ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

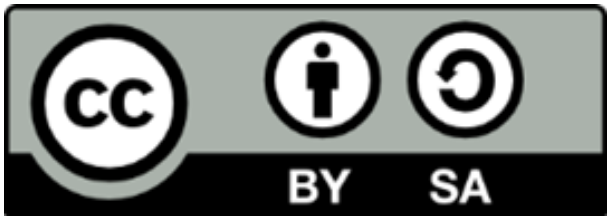
Ενότητα 10β: Αλγόριθμοι Γραφημάτων-Γραφήματα-Αναπαράσταση Γραφημάτων- Διερεύνηση Πρώτα σε Πλάτος (BFS)

Μαρία Σατρατζέμη
Τμήμα Εφαρμοσμένης Πληροφορικής

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
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 Μακεδονίας» 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.

Γραφήματα

- **Ορισμός**: Ένα γράφημα G είναι το διατεταγμένο ζεύγος (V, E), όπου
  - $V = \{v_1, v_2, \ldots, v_n\}$ σύνολο κορυφών
  - E = σύνολο ακμών = υποσύνολο του $V \times V$
  - Άρα $|E| = O(|V|^2)$

Τύποι γραφημάτων

- Σε ένα *μη προσανατολισμένο* (*ακατεύθυντο*) γράφημα (*undirected graph):*
  - ακμή (u,v) = ακμή (v,u)
  - Χωρίς βρόχους (self-loops) -ακμή που αρχίζει & τερματίζει στον εαυτό της
- Σε ένα *προσανατολισμένο* (*κατευθυντό*) (*directed graph*):
  - Η ακμή (u,v) κατευθύνεται από την κορυφή u προς την κορυφή v και συμβολίζεται: u→v
- Ένα *συνεκτικό* (*connected graph)* έχει ένα μονοπάτι από κάθε κορυφή του προς κάθε κορυφή του
  - *Μονοπάτι* αλληλουχία κορυφών και ακμών χωρίς επανάληψη κορυφής

Άλλοι Τύποι γραφημάτων

- Ένα *γράφημα με βάρος (εμβαρές) weighted graph* αντιστοιχεί βάρη είτε στις ακμές είτε στις κορυφές
  - π.χ., ένας οδικός χάρτης: οι ακμές μπορεί να έχουν βάρη που αναπαριστούν την απόσταση
- Ένα *multigraph* επιτρέπει πολλαπλές ακμές μεταξύ των ίδιων κορυφών
  - π.χ., το γράφημα κλήσεων συναρτήσεων σε ένα πρόγραμμα (μια συνάρτηση μπορεί να κληθεί από πολλά σημεία από μια άλλη συνάρτηση)

Πυκνά – Αραιά Γραφήματα

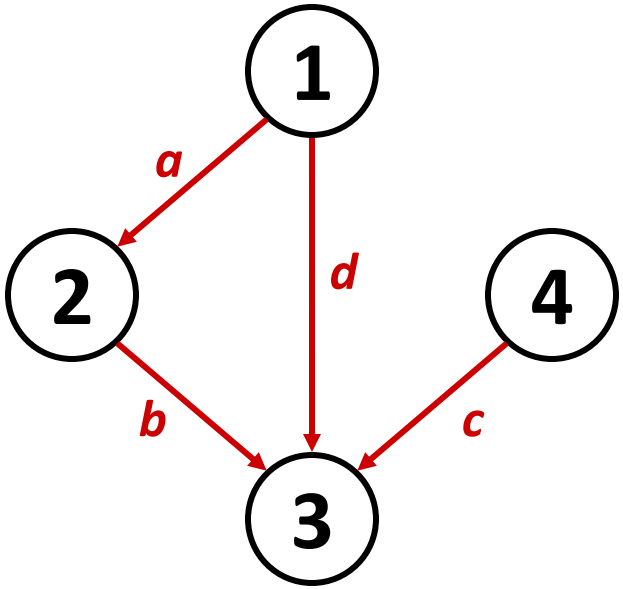
- Συνήθως εκφράζουμε τους χρόνους εκτέλεσης σε σχέση με τους όρους των $|E|$ και $|V|$
  - Αν $|E| \approx |V|^2$ το γράφημα είναι *πυκνό* (*dense)*
  - Αν $|E| \approx |V|$ το γράφημα είναι *αραιό sparse*
- Αν είναι γνωστό ότι το γράφημα είναι αραιό ή πυκνό τότε ενδεχόμενα χρησιμοποιείται διαφορετική δομή δεδομένων

Αναπαράσταση Γραφημάτων

- Υποθέτουμε ότι V = {1, 2, …, *n*}
- Ένας *πίνακας γειτνίασης* (*adjacency matrix)* αναπαριστά το γράφημα ως έναν *n* x *n* πίνακα A:
  - A[*i*, *j*] = 1 αν η ακμή (*i*, *j*) $\in$ E (ή το βάρος της ακμής)
    = 0 αν η ακμή (*i*, *j*) $\notin$ E

Γραφήματα: Πίνακας γειτνίασης

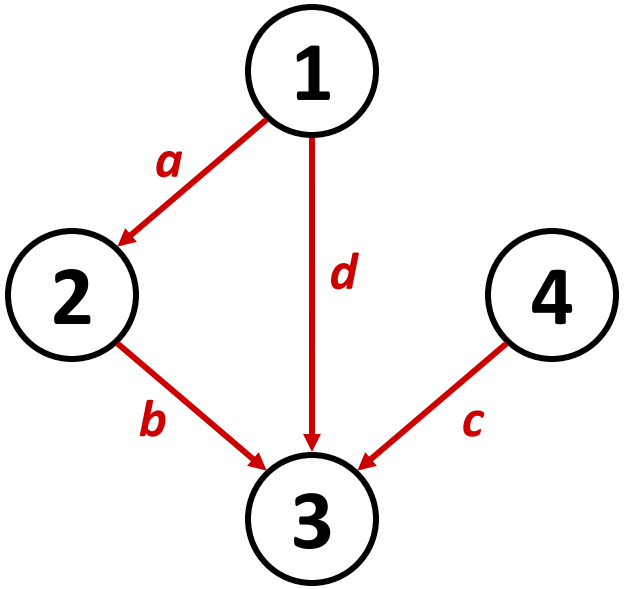
- Παράδειγμα:



A	1	2	3	4
1				
2				
3			??	
4				

Γραφήματα: Πίνακας γειτνίασης

- Παράδειγμα :



A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

Γραφήματα: Πίνακας γειτνίασης

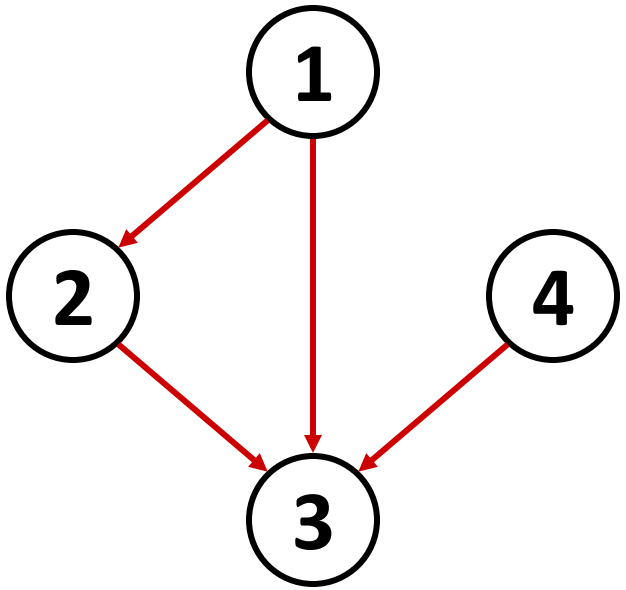
- *Πόσο χώρο απαιτεί ο πίνακας γειτνίασης?*
- Α: $O(V^2)$
  - Μη προσανατολισμένο γράφημα $\rightarrow$ πίνακας συμμετρικός

Γραφήματα: Πίνακας γειτνίασης

- Ο πίνακας γειτνιάσεως
  - Απαιτεί μεγάλο χώρο αποθήκευσης για μεγάλα γραφήματα
  - Αλλά είναι πολύ αποτελεσματικός για μικρά γραφήματα
- Τα περισσότερα μεγάλα ενδιαφέροντα γραφήματα είναι αραιά
  - π.χ., επίπεδα γραφήματα (planar graphs), στα οποία οι ακμές δε διασταυρώνονται, έχουν $|E| = O(|V|)$ με βάση τον τύπο του Euler
  - η *λίστα γειτνίασης (ή κατάλογος γειτνίασης)* (*adjacency list)* είναι συνήθως πιο κατάλληλη για την αναπαράσταση ενός γραφήματος

Γραφήματα: Λίστα γειτνίασης

- Λίστα γειτνίασης: για κάθε κορυφή $v \in V$, αποθηκεύει μια λίστα από κορυφές γειτονικές της κορυφής v
- Παράδειγμα:
  - Adj[1] = {2,3}
  - Adj[2] = {3}
  - Adj[3] = {}
  - Adj[4] = {3}



- Παραλλαγή: μπορεί να διατηρεί μια λίστα από ακμές που συντρέχουν προς την κορυφή

Γραφήματα: Λίστα γειτνίασης

- Πόσος χώρος απαιτείται?

  # : πλήθος

  - Ο ***βαθμός*** (*degree)* μιας κορυφής *v* = # συντρέχουσες ακμές
    - τα προσανατολισμένα γραφήματα έχουν **έσω-βαθμό** (*in-degree*), **έξω-βαθμό** (*out-degree*)
  - Για τα προσανατολισμένα γραφήματα, # των στοιχείων στις λίστες γειτνίασης είναι

    $$\Sigma \text{ out-degree}(v) = |E|$$

    απαιτεί $\Theta(V + E)$ χώρο
  - Για μη-προσανατολισμένα, # των στοιχείων των λιστών γειτνίασης είναι Σ degree(v) = 2 $|E|$ επίσης απαιτεί $\Theta(V + E)$ χώρο μνήμης
- Άρα: οι λίστες γειτνίασης απαιτούν $\Theta(V+E)$ χώρο

Γραφήματα: Λίστα γειτνίασης

- Για προσανατολισμένο γράφημα το άθροισμα των μεγεθών των λιστών γειτνίασης είναι $|E|$ αφού οποιαδήποτε ακμή (u,v) αναπαριστάται μέσω της παρουσίας της u στη λίστα Adj[u]
- Αν το G είναι μη προσανατολισμένο το αντίστοιχο άθροισμα είναι $2\ |E|$, αφού αν η ακμή (u,v) είναι μια μη προσανατολισμένη ακμή, τότε u θα εμφανίζεται στη λίστα γειτνίασης της v και αντιστρόφως
- Άρα: οι λίστες γειτνίασης απαιτούν $\Theta(V+E)$ χώρο μνήμης για τα προσανατολισμένα και μη προσανατολισμένα γραφήματα

Διερεύνηση Γραφήματος

- Δίνεται: ένα γράφημα G = (V, E), προσανατολισμένο ή μη-προσανατολισμένο
- Σκοπός: μεθοδική διερεύνηση κάθε κορυφής και κάθε ακμής
- Αποτέλεσμα: δημιουργία ενός δένδρου για το γράφημα
  - Επιλέξτε μια κορυφή ως ρίζα
  - Επιλέξτε συγκεκριμένες ακμές για να δημιουργήσετε το δένδρο
  - Σημείωση: ενδεχόμενα να δημιουργηθεί ***δάσος*** (*forest)* αν το γράφημα δεν είναι συνεκτικό

Breadth-First Search

- “Διερεύνηση” ενός γραφήματος, μετατροπή του σε δένδρο
  - Μια κορυφή τη φορά
  - Επεκτείνετε το «σύνορο» των εξερευνημένων κορυφών σε όλο το εύρος του συνόρου **ομοιόμορφα-οριζόντια ή κατά πλάτος** (*Breadth-First)*
- Κατασκευή του δένδρου από το γράφημα
  - Επιλέξετε μια κορυφή **αφετηρία** (*source)* για ρίζα
  - Ανακαλύψτε (“discover”) τα παιδιά της, στη συνέχεια τα παιδιά τους, κτλ.

Αλγόριθμος BFS

- Ο αλγόριθμος «χρωματίζει» κάθε κορυφή για να καταγράψει την πορεία της εργασίας
  - Λευκές κορυφές δεν έχουν ακόμη ανακαλυφθεί
    - Όλες οι κορυφές ξεκινούν λευκές
  - Γκρι κορυφές έχουν ανακαλυφθεί αλλά δεν έχουν εξερευνηθεί πλήρως
    - Ενδεχόμενα να έχουν γειτονικές λευκές κορυφές
  - Μαύρες κορυφές έχουν ανακαλυφθεί και πλήρως εξερευνηθεί
    - Είναι γειτονικές μόνο σε μαύρες και γκρι κορυφές
- Εξερευνήστε τις κορυφές «σαρώνοντας» τη λίστα γειτνίασης των γκρι κορυφών

Breadth-First Search: Algorithm

Αλγόριθμος BFS(G, s)

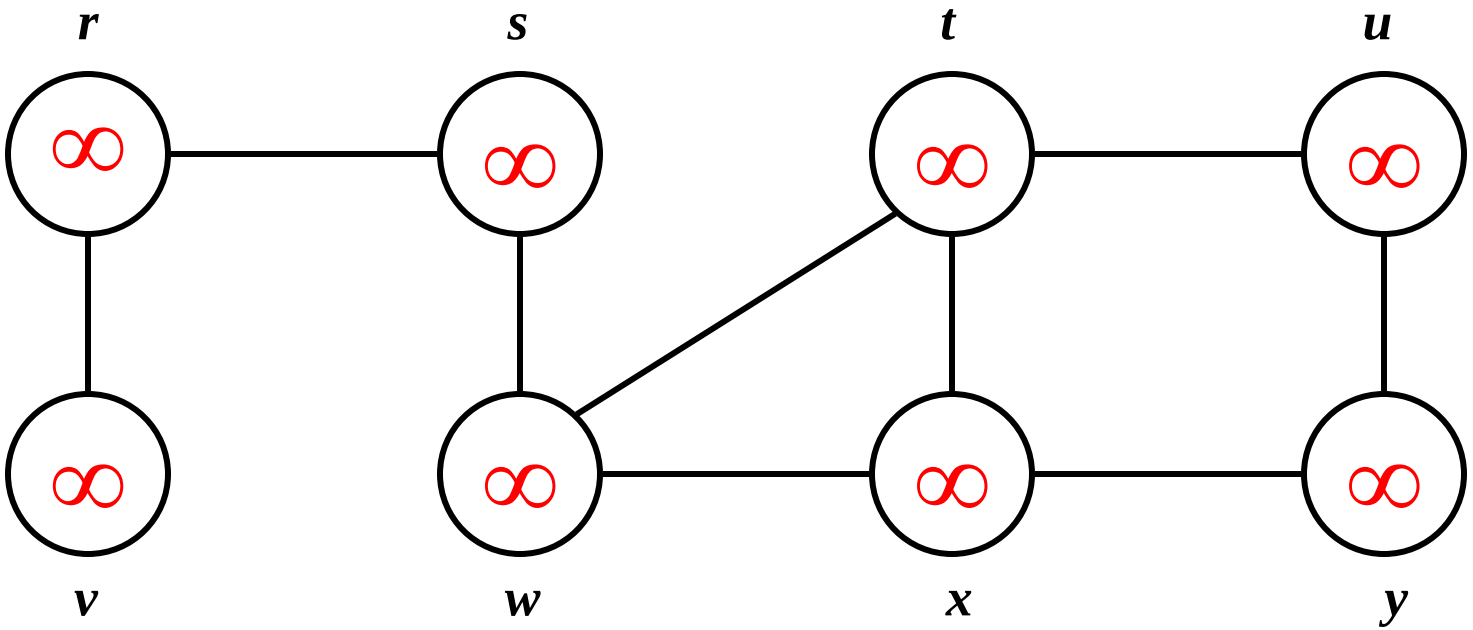
```
1.    for each v ∈ V – {s}
2.        color[v] = WHITE;
3.        d[v] = ∞;
4.        p[v] = null;
5.    color[s] = GREY;
6.    d[s] = 0;
7.    p[s] = null;
8.    Q = ∅
9.    Enqueue(Q, s);          // Q είναι ουρά; Αρχικοποιείται με την κορυφή s
10.   while (Q not empty)
11.       u = RemoveTop(Q);
12.       for each v ∈ Adj[u]
13.           if (color[v] == WHITE)
14.               color[v] = GREY;
15.               d[v] = d[u] + 1;
16.               p[v] = u;
17.               Enqueue(Q, v);
18.       color[u] = BLACK;
```

Στη μεταβλητή color[u] καταχωρίζεται το χρώμα κάθε κορυφής u

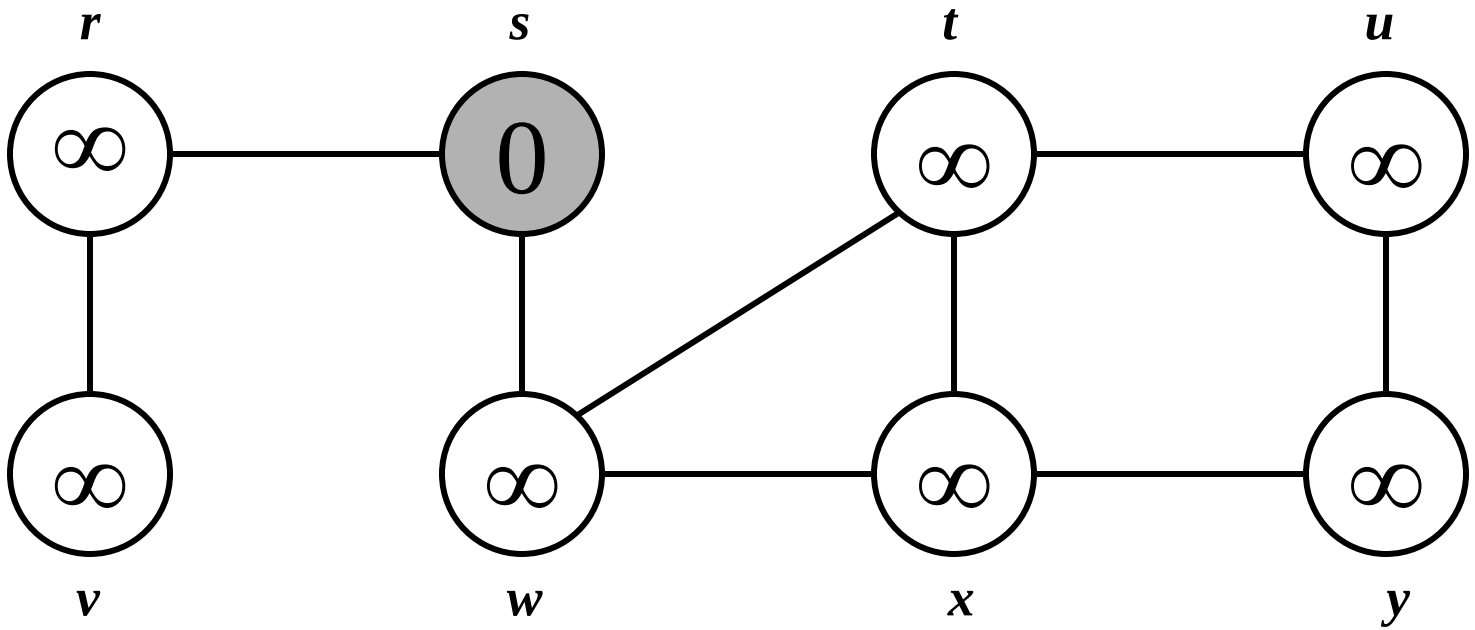
Στη μεταβλητή d[u] καταχωρείται η απόσταση από την κορυφή αφετηρία s μέχρι την u την οποία υπολογίζει ο αλγόριθμος

Στη μεταβλητή p[u] καταχωρίζεται ο προκάτοχος της u

Breadth-First Search: Example

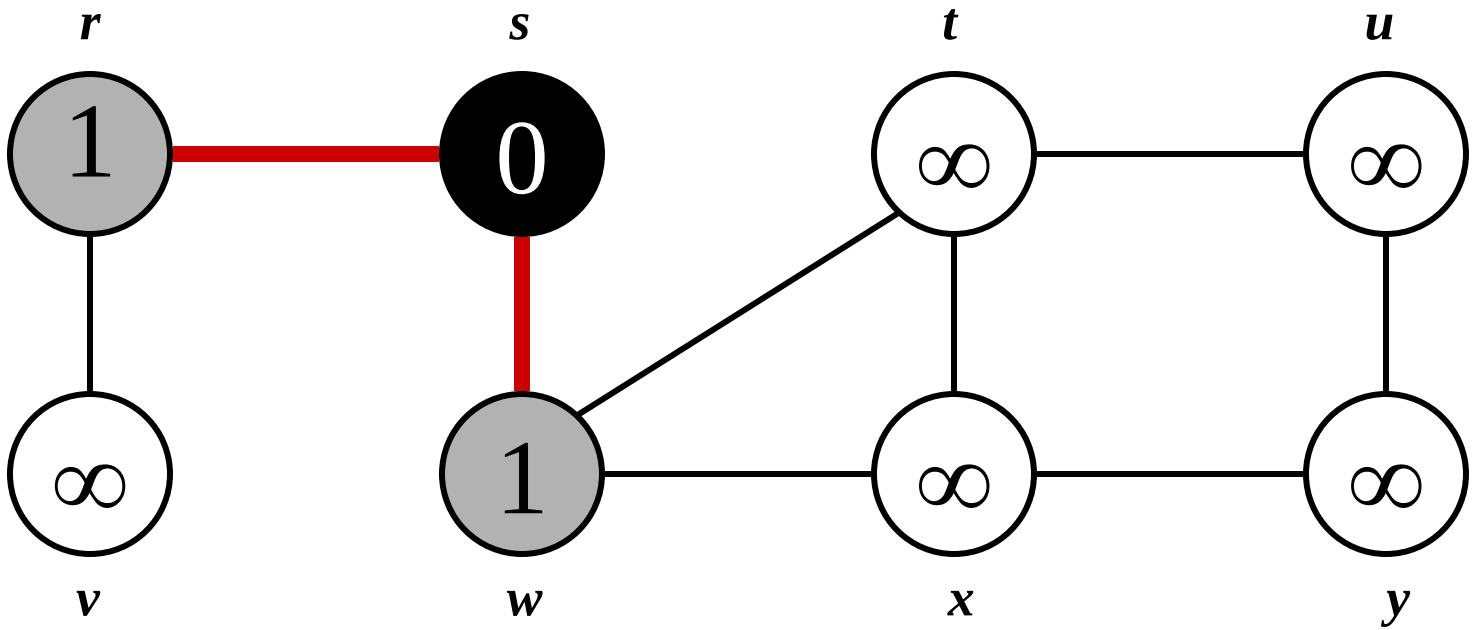


Breadth-First Search: Example



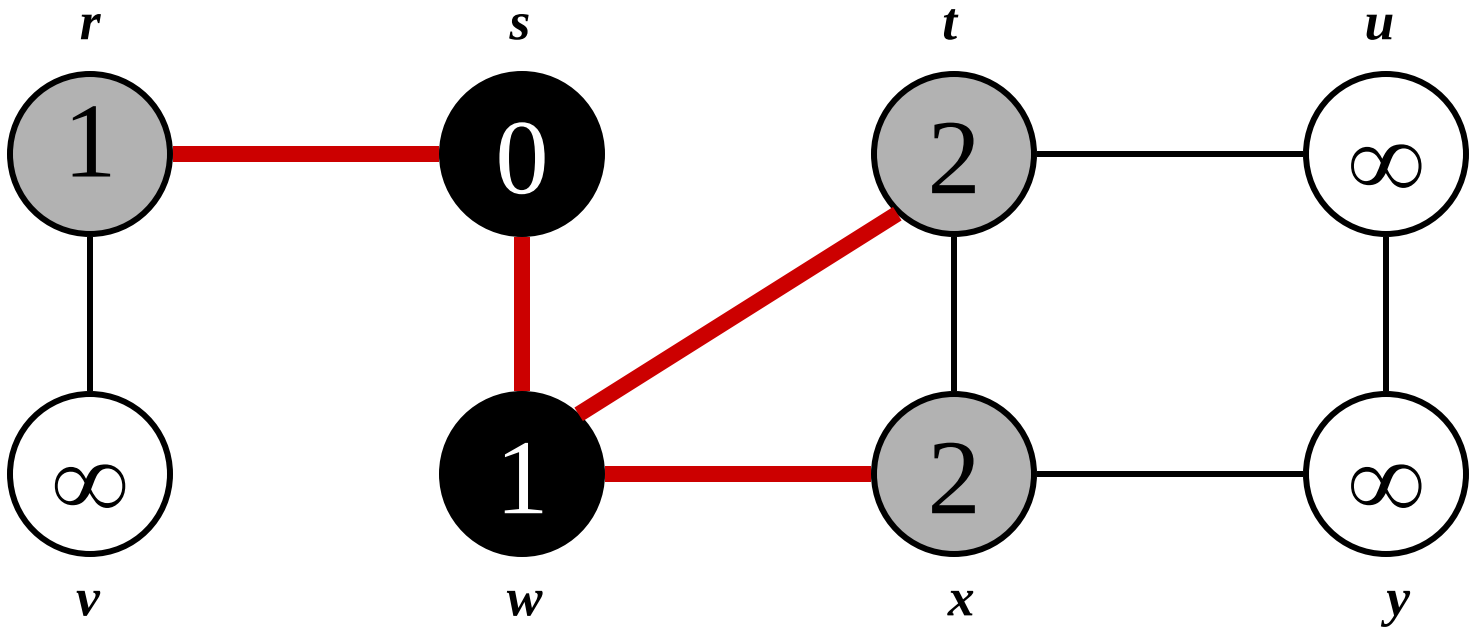
$Q:$ | s |

Breadth-First Search: Example



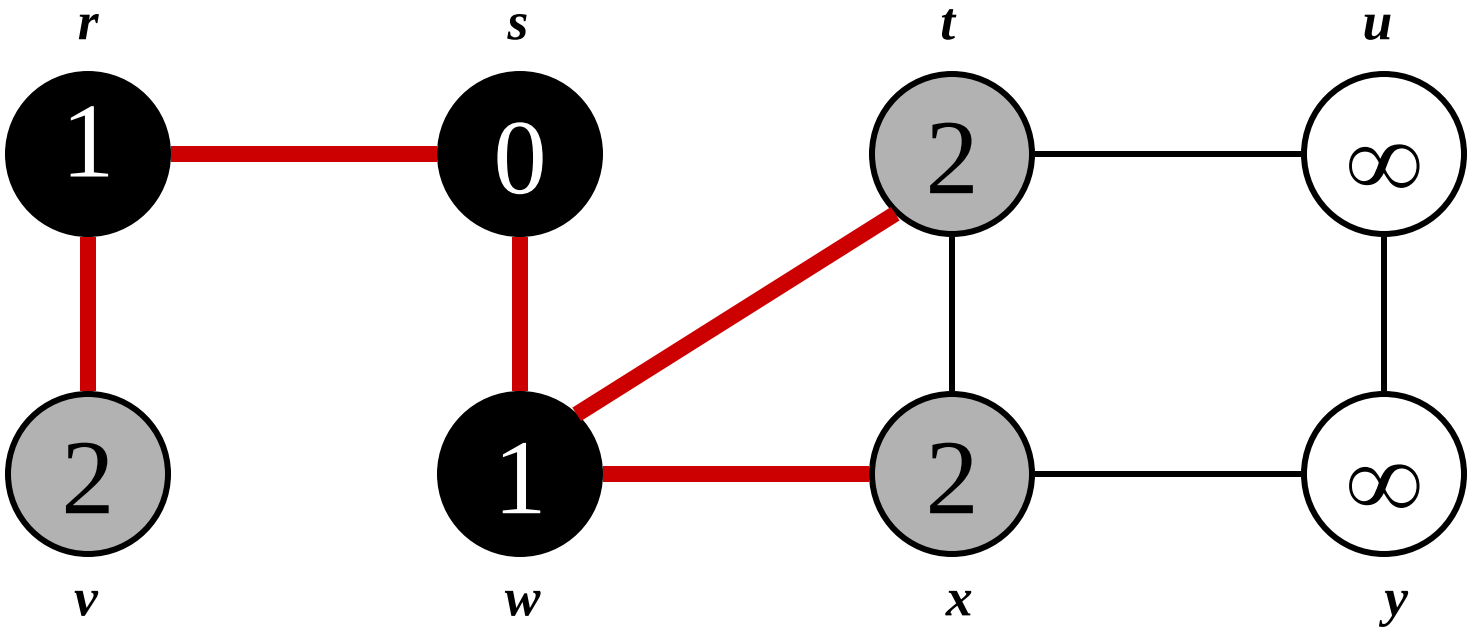
Q: | *w* | *r* |

Breadth-First Search: Example



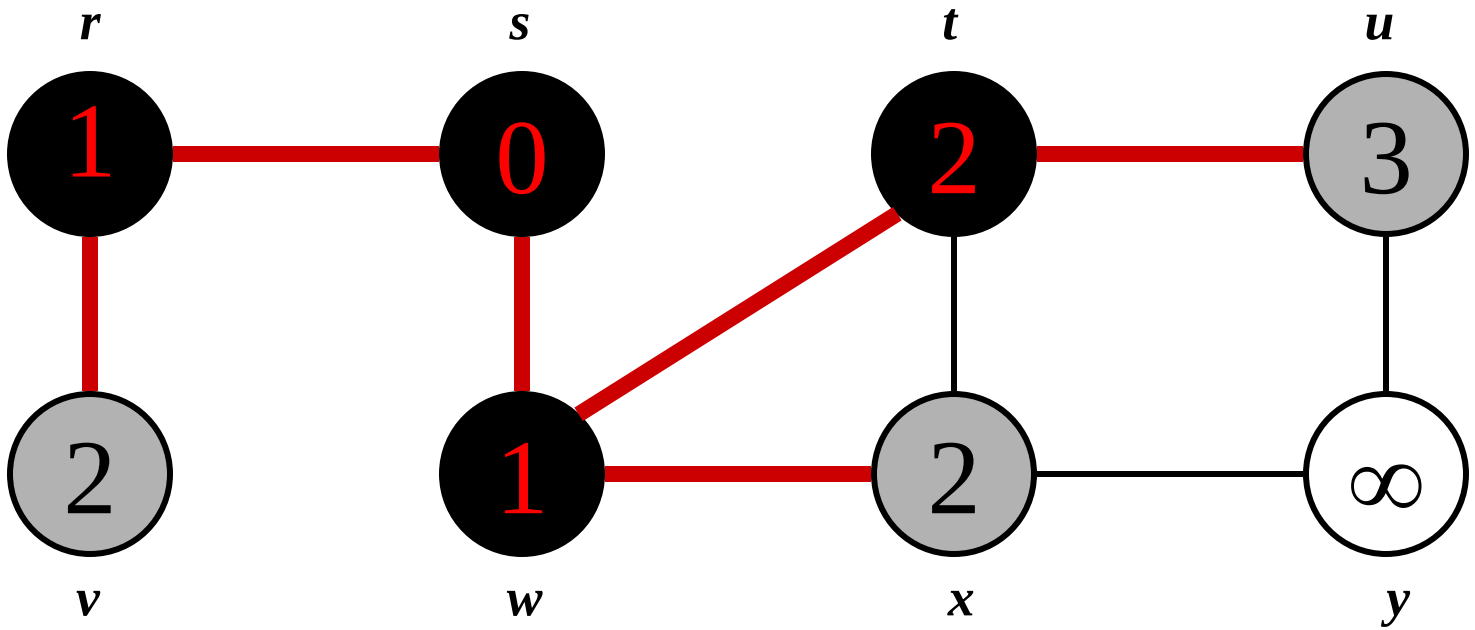
Q: | *r* | *t* | *x* |

Breadth-First Search: Example



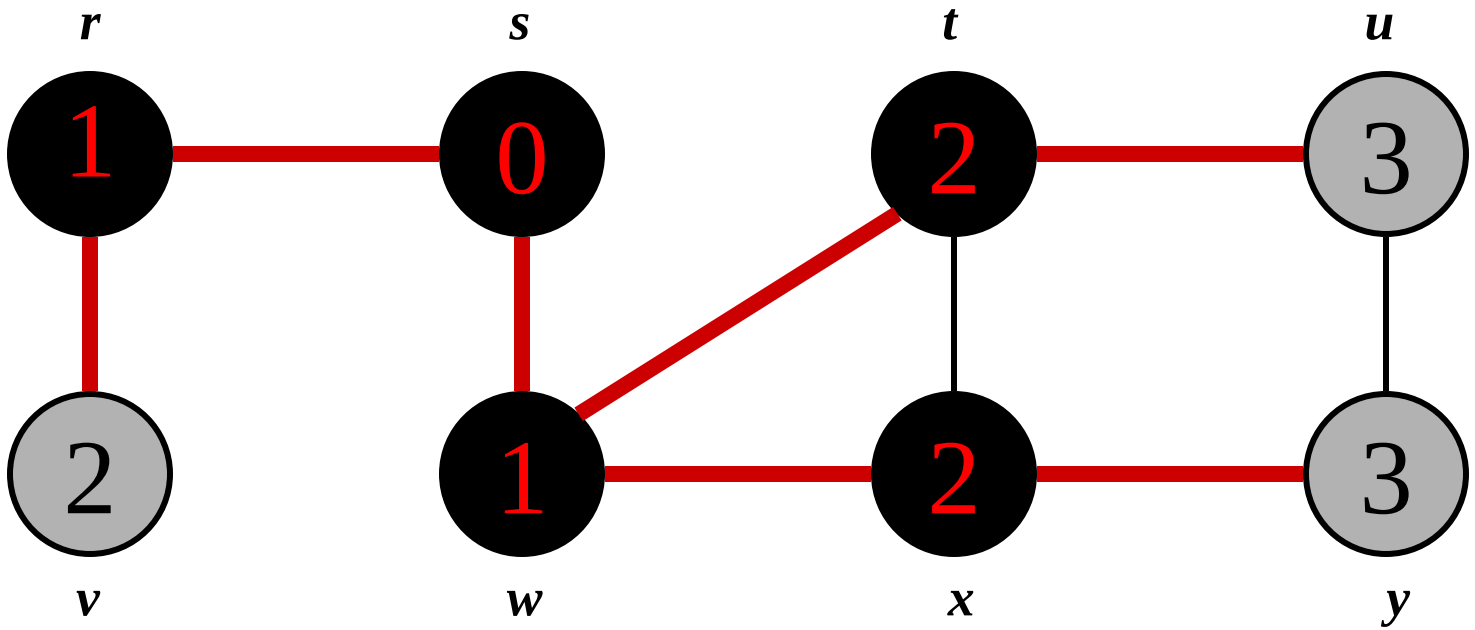
Q: | t | x | v |

Breadth-First Search: Example



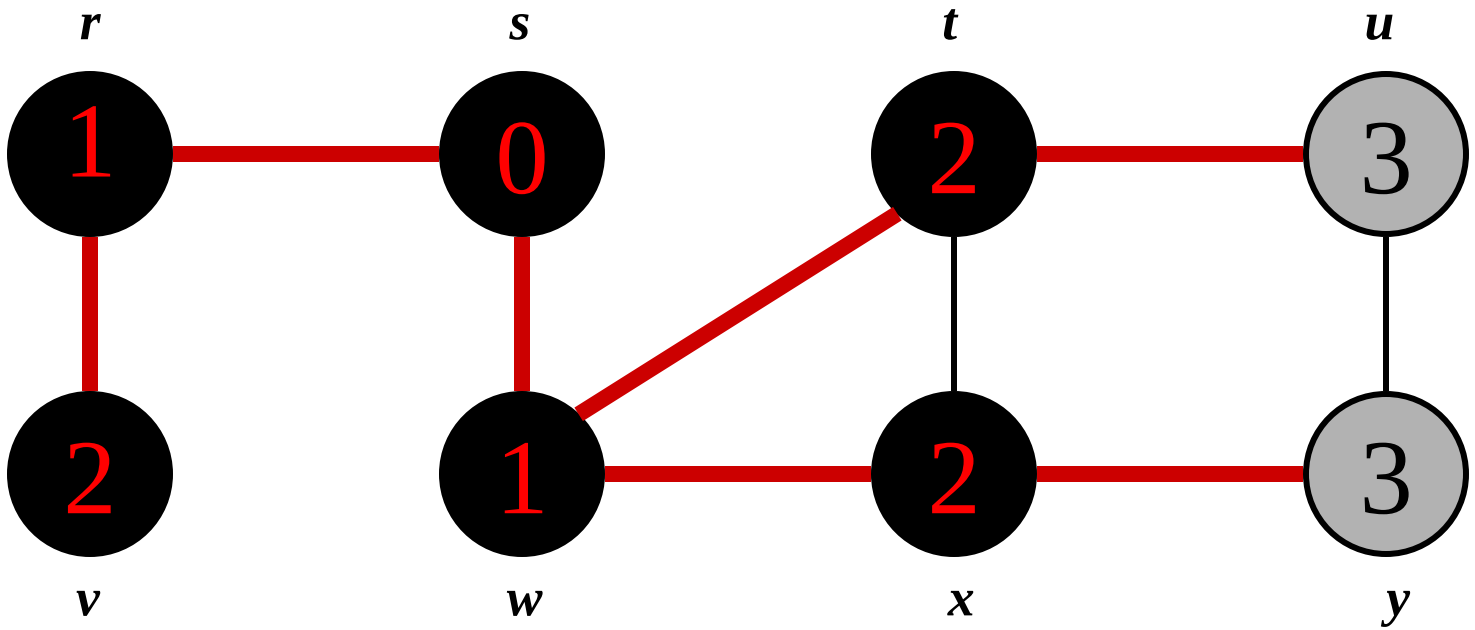
Q: | x | v | u |

Breadth-First Search: Example



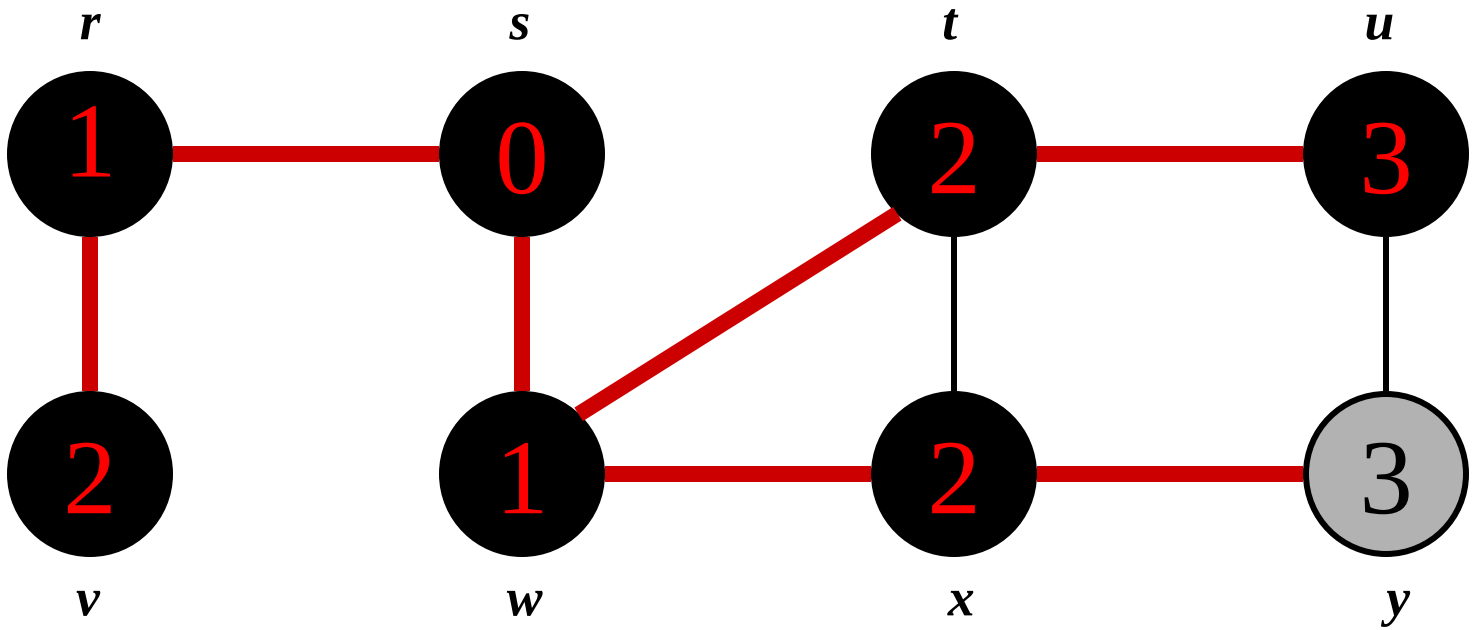
Q: | v | u | y |

Breadth-First Search: Example



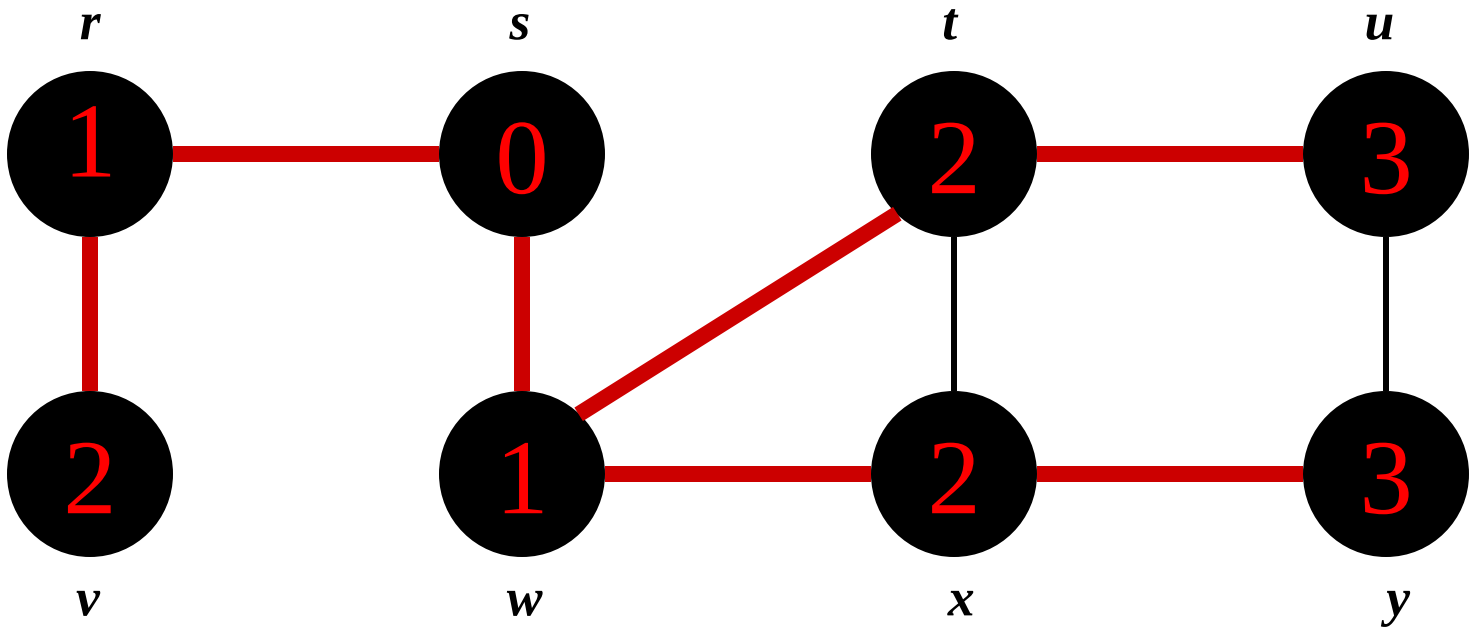
Q: | u | y |

Breadth-First Search: Example



Q: | *y* |

Breadth-First Search: Example



Q: Ø

BFS : Ανάλυση

```
BFS(G, s)
1.      for each v ∈ V – {s}
2.          color[v] = WHITE;
3.          d[v] = ∞;
4.          p[v] = null;
5.      color[s] = GREY;
6.      d[s] = 0;
7.      p[s] = null;
8.      Q = ∅
9.      Enqueue(Q, s);
10.     while (Q not empty)
11.         u = RemoveTop(Q);
12.         for each v ∈ Adj[u]
13.             if (color[v] == WHITE)
14.                 color[v] = GREY;
15.                 d[v] = d[u] + 1;
16.                 p[v] = u;
17.                 Enqueue(Q, v);
18.         color[u] = BLACK;
```

Κάθε κορυφή: O(V)

u = κάθε κορυφή, αλλά μια φορά (γιατί?)

v = κάθε κορυφή που εμφανίζεται σε κάποια από τις λίστες γειτνίασης

Συνολικός χρόνος: O(V+E)

Ποιος είναι ο χρόνος εκτέλεσης?

BFS : Ανάλυση

Μετά την απόδοση αρχικών τιμών καμιά κορυφή δε χρωματίζεται ποτέ λευκή και επομένως ο έλεγχος της γραμμής 13 διασφαλίζει ότι κάθε κορυφή προστίθεται στη ουρά το πολύ 1 φορά και επομένως αφαιρείται από αυτήν επίσης το πολύ μια φορά.

Οι πράξεις της προσθήκης και της αφαίρεσης κορυφής στην ουρά απαιτούν χρόνο $O(1)$. Άρα ο συνολικός χρόνος για τις πράξεις της ουράς είναι $O(V)$.

Η λίστα γειτνίασης κάθε κορυφής διατρέχεται μόνο όταν η κορυφή αφαιρείται από την ουρά, άρα κάθε λίστα διατρέχεται το πολύ μια φορά.

Το άθροισμα των μηκών όλων των λιστών γειτνίασης είναι $O(E)$ ο συνολικός χρόνος που αναλώνεται για τη σάρωση των λιστών αυτών είναι $O(E)$.

Η επιβάρυνση από την απόδοση αρχικών τιμών είναι $O(V)$ και συνεπώς ο συνολικός χρόνος εκτέλεσης της BFS είναι $O(V+E)$, γραμμικός ως προς το μέγεθος του G μέσω λίστας γειτνίασης.

BFS : Ανάλυση (χώρος)

```
BFS(G, s)
1.      for each v ∈ V – {s}
2.          color[v] = WHITE;
3.          d[v] = ∞;
4.          p[v] = null;
5.      color[s] = GREY;
6.      d[s] = 0;
7.      p[s] = null;
8.      Q = ∅
9.      Enqueue(Q, s);
10.     while (Q not empty)
11.         u = RemoveTop(Q);
12.         for each v ∈ Adj[u]
13.             if (color[v] == WHITE)
14.                 color[v] = GREY;
15.                 d[v] = d[u] + 1;
16.                 p[v] = u;
17.                 Enqueue(Q, v);
18.         color[u] = BLACK;
```

Ποιο είναι το συνολικό κόστος για την αποθήκευση του δένδρου

Συνολικός χώρος:
O(max(degree(v))) = O(E)

Breadth-First Search: Ιδιότητες

- BFS υπολογίζει την *απόσταση του συντομότερου μονοπατιού* (*shortest-path distance)* από την κορυφή αφετηρία
  - Απόσταση συντομότερου μονοπατιού $\delta(s,v)$ = ελάχιστος αριθμός ακμών από την *s* στη *v*, ή ∞ αν *v* δεν είναι προσπελάσιμη από την *s*
- BFS κατασκευάζει το *breadth-first tree*, στο οποίο τα μονοπάτια προς τη ρίζα είναι τα συντομότερα μονοπάτια του *G*
  - Ο BFS αλγόριθμος μπορεί να χρησιμοποιηθεί για να υπολογιστούν τα συντομότερα μονοπάτια από μια κορυφή προς μια άλλη σε $O(V+E)$ χρόνο

Τέλος Ενότητας