ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

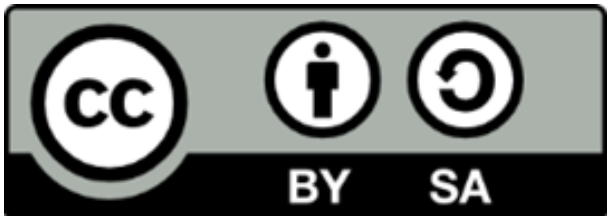
Ενότητα 12: Αλγόριθμοι Γραφημάτων/Συντομότατα μονοπάτια/Αλγόριθμος Bellman-Ford/Αλγόριθμος Dijkstra/Floyd-Warshall

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

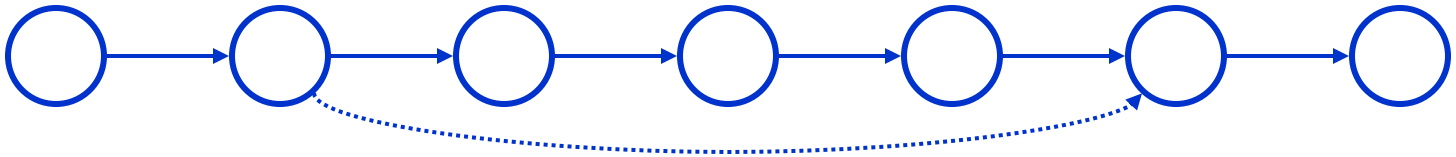
- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση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

Single-Source Shortest Path

- Πρόβλημα: Για δοθέν προσανατολισμένο γράφημα με βάρος G, να βρεθεί το ελάχιστο μονοπάτι από μια δοθείσα αρχική κορυφή (source) s προς μια άλλη κορυφή v
  - "Συντομότατο μονοπάτι" = ελάχιστο βάρος
  - Βάρος μονοπατιού είναι το άθροισμα των (βαρών) των ακμών
  - πχ., δρόμος σε χάρτη: ποιο είναι το συντομότατο μονοπάτι από το Λευκό Πύργο μέχρι τον Άγιο Δημήτριο?

Ιδιότητες συντομότατου μονοπατιού

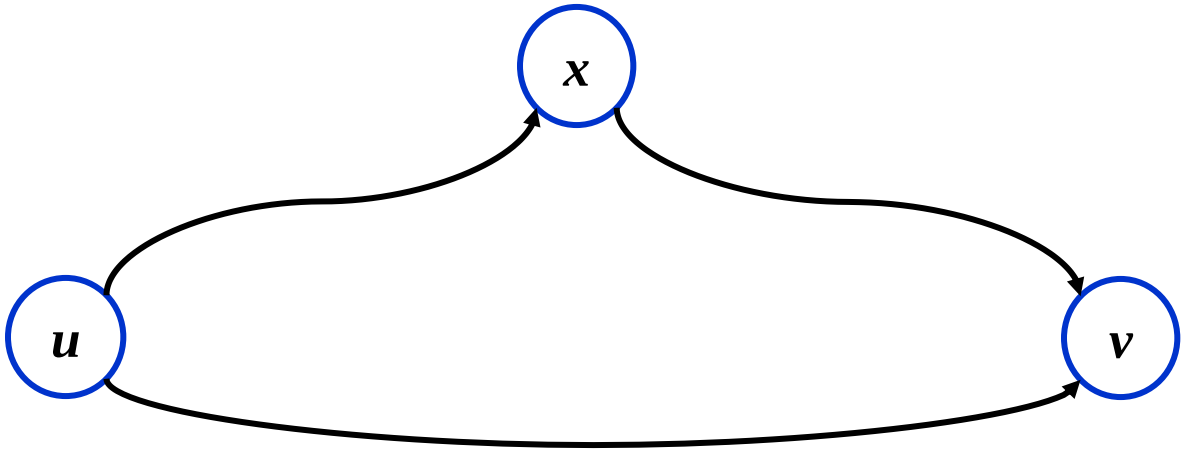
- Εχουμε βέλτιστη υποδομή (*substructure)*: το συντομότατο μονοπάτι αποτελείται από συντομότατα υπο-μονοπάτια



  - Απόδειξη: ας υποθέσουμε ότι κάποιο από τα υπομονοπάτια δεν είναι συντομότατο μονοπάτι
    - Θα πρέπει να υπάρχει ένα συντομότατο υπο-μονοπάτι
    - Θα πρέπει να μπορούμε να αντικαταστήσουμε το συντομότατο υπομονοπάτι με ένα συντομότατο μονοπάτι
    - Αλλά το συνολικό μονοπάτι δεν θα είναι το συντομότατο μονοπάτι Αντίφαση

Ιδιότητες συντομότατου μονοπατιού

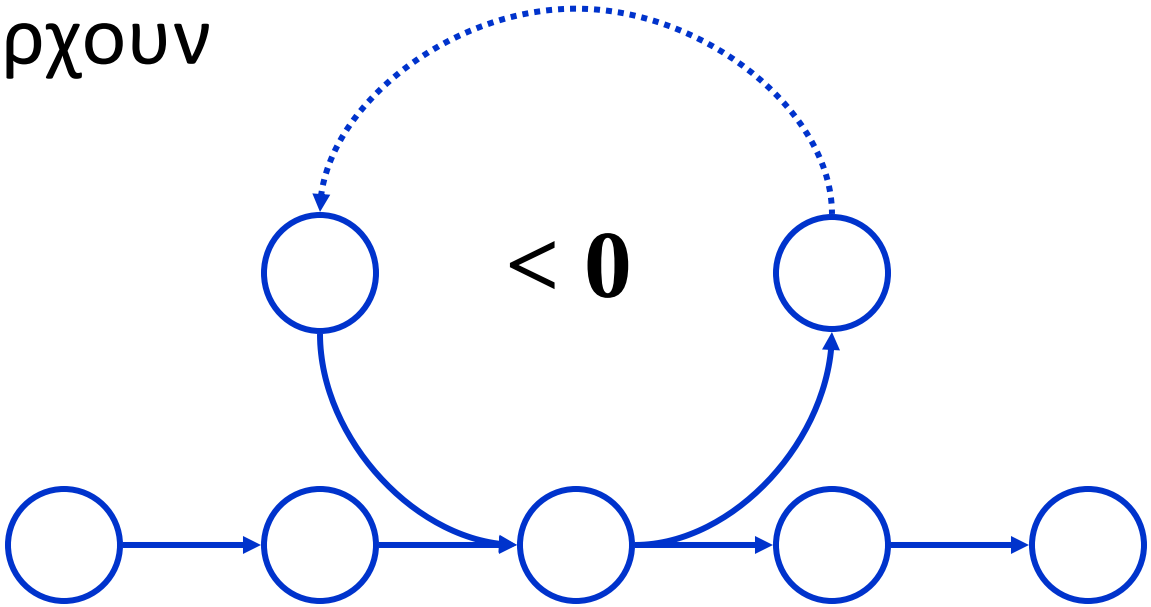
- Ορίζουμε με $\delta(u,v)$ το βάρος του συντομότατου μονοπατιού από την u προς την v
- Τα συντομότατα μονοπάτια ικανοποιούν την *τριγωνική ανισότητα (triangle inequality)*: $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$



Αυτό το μονοπάτι δεν είναι μακρυτερο από οποιοδήποτε άλλο μονοπάτι

Ιδιότητες συντομότατου μονοπατιού

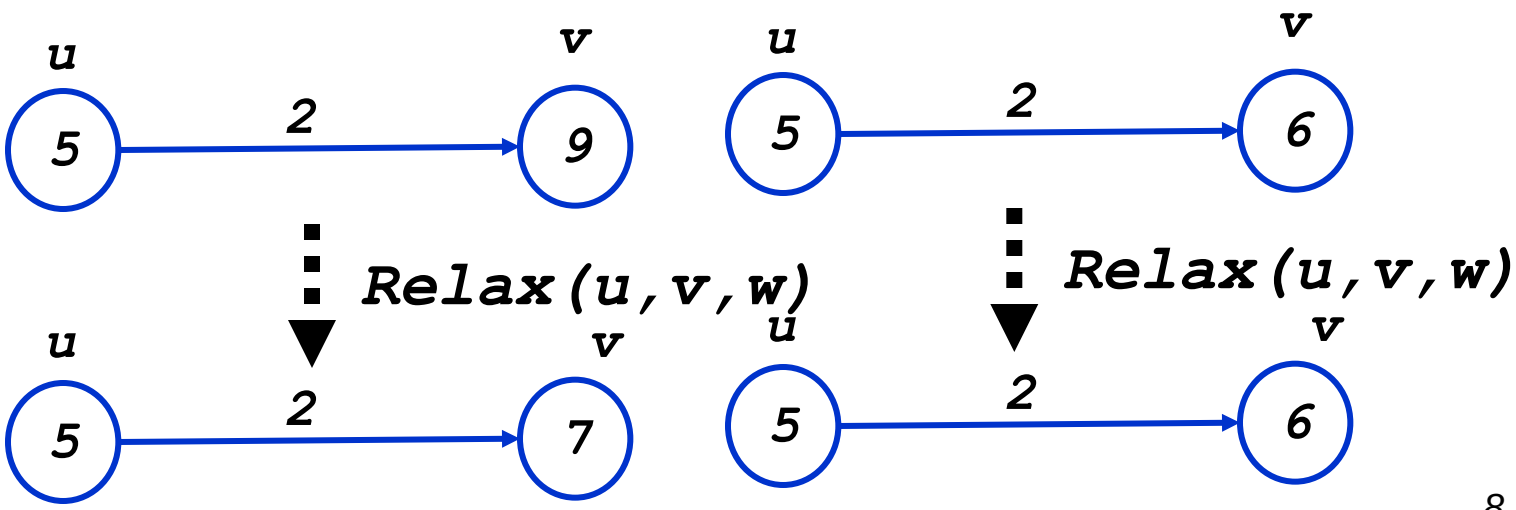
- Σε γραφήματα με κύκλους αρνητικού βάρους μερικά συντομότατα μονοπάτια δεν υπάρχουν



Χαλάρωση

- Η τεχνική σε αλγόριθμους συντομότατου μονοπατιού είναι η τεχνική της *χαλάρωσης* (*relaxation)*
- ιδέα: για όλες τις v, τηρούμε ένα πεδίο d[v] που αντιπροσωπεύει ένα άνω φράγμα του βάρους της συντομότατης διαδρομής δ(s,v) από την αφετηρία s μέχρι τον κόμβο v

Relax(u,v,w) { if (d[v] > d[u]+w) then d[v]=d[u]+w;}



Bellman-Ford Αλγόριθμος

```
BellmanFord()
1.  for each v ∈ V
2.        d[v] = ∞;
3.  d[s] = 0;
4.  for i=1 to |V|-1
5.    for each edge (u,v) ∈ E
6.       Relax(u,v, w(u,v));
7.  for each edge (u,v) ∈ E
8.     if (d[v] > d[u] + w(u,v))
9.        return "no solution";
```

Αρχικοποίηση d[], που θα συγκλίνει προς τη συντομότερη τιμή δ

Χαλάρωση: κάνε |V|-1 περάσματα, «χαλαρώνοντας» κάθε κορυφή

Έλεγχος της λύσης κάτω από ποιες συνθήκες βρίσκουμε τη λύση?

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Bellman-Ford Algorithm

```
BellmanFord()
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0;
    for i=1 to |V|-1
        for each edge (u,v) ∈ E
            Relax(u,v, w(u,v));
    for each edge (u,v) ∈ E
        if (d[v] > d[u] + w(u,v))
            return "no solution";
```

Ποιος είναι ο χρόνος εκτέλεσης?

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Bellman-Ford Ανάλυση

```
BellmanFord()
1.  for each v ∈ V
2.        d[v] = ∞;
3.  d[s] = 0;
4.  for i=1 to |V|-1
5.    for each edge (u,v) ∈ E
6.       Relax(u,v, w(u,v));
7.  for each edge (u,v) ∈ E
8.     if (d[v] > d[u] + w(u,v))
9.        return "no solution";
```

Εντολές 1-3: Αρχικοποίηση Θ(V)

Εντολές 4-6: για κάθε μια από τις |V|-1 διελεύσεις όλων των ακμών: χρόνος Θ(E)

Εντολές 7-9: απαιτεί χρόνο O(E)

Συνολικός χρόνος O(VE)

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Bellman-Ford Algorithm

```
BellmanFord()
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0;
    for i=1 to |V|-1
        for each edge (u,v) ∈ E
            Relax(u,v, w(u,v));
    for each edge (u,v) ∈ E
        if (d[v] > d[u] + w(u,v))
            return "no solution";
```

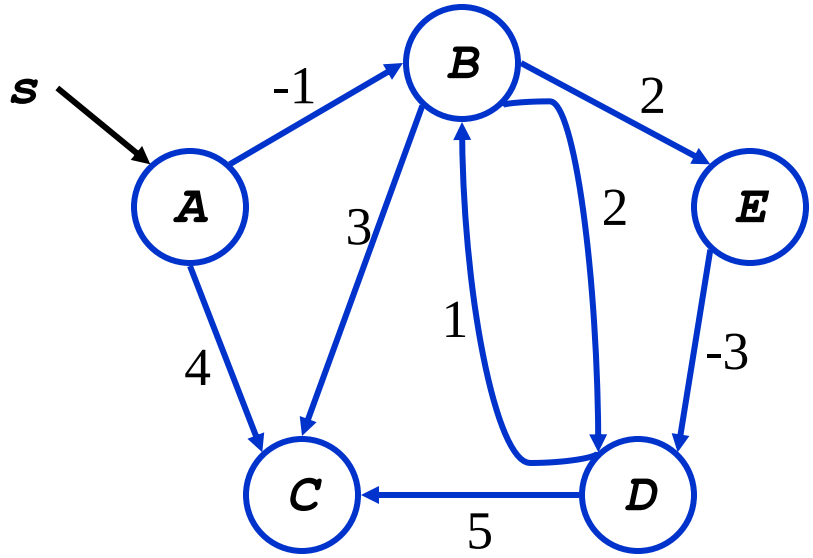
Ποιος είναι ο χρόνος εκτέλεσης?

A: O(VE)

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Bellman-Ford Algorithm

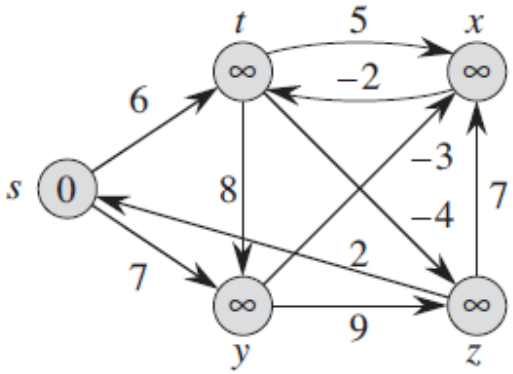
```
BellmanFord()
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0;
    for i=1 to |V|-1
        for each edge (u,v) ∈ E
            Relax(u,v, w(u,v));
    for each edge (u,v) ∈ E
        if (d[v] > d[u] + w(u,v))
            return "no solution";
```



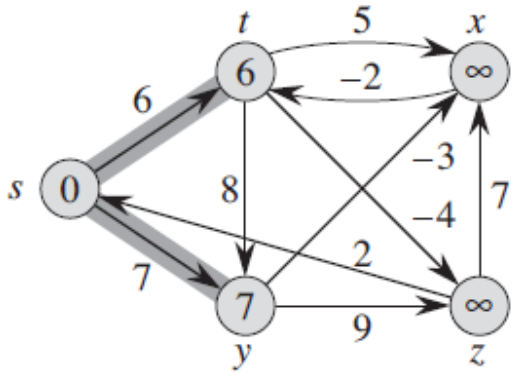
παράδειγμα

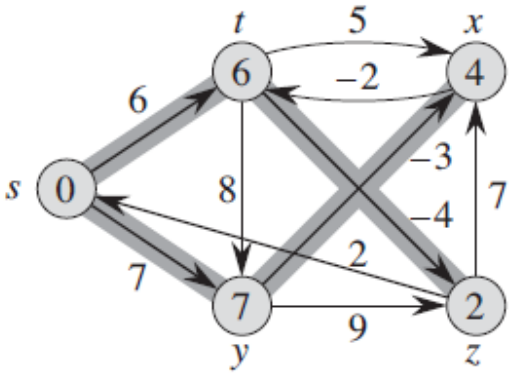
```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

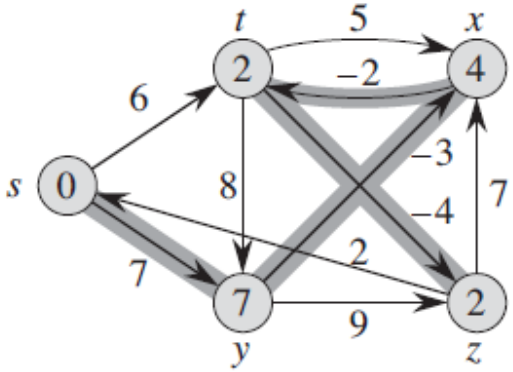
Bellman-Ford Algorithm

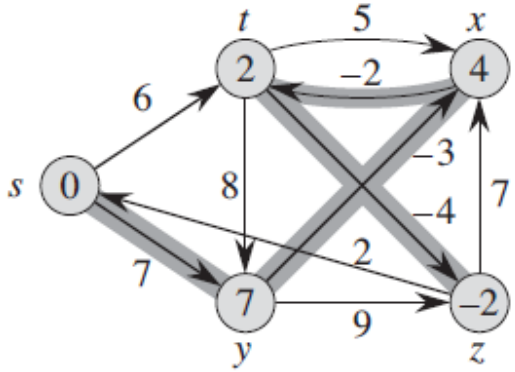


Πριν τη 1η επανάληψη









Η s είναι η κορυφή αφετηρία. Οι σκιασμένες ακμές δηλώνουν προηγούμενες τιμές: Αν η ακμή (u, v) είναι σκιασμένη τότε ο πρόγονος της v είναι η u. Σε κάθε επανάληψη χαλαρώνουν οι ακμές με τη σειρά (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y).

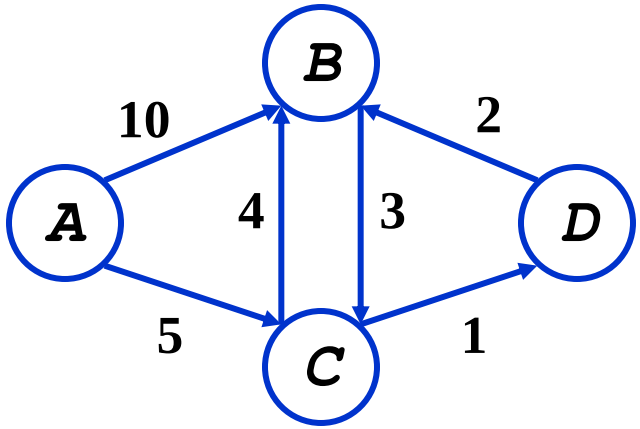
Dijkstra's Algorithm

- Αν δεν υπάρχουν αρνητικά βάρη στις ακμές και με καλή υλοποίηση μπορεί να επιτυχεθεί χρόνος μικρότερος του BF
- Παρόμοιος με τον breadth-first search
  - Αυξάνουμε ένα δένδρο σταδιακά, προωθώντας κορυφές από ουρά
- Επίσης παρόμοιος με τον Prim's για το MST
  - Χρήση ουράς προτεραιότητας ελαχίστου με κλειδιά τις τιμές d[v]

Dijkstra's Algorithm

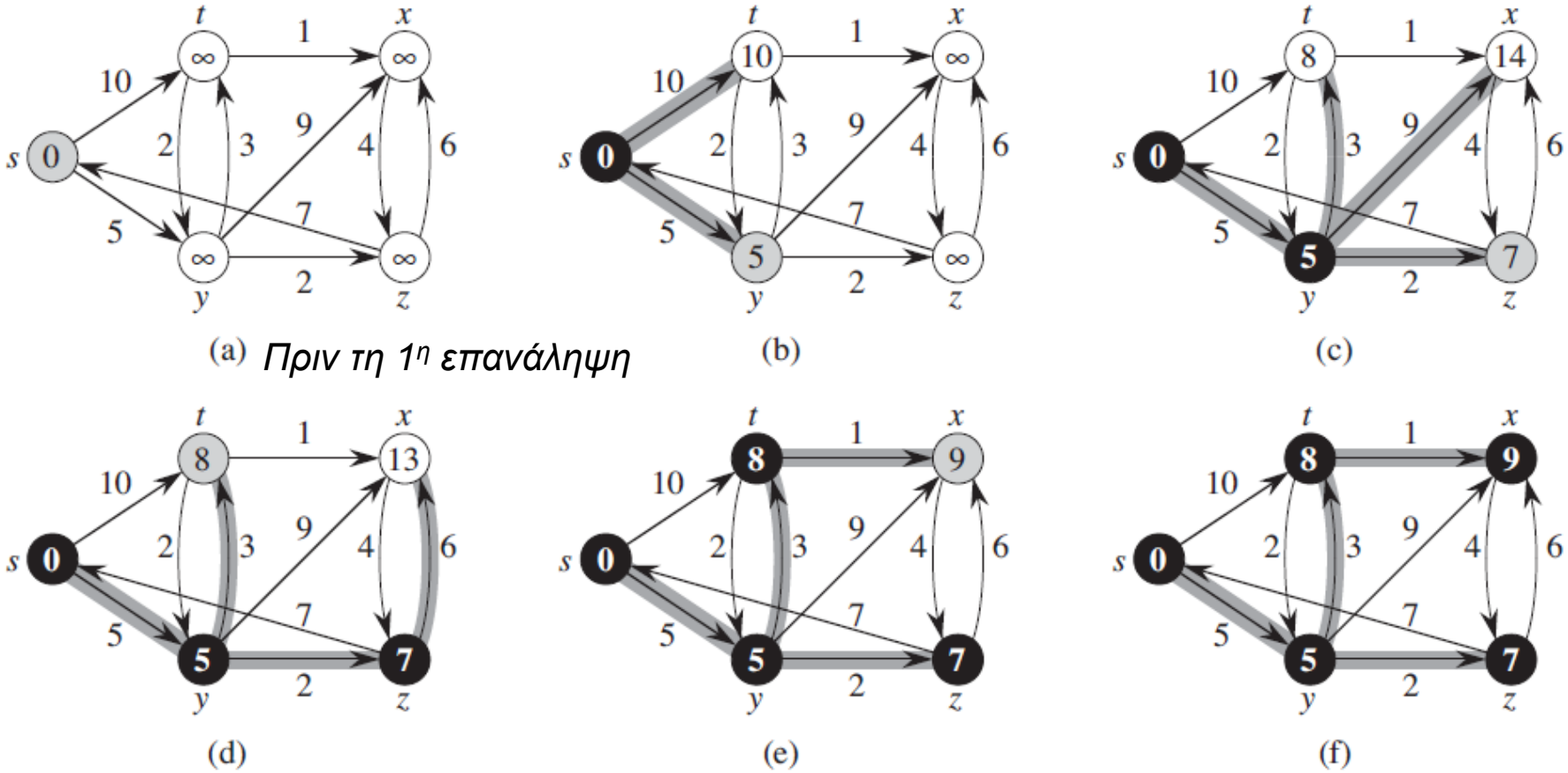
```
Dijkstra(G)
1.     for each v ∈ V
2.        d[v] = ∞;
3.     d[s] = 0; S = ∅; Q = V;
4.     while (Q ≠ ∅)
5.        u = ExtractMin(Q);
6.        S = S U {u};
7.        for each v ∈ u->Adj[]
8.           if (d[v] > d[u]+w(u,v))
9.              d[v] = d[u]+w(u,v);
```

Relaxation Step



Σημείωση:
κλήση της Q->DecreaseKey()

Dijkstra's Algorithm



(a) *Πριν τη 1η επανάληψη* (b) (c)

(d) (e) (f)

Η s είναι η κορυφή αφετηρία. Οι σκιασμένες ακμές δηλώνουν προηγούμενες τιμές: Αν η ακμή (u, v) είναι σκιασμένη τότε ο πρόγονος της v είναι η u. Οι μαύρες κορυφές ανήκουν στο σύνολο S, και άσπρες ανήκουν στη ελάχιστη ουρά προτεραιότητας Q=V-S. Η σκιασμένη κορυφή είναι η u της γραμμής 5. Τα γραφήματα b-f μετά την ολοκλήρωση του while

Dijkstra's Algorithm

```
Dijkstra(G)
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0; S = ∅; Q = V;
    while (Q ≠ ∅)
        u = ExtractMin(Q);
        S = S U {u};
        for each v ∈ u->Adj[]
            if (d[v] > d[u]+w(u,v))
                d[v] = d[u]+w(u,v);
```

Πόσες φορές καλείται η ExtractMin()?

Πόσες φορές καλείται η DecraseKey()?

Ποιος είναι ο συνολικός χρόνος?

Dijkstra's Algorithm

```
Dijkstra(G)
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0; S = ∅; Q = V;
    while (Q ≠ ∅)
        u = ExtractMin(Q);
        S = S U {u};
        for each v ∈ u->Adj[]
            if (d[v] > d[u]+w(u,v))
                d[v] = d[u]+w(u,v);
```

Πόσες φορές καλείται η ExtractMin()?

Πόσες φορές καλείται η DecraseKey()?

Α: O(E lg V) χρησιμοποιώντας binary heap για Q
Μπορεί να επιτευχθεί O(V lg V + E) με Fibonacci heaps

Ανάλυση του αλγόριθμου του Dijkstra -1-

- Ο αλγόριθμος διατηρεί μια ουρά προτεραιότητας ελαχίστου καλώντας 3 πράξεις ουράς προτεραιότητας: InsertMin (εμπεριέχεται στη γραμ. 3) ExtractMin(Q) γραμ.5, DecraseKey (εμπεριέχεται στη ΧΑΛΑΡΩΣΗ γραμ. 9). Η πράξη της InsertMin εκτελείται μια φορά για κάθε κορυφή όπως και η ExtractMin. Καθώς κάθε κορυφή προστίθεται στο σύνολο S μια μόνο φορά, κάθε ακμή από τη λίστα γειτνίασης Adj[] εξετάζεται μέσα στο βρόχο ακριβώς μια φορά. Το συνολικό πλήθος των ακμών είναι |E| έχουν |E επαναλήψεις του βρόχου και άρα |E| το πολύ πράξεις ExtractMin.
- Ο χρόνος εκτέλεσης εξαρτάται από την υλοποίηση της ουράς προτεραιότητας ελαχίστου.

Ανάλυση του αλγόριθμου του Dijkstra -2-

- Αν θεωρήσουμε ότι η ουρά τηρείται με βάση την αρίθμηση των κόμβων, από 1 έως |V| οπότε καταχωρίζουμε την τιμή d[v] στο v-ιστό στοιχείο του πίνακα. Κάθε πράξη εισαγωγής και μείωσης κλειδιού απαιτεί χρόνο O(1) ενώ κάθε πράξη εξαγωγής ελαχίστου απαιτεί O(V) (αφού θα πρέπει να εξεταστεί όλος ο πίνακας).
- Αρα ο συνολικός χρόνος είναι $O(V^2 + E)= O(V^2)$

Ανάλυση του αλγόριθμου του Dijkstra -3-

- Αν το γράφημα είναι αραιό είναι προτιμότερο να υλοποιήσουμε την ουρά προτεραιότητας ελαχίστου μέσω ενός δυαδικού σωρού ελαχίστου. Στην περίπτωση αυτή κάθε πράξη εισαγωγής απαιτεί χρόνο O(lgV), και υπάρχουν |V| πράξεις. Ο χρόνος που απαιτείται για την κατασκευή του δυαδικού σωρού ελαχίστου είναι O(V). Κάθε πράξη μείωσης κλειδιού απαιτεί χρόνο O(lgV) και υπάρχουν |E| τέτοιες πράξεις.
- Αρα ο συνολικός χρόνος είναι O(V+E)lgV ή O(ElgV) αν όλοι οι κόμβοι είναι προσπελάσιμοι από την αφετηρία.
- Αυτός ο χρόνος αποτελεί βελτίωση σε σχέση με την απλή υλοποίηση που απαιτεί χρόνο $O(V^2)$

Ανάλυση του αλγόριθμου του Dijkstra -4-

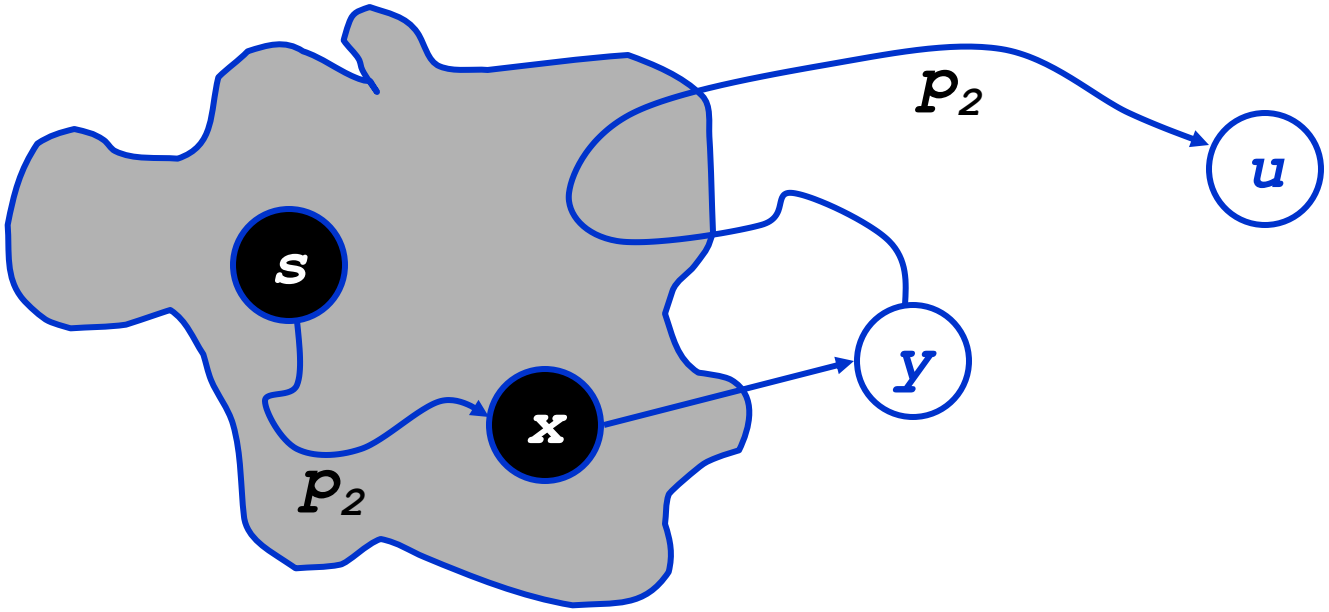
- Μπορούμε να πετύχουμε χρόνο O(VlgV+E) αν υλοποιήσουμε την ουρά προτεραιότητας ελαχίστου με ένα σωρό Fibonacci.
- Το λογιστικό κόστος κάθε μιας από της |V| πράξεις εξαγωγής ελαχίστου είναι O(lgV) ενώ κάθε κλήση της μείωσης κλειδιού (το συνολικό πλήθος των οποίων είναι |E|) απαιτεί χρόνο μόλις O(1).
- Το αρχικό κίνητρο για την ανάπτυξη των σωρών Fibonacci ήταν ακριβώς ο αλγόριθμος του Dijkstra όπου παρατήρησαν ότι κατά κανόνα ήταν πολύ περισσότερες οι κλήσεις μείωσης κλειδιού από της εξαγωγής ελαχίστου οπότε οποιαδήποτε μέθοδος μειώνει το χρόνο της μείωσης κλειδιού χωρίς να αυξάνει της εξαγωγής κλειδιού δίνει ασυμπτωτικά ταχύτερη υλοποίηση από το δυαδικό σωρό.

Dijkstra's Algorithm

```
Dijkstra(G)
1.     for each v ∈ V
2.         d[v] = ∞;
3.     d[s] = 0; S = ∅; Q = V;
4.     while (Q ≠ ∅)
5.         u = ExtractMin(Q);
6.         S = S U{u};
7.         for each v ∈ u->Adj[]
8.             if (d[v] > d[u]+w(u,v))
                   d[v] = d[u]+w(u,v);
```

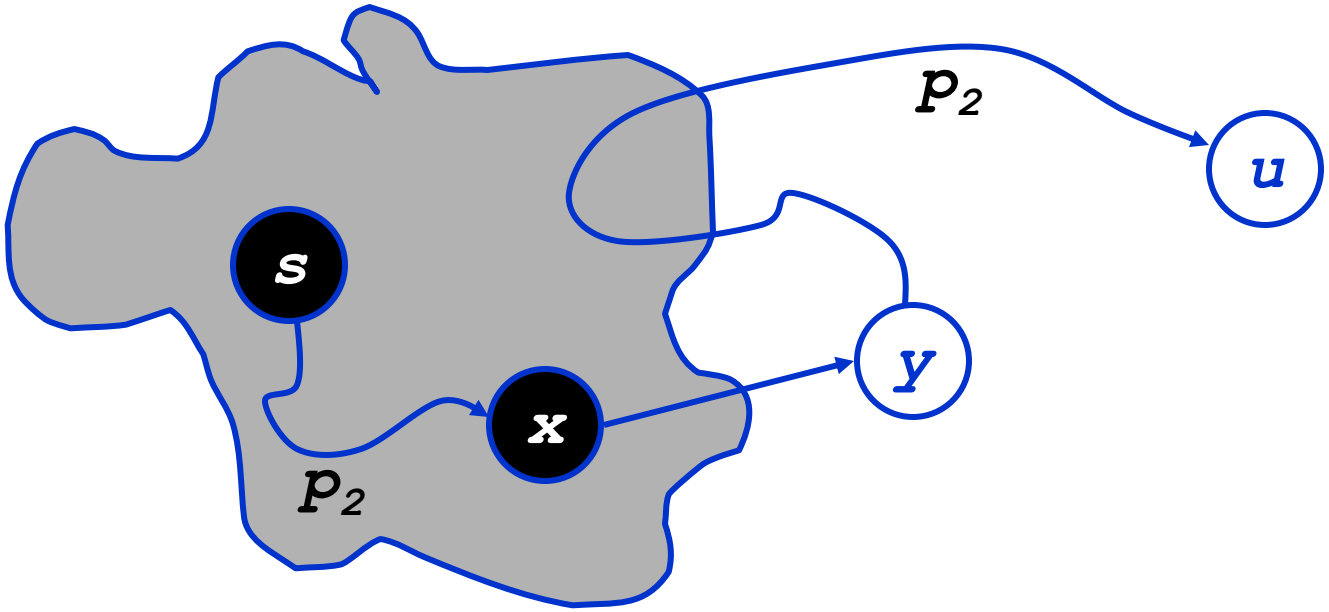
Correctness: we must show that when u is removed from Q, it has already converged

Correctness Of Dijkstra's Algorithm



- Note that $d[v] \geq \delta(s,v)\ \forall v$
- Let u be first vertex picked s.t. $\exists$ shorter path than d[u] $\Rightarrow d[u] > \delta(s,u)$
- Let y be first vertex $\in$ V-S on actual shortest path from s$\rightarrow$u $\Rightarrow d[y] = \delta(s,y)$
  - Because d[x] is set correctly for y's predecessor x $\in$ S on the shortest path, and
  - When we put x into S, we relaxed (x,y), giving d[y] the correct value

Correctness Of Dijkstra's Algorithm



- Note that $d[v] \geq \delta(s,v)\ \forall v$
- Let u be first vertex picked s.t. $\exists$ shorter path than d[u] $\Rightarrow d[u] > \delta(s,u)$
- Let y be first vertex $\in$ V-S on actual shortest path from s→u $\Rightarrow d[y] = \delta(s,y)$
- $d[u] > \delta(s,u)$
  $= \delta(s,y) + \delta(y,u)$ (*Why?*)
  $= d[y] + \delta(y,u)$
  $\geq d[y]$ But if d[u] > d[y], wouldn't have chosen u. Contradiction.

Ομοιότητα αλγόριθμου Dijkstra με BFS & Prim

- Ο αλγόριθμος του Dijkstra έχει κάποια ομοιότητα με τον αλγόριθμό BFS και Prim.
- Η ομοιότητα με τον BFS έγκειται στην αντιστοιχία του συνόλου S με το σύνολο των μαύρων κορυφών σε μια BFS διερεύνηση όπως τα βάρη των συντομότερων διαδρομών έχουν πάρει τις τελικές τιμές τους έτσι και στις οριζόντιες αποστάσεις των μαύρων κορυφών σε μια BFS διερεύνηση έχουν τις σωστές τιμές.

Dijkstra - Prim

```
Dijkstra(G)
for each v ∈ V
   d[v] = ∞;
d[s] = 0; S = ∅; Q = V;

while (Q ≠ ∅)
   u = ExtractMin(Q);
   S = S U {u};
   for each v ∈ u->Adj[]
      if (d[v] >
       d[u]+w(u,v))
       d[v] = d[u]+w(u,v);
```

```
MST-Prim(G, w, r)
Q = V[G];
for each u ∈ Q
   key[u] = ∞;
key[r] = 0;p[r] = NULL;
while (Q not empty)
   u = ExtractMin(Q);
   for each v ∈ Adj[u]
      if (v ∈ Q and w(u,v)
            < key[v])
         p[v] = u;
         DecreaseKey(v,
            w(u,v));
```

DAG Shortest Paths

- Πρόβλημα: εύρεση συντομότατων μονοπατιών σε DAG
  - Ο αλγόριθμος Bellman-Ford απαιτεί O(VE) χρόνο.
  - *Μπορούμε να πετύχουμε καλύτερο χρόνο?*

DAG Shortest Paths

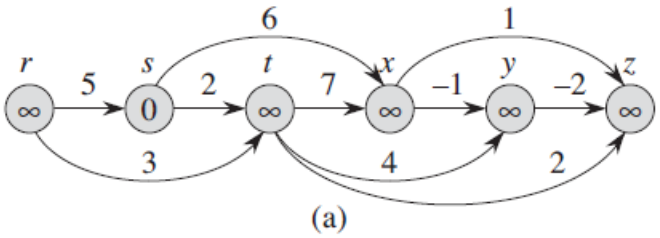
- ιδέα: topological sort
  - Αν είμαστε τυχεροί και επεξεργαστούμε τις κορυφές στο συντομότατο μονοπάτι από αριστερά προς τα δεξιά, τότε αυτό μπορεί να γίνει με ένα πέρασμα.
  - Κάθε μονοπάτι σε ένα dag είναι μια υπο-ακολουθία από τοπολογικά διατεταγμένες κορυφές, έτσι η επεξεργασία των κορυφών μ' αυτή τη σειρά, θα δημιουργήσει κάθε μονοπάτι με την προς τα μπρος διάσχιση (δε θα χαλαρώσουμε ποτέ ακμές εξερχόμενες από κορυφές πριν ολοκληρώσουμε με όλες τις ακμές που συντρέχουν στις κορυφές).
  - Ετσι χρειάζεται μόνο ένα πέρασμα: *Ποιος θα είναι ο χρόνος εκτέλεσης?*

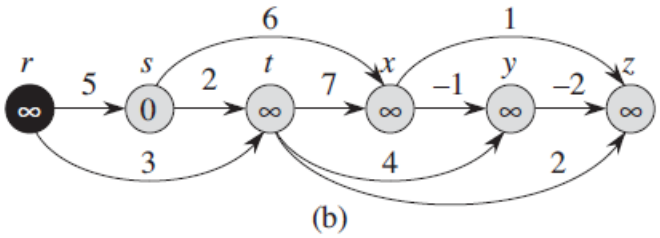
DAG Shortest Paths Algorithm

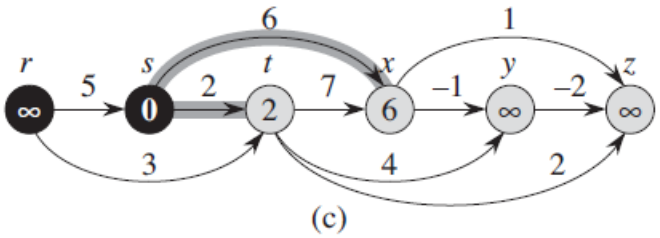
```
DAG_shortest_path()
   1. Topological_Sort();
   2. for each v ∈ V
            d[v] = ∞;
       d[s] = 0;
   3. for each vertex u, taken in
          topologically sorted order
   4.     for each vertex v ∈ Adj(u)
   5.         RELAX(u, v, w);
```

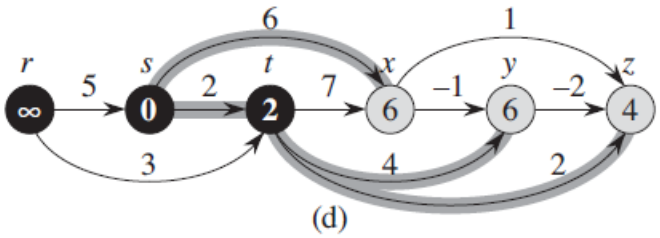
```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

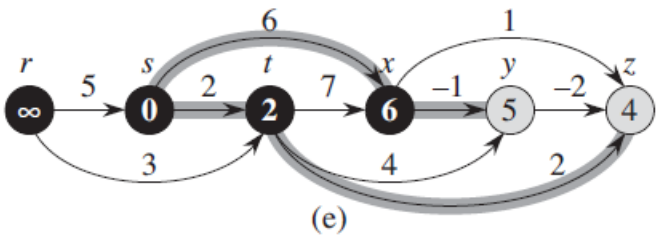
DAG Shortest Paths Algorithm

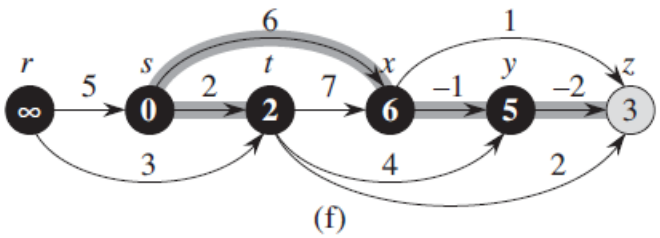


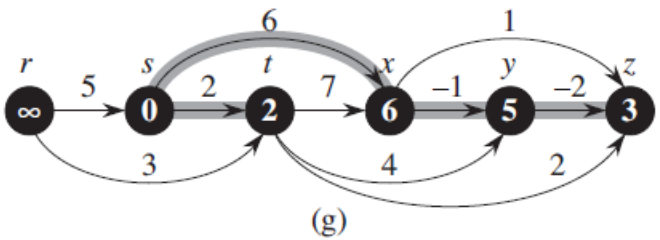












(a) Από την 1η επανάληψη του βρόχου γραμμές 3-5 (b)-(g) η κατάσταση μετά από κάθε επανάληψη του βρόχου για τις γραμμές 3-5. Ο καινούργιος μαύρος κόμβος σε κάθε επανάληψη είναι ο αντίστοιχος κόμβος u της επανάληψης αυτής. Οι τιμές στο σχήμα (g) είναι οι τελικές.

Αλγόριθμος Floyd-Warshall

- Θεωρούμε γράφημα G(V, E, w) με βάρη στις ακμές.
- Καθορισμένη (αυθαίρετη) αρίθμηση κορυφών v_1, v_2, …, v_n.
- Αναπαράσταση γραφήματος με πίνακα γειτνίασης:

$$w(v_i, v_j) = \begin{cases} 0 & v_i = v_j \\ w(v_i, v_j) & v_i \neq v_j \ \ (v_i, v_j) \in E \\ \infty & v_i \neq v_j \ \ (v_i, v_j) \notin E \end{cases}$$

- Υπολογισμός απόστασης $d(v_i, v_j)$ από $d(v_i, v_k)$, $d(v_k, v_j)$ για όλα τα $k \in V \setminus \{v_i, v_j\}$:

$$d(v_i, v_j) = \min\{w(v_i, v_j), \min_{v_k \in V \setminus \{v_i, v_j\}} \{d(v_i, v_k) + d(v_k, v_j)\}\}$$

- Φαύλος κύκλος(;): $d(v_i, v_k) \rightarrow d(v_i, v_j)$ και $d(v_i, v_j) \rightarrow d(v_i, v_k)$

Αλγόριθμος Floyd-Warshall

- $D_k[v_i, v_j]$: μήκος συντομότερου v_i - v_j μονοπατιού με ενδιάμεσες κορυφές μόνο από $V_k = \{v_1, \ldots, v_k\}$
- Αρχικά $D_0[v_i, v_j] = w(v_i, v_j)$ γιατί $V_0 = \emptyset$.
- Έστω ότι γνωρίζουμε $D_{k-1}[v_i, v_j]$ για όλα τα ζεύγη v_i, v_j
- $D_k[v_i, v_j]$ διέρχεται από v_k καμία ή μία φορά μονοπάτι!):
- Αναδρομική σχέση για D_0, D_1, ..., D_n :

$$D_k[v_i, v_j] = \begin{cases} w(v_i, v_j) & k = 0 \\ \min\{D_{k-1}[v_i, v_j], D_{k-1}[v_i, v_k] + D_{k-1}[v_k, v_j]\} & k = 1, \ldots, n \end{cases}$$

- Υπολογισμός D_n με **δυναμικό προγραμματισμό**.
- Κύκλος αρνητικού μήκους αν $D_n[v_i, v_i] < 0$.

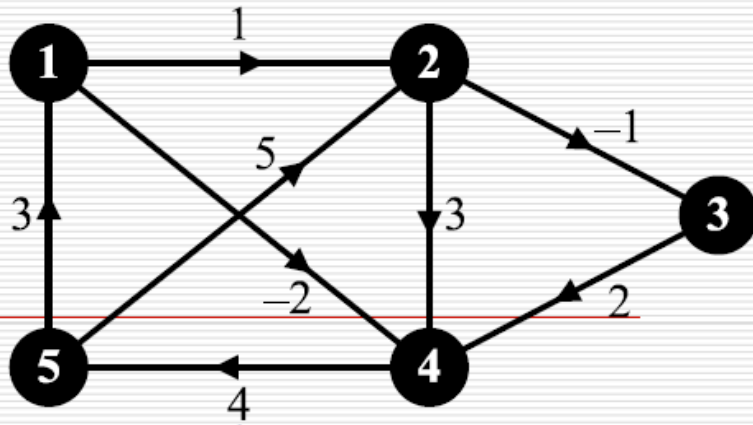
Αλγόριθμος Floyd-Warshall

- Τυπικός δυναμικός προγραμματισμός:

```
Floyd-Warshall(G(V, E, w))
    for i ← 1 to n do
        for j ← 1 to n do
            if (v_i, v_j) ∈ E then D_0[i, j] ← w(v_i, v_j);
            else D_0[i, j] ← ∞;
        D_0[i, i] ← 0;
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                if D_{k-1}[i, j] > D_{k-1}[i, k] + D_{k-1}[k, j] then
                    D_k[i, j] ← D_{k-1}[i, k] + D_{k-1}[k, j];
                else D_k[i, j] ← D_{k-1}[i, j];
```

$\Theta(n^3)$

Παράδειγμα



$$D_0 = \begin{pmatrix} 0 & 1 & \infty & -2 & \infty \\ \infty & 0 & -1 & 3 & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & 0 & 4 \\ 3 & 5 & \infty & \infty & 0 \end{pmatrix}$$

$$D_1 = \begin{pmatrix} 0 & 1 & \infty & -2 & \infty \\ \infty & 0 & -1 & 3 & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & 0 & 4 \\ 3 & 4 & \infty & 1 & 0 \end{pmatrix}$$

$$D_2 = \begin{pmatrix} 0 & 1 & 0 & -2 & \infty \\ \infty & 0 & -1 & 3 & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & 0 & 4 \\ 3 & 4 & 3 & 1 & 0 \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0 & 1 & 0 & -2 & \infty \\ \infty & 0 & -1 & 1 & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & 0 & 4 \\ 3 & 4 & 3 & 1 & 0 \end{pmatrix}$$

$$D_4 = \begin{pmatrix} 0 & 1 & 0 & -2 & 2 \\ \infty & 0 & -1 & 1 & 5 \\ \infty & \infty & 0 & 2 & 6 \\ \infty & \infty & \infty & 0 & 4 \\ 3 & 4 & 3 & 1 & 0 \end{pmatrix}$$

$$D_5 = \begin{pmatrix} 0 & 1 & 0 & -2 & 2 \\ 8 & 0 & -1 & 1 & 5 \\ 9 & 10 & 0 & 2 & 6 \\ 7 & 8 & 7 & 0 & 4 \\ 3 & 4 & 3 & 1 & 0 \end{pmatrix}$$

Υπολογισμός Συντομότατων Μονοπατιών

- $P_k[v_i, \cdot\,]$: ΔΣΜ(v_i) με ενδιάμεσες κορυφές μόνο από V_k .
- Αποστάσεις $D_k[v_i, \cdot\,]$ αντιστοιχούν σε μονοπάτια $P_k[v_i, \cdot\,]$.
- $P_k[v_i, v_j]$: προηγούμενη κορυφή της v_j στο συντομότατο $v_i - v_j$ μονοπάτι με ενδιάμεσες κορυφές μόνο από V_k .
- P_0 καθορίζεται από πίνακα γειτνίασης:

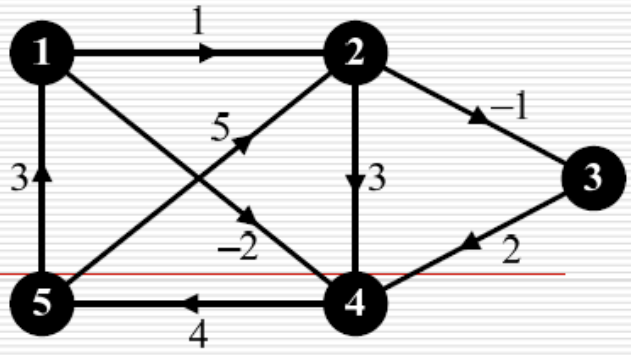
$$P_0[v_i, v_j] = \begin{cases} \text{NULL} & \text{αν } i = j \text{ ή } (v_i, v_j) \notin E \\ v_i & \text{διαφορετικά} \end{cases}$$

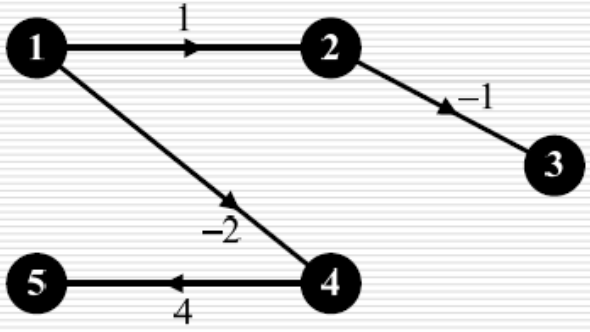
- Αναδρομική σχέση για $P_0, P_1, \ldots, P_n$:

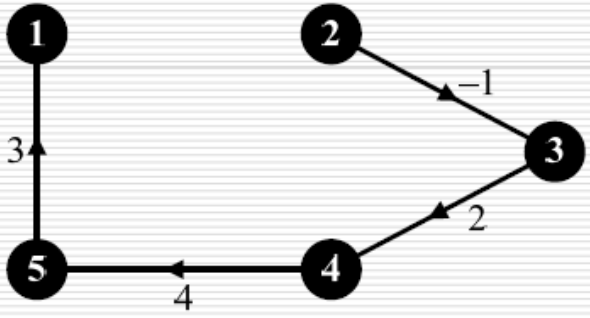
$$P_k[v_i, v_j] = \begin{cases} P_{k-1}[v_i, v_j] & D_{k-1}[v_i, v_j] \le D_{k-1}[v_i, v_k] + D_{k-1}[v_k, v_j] \\ P_{k-1}[v_k, v_j] & D_{k-1}[v_i, v_j] > D_{k-1}[v_i, v_k] + D_{k-1}[v_k, v_j] \end{cases}$$

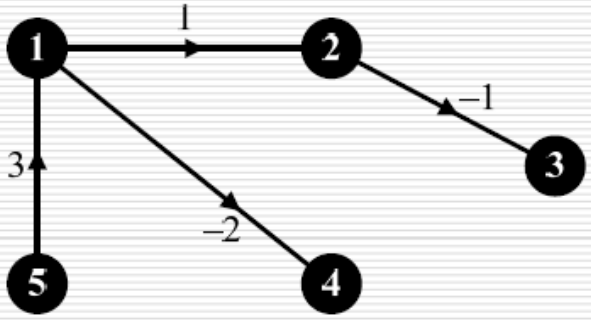
- Υπολογισμός P_n ταυτόχρονα με υπολογισμό D_n .
- Εύκολη τροποποίηση προηγούμενης υλοποίησης.

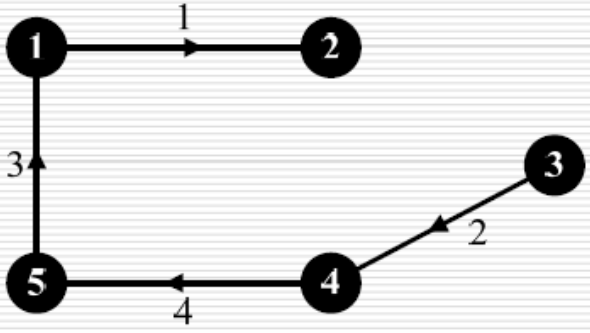
Παράδειγμα

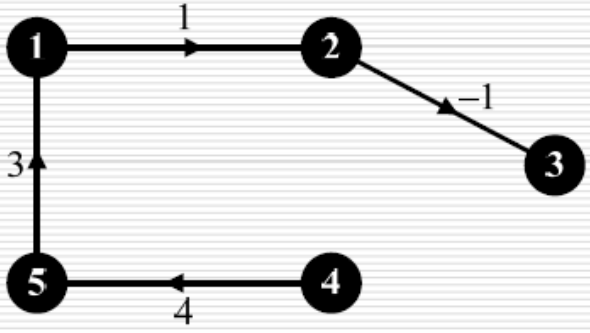












$$P_5 = \begin{pmatrix} \text{NULL} & 1 & 2 & 1 & 4 \\ 5 & \text{NULL} & 2 & 3 & 4 \\ 5 & 1 & \text{NULL} & 3 & 4 \\ 5 & 1 & 2 & \text{NULL} & 4 \\ 5 & 1 & 2 & 1 & \text{NULL} \end{pmatrix}$$

Τέλος Ενότητας