Διαπεράσεις Μη Κατευθυνόμενων Γραφημάτων

- Κάθε διαδικασία συστηματικής και εξαντλητικής εξερευνήσεως ενός γραφήματος, με την εξέταση των κορυφών και ακμών του
- Παρ' όλη την απλότητά τους, οι διαπεράσεις είναι ισχυρές και ικανές να ανακαλύψουν τις ιδιότητες των γραφημάτων

Γενικός Αλγόριθμος Διαπεράσεως

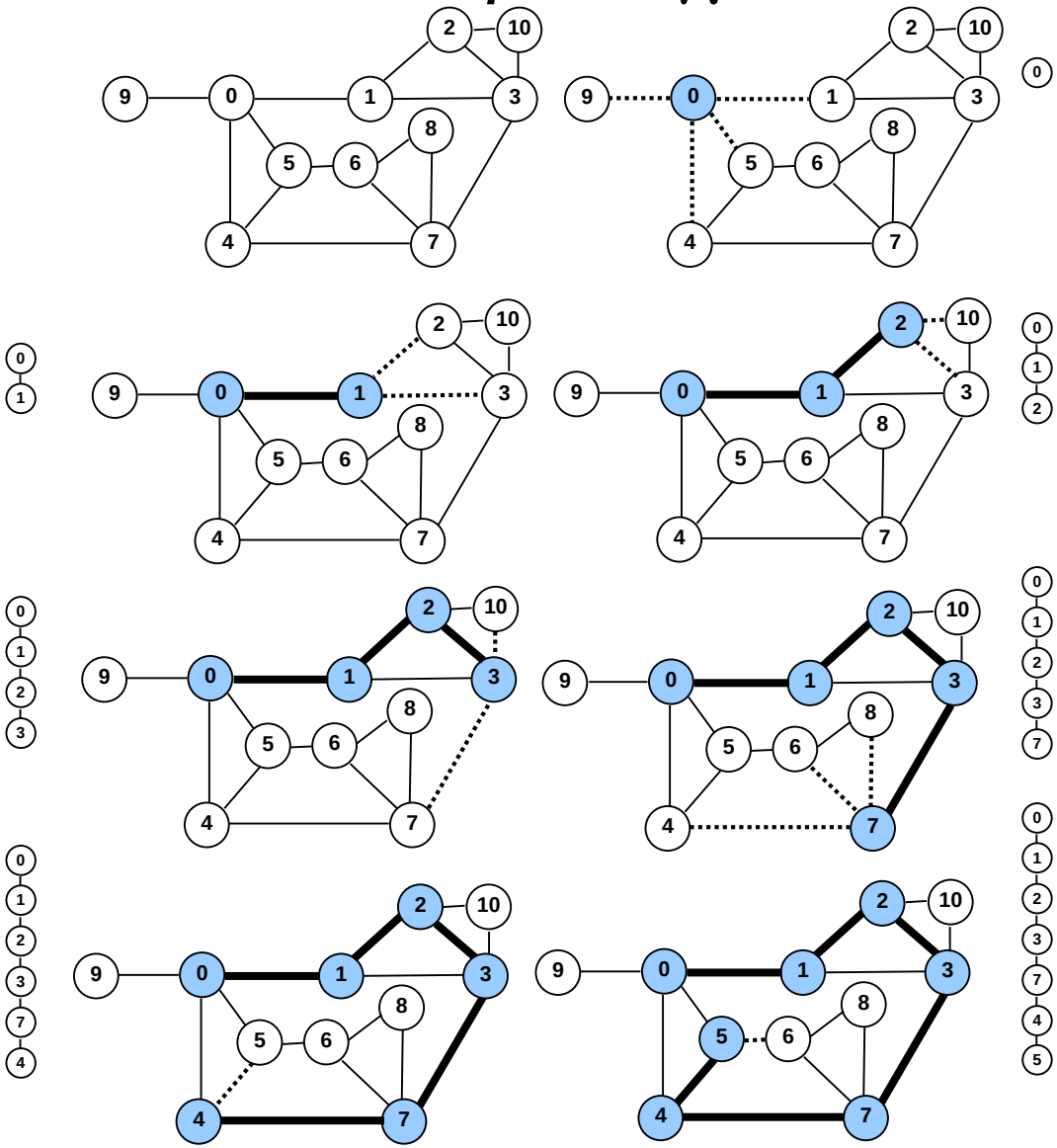
Algorithm GRAPHsearch(graph G)
Input: Γράφημα G
Output: Διαπέραση βάσει graphSearch

```
1.  int v, order=0;
2.  for (v = 0; v < G.V; v++)
3.      G.pre[v] = -1;
4.  for (v = 0; v < G.V; v++)
5.      if (G.pre[v] == -1)
6.          graphSearch(G, v);
```

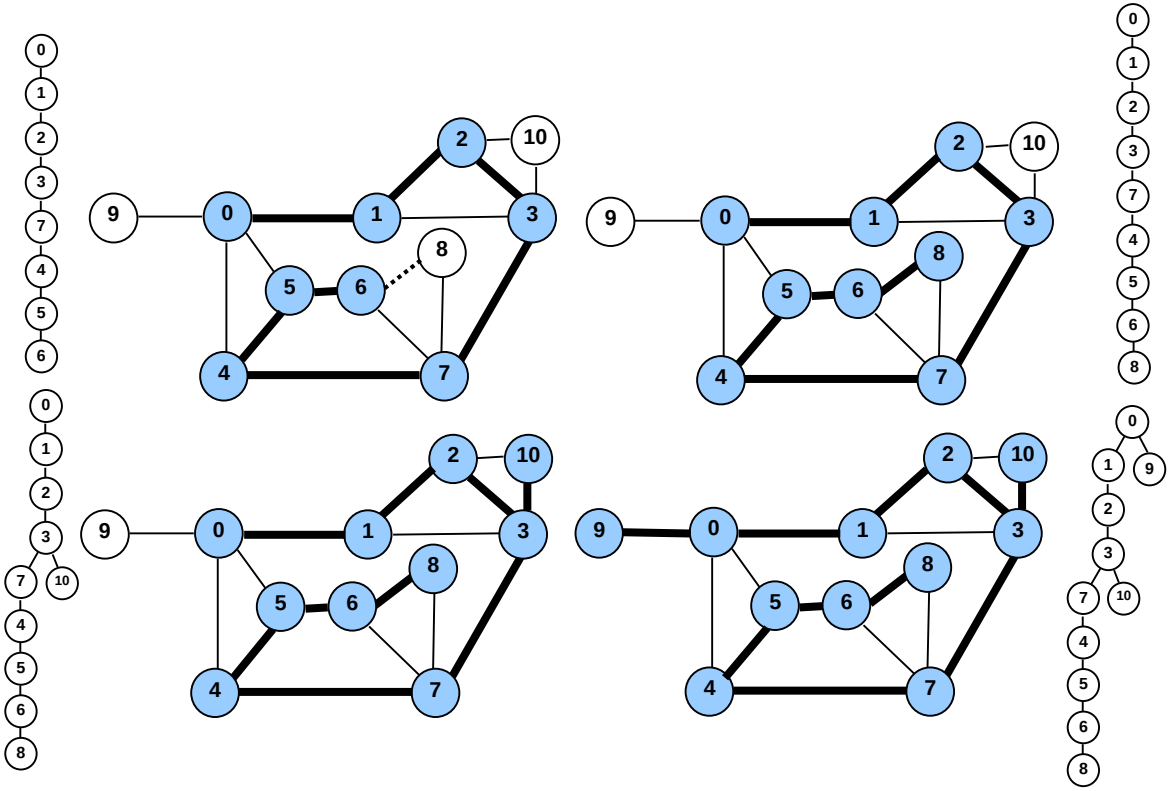
Αναζήτηση σε Βάθος

- Αρχικά όλες οι κορυφές θεωρούνται ως ανεξέταστες
- Μέχρι να εξερευνήσουμε όλες τις κορυφές, επαναληπτικά, κάνουμε τα ακόλουθα:
  - Επιλέγουμε μία ανεξερεύνητη και την κηρύσσουμε εξερευνημένη
  - Εάν υπάρχει γειτονική της ανεξέταστη, την επιλέγουμε ως επόμενη προς θεώρηση κορυφή
  - Η επιλογή γίνεται βάσει ονόματος (συνήθως επιλέγεται το μικρότερο διαθέσιμο)

Παράδειγμα



Παράδειγμα (συν.)



Υλοποίηση

Algorithm dfsList(graph G, vertex v)

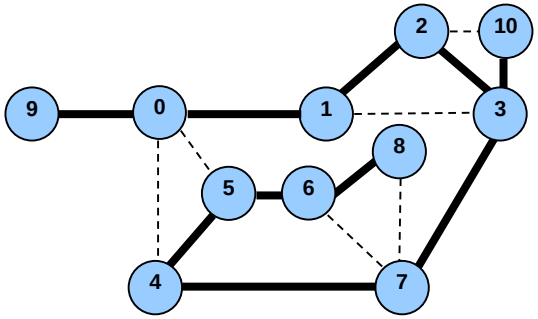
```
1.  G.pre[v] = order++;
2.  for (x = G.List[v]; x != null; x = x.getNext())
3.        if (pre[x.v] == -1)
4.           dfsList(G, x.v);
```

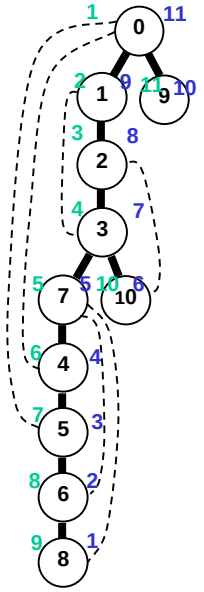
Algorithm dfsMatrix(graph G, vertex v)

```
1.  G.pre[v] = order++;
2.  for (w = 0; w < G.V; w++)
3.     if (G.A[v][w] != 0)
4.        if (pre[w] == -1)
5.           dfsMatrix(G, w);
```

- **Χρόνος:** Γραμμικός στο μέγεθος του *G* (κάθε ακμή εξετάζεται το πολύ δύο φορές και κάθε κορυφή ακριβώς μία φορά)

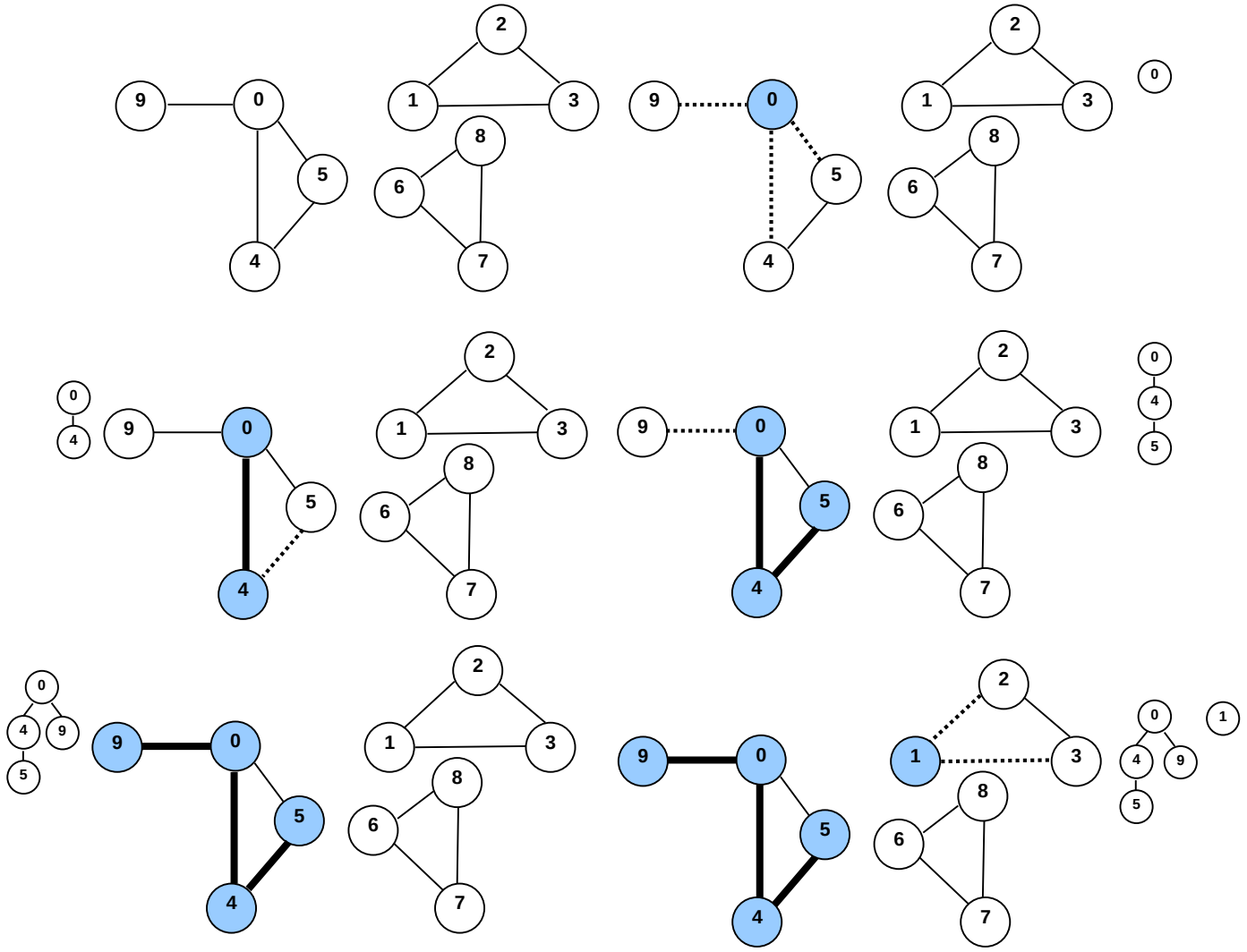
Δένδρο ΑσΒ

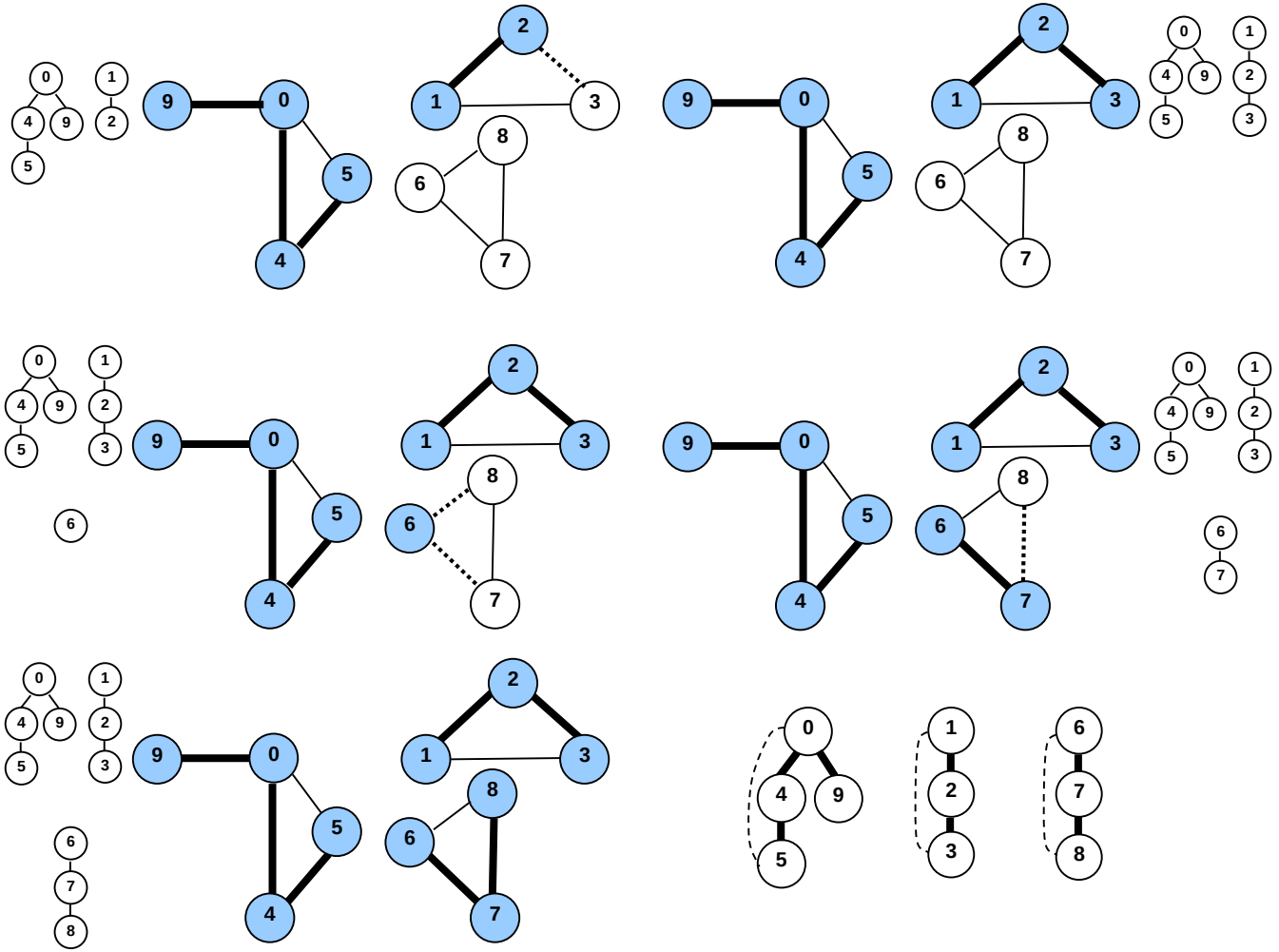




- Κατηγορίες ακμών
  - Ακμές δένδρου (ακμές της εξερεύνησης)
  - Οπισθοακμές (ήδη εξερευνημένες)
- Αριθμοί προδιατάξεως και μεταδιατάξεως (περατώσεως)
  - *v* απόγονος της *w* εάν έχει μεγαλύτερο αριθμό προδιατάξεως και μικρότερο αριθμό μεταδιατάξεως
  - *v* πρόγονος της *w* εάν έχει μικρότερο αριθμό προδιατάξεως και μεγαλύτερο αριθμό μεταδιατάξεως
  - σε κάθε άλλη περίπτωση δεν σχετίζονται...
- Κατασκευή ανάστροφου δένδρου ΑσΒ μέσω βοηθητικού πίνακα, όπου θα σημειώνεται ο πατέρας κάθε ανακαλυφθείσας κορυφής (γιατί συμφέρει;)

Παράδειγμα μη Συνεκτικού μη Κατευθυνόμενου Γραφήματος




0 1 4 9 2 5
9 0 5 4
2 1 3 8 6 7
9 0 5 4
2 1 3 8 6 7
0 1 4 9 2 5 3
6 7
9 0 5 4
2 1 3 8 6 7
9 0 5 4
2 1 3 8 6 7
0 1 4 9 2 5 3
6 7
9 0 5 4
2 1 3 8 6 7
0 4 9 5
1 2 3
6 7 8

Εφαρμογές ΑσΒ

- **Εντοπισμός Κύκλου:**

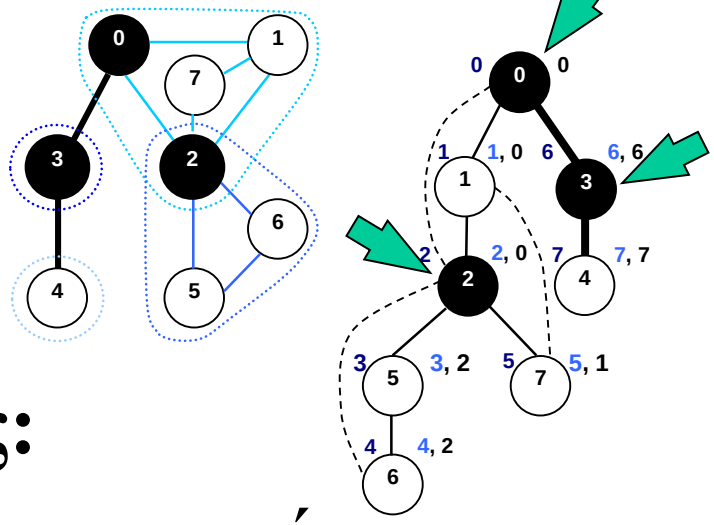
  Αφ' ης στιγμής ανακαλυφθεί οπισθοακμή
- **Εύρεση Απλού Μονοπατιού μεταξύ *v* και *w*:**

  Εκκίνηση ΑσΒ από την *v* (απόδειξη με επαγωγή στο μήκος του μονοπατιού)
- **Επικαλύπτον Δένδρο ή Δάσος (απλή συνεκτικότητα):**

  Εάν το γράφημα είναι συνεκτικό, τότε το δένδρο ΑσΒ είναι επικαλύπτον.

  Διαφορετικά, κάθε επανεκκίνηση της ΑσΒ ανακαλύπτει και μία συνεκτική συνιστώσα (ελάχιστες οι τροποποιήσεις των αλγορίθμων- μόνο ένας πίνακας και μία βοηθητική μεταβλητή)

Εφαρμογές ΑσΒ (συν.)



- **Σημεία Αρθρώσεως:**
  Προϋποθέσεις για μία κορυφή *v*:
  - Είτε είναι ρίζα του ΑσΒ με ≥ 2 παιδιά
  - Είτε δεν υπάρχει απόγονός της *w* που να συνδέεται, μέσω οπισθοακμής, με απόγονό της *u* και, άρα, *όλα τα μονοπάτια μεταξύ των u,w περνούν από την v*

Algorithm articPoint(graph g, vertex v)

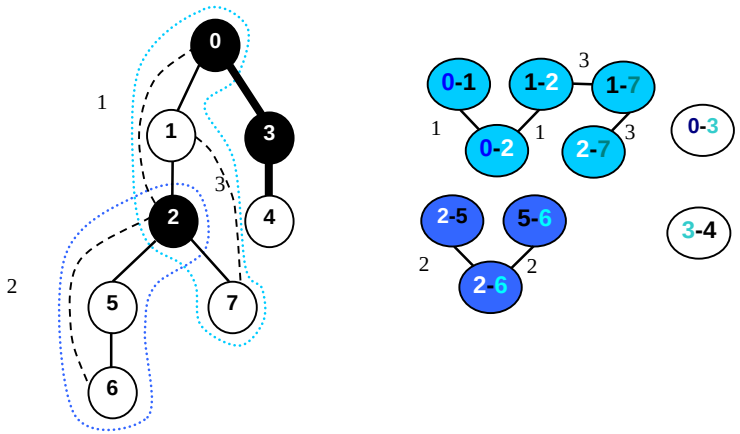
```
1.  g.low[v] = g.pre[v] = g.order++;
2.  for (x = g.List[v]; x != null; x = x.getNext())
3.    if (g.pre[w = x.v] == -1){    // δεν έχει εξερευνηθεί
4.       g.T[w] = v;
5.       articPoint(g, w);              // αναδρομική επίσκεψη στον w
6.       if (g.low[v] > g.low[w])    // ο w «βλέπει» απόγονο του v, επομένως τον βλέπει και ο v
7.          g.low[v] = low[w];
8.       if (g.low[w] >= g.pre[v])  // ο w δεν «βλέπει» ψηλότερα από τον  v
9.          g.artPoint.insLast(v);
10.   }
11.   else if ((w != g.T[v]) && (g.low[v] > g.pre[w])) // ο w πρόγονος και όχι πατέρας
12.      g.low[v] = g.pre[w];
```

- Ο g.low[v] περιέχει τον μικρότερο από: α) τον αριθμό προδιατάξεως της *v*, και β) από τον μικρότερο αριθμό προδιατάξεως κορυφής που μπορεί να ανακαλυφθεί από απλό μονοπάτι ακμών δένδρου ακολουθούμενο από μία οπισθοακμή
- Χρόνος: Γραμμικός λόγω ΑσΒ
- Μπορεί να εντοπίσει και τις γέφυρες (εξάσκηση)

Εφαρμογές ΑσΒ (συν.)

- **Δισυνεκτικές Συνιστώσες:**
  - **Πρώτη Αντιμετώπιση**
    - Σχέση ισοδυναμίας L επί του συνόλου ακμών E: ***eLo, e,o*** $\in$ ***E αν υπάρχει απλός κύκλος που τις περιέχει***
    - Οι κλάσεις ισοδυναμίας της L ***εξ ορισμού είναι και δισυνεκτικές συνιστώσες!***
    - Μία κορυφή είναι σημείο αρθρώσεως ***εάν συνδέεται με ακμές διαφορετικών κλάσεων ισοδυναμίας!***
    - Μία ακμή είναι γέφυρα ***εάν η κλάση ισοδυναμίας της περιέχει μόνον αυτήν!***

Εφαρμογές ΑσΒ (συν.)



- Εκτελούμε μία ΑσΒ, προσπαθώντας να κτίσουμε βοηθητικό γράφημα, το οποίο έχει τις κορυφές του 1-1 αντίστοιχες με το σύνολο των ακμών του. Για κάθε οπισθοακμή που ανακαλύπτουμε, ενώνουμε όλες τις κορυφές ακμές του κύκλου με την κορυφή-οπισθοακμή
- Με αποτέλεσμα, όλοι οι κύκλοι μίας δισυνεκτικής συνιστώσας να διασυνδέονται σε συνεκτική συνιστώσα στο βοηθητικό γράφημα, καθώς διαμοιράζονται ακμές μονοπατιού από ακμές δένδρου ΑσΒ
- Κόστος $O(VE)$ καθώς ένας κύκλος μπορεί να έχει $O(V)$ ακμές. Βελτίωση σε $O(V+E)$, εάν παραχθεί ένα επικαλύπτον δάσος του βοηθητικού γραφήματος

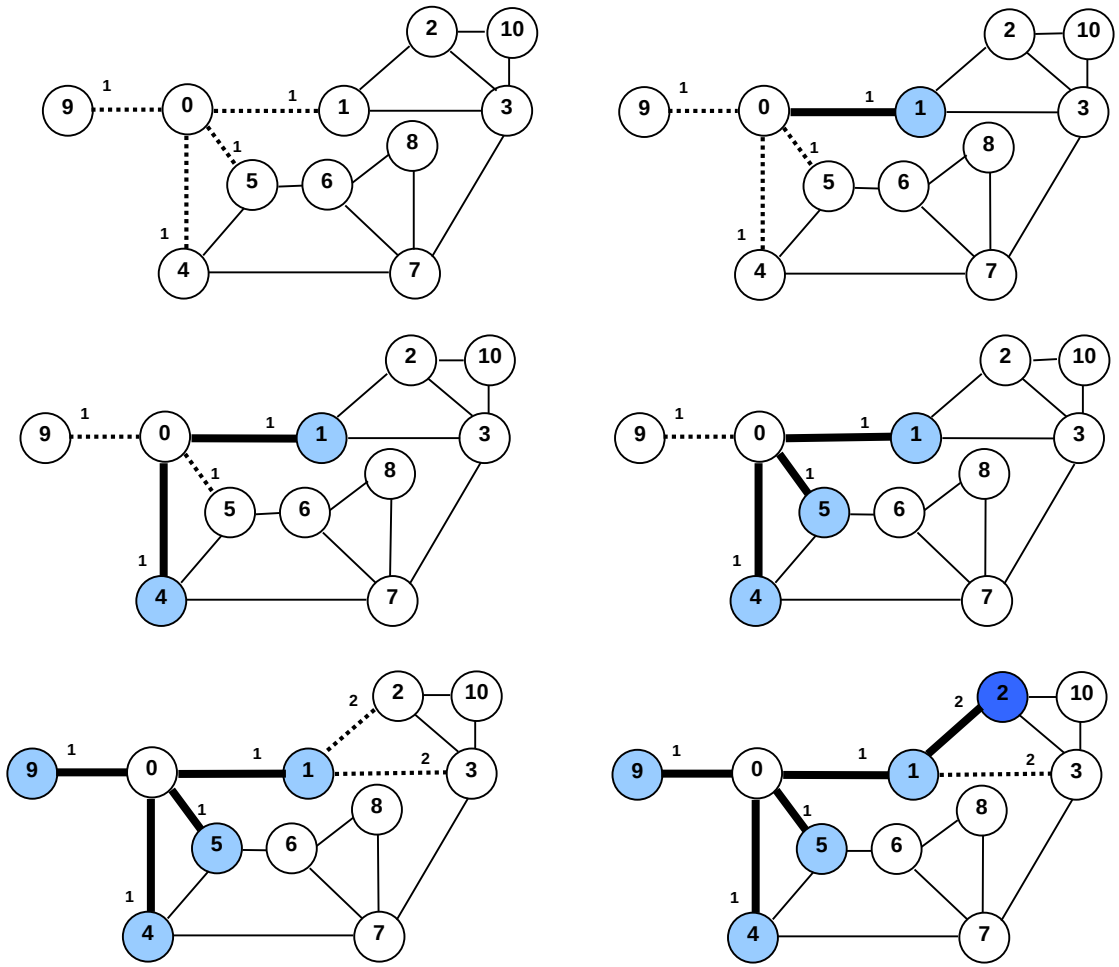
Εφαρμογές ΑσΒ (συν.)

- Εναλλακτικά, μπορεί να χρησιμοποιηθεί ο αλγόριθμος εντοπισμού των σημείων αρθρώσεως:
  - Αρκεί κάθε φορά που πρωτοανακαλύπτουμε είτε ακμή δένδρου είτε οπισθοακμή να την τοποθετούμε σε στοίβα και μετά, με διαδοχικά ποπ, μόλις ανακαλυφθεί το σημείο αρθρώσεως, να ανακτούμε όλη την συνιστώσα!!!
  - Τυπική απόδειξη με επαγωγή

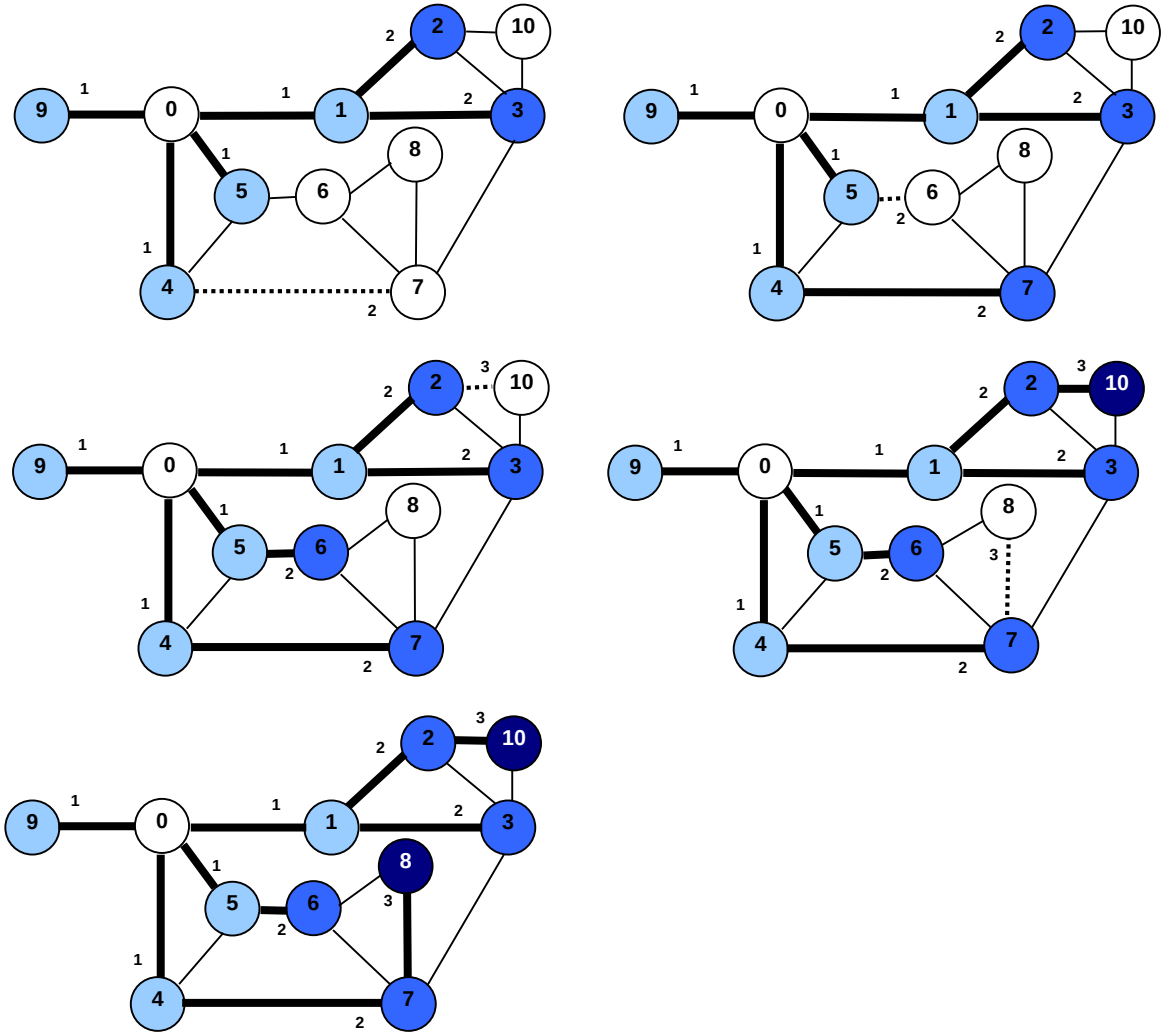
Αναζήτηση κατά Πλάτος (ΑκΠ)

- Εναλλακτική της ΑσΒ
- Πετυχαίνει διαπέραση των κορυφών κατά αύξοντα μήκη ελαχίστων μονοπατιών
- Επίσκεψη των κορυφών με χρήση FiFo, ώστε να μεταβαίνουμε σε όλες τις γειτονικές ανεξέταστες κορυφές κάθε εξετασμένης κορυφής
- Η ΔΙΑΠΕΡΑΣΗ ΓΙΝΕΤΑΙ **ΚΑΤΑ ΚΥΜΑΤΑ:**
  - Στο πρώτο, ανακαλύπτονται όλες οι κορυφές αποστάσεως 1, στο δεύτερο, όλες οι αποστάσεως 2 κοκ.

Παράδειγμα ΑκΠ



Παράδειγμα ΑκΠ (συν.)

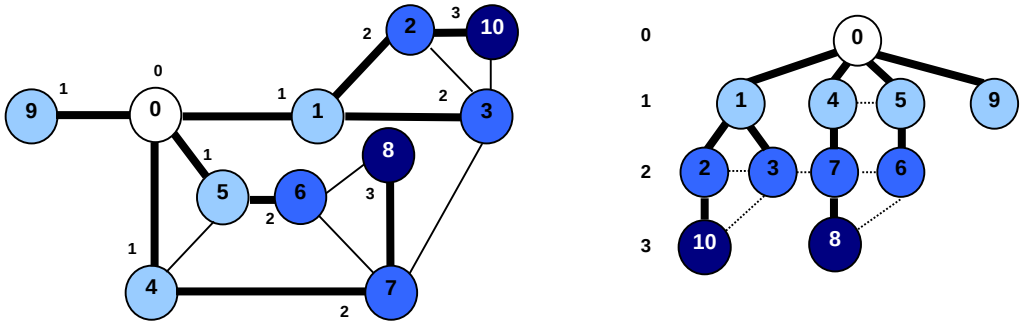


Algorithm bfsMatrix(graph G, vertex v)

```
1.  fifo.enqueue(v);
2.  G.pre[v] = order++;
3.  while (!fifo.isEmpty()){
4.    w = fifo.dequeue();
5.    for (x = 0; x < G.V; x++)
6.      if ((G.A[w][x] == 1) && (G.pre[x] == -1)){
7.         fifo.enqueue(x);
8.         G.pre[x] = order++;
9.      }
10. }
```

- **ΧΡΟΝΟΣ:** $O(V+E)$ καθώς κάθε κορυφή εισάγεται και εξάγεται από την ουρά μία φορά και κάθε ακμή εξετάζεται μία φορά από κάθε κατεύθυνση

Δένδρο ΑκΠ



- Κατηγορίες ακμών
  - Ακμές δένδρου
  - Διασταυρώσεως (μεταξύ επιπέδων που διαφέρουν κατά ένα)
  - ΔΕΝ ΥΠΑΡΧΟΥΝ ΟΠΙΣΘΟΑΚΜΕΣ (γιατί;)

Εφαρμογές

- **Εντοπισμός Κύκλου:**

  Αφ' ης στιγμής ανακαλυφθεί ακμή διασταυρώσεως
- **Εύρεση Απλού Μονοπατιού μεταξύ *v* και *w*:**

  Εκκίνηση ΑκΠ από την *v* (απόδειξη με επαγωγή στο μήκος του μονοπατιού)
- **Επικαλύπτον Δένδρο ή Δάσος (απλή συνεκτικότητα):**

  Εάν το γράφημα είναι συνεκτικό, τότε το δένδρο ΑκΠ είναι επικαλύπτον.

  Διαφορετικά, κάθε επανεκκίνηση της ΑκΠ ανακαλύπτει και μία συνεκτική συνιστώσα (ελάχιστες οι τροποποιήσεις των αλγορίθμων- μόνο ένας πίνακας και μία βοηθητική μεταβλητή)
- **Συντομότερα, σε # ακμών, μονοπάτια από μία κορυφή-πηγή:**

  Εκκίνηση ΑκΠ από την *v*

Γενίκευση

Algorithm pqSearchList(graph G, vertex v)

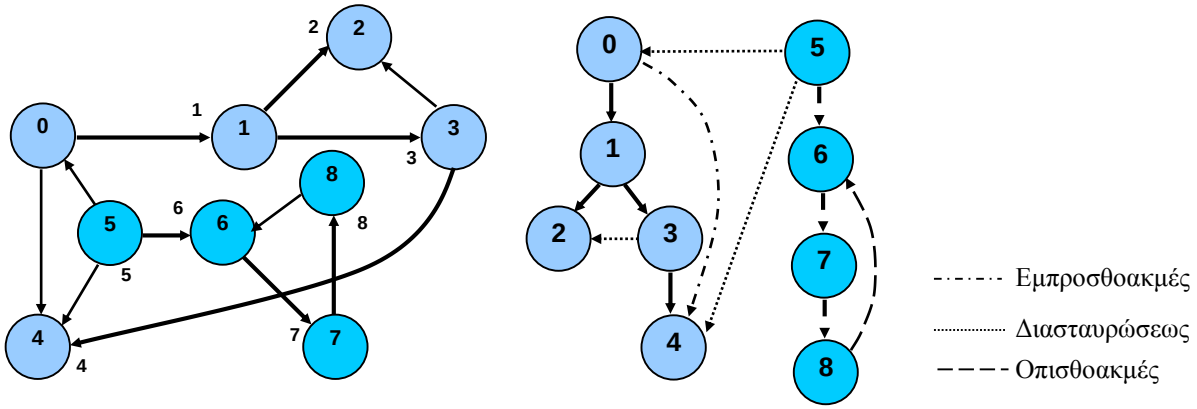
```
1.  pqueue.enqueue(v);
2.  G.pre[v] = order++;
3.  while (!pqueue.isEmpty()){
4.    w = pqueue.dequeue();
5.    G.T[w] = v;
6.    for (x = G.List[w]; x != null; x = x.getNext())
7.      if (G.pre[x.v] == -1){
8.         pqueue.enqueue(x);
9.         G.pre[x.v] = order++;
10.   }
11. }
```

- Το πώς γίνεται η επιλογή της επόμενης κορυφής καθορίζει και το είδος του ψαξίματος...

Κατευθυνόμενα Γραφήματα

- Αποτελούν πλούσια σε ιδιότητες συνδυαστικά αντικείμενα:
  - Διαπεράσεις
  - Άκυκλα κατευθυνόμενα γραφήματα
  - Αλγόριθμους για ζητήματα διασυνδέσεως

AσB



- Είδη Ακμών:
  - Ακμές Δένδρου
  - Εμπροστοακμές
  - Οπισθοακμές
  - Ακμές Διασταυρώσεως (πάντοτε από δεξιά προς αριστερά-γιατί;)

Εφαρμογές ΑσΒ

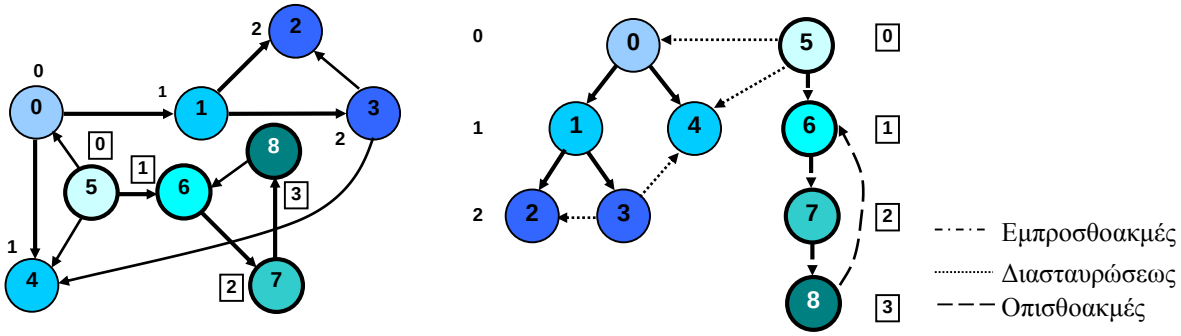
- **Προσπελάσιμες από κορυφή v κορυφές του γραφήματος:**

  Με επαγωγή στο μέγεθος του γραφήματος
- **Εντοπισμός κύκλου:**

  Μόλις βρεθεί η οπισθοακμή
- **Ισχυρά συνεκτικότητα – Μεταβατική κλειστότητα:**

  Θα δούμε πώς...

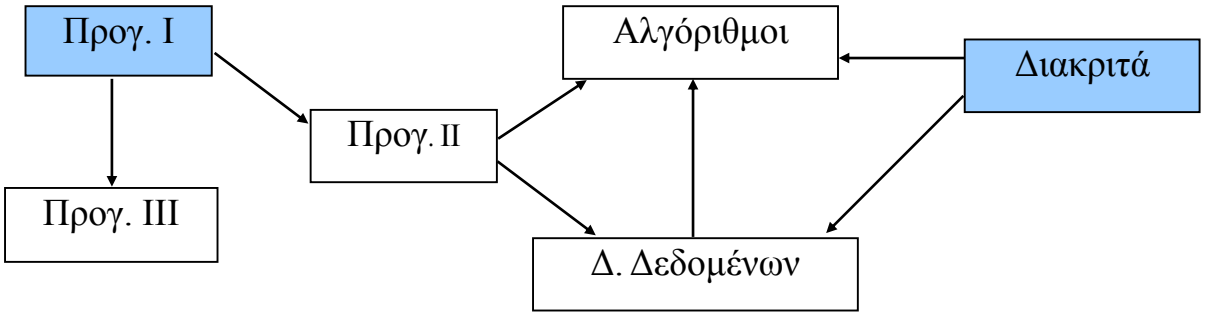
ΑκΠ



- Είδη Ακμών
  - Ακμές Δένδρου
  - Ακμές Διασταυρώσεως
  - Οπισθοακμές
  - ΔΕΝ ΥΠΑΡΧΟΥΝ ΕΜΠΡΟΣΤΟΑΚΜΕΣ

ΚΑΓ

- Ιδιαίτερη κατηγορία γραφημάτων, τα οποία δύνανται να θεωρηθούν και ένα είδος δένδρου.
- Εφαρμογές
  - Προγραμματισμός εργασιών έργου
  - Περιορισμοί στο πρόγραμμα σπουδών



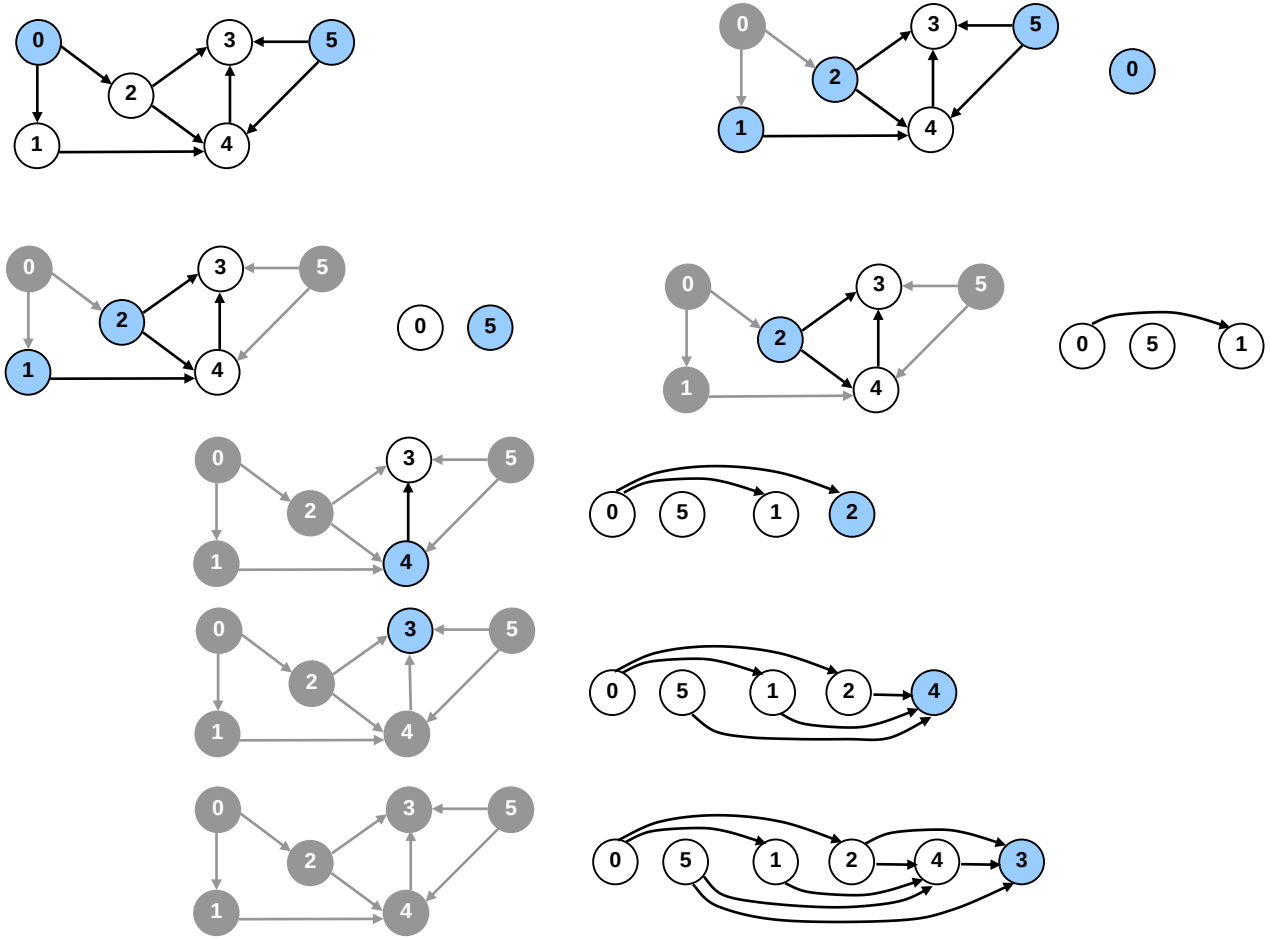
Τοπολογική Διάταξη

- Επιβολή διατάξεως στις κορυφές του γραφήματος, ώστε μία κορυφή *v* να προηγείται όλων των κορυφών *w*, με $(v,w) \in E$
- Πχ, στο ΚΑΓ- διάγραμμα έργων, η εκτέλεση των υποεργασιών κατά τοπολογική διάταξη εγγυάται την ορθή εκτέλεση του έργου

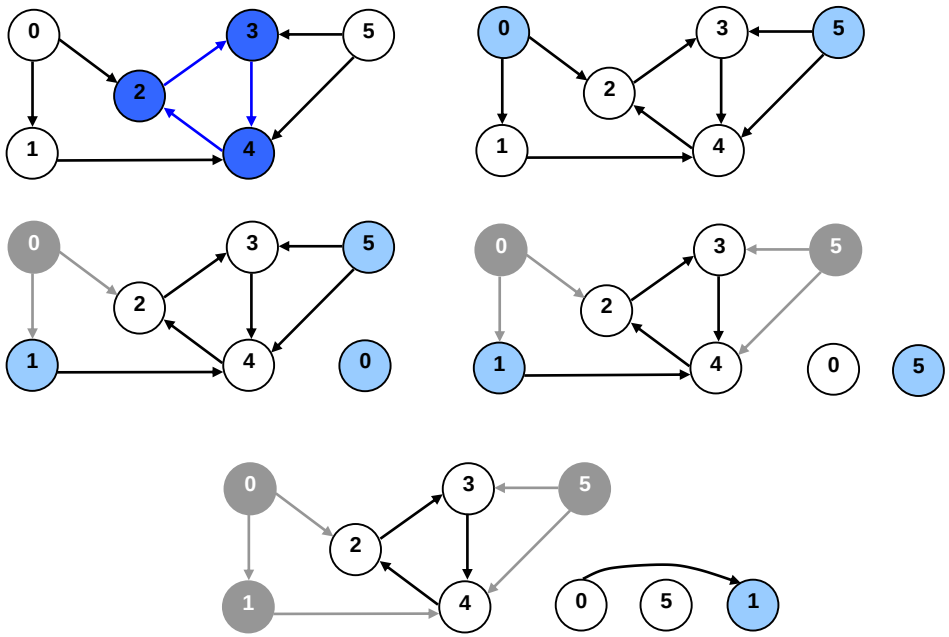
1η Λύση

- Συστηματική αφαίρεση πηγών
- Εντοπίζει, μάλιστα, και όσα κατευθυνόμενα γραφήματα δεν είναι άκυκλα και, επομένως, λανθασμένα δόθηκαν ως είσοδος στον αλγόριθμο

Στιγμιότυπο



Στιγμιότυπο με Κύκλο



Αλγόριθμος

Algorithm topolSortList(graph g, **int**[] sort)

```
1.  int[] aux;
2.  for (v = 0; v < G.V; v++){                  // αρχικοποίηση
3.     aux[v] = 0;
4.     sort[v] = -1;
5.  }
6.  for (v = 0; v < g.V; v++)                   // υπολογισμός βαθμού εισόδου κορυφών
7.    for (x = G.List[v]; x != null; x = x.getNext())
8.      aux[x.v]++;
9.   FiFo pq = new FiFo();
10. for (v = 0; v < g.V; v++)
11.    if (aux[v] == 0) pq.enqueue(v);     // έγινε πηγή
12. for (i = 0; !pq.isEmpty(); i++){
13.    order[i] = (v = pq.dequeue());       // αφαίρεση της επόμενης πηγής
14.       for (x = g.List[v]; x != null; x = x.getNext())
15.          if (--aux[x.v] == 0)              // ενημέρωση των βαθμών εισόδου των γειτονικών κορυφών
16.             pq.enqueue(x.v);
17. }
```

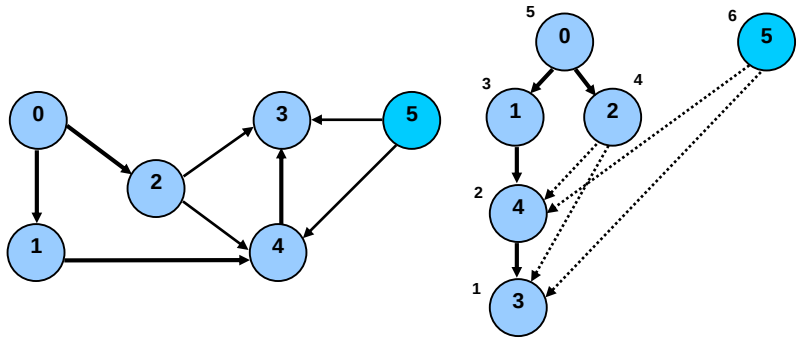
2η Λύση

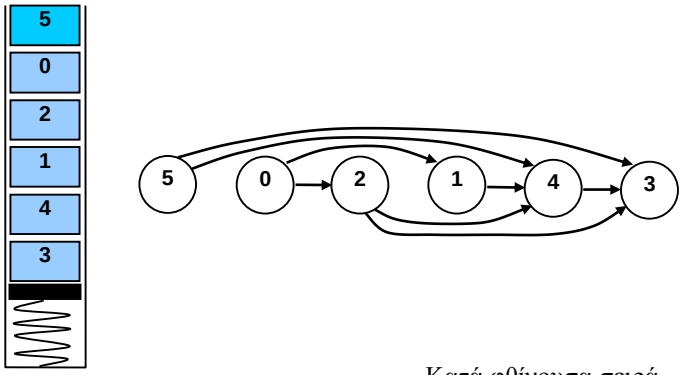
- Η μεταδιάταξη αποτελεί μία αντίστροφη τοπολογική διάταξη σε ένα ΚΑΓ:

  μία κορυφή περατώνει πιο γρήγορα όσο περισσότερο εξαρτημένη είναι

- Οπότε:
  - είτε χρήση στοίβας για την αντιστροφή της
  - είτε διενέργεια ΑσΒ στο ανάστροφο ΚΑΓ

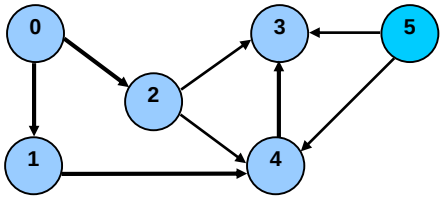
Χρήση Στοίβας

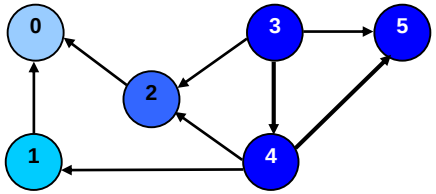


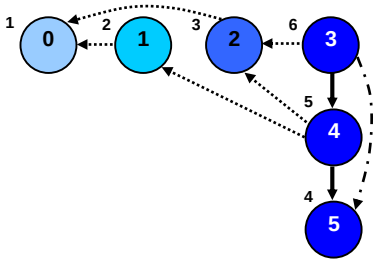


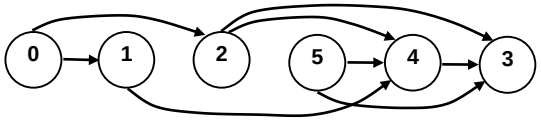
Κατά φθίνουσα σειρά περάτωσης (postorder) (α)

Χρήση Ανάστροφου Γραφήματος









Κατά αύξουσα σειρά περάτωσης (postorder) στο ανάστροφο (β)

Μεταβατική Κλειστότητα

- Απλούστατη Λύση:
  - V ΑσΒ, μία από κάθε κορυφή v, ώστε να ανακαλυφθούν όλες οι προσπελάσιμες, από την v, κορυφές
  - Κόστος $O(V(E+V))$
    - Καλό για αραιά γραφήματα
    - Όμως, $O(V^3)$ για πυκνά γραφήματα

Μεταβατική Κλειστότητα για Πυκνά Γραφήματα

- Παρατήρηση:

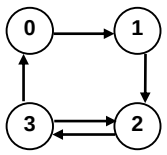
  Εάν A είναι ο πίνακας γειτνιάσεως και σχηματίσουμε το $A \times A = A^2$, ερμηνεύοντας τα $\{\cdot, +\}$ ως $\{\wedge, \vee\}$, αντίστοιχα, τότε ανακαλύπτουμε όλα τα μονοπάτια μήκους ακριβώς 2, καθώς:

  $A^2[i][j]=1$ ανν $\{\exists \geq 1\ k \neq i,j: A[i][k] \cdot A[k][j]=1\}$,

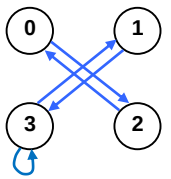
  $$\left(A^2[i][j] = \bigvee_k (A[i][k] \wedge A[k][j])\right)$$

  ενώ τα μήκους ένα (απευθείας συνδέσεις) διαγράφονται.

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



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

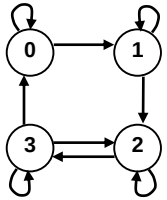


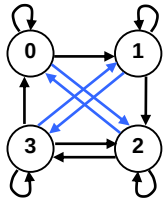
- $A^3=A^2A$, δίδει τα μονοπάτια μήκους ακριβώς 3
- $A^4=A^3A$, δίδει τα μονοπάτια μήκους ακριβώς 4, κ.ο.κ.

Μεταβατική Κλειστότητα (συν.)

- Προσθέτουμε 1 στις διαγωνίους (δηλ., βρόχους στο γράφημα)
- Τότε κάθε δύναμη A^i, $i \le V$, μάς δίδει όλα τα μονοπάτια με μήκος ***το πολύ*** i (οι βρόχοι που προσθέσαμε, μάς επιτρέπουν να κρατούμε τα '1' του A^{i-1})
- Μεγαλύτερες δυνάμεις δεν παρέχουν καμμία επιπρόσθετη πληροφορία

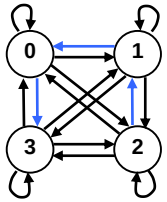
$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

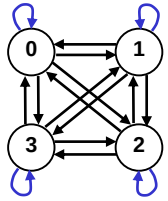




$$A^2 = AA = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$A^3 = A^2A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$





$$A^4 = A^3A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

- Λύση κόστους $O(V^3V)=O(V^4)$
- Μειώνεται σε $O(V^3 \log V)$ με διαδοχικούς τετραγωνισμούς:

$$A^2,\ A^4=A^2A^2,\ A^8=A^4A^4,\ \ldots$$

Ακόμη καλύτερη λύση: Αλγόριθμος Warshall

- Δίδεται τυχαία διάταξη στις κορυφές
- Σχηματίζεται μία ακολουθία γραφημάτων $G_1, G_2, \ldots, G_V$, ώστε:

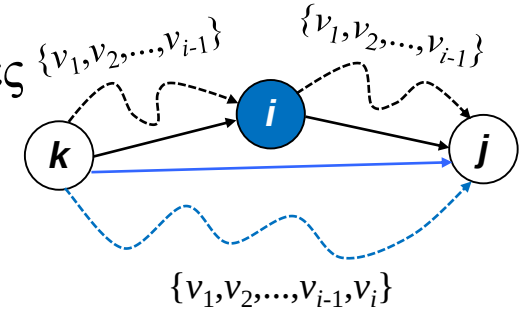
  $$G_i = G_{i-1} \cup \{(v_k, v_j) \mid (v_k, v_i) \wedge (v_i, v_j) \in G_{i-1}\}$$

- $G^* = G_V$. Με επαγωγή στο i:

  Έστω ότι το G_{i-1} έχει συνδέσεις για όλα τα μονοπάτια $v_k \in v_j$, αποτελούμενα *αποκλειστικά* από ενδιάμεσες κορυφές $v_1, v_2, \ldots, v_{i-1}$.

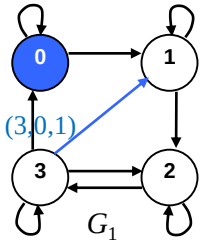
  Για $i = 1$, είναι αληθές.

  Τότε, εκ κατασκευής, το G_i έχει συνδέσεις για όλα τα μονοπάτια $v_k \in v_j$, αποτελούμενα αποκλειστικά από τις ενδιάμεσες κορυφές $v_1, v_2, \ldots, v_i$, καθώς παίρνει όλα του G_{i-1} συν τα καινούρια με ενδιάμεση κορυφή την v_i



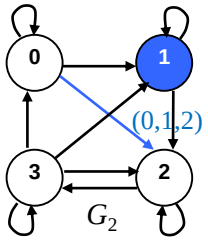
Παράδειγμα

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



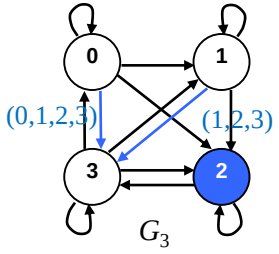
G_1

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



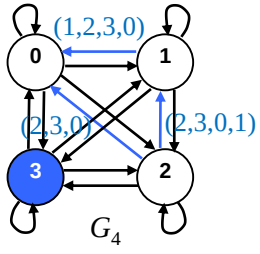
G_2

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



G_3

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



G_4

Algorithm warshallTC(graph g)

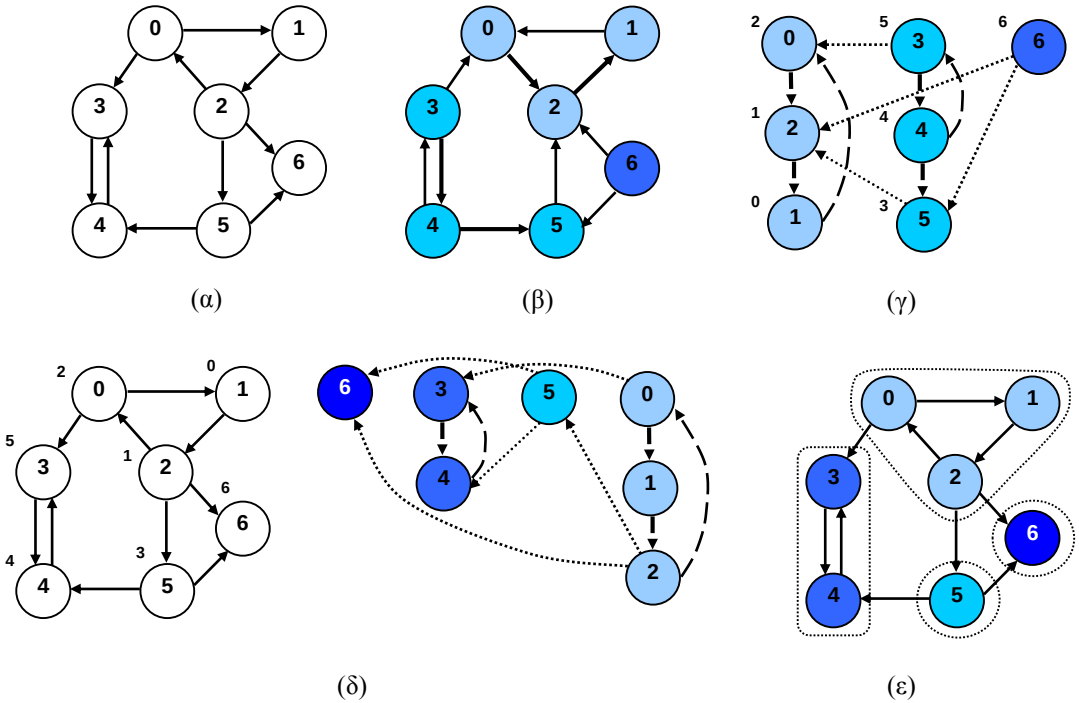
```
1.   for (v = 0; v < g.V; v++)
2.     for (w = 0; w < g.V; w++)
3.       g.TrClosure[v][w] = g.A[v][w];
4.   for (v = 0; v < g.V; v++)
5.       g.TrClosure[v][v] = 1;
6.    for (i = 0; i < g.V; i++)  // i-στός γύρος
7.      for (v = 0; v < g.V; v++)
8.        if (g.TrClosure[v][i] == 1)  // έχει νόημα η δοκιμή
9.          for (w = 0; w < g.V; w++)
10.           if (g. TrClosure[i][w] == 1)
11.             g.tcTrClosure [v][w] = 1;
```

- Χρόνος $O(V^3)$

Ισχυρά Συνεκτικές Συνιστώσες

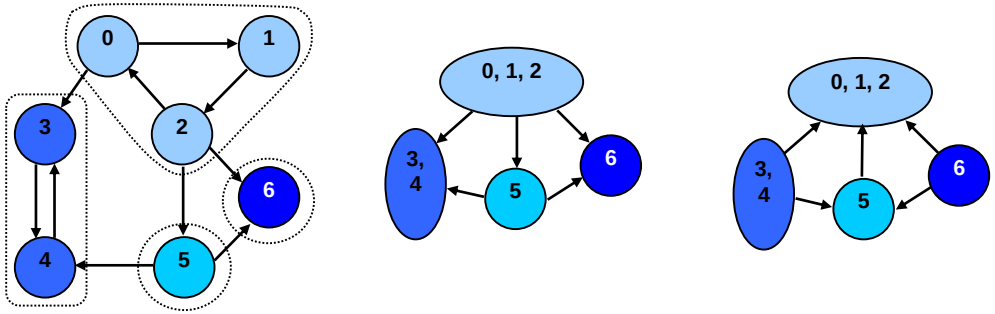
- **Λύση Kosaraju:** Πολύ απλή στην περιγραφή και υλοποίηση!
  - Μια AσB στο G^R
  - Βάσει της φθίνουσας προκύπτουσας postorder διάταξης, μια δεύτερη AσB στο G

Παράδειγμα



Διαίσθηση...

- Φανταστείτε ότι οι συνιστώσες ήταν γνωστές.
- Συμπίεση τους σε υπερκόμβους και αναστροφή του ΚΑΓ που προκύπτει.
- Η ΑσΒ τις διαπερνά, από τις περισσότερο προς λιγότερο εξαρτημένες, δίχως να τις εγκαταλείψει....



(α) (β) (γ)

Τυπικότερη Απόδειξη

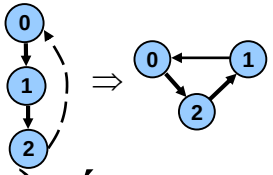
- Δύο κορυφές v,w ανήκουν στην ίδια ισχυρά συνιστώσα ανν ανήκουν στο ίδιο δένδρο AσB

($\Rightarrow$) Προφανές, από την λειτουργία της AσB

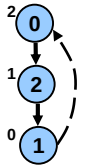
($\Leftarrow$) Θ.δ.ο. $v \varnothing \rho \varnothing w$ (ρ ρίζα δένδρου AσB στο G)

$v \varnothing \rho$, καθώς:

$\rho \rightarrow v$ στο $G \Rightarrow v \rightarrow \rho$ στο G^R



$\Rightarrow v \leftarrow \rho$ στο G^R καθώς post(ρ) > post(v), άρα δεν είναι δυνατόν να ανήκουν σε διαφορετικά υποδένδρα



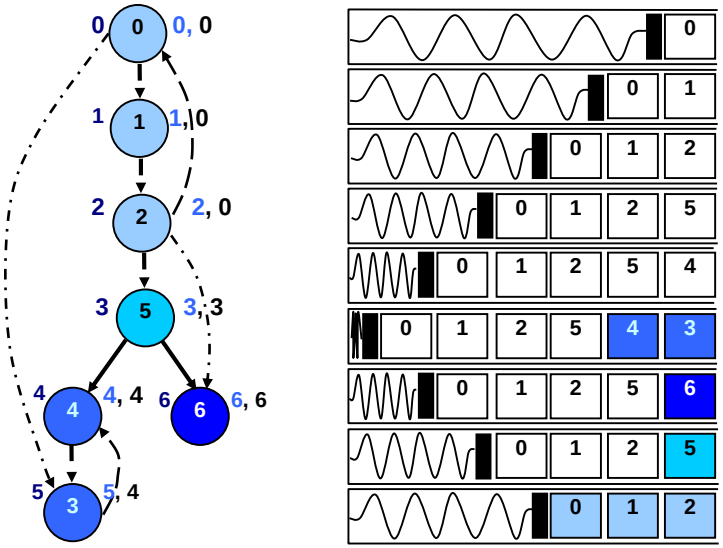
$\Rightarrow v \rightarrow \rho$ (στο G)

$\rho \varnothing w$: αποδεικνύεται παρομοίως

Λύση Tarjan

- Μειονεκτήματα Kosaraju:
  - Υπολογισμός του G^R
  - 2 ΑσΒ
- Πρόταση Tarjan
  - 1 ΑσΒ
  - Βοηθητική Stack
  - Βοηθητικός Πίνακας Low (όπως στον υπολογισμό των Δισυνεκτικών Συνιστωσών)

Παράδειγμα


0
0, 0
1
1, 0
2
2, 0
3
3, 3
4
4, 4
6
6, 6
5
5, 4
0
0 1
0 1 2
0 1 2 5
0 1 2 5 4
0 1 2 5 4 3
0 1 2 5 6
0 1 2 5
0 1 2

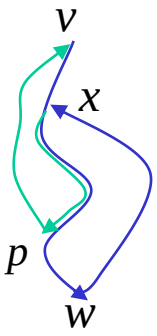
Algorithm tarjanSC(graph g, **int** w)

```
1.  g.pre[w] = g.order++;
2.  g.low[w] = g.pre[w];
3.  min = low[w];
4.  g.stack.push(w);
5.  for (x = g.List[w]; x != null; x = x.getNext()){
6.    if (pre[x.v] == -1)
7.       tarjanSC(g, x.v);
8.    if (g.low[x.v] < min)   // οπισθοακμή
9.       min = g.low[x.v];
10. }
11. if (min < g.low[w]){     // δεν είναι αρχή συνιστώσας
12.   g.low[w] = min;        // ενημέρωση και
13.   return;                // επιστροφή
14. }
15. do {                     // αρχή συνιστώσας
16.    g.sc[(v = g.stack.pop()] = g.strcompnum;
17.    g.low[v] = g.V;       // γείωσή τους ώστε να μην επηρεάζουν τους υπολογισμούς!
18. }
19. while (v != w);
20. g.strcompnum++;       // νέος αριθμός συνιστώσας
```

- Χρόνος $O(V+E)$ αφού αποτελεί επέκταση της ΑσΒ!!!

Ορθότητα (σχεδιάγραμμα)

- Εάν για κάθε απόγονο w της v δεν υπάρχει οπισθοακμή που να ξεπερνά την v, τότε μπορεί κανείς να σχηματίσει κατευθυνόμενο κύκλο που να περιλαμβάνει την w και την v:

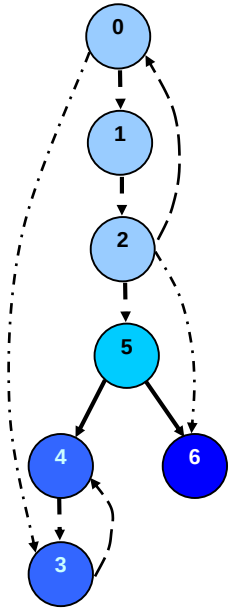


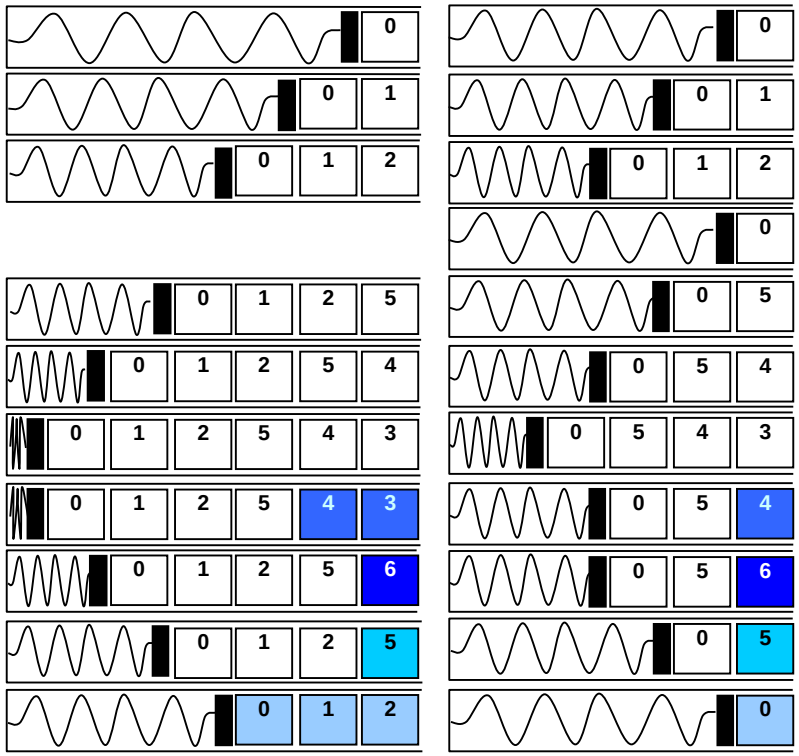
  - $v \in w$ (με ακμές δένδρου) $\in x$ (απόγονο της w με οπισθοακμή προς y που ξεπερνά την w) $\in y$ $\in z$ (απόγονο της y με οπισθοακμή προς q που ξεπερνά την y) $\in q \in \cdots \in p \in v$

Λύση Gabow

- 1 AσB και 2 stack, ώστε να αποφεύγεται η χρήση του low:
  - 1η stack για την καταχώρηση των κορυφών κατά σειρά ανακαλύψεως
  - 2η stack για εξομοίωση του low

Παράδειγμα


0
1
2
5
4
6
3


0
0 1
0 1 2
0 1 2 5
0 1 2 5 4
0 1 2 5 4 3
0 1 2 5 4 3
0 1 2 5 6
0 1 2 5
0 1 2
0
0 1
0 1 2
0
0 5
0 5 4
0 5 4 3
0 5 4
0 5 6
0 5
0

Algorithm gabowSC(graph g, vertex w)

```
1.  g.pre[w] = g.order++;
2.  g.stack.push(w);
3.  g.pstack.push(w);
4.  for (x = g.List[w]; x != null; x = x.getNext())
5.      if (g.pre[x.v] == -1)           // ανεξερεύνητος απόγονος
6.        gabowSC(g, x.v);
7.      else if (g.sc[x.v] == -1)       // οπισθοακμή προς x.v εντός ανεξερεύνητης συνιστώσας
8.        while (g.pre[g.pstack.top()] > g.pre[x.v])          // ποπ ώστε να μείνει η ψηλότερη
9.            g.pstack.pop();                                 // από αυτές που είδαμε
10. if (g.pstack.top() != w)          // δεν αποτελεί σημείο εισόδου
11.    return;
12. else
13.    g.pstack.pop();                // αποτελεί κορυφή εισόδου που
14. do                                // «φράσσει» τα περιεχόμενα της πρώτης stack
15.    g.sc[(t=g.stack.pop())] = g.strcompnum;
16. while (t != w);
17. g.strcompnum++; // νέος επόμενος αριθμός συνιστώσας
```

- Χρόνος $O(V+E)$