Βασικοί αλγόριθμοι σε γραφήματα και εφαρμογές τους

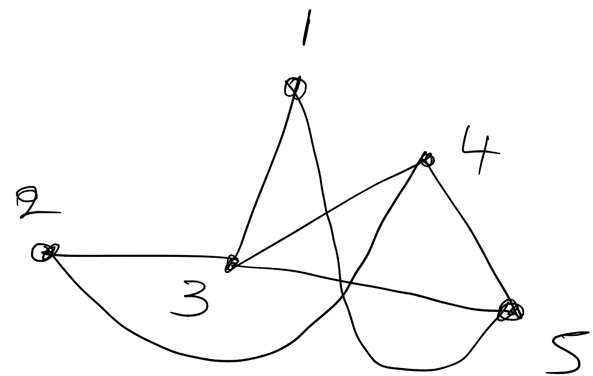
Ευριπίδης Μάρκου

- Γραφήματα αναπαράσταση και τεχνικές αναζήτησης
- Λίστες γειτνίασης
- Πίνακας γειτνίασης
- bfs Ιδιότητες και εφαρμογές
- dfs Ιδιότητες και εφαρμογές
- Min spanning trees και εφαρμογές
- Αλγόριθμοι Prim and Kruskal
- Shortest paths
- Αλγόριθμος Bellman-Ford
- Αλγόριθμος Dijkstra
- Αλγόριθμος Floyd-Warshall
- Εφαρμογές

<u>Γράφημα</u> : $G(V, E)$

$$|V| = n \geq 1$$

$$0 \leq |E| \leq \frac{n(n-1)}{2}$$



$V = \{1, 2, 3, 4, 5\}$

$E = \{(1,3), (1,5), (2,3), (2,4), (3,4), (3,5), (4,5)\}$

- συνεκτικό γράφημα
- μονοπάτι
- κύκλος
- πλήρες γράφημα
- κατευθυνόμενο γράφημα
- δέντρο

- Τρόποι αναπαράστασης (αποθήκευσης - κωδικοποίησης) ενός γραφήματος
- Αλγόριθμοι αναζήτησης σε γράφημα:

• Ακολουθούμε συστηματικά τις ακμές του γραφήματος και επισκεπτόμαστε τις κορυφές του.	• Μπορούν να μας αποκαλύψουν πολλές πληροφορίες σχετικά με τη δομή του γραφήμ

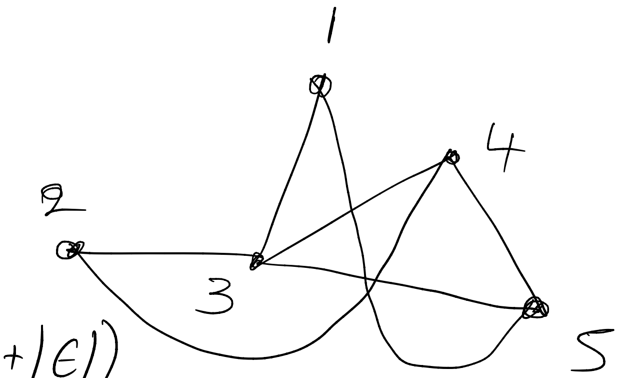
• Αποτελούν δομικό στοιχείο ενός αλγορίθμου που εφαρμόζεται σε ένα γράφημα.

- Αναπαράσταση γραφημάτων

• Λίστες γειτνίασης

1 → 3, 5
2 → 3, 4
3 → 2, 1, 4, 5
4 → 3, 2, 5
5 → 4, 3, 1

$|V| = n$



χώρος: $\Theta(|V| + |E|)$

$(i,j) \overset{?}{\in} E(G) \longrightarrow$ Χρόνος: $O(n)$

• Πίνακας γειτνίασης

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

Γενικά $A_{n \times n}$: $A_{ij} = \begin{cases} 1, & (i,j) \in E(G) \\ 0, & \text{αλλιώς} \end{cases}$

Χώρος: $\Theta(n^2)$

$(i,j) \in E(G) \longrightarrow$ Χρόνος: $O(1)$

- Αναπαράσταση με λίστες γειτνίασης είναι ασυμπτωτικά τουλάχιστον τόσο αποδοτική όσο η αναπαράσταση με πίνακα γειτνίασης.
- Αναπαράσταση με πίνακα γειτνίασης είναι απλούστερη και χρησιμοποιεί n^2 Bits.

<u>Ασκήσεις:</u>

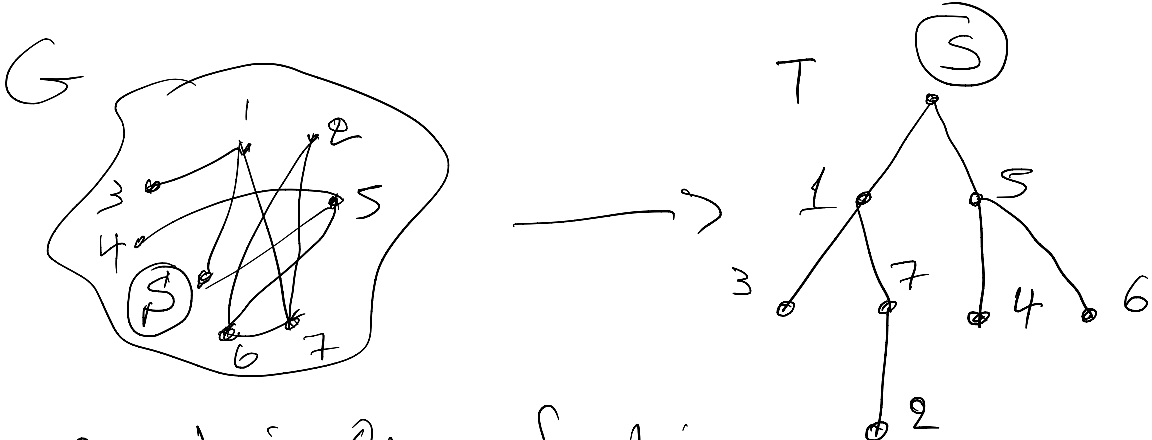
- Υπολογίστε το G^2, όπου 2 κορυφές u, v, συνδέονται με ακμή ανν υπάρχει μονοπάτι μήκους 2 που τις συνδέει στο G.

• Έστω κατευθυνόμενο γράφημα $G(V,E)$.
Να δώσετε έναν αλγόριθμο που αποφασίζει σε χρόνο $O(|V|)$ αν υπάρχει ένας κόμβος με in-degree $|V|-1$ και out-degree 0 όταν δίνεται ο πινακάς γειτνίασης του G.

Breadth-First-Search (BFS)
(Αναζήτηση κατά πλάτος)

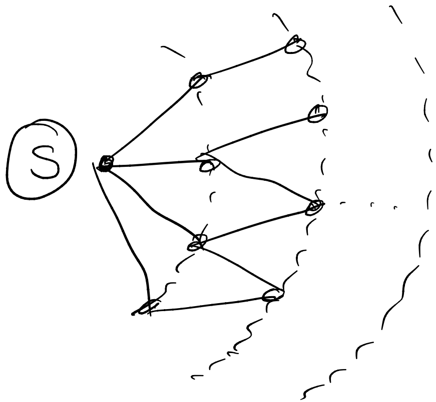
Είσοδος: $G(V,E)$, $s \in V$

- συστηματική εξερεύνηση του G για την ανακάλυψη κάθε κορυφής που "φτάνεται" από την s
- υπολογίζει την απόσταση (μήκος μικρότερου μονοπατιού) κάθε τέτοιας κορυφής από την s
- παράγει ένα BFS-tree με ρίζα το s που περιέχει όλους τους κόμβους που "φτάνονται" από το s.



$\forall v \in T$
$p(s \to v)$ in T
is shortest
in G

– Ο αλγόριθμος δουλεύει και σε κατευθυνόμενα γραφήματα.



ο αλγόριθμος ανακαλύπτει όλες τις κορυφές που βρίσκονται σε απόσταση k από την s πριν βρει κορυφή που βρίσκεται σε απόσταση $k+1$.

```
Procedure BFS(G, s)
{ for each v ∈ V(G) do                                   O(|V|)
      { π(v) := NIL; d(v) := ∞; visited(v) := FALSE; }
  d(s) := 0; visited(s) := TRUE; EnQueue(s);
  repeat DeQueue(u)
          for each node w adjacent to u do
                if not visited(w) then
                { visited(w) := TRUE;                     O(|E|)
                  d(w) := d(u) + 1;
                  π(w) := u;
                  EnQueue(w);
                }
  until Queue is empty;
}
```

- Χρονική πολυπλοκότητα: $O(|V|+|E|)$
- Ιδιότητες

— Αν η ουρά έχει: $v_1, \ldots, v_r$ τότε

- $d(v_i) \leq d(v_{i+1})$
- $d(v_r) \leq d(v_1)+1$

Δηλαδή υπάρχουν το πολύ 2 διαφορετικές τιμές για τις αποστάσεις των κορυφών που βρίσκονται στην ουρά.

— $d(v_i) = \delta(s, v_i)$ (μήκος συντομότερου μονοπατιού)

— BFS-tree περιέχει συντομότερα μονοπάτια από το s σε κάθε κορυφή.

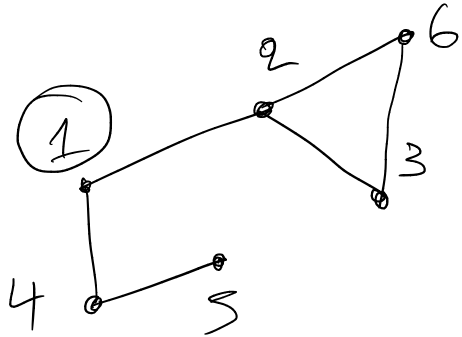
Ασκήσεις

- Δώστε έναν αλγόριθμο που αποφασίζει σε χρόνο $O(|V|+|E|)$ αν ένα γράφημα $G(V,E)$ είναι διμερές.
- Δώστε έναν αποδοτικό αλγόριθμο που υπολογίζει τη διάμετρο* ενός δέντρου

* διάμετρος ενός $T(V,E) \stackrel{\Delta}{=} \max\limits_{u,v \in T} \delta(u,v)$

Depth-First Search
(Αναζήτηση κατά βάθος)

- η εξερεύνηση των ακμών γίνεται με την εξής σειρά: πρώτα η ακμή που συνδέει την κορυφή που ανακαλύφθηκε πιο πρόσφατα και έχει ακόμη ακμές που δεν έχουν εξερευνηθεί.
- Όταν έχουν εξερευνηθεί όλες οι ακμές που συνδέουν έναν κόμβο v τότε η εξερεύνηση συνεχίζεται με την ακμή που συνδέει τον κόμβο από τον οποίο ανακαλύφθηκε ο v.



$\xrightarrow{\text{DFS}}$ 1, 2, 3, 6, 4, 5

Procedure DFS(G, s)

{ <u>for</u> each $v \in G$ <u>do</u>

{ visited(v) := FALSE; π(v) := NIL; }

dfs-visit(s);

}

```
dfs-visit(s)
{ visited(s):= TRUE;
  for each node w adjacent to s do
      if not visited(w) then
      { π(w):=s;
        dfs-visit(w);
      }
}
```

- Χρονική πολυπλοκότητα: $\Theta(|V|+|E|)$

<u>Ασκήσεις</u>

- Δώστε έναν αλγόριθμο που υπολογίζει σε γραμμικό χρόνο τον αριθμό των μονοπατιών μεταξύ 2 κορυφών s, t σε ένα DAG.
- Δώστε έναν αλγόριθμο που αποφασίζει αν ένα μη-κατευθυνόμενο γράφημα περιέχει κύκλο.

Η διαδικασία DFS μπορεί να χρησιμοποιηθεί για να βρει τις συνεκτικές συνιστώσες ενός κατευθυνόμενου γραφήματος.

$DFS(G)$, $DFS(\bar{G})$

όπου $\bar{G}(V, \bar{E})$: $(u,v) \in \bar{E}$ iff $(v,u) \in E$

Ασκήσεις

- Δώστε έναν αλγόριθμο πολυπλοκότητας $O(|E|)$ που βρίσκει μια διαδρομή Euler σε ένα γράφημα G, αν υπάρχει.

Minimum Spanning Trees

(Ελάχιστο Συνδετικό Δένδρο)

- Διατήρηση ή δημιουργία ενός συνδεδεμένου δικτύου με ελάχιστο κόστος.

Ιδιότητες:

- Έστω (u,v) μια ακμή του G με ελάχιστο βάρος. Τότε η $(u,v) \in T$, όπου T ένα MST(G).
- Έστω (u,v) μια ακμή με μέγιστο βάρος που ανήκει σε κάποιο κύκλο του G. Τότε υπάρχει ένα MST(G) που δεν περιέχει τη (u,v).

- Αν τα βάρη όλων των ακμών του G είναι θετικά τότε οποιοδήποτε υποσύνολο ακμών που συνδέει όλες τις κορυφές και έχει ελάχιστο συνολικό βάρος πρέπει να είναι δέντρο. Αν υπάρχουν αρνητικά βάρη τότε δεν ισχύει το παραπάνω.

Άσκηση

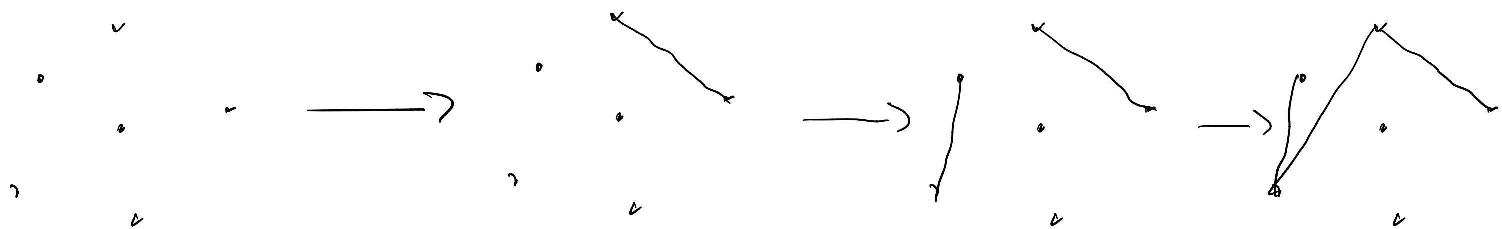
Έστω γράφημα G και T = MST(G). Μειώνουμε το βάρος κάποιας ακμής που δεν ανήκει στο T. Δώστε έναν αλγόριθμο που βρίσκει ένα MST στο καινούριο γράφημα.

Αλγόριθμος του Kruskal

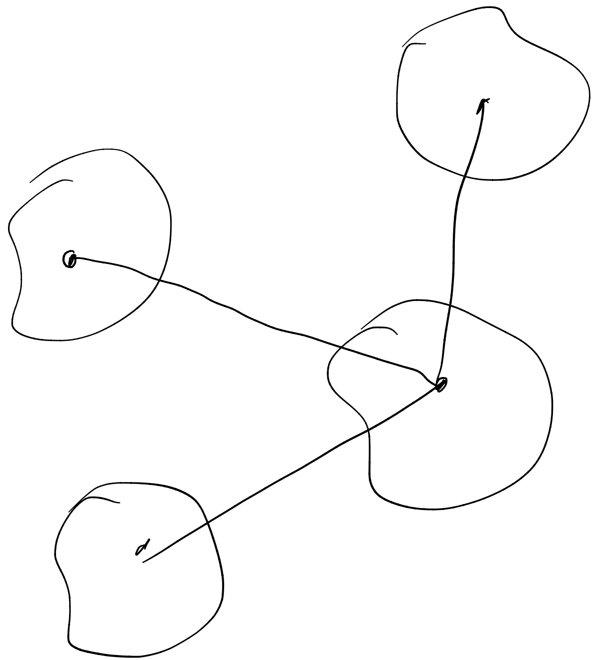
- $T = \emptyset$
- sort edges in non-decreasing order according to weights in list L
- <u>for</u> each edge $e \in L$ <u>do</u>
  - <u>if</u> $T+e$ does not contain a cycle <u>then</u>
    $T = T+e$

Πολυπλοκότητα: $O(E \log E) + O(E \cdot a(V)) = O(E \log V)$

• Ιδέα του αλγορίθμου του Kruskal:
Ξεκινά από ένα δάσος από δέντρα και σε κάθε φάση ενώνει 2 δέντρα σε ένα διαλέγοντας τη μικρότερου βάρους ακμή που συνδέει 2 δέντρα.



κλπ.

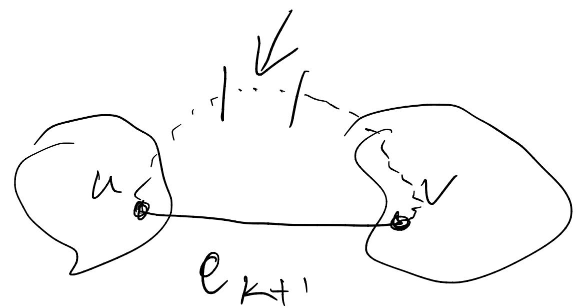


Αλγόριθμος

$e_1 \le e_2 \le \dots \le e_k \le e_{k+1} \dots$

Βέλτιστη

$e_1 \le \dots \le e_k \le e_{k+1} \le$



$w(e_{k+1}) \le (e'_{k+1})$

Αλγόριθμος του Prim

Ιδέα

- Ξεκινά από έναν τυχαίο κόμβο και σε κάθε φάση προσθέτει στο δέντρο την πιο φθηνή ακμή που ενώνει κάποια κορυφή του δέντρου με κάποια κορυφή που δεν ανήκει στο δέντρο

- $V(T) = \emptyset$; Insert $V(G)$ into a set Q;
- <u>For</u> each $u \in Q$ <u>do</u>

  $key[u] := \infty$; $\pi[u] := NIL$;

  $key[r] := 0$; Πολυπλοκότητα: $O(E \log V)$

  <u>While</u> $Q \neq \emptyset$ <u>do</u>

  - extract $u \in Q$: $key[u]$ is a minimum

    $V(T) = V(T) + \{u\}$;

    <u>for</u> each node v adjacent to u <u>do</u>

    <u>if</u> $v \in Q$ and $w(u,v) < key[v]$ <u>then</u>

    } $key[v] := w(u,v)$;
       $\pi[v] := u$;

    {

Ασκήσεις

- Έστω G και $T = MST(G)$. Αν προσθέσουμε ένα νέο κόμβο στο G καθώς και ακμές που τον συνδέουν στο G, πόσο γρήγορα μπορούμε να βρούμε το νέο spanning tree?

- Έστω γράφημα G και ένα δένδρο T με $V(T) \subseteq V(G)$. Πόσο γρήγορα μπορούμε να αποφασίσουμε αν $T = MST(G)$?

Shortest Paths problem

Με δεδομένο ένα γράφημα G με βάρη στις ακμές και δύο κόμβους s, t να βρεθεί ένα από τα συντομότερα μονοπάτια από το s στο t

- Αν απαριθμήσουμε όλα τα διαφορετικά μονοπάτια από το s στο t και διαλέξουμε ένα από εκείνα που έχουν το μικρότερο συνολικό κόστος? ← $(n-2)!$

s • n-2 κορυφές • t

Σχετικά προβλήματα (Shortest Path(u,v): $sp(u,v)$ ή $\delta(u,v)$)

- input(G, s, t): Find $sp(s,t)$
- input(G, s): Find $sp(s,u)$ $\forall u$
- input(G, t): Find $sp(u,t)$ $\forall u$
- input(G): Find $sp(u,v)$ $\forall u, v$

- κοινό δομικό στοιχείο των αλγορίθμων που λύνουν το πρόβλημα:

Αν $p = \langle v_1, v_2, \ldots, v_k \rangle$ και $p = sp(v_1, v_k)$ τότε

$\forall i, j: 1 \leq i < j \leq k \quad p_{ij} = \langle v_i \ldots v_j \rangle : p_{ij} = sp(v_i, v_j)$

Για είσοδο $(G, s) \longrightarrow sp(s, v)\ \forall v$

- αν υπάρχουν ακμές με αρνητικό βάρος τότε το πρόβλημα παραμένει καλά ορισμένο αν δεν υπάρχει κύκλος με συνολικό αρνητικό βάρος που φτάνεται από το s.

Βασική διαδικασία

$sp(s,u) \leq d(s,u)$

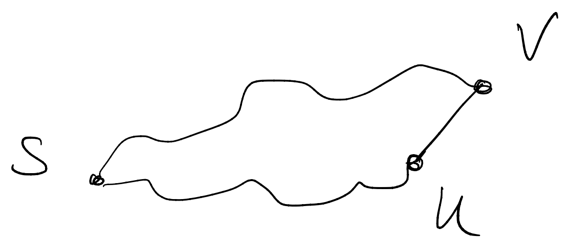
Relax (u, v, w)

<u>if</u> $d(s,v) > d(s,u) + w(u,v)$ <u>then</u>

{ $d(s,v) := d(s,u) + w(u,v)$;

$\pi(v) := u$;

}



Οι αλγόριθμοι Dijkstra και Bellman-Ford διαφέρουν ως προς:

- πόσες φορές εκτελούν τη διαδικασία relax στις ακμές
- με ποια σειρά

Dijkstra: μια φορά για κάθε ακμή

Bellman-Ford: πολλές φορές

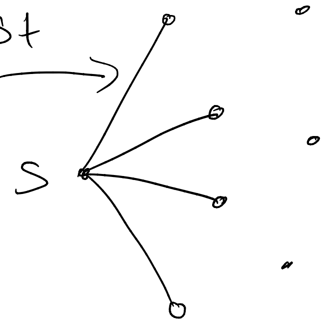
```
Bellman-Ford (G, s)
for i := 1 to |V(G)| - 1 do
    for each edge (u, v) ∈ E(G) do
        Relax(u, v, w);
for each edge (u, v) ∈ E(G) do
    if d(s, v) > d(s, u) + w(u, v) then
        return (FALSE)
```

- Πολυπλοκότητα $O(VE)$

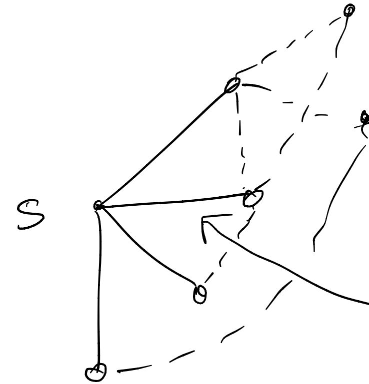
Χαρακτηριστικά του αλγορίθμου Bellman-Ford

- δουλεύει και αν υπάρχουν ακμές με αρνητικά βάρη
- Βρίσκει αν υπάρχει κύκλος που φτάνεται από το s και έχει συνολικό αρνητικό βάρος
- Μετά από i επαναλήψεις έχουν βρεθεί όλα τα shortest paths με $\leq i$ ακμές.
- υπάρχει αρνητικός κύκλος που φτάνεται από το s ανν $\exists (u,v): d(s,v) > d(s,u) + w(u,v)$

Αν shortest δεν θα αλλάξει αργότερα

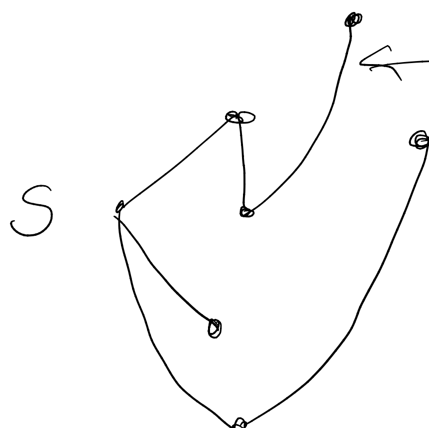


$i = 1$

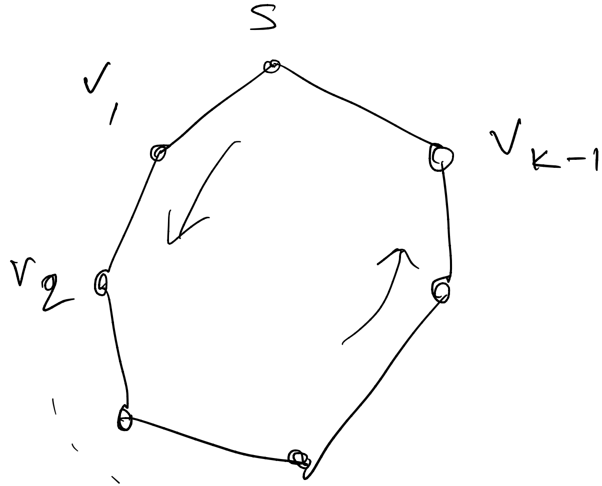


$i = 2$

Αν υπάρχει shortest μήκους 2 το βρίσκει



Αν υπάρχει shortest μήκους 3 το βρίσκει



Άρα $\sum_{i=1}^{k} w(v_{i-1}, v_i) \geq 0$

Αν $d(s, v_i) \leq d(s, v_{i-1}) + w(v_{i-1}, v_i) \quad 1 \leq i \leq k$

όπου $v_0 = v_k = s$

τότε $\sum_{i=1}^{k} d(s, v_i) \leq \sum_{i=1}^{k} d(s, v_{i-1}) + \sum_{i=1}^{k} w(v_{i-1}, v_i)$

$a + 0 \qquad 0 + a$

Άσκηση

- Δώστε αλγόριθμο που υπολογίζει τα shortest paths από κορυφή s σε $O(V+E)$ χρόνο σε ένα DAG.

Ο αλγόριθμος του Dijkstra

$S := \emptyset$;

$Q := V(G)$

<u>while</u> not empty(Q) <u>do</u>

{ extract node u with shortest path from Q

$S := S + \{u\}$

<u>for</u> each node v adjacent to u <u>do</u>

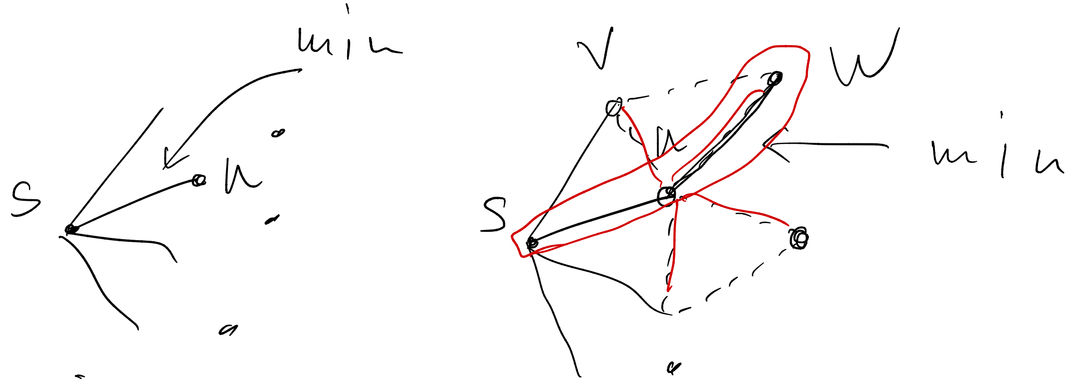
Relax(u, v, w)

}

Πολυπλοκότητα $O(V^2 + E) = O(V^2)$

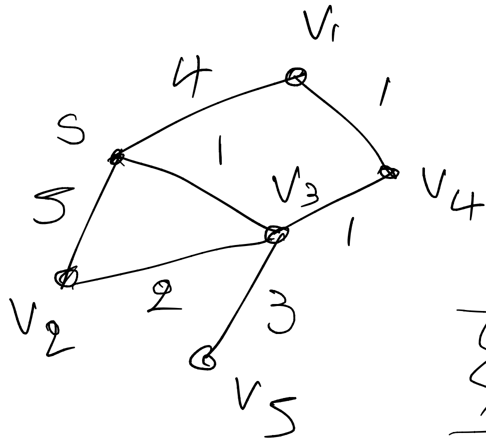
Χαρακτηριστικά του αλγόριθμου Dijkstra

- $n-1$ φάσεις
- σε κάθε φάση ένα shortest path
- greedy τεχνική
- δεν δουλεύει αν έχουμε αρνητικά βάρη
- σε κάθε φάση i υπολογίζει ένα shortest path (s, v_i)



$i = 1$ $\quad$ $i = 2$

— σε κάθε φάση i υπολογίζει όλα τα shortest paths που επιτρέπεται να περάσουν από κάθε $v \in S_{i-1}$, όπου S_{i-1} περιέχει τους κόμβους των οποίων τα shortest paths έχουν βρεθεί πριν τη φάση i



	v_1	v_2	v_3	v_4	v_5
S	4	5	1	∞	∞
$\{s, v_3\}$	4	3^{v_3}		2^{v_3}	4^{v_3}
$\{s, v_3, v_4\}$	3^{v_4}	3^{v_3}			4^{v_3}
$\{s, v_3, v_4, v_1\}$					
$\vdots$					

$$\delta(s, v_3) + w(v_3, v_2) < d(s, v_2)$$

Έστω το εξής πρόβλημα

$x_1 - x_2 \le 0$

$x_2 - x_4 \le 5$

$x_3 - x_5 \le -7$

⋮

Υπάρχουν τιμές για τα x_i που ικανοποιούν τις ανισώσεις?

τα x_i μπορεί να αναπαριστούν γεγονότα και κάθε ανίσωση να αναπαριστά μια συνθήκη για τα γεγονότα x_i, x_j

π.χ., $x_i - x_j \le 5$ ή $x_i - x_j \le -3$

$(x_j \ge x_i + 3)$

Κατασκευάζουμε ένα γράφημα G ως εξής:

$\forall\, x_i \rightarrow$ παίρνω κορυφή v_i

$(v_i, v_j) \in E(G)$ iff $x_j - x_i \leq B_k$

Προσθέτουμε κορυφή v_0 και ακμές

$(v_0, v_j)\ \forall\, v_j \in G$

$w(v_0, v_j) = 0$

$w(v_i, v_j) = B_k$

$\left.\begin{array}{l} x_1, \dots, x_n \\ m \text{ ανισώσεις} \end{array}\right\} \rightarrow \begin{array}{l} n+1 \text{ κορυφές} \\ n+m \text{ ακμές} \end{array}$

$x_1 - x_2 \leq 0$

$x_1 - x_5 \leq -1$

$x_2 - x_5 \leq 1$

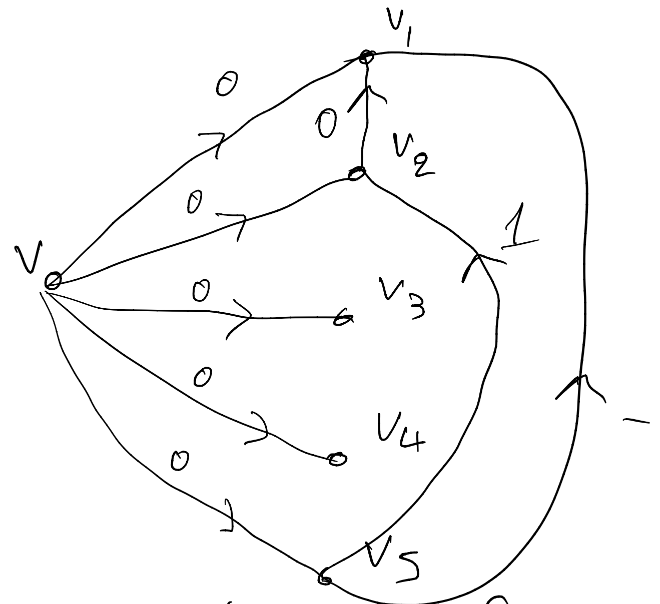
$x_3 - x_1 \leq 5$

$x_4 - x_1 \leq 4$

$x_4 - x_3 \leq -1$

$x_5 - x_3 \leq -3$

$x_5 - x_4 \leq -3$



Αν το γράφημα δεν περιέχει κύκλους με αρνητικό βάρος τότε το $x_i = sp(v_0, v_i)$ είναι εφικτή λύση αλλιώς δεν υπάρχει λύση.

Ασκήσεις

Arbitrage:

$c_1, \dots, c_n$ νομίσματα

$$R = \begin{bmatrix} \quad \end{bmatrix}_{n \times n}$$

$1 \; c_i$ αγοράζει $R_{ij} \; c_j$

Δώστε αλγόριθμο που υπολογίζει αν υπάρχει ακολουθία $\langle c_{i_1}, c_{i_2} \dots c_{i_k} \rangle$:

$$R_{i_1 i_2} \cdot R_{i_2 i_3} \cdots R_{i_k i_1} > 1$$

Αλγόριθμος Floyd-Warshall

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & , \; k=0 \\ \min\left(d_{ij}^{(k-1)}, \; d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & , \; k \geq 1 \end{cases}$$

↓ shortest path από το i στο j όπου όλοι οι ενδιάμεσοι κόμβοι ανήκουν στο $\{1, \ldots, k\}$

- πολυπλοκότητα $O(V^3)$
- λειτουργεί με αρνητικά βάρη

- Εισαγωγή στους Αλγόριθμους,
  T. Cormen, C. Leiserson, R. Rivest,
  C. Stein

- csacademy.com/ieeextreme-practice/contest/archive
  tag → Graphs