ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Τεχνολογίες Υλοποίησης Αλγορίθμων

Χρήστος Ζαρολιάγκης

Καθηγητής
Τμήμα Μηχ/κων Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών
email: zaro@ceid.upatras.gr

Ενότητα 6

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
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 Πατρών»** 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.

Περιεχόμενα

- Συντομότερες Διαδρομές – Γενικά
- Αλγόριθμοι Συντομότερων Διαδρομών και Υλοποιήσεις
  - Κατευθυνόμενα Ακυκλικά Γραφήματα
  - Μη αρνητικά κόστη ακμών
  - Αλγόριθμος Dijkstra: υλοποίηση & αξιολόγηση

Πρόβλημα Συντομότερης Διαδρομής

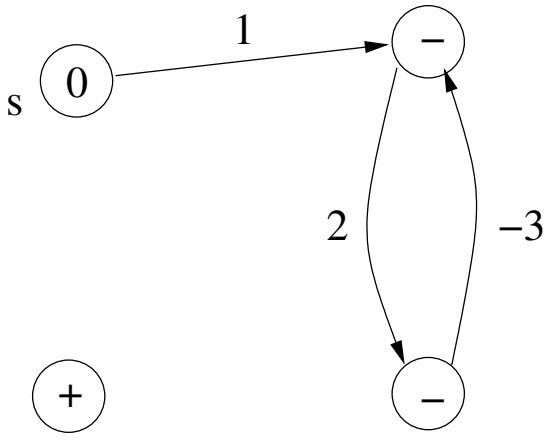
- Κατευθυνόμενο γράφημα $G = (V, E)$ με συνάρτηση κόστους $c : E \to \mathbb{R}$
- Κόστος $c(p)$ διάδρομής $p = [v_1, v_2, \ldots, v_k]$

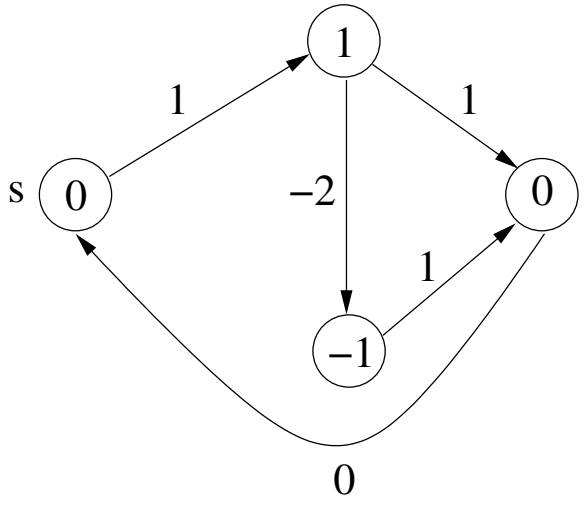
$$c(p) = \sum_{i=1}^{k-1} c(v_i, v_{i+1})$$

- Δεδομένων δύο κορυφών v και w, να βρεθεί η διαδρομή από την v στην w με το *ελάχιστο κόστος* ή *απόσταση*, που συμβολίζεται ως $\delta(v, w)$

Πρόβλημα Συντομότερης Διαδρομής

- $$\delta(v,w) = \begin{cases} \inf\{c(p); p \text{ είναι μια } v\text{-}w \text{ διαδρομή}\} \\ +\infty, \quad \text{αν } \nexists\ v\text{-}w \text{ διαδρομή} \end{cases}$$





- *Αρνητικός κύκλος* $\mathcal{C}$: $c(\mathcal{C}) < 0$

Ιδιότητες Συντομότερων Διαδρομών

1. $\exists$ μια v-w διαδρομή που περιέχει αρνητικό κύκλο *αvv* $\delta(v, w) = -\infty$

2. Αν $\exists$ v-w διαδρομή και $\nexists$ v-w διαδρομή που να περιέχει αρνητικό κύκλο, τότε $-\infty < \delta(v, w) < +\infty$, και $\delta(v, w)$ είναι το κόστος μιας απλής v-w διαδρομής

Εύρεση ΣΔ από μια Αρχική Κορυφή (ΕΣΔΑΚ)

- Πρόβλημα ΕΣΔΑΚ:

  δεδομένου ενός κατευθυνόμενου γραφήματος $G = (V, E)$ και μιας αρχικής κορυφής $s \in V$, να υπολογισθούν τα $\delta(s, v), \forall v \in V$, καθώς και οι συντομότερες s-v διαδομές

- Απλοποίηση συμβολισμού: $\delta(s, v) \equiv \delta(v), \forall v \in V$

- *Πόσος χώρος χρειάζεται για την αποθήκευση του αποτελέσματος (εξόδου) ενός αλγορίθμου ΕΣΔΑΚ ;*

3. $$\delta(s) = \min\{0, \min\{\delta(u) + c(e); e = (u, s) \in E\}\}$$

   και

   $$\delta(v) = \min\{\delta(u) + c(e); e = (u, v) \in E\}$$

   για $v \neq s$

4. Έστω d μια συνάρτηση $V :\rightarrow \mathbb{R} \cup \{-\infty, +\infty\}$ με
   - $d(v) \geq \delta(v), \forall v \in V$,
   - $d(s) \leq 0$, και
   - $d(v) \leq d(u) + c(u, v), \forall e = (u, v) \in E$ (⋆)

   Τότε, $d(v) = \delta(v), \forall v \in V$

- *Παρατήρηση 1:* Αν $\exists$ αρνητικός κύκλος, τότε $\nexists$ $d(\cdot)$ που να ικανοποιεί τις παραπάνω συνθήκες
- *Παρατήρηση 2:* $\geq n - 1$ ανισότητες στην (⋆) πρέπει να ισχύουν με ισότητα

- Διαχωρισμός του V σε

$$\begin{aligned} V^- &= \{v \in V; \delta(v) = -\infty\} \\ V^f &= \{v \in V; -\infty < \delta(v) < +\infty\} \\ V^+ &= \{v \in V; \delta(v) = +\infty\} \end{aligned}$$

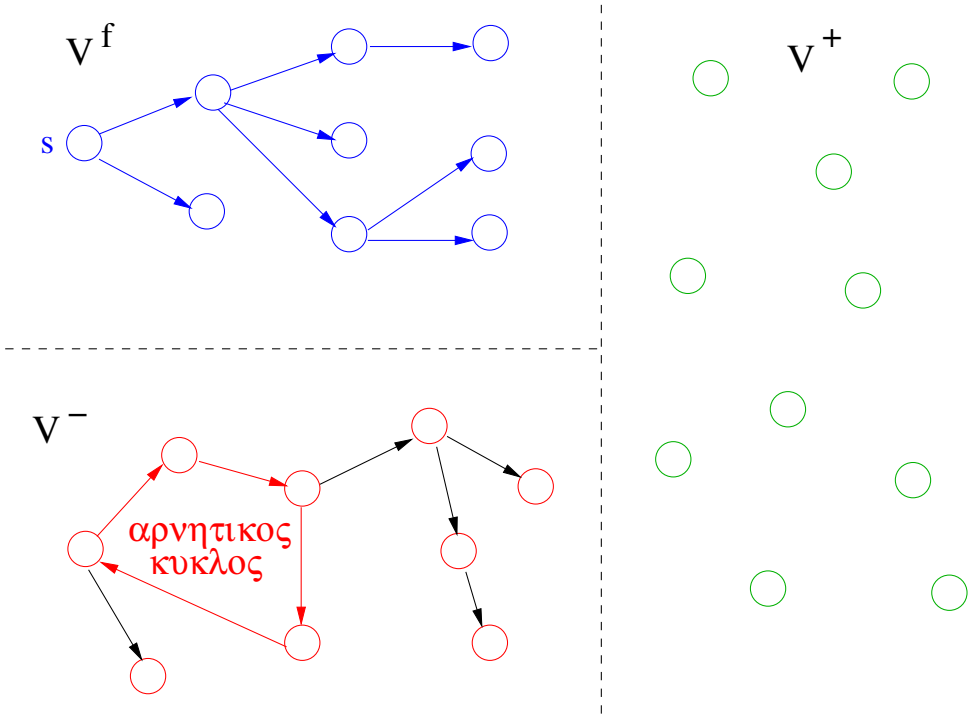
- **Δένδρο Συντομότερων Διαδρομών:** δένδρο οριζόμενο στο V^f, με ρίζα s, τέτοι ώστε $\forall v \in V^f$ η δενδρική s-v διαδρομή είναι μια συντομότερη s-v διαδρομή στο G

$$\Downarrow$$

Απαιτήσεις χώρου για το αποτέλεσμα ΕΣΔΑΚ (έξοδος αλγορίθμου): $O(n)$

ΕΣΔΑΚ – Έξοδος Αλγορίθμου

- $\delta(v), \forall v \in V$
- Δένδρο συντομότερων διαδρομών (ΔΣΔ) στο V^f
- Συλλογή αρνητικών κύκλων και δένδρα που απολήγουν από αυτούς στο V^-



Επίλυση Προβλήματος ΕΣΔΑΚ (G, s, c)

- `node_array<NT> dist`
  `node_array<edge> pred`
  - $dist[v]$ αποθηκεύει την απόσταση της κορυφής v από την s
  - $pred[v]$ αποθηκεύει την τελευταία ακμή μιας s-v ΣΔ αν $\exists$, αλλιώς nil

- Ο πίνακας $pred$ ορίζει το «γράφημα προκατόχων» (predecessor graph)

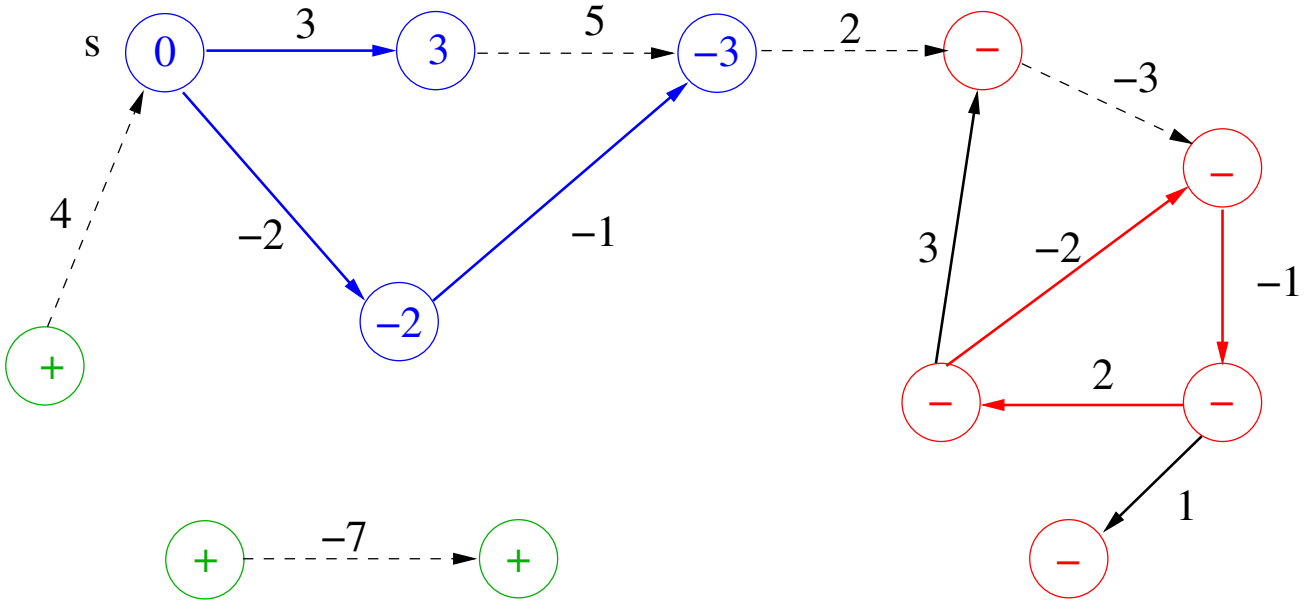
$$P = \{pred[v]; v \in V \text{ και } pred[v] \neq nil\}$$

Επίλυση Προβλήματος ΕΣΔΑΚ (G, s, c)

Ιδιότητες του ζεύγους $(dist, pred)$

- $s \in V^f$ αvv $pred[s] = nil$, και $s \in V^-$ αvv $pred[s] \neq nil$
- Για $v \neq s$: $v \in V^+$ αvv $pred[v] = nil$ και $v \in V^f \cup V^-$ αvv $pred[v] \neq nil$
- $v \in V^f$ αν $\exists$ μία s-v διαδρομή στο P και $s \in V^f$. Το επαγώμενο, από τις κορυφές του V^f, υπογράφημα του P είναι ένα ΔΣΔ και $dist[v] = \delta(v)$, $\forall v \in V^f$
- Όλοι οι κύκλοι στο P έχουν αρνητικό κόστος και $v \in V^-$ αvv η v βρίσκεται σε κύκλο στο P ή είναι προσπελάσιμη από έναν κύκλο στο P

Επίλυση Προβλήματος ΕΣΔΑΚ (G, s, c)



Οι διακεκομμένες ακμές δεν ανήκουν στο P

Πρόβλημα ΕΣΔΑΚ – Αλγόριθμοι & Υλοποιήσεις

- Κατευθυνόμενα ακυκλικά γραφήματα
- Μη αρνητικά κόστη ακμών

Πρόβλημα ΕΣΔΑΚ – Κατευθυνόμενα ακυκλικά γραφήματα

- `void ACYCLIC_SHORTEST_PATH_T(const graph& G, node s,`
  `const edge_array<NT>& c,`
  `node_array<NT>& dist,`
  `node_array<edge>& pred)`

- Χρόνος: $O(n+m)$

Ιδέα Αλγορίθμου

- Έστω G ένα κατευθυνόμενο ακυκλικό γράφημα, έστω $v_1, v_2, \ldots, v_n$ η τοπολογική του διάταξη, δηλ. $(v_i, v_j) \in E \Rightarrow i < j$, και έστω $s = v_k$
- Κορυφές v_ℓ με $\ell < k$ δεν είναι προσπελάσιμες από την s
- Για τις υπολοιπες κορυφές v_j και τις r εισερχόμενες ακμές τους (v_i, v_j), ισχύει ότι $(i, j \geq k)$:

$$\delta(v_j) = \min_{1 \leq i \leq r} \{\delta(v_i) + c(v_i, v_j)\}$$

- Ένας άλλος τρόπος υπολογισμού του παραπάνω ελάχιστου:

$\delta(v_j) = +\infty$
for $i = 1$ to r
    if $\delta(v_j) > \delta(v_i) + c(v_i, v_j)$ then
        $\delta(v_j) = \delta(v_i) + c(v_i, v_j)$

Πρόβλημα ΕΣΔΑΚ – Κατευθυνόμενα ακυκλικά γραφήματα

```
// initialization - computing topological order

node_array<int> top_ord(G);
TOPSORT(G,top_ord);
    // top_ord is now a topological ordering of G

int n = G.number_of_nodes();

array<node> v(1,n);

node w;
forall_nodes(w,G) v[top_ord[w]] = w;
                  // top_ord[v[i]] == i for all i
```

Πρόβλημα ΕΣΔΑΚ – Κατευθυνόμενα ακυκλικά γραφήματα

```
template <class NT>
void ACYCLIC_SHORTEST_PATH_T(const graph& G, node s,
                             const edge_array<NT>& c,
                             node_array<NT>& dist,
                             node_array<edge>& pred)
{
  // initialization - computing  topological order

  forall_nodes(w,G) pred[w] = nil;
  dist[s] = 0;

  for(int i = top_ord[s]; i <= n; i++)
  { node u = v[i];
    if ( pred[u] == nil && u != s ) continue;
    edge e;
    NT du = dist[u];
    forall_adj_edges(e,u)
    { node w = G.target(e);
      if ( pred[w] == nil || du + c[e] < dist[w])
      { pred[w] = e;
        dist[w] = du + c[e];
      }
    }
  }
}
```

Πρόβλημα ΕΣΔΑΚ – Μη αρνητικά κόστη ακμών

```
void DIJKSTRA(graph& G, node s,
              const edge_array<int>& cost,
              node_array<int>& dist,
              p_queue<int,node>& PQ)
```

Πρόβλημα ΕΣΔΑΚ – Αλγόριθμος Dijkstra

- Επαναληπτικός αλγόριθμος (έως ότου όλες οι κορυφές γίνουν ανενεργές)
- Σε κάθε επανάληψη, διατηρεί μια προσωρινή απόσταση $dist[v], \forall v \in V$, και ένα σύνολο *ενεργών* κορυφών
- Αρχικά, μόνο η s είναι ενεργή, $dist[s] = 0$, και για κάθε άλλη κορυφή v, $dist[v] = +\infty$
- Σε κάθε επανάληψη: επιλογή ενεργής κορυφής u με *ελάχιστη* $dist[u]$
  - Η u γίνεται ανενεργή και η $dist[u]$ είναι η τελική απόσταση της u
  - «Χαλάρωση» εξερχόμενων ακμών της (u, v):
    if $dist[v] > dist[u] + c(u, v)$ then $dist[v] = dist[u] + c(u, v)$

Λεπτομέρειες Υλοποίησης

- `dist[v]`: αποθηκεύει τελική και προσωρινή απόσταση μιας κορυφής v
- `PQ`: ουρά προτεραιότητας που αποθηκεύει τα ζεύγη $(dist[v], v)$, όπου v είναι ενεργή
- Κάθε ενεργή κορυφή v πρέπει να γνωρίζει το στοιχείο (`pq_item`) της `PQ`, το οποίο περιέχει το ζεύγος $(dist[v], v)$
  Το στοιχείο αυτό αποθηκεύεται στη θέση `I[v]` ενός `node_array<pq_item> I`

Πρόβλημα ΕΣΔΑΚ – Αλγόριθμος Dijkstra

```
{ node_array<pq_item> I(G);
  node v;

  forall_nodes(v,G) dist[v] = MAXINT;
  dist[s] = 0; I[s] = PQ.insert(0,s);

  while (! PQ.empty())
  {  pq_item it = PQ.find_min();
     node u = PQ.inf(it);
     int du = dist[u];
     edge e;
     forall_adj_edges(e,u)
     {  v = G.target(e);
        int c = du + cost[e];
        if (c < dist[v])
        {  if (dist[v] == MAXINT)
           I[v] = PQ.insert(c,v);
        else
           PQ.decrease_p(I[v],c);
        dist[v] = c;
        }
     }
     PQ.del_item(it);
  }
}
```

Πρόβλημα ΕΣΔΑΚ – Αλγόριθμος Dijkstra: χρονική πολυπλοκότητα

- $\leq n$ λειτουργίες `insert, empty, find_min, delete_min`
- $\leq m$ λειτουργίες `decrease_p`
- Υπόλοιπος χρόνος, εκτός των κλήσεων στην `PQ`, $O(n+m)$
- $$\begin{aligned} T_{total} = O(&n+m \\ &+ n \cdot (T_{insert} + T_{empty} + T_{find\_min} + T_{delete\_min}) \\ &+ m \cdot T_{decrease\_p} \\ &+ T_{create} + T_{destruct}) \end{aligned}$$

Πρόβλημα ΕΣΔΑΚ – Αλγόριθμος Dijkstra: ασυμπτητική ΠΧΠ

	Πολυπλοκότητα Χειρότερης Περίπτωσης
f_heap	$O(m + n \log n)$
p_heap	$O(m + n \log n)$
k_heap	$O(m \log_k n + nk \log_k n)$
bin_heap	$O(m \log n + n \log n)$
list_pq	$O(m + n^2)$
b_heap	$O((m + n)nM)$
r_heap	$O(m + n \log M)$
m_heap	$O(m + max\_dist + \mathrm{M})$

$$
\begin{aligned}
M &= 1 + \text{μέγιστο ακέραιο κόστος ακμής} \\
max\_dist &= \text{μέγιστη απόσταση κορυφής από την } s \\
&\leq (n-1)M
\end{aligned}
$$

Πρόβλημα ΕΣΔΑΚ – Μια διαφορετική υλοποίηση της DIJKSTRA

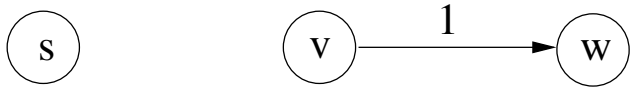
- Αλγόριθμοι ΣΔ: πολύ βολική η αρχικοποίηση αποστάσεων με ∞
- Π.χ. ακέραια κόστη ακμών: φαίνεται λογικό να υλοποιήσουμε το ∞ με `MAXINT`

```
void DIJKSTRA(const graph& G, node s, const edge_array<int>& cost,
                                      node_array<int>& dist)
{ node_pq PQ(G); node v; edge e;

  forall_nodes(v,G) dist[v] = MAXINT;
  dist[s] = 0;
  forall_nodes(v,G) PQ.insert(v,dist[v]);

  while (! PQ.empty())
  {  node v = PQ.delete_min();
     forall_adj_edges(e,v)
     {  node w = G.target(e);
        if ( dist[v] + cost[e] < dist[w] )
        { dist[w] = dist[v] + cost[e]; PQ.decrease_p(w,dist[w]);}
     }
  }
}
```

Προσοχή με τους αριθμητικούς τύπους !

- Δεν πρέπει να ξεχνάτε ότι
  «`MAXINT + 1 = MININT`» $\neq$ «$\infty + 1 = \infty$»

- Αντιπαράδειγμα



- Η προηγούμενη υλοποίηση δουλεύει σωστά όταν όλες οι κορυφές είναι προσπελάσιμες από την s και όταν όλα τα κόστη των πλευρών είναι στο διάστημα $[0..\texttt{MAXINT}/n]$

Πρόβλημα ΕΣΔΑΚ – Μια διαφορετική υλοποίηση της DIJKSTRA

- Πρώτη προσπάθεια σωστής υλοποίησης:
  εισάγει μόνο την *s* στην `PQ` και αλλάζει το μπλόκ της εντολής `if`

```
void DIJKSTRA(const graph& G, node s, const edge_array<int>& cost,
                                                node_array<int>& dist)
{ node_pq PQ(G);
  node v; edge e;

  forall_nodes(v,G) dist[v] = MAXINT;
  dist[s] = 0; PQ.insert(s,0);

  while (! PQ.empty())
  {  node v = PQ.delete_min();
     forall_adj_edges(e,v)
     {  node w = G.target(e);
        if ( dist[v] + cost[e] < dist[w] )
        { int c = dist[v] + cost[e];
          if (dist[w] == MAXINT) PQ.insert(w,c);
          else PQ.decrease_p(w,c);
          dist[w] = c;
        }
     }
  }
}
```

Πρόβλημα ΕΣΔΑΚ – Μια διαφορετική υλοποίηση της DIJKSTRA

- Μια ακόμη καλύτερη υλοποίηση, ανεξάρτητη αριθμητικών τύπων

```
template <class NT>
void DIJKSTRA_T(const graph& G, node s, const edge_array<NT>& cost,
                node_array<NT>& dist, node_array<edge>& pred)
{ node_pq<NT>  PQ(G);
  node v; edge e;
  dist[s] = 0; PQ.insert(s,0);

  forall_nodes(v,G) pred[v] = nil;

  while (!PQ.empty())
  { node u = PQ.del_min();
    NT du = dist[u];
    forall_adj_edges(e,u)
    { v = G.opposite(u,e); // makes it work for ugraphs
      NT c = du + cost[e];
      if (pred[v] == nil && v != s )
        PQ.insert(v,c); // v is reached for the first time
      else if (c < dist[v]) PQ.decrease_p(v,c);
           else continue;
      dist[v] = c;
      pred[v] = e;
    }
  }
}
```

Πρόβλημα ΕΣΔΑΚ – Πειραματική Αξιολόγηση της DIJKSTRA

- Μεγέθη εισόδου: $m = 500000$, $n \in \{2000, 20000, 200000\}$, $M \in \{100, 100000\}$

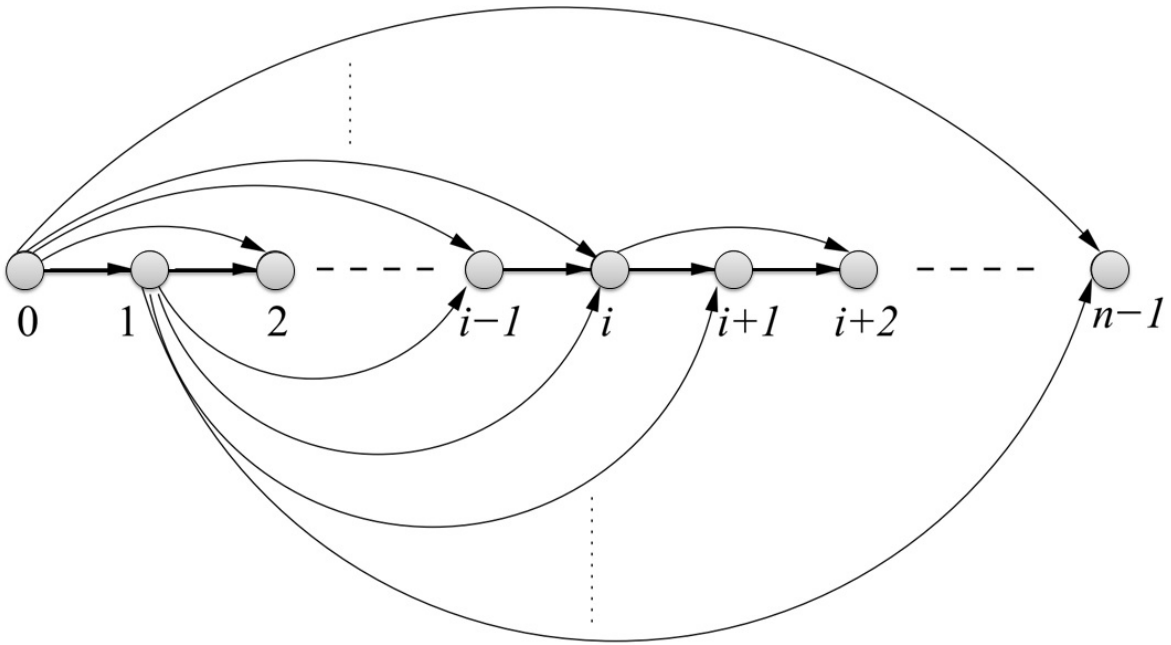
- Τυχαία γραφήματα
  - `r_heap` < `p_heap` < `f_heap`

- Συνθετικά Γραφήματα: εξαναγκάζουν τον αλγόριθμο να εκτελέσει $m - n + 1$ λειτουργίες `decrease_p`
  - `r_heap` < `p_heap` < `f_heap`
  - m/n πολύ μικρό και M πολύ μεγάλο ⇒ `p_heap` < `r_heap` < `f_heap`

Πρόβλημα ΕΣΔΑΚ – Συνθετικό γράφημα για την DIJKSTRA

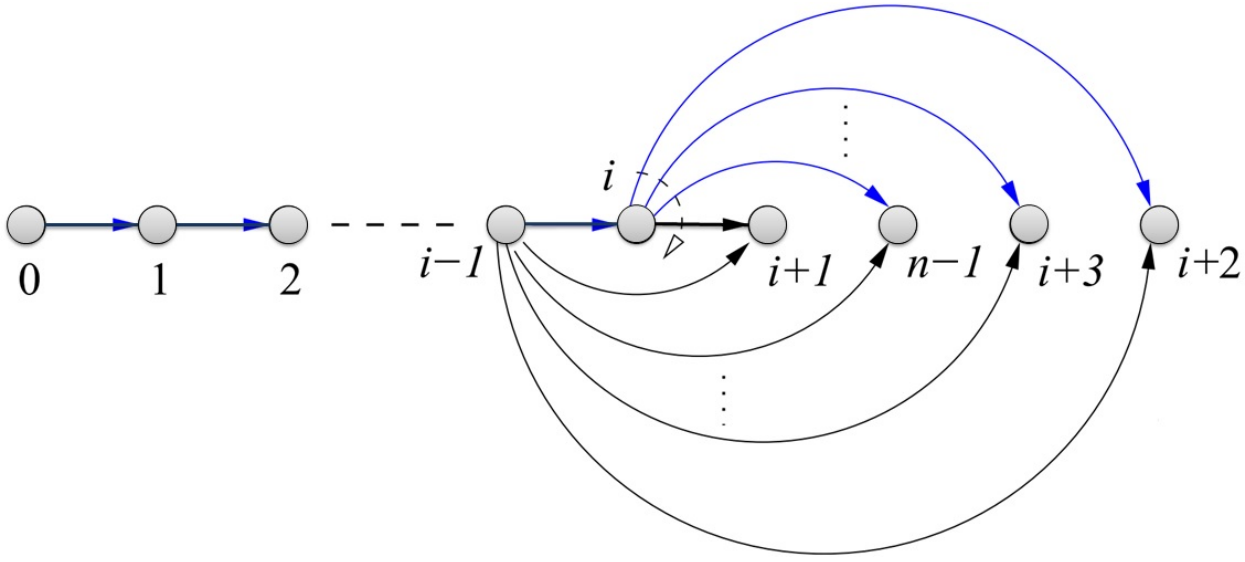
Συνθετικό γράφημμα n κορυφών και $m \leq n(n-1)/2$ ακμών

- Κορυφές $0, 1, \ldots, n-1$
- $n-1$ ακμές $(i, i+1)$, $0 \leq i < n-1$, κάθε μία με κόστος $M \geq 0$
- Οι πρώτες $m' = m - (n-1)$ ακμές της ακολουθίας $(0,2), (0,3), \ldots, (0,n-1), (1,3), (1,4), \ldots, (1,n-1), (2,4), \ldots$.
  Το κόστος $c_{i,j}$ της ακμής (i,j) πρέπει να είναι τέτοιο ώστε να εκτελούνται m' λειτουργίες `decrease_p`



Πρόβλημα ΕΣΔΑΚ – Συνθετικό γράφημα για την DIJKSTRA

Προσδιορισμός κόστους $c_{i,j}$ για εύρεση ΕΣΔΑΚ από την κορυφή 0

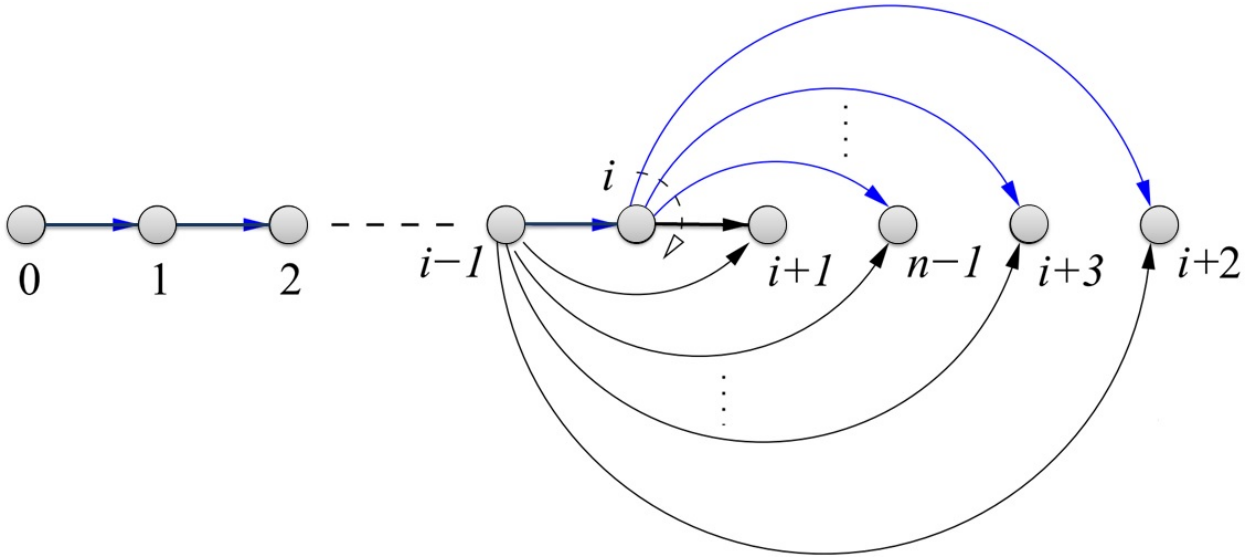


- Μπλέ ακμές = ΔΣΔ με ρίζα την κορυφή 0 μετά τη διαγραφή της i από την ουρά προτεραιότητας Q,
  δηλ. $d(i) = \delta(0, i) = iM$, και $c[0, 1, \ldots, i-1, i, j] = iM + c_{i,j}$

- Οι κορυφές διαγράφονται από την ουρά προτεραιότητας Q σύμφωνα με την αρίθμησή τους, προκαλώντας τη μέγιστη δυνατή αλλαγή στην Q

Πρόβλημα ΕΣΔΑΚ – Συνθετικό γράφημα για την DIJKSTRA

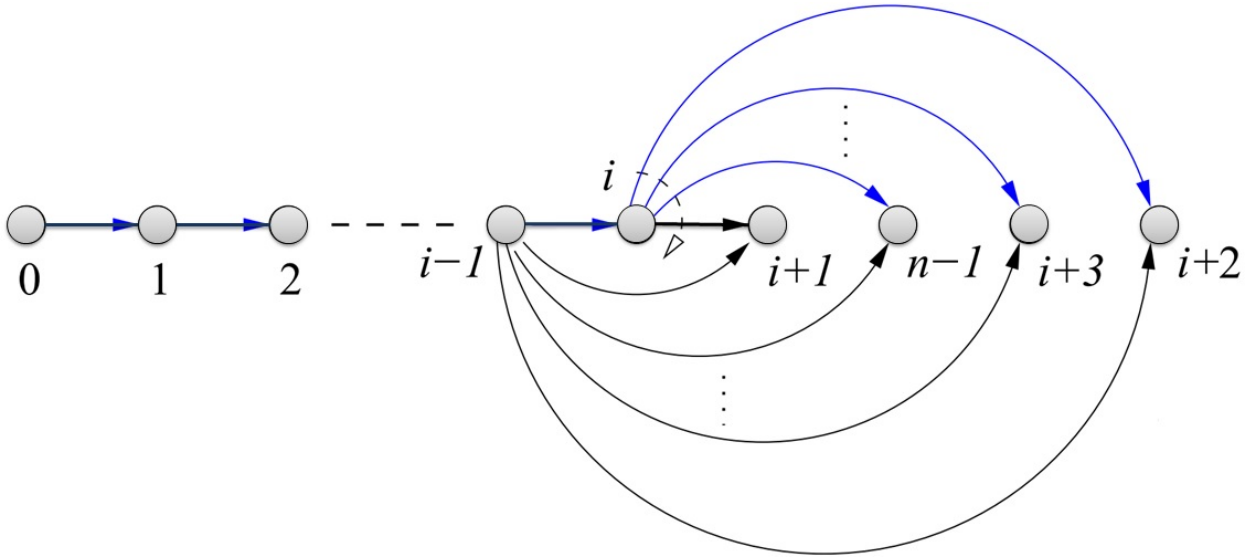
Προσδιορισμός κόστους $c_{i,j}$ για εύρεση ΕΣΔΑΚ από την κορυφή 0



- *Πριν τη διαγραφή της i από την Q*
  - $\{i, i+1, \ldots, n\} \in Q$
  - $d(i) = iM$, και $d(j) = (i-1)M + c_{i-1,j}$, $j > i$
- *Μετά τη διαγραφή της i από την Q*
  - Χαλάρωση εξερχόμενων ακμών σύμφωνα με τη σειρά $(i, i+2), (i, i+3), \ldots, (i, n-1), (i, i+1)$
  - Κάτω από ποιές συνθήκες η χαλάρωση της (i, j) θα προκαλέσει την εκτέλεση μιας `decrease_p` και τη μέγιστη δυνατή αλλαγή στην Q;

Πρόβλημα ΕΣΔΑΚ – Συνθετικό γράφημα για την DIJKSTRA

Προσδιορισμός κόστους $c_{i,j}$ για εύρεση ΕΣΔΑΚ από την κορυφή 0



- Χαλάρωση $(i, j) \Rightarrow$ `decrease_p`, αν $iM + c_{i,j} < (i-1)M + c_{i-1,j}, j > i$
- Μέγιστη αλλαγή στην Q αν

  πριν τη χαλάρωση

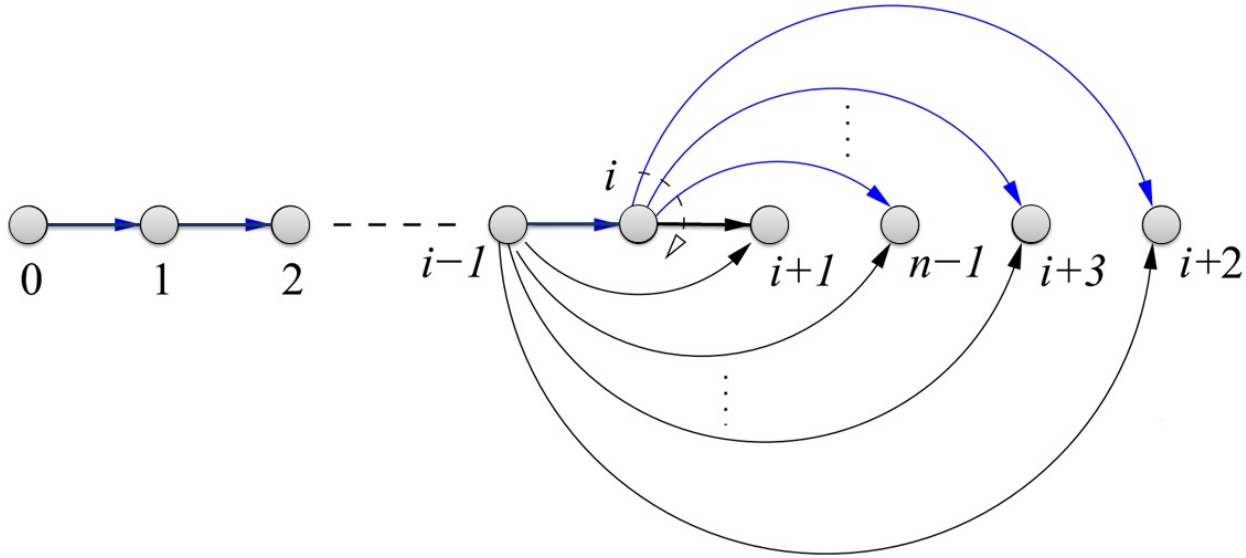
  $d(j-1) < \ldots < d(i+2) < d(n-1) < \ldots < d(j) < \ldots < d(i+1)$

  μετά τη χαλάρωση

  $d(j) < d(j-1) < \ldots < d(i+2) < d(n-1) < \ldots < d(j+1) < \ldots < d(i+1)$

Πρόβλημα ΕΣΔΑΚ – Συνθετικό γράφημα για την DIJKSTRA

Προσδιορισμός κόστους $c_{i,j}$ για εύρεση ΕΣΔΑΚ από την κορυφή 0



- Τα κόστη $c_{i,j}$ πρέπει να επιλεγούν έτσι ώστε
  - $iM + c_{i,i+2} < (i-1)M + c_{i-1,n-1} \Rightarrow (i, i+2)$ προξενεί την κλήση μιας `decrease_p`
  - $c_{i,j} < c_{i,j-1}, \forall\, i+2 < j \leq n-1 \Rightarrow (i,j)$ προξενεί την κλήση μιας `decrease_p`, δηλ., οι ακμές $(i, i+3), (i, i+4), \ldots, (i, n-2), (i, n-1)$ χαλαρώνονται με αυτή τη σειρά
  - $M = c_{i,i+1} < c_{i,n-1} \Rightarrow (i, i+1)$ προξενεί την κλήση μιας `decrease_p`

Πρόβλημα ΕΣΔΑΚ – Συνθετικό γράφημα για την DIJKSTRA

Προσδιορισμός κόστους $c_{i,j}$ για εύρεση ΕΣΔΑΚ από την κορυφή 0

- $M + c_{i,i+2} < c_{i-1,n-1}$
- $M = c_{i,i+1} < c_{i,n-1} < c_{i,n-2} < c_{i,n-3} < \cdots < c_{i,i+3} < c_{i,i+2}$

Επιλογή των $c_{i,j}$ έτσι ώστε να ικανοποιούνται οι ανισότητες

- Προσδιορισμός των m' επιπρόσθετων ακμών
- Ανάθεση κόστους αυτών των m' επιπρόσθετων ακμών σε αντίστροφη σειρά
- Ανάθεση κόστους τελευταίας ακμής ίσο με $M + 1$
- $$c_{i,j} = \begin{cases} c_{i,j+1} + 1 & \text{αν } j < n-1 \\ c_{i+1,i+3} + M + 1 & \text{αν } j = n-1 \end{cases}$$

Τέλος Ενότητας

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση **1.0**.

Σημείωμα Ιστορικού Εκδόσεων Έργου

Σημείωμα Ιστορικού Εκδόσεων Έργου

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει :

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει) μαζί με τους συνοδευόμενους υπερσυνδέσμους