Curs Săptămâna 10

Grafuri: MST, BFS

$G = (V, E)$ { V - nod/vârf; E - muchie

Reprezentare:

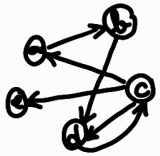
→ **matrice de adiacență**
↳ muchia aparține graficului? $O(1)$
↳ Care sunt vecinii unui nod?
↳ parcurgem linia coresp. $O(n)$

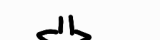
→ **liste de adiacență**

Concepte - Terminologie

- Orientare:

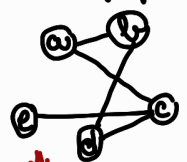
directed graph





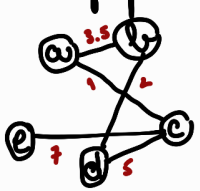
(∃) self edges.

undirected graph



↓ matr. de adiacență simetrică

weighted graph



⇒ dacă (∄) ⇒ ∞

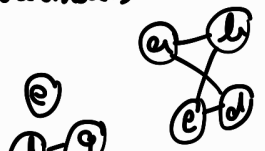
- Gradul unui nod
  ↳ gf. orientat.
  ↳ pt graf. neorientat → grad intern (ce intră)
  → grad extern (ce iese)

- **graf Complet** ⇒ (∃) muchie între orice nod
  - **neorientat**: $\frac{n(n-1)}{2}$ muchii
  - **orientat**: $n(n-1)$ muchii

- **distanța între 2 noduri** = cel mai scurt drum între 2 noduri

- **diametrul unui graf** = lungest shortest path
  ↳ cea mai lungă distanță între 2 noduri din graf

- graf conex ⇒ „putem ajunge de oriunde oriunde :)"
  ↳ altfel:
  Componente conexe
  ↳ pt graf neorientat



↳ pt graf. orientat
Strongly connected graph
→ există drum de la A la B și de la B la A

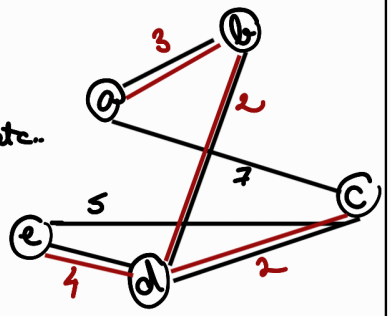
Minimum Spanning Tree

MST

• Kruskal's Algorithm

Se dă graf neorientat, conex, ponderat și care să păstrăm toate nodurile și un subset de muchii a.î. să rămână conex și weight-ul să fie minim → + fără cicluri (= arbore dar fără rădăcină)

Aplicații: network design, rețea electrică/telefonie, etc.



Kruscal's Algorithm – Approach

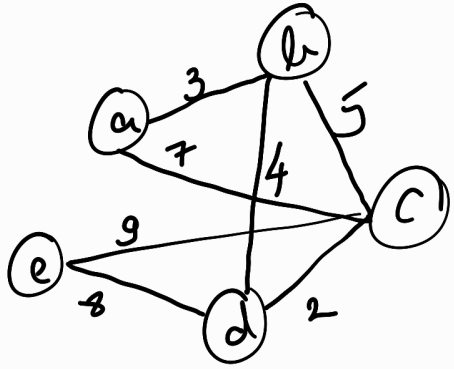
- greedy approach
- la început fiecare nod e izolat
  - sortăm muchiile după cost.
    - ↳ adăugăm sau nu în MST?
      - ↳ dacă închide ciclu ⇒ NU
      - ↳ cum decidem dacă închide sau nu ciclu? Disjoint sets!
    - ↳ subarborele e distinct
- ne oprim atunci când avem |V|-1 muchii adăugate în disjoint set

```
MST_K
  A = ∅
  for each vertex v ∈ G.V                    } O(V)
      MAKE_SET(v)
  sort the edge G.E by w                     } O(E log E)
  for each edge (u,v) ∈ G.E                  → E ori
      if FIND_SET(v) ≠ FIND_SET(u)           } O(d(v))
          A = A ∪ {(u,v)}                    ⇓
          UNION(u,v)
  return A;
```

union by rank + path compression → d(v) ~ O(1)

Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges G.E by weight w
5  for each edge (u, v) ∈ G.E
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```



G.E = {(a,b)3, (a,c)7, (b,c)5, (b,d)4, (c,d)2, (c,e)2, (c,e)9, (d,e)8}

A = { }

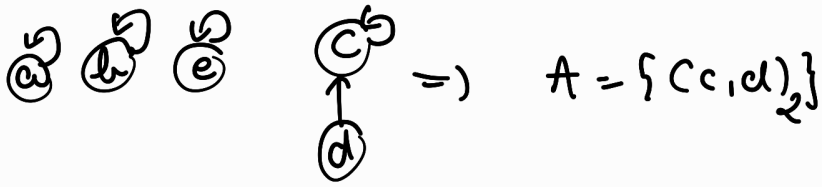
p1)

A = {..}

p2)

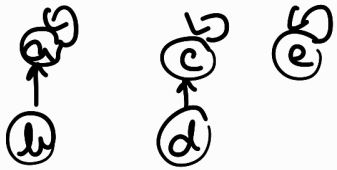
G.E → sortare muchii → c.s {b, d} preferabil

G.E = {(c,d)2, (a,b)3, (b,d)4, (b,c)5, (a,c)7, (d,e)8, (c,e)9}



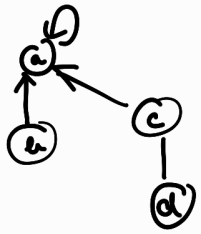
$\Rightarrow A = \{(c,d)_2\}$

$(a,b) \Rightarrow$



$A = \{(c,d)_2, (a,b)_3\}$

(b,d)

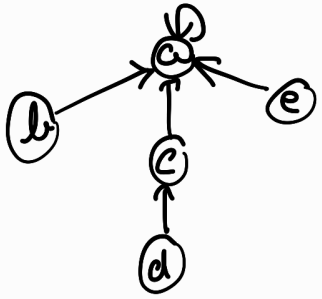


$A = \{(c,d)_2, (a,b)_3, (b,d)_4, \}$

$(b,c) \Rightarrow FindSet(b) = FindSet(c)$ X

$(a,c) \Rightarrow FS(a) = FS(c)$ X

$(d,e) \Rightarrow FS(d) = a \neq FS(e) = e \Rightarrow$



$\Rightarrow$ teoretic aș putea să îl fac să se oprească

are N-1 muchii $\Rightarrow$ am calculat deja MST

$\hookrightarrow$ STOP.

$\hookrightarrow$ la implementare mă opresc!

$A = \{(c,d)_2, (a,b)_3, (b,d)_4, (d,e)_8\}$

Kruskal $\rightarrow$ Complexity

$O(E \log E)$ $\rightarrow$ datorită de sortare

- Discussion:
  - $\rightsquigarrow$ soluția e unică?
    - $\hookrightarrow$ NU:

      graful cu cele mai multe MST:

      graf complet și toate muchiile au aceeași pondere .. factorial
  - $\rightsquigarrow$ Corectitudine?

*exam!

Kruskal's Algorithm - Correctness

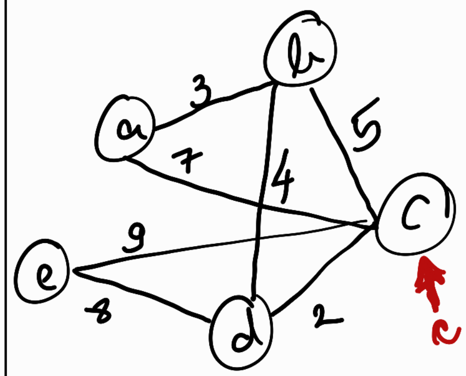
- Let T be the MST built by MST-KRUSKAL.
- Consider $G.E = \{e_1, e_2, \ldots, e_m\}$ the edges of the graph, ordered ascendingly by weight.
- Assume that T is not minimal, and that T^* is the MST ($w(T^*) < w(T)$) having the longest common edge prefix with T.
- Let $e \in G.E$ be the first edge (considering their order) which doesn't belong to $T.E \cap T^*.E$
  - Kruskal would not have excluded that edge, unless it closed a cycle
  - $\Rightarrow e \in T.E$ and $e \notin T^*.E$
- If $e = (x, y)$, there is a path in T^*, P, from x to y (which cannot be the direct edge, because, $(x, y) \notin T^*.E$).
- If all edges in P had a weight smaller than e, Kruskal-MST would have selected them prior to e (e is the first edge for which T and T^* "disagree"), and e would close a cycle.
- $\Rightarrow \exists e' \in P, w(e) \leq w(e')$.
- Let $T_1 = T^* \cup \{e\} \setminus \{e'\}$. $\Rightarrow w(T_1) \leq w(T^*)$.
- T_1 is a MST with a longer common prefix with T than T^* - contradiction.

Algoritmul lui Prim

→ greedy approach
→ construiește un arbore cu rădăcină
→ pornești de la un nod oarecare
↳ în fiecare pas (out of the |V|-1 steps)
↳ selectează cel mai apropiat nod de nodul curent
↳ adaugă nodul și muchia corresp. la arbore

```
PRIM(G,w,r)
(1) { for each u ∈ G.V.
        u.key = ∞            } O(V)
        u.π = NIL
r.key = 0
Q = G.V  //priority queue    } O(V) → build heap....
while Q ≠ 0                  → V
    u = EXTRACT_MIN(Q)
    for each v ∈ G.Adj[u]    → suma list de adiacență ⇒ O[E]
        if v ∈ G and w(u,v) < v.key   ← E
            v.π = u
            v.key = w(u,v)   → trb să se refacă prop. de heap
                               Heap decrease Key
                               ⇓
                               { Binary H ⇒ O(log V)
                               { Fibonacci ⇒ O(1)
```

↳ pt { binary ⇒ O(E log V)
{ Fibonacci ⇒ O(E + V log V)



(1) ⇓ π

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

} Q

produc K și P același arbore pt ex. ăsta !
⇒ nu (∃) alt MST.

costuri distincte ⇒
⇒ are un sg. MST

de examen! oral!

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

⇒

	π	Key
c	NIL	0
a	c	7
b	c	5
d	c	2
e	c	9

} Q ⇓ priority Queue

1) ⇒ extrag c ⇒ 4) parcurg vecinii
⇓
3) găsesc o cheie mai mică?

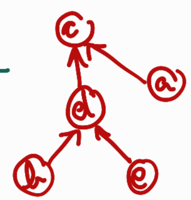
⇒ deja avem arbore acoperire deci nu e de cost minim

	π	Key
c	NIL	0
d	c	2
b	~~c~~ d	~~5~~ 4
a	c	7
e	~~c~~ d	~~9~~ 8

→ extract

} Q

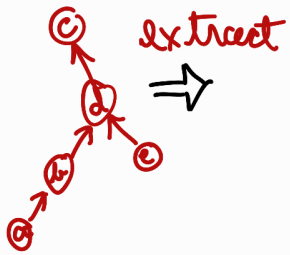
nu (∃) (a,d)



extract min

	π	Key
c	NIL	0
d	c	2
b	d	4
a	~~b~~	~~3~~
e	d	8

extract ⇒



	π	Key
c	NIL	0
d	c	2
b	d	4
a	b	3
e	d	8

}2

extract ⇒

	π	Key
c	NIL	0
d	c	2
b	d	4
a	b	3
e	d	8

// ⇒ Q = NULL

Complexity: { Priority Queue:
Fibonacci Heap: $O(E + V \log V)$
Binary Heap: $O(E \log V)$

worst case • $E \Rightarrow O(V^2)$
• $\log(V^2) \Rightarrow O(\log V)$

Avantaje Prim over Kruskal

Răspunde }
- constr. un arbore cu rădăcină
- mai eficient cu heap Fibonacci
- Soluția e unică?
- E algoritm deterministic?

Corectitudine:

Prim's Algorithm - Correctness

- Let T be the MST build by MST-Prim.
- Let $T.E = \{e_1, e_2, \ldots, e_{n-1}\}$ be the sequence of edges, in the order they are selected by the algorithm.
- Assume that T is not minimal, and that T^* is the MST, $(w(T^*) < w(T))$, which includes the longest prefix $E' = \{e_1, e_2, \ldots, e_{i-1}\}$ from $T.E$, while the edge $e_i = (x, y) \notin T^*.E$. We denote by T' the nodes added to T before the edge e_i.
- In T^* there is a path from x to y. As $y \notin T'$, there is in the path from x to y an edge (a, b) with $a \in T'$ and $b \notin T'$.
- Let $T_1 = T^* \cup \{(x, y)\} \setminus \{(a, b)\}$ - also a spanning tree.
- We have 3 cases:
  - if $w(x, y) < w(a, b)$, then $w(T_1) < w(T^*) \Rightarrow T^*$ is not minimal
  - if $w(x, y) > w(a, b)$, then MST-Prim would have selected (a, b) instead of (x, y)
  - if $w(x, y) = w(a, b)$, then $w(T_1) = w(T^*)$, so T_1 is minimal and contains a longer prefix of $T.E$
- All three cases result in a contradiction, so T^* does not exist $\Rightarrow T$ is minimal.

Breadth-First Search
BFS
-> parcurgere în lățime

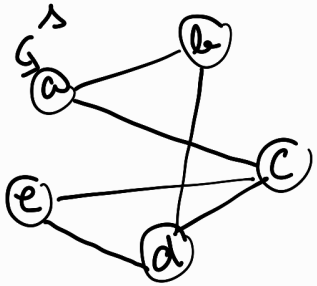
- principiu: • i se dă un nod s
  - produce un arbore cu rădăcină în s
  - toate nodurile aflate la distanța K de la sursă vor fi descoperite la dist K+1 ???
  - arborele rezultat reprezintă drumul de dist minimă de la s la toate nodurile

- nodurile au atributele:
  - color: • WHITE ⇒ nu am ajuns la el
    - GRAY ⇒ am ajuns la el, dar încă nu am terminat
    - BLACK ⇒ am terminat cu el

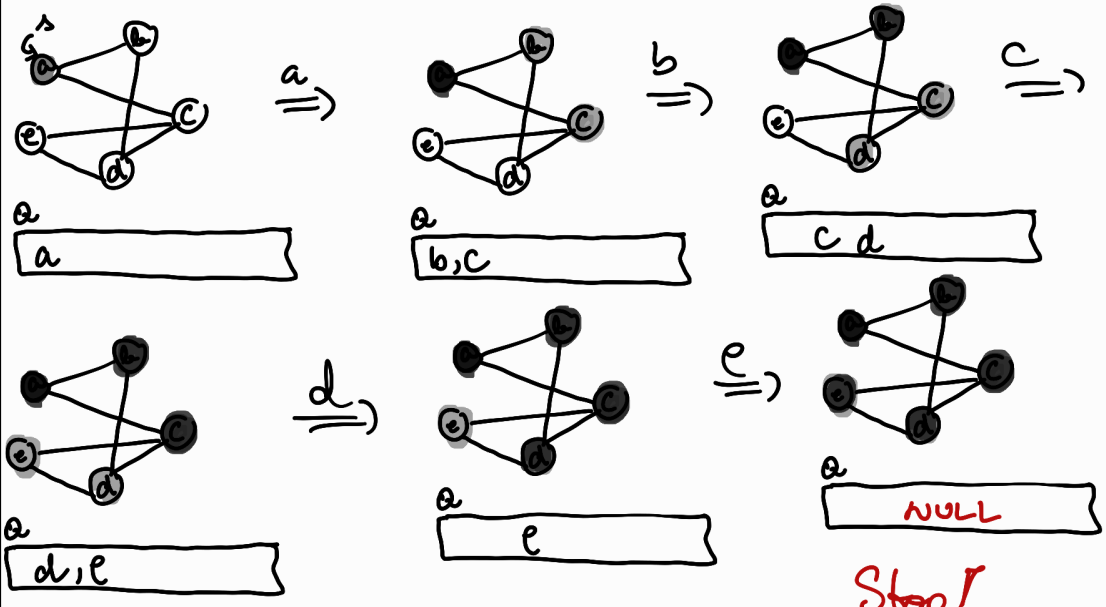
- algoritmul începe prin adăugarea lui s în coadă și colorarea lui cu Gray
- extragem din coadă nodul x
  - ↳ îi colorăm vecinii cu Gray dacă sunt white
  - ↳ colorez nodul extras cu Black

BFS Algorithm

```
BFS(G, s)
1  for each u ∈ G.V − {s}          } O(V)
2      u.color = WHITE
3      u.d = ∞
4      u.π = NIL
5  s.color = GRAY                  } O(1)
6  s.d = 0
7  s.π = NIL
8  Q = ∅ // regular queue
9  ENQUEUE(Q, s)
10 while Q ≠ ∅                     -> |V| steps
11     u = DEQUEUE(Q)
12     for each v ∈ G.Adj[u]       -> |E| total -> pt fiecare vecin al tuturor nodurilor
13         if v.color == WHITE
14             v.color = GRAY
15             v.d = u.d + 1
16             v.π = u
17             ENQUEUE(Q, v)
18     u.color = BLACK
```



Q



Complexitate BSF

- Liste: $O(V+E)$ → graf CONEX: $O(E)$ e cel puțin $O(V)$
  → graf neconex: ? → nu ajunge la toate nodurile
- Matrice de adiacență: $O(V^2)$

? • Cum folosim bfs ca să dat diam. unui arbore?

BSF / LEE => dist. shortest path in a labirinth

Algoritmul lui Lee

LEE

Cum reconstituim drumul? Dest ⟶ sursă.

Reconst Shortest path de la a la n

! plec pe leg. părinte