

Arbore

→ Un arbore cu n vârfuri are $n-1$ muchii

Arbore Partițiali Cost minim. APCM

Algoritmi:

Arbore partițiali

Proprietate

Orice graf neorientat conex conține un arbore partițial

Demonstrație – două tipuri de algoritmi de construcție a unui arbore partițial al unui graf conex $G=(V,E)$:

Prin adăugare de muchii (bottom - up)	Prin eliminare de muchii (cut - down)
$T \leftarrow (V, \emptyset)$ cat timp T nu este conex executa <ul style="list-style-type: none">alege $e \in E(G) - E(T)$ care unește două componente conexe din T (nu formează cicluri cu muchiile din T)$E(T) \leftarrow E(T) \cup \{e\}$ returneaza T	$T \leftarrow (V, E)$ cat timp T conține cicluri executa <ul style="list-style-type: none">alege $e \in E(T)$ o muchie dintr-un ciclu$E(T) \leftarrow E(T) - \{e\}$ returneaza T
În final T este conex și aciclic, deci arbore	În final T este aciclic și conex (s-au eliminat doar muchii din ciclu), deci arbore

Grafuri

Graf orientat

$$G = (V, E)$$

V - finită

E - perechi (ordonate) de 2 elemente **din V** distincte

- $v \in V$ - **vârf**
- $e = (u, v) = uv$ - **arc**
 $u = e^-$ - **vârf initial / origine / extremitate initială**
 $v = e^+$ - **vârf final / terminus / extremitate finală**

Graf neorientat

$$G = (V, E)$$

V - finită

E - **submulțimi** de 2 elemente (distincte) din V

- $v \in V$ - **vârf / nod**
- $e = \{u, v\} = (u, v) = uv$ - **muchie**
 u, v - **capete / extremități**

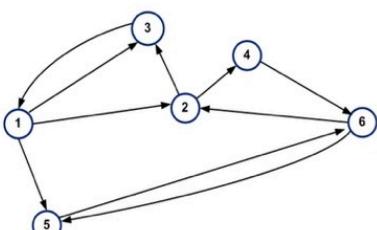
Graf orientat

• **grad interior** $d_G^-(u) = |\{e \in E \mid u \text{ extremitate finală pentru } e\}|$

• **grad exterior** $d_G^+(u) = |\{e \in E \mid u \text{ extremitate initială pentru } e\}|$

• **grad** $d_G(u) = d_G^+(u) + d_G^-(u)$

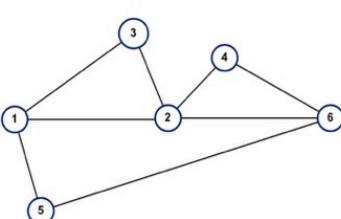
$$s^-(G) = \{d_G^-(v_1), \dots, d_G^-(v_n)\}$$



Graf neorientat

• **grad** $d_G(u) = |\{e \in E \mid u \text{ extremitate pentru } e\}|$

$$s^+(G) = \{d_G^+(v_1), \dots, d_G^+(v_n)\}$$



Grade – Proprietate

Graf orientat

$$\sum_{u \in V} d_G^-(u) = \sum_{u \in V} d_G^+(u) = |E|$$

Graf neorientat

$$\sum_{u \in V} d_G(u) = 2|E|$$

Idee demonstrație – Cu cât contribuie un arc / o muchie la sumă?

Grade multigraf neorientat

- $G = (V, E, r)$ – multigraf

$$d_G(u) = |\{e \in E \mid e \text{ nu este buclă, } u \text{ extremitate a lui } e\}| + \\ + 2 \cdot |\{e \in E \mid e \text{ este buclă, } u \text{ extremitate a lui } e\}|$$

Doară grafuri sunt izomorfe dacă:

Grafurile G_1 și G_2 sunt **izomorfe** ($G_1 \sim G_2$) \Leftrightarrow există $f : V_1 \rightarrow V_2$ bijectivă cu

$$uv \in E_1 \Leftrightarrow f(u)f(v) \in E_2$$

pentru orice $u, v \in V_1$

(f conservă adiacența și neadiacența)

Graf bipartit:

Graf bipartit

- Un graf neorientat $G = (V, E)$ se numește **bipartit** \Leftrightarrow
există o partiție a lui V în două submulțimi V_1, V_2 (bipartiție):

$$V = V_1 \cup V_2$$

$$V_1 \cap V_2 = \emptyset$$

astfel încât orice muchie $e \in E$ are o extremitate în V_1 și cealaltă în V_2 :

$$|e \cap V_1| = |e \cap V_2| = 1$$

Parcurgerea în lățime:

PSEUDOCOD:

Inițializări

pentru fiecare $x \in V$ execută

```
viz[x] = 0
tata[x] = 0
d[x] = ∞
```

BFS(s)

```
coada C = ∅
adauga(s, C)
viz[s] = 1; d[s] = 0
cat timp C ≠ ∅ execută
    i = extrage(C);
    afiseaza(i);
```

pentru j vecin al lui i ($ij \in E$)

```
daca viz[j]==0 atunci
    adauga(j, C)
    viz[j] = 1
    tata[j] = i
    d[j] = d[i]+1
```

COMPLEXITATE:

Timp: $O(n^2)$ n -nr de noduri

Spatiu: $O(n+m)$ n -nr de noduri
 m -nr de muchii

PROPRIETĂȚI

Muchiile folosite pentru a deschide
graful formează un arbore.

Muchiile care nu sunt în arbore
închid cicluri.

APLICAȚII:

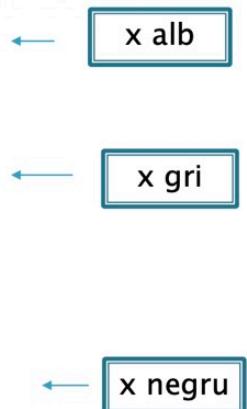
- nr elemente conexe
- Transmiterea unui mesaj în rețea
- distanțe și lățimi

→ distanța de la un nod la toate nodurile: graf neorientat.

Parcurgerea în adâncime:

PSEUDOCOD:

```
DFS(x)
    //incepe explorarea varfului x
    viz[x] = 1
    pentru fiecare xy ∈ E //y vecin al lui x
        daca viz[y]==0 atunci
            tata[y] = x
            d[y] = d[x]+1 //nivel, nu distanta
            DFS(y)
    //s-a finalizat explorarea varfului x
```



Apel:

```
DFS(s)
```

PROPRIETĂȚI:

→ Determinare cicluri

```

DFS(x)
    culoare[x] = gri
    timp = timp + 1
    desc[x] = timp; //incepe explorarea varfului x
    pentru fiecare xy ∈ E //y vecin al lui x
        daca culoare[y]==alb atunci
            tata[y] = x
            d[y] = d[x]+1 //nivel, nu distanta
            DFS(y)
    culoare[x] = negru
    timp = timp + 1
    fin[x] = timp //s-a finalizat explorarea varfului x

```

Apel:

```

pentru fiecare x in V executa
    culoare[x] = alb
timp = 0
pentru fiecare x in V executa
    daca culoare[x] == alb atunci
        DFS(x)

```

Sortare Topologică.

Sunt top a lui g = „ordonarea vf a.î. dacă $uv \in E$ atunci u se află înaintea lui v”

Aplicații: → ordinea de calc. în proiecte în care intervin relații de dependență
→ dñumuri critice

PSEUDOCOD: I)

(dacă n-are
circuit)

Algoritm

```

cât timp |V(G)| > 0 execută
    alege v cu  $d^-(v) = 0$ 
    adaugă v în ordonare
     $G \leftarrow G - v$ 

```

Implementare - similar BF

- Pornim cu toate vârfurile cu grad intern 0 și le adăugăm într-o coadă
- Repetăm:
 - extragem un vârf din coadă
 - îl eliminăm din graf (= scădem gradele interne ale vecinilor, nu îl eliminăm din reprezentare)
 - adăugăm în coadă vecinii al căror grad intern devine 0

```

coada C = Ø;
adauga in C toate vârfurile v cu  $d^-(v)=0$ 

```

```

cat timp C ≠ Ø execută
    i ← extrage(C);
    adauga i in sortare

```

```

    pentru ij ∈ E execută
         $d^-[j] = d^-[j] - 1$ 
        daca  $d^-[j]==0$  atunci
            adauga(j, C)

```

PSEUDOCOD: II)

```
Stack S  
  
DFS(i)  
    viz[i] = 1  
    pentru ij ∈ E  
        daca viz[j]==0 atunci  
            DFS(j)  
        //i este finalizat  
        push(S, i)  
  
    pentru i ∈ V execută  
        daca viz[i]==0 atunci  
            DFS(i)  
  
    cat timp S este nevida execută  
        u = pop(S)  
        adaugă u în sortare
```

Kosaraju: Determinarea elementelor Tare conexe

Observații

Într-un graf orientat:

- Componentele tare conexe sunt vîrf-disjuncte
- Orice vîrf aparține unei (unice) componente tare-conexe
- Un arc poate să nu aparțină niciunei componente tare-conexe
- Componente tare conexe din $G =$ componente tare conexe din G^T

PSEUDOCOD:

Pasul 0. Construim G^T

```
stack S  
  
DFS(G, i)  
    viz[i] = 1  
    pentru ij ∈ E(G)  
        daca viz[j]==0 atunci  
            DFS(j)  
        push(S, i) //i este finalizat  
  
    pentru x ∈ V execută  
        daca viz[x]==0 atunci  
            DFS(G, x)
```

▶ Pasul 1. Parcurgem DFS graful G +
introducem într-o stivă S fiecare varf la momentul la care este finalizat
(pentru a obține o ordonare descrescătoare a varfurilor după timpul de finalizare)

```

marcăm toate vîrfurile ca fiind nevizitate
cat timp S este nevida
x = pop(S)
daca x este nevizitat atunci
DFS(GT, x)
afiseaza componenta tare conexă (formată cu
vîrfurile vizitate în DFS(GT, x))

```

▶ Pasul 2. Parcurgem DFS graful G^T considerând vîrfurile în ordinea în care sunt extrase din S (descrescătoare după timpul de finalizare de la Pasul 1):

COMPLEXITATE: $O(n+m)$: 2 parcurgeri + construcție G^T

Arborei parțiali de cost minim

Grafuluri ponderate

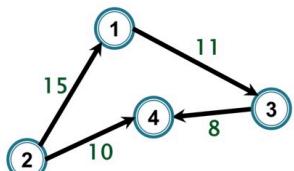
$G=(V,E)$ ponderat : $w:E \rightarrow \mathbb{R}$ funcție pondere

REPREZENTARE:

I

▶ Matrice de costuri (ponderi) $W = (w_{ij})_{i,j=1..n}$

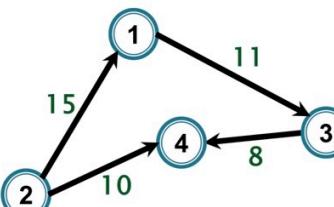
$$w_{ij} = \begin{cases} 0, & \text{daca } i = j \\ w(i,j), & \text{daca } ij \in E \\ \infty, & \text{daca } ij \notin E \end{cases}$$



0	∞	11	∞
15	0	∞	10
∞	∞	0	8
∞	∞	∞	0

II

▶ Liste de adiacență



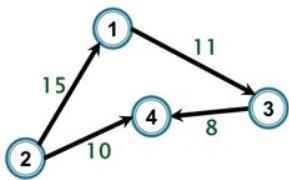
1: 3 / 11

2: 1 / 15, 4 / 10

3: 4 / 8

4:

- ▶ Liste de muchii/arce



1 3 11
2 1 15
2 4 10
3 4 8

Arborei parțiali de cost minim

Fie $G = (V, E, w)$ conex ponderat.

$\text{APCM}(G) = \text{AP}(G)$ cu prop $w(T_{\min}) = \min \{ w(T) \mid T \text{ AP al lui } G\}$

\parallel
 T_{\min}

Kruskal

- La un pas este selectată o muchie de cost minim din G care nu formează cicluri cu muchiile deja selectate (care unește două componente conexe din graful deja construit)

PSEUDOCOD:

```

sorteaza(E)
  for(v=1;v<=n;v++)
    Initializare(v);

  nrmsel=0

  for(uv ∈ E)
    if(Reprez(u) !=Reprez(v))
    {
      E(T) = E(T) ∪ {uv};
      Reuneste(u,v);
      nrmsel=nrmsel+1;
      if(nrmsel==n-1)
        STOP;
    }
  }

Initializare(int u){
  tata[u]=h[u]=0;
}

int Reprez(int u){
  while(tata[u]!=0)
    u = tata[u];
  return u;
}

Reuneste(int u,int v)
{
  int ru,rv;
  ru = Reprez(u);
  rv = Reprez(v);
  if (h[ru] > h[rv])
    tata[rv] = ru;
  else{
    tata[ru] = rv;
    if(h[ru] == h[rv])
      h[rv] = h[rv]+1;
  }
}
  
```

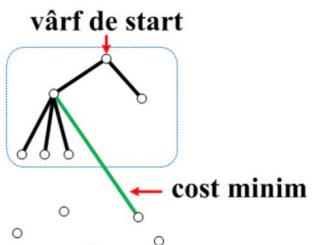
Complexitate: $O(m \log n)$

Prim

Prim

- La un pas:

Muchiile selectate formează un arbore



Este selectată o muchie de cost minim care unește un vîrf din arbore cu unul care nu este în arbore(neselectat)

Cum funcționează?

→ Avem un arbore (initial doar nodul de start)

→ Se alege un vîrf cu costul minim de a îl lega de arbore (poate fi legat cu mai multe muchii, dar min-heap o să ne-o dea pe cea mai bună, cost mai mic)

→ Adăugăm nodul cu muchia minimă în graf; îl marcam ca vizitat.

PSEUDOCOD: (Min Heap)

Prim(G, w, s)

```
pentru fiecare  $u \in V$  executa  
     $d[u] = \infty$ ;  $tata[u] = 0$   
     $d[s] = 0$   
initializează  $Q$  cu  $V$   
cat timp  $Q \neq \emptyset$  executa  
     $u = \text{extrage vîrf cu eticheta } d \text{ minimă din } Q$   
    pentru fiecare  $v$  adiacent cu  $u$  executa  
        dacă  $v \in Q$  și  $w(u, v) < d[v]$  atunci  
             $d[v] = w(u, v)$   
             $tata[v] = u$   
            //actualizeaza  $Q$  - pentru  $Q$  heap  
    scrie  $(u, tata[u])$ , pentru  $u \neq s$ 
```

COMPLEXITATE:
 $O(m \log n)$

Clustering

- Inițial fiecare obiect (cuvânt) formează o clasă
- La un pas determinăm **cele mai asemănătoare (apropiate) două obiecte** aflate în clase diferite (cu distanța cea mai mică între ele) și unim clasele lor
- Repetăm până obținem k clase $\Rightarrow n - k$ pași

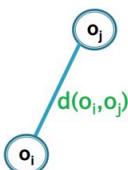
PSEUDOCOD:

Pseudocod:

- Inițial fiecare obiect (cuvânt) formează o clasă
- pentru $i = 1, n-k$
 - alege două obiecte o_r, o_t din clase diferite cu $d(o_r, o_t)$ minimă
 - reunește (clasa lui o_r , clasa lui o_t)
- afișează cele k clase obținute



Modelare cu graf ponderat (**complet**)
 $\Rightarrow n - k$ pași din algoritmul lui Kruskal



Pseudocod – modelare cu graf complet G:

$$V = \{o_1, \dots, o_n\}, \quad w(o_i, o_j) = d(o_i, o_j)$$

Inițial fiecare vîrf formează o componentă conexă (clasă): $T' = (V, \emptyset)$

pentru $i = 1, n-k$

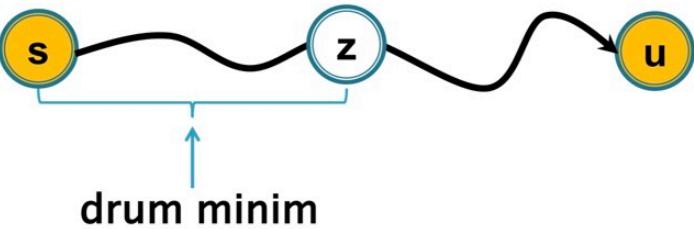
- alege o muchie $e_i = uv$ de **cost minim** din G astfel încât u și v sunt în componente conexe diferențiale ale lui T'
- reunește componenta lui u și componenta lui v : $E(T') = E(T') \cup \{uv\}$

returnează cele k mulțimi formate cu vîrfurile celor k componente conexe ale lui T'

→ Basically Kruskal doar că adaug $n-k$ muchii în TPCM nu $n-1$.

Drumuri Minime în grafuri ponderate

- **Observația 2.** Dacă P este un drum minim de la s la u și z este un vârf al lui P , atunci subdrumul lui P de la s la z este drum minim de la s la z .



Drumuri minime de la un nod la toate celelalte

Problema drumurilor minime de sursă unică (de la s la celelalte vârfuri)

Se dau:

- un graf **orientat** ponderat $G = (V, E, w)$, cu
 $w : E \rightarrow \mathbb{R}$
- un vârf de start s

Să se determine distanța de la s la fiecare vârf x al lui G / la un vârf dat t (și un arbore al distanțelor față de s / un drum minim elementar de la s la t)

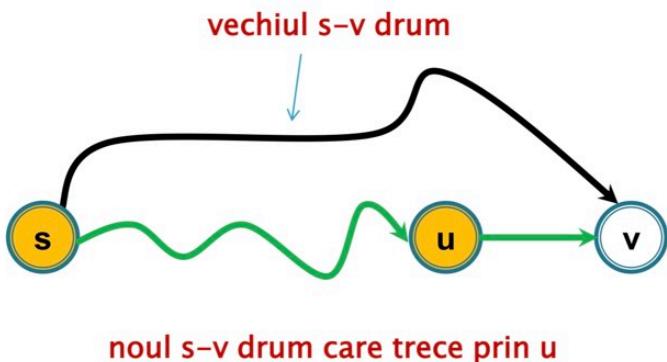
Definiție: Pentru un vârf dat s un **arbore al distanțelor față de s** = un subgraf T al lui G care conservă distanțele de la s la celelalte vârfuri accesibile din s

$$\delta_T(s, v) = \delta_G(s, v), \quad \forall v \in V \text{ accesibil din } s,$$

graful neorientat asociat lui T fiind arbore cu rădăcina în s (cu arcele corespunzătoare orientate de la s la frunze)

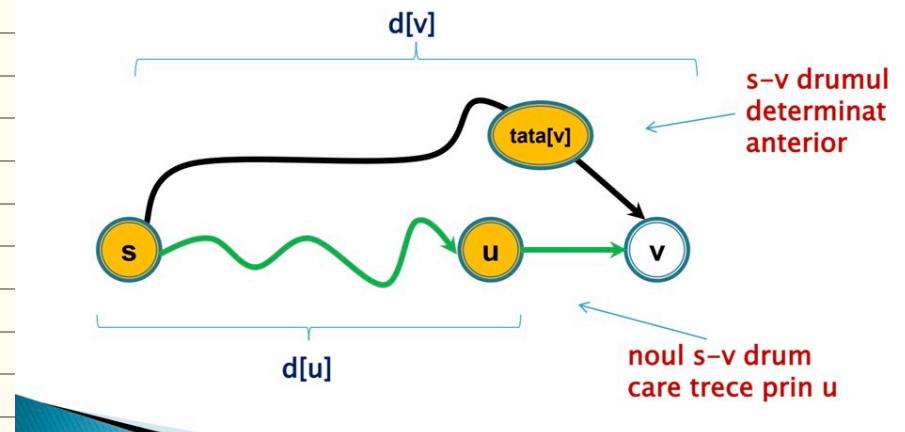
- Algoritmi pentru grafuri orientate cu circuite, dar cu ponderi pozitive – Dijkstra
- Algoritmi pentru grafuri orientate fără circuite (cu ponderi reale) DAGs = Directed Acyclic Graphs
- Algoritmi pentru grafuri orientate cu circuite și ponderi reale, care detectează existența de circuite negative – Bellman–Ford

- Relaxarea unui arc (u, v) = a verifica dacă $d[v]$ poate fi îmbunătățit extinzând drumul minim deja găsit de la s la u cu arcul uv



- Relaxarea unui arc (u, v) :

dacă $d[u] + w(u,v) < d[v]$ atunci
 $d[v] = d[u] + w(u,v);$
 $tata[v] = u$



Drumuri minime de sursă unică în grafuluri aciclice DAG

► Ipoteze:

- Graful nu conține circuite
- Arcele pot avea și cost negativ

► Amintim:

Când considerăm un vârf v , pentru a calcula $d(s,v)$ ar fi util să știm deja $\delta(s,u)$ pentru orice u cu $uv \in E \Rightarrow$

- Ar fi utilă o ordonare a vârfurilor astfel încât dacă $uv \in E$, atunci u se află înaintea lui v



O astfel de ordonare există dacă graful nu conține circuite = sortarea topologică

PSEUDOCOD:

```
s - vârful de start

//initializam distante - ca la Dijkstra
pentru fiecare  $u \in V$  execută
     $d[u] = \infty$ ;  $tata[u] = 0$ 
 $d[s] = 0$ 

//determinăm o sortare topologică a vârfurilor
SortTop = sortare_topologica(G)

pentru fiecare  $u \in \text{SortTop}$ 
    pentru fiecare  $uv \in E$  execută
        daca  $d[u] + w(u,v) < d[v]$  atunci //relaxăm uv
             $d[v] = d[u] + w(u,v)$ 
            tata[v] = u

scrie d, tata
```

COMPLEXITATE: initializare : $O(n)$

sort topologică : $O(m+n)$

$m *$ relaxare uv : $O(m)$

$O(m+n)$ +

Dijkstra: Drumuri minime de la un nod la celealte, ponderat, ciclic

► Ipoteză:

Presupunem că arcele au cost pozitiv

(graful poate conține circuite)

- generalizare a ideii de parcurgere BF
- dacă toate arcele au cost egal Dijkstra = BF

PSEUDOCOD:

Dijkstra(G, w, s)

```
initializează multimea vârfurilor neselectate Q cu V
pentru fiecare u∈V execută
    d[u] = ∞; tata[u]=0
d[s] = 0
cat timp Q ≠ ∅ execută
    u = extrage vârf cu eticheta d minimă din Q
    pentru fiecare uv∈E execută
        daca d[u]+w(u,v)< d[v] atunci
            d[v] = d[u]+w(u,v)
            tata[v] = u
    scrie d, tata
//scrie drum minim de la s la t un varf t dat folosind tata
```

⇒ $O(n^2)$

Dijkstra(G, w, s) - Q min-heap in raport cu d

```
pentru fiecare u∈V execută
    d[u] = ∞; tata[u]=0
d[s] = 0
Q = V //creare heap cu cheile din d
cat timp Q ≠ ∅ execută
    u = extrage_min(Q)
    pentru fiecare uv∈E execută
        daca d[u]+w(u,v)< d[v] atunci
            d[v] = d[u]+w(u,v)
            repara(Q,v)
            tata[v] = u
    scrie d, tata
//scrie drum minim de la s la t un varf t dat folosind tata
```

⇒ $O(m \log n)$

► Observație. Pentru a determina drumul minim între două vârfuri s și t date putem folosi algoritmul lui Dijkstra cu următoarea modificare:

- dacă vârful u ales este chiar t , **algoritmul se oprește**;
- drumul de la s la t se afișează folosind vectorul *tata* (vezi BF)

Putem modifica algoritmul lui Dijkstra de determinare de drumuri minime în grafuri (nu neapărat aciclice) a.î. să determine drumuri maxime (elementare) de la S la celelalte vârfuri

- Modificăm astfel doar inițializarea distanțelor (cu $-\infty$ în loc de $+\infty$) și inversam condiția de la relaxarea arcelor pentru a calcula maxim în loc de minim



Corectitudine - probabil similar cu Dijkstra?!!

Bellman-Ford: Distanțe minime de la un nod la celelalte, graf cu ponderi negative, ciclic

► Ipoteză:

Arcele pot avea și cost negativ

Graful nu conține circuite de cost negativ

(dacă există – algoritmul le va detecta => nu soluție)

Toate vârfurile sunt accesibile din s

► Idee: La un pas relaxăm toate arcele

(nu relaxăm arcele dintr-un vârf selectat u, ci din toate vârfurile)

► **n-1 etape** – după k etape $d[u] \leq$ costul minim al unui drum de la s la u cu cel mult k arce (programare dinamică)

=> după k etape sunt corect calculate etichetele $d[u]$ pentru acele vârfuri u pentru care există un s-u drum minim cu cel mult k arce

PSEUDOCOD:

```

pentru fiecare  $u \in V$  execută
     $d[u] = \infty$ ;  $tata[u] = 0$ 
 $d[s] = 0$ 

pentru  $k = 1, n-1$  execută
    pentru fiecare  $u \in E$  execută
        dacă  $d[u] + w(u, v) < d[v]$  atunci
             $d[v] = d[u] + w(u, v)$ 
             $tata[v] = u$ 

```

Complexitate: $O(nm)$

Considerând că graful este corect definit (fără cicluri negative)

Dacă primim un graf corect, atunci trebuie să mai facem verificarea

```

pentru fiecare  $u \in E$  execută
    dacă  $d[u] + w(u, v) < d[v]$  atunci

```

return 0;

STOP: ciclu negativ

PSEUDOCOD OPTIMIZARI:

```

queue<int> q; //coada cu vf a căror eticheta s-a actualizat
d[s] = tata[s] = 0;
q.push(s);
in_q[s] = 1; //dacă un varf este deja în coadă - il marchez
while (!q.empty()) {
    u = q.front();
    q.pop();
    in_q[u] = 0;
    for (j=0; j<la[u].size(); j++) {
        v = la[u][j].first;
        w_uv = la[u][j].second;
        if (d[u]<infinit && d[u] + w_uv < d[v]) {
            d[v] = d[u] + w_uv;
            tata[v] = u;
            if (in_q[v]==0) {
                q.push(v);
                in_q[v] = 1;
            }
        }
    }
}

```

- La un pas este suficient să relaxăm arcele din vîrfuri ale căror etichetă s-a modificat anterior.

- Putem să menținem evidența acestora astfel:

- folosind un vector de vizitat în care marcam vîrfurile a căror etichetă s-a actualizat la etapa curentă
- SAU - o coadă cu vîrfurile a căror etichetă s-a modificat până la etapa curentă => implementare diferită

Floyd-Warshall: Drumuri minime între toate perechile de vârfuri

Problema drumurilor minime între toate perechile de vârfuri

Se dă:

- un graf **orientat** ponderat $G = (V, E, w)$

Pentru oricare două vârfuri x și y al lui G să se determine distanța de la x la y și un drum minim de la x la y

Ponderile pot fi și negative dar NU există circuite cu cost negativ în G

- ▶ Fie $W = (w_{ij})_{i,j=1..n}$ **matricea costurilor** grafului G :

$$w_{ij} = \begin{cases} 0, & \text{daca } i = j \\ w(i,j), & \text{daca } ij \in E \\ \infty, & \text{daca } ij \notin E \end{cases}$$

- ▶ Vrem să calculăm matricea distanțelor $D = (d_{ij})_{i,j=1..n}$:

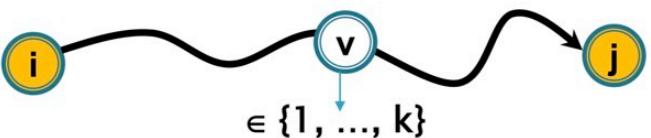
$$d_{ij} = \delta(i, j)$$

► Ideea algoritmului Floyd–Warshall:

Programare Dinamică – Subprobleme:

Pentru $k = 1, 2, \dots, n$ calculăm pentru oricare două vârfuri i, j :

costul minim al unui drum (elementar) de la i la j care are ca vârfuri intermediare doar vârfuri din mulțimea $\{1, 2, \dots, k\}$



► Ideea algoritmului Floyd–Warshall:

Astfel, pentru $k = 1, 2, \dots, n$ calculăm matricea

$$D^{(k)} = (d_{ij}^k)_{i,j=1..n} :$$

d_{ij}^k = costul minim al unui drum de la i la j care are vârfurile intermediare în $\{1, 2, \dots, k\}$

► Se obține astfel relația

$$d_{ij}^k = \min \{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\}$$

► Observații

- Avem

$$d_{ik}^k = d_{ik}^{k-1}$$

$$d_{kj}^k = d_{kj}^{k-1}$$

de aceea în implementarea algoritmului putem folosi o singură matrice

PSEUDOCOD:

COMPLEXITATE: $O(n^3)$

Floyd(G, w)

```
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++) {
        d[i][j]=w[i][j];
        if(w[i][j]== ∞)
            p[i][j]=0;
        else
            p[i][j]=i;
    }
    for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(d[i][j]>d[i][k]+d[k][j]){
                    d[i][j]=d[i][k]+d[k][j];
                    p[i][j]=p[k][j];
                }
}
```

- ▶ Ieșire: matricea d = matricea distanțelor
- ▶ Afisarea unui drum de la i la j , **daca $d[i][j] < \infty$** , se face folosind matricea p

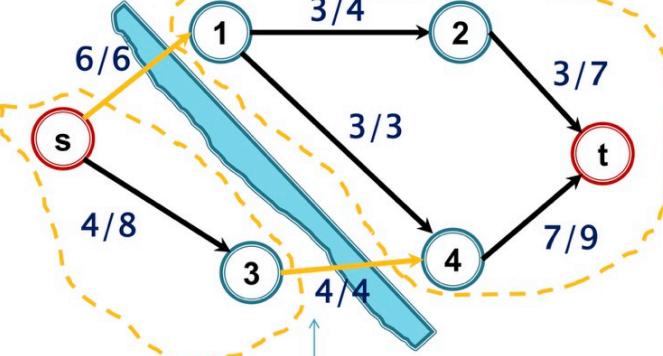
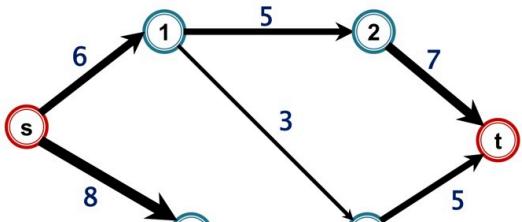
```
void drum(int i,int j){
    if(i!=j)
        drum(i,p[i][j]);
    cout<<j<<" ";
}
```

Fluxuri Maxime în Rețele de transport

Fluxuri



- Avem o rețea în care
 - arcele au limitări de capacitate
 - nodurile = joncțiune
- Care este cantitatea maximă care poate fi transmisă prin rețea de la surse la destinații? (în unitatea de timp)



- singurele arce ("poduri") care trec din regiunea lui s în cea a lui t nu mai pot fi folosite pentru a trimite flux (au fluxul = capacitatea) \Rightarrow fluxul este maxim
- s-t călătoră

Fluxuri în rețele de transport

- Rețea de transport** $N = (G, S, T, I, c)$ unde
 - $G = (V, E)$ - graf orientat cu
 - $V = S \cup I \cup T$
 - S, I, T disjuncte, nevide
 - S – mulțimea **surselor** (intrărilor)
 - T – mulțimea **destinațiilor** (ieșiri)
 - I – mulțimea vârfurilor **intermediare**
 - $c : E \rightarrow \mathbb{N}$ funcția **capacitate** (cantitatea maximă care poate fi transportată prin fiecare arc)

PROPRIETĂȚI:

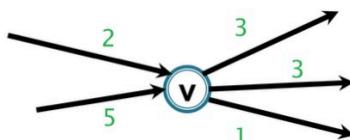
- Un **flux** într-o rețea de transport $N = (G, S, T, I, c)$ este o funcție $f : E \rightarrow \mathbb{N}$ cu proprietățile

1) $0 \leq f(e) \leq c(e), \forall e \in E(G)$ *condiția de mărginire*

2) Pentru orice vârf **intermediar** $v \in I$

$$\sum_{uv \in E} f(uv) = \sum_{vu \in E} f(vu) \quad \text{condiția de conservare a fluxului}$$

(fluxul total care intră în v = fluxul total care ieșe din v)



Determinarea unui flux maxim

Fie $N = (G, \{s\}, \{t\}, I, c)$ o rețea

- Un **lanț** este o succesiune de vârfuri distincte și **arce** din G

$$P = [v_0, e_1, v_1, \dots, v_{k-1}, e_k, v_k]$$

unde arcul e_i este fie $v_{i-1}v_i$, fie v_iv_{i-1}

(P este lanț elementar în graful neorientat asociat lui G)

Dacă

- $e_i = v_{i-1}v_i \in E(G)$, e_i s.n **arc direct (înainte)** în P
- $e_i = v_iv_{i-1} \in E(G)$, e_i s.n **arc invers (înapoi)** în P



NOTAȚII:

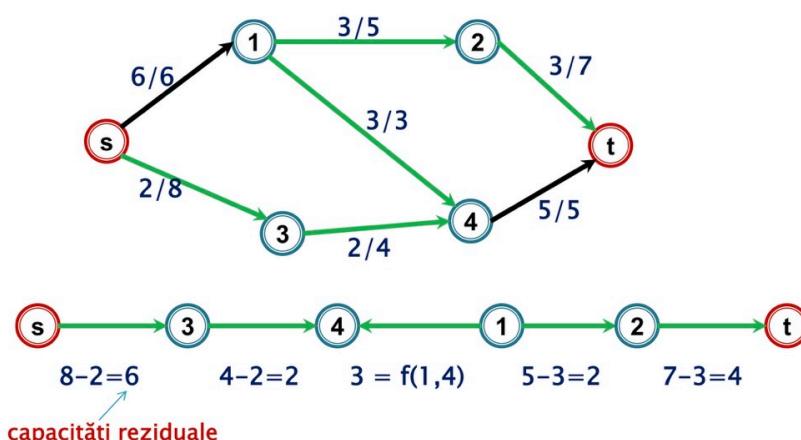
$$f^+(v) = \sum_{vu \in E} f(vu) \quad \text{= fluxul care ieșe din } v$$

$$f^-(v) = \sum_{uv \in E} f(uv) \quad \text{= fluxul care intră în } v$$

- Condiția de conservare a fluxului devine:

$$f^-(v) = f^+(v), \forall v \in I$$

- Fie N rețea, f flux în N , P un lanț
- Asociem fiecărui arc e din P o pondere, numită **capacitate reziduală** în P



- ▶ Fie N rețea, f flux în N , P un lanț
- ▶ Asociem fiecărui arc e din P o pondere, numită **capacitate reziduală** în P :

$$i_p(e) = \begin{cases} c(e) - f(e), & \text{dacă } e \text{ este arc direct în } P \\ f(e), & \text{dacă } e \text{ este arc invers în } P \end{cases}$$

= cu cât mai poate fi modificat fluxul pe arcul e , de-a lungul lanțului P

- ▶ Un s-t lanț P se numește
 - **f-nesaturat (f-drum de creștere)** *augmenting path* dacă $i(P) \neq 0$
 - **f-saturat** dacă $i(P) = 0$

- ▶ Capacitatea reziduală a unui lanț P este

$$i(P) = \min\{i_p(e) \mid e \in E(P)\}$$
 = cu cât mai poate fi modificat fluxul de-a lungul lanțului P
- ▶ Convenție: Dacă $E(P) = \emptyset$, $i(P) = \infty$

- ▶ Fie N - rețea, f flux în N , P un **s-t lanț f-nesaturat**.
- ▶ Fluxul revizuit de-a lungul lanțului P se definește ca fiind $\mathbf{f}_P : E \rightarrow \mathbb{N}$,

$$f_P(e) = \begin{cases} f(e) + i(P), & \text{dacă } e \text{ este arc direct în } P \\ f(e) - i(P), & \text{dacă } e \text{ este arc invers în } P \\ f(e), & \text{altfel} \end{cases}$$

PSEUDOCOD:

- Fie f un flux în N (de exemplu $f \equiv 0$ fluxul vid:
 $f(e) = 0, \forall e \in E$)
- Cât timp există un s-t lanț f-nesaturat P în G
 - determină un astfel de lanț P
 - revizuește fluxul f de-a lungul lanțului P
- returnează f

COMPLEXITATE:

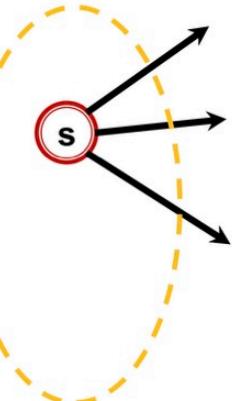
Complexitate

- $O(mL)$, unde L este capacitatea minimă a unei tăieturi

Avem $L \leq c^+(s) \leq n \cdot C$, unde $C = \max\{c(e) | e \in E(G)\} \Rightarrow$

- $O(nmC)$ unde

$$C = \max\{c(e) | e \in E(G)\}$$



Edmonds-Karp

Schema:

```
initializeaza_flux_nul()
cat timp (construieste_s-t_lant_nesat_BF())=true) executa
    revizuieste_flux_lant()
afiseaza_flux()
```

PSEUDOCOD - Revizuire lanț s-t nesaturat

```
construieste_s-t_lant_nesat_BF()
    pentru (v∈V) executa tata[v] ← 0; viz[v] ← 0
    coada C ← ∅
    adauga(s, C)
    viz[s] ← 1
    cat timp C ≠ ∅ executa
        i ← extrage(C)
        pentru (ij ∈ E) executa arc direct
            dacă (viz[j]=0 și c(ij)-f(ij)>0) atunci
                adauga(j, C)
                viz[j] ← 1; tata[j] ← i
                daca (j=t) atunci STOP și returnează true(1)
            pentru (ji ∈ E) executa arc invers
                daca (viz[j]=0 și f(ji)>0) atunci
                    adauga(j, C)
                    viz[j] ← 1; tata[j] ← -i
                    daca (j=t) atunci STOP și returnează true(1)
    returnează false(0)
```

Algoritmul Edmonds-Karp

Complexitate

- Algoritm generic Ford Fulkerson O(mL)/O(nmC)
- Implementare Edmonds Karp O(nm²)

Amintim:

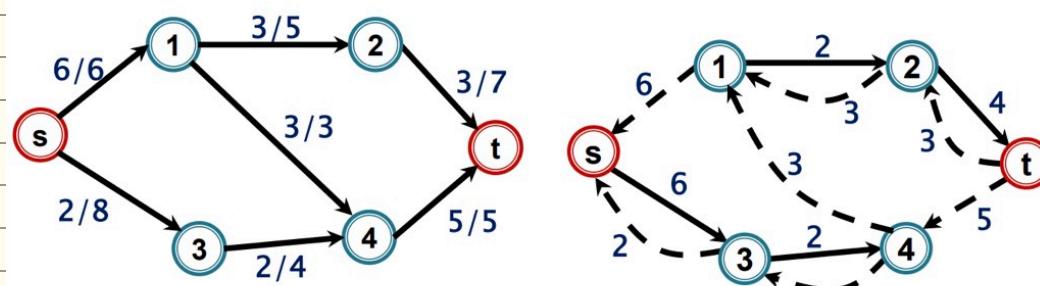
a determina un s-t lanț nesaturat folosind BF în G

↔

a determina un s-t drum folosind BF în graful rezidual G_f

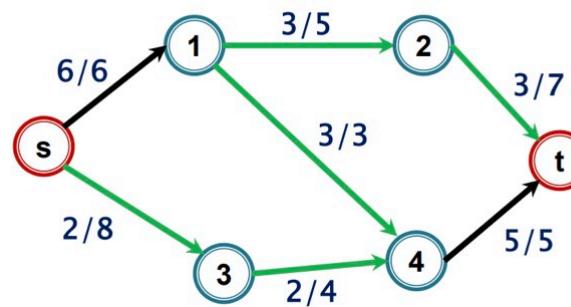
IMPLEMENTARE GRAF REZIDUAL:

Rețeaua de transport



Graful rezidual

Graf rezidual



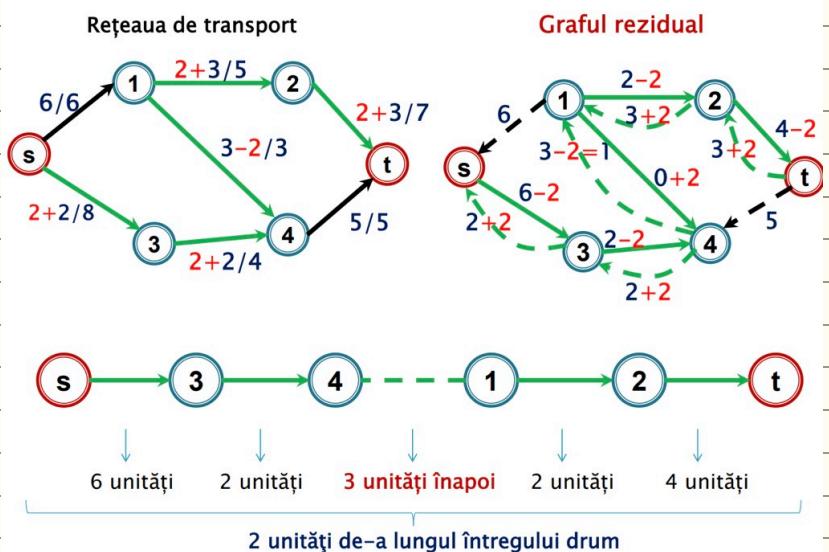
s-t lanț f-nesaturat



(Multi)graful rezidual G_f asociat fluxului $f =$ graf orientat ponderat, cu funcția de pondere (de capacitate) c_f construit astfel:

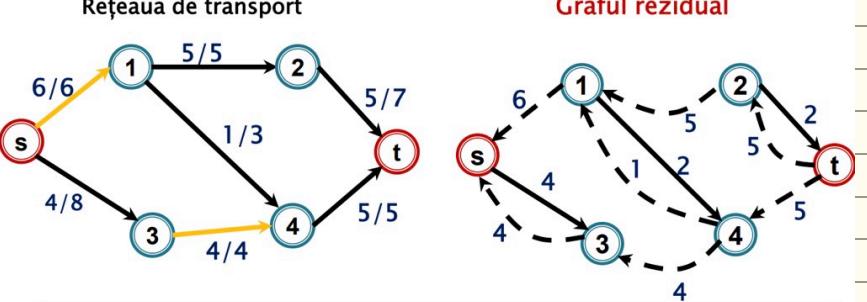
- $V(G_f) = V(G)$
- Pentru fiecare arc $e = uv$ cu $c(e) - f(e) > 0$, adăugăm arcul e la G_f cu ponderea $c_f(e) = c(e) - f(e)$
- Pentru fiecare arc $e = uv$ cu $f(e) > 0$, adăugăm la G_f arcul $e^{-1} = vu$ (inversul arcului e) cu ponderea asociată $c_f(e^{-1}) = f(e)$.

Rețeaua de transport



Graful rezidual

Rețeaua de transport



Graful rezidual

tăietură saturată \Leftrightarrow nu mai există s-t drum în graful rezidual

\Leftrightarrow s-t flux maxim

este tăietură minimă

PSEUDOCOD:

Algoritmul Edmonds-Karp – implementare folosind graf rezidual

Detalii (implementare – laborator)

1. Setăm $f := 0$ fluxul vid ($f(e) = 0, \forall e \in E$)

2. Construim $G_f :=$ graful rezidual pentru f

3. Cât timp există un s-t drum în G_f

- determină P un s-t drum minim în G_f folosind BF (pentru

arcele cu $c_f(e) > 0$

- actualizează G_f

pentru $e \in E(P) \subseteq E(G_f)$

$$c_f(e) \leftarrow c_f(e) - cf_P;$$

$$c_f(e^{-1}) \leftarrow c_f(e^{-1}) + cf_P$$

4. returnează f

```
#include<fstream>
#include<iostream>
#include <queue>
#include <cstring>
#define MAXN 10001
using namespace std;

int n, m;
int rezid[MAXN][MAXN];
vector<int> lrezid[MAXN];
int viz[MAXN], tata[MAXN];

int bfs() {
    int s = 0;
    int t = n - 1;

    for (int i = 0; i < n; i++) {
        viz[i] = 0;
        tata[i] = 0;
    }

    queue<int> q;
    q.push(s);
    viz[s] = 1;

    while (!q.empty()) {
        int x = q.front();
        q.pop();

        for (int y : lrezid[x]) {
            if (!viz[y] && rezid[x][y] > 0) {
                viz[y] = 1;
                tata[y] = x;

                if (y == t)
                    return 1;
                q.push(y);
            }
        }
    }
    return 0;
}
```

```
int main() {
    ifstream fs("maxflow.in");
    ofstream g("maxflow.out");

    fs >> n >> m;

    for (int i = 0; i < m; i++) {
        int x, y, c;
        fs >> x >> y >> c;
        x--; y--;
        lrezid[x].push_back(y);
        lrezid[y].push_back(x);
        rezid[x][y] += c;
    }

    fs.close();

    int fmax = 0;
    int s = 0;
    int t = n - 1;

    while (bfs()) {
        int iP = 110001;

        t=n-1;
        while(t!=0) {
            if(iP>rezid[tata[t]][t])
                iP= rezid[tata[t]][t];
            t=tata[t];
        }

        t=n-1;
        while(t!=0) {
            rezid[tata[t]][t]-=iP;
            rezid[t][tata[t]]+=iP;
            t=tata[t];
        }

        fmax += iP;
    }

    g << fmax;
    g.close();

    return 0;
}
```

1/2

2/2

Colorări în grafuri.

APLICAȚII: → graf de conflicte

→ de câte săli este nevoie min pt un eveniment

→ colorarea unei hârti

→ graf bipartit

Graf Bipartit

Observații

► $G = (V, E)$ bipartit \Leftrightarrow

există o 2-colorare proprie a vârfurilor (bicolorare):

$$c : V \rightarrow \{1, 2\}$$

(i.e. astfel încât pentru orice muchie $e=xy \in E$ avem $c(x) \neq c(y)$)

► $G = (V, E)$ bipartit $\Rightarrow \chi(G) \leq 2$

Propoziție

Un arbore este graf bipartit

► Teorema König - Caracterizarea grafurilor bipartite

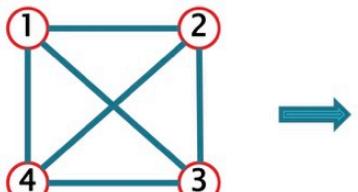
Fie $G = (V, E)$ un graf cu $n \geq 2$ vârfuri.

Avem

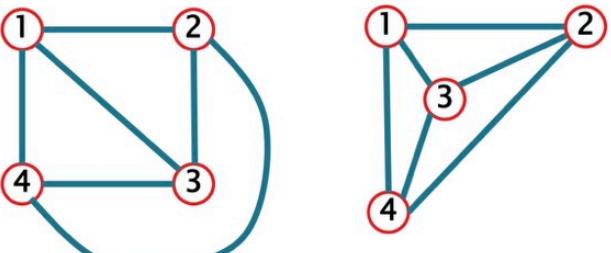
G este bipartit \Leftrightarrow toate ciclurile elementare
din G sunt pare

Grafuri Planare

- $G = (V, E)$ graf neorientat s.n. planar \Leftrightarrow admite o reprezentare în plan a.î. muchiilor le corespund segmente de curbe continue care **nu se intersectează** în interior unele pe altele
- O astfel de reprezentare s.n. hartă a lui G

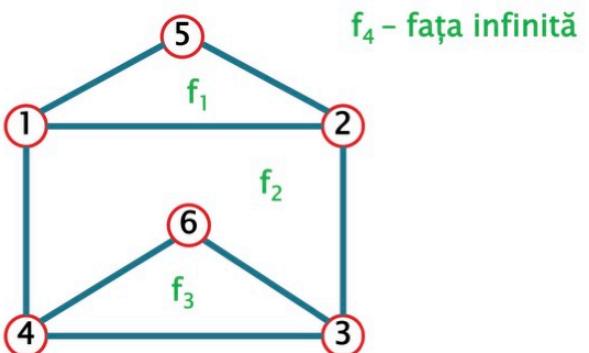


$G \sim K_4$



hărți

- ▶ Fie $G = (V, E)$ graf planar, M o hartă a sa
- ▶ M induce o împărțire a planului într-o mulțime F de părți convexe numite **fețe**
- ▶ Una dintre acestea este **fața infinită (exterioară)**



- ▶ Pentru o față $f \in F$ definim
 - $d_M(f) = \text{gradul feței } f = \text{numărul muchiilor lanțului închis (frontierei) care delimitizează } f$ (*câte muchii sunt parcuse atunci când traversăm frontiera*)

Observație: Hărți diferite ale aceluiași graf pot avea secvența gradelor fețelor **diferită**

PROPRIETĂȚI:

- ▶ $M = (V, E, F)$ hartă

- Avem

$$\sum_{f \in F} d_M(f) = 2|E|$$

(deoarece o muchie este incidentă cu două fețe)

▶ Teorema poliedrală a lui EULER

Fie $G = (V, E)$ un graf planar **conex** și $M = (V, E, F)$ o hartă a lui. Are loc relația

$$|V| - |E| + |F| = 2$$

▶ Consecință

Orice hartă M a lui G are $2 - |V| + |E|$ fețe

▶ Proprietăți

Fie $G=(V, E)$ un graf planar conex cu $n=|V|>2$ și $m=|E|$.

Atunci:

- a) $m \leq 3n - 6$
- b) $\exists x \in V$ cu $d(x) \leq 5$.

▶ Consecință

K_5 nu este graf planar

▶ Proprietăți (temă)

Fie $G=(V, E)$ un graf planar conex bipartit cu $n=|V|>2$ și $m=|E|$. Atunci:

- a) $m \leq 2n - 4$
- b) $\exists x \in V$ cu $d(x) \leq 3$.

▶ Consecință

$K_{3,3}$ nu este graf planar

! K_n : graf complet cu n noduri.

$K_{m,n}$: graf complet bipartit. $m = \text{nr de noduri din prima partitură}$
 $n = \text{nr de noduri din a doua partitură}$

► Teorema lui Kuratowski

G este graf planar \Leftrightarrow nu conține subdiviziuni ale lui $K_{3,3}$
și ale lui K_5 .

PSEUDOCOD:

Graf planar

► Teorema celor 6 culori

Orice graf planar conex este 6 -colorabil.

► Algoritm de colorare a unui graf planar cu 6 culori

colorare(G)

daca $|V(G)| \leq 6$ atunci coloreaza varfurile cu
culori distincte din $\{1, \dots, 6\}$

altfel

alege x cu $d(x) \leq 5$

colorare($G - x$)

colorează x cu o culoare din $\{1, \dots, 6\}$

diferită de culorile vecinilor deja

colorați (!se poate, x are cel mult 5

vecini din $G - x$)

Determinarea iterativă a ordinii în care sunt colorate vârfurile
la pasul 1 se poate face similar cu determinarea unei ordonări
topologice:

```
coada C = Ø;  
adăuga în C toate vârfurile v cu d[v] ≤ 5 și  
marchează-le ca vizitate  
stiva S = Ø  
cat timp C ≠ Ø execută  
    i ← extrage(C);  
    adăuga i în S  
    pentru ij ∈ E execută  
        d[j] = d[j] - 1  
        dacă d[j] ≤ 5 și j este nevizitat atunci  
            adăuga(j, C)  
cat timp S este nevidă execută  
    u = pop(S)  
    adăuga u în sortare
```

Problema: Este un graf P-colorabil?

Grafuri Euleriene

Fie G graf neorientat

- ▶ **Ciclu eulerian** al lui G = ciclu C în G cu

$$E(C) = E(G)$$

- ▶ **G eulerian** = conține un **ciclu** eulerian

- ▶ **Lanț eulerian** al lui G = lanț simplu P în G cu

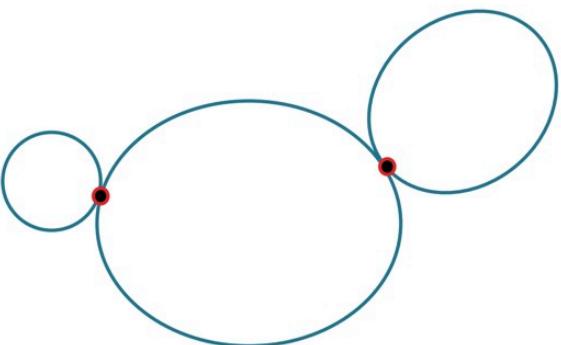
$$E(P) = E(G)$$

→ Ciclu Eulerian: Toate $d(v) \geq G$ par e

Lanț Eulerian: are fie 0 sau 2 noduri de grad impar

Determinarea unui ciclu eulerian într-un graf conex (sau un graf conex+ vîrfuri izolate) cu toate vîrfurile de grad par

- bazat pe ideea demonstrației Teoremei lui Euler –
fuziune de cicluri (succesiv)



Algoritmul lui Hierholzer

- ▶ **Pasul 0** – verificare condiții (conex+vf. izolate, grade pare)
- ▶ **Pasul 1:**
 - alege $v \in V$ arbitrar
 - construiește C un ciclu în G care începe cu v (cu algoritmul din Lema)
- ▶ **cât timp $|E(C)| < |E(G)|$ execută**
 - selectează $v \in V(C)$ cu $d_{G-E(C)}(v) > 0$ (în care sunt incidente muchii care nu aparțin lui C)
 - construiește C' un ciclu în $G - E(C)$ care începe cu v
 - $C =$ ciclul obținut prin fuziunea ciclurilor C și C' în v
- ▶ **scrie C**

► Posibile implementări

- Varianta – Nerecursiv – Liste dublu înlăntuite/stive
- Muchiile folosite – marcate (nu neapărat șterse)

```
stiva St
push(1,St)
cat timp St este nevida execută
    v = top(St)
    daca grad(v) != 0 atunci
        alege vw o muchie incidentă în v
        sterge muchia vw din G
        push(w,St)
    altfel
        C = C + v
        pop(St)
```

PROPRIETĂȚI

Teorema lui Euler

Fie $G=(V, E)$ un graf neorientat, conex, cu $E \neq \emptyset$.

Atunci

G are un lanț eulerian $\Leftrightarrow G$ are cel mult două vârfuri de grad impar

Observație

– Fie $P = [v_1, \dots, v_k]$ dum

- Dacă $v_1 \neq v_k$, atunci vârfurile interne v din P au

$$d_P^-(v) = d_P^+(v), \text{ iar pentru extremități:}$$

$$d_P^-(v_1) = d_P^+(v_1) - 1, d_P^-(v_k) = d_P^+(v_k) + 1$$

- Dacă $v_1 = v_k$, atunci toate vârfurile v din P au gradul intern în P egal cu cel extern:

$$d_P^-(v) = d_P^+(v)$$

Teorema lui Euler

Fie $G = (V, E)$ un graf orientat, conex (= graful neorientat asociat este conex), cu $E \neq \emptyset$.

Atunci

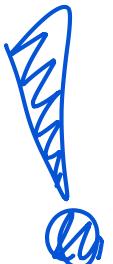
$$G \text{ este eulerian} \Leftrightarrow \forall v \in V \quad d_G^-(v) = d_G^+(v)$$

Fie $G=(V, E)$ un (multi)graf orientat, conex, cu $E \neq \emptyset$.

Atunci

G are un drum eulerian \Leftrightarrow

$$\begin{aligned} & (\forall v \in V \quad d_G^-(v) = d_G^+(v)) \text{ sau} \\ & (\exists x \in V \text{ cu } d_G^-(x) = d_G^+(x) - 1, \\ & \quad \exists y \in V, y \neq x \text{ cu } d_G^-(y) = d_G^+(y) + 1, \\ & \quad \forall v \in V - \{x, y\} \quad d_G^-(v) = d_G^+(v)) \end{aligned}$$



► **k-descompunere euleriană în lanțuri** a unui graf $G =$

o mulțime de k lanțuri simple, muchie-disjuncte

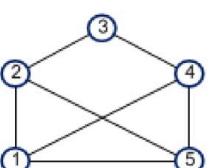
$$\Delta = \{P_1, P_2, \dots, P_k\}$$

ale căror muchii induc o k-partiție a lui $E(G)$:

$$E(G) = E(P_1) \cup E(P_2) \cup \dots \cup E(P_k)$$

► **Interpretare**

De câte ori (minim) trebuie să ridicăm creionul de pe hârtie pentru a desena diagrama?



Descompuneri euleriene în lanțuri

Teoremă – Descompunere euleriană

Fie $G=(V, E)$ un graf orientat, conex (= graful neorientat asociat este conex), cu **exact $2k$ vârfuri de grad impar** ($k>0$). Atunci există o k -descompunere euleriană a lui G și k este cel mai mic cu această proprietate.

Distanță de editare

Operații: **INSERARE**: adaugă un caracter
cost 1

STERGERE: elimină un caracter

SUBSTITUȚIE: înlocuiește un caracter (cost 0 dacă sunt identice)

Caz de bază: $dp[0][0] = 0$

$$dp[i][0] = i.$$

$$dp[0][j] = j$$

Formula de recurență: $dp[i][j] = \begin{cases} dp[i-1][j-1]; dc \neq B[j] \\ 1 + \min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1]); dc \neq B[j] \end{cases}$

stergere

inserare

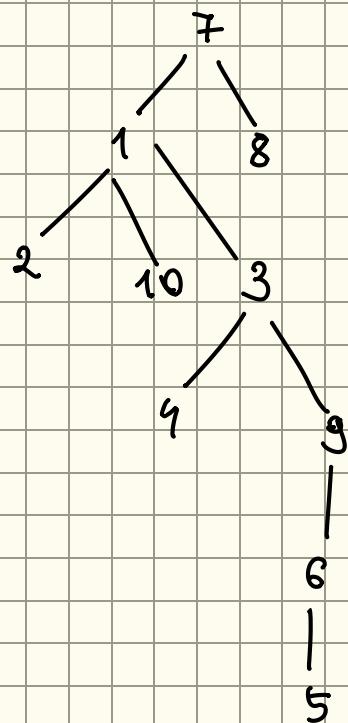
substituție

Model Examen 2023

1. Graful are ciclu $4 \rightarrow 7 \rightarrow 8 \rightarrow 3 \rightarrow 4 \Rightarrow$ nu există o sortare topologică.

2. Parcungetea în lățime parcurge vecinii unui nod

$\text{Q: } 7 \times 8 \times 10 \times 4 \neq 5$



$\text{dist}[\text{start}] = 0$

(u vecin al lui v)

dacă $\text{dist}[u] = \text{dist}[v] + 1$.

return $\text{dist}[g]$.

3. $(g, 6, 5)$

(11)

$(1, 3, 4, 7, 8)$

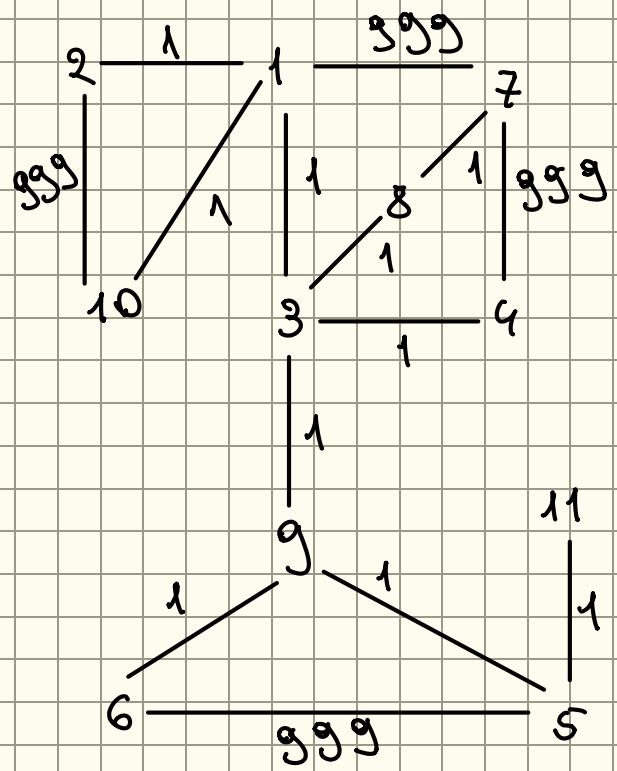
$(2, 10)$

adaug $(5, 4)$ muchie

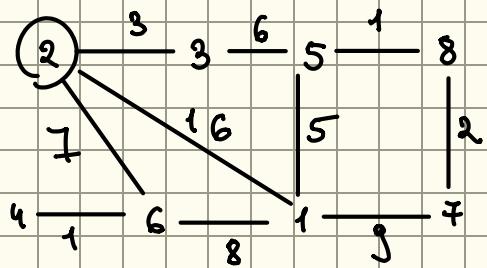
$\Rightarrow (1, 3, 4, 7, 8, 9, 6, 5)$

componente conexe

4)



5.



Dijkstra: Distanța de la 2 la toate nodurile

	1	2	3	4	5	6	7	8
d/tata	$\infty/0$							
	-	-	-	-	-	-	-	-
t6/2	-	$0/0$	$3/2$	-	-	$7/2$	-	-
	-	-	$3/2$	-	$9/3$	-	-	-

15/6	-	-	8/6	-	7/2	-	-
-	-	-	8/6	-	-	-	-
14/5	-	-	-	9/3	-	-	10/5

Q: 1 2 3 4 5 6 7 8

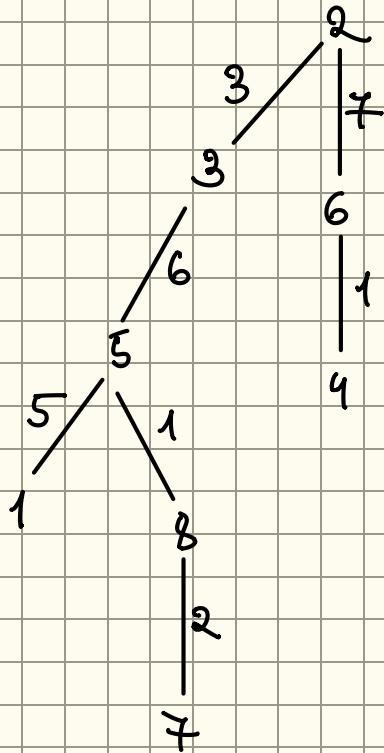
1 2 3 4 5 6 7 8
d: 14 0 3 8 9 7 12 10
tata: 5 0 2 6 3 2 8 5

6) Prim

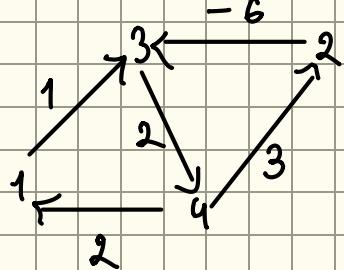
1	2	3	4	5	6	7	8
∞/0	0/0	∞/0	∞/0	0/0	0/0	0/0	0/0
16/2	-	3/2	-	-	7/2	-	-
-	-	3/2	-	6/3	-	-	-
5/5	-	-	-	6/3	-	-	1/5
-	-	-	-	-	2/8	1/5	-
-	-	-	-	-	-	2/8	-
5/5	-	-	-	-	-	-	-
-	-	-	1/6	-	7/2	-	-
			4/6				

Q: 1 2 3 4 5 6 + 8

d:
tata:
$$\begin{array}{r} 12345678 \\ 50316721 \\ \hline 50263285 \end{array}$$



7.



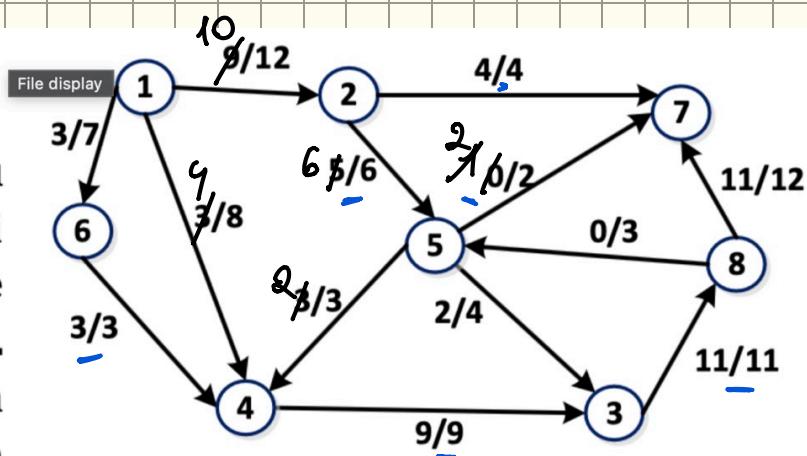
$$\begin{matrix} 1 & 2 & 3 & 4 \\ 0 & 5 & -1 & 1 \end{matrix}$$

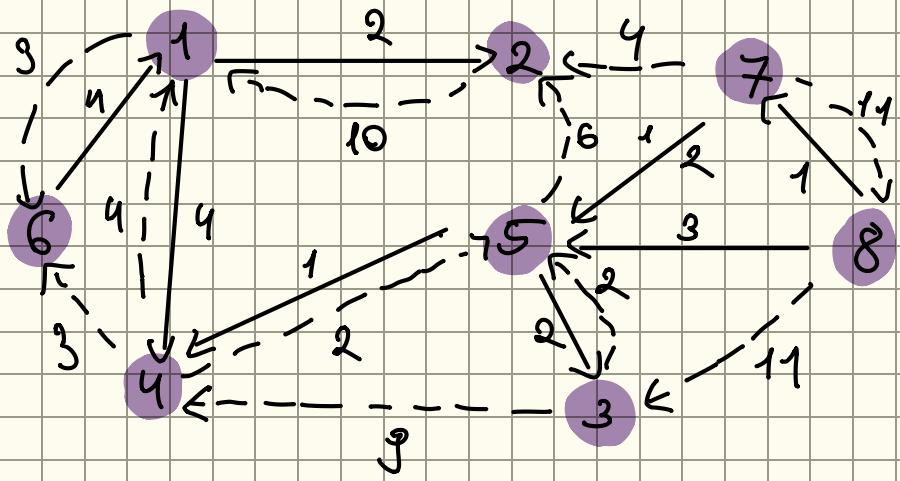
Dacă primim un graf oarecare,
atunci trebuie să mai facem
verificarea

pentru fiecare $u \in E$ execută
daca $d[u] + w(u, v) < d[v]$ atunci

```
return 0;
STOP: ciclu negativ
```

8.





$$1 \xrightarrow{3} 2 \xrightarrow{1} 5 \xrightarrow{2} 7$$

$$1 \xrightarrow{5} 4 \xrightarrow{3} 5 \xrightarrow{1} 7$$

$$\min = 1$$

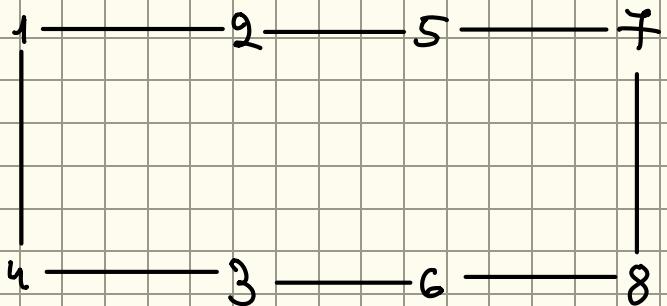
$$1 \longrightarrow 4 \dashrightarrow 5 \longrightarrow$$

Täietuna: 2-7
5-7
3-8

9. a) $G = (V, E)$ - graf hamiltonian

$X =$ multime vârfuri

$X \subset V$



Extragem din G doar ciclul Hamiltonian.

Dacă avem un lant, cu $|V|$ noduri $|E|$ muchii.

În edge-case-ul în care nodurile $v \in X$ nu sunt adiacente
dar $|X|$ maxim \Rightarrow fiecare $v \in X$ împarte ciclul într-o nouă comp conexă.

$$\Rightarrow \text{nrCompConexe} \leq |X|$$

Adaugăm înapoi muchiile din G

Acestea pot lega comp conexe, dar nu le pot face mai multe

$$\Rightarrow \text{nrCompConexe} \leq |X|$$

b) Pp că Graful este ham

\Rightarrow În ciclul ham 1-2-3-4-5-6-7-1

Dacă $X = \{1, 2, 3\}$, atunci graful devine 4 noduri izolate 4, 5, 6, 7

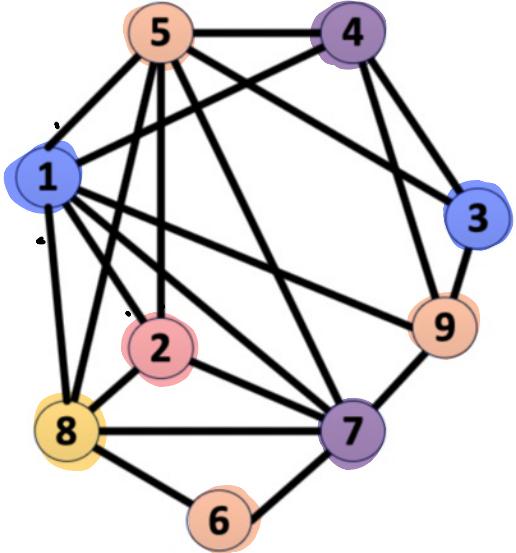
$\Rightarrow \text{nrComplConexe} = 4 > |X|$

\Rightarrow nu există un ciclu ham cu toate nodurile \Rightarrow PP este falsă

c) Graful din imagine.

(10)

11)



v: X X X XX X X XX
 1 2 3 4 5 6 7 8 9
 d: 6 5 3 4 6 2 6 5 9
 5 5 5 4

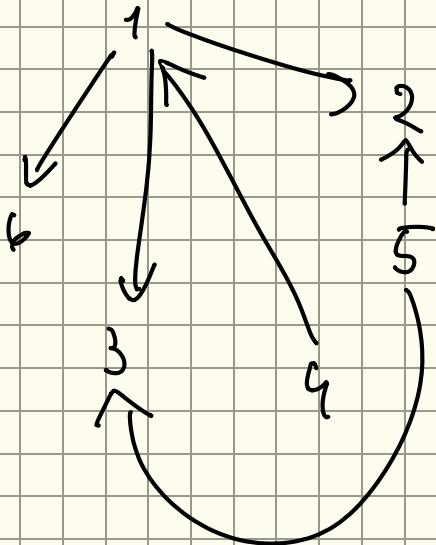
Q: 2 3 4 6 8 9 1 5 7

7
5
1
9
8
6
4
3

S = 2

i=2

12)



Model 4 2022-2023

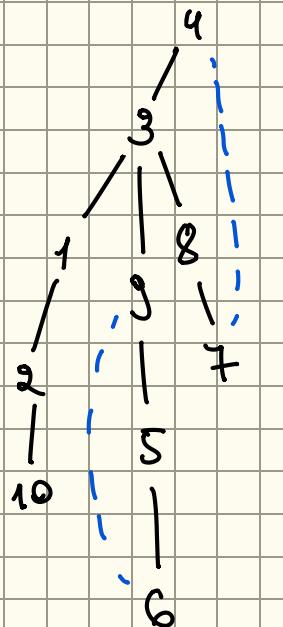
1) O muchie critică este o muchie care dacă ar fi scoasă ar crea o altă componentă conexă.

În acest sens mc sunt: 2-10

2-1

3-9

2) Viz: 1 2 3 4 5 6 7 8 9 10
X X X X X X X X X X

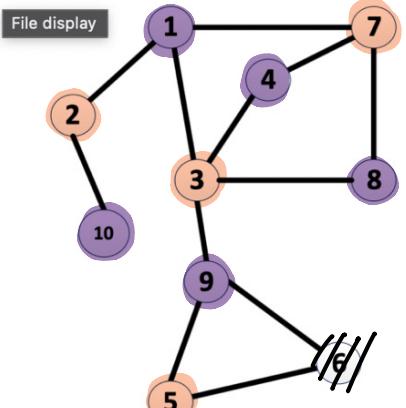


→ algoritmul parcurge recursiv vecinii nodului în adâncime

→ dacă a ajuns la un nod fără vecini se întoarce din recursie până la urm nod cu vecini nevizitati

→ dacă găsește o muchie de întoarcere ($v[i][vecin] = -1$) ignoră acel path și nu merge mai departe

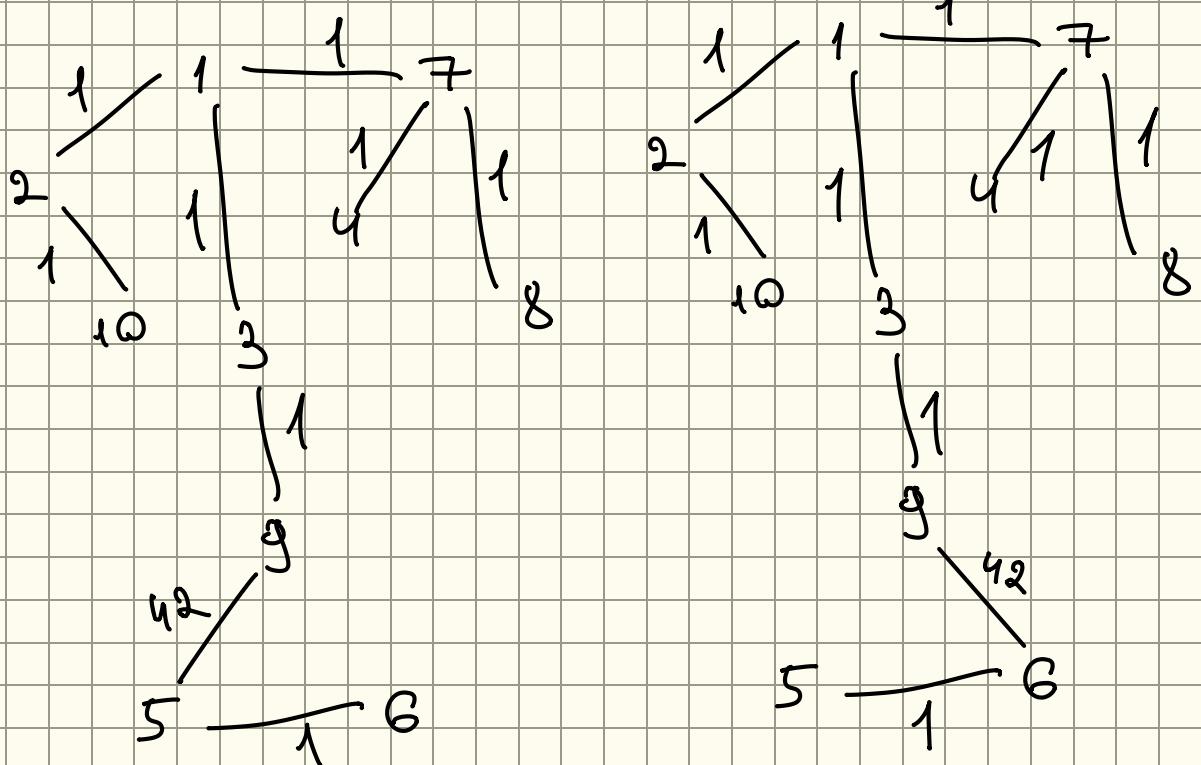
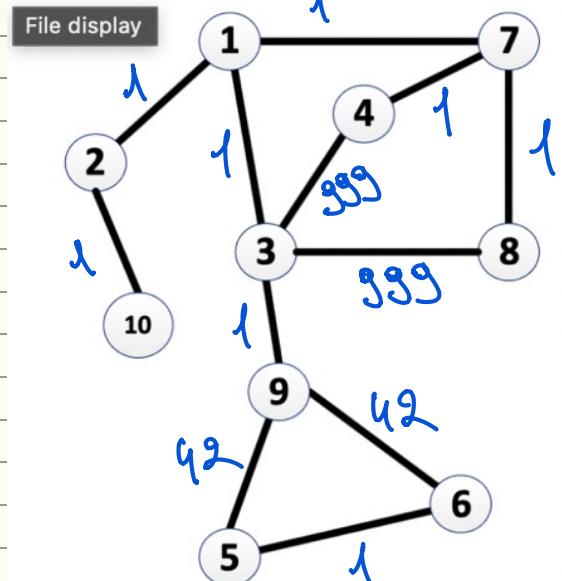
3)



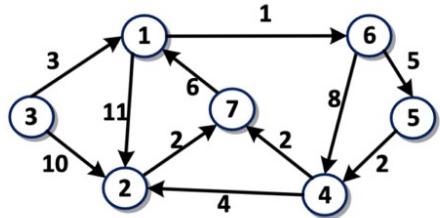
Am colorat cu 2 culori graful
iar componenta biconexă 5-6-9
este un graf complet cu 3 noduri
deci nr ei cromatic este exact
3, de aceea un nod din ea
trebuie neapărat scos, am
ales 6.

\Rightarrow Bipartitia: $(2, 3, 5, 7) \cup (1, 4, 8, 9, 10)$

4.



5.

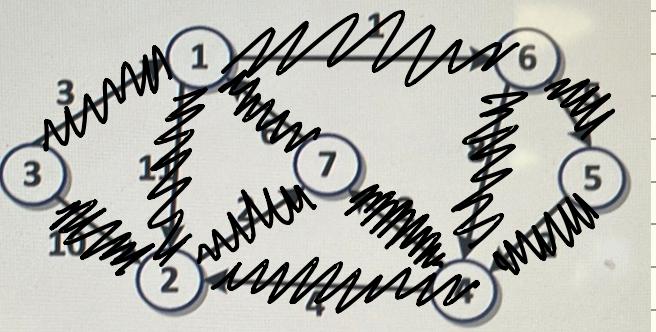


1	2	3	4	5	6	7
$\infty/0$	$\infty/0$	$0/0$	$\infty/0$	$\infty/0$	$\infty/0$	$\infty/0$
$3/3$	$10/3$	$0/0$	—	—	—	—
$3/3$	—	—	—	—	$4/1$	—
—	—	—	$12/6$	$9/6$	$4/1$	—
—	—	—	$11/5$	$9/6$	—	$12/2$
—	$10/3$	—	—	—	—	$12/2$
—	—	$11/5$	—	—	—	$12/2$

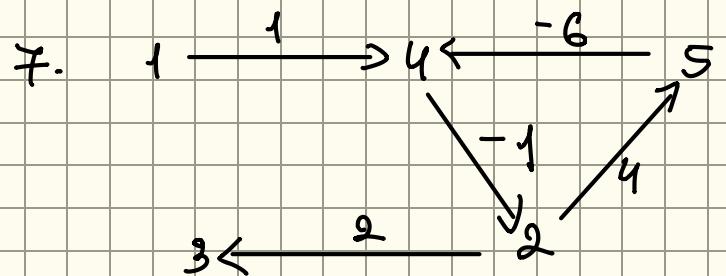
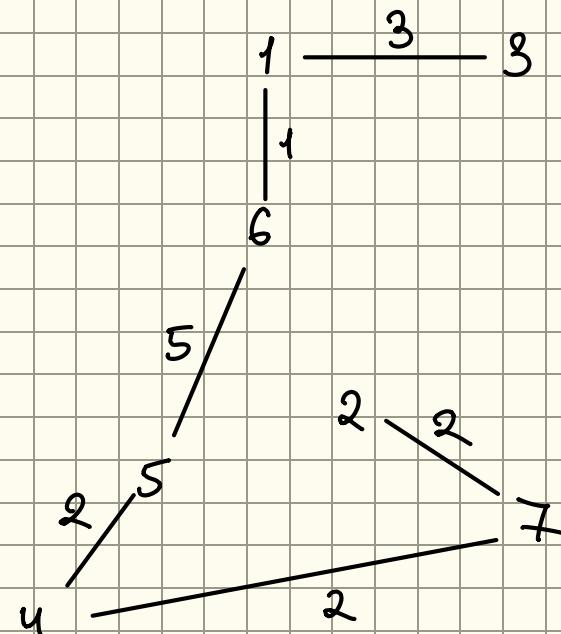
Q: 1 2 3 4 5 6 7

d : 1 2 3 4 5 6 7
 $tata$: 3 3 0 5 6 1 2

6) Kruskal sortează muchiile crescător după pondere.



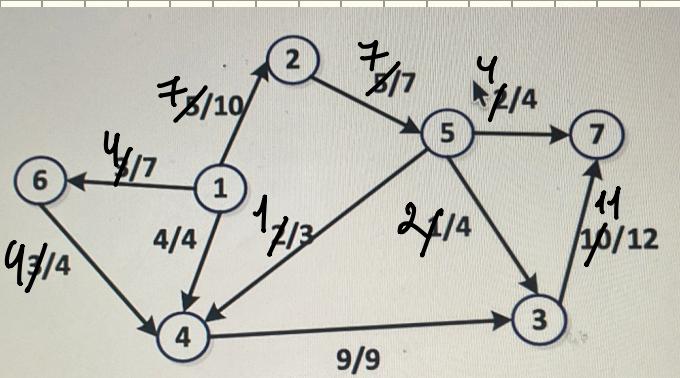
$$\begin{array}{r}
 \bullet 1 \xrightarrow{1} 6 \\
 \bullet 2 \xrightarrow{2} 7 \\
 \bullet 4 \xrightarrow{2} 5 \\
 \bullet 4 \xrightarrow{2} 7 \\
 \bullet 3 \xrightarrow{3} 1 \\
 \times 2 \xrightarrow{4} 9 \\
 \bullet 6 \xrightarrow{5} 5 \\
 \times 1 \xrightarrow{6} 7 \\
 \times 6 \xrightarrow{8} 4 \\
 \times 3 \xrightarrow{10} 2 \\
 \times 2 \xrightarrow{11} 1
 \end{array}$$



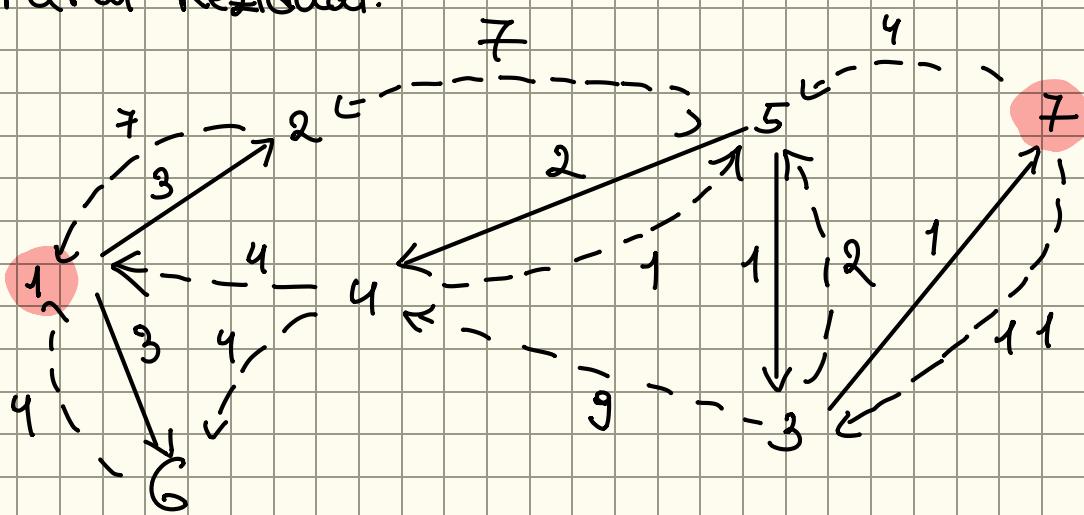
→ Deja facut.

1	2	3	4	5
0	∞	∞	∞	∞

8.



Grafal Residual:



$$1 \xrightarrow{5} 2 \xrightarrow{2} 5 \xrightarrow{2} 7 \quad \min = 2.$$

$$1 \xrightarrow{4} 6 \xrightarrow{1} 4 - \xrightarrow{2} 5 \xrightarrow{3} 3 \xrightarrow{2} 7 \quad \min=1$$

$$S = \{1, 2, 6\}$$

$$T = \{3, 4, 5, 7\}$$

2 - 5

$$6 = 9$$

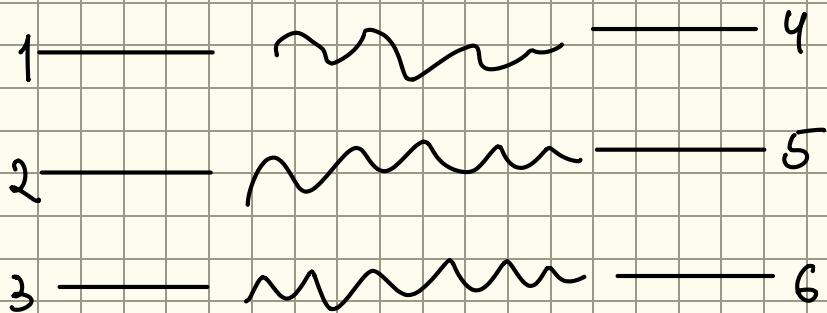
$$d - y$$

9. G - neonorientat conex

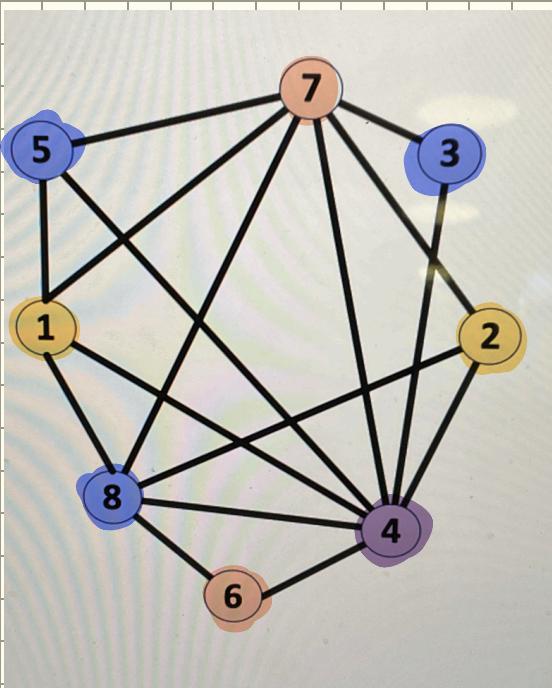
k - noduri - grad impar

$\exists [k/2]$ lanturi simple care nu au muchii în comun

$$k=6 \quad k/2 = 3$$



11



V X X X X X X X
1 2 3 4 5 6 7 8
d 4 3 2 7 3 2 6 5
8
5

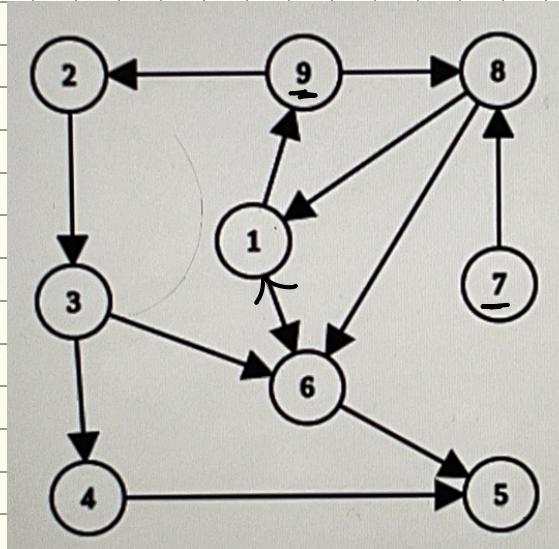
Q: 1 2 3 5 6 8 7 4

4
7
8
6
5
3
2
1

S: _____

SUBIECT 1 2022

1.



1) Nu admite, are mai multe cicluri printre care $8 \rightarrow 1 \rightarrow 9 \rightarrow 8$.

Dacă eliminăm $1 \rightarrow 9$ nu mai avem cicluri și admite top sort.

$$Q = 7 \ 8 \ 2 \ 1 \ 3 \ 4 \ 6 \ 5$$

$$i = 7 \ 8 \ 2 \ 1 \ 3 \ 4 \ 6 \ 5$$

$$\begin{matrix} d: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \text{---:} & x & x & x & x & x & 8 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 1 & 0 \end{matrix}$$

TS: 7, 9, 8, 2, 1, 3, 4, 6, 5

$$2. \quad \{7\}$$

$$\{8, 9, 1\}$$

$$\{2\}$$

$$\{3\}$$

$$\{4\}$$

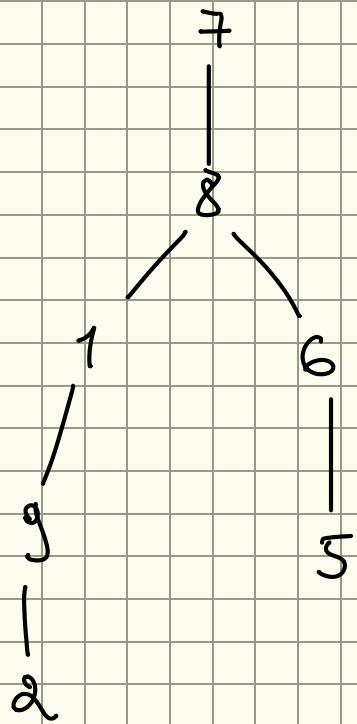
$$\{6\}$$

$$\{5\}$$

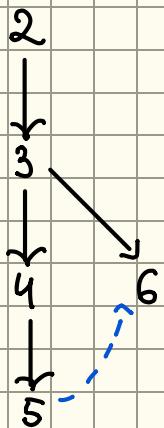
3. BF(7)

Viz: X 1 2 3 4 5 6 7 8 9
X X X X X X X X

Q: 7 8 1 8 8 5 2



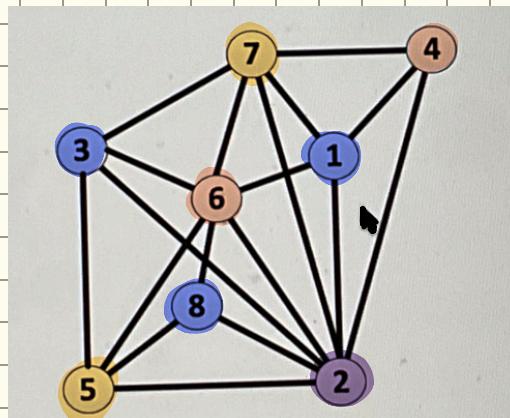
4) df(2)



5.

	e	v	i	a	t	a
	0	1	2	3	4	5
e	0	0	1	2	3	4
r	1	1	1	2	3	4
e	2	2	2	2	3	4
s	3	3	3	3	3	4
t	4	4	4	4	4	3
q	5	5	5	5	4	3
n	6	6	6	5	5	4
t	7	7	7	6	5	5
a	8	8	8	8	7	6

\Rightarrow distanta de editare 5.



V:	X	X	X	X	X	X
1	2	3	4	5	6	7
d:	4	7	4	3	4	8

Q: 1 3 4 5 7 8 6 2

Sortare: 2 6 8 7 5 4 3 2 1

S: 1

Graful este planar deoarece $m \leq 3n - 6$

$\exists x \in V \text{ cu } d(x) \leq 5$

7. Problema se rezolvă la găsirea unui AFCM

Potem folosi algoritmul lui Prim cu minheap.

PRIM(G, W, S)

pentru fiecare $u \in V$ execută

$$\begin{cases} d[u] = \infty \\ \text{tata}[u] = 0 \end{cases}$$

initializăm Q cu cuV

cât timp $Q \neq \emptyset$ execută

$u =$ extragem vârf din Q cu eticheta minimă.

pentru fiecare $v \sim u$ execută

dacă $v \in Q$ și $\text{cost}(u, v) < d[v]$ atunci

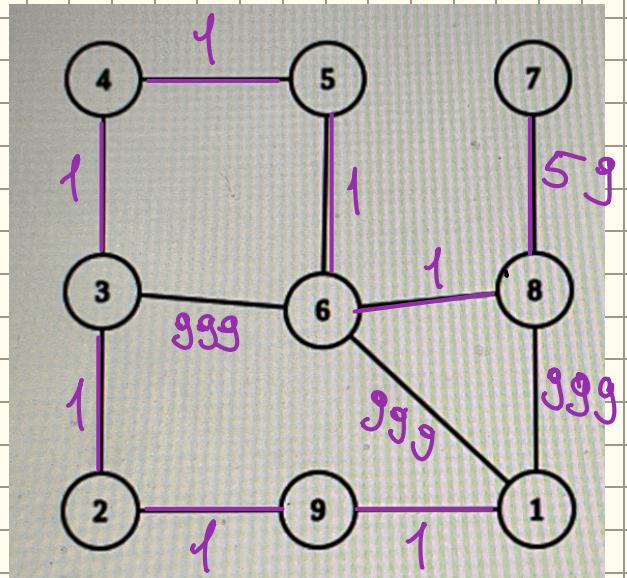
$$\begin{cases} d[v] = \text{cost}(u, v) \\ \text{tata}[v] = u \end{cases}$$

repară min heap Q .

sonie APCM.

MODEL 2 2021-2023

4:

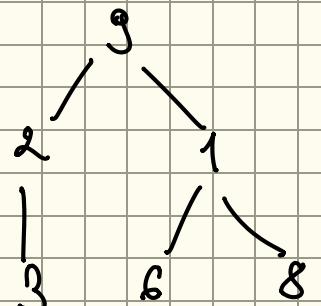


1. 8

2. 7-8

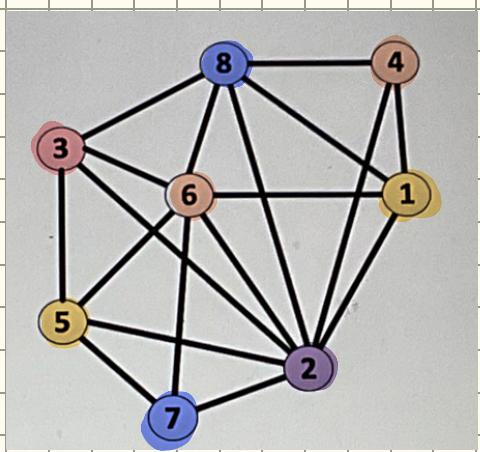
3. bf(9)

1 2 3 4 5 6 7 8 9
V X X X X X X X X



Q: 8 1 2 6 8 3

	0	1	2	3	4	5	6
	E	e	x	a	m	e	n
5.	0	E	0	1	2	8	9
	1	m	1	1	2	3	3
	2	a	2	2	2	2	4
	3	n	3	3	3	3	4
	4	i	4	4	4	4	5
	5	r	5	5	5	5	5
	6	e	6	5	6	6	6



v: X X X X X X X X
1 2 3 4 5 6 7 8

d: 4 7 9 3 9 6 3 5
6 5

Q: X 3 4 5 7 8 6 2

2
6
8
7
5
4
3

S: 1

Sortare: 2 6 8 7 5 4 3 1

Colorare: C₄ C₁ C₅ C₂ C₄ C₂ C₃ C₃

$$18 \leq 18$$

$$18 \leq 24 - 6$$

$$18 \leq 3 \cdot 8 - 6$$

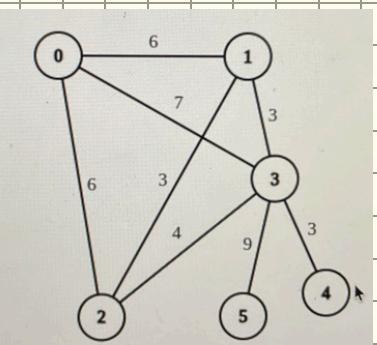
1. $m \leq 3n - 6$
2. $d(v_1) = 4 \leq 5$

$\left. \begin{array}{l} m \leq 3n - 6 \\ d(v_1) = 4 \leq 5 \end{array} \right\} \Rightarrow \text{graful este planar.}$

7) Considerăm

Volumetria
si date
0

MODEL 3 2021-2022



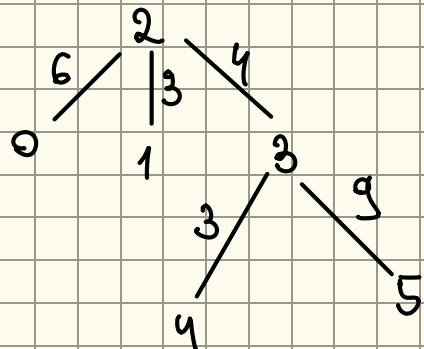
1.	0	1	2	3	4	5
	$\infty/0$	$\infty/0$	$\infty/0$	$\infty/0$	$0/0$	$\infty/0$
	—	—	—	$3/4$	$0/0$	—
	$10/3$	$6/3$	—	$3/4$	—	—

4
3
3
7
0

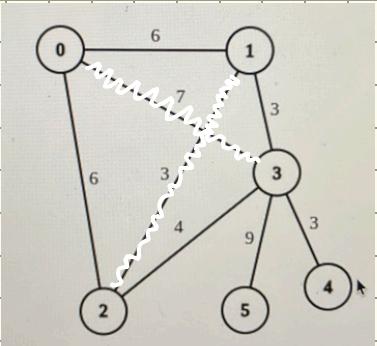
2.

	0	1	2	3	4	5
$\infty/0$	$\infty/0$	$0/0$	$\infty/0$	$\infty/0$	$\infty/0$	$\infty/0$
$6/2$	$3/2$	$0/0$	$4/2$	—	—	—
—	$3/2$	—	—	—	—	—
—	—	—	$4/2$	$3/3$	$9/3$	—
—	—	—	—	$3/3$	—	—
$6/2$	—	—	—	—	—	—

Q: 0 1 2 3 4 5

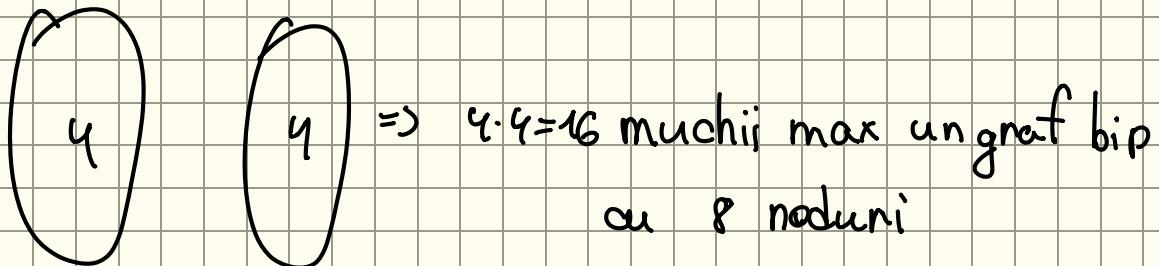
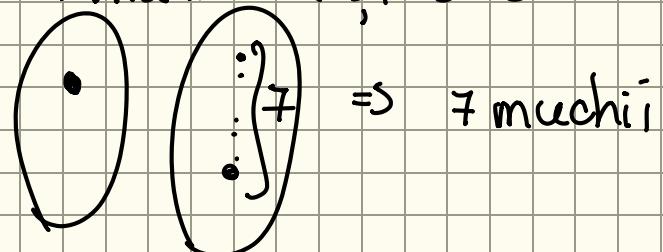


3.

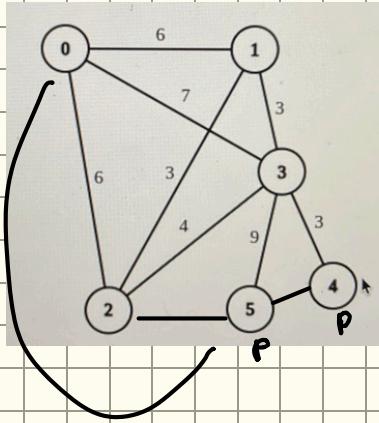


Graful nu este bipartit deoarece are cicluri impare.

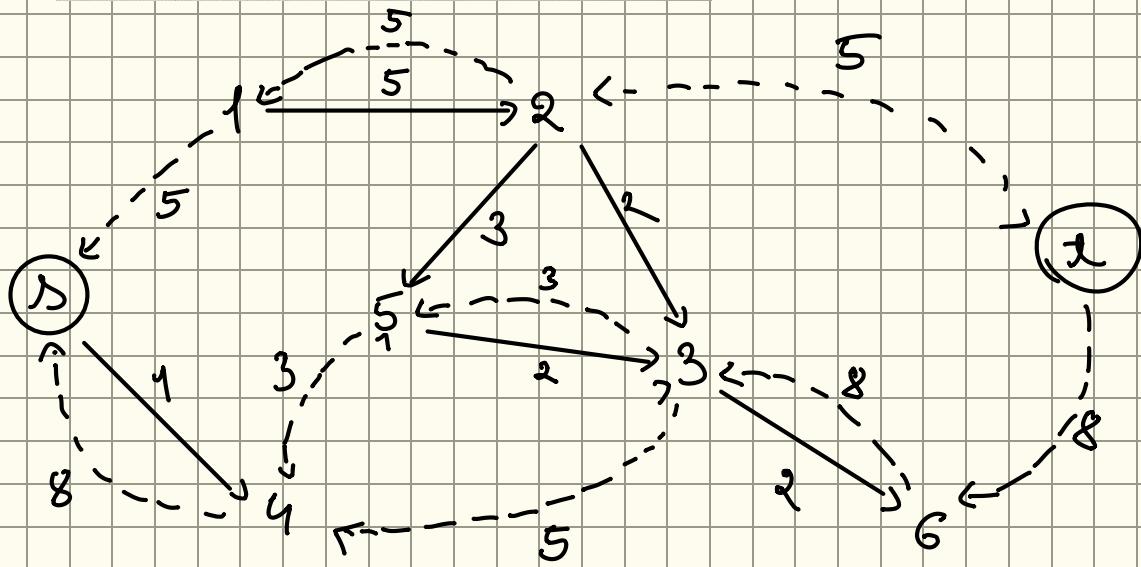
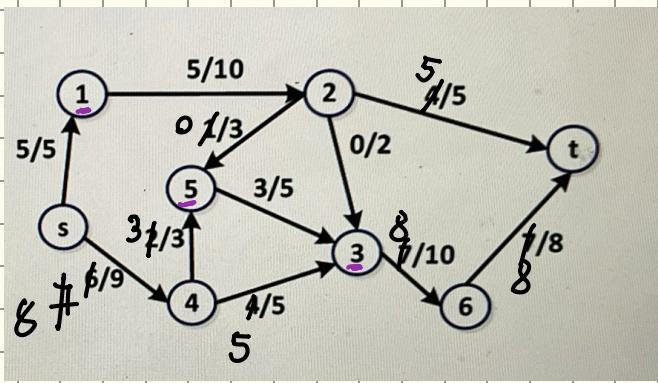
Eliminăm 1-2 și 0-3



4. Nu există deosebite care noduri de grad impar.



5.



$$D \xrightarrow{3} 4 \xrightarrow{1} 5 \xrightarrow{-1} 2 \xrightarrow{1} t$$

$$D \xrightarrow{2} 4 \xrightarrow{1} 3 \xrightarrow[3]{\quad} 6 \xrightarrow[1]{\quad} t \quad \min=1.$$

înălțime: $S = \{D, 4\}$

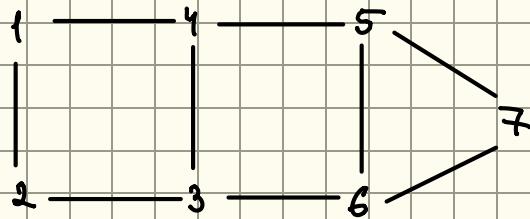
$T = \{1, 5, 3, 2, 6, t\}$

muchiile: $D-1$

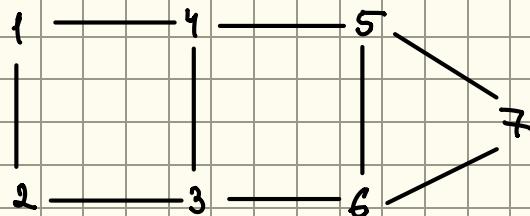
$4-5$

$4-3$

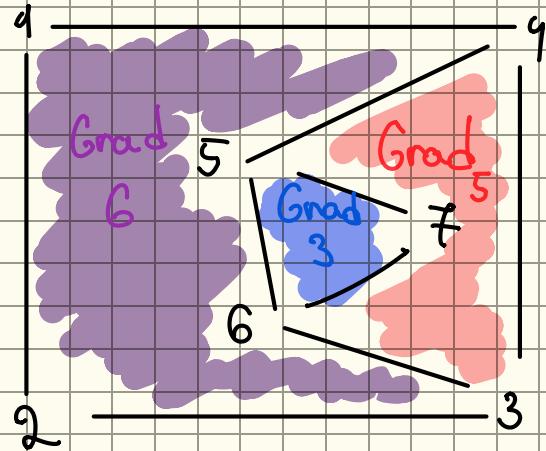
6. Graful:



Harta cu 2 fete gr 4



Harta cu fețe fără gr 4



b) $M = (V, E, F)$

$n > 6$

m muchii

$$d(v) \geq 9 \quad \forall v \in V$$

$$m \leq 3n - 6$$

M are cel puțin 6 vârfuri cu grad ≤ 5 .

$$\sum_{v \in V} d(v) \geq 4n$$

$$n - m + f = 2$$

$$3f \leq 2m \Rightarrow f \leq \frac{2m}{3}$$

$$n-m + \frac{2m}{3} \geq 2$$

$$m \leq 3n-6$$

$$n \geq 7$$

$$d(v) \geq 4 \quad \forall v \in V$$

$$m \leq 3n-6$$

$$2m \leq 6n-12 \quad 2m \leq$$

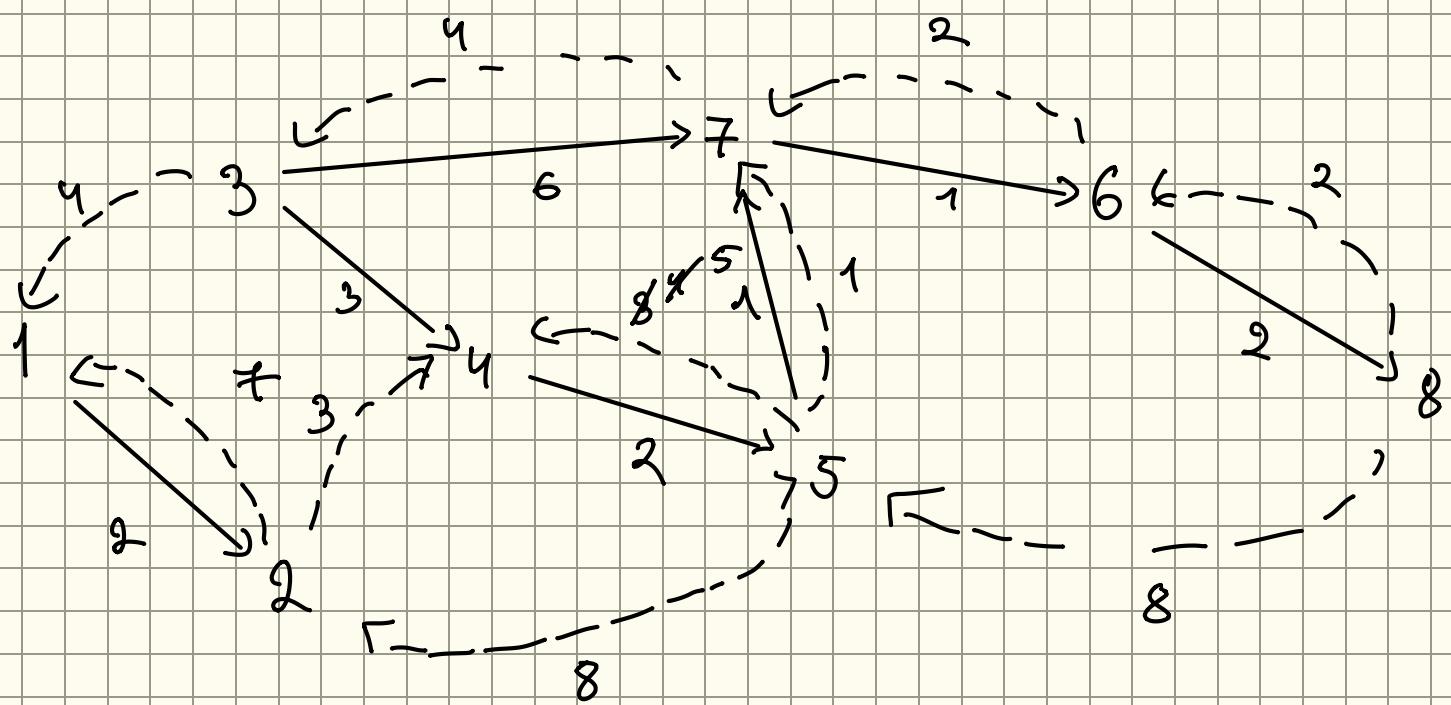
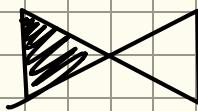
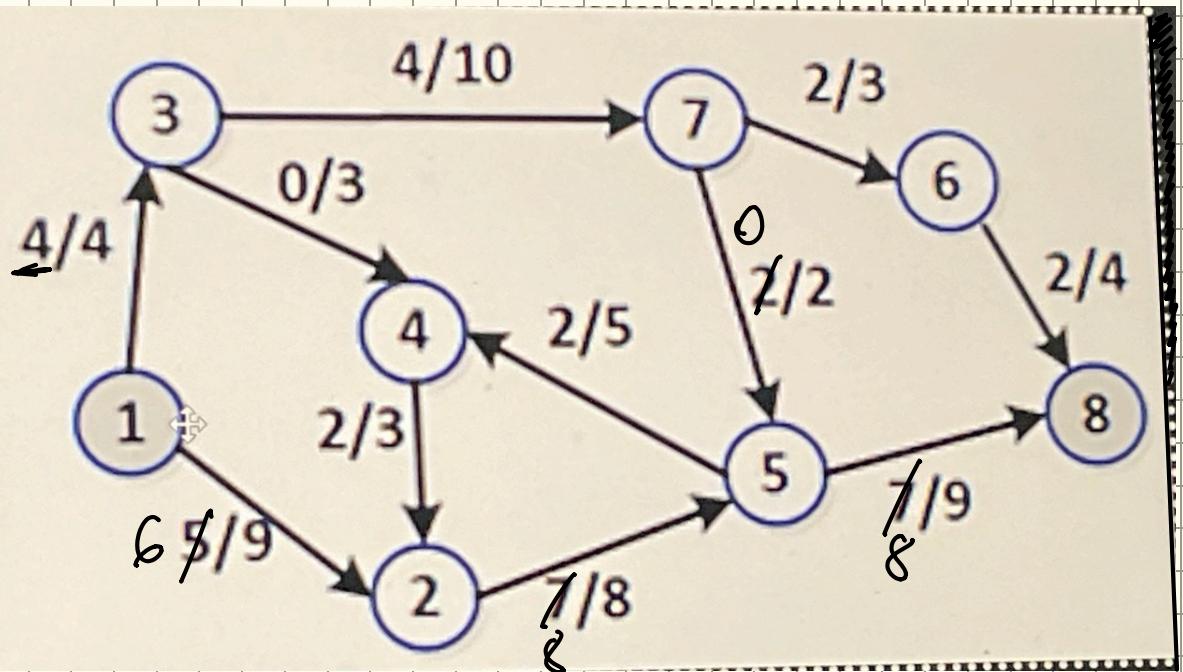
.

.

.

9

.



$$1 \xrightarrow{4} 2 \xrightarrow{1} 5 \xrightarrow{2} 8$$

$$1 \xrightarrow{3} 2 \xrightarrow{1} 4 \xrightarrow{3} 5 \xrightarrow{1} 8$$

$n > 3$

$$\min(d(v)) = 2 \quad \forall v \in V$$

$$|d(v) \leq 5| = 3? \quad \forall v \in V$$

$$m \leq 3n - 6$$

$$n - m + f = 2$$

$$2m \leq 6n - 12$$

$$\sum_v d(v) = 2m$$

$$\sum d(v) \leq 6n - 12$$

$$\sum d(v) \geq 8$$

$$2m \geq 8$$

$$m > 4$$

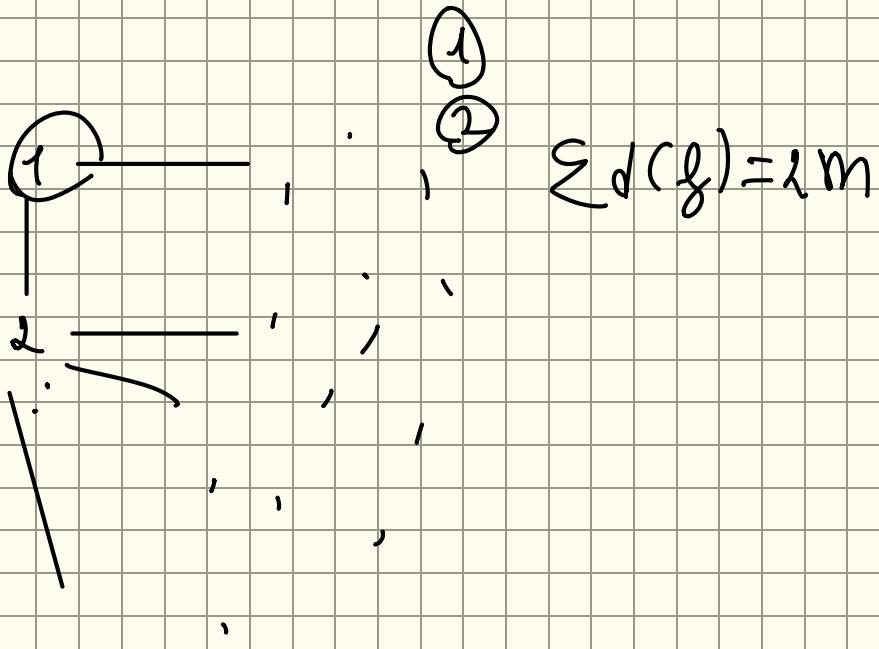
$$\sum d(f) = 2m$$

PP ca } $|d(v)| \leq 5| < 3 \Leftrightarrow$

$$\sum d(v) = 2m$$

$$\sum d(v) \geq 2 \cdot 2 + (n-2) \cdot 6 = 4 + 6n - 12 = 6n - 8$$

$$2m \geq 6n - 8 \quad m > 3n - 4 \quad \text{FALSE}$$



$$m \leq 3n - 6$$

$$\sum d(v) \geq 2$$

$$\sum d(v) \geq 2m$$

$$n - m + f = 2$$

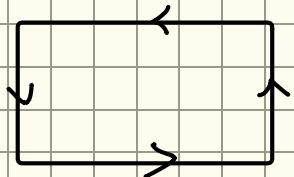
$$n \leq 3m - 6$$

$$\sum d(f_i) = 2m \quad \forall i$$

M

pp cā

$d(v)$ par



.....

$$\text{If } x, y \in V \text{ und } d(x)^- - d(x)^+ = -1$$

$$d(\bar{y}) - d(y)^+ = 1$$

Model examen 2025

Toekomst
is een beschrijvende
conceptie (C)

1) $\{1, 2, 3, 5\}$

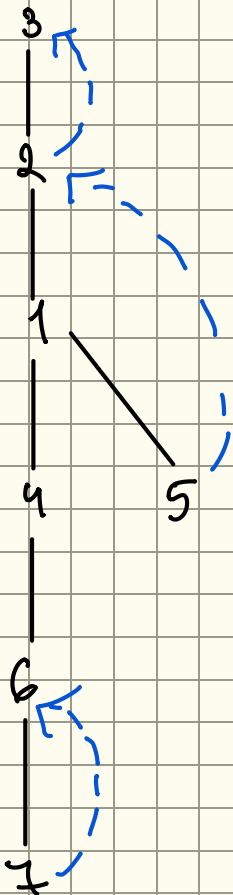
$\{6, 7\}$

$\{9\}$

2) $\text{dfs}(3)$

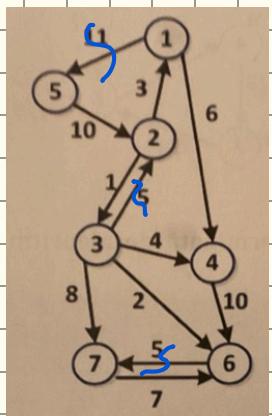
viz: 1 2 3 4 5 6 7

viz: $\begin{matrix} \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} \end{matrix}$



3. G admite top sort \Leftrightarrow nu are cicluri.

Scoatem 1-5; 3-2; 6-7



Gradu Intrare (muchii care intră în)

↓	1	2	3	4	5	6	7
d-	X	X	X	2	0	3	X
d+	0	0	0	1	2	0	0
	0	0	0	0	0	0	0

$$Q: \cancel{5} \cancel{2} \cancel{1} \cancel{3} \cancel{4} \cancel{7} \cancel{6}$$
$$i = \cancel{5} \cancel{2} \cancel{1} \cancel{3} \cancel{4} \cancel{7}$$

Top sort: 5, 2, 1, 3, 4, 7, 6

4. G are drum eulerian $\Leftrightarrow d^-(v) = d^+(v) \quad \forall v \in V$

SACU

$$\forall v \in V \quad d^-(v) - d^+(v) = -1$$

$$\forall v \in V \quad d^-(v) - d^+(v) = 1$$

$$\forall v \in V \setminus \{x, y\} \quad d^-(v) - d^+(v) = 0.$$

nod v	$d^-(v)$	$d^+(v)$	$d^-(v) - d^+(v)$
1	1	2	-1 0
2	2	2	0
3	1	4 3 2	-8 2 1
4	2	1	1 0
5	1	1	0
6	3 2	1	2 1
7	2	1	1 0

Stengem:

1-4
3-6
2 ~ 3

$$\Rightarrow d^-(3) - d^+(3) = -1$$

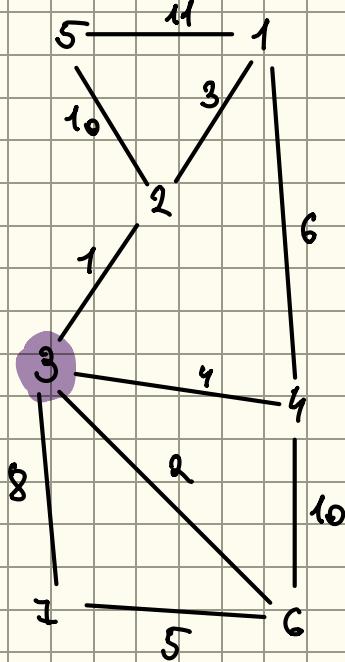
$$d^-(6) - d^+(6) = 1$$

$$d^-(v) - d^+(v) = 0 \quad \forall v \in \{1, 2, 4, 5, 7\}.$$

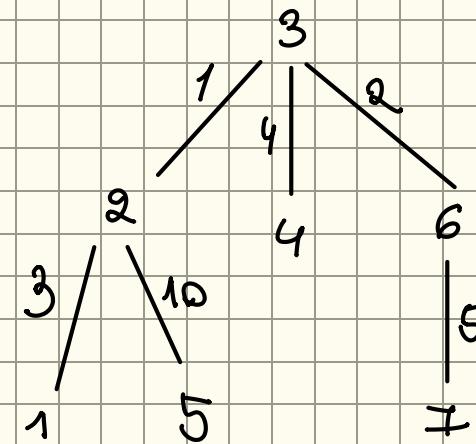
5. if ($d[j][k] < d[j][i] + d[i][k]$)
 $[\quad d[j][k] = d[j][i] + d[i][k];$
 $\quad *$

	1	2	3	4	5	6	7
1	∞	∞	∞	6	11	∞	∞
2	3	∞	1	∞	∞	∞	∞
3	∞	5	∞	4	∞	2	8
4	∞	∞	∞	∞	∞	1000	
5	∞	10	∞	∞	∞	∞	∞
6	∞	∞	∞	∞	∞	5	
7	∞	∞	∞	∞	7	∞	

6.



(1)	(2)	(3)	(4)	(5)	(6)	(7)
$\infty/0$	$\infty/0$	$0/0$	$\infty/0$	$\infty/0$	$\infty/0$	$\infty/0$
—	1/3	0/0	4/3	—	2/3	8/3
3/2	1/3	—	—	10/2	—	—
—	—	—	—	—	2/3	5/6
3/2	—	—	—	—	—	—
—	—	—	4/3	—	—	—
—	—	—	—	—	5/6	—
—	—	—	4/3	—	—	—



Cost APCM = 25.

7. $P(n)$ - adăugarea unei muchii de cost minim uvr cu $u \in C$ și $v \notin C$, $v \notin T$
 $\Rightarrow APCM$.

$P(1)$: legă două noduri izolate $\Rightarrow APCM$ adevărat.

$P(k) \rightarrow P(k+1)$

$P(k)$: un APCM valid notat C

$P(k+1)$: Adăugarea unei muchii de cost minim la C notată e

Presupunem $P(k)$ adevărat (C APCM valid)

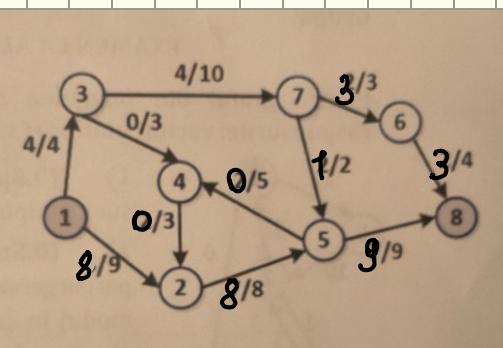
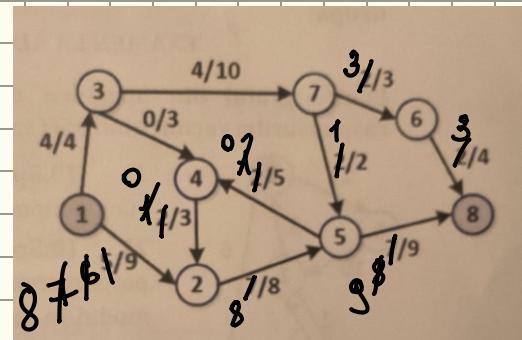
$C - APCM \Rightarrow \text{cost}(C)$ este minim

e -minim $\Rightarrow \text{cost}(e)$ este min dintre muchii (muchia de cost min va apărea sigur în APCM)

$\Rightarrow \text{cost}(C+e) = \text{minimul posibil până la pas } k+1$ al alg.

\Rightarrow Prin metod. induc. mat $P(n)$ adevărat.

8.



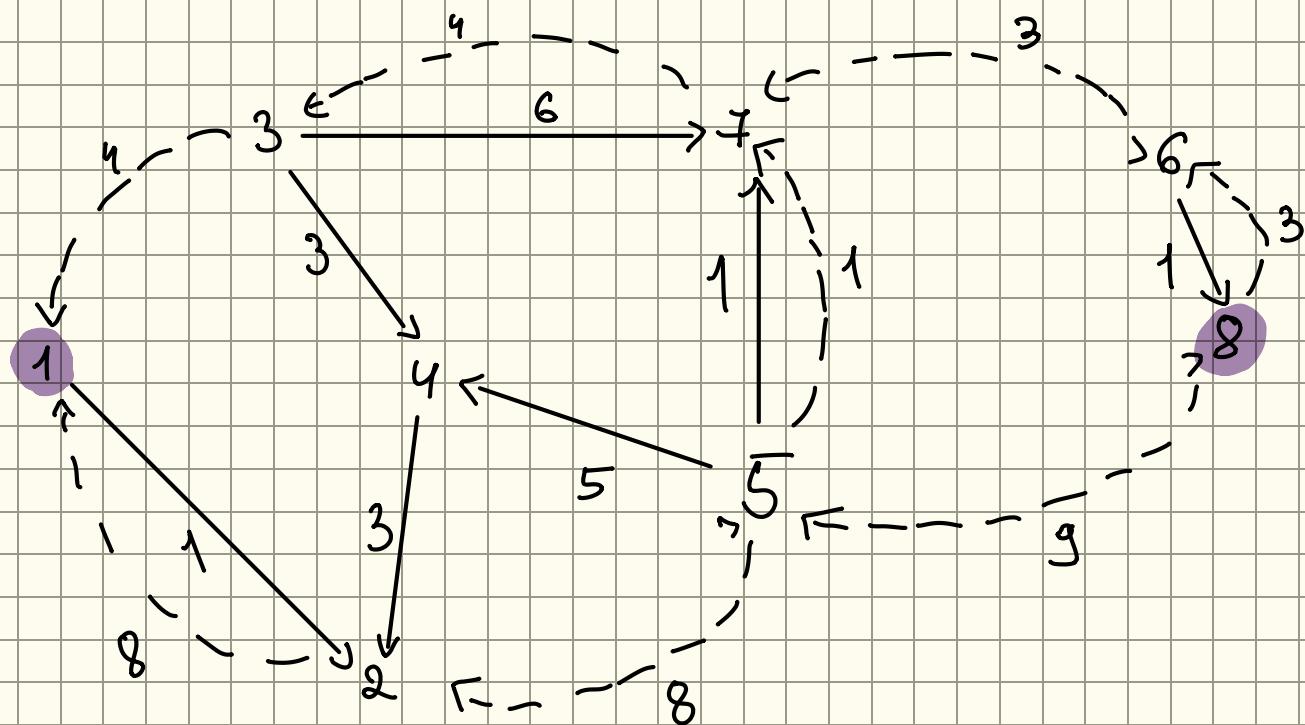
Tăietura

$$\{1, 2\}$$

$$\{3, 4, 5, 6, 7, 8\}$$

$$\begin{cases} 1-3 \\ 2-5 \end{cases}$$

$$\Rightarrow \text{flux} = 9 + 8 = 12$$



$$1 \xrightarrow{4} 2 \xrightarrow{1} 5 \xrightarrow{2} 8 \quad \text{min=1}$$

$$1 \xrightarrow{3} 2 \xrightarrow{2} 4 \xrightarrow{2} 5 \xrightarrow{1} 8 \quad \text{min=1}$$

$$1 \xrightarrow{2} 2 \xrightarrow{1} 4 \xrightarrow{1} 5 \xrightarrow{1} 7 \xrightarrow{1} 6 \xrightarrow{2} 8 \quad \text{min=1}$$

$$9. \chi(K_7) = 7.$$

a) nu poate fi planar deoarece $\chi(G) = 7 \Leftrightarrow d(v) = 6 \forall v \in V$

pt ca un graf să fie planar trebuie $\exists v \in V$ cu $d(v) \leq 5$

b)

c)

10.

	0	1	2	3	4	5	6
	E	C	A	P	A	C	E
0	E	O	O	O	O	O	O
1	P	O	O	1	1	1	1
2	a	O	0	1	1	2	2
3	I	O	0	1	1	2	2
4	e	O	0	1	1	2	2
5	t	O	0	1	1	2	2
6	a	O	0	1	1	2	2

$$dp[i][0] = 0 \quad \forall i$$

$$dp[0][j] = 0 \quad \forall j$$

$$dp[i][j] = \begin{cases} dp[i-1][j-i] + 1 & dp[S1[i]] == S2[j] \\ \max(dp[i-1][j], dp[i][j-1]) & S1[i] != S2[j] \end{cases}$$

$$dp[i][j] = LCS(S1[0:i], S2[0:j])$$

