωρείται πως έχει εξερευνηθεί την πρώτη φορά που συναντάται στη διάσχιση, οπότε και καθίσταται μη λευκός.
  - **Γκρίζοι κόμβοι:** είναι κόμβοι που έχουν εξερευνηθεί αλλά ίσως έχουν γείτονες που δεν έχουν ακόμη εξερευνηθεί (δηλαδή που είναι ακόμη λευκοί). Αποτελούν το σύνορο μεταξύ μαύρων και λευκών κόμβων.
  - **Μαύροι κόμβοι:** είναι κόμβοι των οποίων όλοι οι γείτονες έχουν εξερευνηθεί (δηλαδή οι γειτονικοί κόμβοι αυτών είναι είτε γκρίζοι ή μαύροι).
- Ο διαχωρισμός μεταξύ γκρίζων και μαύρων κόμβων γίνεται για να εξασφαλισθεί ότι η διάσχιση των κόμβων θα γίνει «κατά πλάτος».

Διάσχιση «Κατά Πλάτος»

Βασικές Ιδέες Αλγορίθμου

- Αρχικά, ο μοναδικός γκρίζος κόμβος είναι ο s.
- Κάθε χρονική στιγμή, οι γκρίζοι κόμβοι είναι αποθηκευμένοι σε μια ουρά Q (που αρχικά περιέχει μόνο τον s).
- Επαναληπτικά, εκτελούνται τα εξής:
  1. Εξαγωγή ενός γκρίζου κόμβου u από την Q;
  2. Διερεύνηση των γειτονικών κόμβων του u και εισαγωγή στην ουρά όσων εξ αυτών είναι λευκοί.
  3. Ακριβώς πριν την εισαγωγή στην Q ενός λευκού κόμβου v εκτελούνται τα εξής:
     - Το χρώμα του v αλλάζει σε γκρίζο.
     - Ο u ορίζεται να είναι ο γονικός κόμβος του v στο «κατά πλάτος» δένδρο που δημιουργείται και η ακμή (u,v) εισάγεται στο δένδρο (ο u ονομάζεται **προκάτοχος** του v στο δένδρο).
     - Η απόσταση του v από τον κόμβο εκκίνησης s υπολογίζεται και καταχωρείται σε ένα κατάλληλο πεδίο του struct του κόμβου v.
  4. Όταν όλοι οι γειτονικοί κόμβοι του u έχουν εξετασθεί (και οι ενέργειες του βήματος 3 έχουν εκτελεστεί για όσους εξ αυτών είναι λευκοί), το χρώμα του u αλλάζει σε μαύρο.

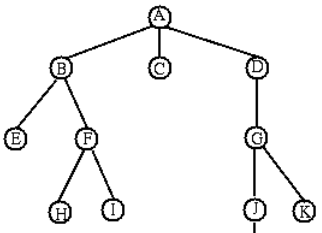
Διάσχιση «Κατά Πλάτος»

Η διάσχιση «κατά πλάτος» σας θυμίζει κάποια από τις διασχίσεις που έχουν μελετηθεί σε δένδρα;

Διάσχιση Δένδρου κατά Επίπεδα (κατά πλάτος)
Επισκέπτεται τους κόμβους κατά αύξον βάθος.

Χρήση Ουράς
- Αρχικά η ουρά περιέχει μόνο τη ρίζα.
- Επαναληπτικά: κάνουμε Deque ένα στοιχείο της ουράς και προσθέτουμε τα παιδιά από αριστερά προς τα δεξιά του στοιχείου αυτού.

```
Procedure LevelOrder(pointer r) {
  Queue Q;  pointer P;
  MakeEmptyQueue(Q); Enqueue(Q,r);
  while (! IsEmptyQueue(Q)) {
     P = Dequeue(Q);
     Visit(P);
     foreach child c of P, in order, do
          Enqueue(c);
  } }
```



Παράδειγμα

Περιεχόμενα Ουράς
A
B, C, D
C, D, E, F
D, E, F
E, F, G
F, G
G, H, I
H, I, J, K
I, J, K
J, K
K, L
L
<empty>

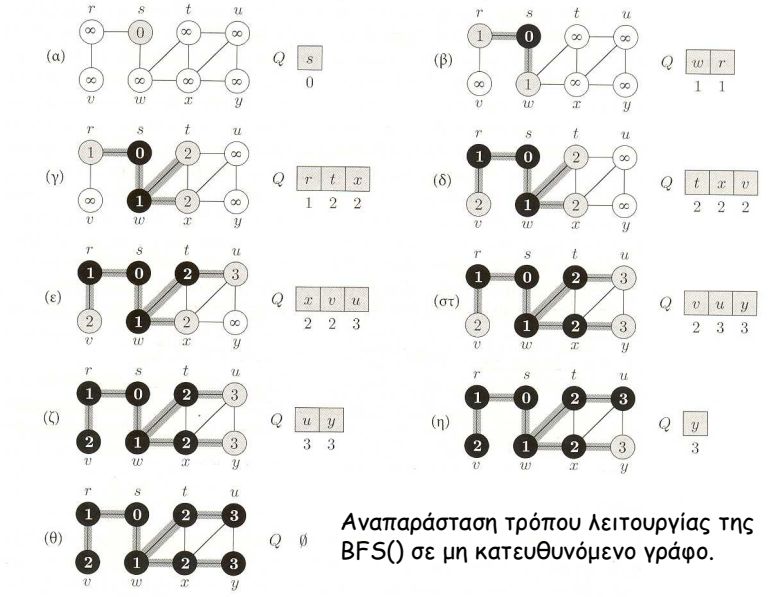
Διάσχιση «Κατά Πλάτος»

- Ο κώδικας υποθέτει ότι ο γράφος G αναπαρίσταται μέσω λίστας γειτνίασης.
- Σε κάθε κόμβο v του γράφου διατηρούνται διάφορες πληροφορίες:
  - ο v->c: το χρώμα του κόμβου v
  - ο v->d: η απόσταση του κόμβου v από τον s
  - ο v->p: ο γονικός κόμβος του v στο κατά πλάτος δένδρο
- Ο αλγόριθμος χρησιμοποιεί μια ουρά Q για τη διαχείριση του συνόλου των γκρίζων κόμβων.

```
procedure BFS(graph G, node s) {
   Queue Q;
   node u, v;
   foreach node u ∈ V(G) – {s} {          // αρχικοποίηση
      u->c = WHITE;                       // των κόμβων
      u->d = ∞;                           // του γράφου
      u->p = nil;
   }
   s->c = GRAY;
   s->d = 0;                              // αρχικοποίηση κόμβου
   s->p = nill;
   MakeEmptyQueue(Q);                     // s και ουράς Q
   Enqueue(Q, s);
   while (!IsEmptyQueue(Q)) {
      u = Dequeue(Q);

      // διάσχιση λίστας γειτονικών κόμβων του u
      foreach v ∈ u->Adj {
          if (v->c = WHITE) {
             v->c = GRAY;
             v->d = u->d+1;
             v->p = u;
             Enqueue(Q,v);
          }
      }
      u->c = BLACK;
   }
```

Διάσχιση «Κατά Πλάτος»



Αναπαράσταση τρόπου λειτουργίας της BFS() σε μη κατευθυνόμενο γράφο.

Διάσχιση «Κατά Πλάτος»

- Τα αποτελέσματα της διάσχισης «κατά πλάτος» είναι πιθανό να εξαρτώνται από τη σειρά με την οποία εξετάζονται οι γείτονες ενός δεδομένου κόμβου.
  - το τελικό δένδρο μπορεί να ποικίλλει αλλά οι αποστάσεις που προκύπτουν είναι πάντα οι ίδιες.

Χρονική Πολυπλοκότητα

- Κάθε κόμβος εισάγεται στην ουρά και αφαιρείται από αυτήν το πολύ μια φορά -> $O(1)$
  - Συνολικός χρόνος που αναλώνεται στις λειτουργίες της ουράς -> $O(n)$
- Η λίστα γειτνίασης κάθε κόμβου διατρέχεται μία μόνο φορά. Το άθροισμα των μεγεθών όλων των λιστών γειτνίασης είναι $O(m)$.
- Επιβάρυνση από την απόδοση αρχικών τιμών -> $O(n)$.
- Ο συνολικός χρόνος εκτελέσης της διάσχισης «Κατά πλάτος» είναι $O(n+m)$.

Διάσχιση «Κατά βάθος» (Depth First Search)

- Η διάσχιση επεκτείνεται (όσο αυτό είναι δυνατό) προς κόμβους σε μεγαλύτερα «βάθη» στο γράφο.
- Οι ακμές εξερευνούνται με αφετηρία τον πιο πρόσφατα εντοπισμένο κόμβο v από τον οποίο εκκινούν μη εξερευνημένες ακμές.
- Αφού εξερευνηθούν όλες οι ακμές του v, η διάσχιση επιστρέφει τον κόμβο από τον οποίο εντοπίστηκε ο v και συνεχίζεται με τις τυχόν άλλες ακμές που εκκινούν από αυτόν.
- Αν εξακολουθούν να υπάρχουν μη-εντοπισμένοι κόμβοι, επιλέγεται ένας από αυτούς και η διάσχιση συνεχίζεται από αυτόν.
- Η όλη διαδικασία επαναλαμβάνεται έως ότου να πραγματοποιηθεί η διάσχιση όλων των κόμβων.

Διάσχιση «Κατά βάθος»

- Κατά τη διάρκεια της διάσχισης, αποδίδονται στους κόμβους χρώματα τα οποία υποδεικνύουν την κατάστασή τους.
  - λευκοί κόμβοι (ανεξερεύνητοι κόμβοι)
  - γκρίζοι κόμβοι (εντοπισμένοι κόμβοι των οποίων η εξερεύνηση δεν έχει ακόμη τελειώσει)
  - μαύροι κόμβοι (η εξερεύνηση τους έχει τελειώσει)
- Κάθε κόμβος συσχετίζεται με δύο χρονοσφραγίδες:
  - d[v]: καταγράφει τη χρονική στιγμή που εντοπίζεται για πρώτη φορά ο v
  - f[v]: καταγράφει τη χρονική στιγμή που ολοκληρώνεται η εξέταση του καταλόγου γειτνίασης του v

$$1 \leq d[u] < f[u] \leq 2n$$

Διάσχιση «Κατά βάθος»

```
DFS(Graph G) {
   for each vertex u ∈ V[G] {
         u -> c = WHITE;
         u -> p = nil;
   }
   time  = 0;
   for each vertex u ∈ V[G] {
         if (u -> c == WHITE)
                DFS-Visit(u);
   }
}
```

```
DFS-Visit(Node u) {
    u -> c = GRAY;
    time = time + 1;
    u -> d = time;
    for each v ∈ Adj[u] {
         if (v -> c == WHITE) {
            v -> p = u;
            DFS-Visit(v);
    }
    u -> c = BLACK;
    u->f = time + 1;
}
```


Διάσχιση «Κατά βάθος»

Χρόνος Εκτέλεσης DFS

- Χρόνος εκτέλεσης 1ου for της DFS() -> O(n)
- Χρόνος εκτέλεσης 2ου for της DFS() χωρίς να υπολογίζουμε τον χρόνο που απαιτείται για την Dfs-Visit() -> O(n)
- Η DFS-Visit() καλείται ακριβώς μια φορά για κάθε κόμβο (αφού καλείται μόνο αν ο κόμβος είναι λευκός) και αμέσως μετά την κλήση της ο κόμβος χρωματίζεται γκρίζος και δεν ξαναγίνεται ποτέ λευκός.
- Το συνολικό κόστος για την εκτέλεση της for της DFS-Visit() είναι Θ(m).
- Το συνολικό κόστος εκτέλεσης της DFS είναι O(n+m).

Διάσχιση «Κατά βάθος»