Curs #11

1) Single Source Shortest path
   - Dijkstra
   - Bellman-Ford
   - SSSP in DAG
2) All Pairs Shortest Paths
   - Floyd-Warshall
   - Transitive closure

I. Shortest Path problem definition

- $G(V, E)$ ⇝ graf ponderat
- „path" ~ drum = secvență de vf. legate de muchii
- the shortest path weight from u to v:

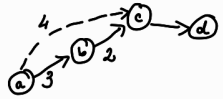
$$\delta(u,v) = \begin{cases} \min\{w(p)\} \text{ if } \exists p \le u, \dots, v \ge \\ \infty, \text{altfel} \end{cases}$$

↳ cel mai mic drum de la u la v este orice drum p pt care $w(p) = \delta(u,v)$

Lemma

Given a weighted graph $G = (V, E)$ with $w : E \to \mathbb{R}$, let $p = < v_0, v_1, \dots v_k >$ be a shortest path from v_0 to v_k. For any i, j, $0 \le i \le j \le k$, the subpath $p_{ij} = < v_i, v_{i+1}, \dots v_j >$ is a shortest path from v_i to v_j.

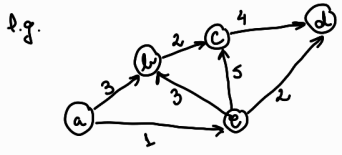
obs: dacă drumul e S.P. ⇒ toate sub-drumurile sunt shortest



obs 2: dacă ∃ subpath mai mic ⇒ drumul $v_0 \rightsquigarrow v_k$ îl va folosi

obs 3: dacă drumul este un shortest path ⇒ TOATE sub-path-urile sunt shortest path

↳ invers NU este adevărat



e.g. <a,b>, <b,c>, <c,d>, <a,b,c> , <b,c,d> sunt shortest paths

DAR <a,b,c,d> NU e sh. path ↓ <a,e,d> e S.h. path

Representing Shortest path

- $v.d$ = weight of the s.p. from source s to v
- $v.\pi$ = parent / predecessor in s.p. from s to v

INITIALIZE-SINGLE-SOURCE (G, s)

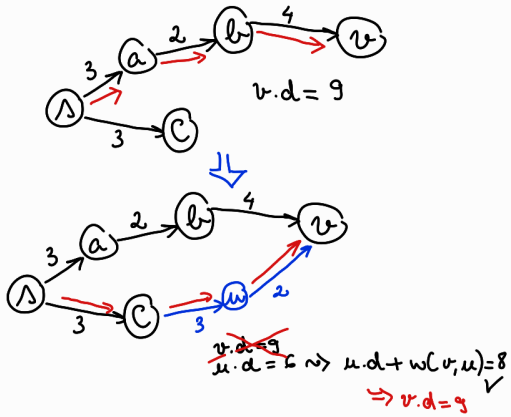
```
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

Relaxation technique

- se încearcă îmbunătățirea s.p. găsită până în acel moment.
- putem găsi un drum mai scurt de la s până la v trecând prin u?

RELAX

```
if v.d > u.d + w(u,v)
    v.d = u.d + w(u,v)
    v.π = u
```

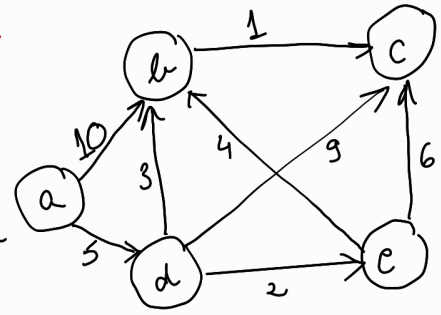


Dijkstra's Algorithm

Dijkstra's Algorithm

DIJKSTRA(G, w, s) ⇝ w = weight, s = source

```
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅                    → shortest path
3  Q = G.V                  → queue
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```



	π	d
a	NIL	0
b	NIL	∞
c	NIL	∞
d	NIL	∞
e	NIL	∞

Q

ex: shortest path from a

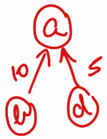
Extract Min => a~0

	π	d
a	NIL	0
b	NIL	∞
c	NIL	∞
d	NIL	∞
e	NIL	∞

S = {a}; Q = {b, c, d, e}

Relax ⇒

	π	d
a	NIL	0
d	a	5
b	a	10
c	NIL	∞
e	NIL	∞



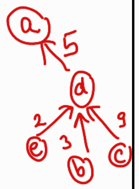
Extract Min => d

	π	d
a	NIL	0
d	a	5
b	a	10
c	NIL	∞
e	NIL	∞

S = {a, d}; Q = {b, c, e}

Relax ⇒

	π	d
a	NIL	0
d	a	5
e	d	7
b	d	8
c	d	14

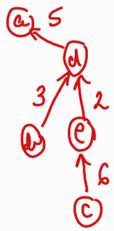


	π	d
a	NIL	0
d	a	5
e	d	7
b	d	8
c	d	14

S = {a, d, e}; Q = {b, c}

Relax ⇒

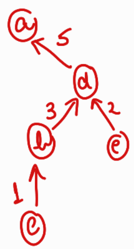
	π	d
a	NIL	0
d	a	5
e	d	7
b	d	8
c	e	13



	π	d
a	NIL	0
d	a	5
e	d	7
b	d	8
c	e	13

Relax ⇒

	π	d
a	NIL	0
d	a	5
e	d	7
b	d	8
c	b	9



	π	d
a	NIL	0
d	a	5
e	d	7
b	d	8
c	b	9

End

DIJKSTRA – COMPLEXITY

PRIORITY Queue → regular array / binary heap / fibonacci heap

Dijkstra's Algorithm

DIJKSTRA(G, w, s)

Line	r.a	b.h	f.h
INITIALIZE-SINGLE-SOURCE(G, s)	$O(V)$	$O(V)$	$O(V)$
$S = \emptyset$			
$Q = G.V$	$O(V)$	$O(V)$	$O(V)$
while $Q \neq \emptyset$	$\lvert V\rvert$ st.	$\lvert V\rvert$	$\lvert V\rvert$ steps
u = EXTRACT-MIN(Q)	$O(V)$	$O(\log V)$	$O(\log V)$
$S = S \cup \{u\}$			
	după ce	iterăm	prin toate V
for each vertex $v \in G.Adj[u]$	$\lvert E\rvert$ total	$\lvert E\rvert$ total	$\lvert E\rvert$ total
RELAX(u, v, w)	$O(1)$	$O(\log V)$	$O(1)$
+	$O(V \cdot V + E \cdot 1) = O(V^2)$	$O(V\log V + E\log V) = O(E\log V)$	$O(V\log V + E \cdot 1) = O(V\log V + E)$

e sau nu O(1)? NU trebuie refăcut heap-ul!

?

→ dacă modific dist => => structura P.Q se schimbă
↳ updating distance → heap decrease key
↳ descreșterea unei chei din heap => merge în sus pe heap.

Deci
r.a => O(1) => doar swap.
b.h => O(log V) => mă duc pe o ramură în b.h

regular Array: $O(V^2)$ → e cea mai proastă tot timpul

Binary Heap: $O(E \log V)$

Fibonacci Heap: $O(E + V\log V)$

↓ WORST CASE for E

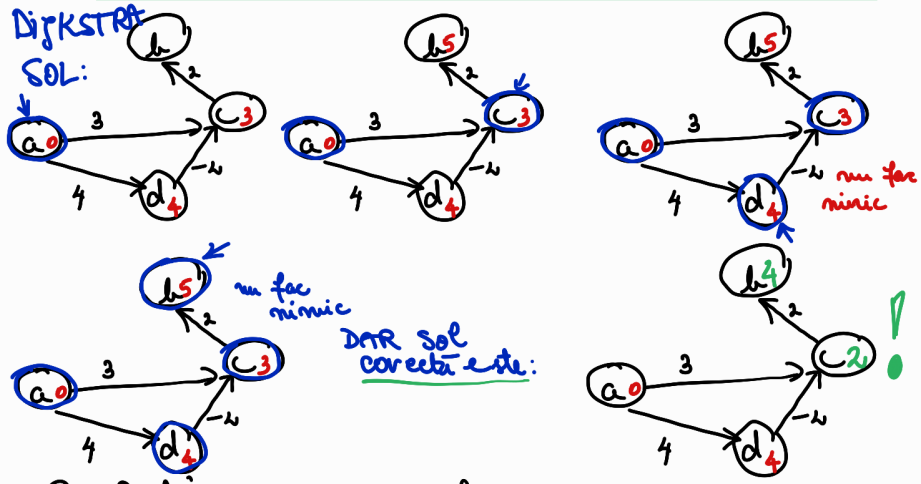
$E = \frac{V(V-1)}{2}$

↓ pt B.H/F.H. e mai ineficient w.c. for E => deci folosesc R.A.

Grafuri → rare ($V/\log V = E$) ⇝ B.H
Grafuri → dense ($\sim V^2 = E$) ⇝ R.A

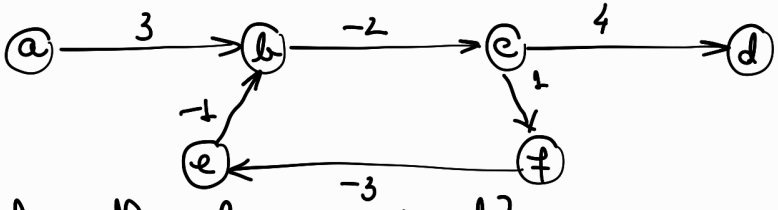
Când nu funcționează DIJKSTRA?

1) ⇝ DACĂ AVEM MUCHII DE COST NEGATIV?



Soluție: mai multe relaxări prin noduri

2) ⇝ Dacă avem cicluri negative?



Shortest path from a to d?

$<a,b,c,d> = 5$

$<a,b,c,f,e,b,c,d> = 2$

$<a,b,c,f,e,b,c,f,e,b,c,d> = -1$

$\vdots \dots >$ mereu o să scadă ....

silly obs :) ⇝ dacă am avea ciclu cu w. poz ar crește deci Dijkstra nu l-ar lua în considerare

Bellman-Ford approach $\rightsquigarrow O(V \cdot E)$

⇝ mai lent decât Dijkstra

⇝ facem $|V|-1$ pași

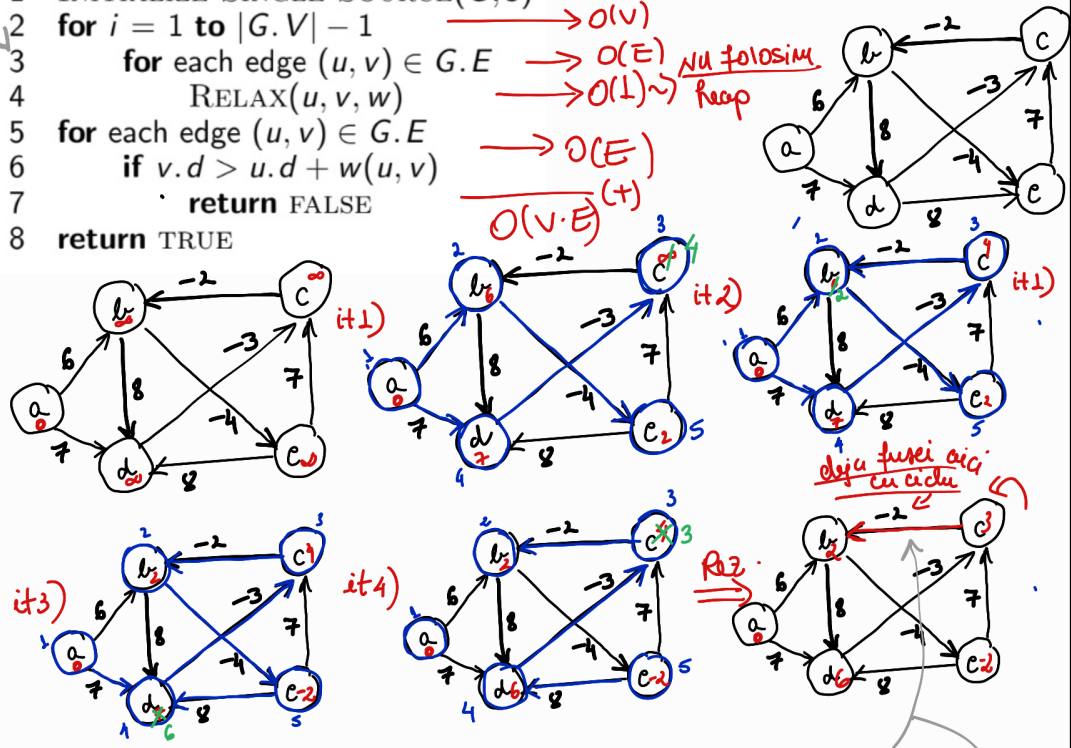
pt fiecare pas ⇝ apelăm RELAX pe fiecare muchie

⇝ rezolvă pb. ponderilor negative

Bellman-Ford verifică dacă alg. are sau nu cicluri negative

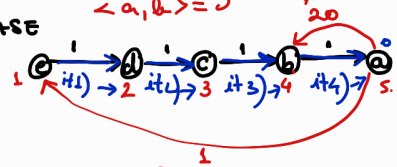
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)            → O(V)
2  for i = 1 to |G.V| − 1                    → O(E)   (NU folosim heap)
3      for each edge (u, v) ∈ G.E            → O(1) ⇝
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E                → O(E)
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

$O(V \cdot E)$ (+)



De ce $|V|-1$ pași?

eg. WORST CASE




```
for each edge ∈ G.E
    if v.d > u.d + w(u,v)
        return false
return true
```

Complexity: $O(V \cdot E)$

SSSP în DAGs

Shortest subpath S? problem ↳ graf orientat fără cicluri
⇝ sorted ...?

- DAG = Directed Acyclic Graph (graf orientat fără cicluri)
- no cycles ⇒ no negative weight cycles
- the vertices can be topologically sorted ⇝ DFS = (V+E) ⇝ O(V+E)
  ⇝ dacă ∃ u⇝v, u vine înainte de v
- SSSP ⇝ luăm vf în ordinea sort. topologice
  → trecem o singură dată prin vf. sortate topologic ⇝ relax pt fiecare muchie

⇒ $O(V+E)$

```
DAG-SHORTEST-PATHS(G, w, s)
  topologically sort the vertices of G // see seminar     -> O(V+E)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for each vertex u, taken in topologically sorted order  -> |V|     } V+E
      for each vertex v ∈ G.Adj[u]                        -> |E| în total
          RELAX(u, v, w)
```

All-Pairs Shortest Paths

⇝ drumul minim între fiecare pereche de vârfuri
↳ rezultă o matrice

⇝ vertices: 1, 2, ... n

⇝ approach = dynamic programming

W = matrice de adiacență ponderată

$$w_{ij} = \begin{cases} 0, & i=j \\ \text{weight edge}, & i\neq j \text{ și } \exists \text{ edge}(i,j) \\ \infty, & \text{altfel} \end{cases}$$

- Simple approach ⇝ SSSP on every node

Floyd-Warshall Algorithm

- $d_{ij}^{(k)}$ ⇝ weight of shortest path de la i la j pt care fiecare nod intermediar e în setul $\{1, 2, \dots, k\}$

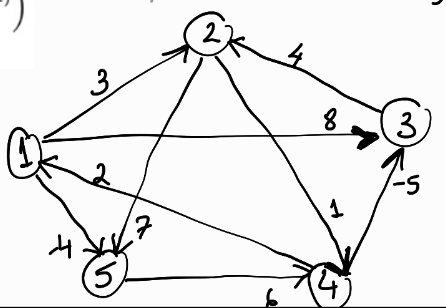
$$d_{ij} = \begin{cases} w_{ij}, & k=0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{jk}^{(k-1)}\right), & k\geq 1 \end{cases}$$

- dynamic programming approach
- $D^{(n)} = (d_{ij}^{(n)})$ ⇝ dă răsp. final ⇒ $d_{ij}^{(n)} = \delta(i,j)$

```
FLOYD-WARSHALL(W)
  n = W.rows
  D^(0) = W
  for k = 1 to n                                        -> O(n)
      let D^(k) = (d_ij^(k)) be a new n × n matrix             } O(n^2)
      for i = 1 to n                                    -> O(n)
          for j = 1 to n                                -> O(n)
              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
                  if (d_ij > d_ik + d_kj)
                      d_ij = d_ik + d_kj
  return D^(n)
```

Complexity:
- TIME: $O(V^3)$
- SPACE: $O(V^3)$ ⇝ DAR optimizat ↳ $O(V^2)$

Complexity?
$O(n^3)$ (TIME)
$O(n^3)$ → (Space) → $O(n^2)$
↳ pe alg dat
DAR
↳ e suficient să avem doar mat. Strc
↳ o singură mat D unde pot adăuga modificările
$O(n^2)$



$D^{(0)}$

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

$D^{(1)}$

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

1 ↝ 2 ↝ 5
4 ↝ 1 ↝ 2
4 → 1 → 2
4 ↝ 1 ↝ 5

$D^{(2)}$

	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

↝ mă uit la 2 ↝ 5 ↝ 4
doar pe cel 5,4 pot face modif. ($\neq \infty$)
+ trb să fie drum $(2,j) \neq \infty$
1: 1-2-4, 1-2-5
3: 3-2-4, 3-2-5
4:
5 ✗

$D^{(3)}$

	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	∞	∞	∞	6	0

1-3-2 = 8 + 4 | 4-3-2 = -5 + 4 = -1
2-3-2 ✗ | 5-3-2 = ✗ ↝ am inf.

$D^{(4)}$

	1	2	3	4	5
1	0	3	-1	4	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

↝ mă uit după $(4,j) \neq \infty \Rightarrow 1,2,3,5$.

$D^{(5)}$

	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

Reconstrucția drumului pentru FLOYD-WARSHALL

$O(V^3)$ ↪ $O(V^3+V)$

- build matrix of parents $\Pi = (\pi_{ij}) \rightsquigarrow$ părinții

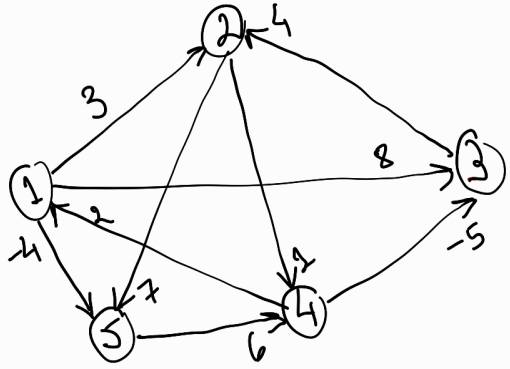
$$\pi_{ij}^{(0)} = \begin{cases} NIL & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

↪ $\Pi = \Pi^{(n)}$ ↝ o să fie res finală

FLOYD-WARSHALL-2(W) → $O(V^3)$

```
1  n = W.rows
2  for i = 1 to n
3      for j = 1 to n
4          d_ij = w_ij
5          if i == j or w_ij == ∞
6              π_ij = NIL
7          else π_ij = i
8  for k = 1 to n
9      for i = 1 to n
10         for j = 1 to n
11             if d_ij > d_ik + d_kj
12                 d_ij = d_ik + d_kj
13                 π_ij = π_kj
14 return D, Π
```



PATH-CONSTRUCTION(Π, i, j)

```
1  if π_ij == NIL
2      return NIL
3  path = {j}
4  while j ≠ i
5      j = π_ij
6      path = {j} ∪ path
7  return path
```

→ $O(V)$

$i=2, j=5$

$\Pi =$

	1	2	3	4	5
1	NIL	3	4	5	1
2	4	NIL	4	2	1
3	4	3	NIL	2	1
4	4	3	4	NIL	1
5	4	3	4	5	NIL

$$\Pi = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} NIL & 3 & 4 & 5 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{pmatrix} \end{array}$$

$i=2; j=\neq 1$ | $j=\Pi[2][1]=4$ | $j=\Pi[2][4]=2$

path $=5$ | path $=1,5$ | path $=4,1,5$

$j=\Pi[2,2] \rightsquigarrow$ stop

path $=\{2,4,1,5\}$

Inchidere tranzitivă a unui Graf

$\rightsquigarrow$ Similar FLOYD - WARSHALL

Definition

Given a directed graph $G = (V, E)$, with $V = \{1, 2, \ldots, n\}$, we define the **transitive closure** as $G = (V, E^*)$, where $E^* = \{(i, j) : \exists i \rightsquigarrow j\}$.

$\rightsquigarrow$ graf direct

$\leadsto$ transitive closure as $G = (V, E^*)$ where $E^* = \{i,j : \exists i \rightsquigarrow j\}$

$\rightsquigarrow$ există muchie de la i la $j \Leftrightarrow (\exists)$ drum în graf de la i la j

- dacă vrem să știm dacă $\exists$ drum de la i la j pt toate perechile
- Sol 1: F.W. cu $w_{ij} = 1\ (\forall)(i,j) \in E$ și verificăm dacă $d_{ij} < \infty$
- Sol 2: schimbăm op. aritmetice în operatori logici

$$t_{ij}^{(0)} = \begin{cases} 0, & i \neq j \text{ și } (i,j) \notin E \\ 1, & i = j \text{ sau } (i,j) \in E \end{cases}$$

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)} \right), \ k \geq 1$$

TRANSITIVE-CLOSURE(G)

```
1  n = |G.V|
2  let T = (t_ij) be a new n × n matrix
3  for i = 1 to n
4       for j = 1 to n
5            if i == j or (i, j) ∈ G.E
6                 t_ij = 1
7            else t_ij = 0
8  for k = 1 to n
9       for i = 1 to n
10           for j = 1 to n
11                t_ij = t_ij ∨ (t_ik ∧ t_kj)
12 return T
```