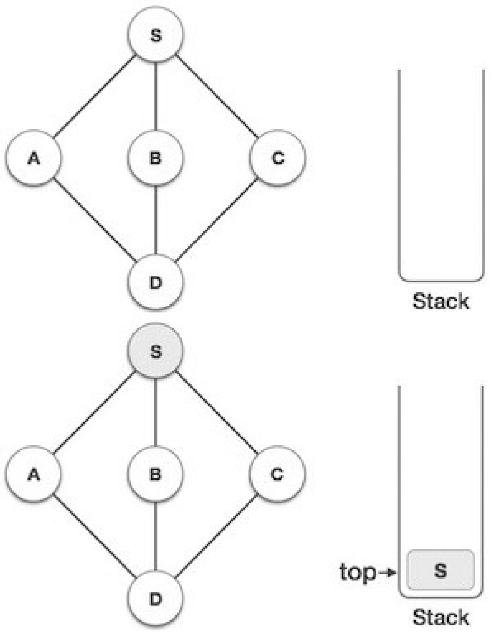
Διάσχιση γράφων

Αλγόριθμοι μείωσε και βασίλευε
Διάσχιση γράφων

- Αναζήτηση κατά βάθος (depth-first search - DFS)
  - Με σωρό – ώθηση και εξαγωγή από την κορυφή του σωρού
- Αναζήτηση κατά πλάτος (breadth-first search - BFS)
  - Με ουρά

Αλγόριθμοι μείωσε και βασίλευε
Διάσχιση γράφων - Αναζήτηση κατά βάθος

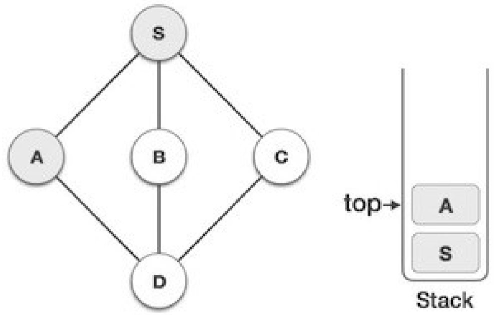


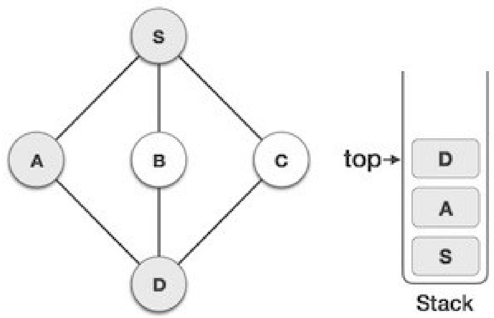
- Κανόνας 1 – Επίσκεψη γειτονικού μη επισκεπτόμενου κόμβου. Σημείωσή του ως επισκεπτόμενο. Ώθηση του στο σωρό

- Κανόνας 2 – Εάν δεν υπάρχει κανέναν γειτονικός κόμβος, εξαγωγή κόμβου από το σωρό

- Κανόνας 3 – Επανάληψη κανόνων 1 και 2 μέχρις ώτου ο σωρός είναι άδειος

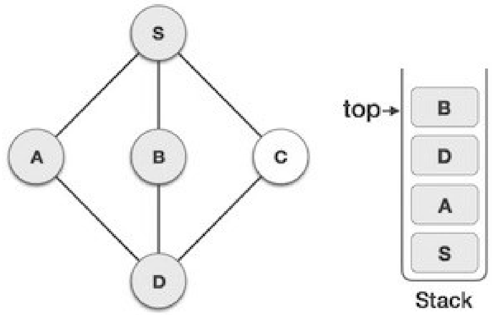
Αλγόριθμοι μείωσε και βασίλευε
Διάσχιση γράφων - Αναζήτηση κατά βάθος

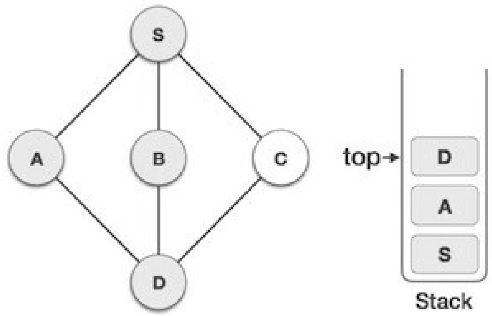




- Κανόνας 1 – Επίσκεψη γειτονικού μη επισκεπτόμενου κόμβου. Σημείωσή του ως επισκεπτόμενο. Ώθηση του στο σωρό
- Κανόνας 2 – Εάν δεν υπάρχει κανέναν γειτονικός κόμβος, εξαγωγή κόμβου από το σωρό
- Κανόνας 3 – Επανάληψη κανόνων 1 και 2 μέχρις ώτου ο σωρός είναι άδειος

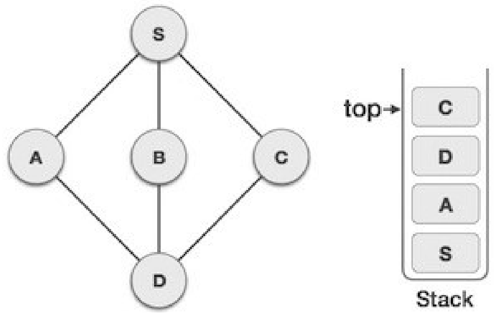
Αλγόριθμοι μείωσε και βασίλευε
Διάσχιση γράφων - Αναζήτηση κατά βάθος





- Κανόνας 1 – Επίσκεψη γειτονικού μη επισκεπτόμενου κόμβου. Σημείωσή του ως επισκεπτόμενο. Ώθηση του στο σωρό
- Κανόνας 2 – Εάν δεν υπάρχει κανέναν γειτονικός κόμβος, εξαγωγή κόμβου από το σωρό
- Κανόνας 3 – Επανάληψη κανόνων 1 και 2 μέχρις ώτου ο σωρός είναι άδειος

Αλγόριθμοι μείωσε και βασίλευε
Διάσχιση γράφων - Αναζήτηση κατά βάθος

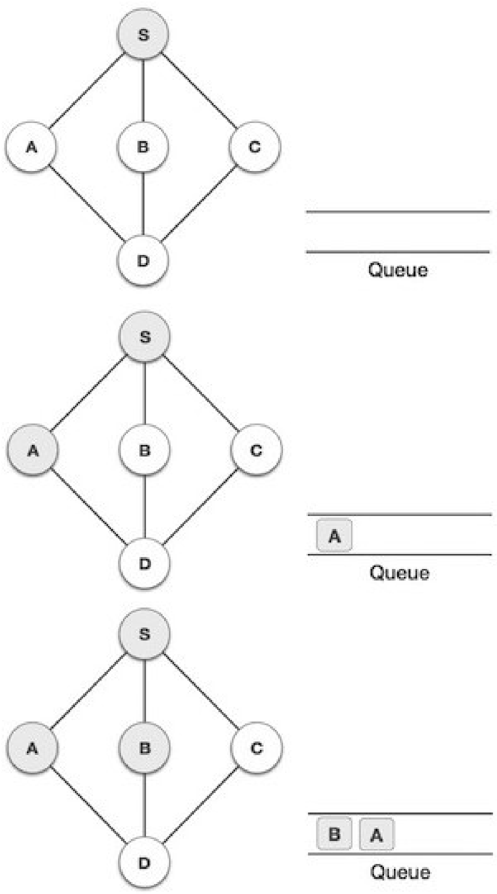


- Κανόνας 1 – Επίσκεψη γειτονικού μη επισκεπτόμενου κόμβου. Σημείωσή του ως επισκεπτόμενο. Ώθηση του στο σωρό

- Κανόνας 2 – Εάν δεν υπάρχει κανέναν γειτονικός κόμβος, εξαγωγή κόμβου από το σωρό

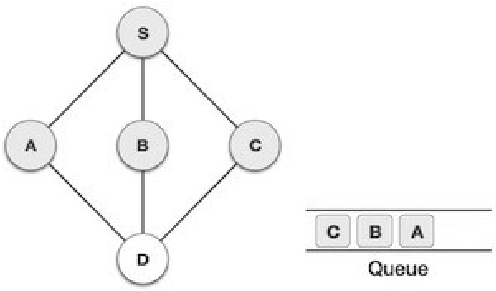
- Κανόνας 3 – Επανάληψη κανόνων 1 και 2 μέχρις ώτου ο σωρός είναι άδειος

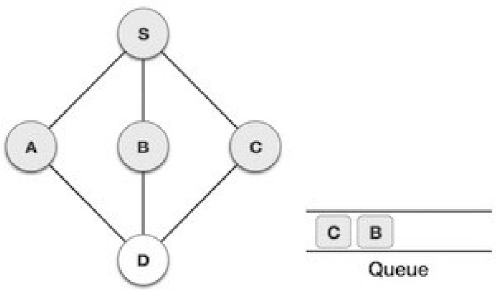
Αλγόριθμοι μείωσε και βασίλευε
Διάσχιση γράφων - Αναζήτηση κατά πλάτος

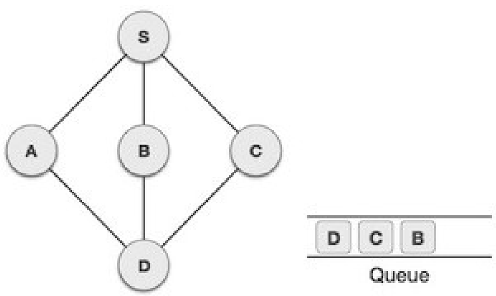


- Κανόνας 1 – Επίσκεψη γειτονικού μη επισκεπτόμενου κόμβου. Σημείωσή του ως επισκεπτόμενο. Εισαγωγή του σε ουρά
- Κανόνας 2 – Εάν δεν υπάρχει κανέναν γειτονικός κόμβος, αφαίρεση πρώτου κόμβου από την ουρά
- Κανόνας 3 – Επανάληψη κανόνων 1 και 2 μέχρις ώτου η ουρά είναι άδεια

Αλγόριθμοι μείωσε και βασίλευε
Διάσχιση γράφων - Αναζήτηση κατά πλάτος





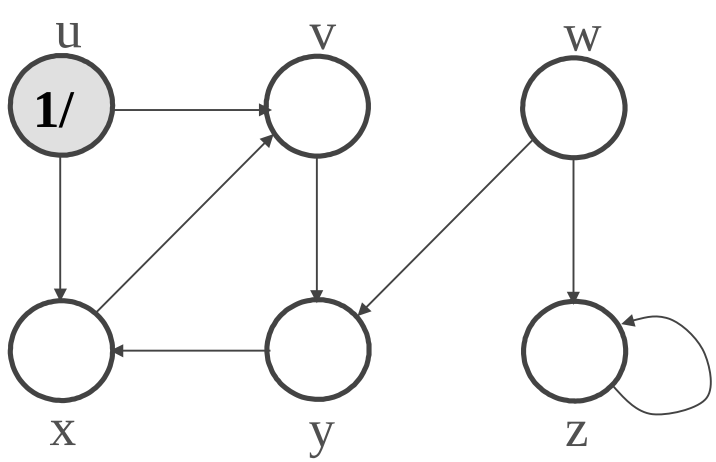


- Κανόνας 1 – Επίσκεψη γειτονικού μη επισκεπτόμενου κόμβου. Σημείωσή του ως επισκεπτόμενο. Εισαγωγή του στην αρχή της ουράς

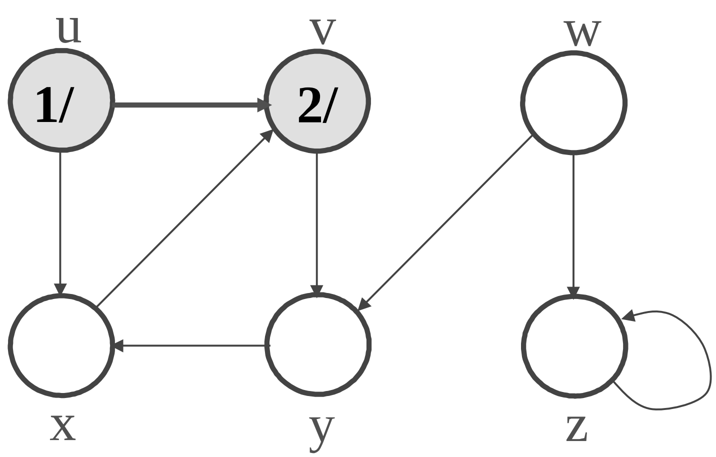
- Κανόνας 2 – Εάν δεν υπάρχει κανέναν γειτονικός κόμβος, αφαίρεση πρώτου κόμβου από το τέλος της ουράς

- Κανόνας 3 – Επανάληψη κανόνων 1 και 2 μέχρις ώτου η ουρά είναι άδεια

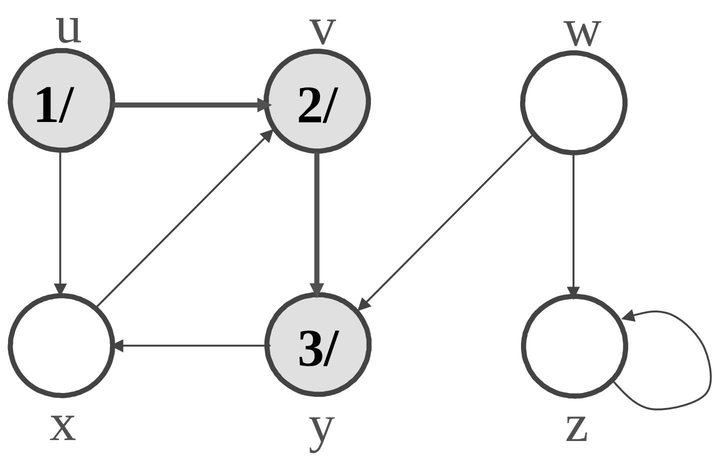
Παράδειγμα (DFS)



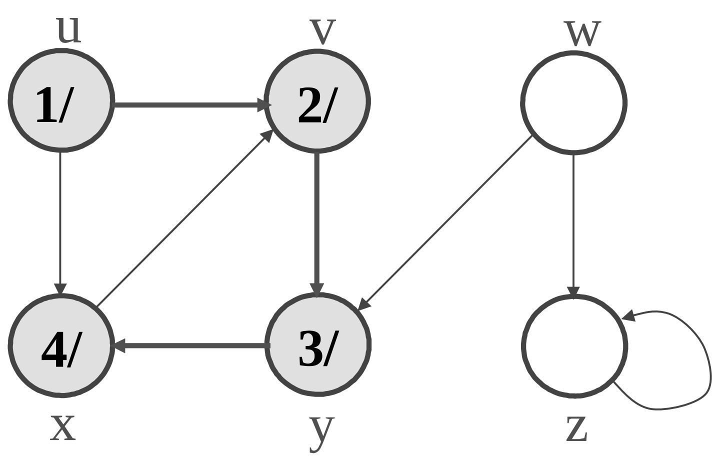
Παράδειγμα (DFS)



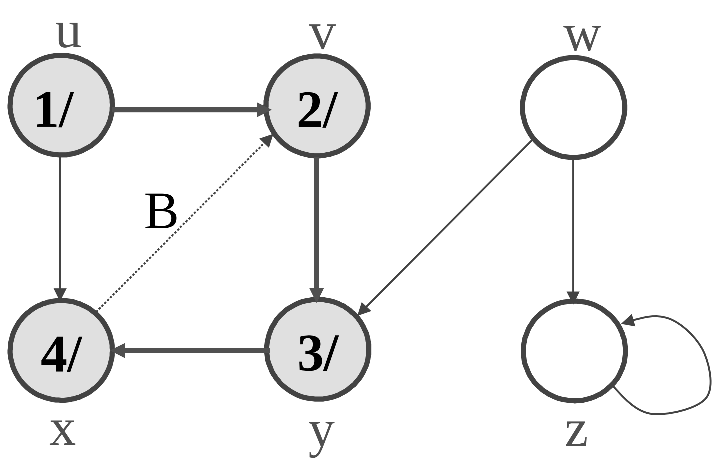
Παράδειγμα (DFS)



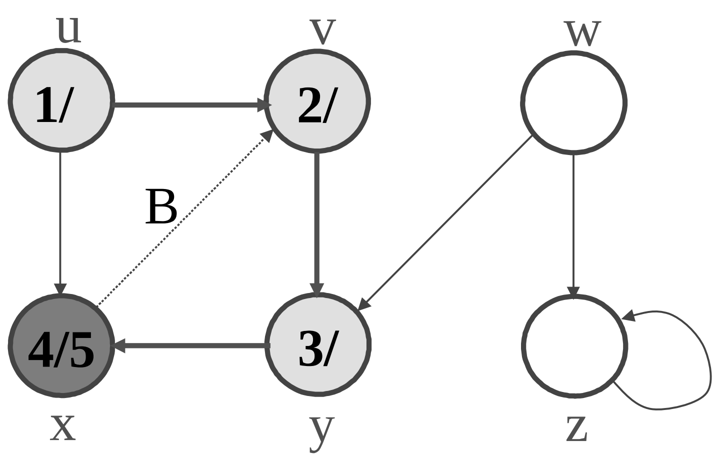
Παράδειγμα (DFS)



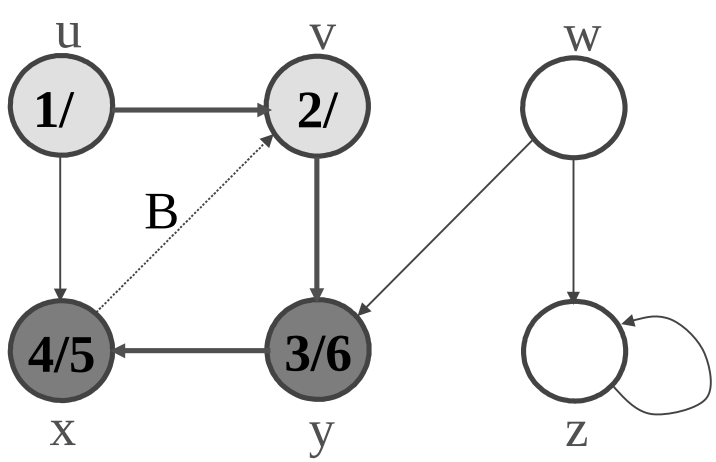
Παράδειγμα (DFS)



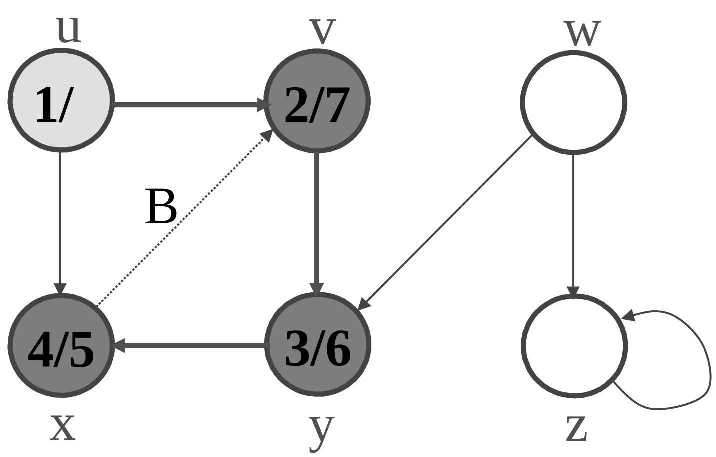
Παράδειγμα (DFS)



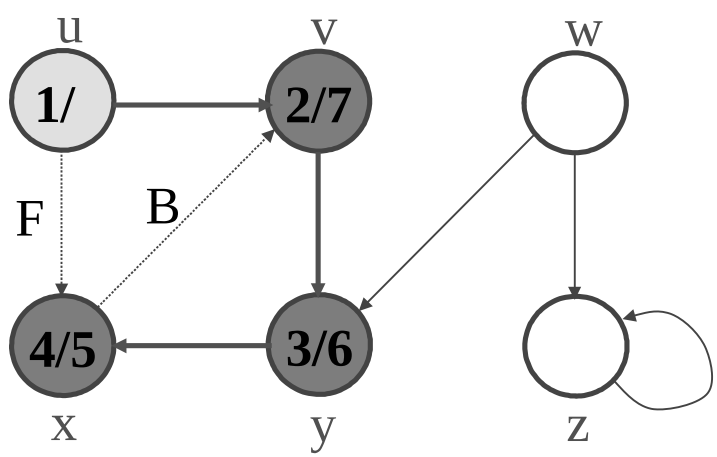
Παράδειγμα (DFS)



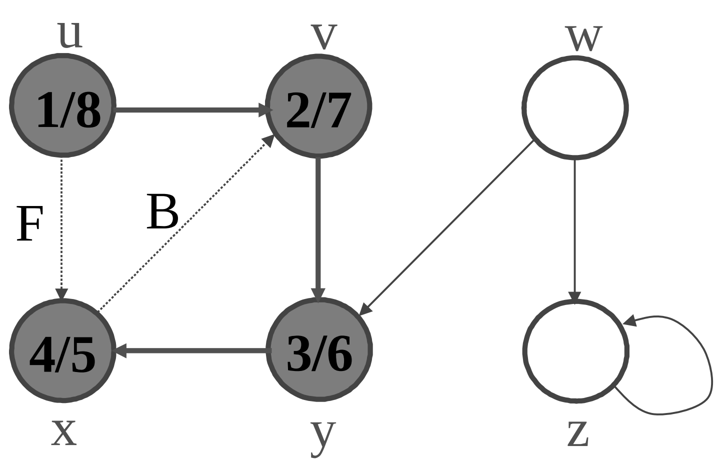
Παράδειγμα (DFS)



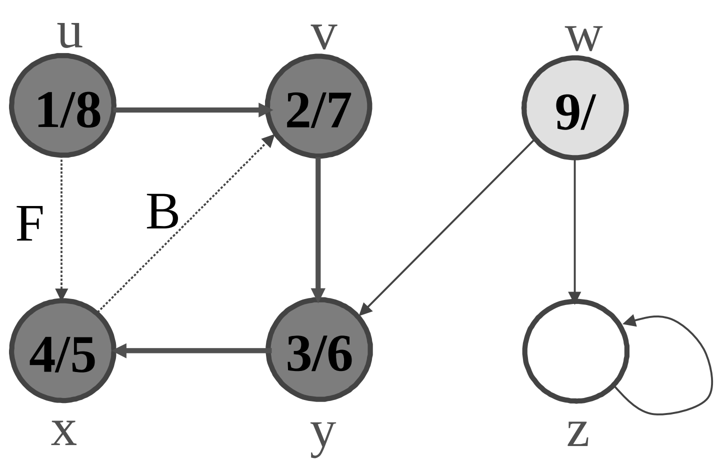
Παράδειγμα (DFS)



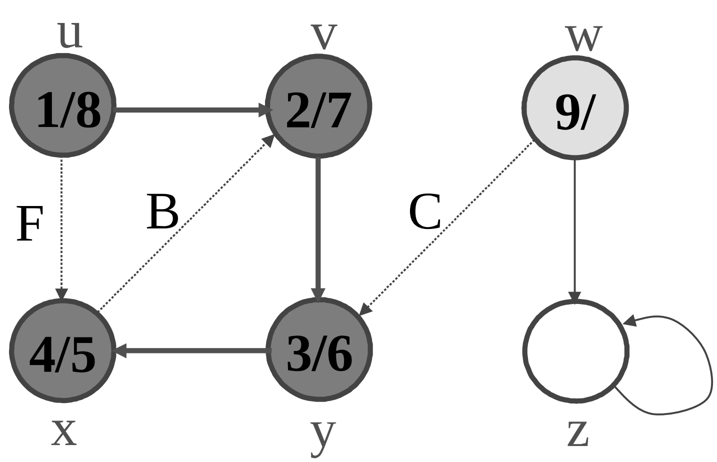
Παράδειγμα (DFS)



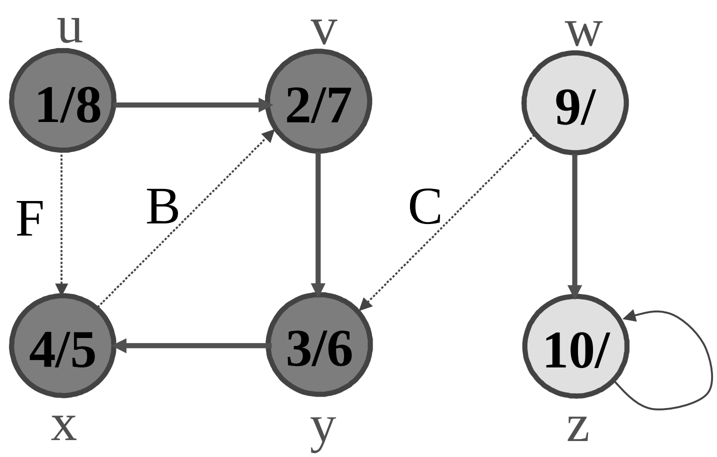
Παράδειγμα (DFS)



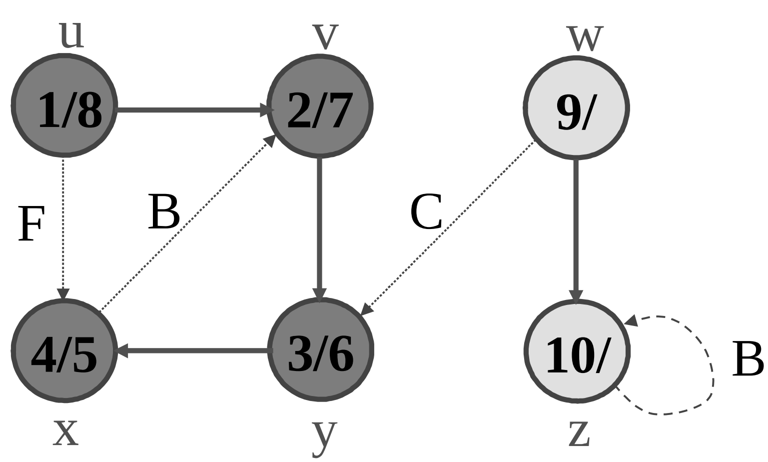
Παράδειγμα (DFS)



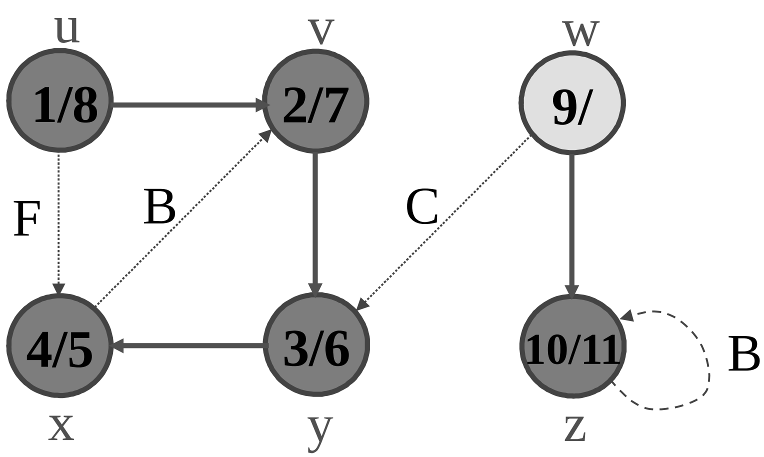
Παράδειγμα (DFS)



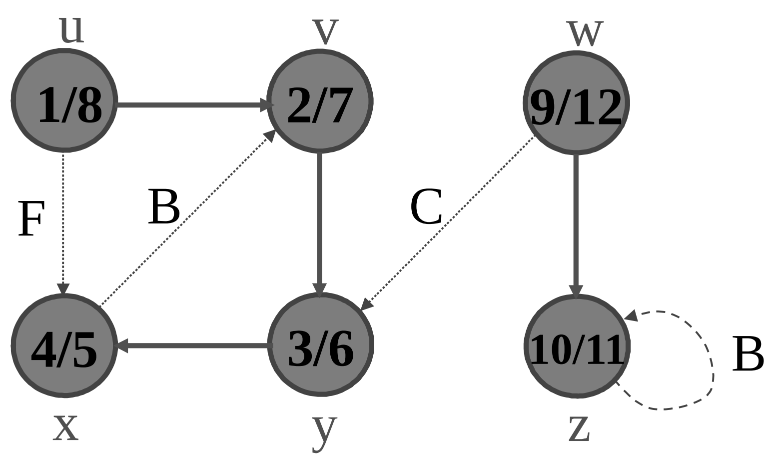
Παράδειγμα (DFS)



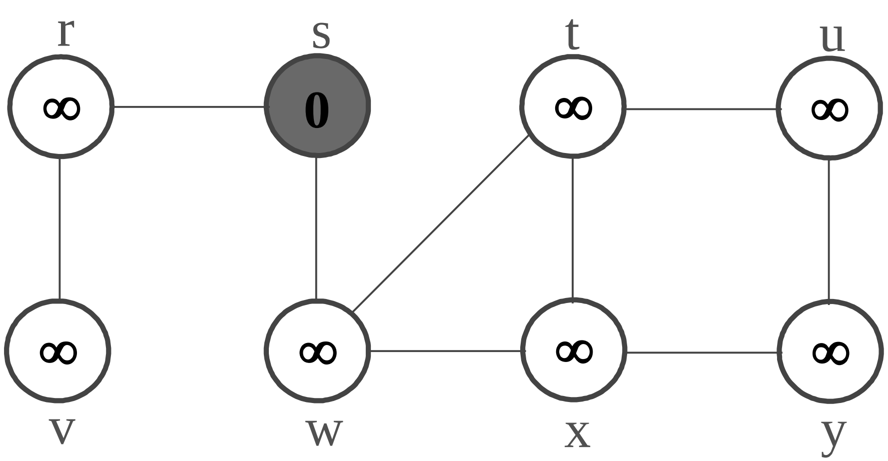
Παράδειγμα (DFS)



Παράδειγμα (DFS)

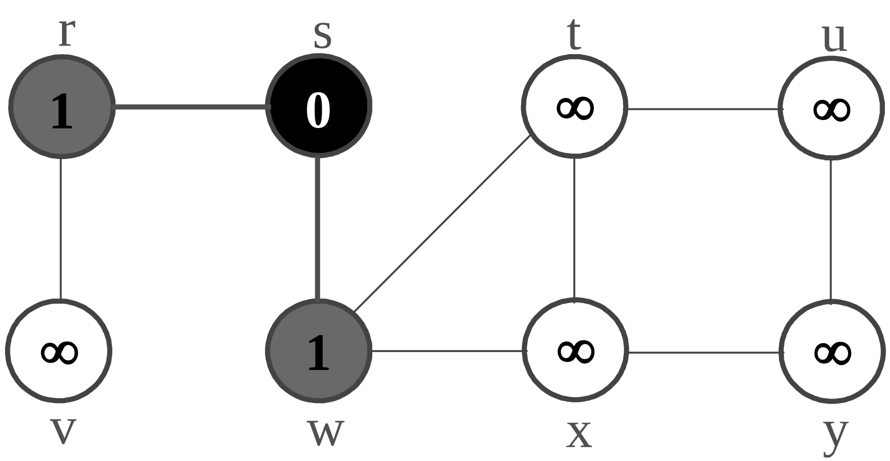


Παράδειγμα (BFS)



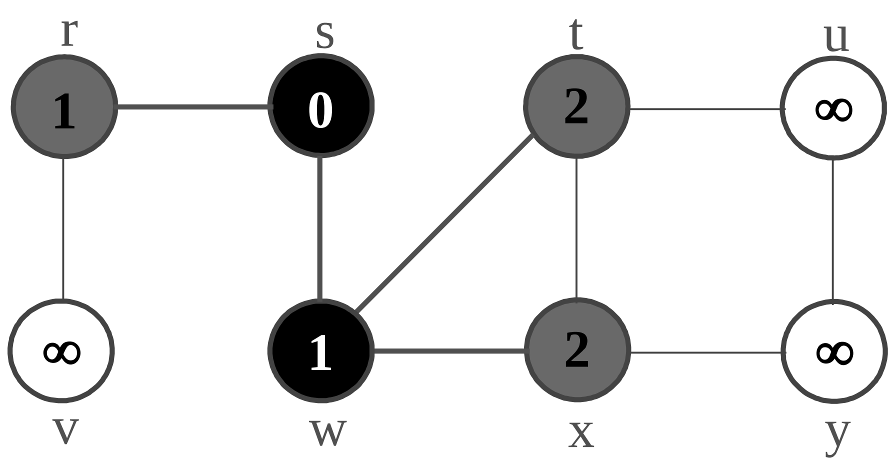
Q: s
0

Παράδειγμα (BFS)



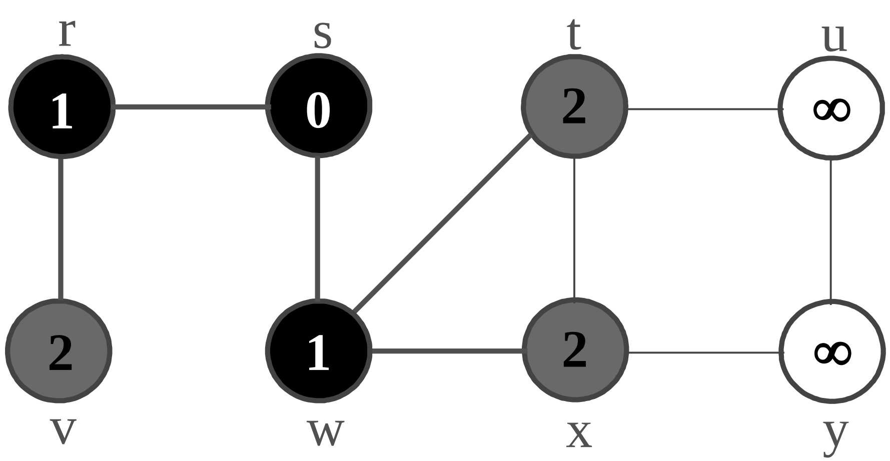
Q:	w	r
	1	1

Παράδειγμα (BFS)



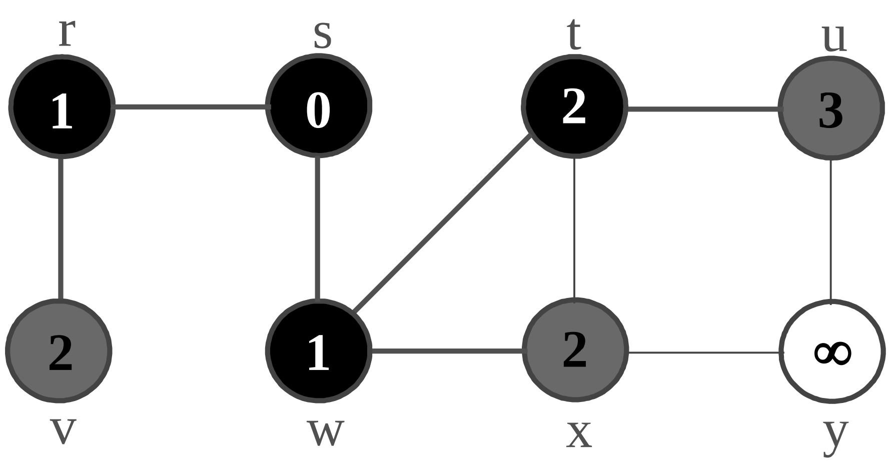
Q:	r	t	x
	1	2	2

Παράδειγμα (BFS)



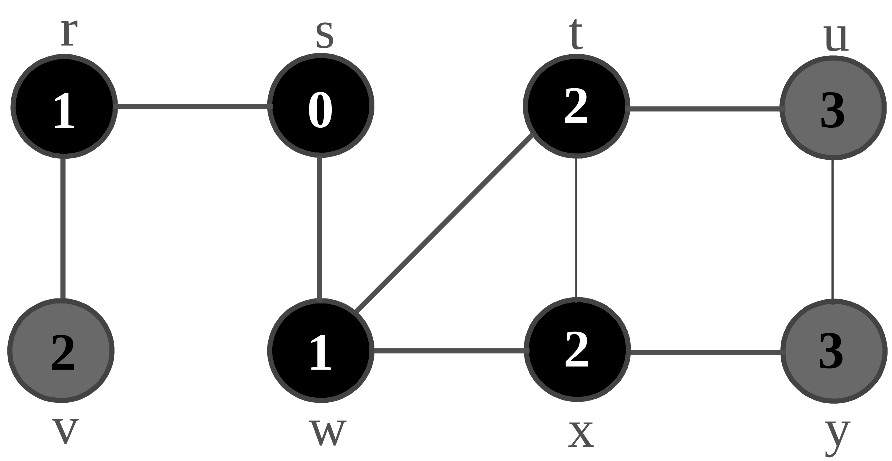
Q:	t	x	v
	2	2	2

Παράδειγμα (BFS)



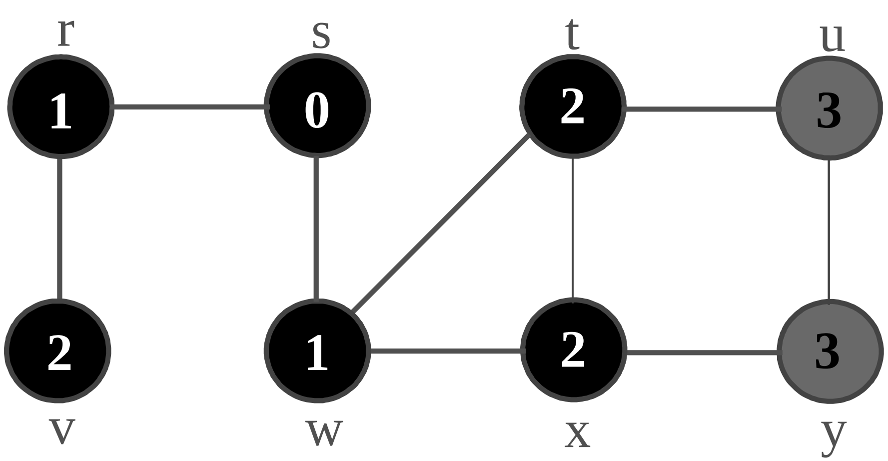
Q:	x	v	u
	2	2	3

Παράδειγμα (BFS)



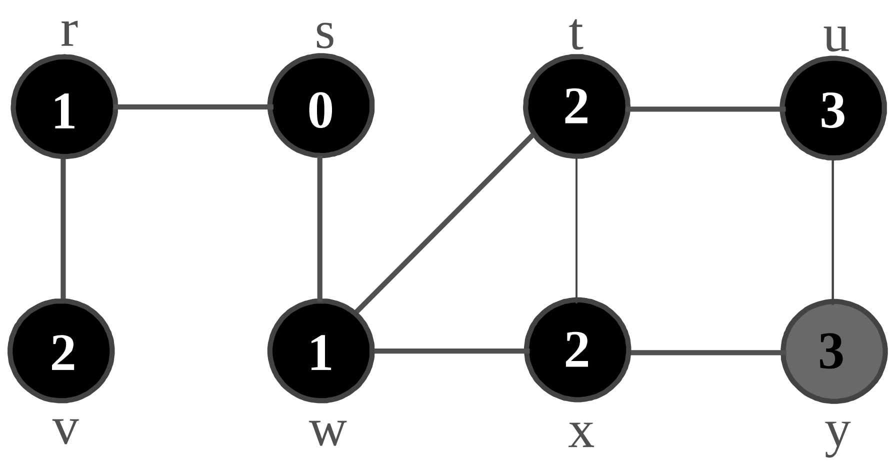
Q:	v	u	y
	2	3	3

Παράδειγμα (BFS)



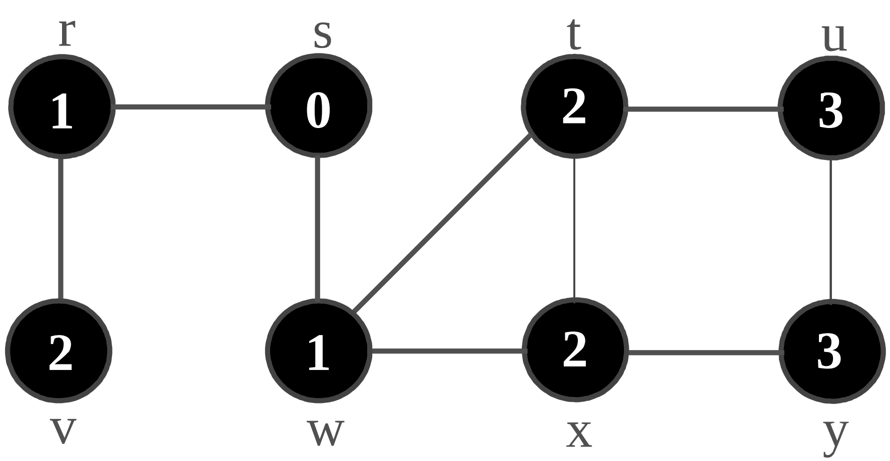
Q:	u	y
	3	3

Παράδειγμα (BFS)



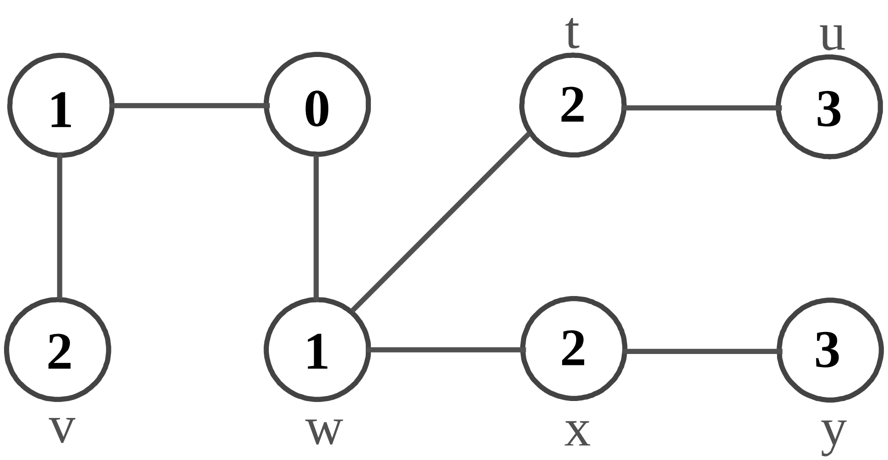
Q: y
3

Παράδειγμα (BFS)



Q: $\emptyset$

Παράδειγμα (BFS)



BFS Δένδρο

Συντομότερα μονοπάτια

Συντομότερα μονοπάτια

- **Σε μη ζυγισμένο γράφο τα συντομότερα μονοπάτια βρίσκονται με διάσχιση BFS**
- Αλγόριθμοι για ζυγισμένους γράφους:
  - Dijkstra (1-to-all, χωρίς αρνητικά βάρη)
  - Bellman-Ford (1-to-all, με αρνητικά βάρη)
  - Johnson (all-to-all, με αρνητικά βάρη)
  - Floyd (all-to-all)
  - Warshall (μεταβατική κλειστότητα)

Αλγόριθμος Dijkstra (1956, 1959)

- **Προσέγγιση**: Άπληστη
- **Είσοδος**: ζυγισμένος γράφος G={E,V} και κορυφή v∈V, με μη αρνητικά βάρη
- **Έξοδος**: τα συντομότερα μονοπάτια από την κορυφή v∈V προς όλες τις άλλες κορυφές (1-to-all)
- Λειτουργεί αντίστοιχα με τον Prim για MST: Υπολογισμός σε κάθε βήμα της κορυφής με τη μικρότερη απόσταση που δεν έχει υπολογιστεί ήδη

Αλγόριθμος Dijkstra (1956, 1959)

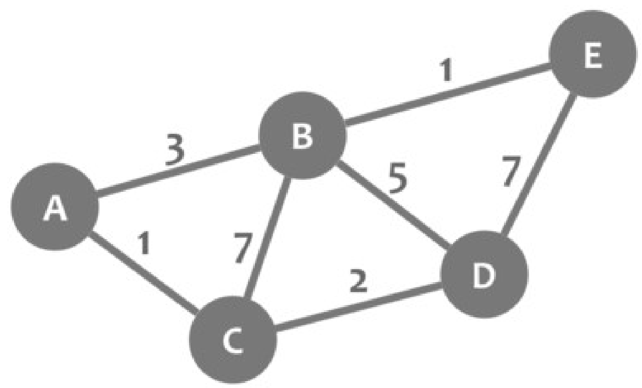
- Σημείωση όλων των κορυφών σαν μη επισκεπτόμενων
- Θέση *προσωρινών* αποστάσεων σε όλες τις κορυφές: 0 στην αρχική κορυφή και άπειρο στις υπόλοιπες
- Για την τρέχουσα κορυφή, υπολογισμός της *προσωρινής* της απόστασης προς όλες τις μη επισκεπτόμενες γειτονικές της. Αντικατάσταση της ήδη υπάρχουσας απόστασης εάν η υπολογισμένη είναι μικρότερη
- Όταν υπολογιστούν οι αποστάσεις πρός όλες τις γειτονικές κορυφές, σημείωση της τρέχουσας κορυφής σαν επισκεπτόμενη. Μια επισκεπτόμενη κορυφή δεν ελέγχεται ξανά
- Εάν δεν υπάρχει σύνδεση μεταξύ της τρέχουσας κορυφής και των υπολοιπόμενων μη επισκεπτόμενων κορυφών, ο αλγόριθμος τερματίζει
- Αλλιώς επιλογή του με επισκεπτόμενου κόμβου με την μικρότερη *προσωρινή* απόσταση και επανάληψη

Αλγόριθμος Dijkstra (1956, 1959)

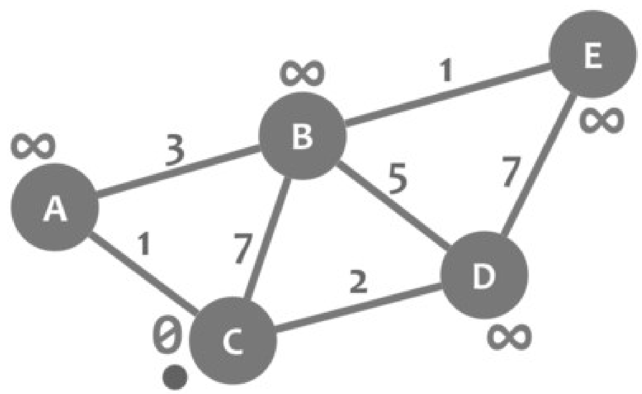
```
function Dijkstra(Graph, source):
    dist[source] ← 0                                  // Initialization

    create vertex set Q

    for each vertex v in Graph:
        if v ≠ source
            dist[v] ← INFINITY                        // Unknown distance from source to v
            prev[v] ← UNDEFINED                       // Predecessor of v

        Q.add_with_priority(v, dist[v])


    while Q is not empty:                             // The main loop
        u ← Q.extract_min()                           // Remove and return best vertex
        for each neighbor v of u:                     // only v that is still in Q
            alt = dist[u] + length(u, v)
            if alt < dist[v]
                dist[v] ← alt
                prev[v] ← u
                Q.decrease_priority(v, alt)

    return dist[], prev[]
```

- Χρήση ουράς προτεραιότητας

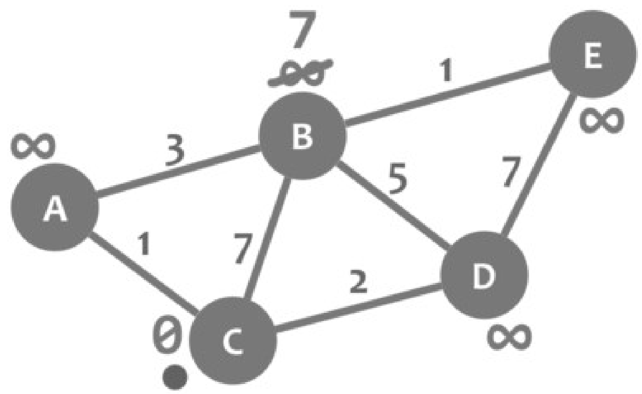
Αλγόριθμος Dijkstra (1956, 1959)



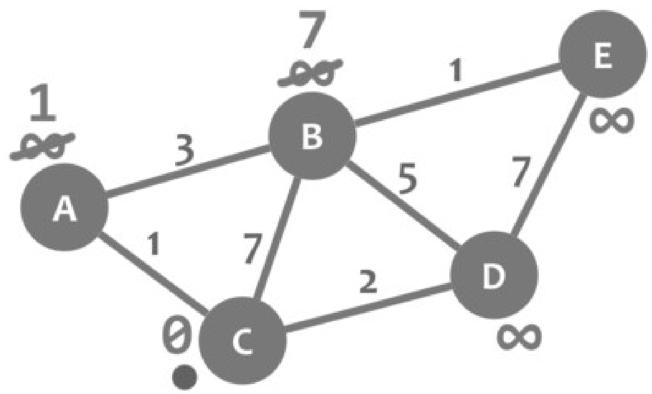
Αλγόριθμος Dijkstra (1956, 1959)



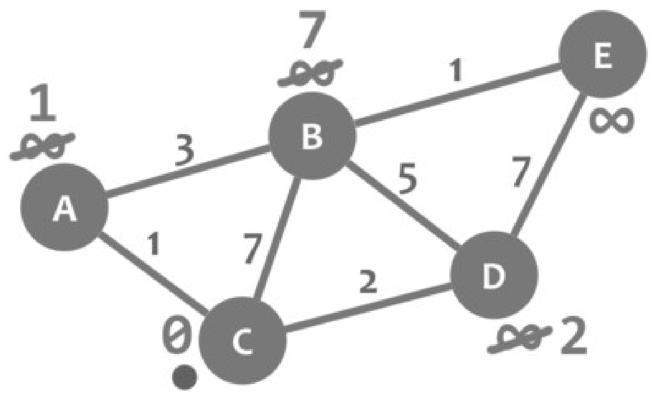
Αλγόριθμος Dijkstra (1956, 1959)



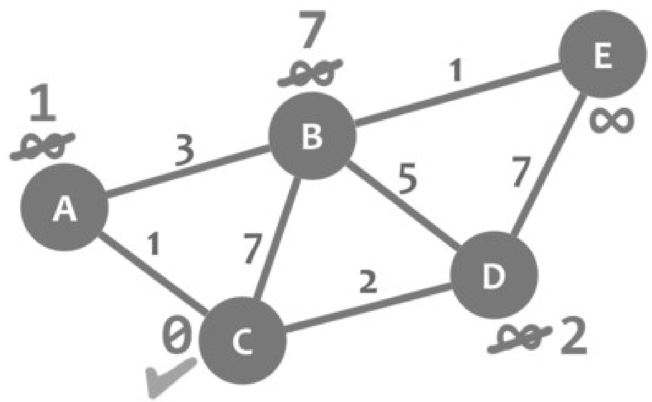
Αλγόριθμος Dijkstra (1956, 1959)



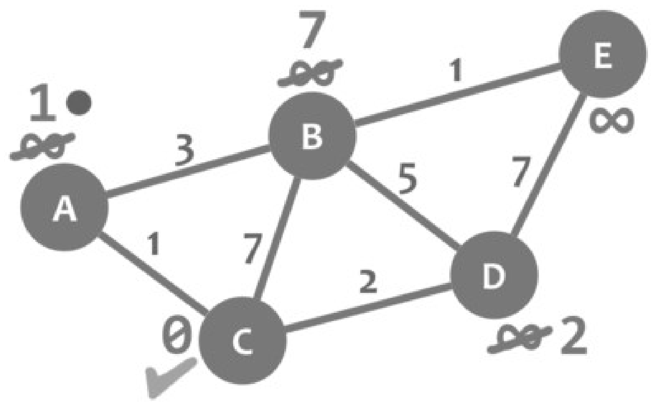
Αλγόριθμος Dijkstra (1956, 1959)



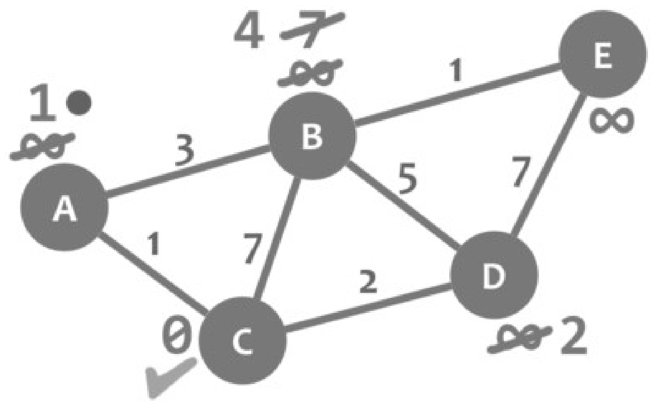
Αλγόριθμος Dijkstra (1956, 1959)



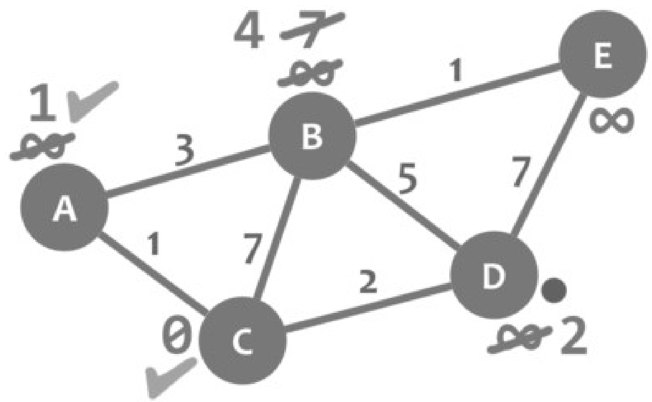
Αλγόριθμος Dijkstra (1956, 1959)



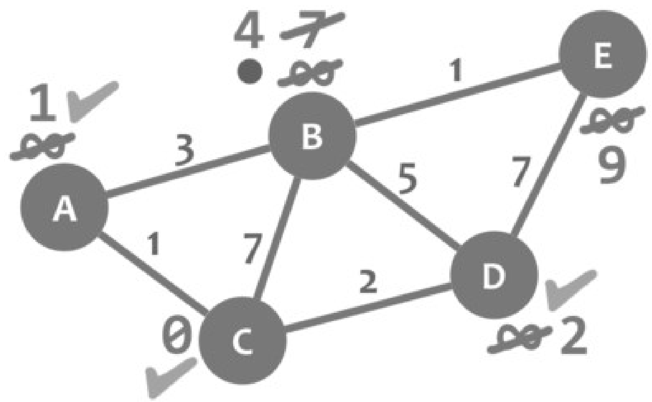
Αλγόριθμος Dijkstra (1956, 1959)



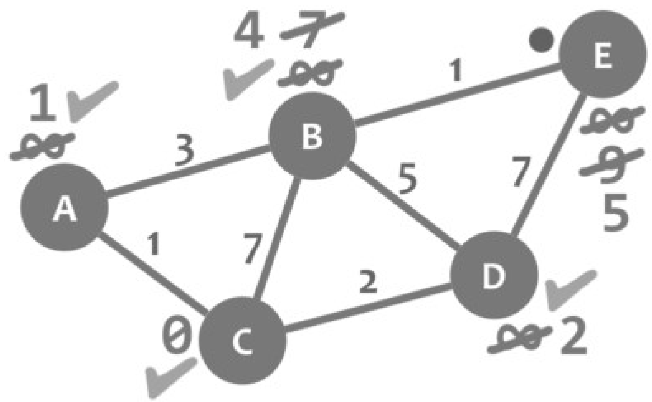
Αλγόριθμος Dijkstra (1956, 1959)



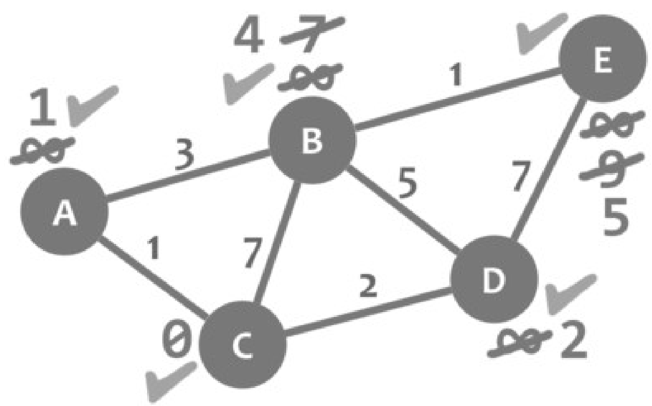
Αλγόριθμος Dijkstra (1956, 1959)



Αλγόριθμος Dijkstra (1956, 1959)

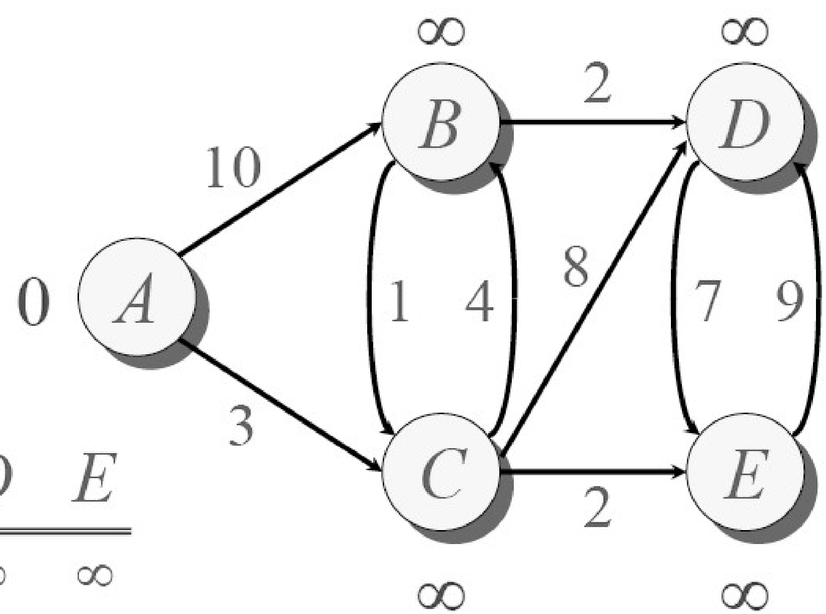


Αλγόριθμος Dijkstra (1956, 1959)



Αλγόριθμος Dijkstra (1956, 1959)

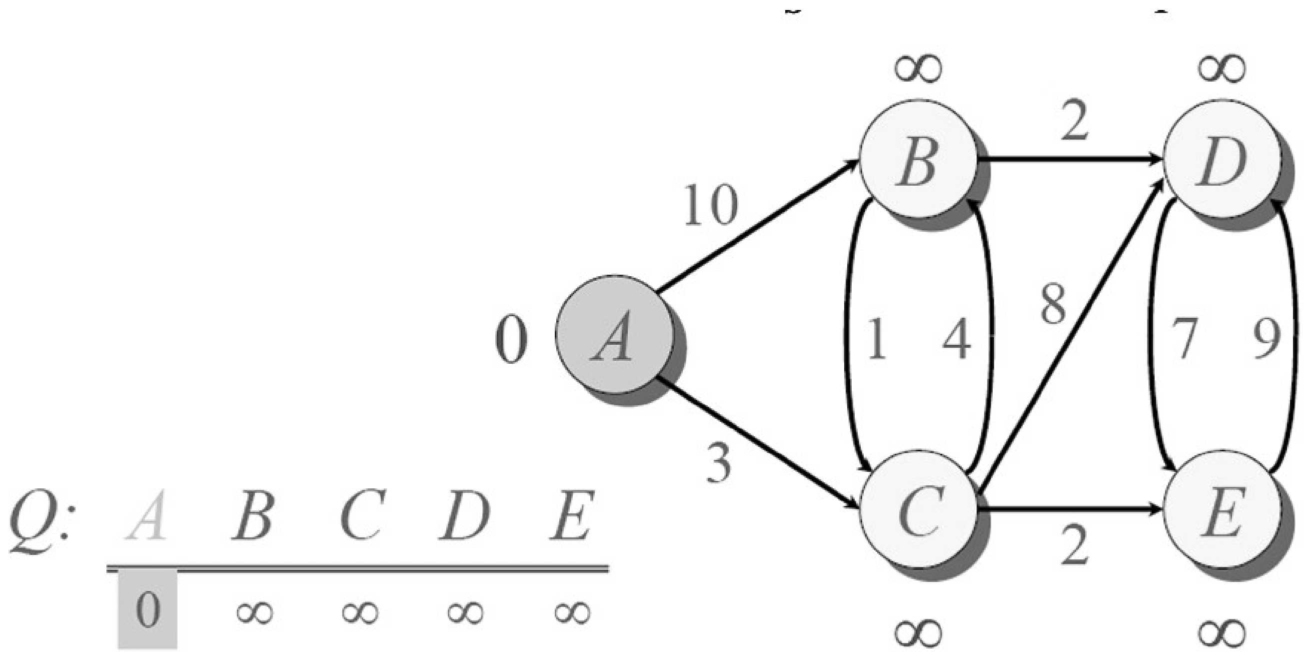
Initialize:



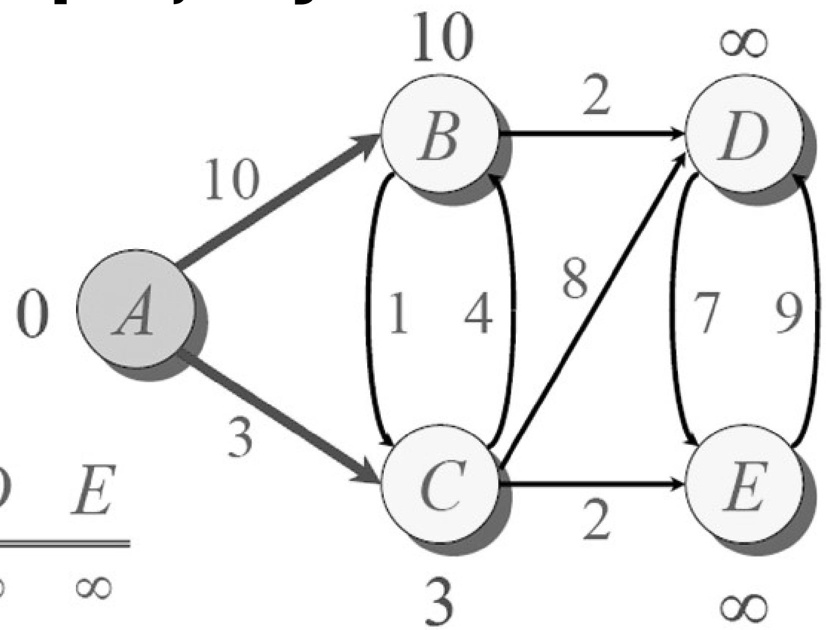
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Αλγόριθμος Dijkstra (1956, 1959)



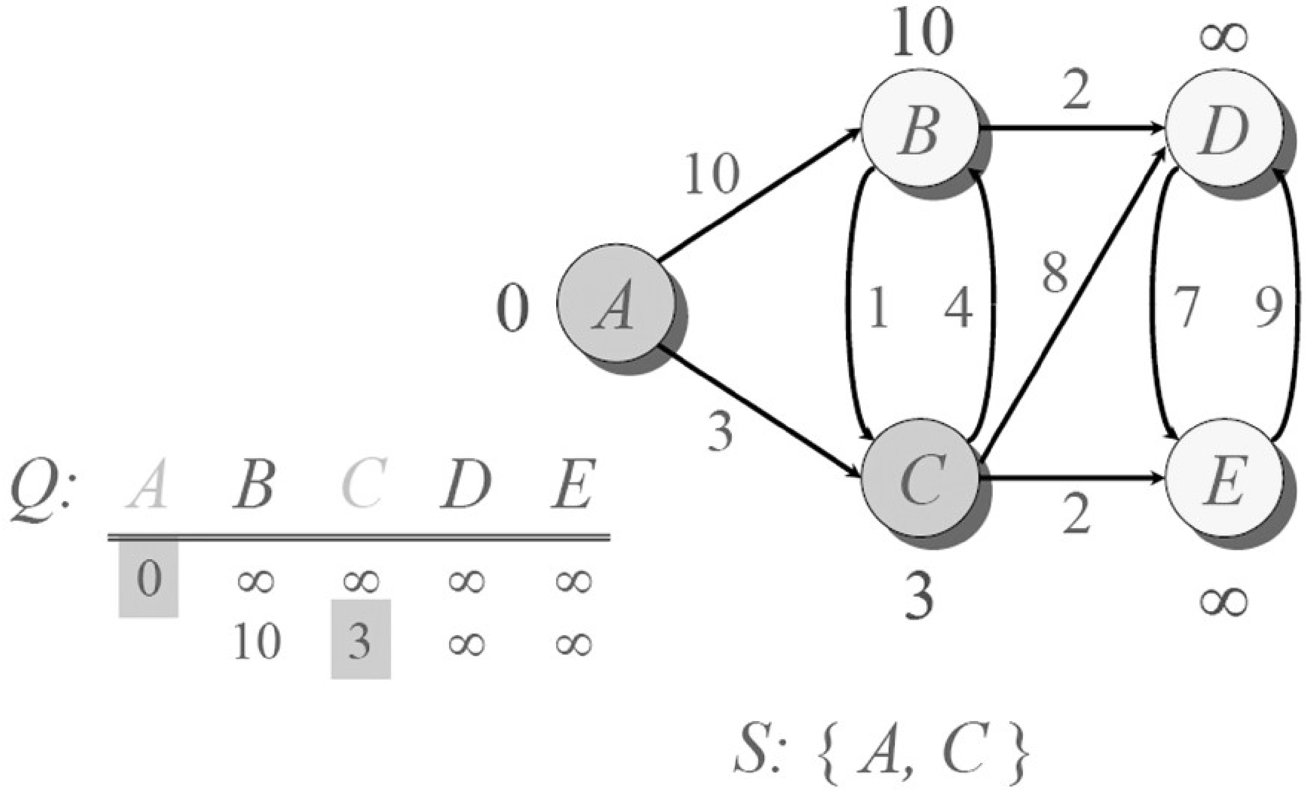
Αλγόριθμος Dijkstra (1956, 1959)



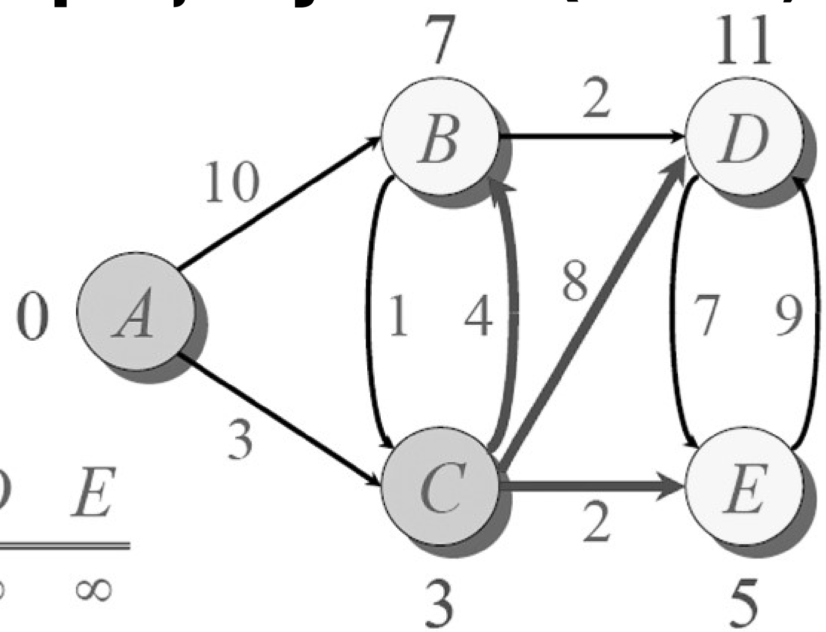
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

Αλγόριθμος Dijkstra (1956, 1959)



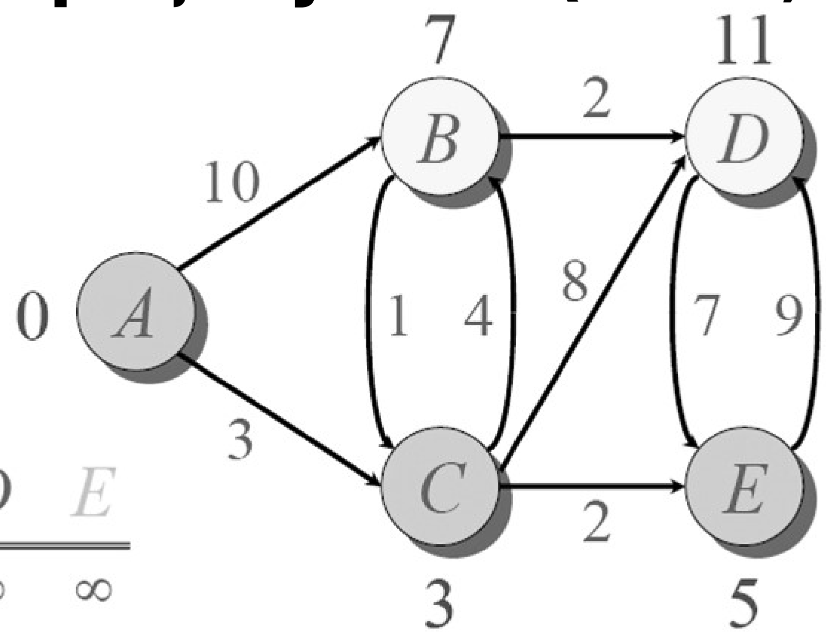
Αλγόριθμος Dijkstra (1956, 1959)



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

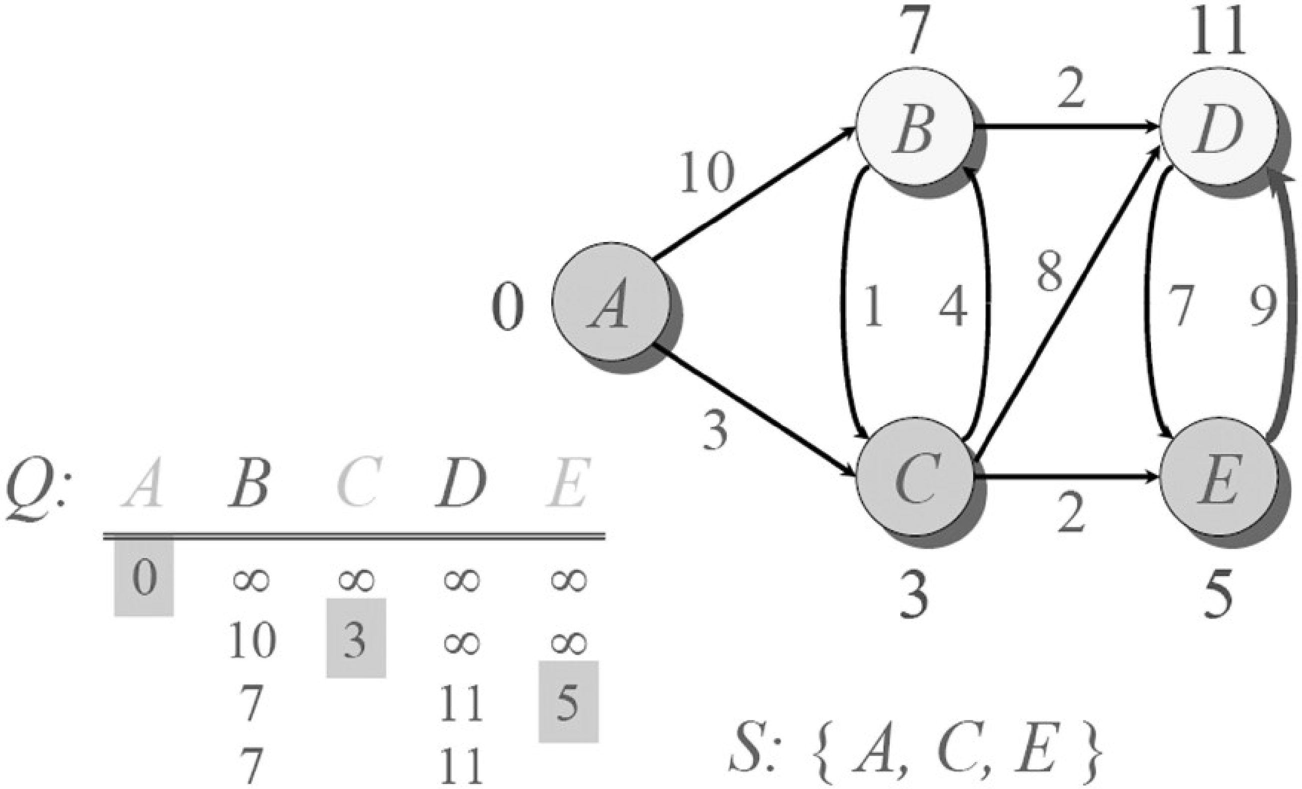
Αλγόριθμος Dijkstra (1956, 1959)



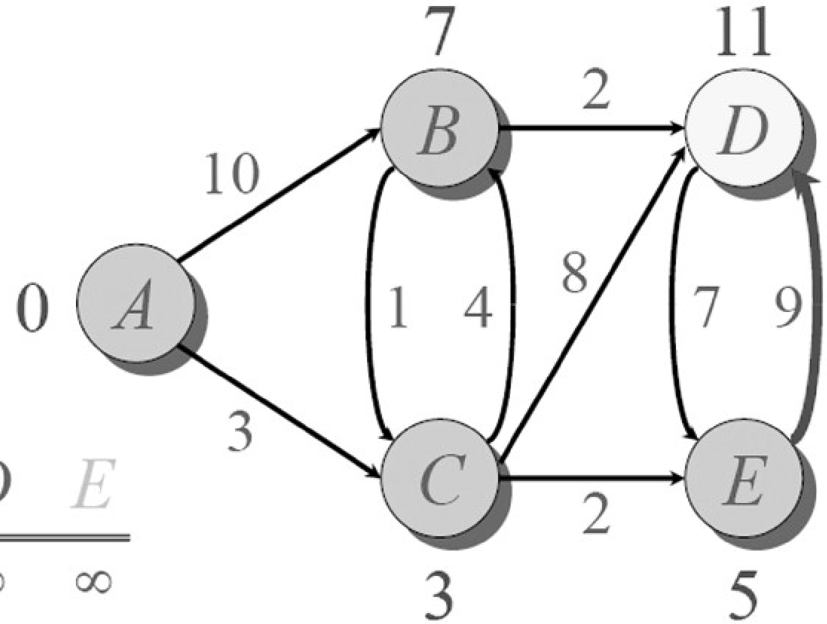
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

Αλγόριθμος Dijkstra (1956, 1959)



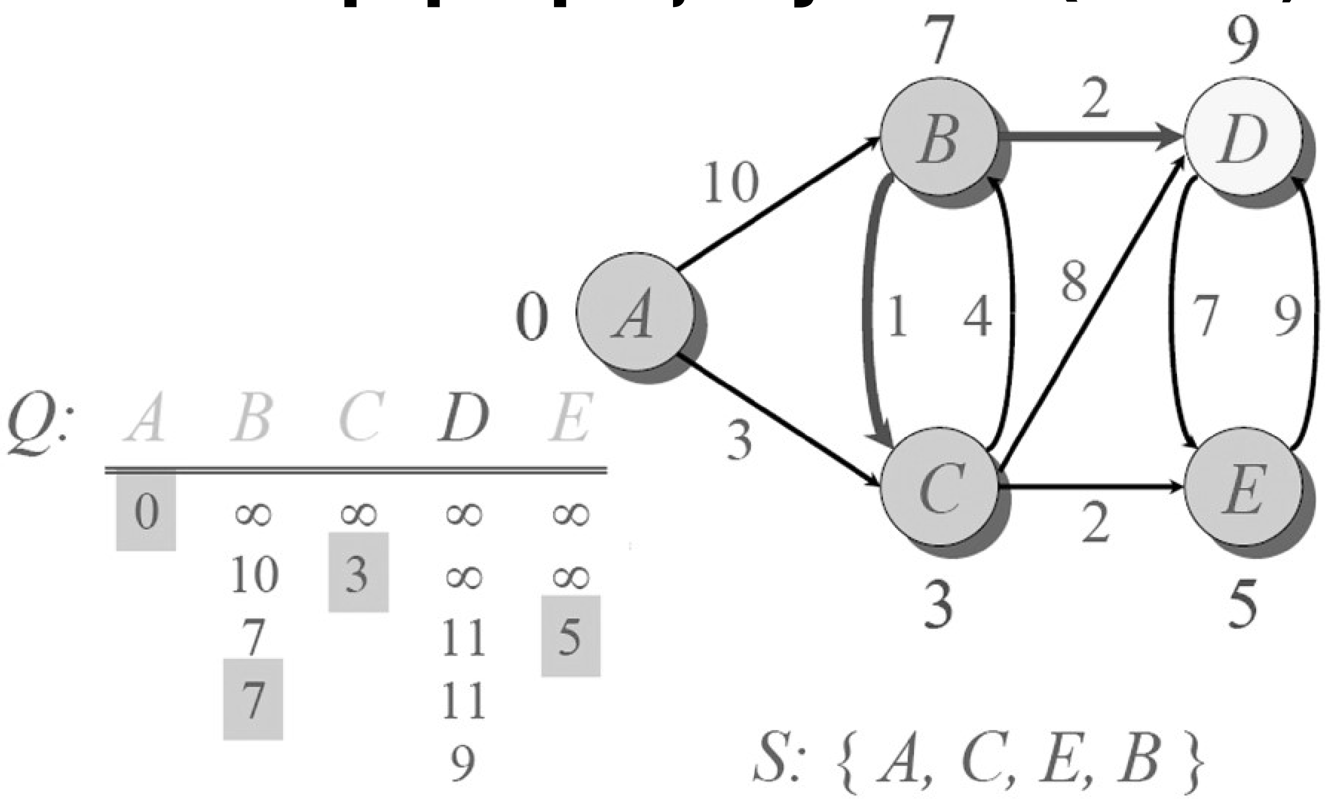
Αλγόριθμος Dijkstra (1956, 1959)



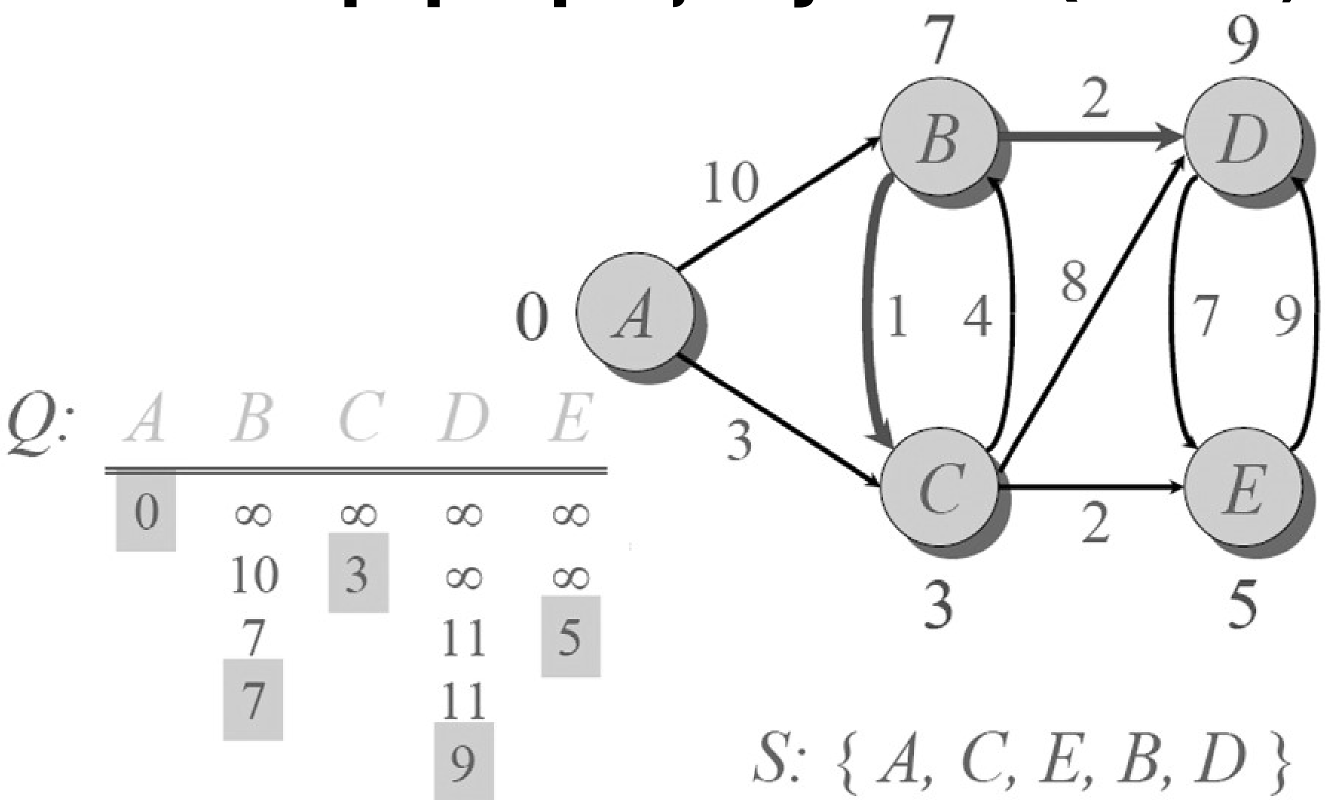
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }

Αλγόριθμος Dijkstra (1956, 1959)



Αλγόριθμος Dijkstra (1956, 1959)



Αλγόριθμος Dijkstra (1956, 1959)

- Η υλοποίηση του Dijkstra χρησιμοποιούσε πίνακες (1956) και συνδεδεμένες λίστες (1959).
- Με σειριακή αναζήτηση σε πίνακες ή λίστες προκύπτει κόστος $O(n^2)$
- Για αραιούς γράφους με χρήση συνδεδεμένων λιστών και
  - δυαδικό σωρό προκύπτει $O((m+n)\log n)$
  - σωρό Fibonacci προκύπτει $O(m+n\log n)$

Αλγόριθμος Bellman-Ford

- **Προσέγγιση**: Δυναμικός προγραμματισμός
- **Είσοδος**: ζυγισμένος γράφος G={E,V} και κορυφή v∈V, με αρνητικά βάρη, χωρίς αρνητικούς κύκλους
- **Έξοδος**: τα συντομότερα μονοπάτια (ή τα ίδια τα συντομότερα μονοπάτια) από την κορυφή v∈V προς όλες τις άλλες κορυφές (1-to-all)
- Ποιό αργός από τον αλγόριθμο Dijkstra αλλά λαμβάνει υπόψιν του αρνητικά βάρη. Το πολύ n - 1 επαναλήψεις – εξετάζει και τις m ακμές σε κάθε επανάληψη
- Ο αλγόριθμος αρχικά προτάθηκε από τον Alfonso Shimbel (1955) αλλά πήρε το όνομα των Richard Bellman και Lester Ford που τον δημοσίευσαν ανεξάρτητα το 1958 και το 1956 αντιστοίχως. Ο Edward Moore επίσης δημοσίευσε τον ίδιο αλγόριθμο το 1957 και για αυτό το λόγο ονομάζεται και αλγόριθμος Bellman-Ford-Moore.

Αλγόριθμος Bellman-Ford

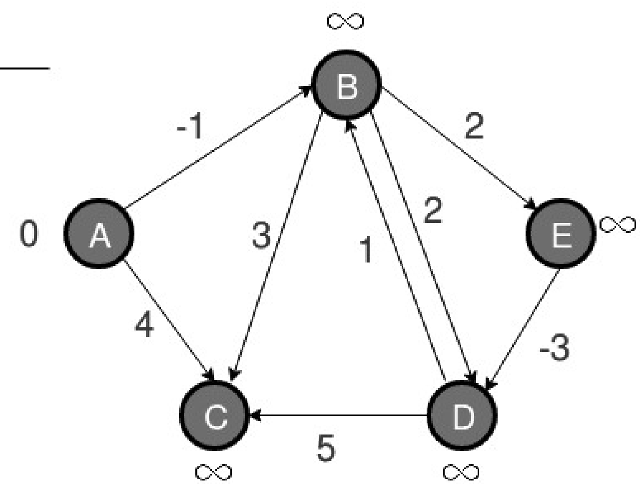
- Σημείωση όλων των κορυφών σαν μη επισκεπτόμενων
- Θέση *προσωρινών* αποστάσεων σε όλες τις κορυφές: 0 στην αρχική κορυφή και άπειρο στις υπόλοιπες
- |V| - 1 επαναλήψεις: Για κάθε ακμή (u,v) υπολογισμός της απόστασης και ενημέρωση της αν μικρότερη από την ήδη υπάρχουσα

Αλγόριθμος Bellman-Ford

```
// Step 1: initialize graph
for each vertex v in vertices:
    distance[v] := inf              // At the beginning , all vertices have a weight of infinity
    predecessor[v] := null          // And a null predecessor

distance[source] := 0               // Except for the Source, where the Weight is zero

// Step 2: relax edges repeatedly
for i from 1 to size(vertices)-1:
    for each edge (u, v) with weight w in edges:
        if distance[u] + w < distance[v]:
            distance[v] := distance[u] + w
            predecessor[v] := u

// Step 3: check for negative-weight cycles
for each edge (u, v) with weight w in edges:
    if distance[u] + w < distance[v]:
        error "Graph contains a negative-weight cycle"
return distance[], predecessor[]
```

Αλγόριθμος Bellman-Ford

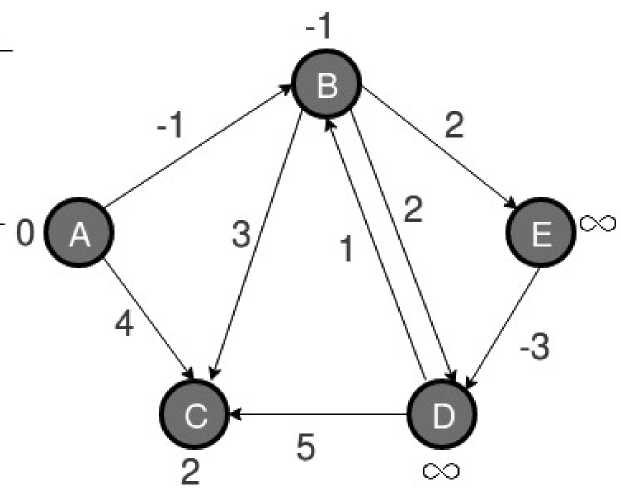
A	B	C	D	E
0	∞	∞	∞	∞



- Έστω ότι οι ακμές υπολογίζονται με τη σειρά: (B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

Αλγόριθμος Bellman-Ford

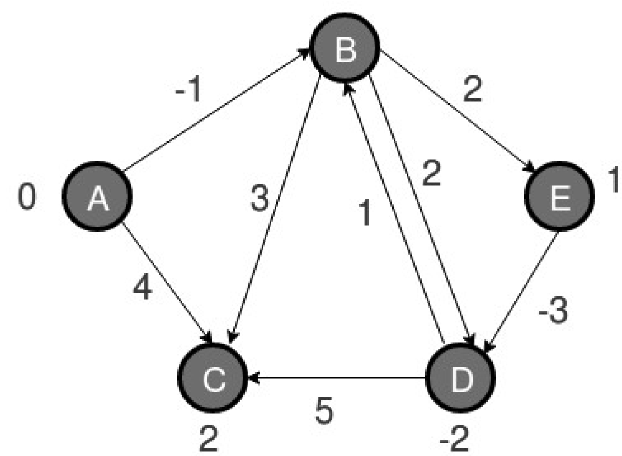
A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞



- Έστω ότι οι ακμές υπολογίζονται με τη σειρά: (B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

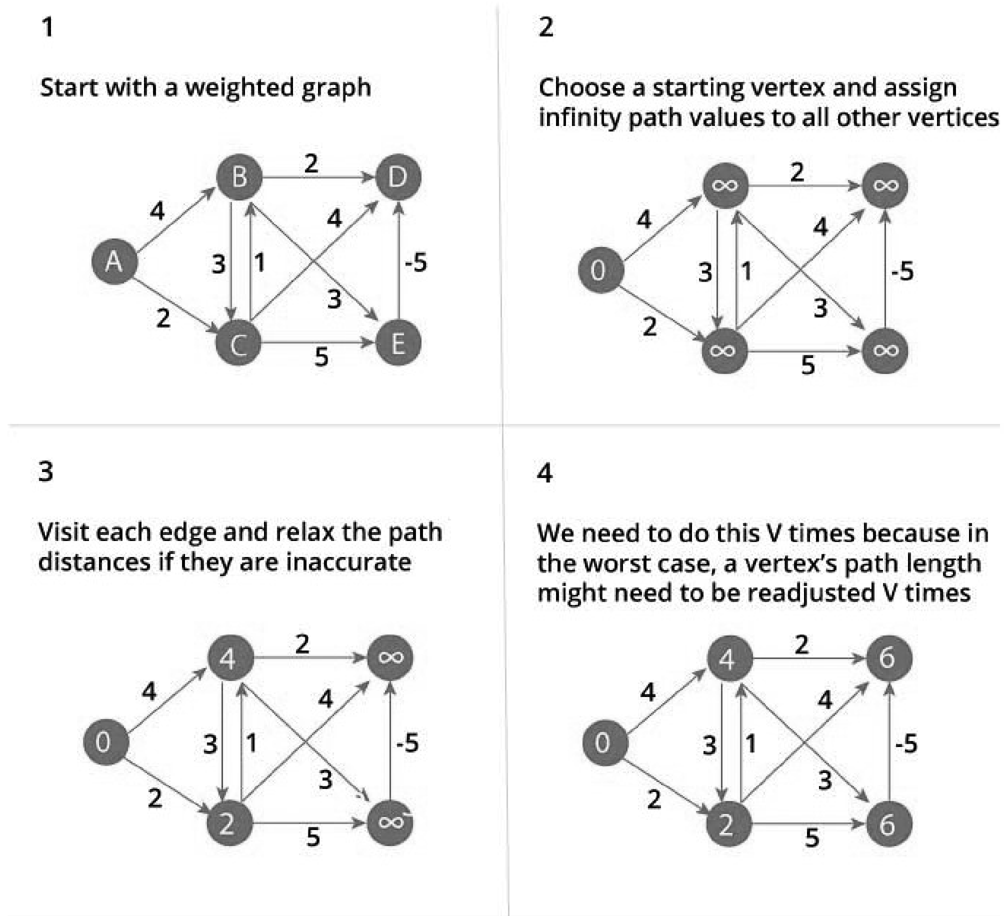
Αλγόριθμος Bellman-Ford

A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1
0	-1	2	1	1
0	-1	2	-2	1



- Έστω ότι οι ακμές υπολογίζονται με τη σειρά: (B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

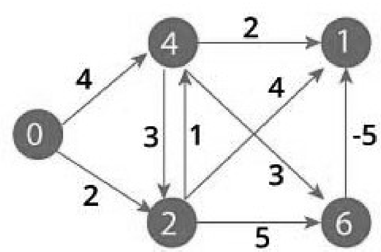
Αλγόριθμος Bellman-Ford



Αλγόριθμος Bellman-Ford

5

Notice how the vertex at the top right corner had its path length adjusted



6

After all the vertices have their path lengths, we check if a negative cycle is present.

A	B	C	D	E
0	∞	∞	∞	∞
0	4	2	∞	∞
0	3	2	6	6
0	3	2	1	6
0	3	2	1	6

Αλγόριθμος Bellman-Ford

- Η πολυπλοκότητα του αλγορίθμου Bellman-Ford είναι O(nm)

```
// Step 1: initialize graph
for each vertex v in vertices:
    distance[v] := inf              // At the beginning , all vertices have a weight of infinity
    predecessor[v] := null          // And a null predecessor

distance[source] := 0               // Except for the Source, where the Weight is zero

// Step 2: relax edges repeatedly
for i from 1 to size(vertices)-1:
    for each edge (u, v) with weight w in edges:
        if distance[u] + w < distance[v]:
            distance[v] := distance[u] + w
            predecessor[v] := u

// Step 3: check for negative-weight cycles
for each edge (u, v) with weight w in edges:
    if distance[u] + w < distance[v]:
        error "Graph contains a negative-weight cycle"
return distance[], predecessor[]
```

O(n)

O(nm)

O(m)

Αλγόριθμος Bellman-Ford

- Να εφαρμοσθεί ο αλγόριθμος Bellman-Ford στο γράφο του σχήματος