

## Notiuni introductive

Multiset = o "multime" unde elementele se pot repeta

Proprietăți graf orientat

$$G = (V, E)$$

V - fizită

E - perechi (ordonate) de 2

elemente distincte din V

$v \in V$  - vîrf

$$e = \{u, v\} = uv - arc$$

$u = e^-$  - vîrf initial / origine / extremitate

$$d_G^-(u) = d_G^+(u) = |E|$$

initială

$v = e^+$  - vîrf final / terminus / extremitate finală

$$\text{grad} = d^- + d^+$$

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

Pentru un multigraf orientat / neorientat :

$G = (V, E, r)$ , unde  $r(e)$  este multiplicitatea muchiei e

$$e = \{u, v\} = buclă$$

e cu  $r(e) > 1$  = muchie multiplă / arc multiplu

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

Fie  $G = (V, E)$  un graf

- $u$  și  $v$  sunt adiacente dacă  $uv \in E$

- Un vecin al lui  $u \in V$  este un vîrf adiacent cu el

- $N_G(u)$  = multimea vecinilor lui u

unde elementele se pot repeta

Proprietăți graf neorientat

$$G = (V, E)$$

V - fizită

E - submultime de 2 elemente (distincte) din V

$v \in V$  - vîrf / nod

$$e = \{u, v\} = (u, v)$$

= uv - muchie

$u, v$  capete / extremități

grad = numărul de muchii adiacențe

$$d_G(u) = 2 |E|$$

grad interior =  $d^-$   
= intră în

grad exterior =  $d^+$   
= ieșe din

$$\text{grad} = d^- + d^+$$

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

• O muchie  $e \in E$  este incidentă cu un vârf și dacă nu este extremitate a lui  $e$

• și  $f \in E$  sunt adiacente dacă există un vârf în care sunt incidente (au o extremitate în comun)

! Pentru grafuri orientate avem drum și pentru grafuri neorientate avem lant, iar la orientare avem vârfuri și la neorientare avem noduri.

Fie  $G$  un graf orientat/neorientat

• Un drum/lant este o secvență  $P$  de vârfuri

$$P = [v_1, v_2, \dots, v_{k-1}, v_k]$$

atfel încât între oricare două vârfuri consecutive există arc/muchie

$P$  este drum/lant simple dacă nu conține un arc/muchie de mai multe ori.

$P$  este drum/lant elementar dacă nu conține un vârf de mai multe ori.

Lungimea lui  $P = l(P) = k - 1 = |E(P)|$  - număr de cerce/muchii lui  $P$

$v_1$  și  $v_k$  se numesc capetele/extremitățile lui  $P$

$P$  se numește și  $v_1 - v_k$  drum

$$V(P) = \{v_1, v_2, \dots, v_k\}$$

$$e_i = (v_i, v_{i+1})$$

$$E(P) = \{e_1, e_2, \dots, e_{k-1}\}$$

$[v_i \underline{P} v_j] =$  subdrumul lui  $P$  între  $v_i$  și  $v_j$

Pentru două varfuri  $u$  și  $v$  definim distanță de la  $u$  la  $v$  astfel:

$$d_G(u, v) = \begin{cases} 0, & \text{dacă } u=v \\ \infty, & \text{dacă nu există } u-v \text{ drum / lant} \\ \min\{\ell(p)\} & \text{Pe } u-v \text{ drum / lant în } G \end{cases}$$

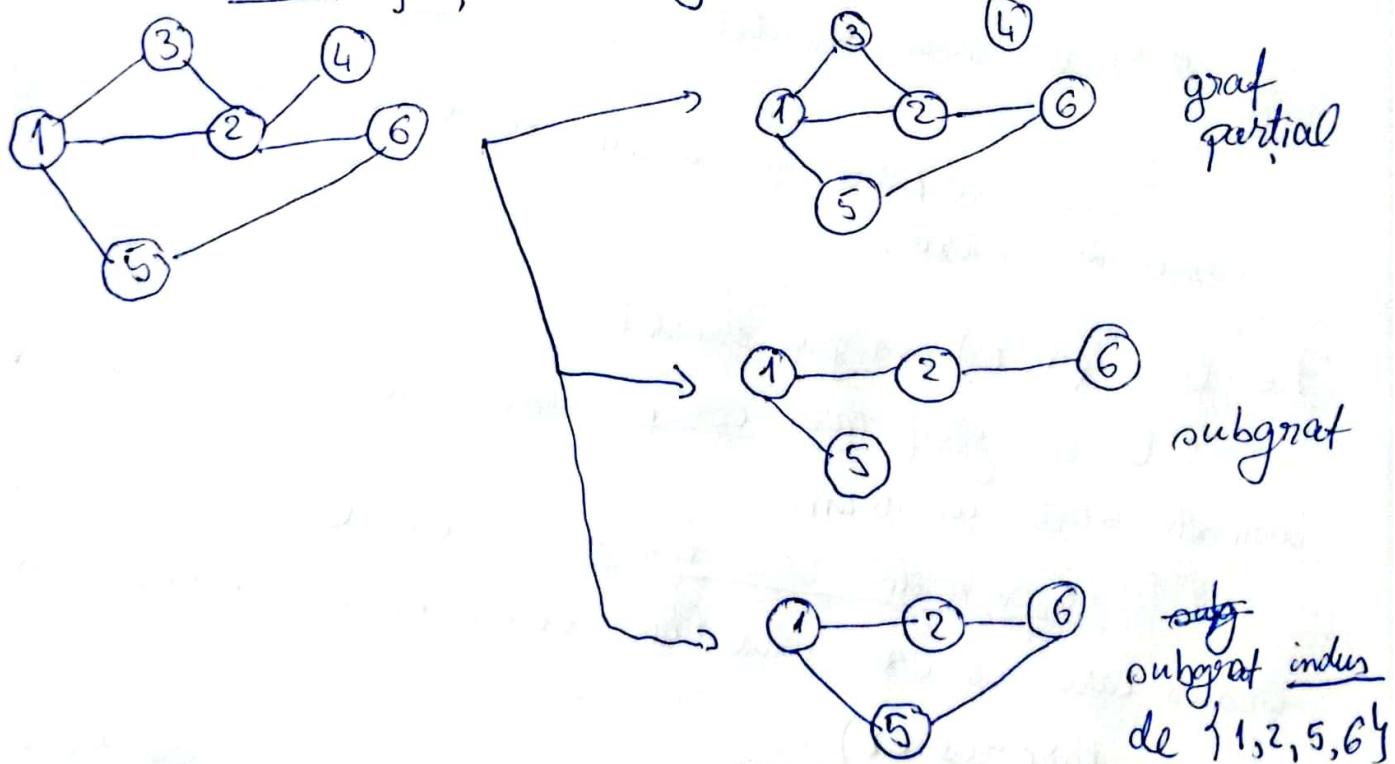
Un drum  $u-v$  de lungime  $d_G(u, v)$  se numește drum minim de la  $u$  la  $v$ .

Vor circuit / ciclu este un drum / lant simple cu capetele identice  $C = [v_1, v_2, \dots, v_k, v_1]$

Circuit / ciclu elementar: drumul / lantul  $[v_1, v_2, \dots, v_k]$  este elementar.

Notări  $V(C)$ ,  $E(C)$

Graf parțial. Subgraf:



Fie  $G = (V, E)$  și  $G_1 = (V_1, E_1)$  două grafuri.

- $G_1$  este subgraf induș al lui  $G$  dacă  $G_1 = G[V_1]$

$V_1 \subseteq V$  și  $e \in E(G)$ ,  $e$  are amebele extremități în  $V_1$ .

•  $G_1$  este graf partor al lui  $G$  ( vom nota  $G_1 \leq G$ )

dacă:  $V_1 = V$ ,  $E_1 \subseteq E$

•  $G_2$  este subgraf al lui  $G$  ( vom nota  $G_2 \preceq G$ )

dacă:  $V_2 \subseteq V$ ,  $E_2 \subseteq E$

### Conexitate

Fie  $G = (V, E)$  un graf neorientat neorientat

•  $G$  este graf conex dacă între orice două vîrfuri distincte există un lanț.

• O componentă conexă a lui  $G$  este un subgraf inducător conex maximal (care nu este inclus în alt subgraf conex)

• Pentru orul orientat - tare-conexitate

• Toate vîrfurile și muchiile lui  $G$  aparțin unei singure componente conexe.

Fie  $G = (V, E)$  un graf orientat

•  $G$  este graf tare-conex dacă între oricare două vîrfuri distincte există un drum

• O componentă tare-conexă a lui  $G$  este un subgraf inducător tare-conex maximal (care nu este inclus în alt subgraf tare-conex)

• Toate vîrfurile lui  $G$  aparțin unei singure componente tare-conexe

• Un arc din  $G$  nu aparține neapărat unei componente tare-conexe

## Egalitate, izomorfism.

Fie  $G_1, G_2$  două grafuri

$$\circ G_1 = (V_1, E_1)$$

$$\circ G_2 = (V_2, E_2)$$

Grafurile  $G_1$  și  $G_2$  sunt izomorfe ( $G_1 \sim G_2 \Leftrightarrow \exists$

$f: V_1 \rightarrow V_2$  bijecțivă cu  $uv \in E_1 \Leftrightarrow f(u)f(v) \in E_2$   
pentru orice  $u, v \in V_1$  ( $f$  conservă și adiacența și  
mediacența).

## Graf bipartit

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

$$V = V_1 \cup V_2$$

$$V_1 \cap V_2 = \emptyset$$

astfel încât orice muchie  $e \in E$  are o extremitate în  $V_1$

și cealaltă în  $V_2$ :  $|e \cap V_1| = |e \cap V_2| = 1$

există o colorare a vîrfurilor

Obs:  $G = (V, E)$  bipartit  $\Leftrightarrow$

cu două culori:  $c: V \rightarrow \{1, 2\}$

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

(bicolorare)

## Arbore

Arbore = conex + aciclic  
graf neorientat

## Leme

1. Orice arbore  $T$  cu  $n \geq 1$  are cel mult ~~cel mult~~ putin două vîrfuri terminale (de grad 1)

Fie  $P$  un lanț elementar maxim în  $T$

Extremităile lui  $P$  sunt vîrfuri terminale, altfel:  
- putem extinde lanțul cu o muchie  
sau

- se înclide un ciclu în  $T$

2. Fie  $T$  un arbore cu  $n \geq 1$  vîrfuri și  $v$  un vîrf terminal în  $T$ . Atunci  $T - v$  este arbore

3. ...

4. Fie  $G$  un graf neorientat conex și  $C$  un ciclu în  $G$ .  
Fie  $e \in E(C)$  o muchie din ciclul  $C$ , atunci  $G - e$  este tot un graf conex

## Arbore parțiali

◦ "Scheletul" grafului

- Transmiterea de mesaje în rețea astfel încât mesajul să ajungă o singură dată în fiecare vîrf
- Conectare fără redundanță + cost minim

## Proprietate

Orice graf neorientat conex conține un arbore parțial (un graf parțial care este arbore)

La arborei avem  $niv[v] =$  nivelul lui  $v =$  distanța de la rădăcina lar

Tată =  $x$  este tata al lui  $y$  dacă există mulțime de  
pașe  $x \rightarrow y$  și  $x$  se află în arbore pe un nivel  
cu 1 mai mic decât  $y$

Fiu =  $y$  este fiu al lui  $x \Leftrightarrow x$  este tatăl lui  $y$

Ascendent / următor =  $x$  este ascendent al lui  $y$  dacă  $x$   
apartine unicului lanț elementar de la  
 $y$  la rădăcină (echivalent, dacă există  
un lanț de la  $y$  la  $x$  care trece  
prin noduri situate pe niveluri din  
ce în ce mai mici).

Descendent / următor =  $y$  descental al lui  $x \Leftrightarrow x$  ascendent al  
sui  $y$

Frumos = nod frumos fiu

Arborii pot fi reprezentati printr-o lista de fiu sau  
printr-un vector tata

folosind vectorul tata putem determina lanturi de la orice  
nod  $x$  la rădăcină, urmând în arbore de la  $x$  la

rădăcină

lanț ( $x$ ):

cât timp  $x_1 = 0$  execută  
acțiunea  $x$   
 $x = \text{Tata}[x]$

lanț  $\checkmark$  reversed  $(x)$

dacă  $x_1 = 0$  atunci  
lanț  $(\text{Tata}[x])$   
acțiunea  $x$

## Algoritmul lui Kosaraju

Pasul 0. Construim graful transpus

Pasul 1. Parcurgem DFS graful  $G^T$  și introducem într-o coadă  $S$  fiecare vîrf la momentul în care este finalizat (pentru a obține o ordonare descreșătoare a vîrfurilor după timpul de finalizare)

coadă  $S$

$\text{DFS}(G, i)$

$\text{viz}[i] = 1$

pentru  $ij \in E(G)$ :

dacă  $\text{viz}[j] = 0$  atunci:

$\text{DFS}(j)$

$\text{push}(S, i)$  // este finalizat

pentru  $x \in V$  execută

dacă  $\text{viz}[x] = 0$  atunci

$\text{DFS}(G, x)$

Pasul 2. Parcurgem DFS în graful transpus considerând vîrfurile în ordinea în care sunt extrase din  $S$  (descreșătoare după timpul de finalizare de la Pasul 1).

marcam toate vîrfurile ca fiind nevizitate  
cât timp  $S$  este nevidă

$x = S.pop()$

dacă  $x$  este nevizitat atunci

$\text{DFS}(G^T, x)$

Afișarea componentelor conexe din  $x$

Complexitate:

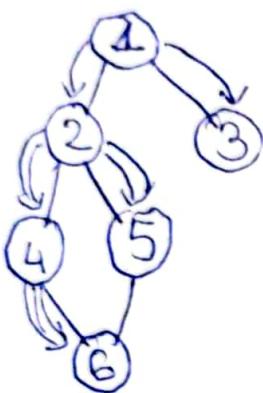
2 parcurgeri + construcția lui  $G^T \Rightarrow O(n+m)$

# BFS - Parcurgerea în lățime

! Se utilizează o coadă

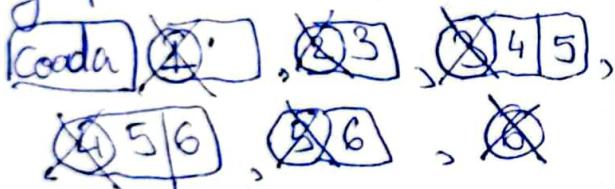
Principiul algoritmului:

- Iau un nod din coadă, bag vecinii lui și tot asa
- La final o să avem o listă formată din ordinea în care au trecut prin coadă



Parcurgerea BFS a

grafului este 1, 2, 3, 4, 5, 6



$d$  - vector de distanțe

$\pi$  - vector de tăți

$Q$  - coada în care păstrăm nodurile

Complexitate:  $O(n + m)$

$m = \text{numărul nodurilor}$   
 $n = \text{numărul muchiilor}$

Pseudocod Cormen:

def CL ( $G, s$ ):

for  $u \in V\text{-}\{s\}$ :

$\text{color}[u] = \text{"alb"}$

$d[u] = \text{float}(\text{inf})$

$\pi[u] = \text{"NIL"}$

$\text{color}[s] = \text{"grî"}$

$d[s] = 0$

$\pi[s] = \text{"NIL"}$

$Q = \{s\}$

while  $Q \neq \emptyset$ :

$u = \text{head}(Q)$

for  $v$  in  $\text{Vecini}[u]$ :

if  $\text{color}[v] == \text{"alb"}$ :

$\text{color}[v] = \text{"grî"}$

$d[v] = d[u] + 1$

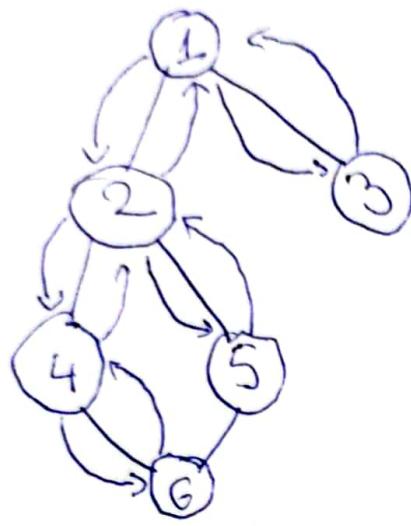
$\pi[v] = u$

$Q.\text{add}(v)$

$Q.\text{pop}()$

$\text{color}[u] = \text{"negru"}$

# DFS - Parcurgerea în adâncime



Parcurgerea DFS a grafului este 1, 2, 4, 6, 5, 3

$\pi$  - vectorul de tacti

$d$  - pasul la care am ajuns la un nod

$f$  - ultimul pas la care am trecut printr-un nod

Complexitate:  $O(n+m)$

$n =$  numărul nodurilor  
 $m =$  numărul muchiilor

Pseudocod Cormen:

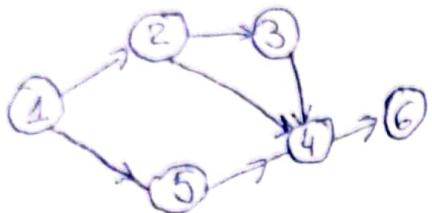
```

def CA(G):
    for u in Noduri:
        culoare[u] = "alb"
         $\pi$ [u] = "nil"
        timp = 0
    for u in Noduri:
        if culoare[u] == "alb":
            CA-vizita(u)

def CA-vizita(u):
    culoare[u] = "gri"
    d[u] = timp
    timp += 1
    for v in lista_adiac[u]:
        if culoare[v] == "alb":
             $\pi$ [v] = u
            CA-vizita(v)
    culoare[u] = "negru"
    f[u] = timp
    timp += 1
  
```

## Sortarea topologică

Def.: Ordinarea varfurilor astfel încât dacă ur e muchie  
atunci u se află înaintea lui și în ordine.



De exemplu, pentru acest graf  
sortare Topologică poate

1, 2, 3, 5, 4, 6

! Nu este unică !

Dacă graful este aciclic, atunci G are sigur o sortare topologică

## Algoritm grafuri acidice

- Este foarte similar cu BFS
  - prenăunte adăugarea nodurilor cu grad intern 0 și le adăugăm într-o colecție  
și luarea și scăderea gradielor interioare  
ale vecinilor (facem o cădere stergere  
a lor din graf)

## Pseudocod

~~while len(Vorlau) > 0:~~

~~for & in moduri~~

$$C = \{4\}$$

C. add (toate modurile an grad intern 0)

while len(C) > 0:

i = C.pop()

i il bagaglio in sortilegio

for i in range[1]:

$$d[j] = 1$$

if  $d[ij] = 0$ :

c. add(j)

Algoritm pentru grafuri ciclice:

- se foloseste de DFS

## Pseudocod

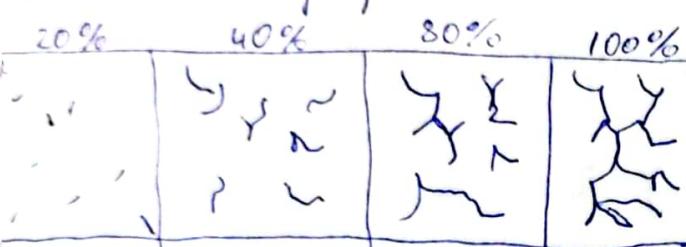
Se face DFS ce time minte și  
pasul la care a ~~ajuns~~ terminat un  
nod și de ordinează crescător pentru  
a ocoate sortarea topologică

## Arborei parțiali de cost minim - APCM

- Pentru determinarea APCM-ului folosim 2 algoritmi:

### KRUSKAL

- generează APCM-ul prin conectarea între nodurile cele mai apropiate



### Algoritm

La un pas, este selectată o muchie de cost minim din  $G$ , care nu formează cicluri cu muchiile deja adăugate.

### Operării necesare

Initializare ( $u$ ) - creare o componentă cu un singur vîrf,  $u$

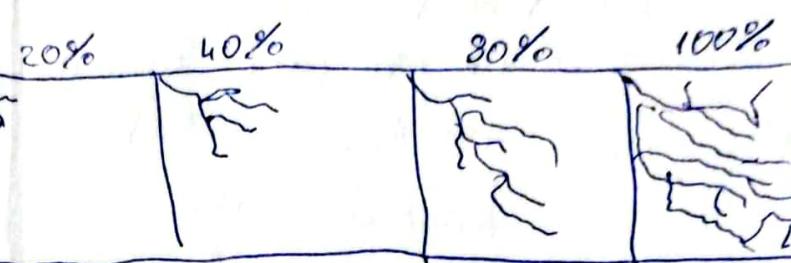
Reprez ( $u$ ) - returnează reprezentantul (culoarea) componentei care conține pe  $u$

Reuneste( $u, v$ ) - unește componenta care conține  $u$  cu cea care conține  $v$

O muchie  $uv$  unește două componente doar dacă  $\text{Reprez}(u) \neq \text{Reprez}(v)$

### PRIM

- generează APCM-ul prin conectarea primului nod la vecini prin cea mai scurtă latură și are grija să poată arborele fără cicluri



### Algoritm

Se pornește de la un vîrf, care formează arborele initial. La un pas, este selectată o muchie de cost minim, de la un vîrf deja adăugat în arbore, la un vîrf neadăugat.

$d[u]$  = costul minimul al unei muchii de la  $u$  la un vîrf deja adăugat

$tata[u]$  = acel vîrf din arbore pentru care se realizează minimul

$(u, tata[v])$  este muchia de cost minim de la  $u$  la un vîrf din arbore

$d[u] = w(u, tata[v])$

### Pseudocod:

Prim ( $G, w, s$ ):

for  $u$  in Noduri:

$d[u] = \text{float}(\text{inf})$

$tata[u] = 0$ .

$d[s] = 0$

$Q = \{Noduri\}$  #  $Q$  este coada

while  $\text{len}(Q) > 0$ :

$u = Q.pop(1)$

for  $v$  in Vecini:

if  $v$  in  $Q$  and  $w(u, v) < d[v]$ :

$d[v] = w(u, v)$

$tata[v] = u$

//actualizare  $Q$

## KRUSKAL

Pseudocod:

sortarea ( $E$ )

for ( $v=1; v \leq n; v++$ ):

    Initializare ( $V$ );

    numsel = 0

for  $u$  in Noduri :

    for  $v$  in vecini [ $u$ ] :

        if (Reprez( $u$ ) != Reprez( $v$ )):

            APC M. append ( $v$ )

        Reuneste ( $u, v$ )

        numsel += 1

        if (numsel ==  $n-1$ )

            break

Initializare:  $r[u] = u$

$O(1)$

Reprezentare:  $r[u]$

$O(1)$

Reuneste :

$r_1 = \text{Reprez}(u) / r_1 = r[u]$

$r_2 = \text{Reprez}(v) / r_2 = r[v]$

for ( $K=1; K \leq n; K++$ ):

    if  $r[K] == r_2$ :

$r[K] = r_1$

$O(m)$

Complexitate:  $O(m \log n + m^2)$

## Var 2 (Union - find)

def Initializare ( $u$ ):

    tata [ $u$ ] = h [ $u$ ] = 0

def Reprez ( $u$ ):

    while tata [ $u$ ] != 0:

$u = \text{tata}[u]$

return  $u$ ;

## Prim

Complexitate:  $O(m \log n)$

Corectitudine Kruskal + Prim

Fie  $A \subseteq E$  o multime de muchii

Notam  $A \subseteq \text{apcm} \Rightarrow \exists T$  un apcm astfel încât  $A \subseteq E(T)$

Atât algoritmul lui Kruskal, cât și cel al lui Prim, funcționează după următoarea schema:

$A = \emptyset$  (multimea muchiilor selec-

tate în arborele construit,

pentru  $i=1, n-1$  execută

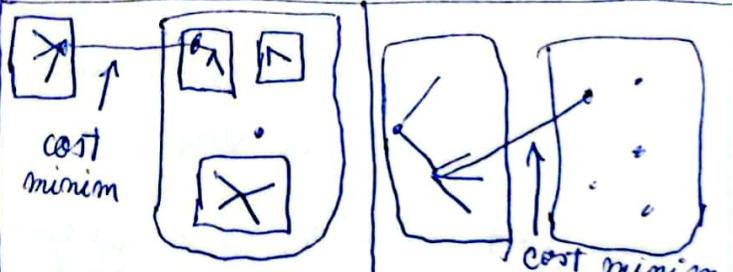
alege o muchie  $e$  a.t.

$A \cup \{e\} \subseteq \text{apcm}$

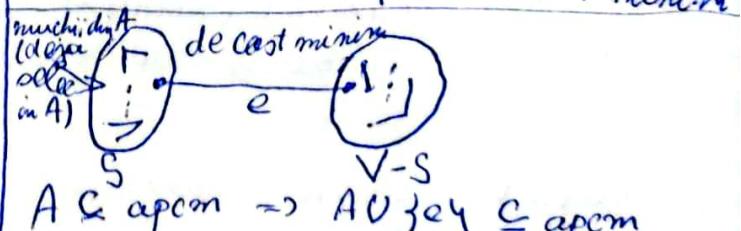
returnează  $T = (V, A)$

Vom demonstra că criteriu de alegeră a muchiei  $e$  la un pas a.i.  $A \subseteq \text{apcm} \Rightarrow A \cup \{e\} \subseteq \text{apcm}$

Kruskal



Prim



$A \subseteq \text{apcm} \Rightarrow A \cup \{e\} \subseteq \text{apcm}$

void Reuneste ( $u, v$ ):

int  $r_u, r_v$

$r_u = \text{Reprez}(u)$

$r_v = \text{Reprez}(v)$

if  $h[r_u] > h[r_v]$ :

    tata [ $r_u$ ] =  $r_v$

else:

    tata [ $r_u$ ] =  $r_v$

    if  $h[r_u] == h[r_v]$

$h[r_v] = h[r_v] + 1$



$O(m \log n)$

## Drumuri minime în grafuri ponderate

Fie

$G$  - un graf orientat ponderat

$P$  - un drum

$w(P)$  - costul / ponderea / lungimea drumului  $P$

! presupunem că  $G$  nu conține arci de pondere negativă

Fie  $s, v \in V$ ,  $s \neq v$

Distanța de la  $s$  la  $v$  =  $\delta_G(s, v) = \min \{w(P) \mid P$  drum de la  $s$  la  $v\}$

$$\delta_G(s, s) = 0$$

Convenție:  $\min \emptyset = \infty$

Un drum  $P$  de la  $s$  la  $v$  se numește drum minim de la  $s$  la  $v$  dacă  $w(P) = \delta_G(s, v)$

Obs. 1: Dacă  $P$  este un drum minim de la  $s$  la  $v$  și nu există arci de cost negativ, atunci  $P$  este drum elementar.

Obs. 2: Dacă  $P$  este un drum minim de la  $s$  la  $v$  și este un vîrf 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 vîrf  $s$  la celelalte vîrfuri  
(de cursă unică)

În cazul unui graf neponderat, problema se rezolvă folosind BFS din  $s$ .

Arborele al distanțelor făcă de  $s$  = subgraf care conservă distanțele de la  $s$  la celelalte vîrfuri

Un apel s este neapărat un arbore de distanțe minime

În ce ordine considerăm varfurile pentru a calcula distanțele față de  $s$ ? "Din aproape în aproape"  $\Rightarrow \delta(s, v) = \min_{w \in V \setminus \{s\}} \{\delta(s, w) + w(s, v)\}$

Ø Dacă există circuite - estimăm distanțele pe parcursul algoritmului

Und folosim algoritmi:

- Grafuri orientate cu circuite, dar cu ponderi pozitive  $\longrightarrow$  Dijkstra
- Grafuri orientate fără circuite, cu ponderi reale  $\longrightarrow$  DAGs (Directed Acyclic Graphs)
- Algoritm pentru grafuri orientate cu circuite și ponderi reale, care detectează existența circuitelor negative  $\longrightarrow$  Bellman-Ford

## DJKSTRA

Ipoteză: Prețumem că arcele au cost pozitiv (graful poate conține circuite)

Idee: La un pas, este alături de vîrf curenț (vizitat) vîrful u care este [estimat] a fi cel mai apropiat de s. Estimarea pentru u = cel mai scurt drum de la s la u determinat până la pasul curent.

Se descooperă noi drumi către vecinii lui  $\Rightarrow$  se actualizează distanțele estimate pentru vecin

Algoritmul lui Dijkstra este o generalizare a idei de parcurgere BF, iar dacă toate arcele au cost egal, atunci Dijkstra e similar la fel ca BFS-ul.

## Pseudocod

$d[u]$  - etichetă de distanță =  $\underset{s-u}{\text{cost minim}}$   
 $\text{Pred}[u]$  = predecesorul lui u  $\underset{\text{dat}}{\text{la un moment}}$   
 pe drumul de cost  
 minim descoperit

La un pas:

- este selectat un vîrf u (neselectat) care "pare" cel mai apropiat de s  $\Rightarrow$  are eticheta d minimă
- se actualizează etichetele  $d[v]$  ale vecinilor lui u
  - tehnică de relaxare a arculor dinu

Relaxarea unei arc ( $u, v$ ) = verificarea dacă există imbunătățire a lui  $d[v]$  dacă trecem prin u

def Relaxare ( $u, v$ ):

if  $d[u] + w(u, v) < d[v]$ :

$$d[v] = d[u] + w(u, v)$$

tata [ $v$ ] =  $u$

def Dijkstra ( $G, w, s$ ):

$$Q = \{V\}$$

for  $u$  in noduri:

$$d[u] = \text{float}(inf)$$

$$\text{tata}[u] = 0$$

$$d[s] = 0$$

while  $\text{len}(Q) > 0$ :

$$u = \cancel{Q}$$

$u$  = extrage varf cu eticheta  $d$  minimă din  $Q$ .

Azi  $v$  în vecini [ $u$ ]:

~~doar~~

if  $d[u] + w(u, v) < d[v]$ :

$$d[v] = d[u] + w(u, v)$$

$$\text{tata}[v] = u$$

Relaxarea

Complexitate:  $\mathcal{O}(n^2)$ , dacă stocăm în vector

$\mathcal{O}(m \log n)$ , dacă avem min-heap

Dramuri menite de cursă unică în grafuri aciclice

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

? Se va folosi sortarea topologică

### Pseudocod

- Idee: Considerăm varfurile în ordinea dată de sortarea topologică. Pentru fiecare varf  $u$ , relaxăm arcele  $uv$  către vecini săi.

$s$  = varful de start

for  $u$  in Noduri:

$d[u] = \text{float}(\inf)$

tata[u] = 0

$d[s] = 0$

SortTop = sortare\_topologică( $G, s$ )

for  $u$  in SortTop:

for  $v$  in vecini[u]:

if  $d[u] + w(u, v) < d[v]$ :

$d[v] = d[u] + w(u, v)$

tata[v] = u

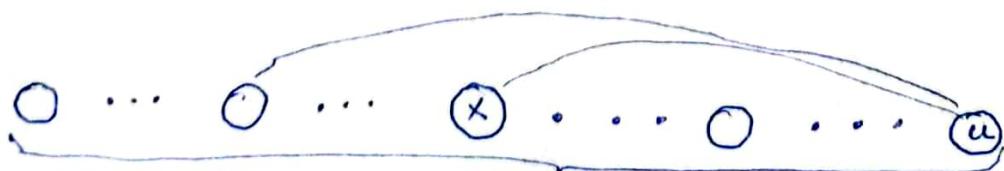
Complexitate:  $O(m+n)$

Corectitudine scurtă:

Algoritmul funcționează corect și dacă există arce cu cost negativ:  $\delta(s, u) = \min \{ \delta(s, x) + w(x, u) \mid x \in E \}$

când algoritmul ajunge la varful  $u$ , avem:

$d[u] = \min \{ d[x] + w(x, u) \mid x \in E \}$



SortTop

## Concluzii drumuri minime

graf neponderat

BF ( $s$ )

coada  $C = \emptyset$   
adăugă ( $s, C$ )

pentru fiecare  $u \in V$ :

$$d[u] = \infty;$$

$$\text{tata}[u] = \text{viz}[u] = 0;$$

$$\text{viz}[s] = 1; \\ d[s] = 0;$$

cât timp  $C \neq \emptyset$ :

$u \leftarrow \text{extrage}(C)$

pentru fiecare  $u \in E$ :

$$\text{dacă } \text{viz}[v] = 0:$$

$$d[v] = d[u] + 1$$

$$\text{tata}[v] = u$$

adăugă ( $v, C$ )

$$\text{viz}[v] = 1$$

graf ponderat

ponderi  $> 0$   
Dijkstra ( $s$ )

(min-heap)  $Q \leftarrow V$   
// se poate da doar vector viz;  $v \in Q \Rightarrow v \in V$  vizitata

pentru fiecare  $u \in V$ :

$$d[u] = \infty;$$

$$\text{tata}[u] = 0$$

$$d[s] = 0;$$

cât timp  $Q \neq \emptyset$ :

$u = \text{extrage}(Q)$  vîrf cu eticheta  $d$  minimă

pentru fiecare  $v \in E$ :

$$\text{dacă } d[u] + w(u,v) < d[v]:$$

$$d[v] = d[u] + w(u,v)$$

$$\text{tata}[v] = u$$

repara ( $v, Q$ )

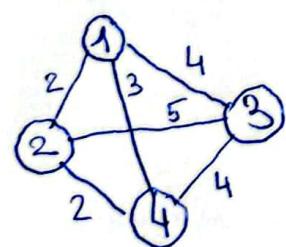
graf ponderat

lățea circuite

$O(n^2)$   
SortTop = sortare - topologică

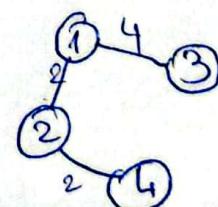
Drumuri minime din  $s \Rightarrow$  arbore de drumuri minime din  $s \neq$  APCM

Arbore al drumurilor minime  
față de 1



APCM

minimum-  
lățea circuit  
total (NIT  
din  $\sigma$ )



Diferența dintre  
distantei:

Prim și Dijkstra se afla la modificarea  
distanțăi:  
if  $d[u] + w(u,v) < d[v]$ :  
 $d[v] = d[u] + w(u,v)$   
 $\text{tata}[v] = u$   
repara ( $v, Q$ )

Prim

if  $v \in Q$  and  $w(u,v) < d[v]$ :  
 $d[v] = w(u,v)$   
 $\text{tata}[v] = u$   
repara ( $v, Q$ )

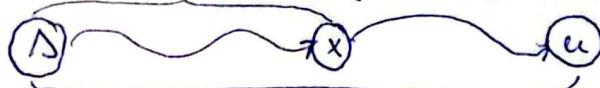
{  
ve  $Q$  e redundant)  
(la Dijkstra)}

## Bellman - Ford

- Arcele pot avea și cost negativ
- Graful nu conține căi de cost negativ  
(Dacă există  $\Rightarrow$  algoritmul le va detecta  $\Rightarrow$  nu are soluție)

Idee: La un pas, relaxăm toate arcele

De câte ori?  $s \xrightarrow{x} \text{drum minim cu } \leq K-1 \text{ arce}$



Dacă  $P$  este  $s$ -y drum  $s$ -u drum minim cu  $\leq K$  arce

cu  $\leq K$  arce  $\Rightarrow [s-x]$  este  $s$ -x drum minim

cu  $\leq K-1$  arce

Un drum minim are cel mult  $n-1$  arce  $\Rightarrow n-1$  etape

## Pseudocod:

for  $u \in \text{Nodeuri}$ :

$d[u] = \infty$ ;

Tata[u] = 0;

$d[s] = 0$ ;

for  $i$  in range( $n$ ):

for  $u, v$  in muchii:

if  $d[u] + w(u, v) < d[v]$ :

$d[v] = d[u] + w(u, v)$

Tata[v] = u

Complexitate:  $O(nm)$

! Se poate optimiza printr-o coadă care păstrează eticheta varfurilor actualizate

Drumuri minime între toate perechile de varfuri

### Floyd - Warshall

Problema: Se dă un graf orientat ponderat  $G = (V, E, w)$ . Pentru oricare 2 varfuri  $x$  și  $y$  ale 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  $G$ .

Soluția 1: Dijkstra din fiecare nod.

!!! Funcționarea doar pe ponderi pozitive

Complexitate =  $n \times$  Dijkstra

Soluția 2: Bellman Ford pentru fiecare nod.

Complexitate =  $n \times$  Bellman Ford  
 $= n^2 \times m = O(n^2 \cdot m)$

Soluția 3: Floyd - Warshall

Fie  $W = (w_{ij})_{i,j=1 \dots n}$ , matricea costurilor grafului  $G$ :

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

Vrem să calculăm matricea distanțelor  $D = (d_{ij})_{i,j=1 \dots n}$ ,

$$d_{ij} = S(i,j).$$

Observație:  $w_{ij}$  = costul minim al anui  $i-j$  drum, fără varfuri intermediare (cu cel mult un arc)

Idee: Pentru  $K = 1, 2, \dots, n$ , calculăm pentru oricare două varfuri  $i, j$ : costul minim al anui drum de la  $i$  la  $j$ , care

are ca varfuri intermediare doar varfuri din mulțimea  $\{1, 2, \dots, K\}$

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

$$D^{(K)} = (d_{ij}^{(K)})_{i,j=1 \dots n}$$

$d_{ij}^{(K)}$  = costul minim al anui drum de la  $i$  la  $j$ , care

are ~~tot~~ varfuri intermediare în  $\{1, 2, \dots, K\}$

Așteptat,  $D^{(0)} = W$ , iar  $D^{(n)} = L$

Idee:

Pentru a reține și un drum minim

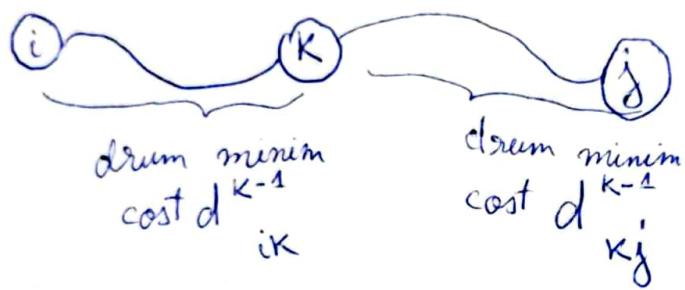
- matrice de predecesori  $P^k = (p_{ij}^k)_{i,j=1,\dots,n}$

-  $p_{ij}^k$  = predecesorul lui  $j$  pe drumul minim curent  
găsit de la  $i \xrightarrow{k} j$  la  $j$ , care are vîrfurile  
intermediare în  $\{1, 2, \dots, k\}$

Idea de calcul a matricei  $D^{(k)}$ :

Fie  $P$  un drum de cost minim de la  $i \xrightarrow{k} j$ ,  
cu vîrfurile intermediare în multimea  $\{1, 2, \dots, k\}$ .

- Dacă vîrful  $K$  este vîrf intermediar al lui  $P$



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

Observații:

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

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

De acela, în implementarea algoritmului, putem folosi o singură matrice:

2. Când se actualizează  $d_{ij}^k = d_{ik}^{k-1} + d_{kj}^{k-1}$ , trebuie actualizat și  $p_{ij}^k$ :  $p_{ij}^k = p_{kj}^{k-1}$

## Implementare Floyd-Warshall

- initializare

$$d[i][j] = w(i, j)$$

if  $i, j \in E$ :

$$\rho[i][j] = i$$

else:

$$\rho[i][j] = 0$$

## Pseudocod

for  $i$  in range( $n+1$ ):

    for  $j$  in range( $1, n+1$ ):

$$d[i][j] = w[i][j]$$

    if ( $w[i][j] == \text{float}(\text{inf})$ ):

$$\rho[i][j] = 0$$

    else:

$$\rho[i][j] = i$$

for  $k$  in range( $1, n+1$ ):

    for  $i$  in range( $1, n+1$ ):

        for  $j$  in range( $1, n+1$ ):

            if  $d[i][j] > d[i][k] + d[k][j]$ :

$$d[i][j] = d[i][k] + d[k][j]$$

$$\rho[i][j] = \rho[k][j]$$

complexitate

$$\mathcal{O}(n^3)$$

Definire: matricea  $d$  = matricea distanțelor minime  
matricea  $\rho$  = matricea ce afisează un drum  $i - j$

def drum (int  $i$ , int  $j$ ):

    if ( $i != j$ )

        drum( $i, \rho[i][j]$ );

        cout << j << "

        print(j, end = " ")

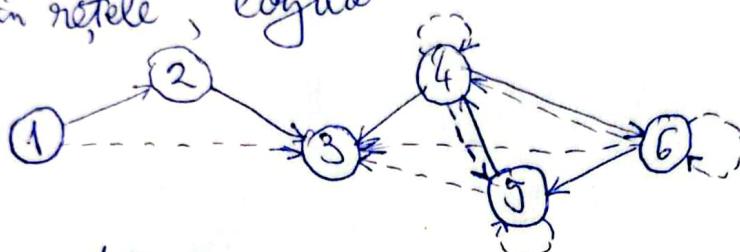
## Algoritmul Roy - Marshall

Aplicație: Închiderea transițivă a unui graf orientat  $G = (V, E)$  (!!! neponderat):  $G^* = (V, E^*)$ , unde  $E^* = \{(i, j) \mid$

există drum  
(de lungime  
minim 1)  
de la  $i$  la  
 $j$  în  $G\}$

Utilitate:

- grupări de obiecte aflate în relație (directă sau indirecță): optimizări în baze de date, analiză în rețele, logica



! Linia punctată reprezintă închiderea transițivă  $\Leftrightarrow$

$\Leftrightarrow$  calculăm matricea existenței drumurilor ~~matricea existenței drumurilor~~ (matricea existenței de adiacență a închiderii transițive)

$$D = (d_{ij})_{i,j=1 \dots n} \quad d_{ij} = \begin{cases} 1, & \text{dacă există drum nevid de la } i \text{ la } j \\ 0, & \text{altfel} \end{cases}$$

Pseudocod:

initial  $d$  = matricea de adiacență

for  $K$  in range ( $1, n+1$ ):

    for  $i$  in range ( $1, n+1$ ):

        for  $j$  in range ( $1, n+1$ ):

$$d[i][j] = d[i][j] \vee (d[i][K] \wedge d[K][j]) \quad \text{SAU} \quad \text{si}$$

Observație:

Dacă  $A$  este matricea de adiacență a unui graf și  $A^K = (a_{ij}^K)_{i,j=1 \dots n}$  puterea  $K$  a matricii ( $K < n$ ) atunci  $a_{ij}^K$  este numărul de drumuri distințe de lungime  $K$  de la  $i$  la  $j$  (! nu neapărat elementare!).

Consecință

$$D = A \vee A^2 \vee \dots \vee A^{m-1}$$

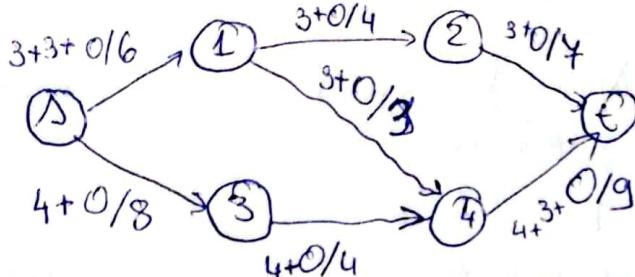
unde o valoare diferită de 0 se interpretează ca True

## Fluxuri maxime în rețele de transport

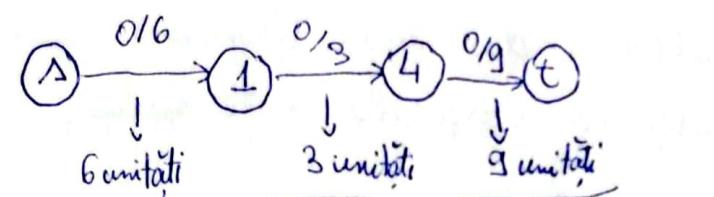
Problema: Avem o rețea în care:

- nodurile au limitări de capacitate
- nodurile = jumătăți

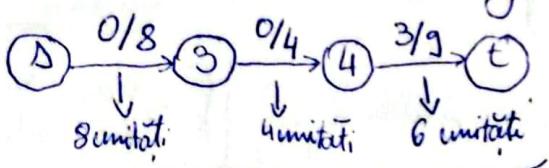
Care este cantitatea maximă care poate fi transmisă prin rețea, de la surse la destinații? (în unitatea de timp)



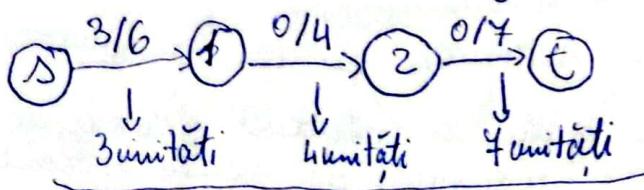
Determinăm drumuri de la  $A \rightarrow T$  prin care să trimitem marfă



3 unități de-a lungul întregului drum

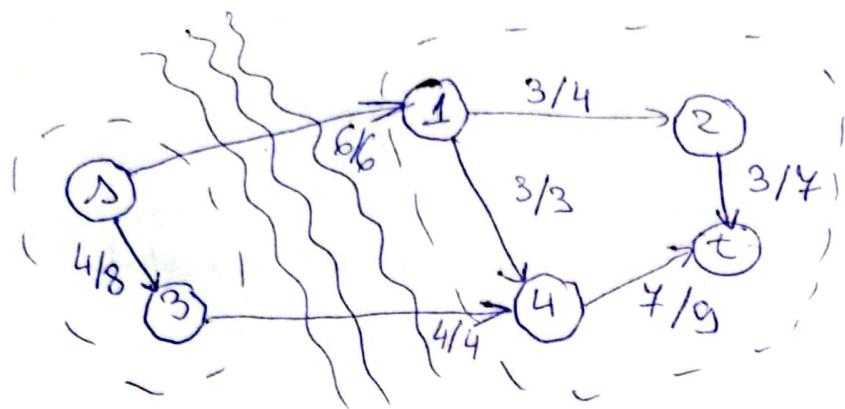


4 unități de-a lungul întregului drum



3 unități de-a lungul întregului drum

Nu mai există drumuri de la  $A \rightarrow T$  pe care mai putem trimite flux



Singurale arce ("poduri") care trece din regiunea lui lat nu mai pot fi folosite pentru a trimite flux (au flux-capacitate)

Retea 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

- $S$  = multimea surselor (intrărilor)

- $T$  = multimea destinațiilor (iesirilor)

- $I$  = multimea varfurilor intermedii

- $c: E \rightarrow \mathbb{N}$  funcția de capacitate (cantitatea maximă care poate fi transportată prin fiecare arc)

Ipoteze pentru retea  $N$ :

- $S = \{s\} \rightarrow$  o singură sursă

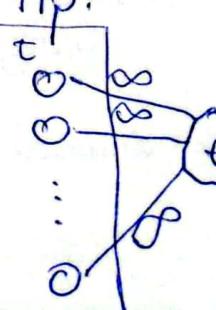
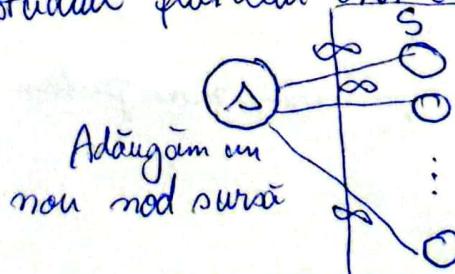
- $T = \{t\} \rightarrow$  o singură destinație

- $d^-(s) = 0 \rightarrow$  în sursă nu intră arce

- $d^+(t) = 0 \rightarrow$  din destinație nu ieș arce

- orice varf este accesibil din  $s$

Ipoteze restrictive - vom arăta că studiul fluxului într-o reteu cu mai multe surse și destinații se poate reduce la studiul fluxului într-o reteu de acest tip.



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

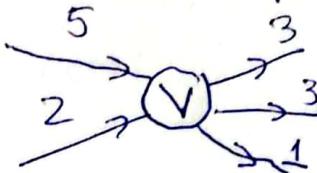
1. condiția de mărginire

$$0 \leq f(e) \leq c(e), \quad \forall e \in E(G)$$

2. condiția de conservare a fluxului

Pentru orice vîrf intermediar  $v \in I$ :  $\sum_{u \in E} f(uv) = \sum_{u \in E} f(vu)$

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



Notății:

$$-\bar{x}$$

$$-f^-(v), f^+(v)$$

$$-f(X, Y), X, Y \subseteq V$$

$$-f^+(X), X \subseteq V \text{ și } f^-(X)$$

În general, pentru orice funcție  $g: E \rightarrow \mathbb{N}$  vom folosi notății similare.

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

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

Condiția de conservare a fluxului devine  $f^-(v) = f^+(v)$

Pentru  $X, Y \subseteq V$  disjuncte:

$$f(X, Y) = \sum_{\substack{u \in E \\ u \in X, v \in Y}} f(uv) = \text{fluxul de la } X \text{ la } Y \\ (\text{pe arcele care ieș din } X \text{ către } Y)$$

Pentru  $X \subseteq V$  disjuncte:

$$f^+(X) = \sum_{\substack{u \in E \\ v \in X, v \notin X}} f(uv) = \text{fluxul care ieșe din } X \\ (\text{din vîrfurile din } X)$$

$$f^-(X) = \sum_{\substack{u \in E \\ v \in X, v \notin X}} f(vu) = \text{fluxul care intră în } X \\ (\text{în nodurile din } X)$$

$$f^+(x) = f(x, v-x) = f(x, \bar{x})$$

$$f^-(x) = f(v-x, x) = f(\bar{x}, x)$$

Valoarea fluxului  $f$  se definește ca fiind

$$\text{val}(f) = f^+(\Delta) = \sum_{S \subseteq E} f(S)$$

$$\text{val}(f) = f^+(\Delta) = f^-(\bar{\Delta})$$

### Problema fluxului maxim

Fie  $N$  o rețea.

Un flux  $f^*$  se numește flux maxim în  $N$  dacă:

$$\text{val}(f^*) = \max \{ \text{val}(f) \mid f \text{ este flux în } N \}$$

### Observație:

Orice rețea admite cel puțin un flux, spre exemplu fluxul vid  $f(e) = 0, \forall e \in E$

## Algoritmul Ford - Fulkerson

(de determinare a unui flux maxim + a unei traieturi minime)

Notiuni necesare descrierii și studiului algoritmului:

- s-t lant f-nesaturat
  - arc direct
  - arc invers
  - capacitate residuală arc, lant
- Operația de revizuire a fluxului de-a lungul unui s-t lant f-nesaturat

- Traietura în rețea
  - capacitatea unei traieturi

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

- Un s-t lant este o succesiune de varfuri distincte și arce din  $G$

$P = [s = v_0, e_1, v_1, \dots, v_{k-1}, e_k, v_k = t]$ , unde arcul  $e_i$  este fie  $v_{i-1}v_i$  fie  $v_iv_{i-1}$ . (Peste lant elementelor în graful neorientat asociat lui  $G$ )

- Dacă
  - $e_i = v_{i-1}v_i \in E(G)$ ,  $e_i$  se numește arc direct

(înainte) în  $P$

$$◦ e_i = v_iv_{i-1} \in E(G), e_i$$
 arc invers (înapoi) în  $P$

⚠ Dacă nu există confuzii, vom omite cercurile în scrierea lantului  $P$ .

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

Fie  $N$  rețea,  $f$  flux în  $N$ ,  $P$  un s-t lant.

Asociem fiecărui arc  $e$  din  $P$  o pondere, numită capacitate residuală în  $P$ .

capacitate residuală = cât mai puțin trimită printre-un

$$ip(e) = \begin{cases} c(e) - f(e), & \text{daca } e \text{ este arc direct in } P \\ f(e), & \text{daca } e \text{ este arc invers in } P \end{cases}$$

$ip(e)$  = cu cat mai poate fi modificat fluxul pe arcurile de-a lungul lantului  $P$

Capacitatea residuală a lantului  $P$  =  $i(P)$

= cu cat putem revizui maxim fluxul de-a lungul lui  $P$  =  $\min \{ ip(e) | e \in P \}$

$P$  se numeste:

- f - saturat dacă  $i(P) = 0$

- f - nesaturat dacă  $i(P) \neq 0$

Fie  $N$ -rețea,  $f$  flux in  $N$ ,  $P$  un s-t lant  $f$ -nesaturat.  
Fluxul revizuit de-a lungul lantului  $P$  se definește astfel  
fără  $f_P : E \rightarrow \mathbb{N}$ .

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

### Proprietățile fluxului revizuit

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

Fie  $P$  un s-t lant  $f$ -nesaturat in  $G$  și  $f_P$  fluxul revizuit de-a lungul lantului  $P$ .

Atunci:

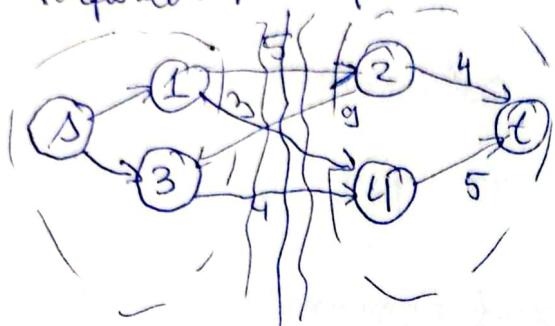
- $f_P$  este flux in  $G$

- $\text{val}(f_P) = \text{val}(f) + i(P) \geq \text{val}(f) + 1$

### Tăietură în retea

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

O tăietură  $K = (X, Y)$  în retea (bi)partiție  $(X, Y)$  a mulțimii varfurilor  $V$  astfel încât  $s \in X$  și  $t \in Y$ .



$$c(K) = 5 + 3 + 4 \\ = 12$$

Fie  $K = (X, Y)$  o tăietură

capacitatea tăieturii  $K = (X, Y)$  este  $c(K) = c(X, Y)$

$$= \sum_{\substack{x \in X, y \in Y \\ xy \in E}} c(xy) = \text{suma capacitatilor arcelor care lez din } X \text{ către } Y$$

Notă:

-  $E^+(K)$  = mulțimea arcelor de la  $X$  la  $Y$

=  $\{xy \in E \mid x \in X, y \in Y\}$  = arce directe ale lui  $K$

-  $E^-(K)$  = mulțimea arcelor de la  $Y$  la  $X$

=  $\{yx \in E \mid x \in X, y \in Y\}$  = arce inverse ale lui  $K$

$$\text{Atunci avem: } c(K) = c(E^+(K))$$

Fie  $N$  o retea,

O tăietură  $\tilde{K}$  se numește tăietură minimă în  $N$

dacă  $c(\tilde{K}) = \min \{c(K) \mid K \text{ este tăietură în } N\}$

Vom demonstra val  $f \leq c(K)$

- Dacă arem egalitate  $\Rightarrow f$  flux maxim,  $K$  tăietură minimă

s-t tăietură saturată residual  $\Rightarrow$  nu mai există  
s-t drum în graful residual  $\Rightarrow$  s-t flux maxim

### Pseudocod

Algoritm generic de determinare a unui flux maxim

Fie  $f$  un flux în  $N$  (de exemplu,  $f = 0$  fluxul vid:  $f(e) =$

Vid)

Cât timp există un lant f-nesaturat  $P$  în  $G$ :

- determinăm un astfel de lant  $P$

- rezarcă fluxul  $f$  de-a lungul lantului  $P$

Returnează  $f$

Pentru a determina și o s-t tăietură minimă, la finalul algoritmului considerăm:

$X$  = multimea vîrfurilor accesibile din s prin lanturi  $f$ -nesaturate

$$K = (X, V - X)$$

Complexitate:

- $O(mL)$ , unde  $L =$  capacitatea minimă a unei tăieturi  $\leq \sum_{e \in E} c(e)$

- $O(nmC)$ , unde  $C = \max\{c(e) | e \in E\}$

Cum determinăm un lant  $f$ -nesaturat?

- Spre exemplu, prin parcurgerea grafului, pornind din vîrful  $s$  și considerând doar arce cu capacitatea residuală pozitivă (în raport cu lanturile construite prin parcurgere, memorate cu vectorul  $tata$ )

"BFS"  $\Rightarrow$  determinăm s-t lanturi  $f$ -nesaturate de lungime minimă  $\Rightarrow$  Algoritmul EDMOND - KARP

= Ford-Fulkerson, în care pleacă un pas minim

Alte criterii de construcție lant  $\Rightarrow$  alți algoritmi

Complexitate  $O(mL)$ , unde  $L = \sum_{S \in E} c(S)$

Complexitate  $O(mC)$ , unde  $C = c(\{sy, V - sy\}) = C^+(S)$

## Algoritmul Edmonds - Karp

### Implementare

initializare - flux\_nul()

căt timp (construieste\_s-t\_lant - mesat\_BF() == true):  
revizuieste\_flux\_lant()

afiseaza\_flux()

Amintim: a determina un s-t lant mesurat folosind BF în  
G  $\Leftrightarrow$  a determina un s-t drum folosind BF în graful  
residual  $G_f$

### Varianta 1 de implementare

- Revizuirea fluxului folosind s-t lanturi din G (fără  
a folosi graful residual)

construieste\_s-t\_lant\_mesat\_BF():

(construieste un s-t lant mesurat, prin parcurgerea BF)

o sunt considerate în parcurgere doar arce pe care  
se poate modifica fluxul, adică având capacitatea  
residuală pozitivă

o returnează false dacă un astfel de lant nu  
există (și true dacă e posibil să-l construi)

revizuieste\_flux\_lant():

o fie  $P$  s-t lantul găsit în construieste\_s-t\_lant  
- mesat\_BF()

o calculăm  $i(P)$

pentru fiecare arc e al lantului  $P$

- ordem cu  $i(P)$  fluxul pe e dacă este arc direct
- scădem cu  $i(P)$  fluxul pe e dacă este arc invers

### Sugestii de implementare

Memoram lanturile (arborele BF), folosind vectorul tata

Convenție - pentru arcele inverse  $(i, j)$  tinem minte  
tată cu semn minus :  $tata[j] = -i$

construieste\_s-t\_lant\_mesat\_BF()

for  $v$  in Noduri :

    tata[v] = 0

    viz[v] = 0

coada C =  $\emptyset$

adaugă (s, C)

~~coada~~

while len(C) > 0 :

    i ← extrage(C)

    for  $j$  in vecinii[i] :

        if viz[j] == 0 and  $c(i,j) > f(j)$  :

            adaugă(j, C)

            viz[j] = 1

            tata[j] = i

            if  $j = t$  :

                return true

    for  $j$  in precedatori[i] :

        if viz[j] == 0 and  $f(j) > 0$  :

            adaugă(j, C)

            viz[j] = 1

            tata[j] = -i

            if  $j = t$  :

                return true

return false

arc direct

arc invers

Complexitate:

- Algoritm generic Ford-Fulkerson -  $O(mL)/O(mn^2)$
- Implementare Edmonds-Karp -  $O(nm^2)$

Schemă

initializarea\_flux\_nul()

cât timp (constr...\_BF()):

revizuirea\_flux\_lant()

afisează\_flux

! Avertisment: a determina un s-t lant nesaturat folosind

BF în  $G \Rightarrow$  a determina un s-t drum folosind  
graful rezidual  $G_f$

Varianta 2 de implementare

- Revizuirea fluxului folosind s-t drumuri din  $G_f$  (graf rezidual)

Schemă nouă

initializarea\_flux\_nul()

cât timp (constr...\_BF() == true):

revizuirea\_flux\_lant()

actualizarea  $G_f$

afisează → flux()

Pseudocod

actualizare  $G_f()$ :

for e in MuchiiFlux:

arc direct  $\rightarrow c_f(e) = c_f(e) - c_f^P$

if  $c_f(e) \leq 0$ :

elimină e din  $G_f$  // reînnoiește

arc indirect  $\rightarrow c_f^{(e^{-1})} = c_f(e^{-1}) + c_f^P$

if  $c_f(e^{-1}) > 0$  and  $e^{-1} \notin G_f$ :

adăuga  $e^{-1}$  la  $G_f$

## Cuplaj

Aplicatie: Sistem de reprezentanti distincti

Fie  $A$  - multime finita cu  $n$  elemente

$$x_1, x_2, \dots, x_m \subseteq A$$

Se numeste sistem de reprezentanti distincti pentru colectia de submultimi  $(x_1, x_2, \dots, x_m)$  un vector  $(r_1, r_2, \dots, r_m)$  cu proprietatile:

$$\circ r_i \in x_i, \forall i = 1, \dots, m$$

$$\circ r_i \neq r_j \Rightarrow \forall i, j = 1, \dots, m, i \neq j$$

Ex:  $A = \{1, 2, 3, 4\}$

$$x_1 = \{2, 3\} \Rightarrow r_1 = 2$$

$$x_2 = \{1, 3, 4\} \Rightarrow r_2 = 3$$

$$x_3 = \{1, 2, 4\} \Rightarrow r_3 = 4$$

Conditiile necesare si suficiente pentru existenta unui sistem de reprezentanti distincti ai unei colectii de submultimi din  $A$ ?

Modelam problema cu ajutorul unui graf bipartit:

o varf  $x_i$  - asociat submultimii  $x_i$   
 $\Rightarrow$  multimea  $X$  de varfuri

o varf  $a_j$  - asociat fiecarui element din  $A$   
 $\Rightarrow$  multimea  $Y$  de varfuri

o muchie de la  $x_i$  la  $a_j \Leftrightarrow a_j \in x_i$

Observatie: Există un sistem de reprezentanti pentru colectia de submultimi  $(x_1, x_2, \dots, x_m)$  ale lui  $A$   
 $\Leftrightarrow$  există un cuplaj al lui  $x$  în grafel asociat

### Teorema lui HALL:

Dacă pentru orice submultime  $S = \{x_{i_1}, x_{i_2}, \dots, x_{i_K}\}$

$\subseteq X$  avem  $|N(S)| \geq |S| = K$  (imaginarea lui  $S$ )

$$N(S) = x_{i_1} \cup x_{i_2} \cup \dots \cup x_{i_K}$$

Astfel, are loc următorul rezultat:

Teoremă - existența unei sisteme de reprezentanți distincti.

Fie  $A$  o multime finită și  $(X_1, X_2, \dots, X_m)$  o colectie de submultimi din  $A$ .

Colectia nu are un sistem de reprezentanți distincti  
 $\Leftrightarrow \exists K$  submultimi în colectie a căror reuniune are mai puțin de  $K$  elemente

Aplicatie : Flux maxim  $\rightarrow$  Cuplaj maxim în grafuri bipartite

### Cuplaj:

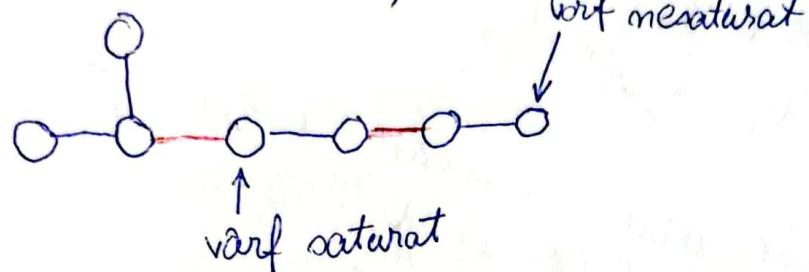
Repere istorice / Aplicații:

- Problema seratei
  - $n$  fete,  $n$  băieți
  - un băiat cunoaște exact  $K$  fete
  - o fată cunoaște exact  $K$  băieți
- Problema organizării de competiții
- Probleme de repartizare
  - lucrători - locuri de muncă
  - profesori - examene / conferințe
  - Problema orarului

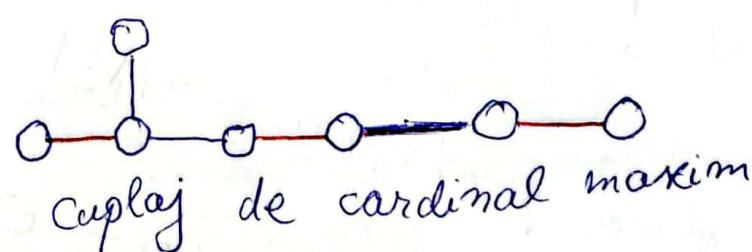
Fie  $G = (V, E)$  un graf si  $M \subseteq E$ .  $M$  se numeste cupaj daca orice două muchii din  $M$  sunt neadiacente.

$V(M) =$  multimea varfurilor  $M$ -saturate

$V(G) - V(M) =$  multimea varfurilor  $M$ -nesaturee



Un cupaj  $M^*$  se numeste cupaj de cardinal maxim (cupaj maxim), daca  $|M^*| \geq |M|$ ,  $\forall M \subseteq E$  cupaj.



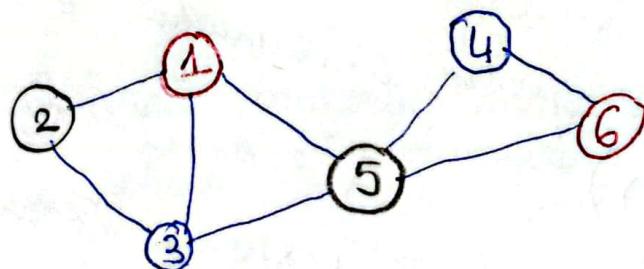
### Graful bipartite

Fie  $G = (V, E)$  graf neorientat:

$c: V \rightarrow \{1, 2, \dots, p\}$  se numeste  $p$ -colorare

a lui  $G$   
 $c: V \rightarrow \{1, 2, \dots, p\}$  cu  $c(x) \neq c(y)$ ,  $\forall xy \in E$  se numeste  $p$ -colorare proprie a lui  $G$

$G$  se numeste  $p$ -colorabil daca admite o  $p$ -colorare proprie



3-colorabil, dar nu si 2-colorabil

$G = (V, E)$  graf neorientat se numește bipartit  $\Leftrightarrow$  există o partitie a lui  $V$  în două submultimi  $V_1$  și  $V_2$  (bipartitie) cu:

- $V = V_1 \cup V_2$
- $V_1 \cap V_2 = \emptyset$

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

Notăm  $G = (V_1 \cup V_2, E)$

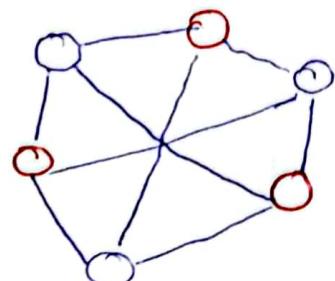
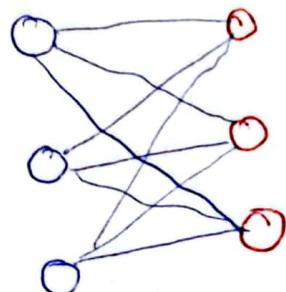
$G = (V, E)$  graf ~~neorientat~~  $\Leftrightarrow$  este bipartit și  $E = \{xy \mid x \in V_1\}$

$y \in V_2\}$ .

Notăm  $K_{p,q}$ , dacă  $p = |V_1|$  și  $q = |V_2|$

**Ex**

$K_{3,3}$



Observație:

$G = (V, E)$  bipartit  $\Leftrightarrow$  există o 2-colorare proprie a varfurilor (bicolorare):

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

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

Alte aplicatii: graf de conflicte (separăm noduri în 2 multimi care nu au muchie/conflict între ele), p-colorari, alocarea de rezurse

Proprietate: Un arbore este bipartit.

- Fixăm o grădăcină
- Colorăm alternativ

Teorema König - caracterizarea grafurilor bipartite

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

Aveam  $G$  este bipartit  $\Leftrightarrow$  toate ciclurile elementare din  $G$  sunt pare.

Demonstratie " $\Rightarrow$ ":

Evident, deoarece un ciclu impar nu poate fi colorat propriu cu 2 culori.

Demonstratie " $\Leftarrow$ ":

Presupunem  $G$  conex.

Colorăm propriu cu 2 culori un arbore parțial  $T$  al său.

Orice altă muchie  $uv$  din graf are extremitățile colorate diferit, deoarece formață un ciclu elementar cu lantul de la  $u$  la  $v$  din arbore, iar acest ciclu are lungime pară, deci  $u$  și  $v$  se află pe niveluri de paritate diferență în  $T$ .

Din Teorema König  $\Rightarrow$  Algoritm pentru a testa dacă un graf este bipartit

o Colorăm (propriu) cu 2 culori un arbore parțial al său, print-o parcursare (colorăm orice vecin j nevizitat al varfului curent i cu o culoare diferență de cea a lui  $i$ )

o Testăm dacă celelalte muchii - de la  $i$  la vecin  $j$  de la vizitati (colorati) au extremitățile și colorate diferență

## Algoritm

Flux maxim  $\rightarrow$  Cuplaj maxim în graferi bipartite

o Reducem problema determinării unui cuplaj maxim într-un graf bipartit  $G$  la determinarea fluxului maxim într-o rețea de transport asociată lui  $G$ .

o Construim rețeaua de transport  $N_G$  asociată lui

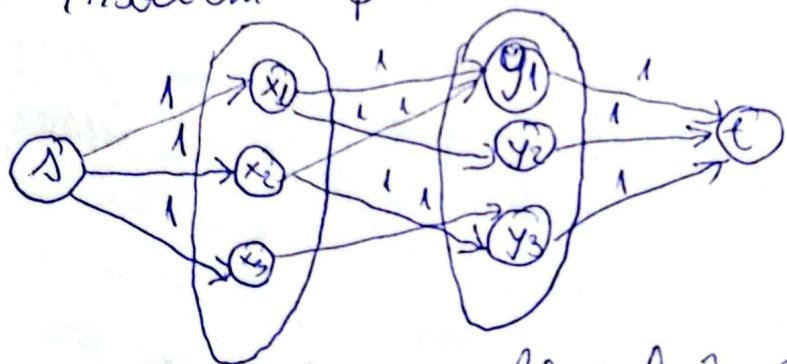
$G$  astfel:

- Adăugăm 2 noduri noi  $s$  și  $t$

- Adăugăm arce  $(s, x_i)$  și  $(y_j, t)$

- Transformăm muchiile  $x_i, y_j$  în arce (de la  $x$  la  $y$ )

- Asociem fiecărui arc capacitatea 1



Cuplaj  $M$  în  $G \Leftrightarrow$  flux  $f$  în rețea  
 $|M| = \text{val}(f)$

Proprietatea 1.

Fie  $G = (X \cup Y, E)$  un graf bipartit și  $M$  un <sup>cuplaj</sup>  $M$  în  $G$ . Atunci există un flux  $f$  în rețeaua de transport asociată  $N_G$  cu  $\text{val}(f) = |M|$ .

Justificare: Dăt un cuplaj  $M$  în  $G$ , se poate construi un flux  $f$  în  $N_G$  cu  $\text{val}(f) = |M|$ , astfel că muchiile din cuplaj sunt cu flux 1 și celelalte flux 0.

## Consecință

$f^*$  flux maxim în  $N \Rightarrow$  cupajul corespunzător  $M^*$  este  
cupaj maxim în  $G$

A determină un cupaj maxim într-un graf bipartit  
 $\hookrightarrow$  A determină un flux maxim în rețeaua asociată

## Algoritm

1. Construim  $N$  rețeaua de transport asociată
2. Determinăm  $f^*$  flux maxim în  $N$
3. Considerăm  $M = \{xy \mid f^*(xy) = 1, x \in X, y \in Y\}$

(pentru fiecare arc  $xy$  cu flux nul din  $N$ , care nu este incident în  $s$  sau în  $t$ , muchia  $xy$  corespunzătoare din  $G$  se adaugă la  $M$ )

4. return  $M$

Complexitate?  $O(mn)$

Aplicație - Construcția unui graf orientat din seventele de grade

Se dau seventele:

$$\cdot S_0^+ = \{d_1^+, \dots, d_n^+\}$$

$$\cdot S_0^- = \{d_1^-, \dots, d_n^-\}$$

$$\text{cu } d_1^+ + \dots + d_n^+ = d_1^- + \dots + d_n^-$$

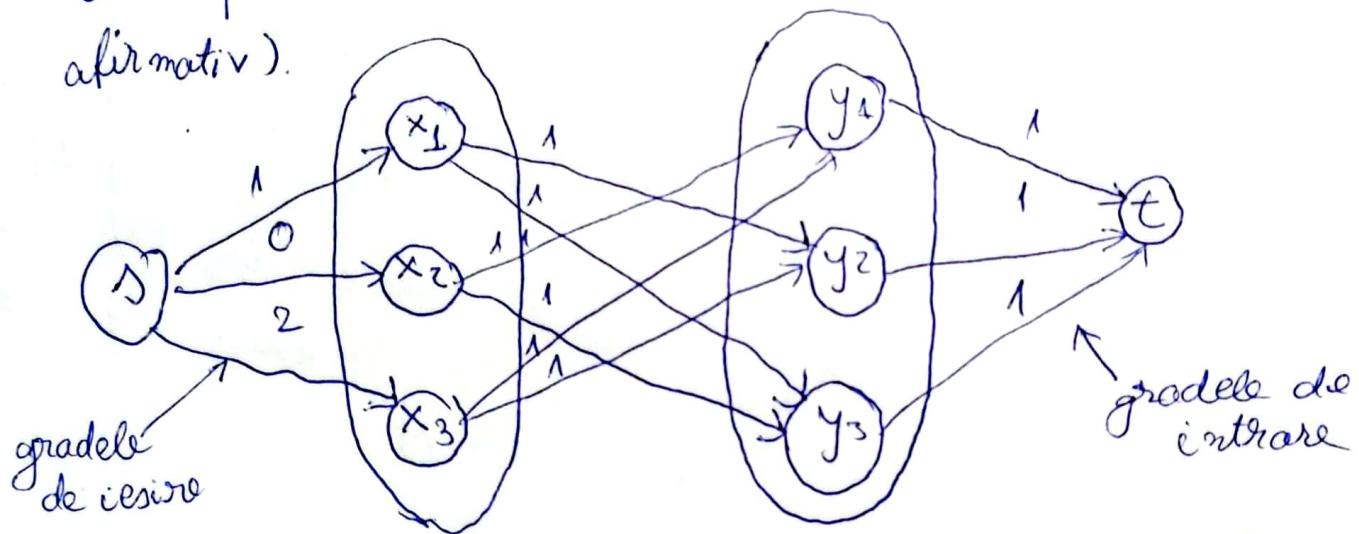
Să se construiască, dacă se poate, un graf orientat  $G$ , cu  $\Delta^+(G) = \Delta_0^+$  și  $\Delta^-(G) = \Delta_0^-$

Exemplu:

$$\Delta_0^+ = \{1, 0, 2\}$$

$$\Delta_0^- = \{4, 4, 1\}$$

Construim o rețea asociată. Elor două secvențe a.i. din fluxul maxim în rețea să putem deduce dacă  $G$  se poate construi + arele grafului  $G$  (în caz afirmativ).



$x_i$  nu se logă de  $y_j$  pentru că reprezintă același nod.

Proprietate:

Există graf cu secvențele date  $\Leftrightarrow$  în rețea asociată, fluxul maxim este:

$$\text{val}(f) = d_1^+ + \dots + d_n^+ = d_1^- + \dots + d_n^-$$

(săturatează toate arele care intră în  $t$  și ieș din  $s$ )

Tăieturile  $(\{s\}, V - \{s\})$ ,  $(V - \{t\}, \{t\})$  sunt minime.

- Algoritm de determinare a unui graf orientat
- $G$  cu  $s^+(G) = s_0^+$  și  $s^-(G) = s_0^-$
1. Construim  $N$  rețea de transport asociată
  2. Determinăm  $f^*$  flux maxim în  $N$
  3. Dacă  $\text{val}(f^*) < d_1^+ + \dots + d_n^+$  atunci  
Nu există  $G$ . STOP
  4.  $V(G) = \{1, \dots, n\}$   
 $E(G) = \{(i, j) \mid x_i, y_j \in N \text{ și } f^*(x_i, y_j) = 1\}$

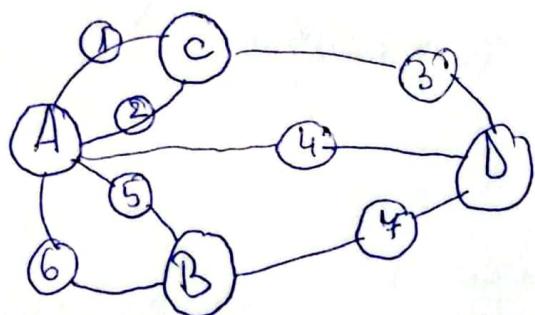
### Teorema lui Menger

Fie  $G$  graf orientat,  $s \rightarrow t$ , două verfurii distincte  
în  $G$ . Numărul minim de arce care trebuie eliminate  
pentru ca  $s \not\rightarrow t$  să nu mai fie concrezate  
printr-un drum (să fie separate) = numărul maxim  
de drumeuri arc-dizjuncte de la  $s$  la  $t$

## Grafuli euleriene

Istoric, Aplicatii

- Este posibil ca un om să facă o plimbare în care să treacă pe toate cele 7 poduri po singură dată? (Problema celor 7 poduri Königsberg)



ciclu eulerian = traseu închis care trece o singură dată prin toate muchiile

Graf eulerian = graf ce conține un ciclu eulerian

~~Problema lui POSTHUMVS~~

Lant eulerian = lant simplu  $P$  în  $G$  cu  $E(P) = E(G)$

Observație:

Fie  $P = [v_1, \dots, v_k]$

o Dacă  $v_1 \neq v_k$ , atunci varfurile interne din  $P$  au gradul în  $P$  par, iar extremitățile au gradul în  $P$  impar

o Dacă  $v_1 = v_k$ , atunci toate varfurile din  $P$  au gradul în  $P$  par

Lemă:

Fie  $G = (V, E)$  un graf neorientat, conex, cu toate varfurile de grad par și  $E \neq \emptyset$ . Atunci, pentru orice  $x \in V$ , există un ciclu  $C$  în  $G$  cu  $x \in V(C)$  (ciclul care conține  $x$ , nu neapărat eulerian, nici necupărat elementar).

Demonstratie - Algoritm de determinare a unui ciclu ce conține  $x$ :

$$i=1 ; v_1 = x$$

$$E(C) = \emptyset$$

Repetă

$$\text{selectarea } e_i = v_i v_{i+1} \in E(G) - E(C)$$

$$E(C) = E(C) \cup \{e_i\}$$

$$i = i + 1$$

Până când  $v_i = x$



$$|E(G)| < \infty, \text{ deci}$$

algoritmul se termină  
( $v_i$  ajunge egal cu  $x$ )

Dacă  $v_i \neq x$ ,

atunci  $d_G(v_i)$  este impar. Din ipoteză

$d_G(v_i)$  este par, deci

$d_{G-E(C)}(v_i)$  e impar deci

$d_{G-E(C)}(v_i) > 0$

$\Rightarrow$  muchia  $e_i$  există

### Teorema lui Euler

Fie  $G = (V, E)$  un (multi)graf neorientat, conex, cu  $E \neq \emptyset$ . Atunci  $G$  este eulerian  $\Leftrightarrow$  orice vîrf din  $G$  are grad par

## Algoritmul lui Hierholzer

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

- ∅ Bazat pe ideea demonstratiei teoremei lui Euler
- ∅ fuziune de cicluri (succesiv)

Pasul 0 - verificarea condițiilor (conex + vf izolate, grade par)

Pasul 1

- o alege  $v \in V$  arbitrar
- o construiește  $C$  un ciclu în  $G$  care începe cu  $v$  (cu algoritmul din Lemă)

cât timp  $|E(C)| < |E(G)|$  execută

- o selectează  $v \in V(C)$  cu  $d_{G-E(C)}(v) > 0$
- (în care sunt indicate muchii care nu aparțin ciclului)
- o construiește ciclul  $C'$  un ciclu în  $G - E(C)$  care începe cu  $v$
- o  $C = C' \cup C'$  și  $C' \cap C = v$

seria  $C$

Complexitate?  $O(m)$

## Possible implementări

Varianta 1 - stiva (DFS)

Idee: Muchiile folosite sunt marcate (nu reprezintă rute păstrate)

Varianta 2 - posibilă implementare recursivă  
euler (nod n)

cât timp  $d(v) > 0$

alege  $v \in V$  o muchie incidentă în  $v$   
sterge muchia  $v \in G$

euler ( $w$ )

$C = C + v$  // adăugăm  $v$  în cicle

Initial

$C = \emptyset$

euler (1)

// pornim construcția din  
// vîrful 1

Observație:

Potem alege muchiile incidente în  $v$ , de exemplu  
în ordinea dată de listele de adiacență și  
atunci bucața de cod ar trebui să fie:

pentru  $v \in V$  :

sterge muchia  $v \in E$  din  $G$

euler ( $w$ )

### Teorema lui Euler:

Fie  $G = (V, E)$  un graf neorientat, conex, cu  $E \neq \emptyset$ .  
 Atunci,  $G$  are un lant eulerian  $\Leftrightarrow G$  are cel mult  
două varfuri de grad impar

### Teorema - Descompunerea euleriană

Fie  $G = (V, E)$  un graf orientat, conex (= graful  
 neorientat asociat este conex), cu exact  $2K$  varfură  
 de grad impar ( $K > 0$ ).

Atunci există o  $K$ -descompunere euleriană a lui  $G$   
 și  $K$  este cel mai mic cu această proprietate.

### Teorema lui Euler

Fie  $G = (V, E)$  un graf orientat, conex (= graful  
 neorientat asociat este conex, cu  $E \neq \emptyset$ ).  
 Atunci  $G$  este eulerian  $\Leftrightarrow \forall v \in V, d_G^-(v) = d_G^+(v)$   
 grad intern = grad extern

~~SAU~~

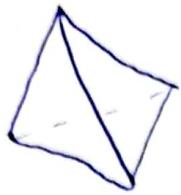
$G$  are un lant eulerian  $\Leftrightarrow$

$$\Leftrightarrow \exists x \in V \text{ cu } d_G^-(x) = d_G^+(x) - 1$$

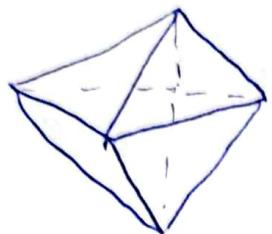
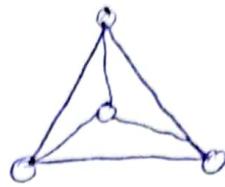
$$\exists y \in V \text{ cu } d_G^-(y) = d_G^+(y) + 1$$

$$\forall v \in V - \{x, y\} \quad d_G^-(v) = d_G^+(v)$$

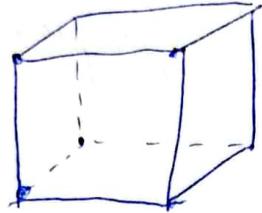
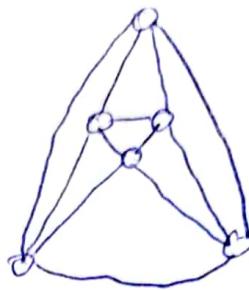
## Grauri planare



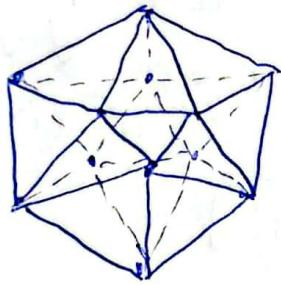
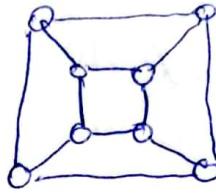
Tetraedru



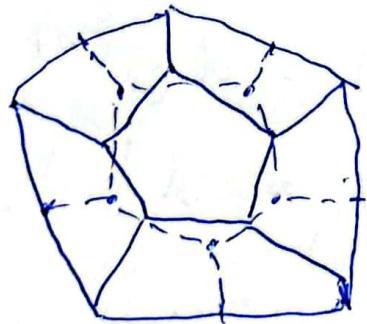
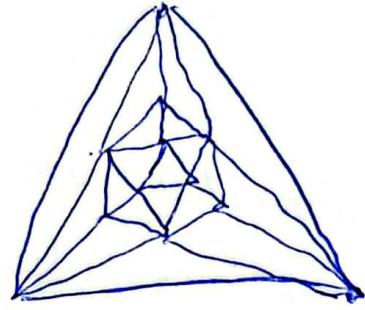
Octaedru



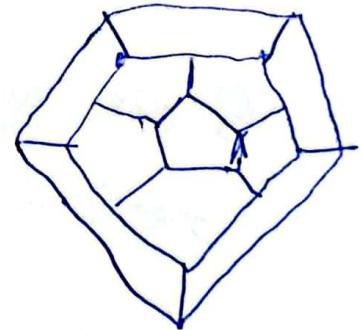
Cub



Icosaedru



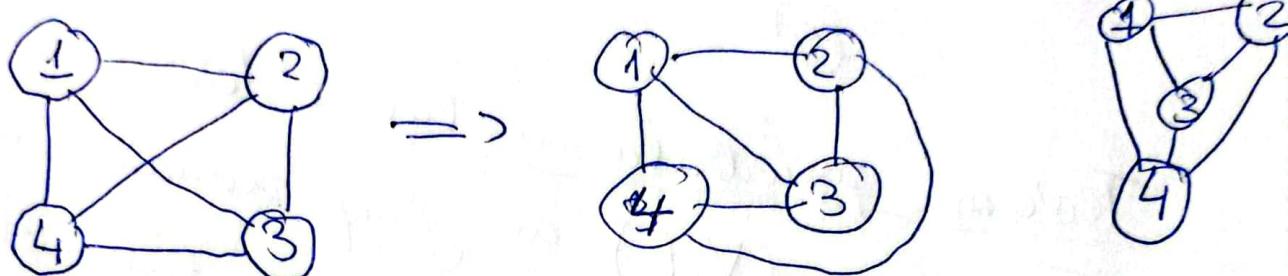
Dodecaedru



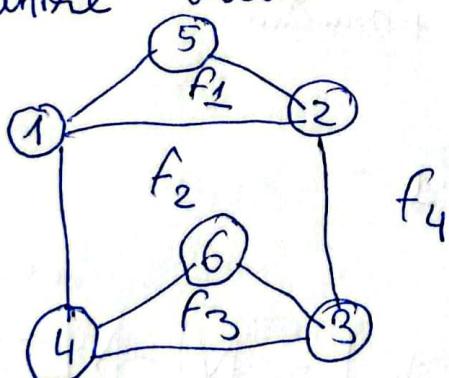
Forma în  
spatiu

Forma ca un  
graf planar

$G = (V, E)$  graf neorientat se numește planar  
 $\Leftrightarrow$  admite o reprezentare în plan, astfel încât  
 muchiile să corespund segmentelor de curbe continue  
 care nu se intersectează în interior unele pe altele  
 O astfel de reprezentare se numește harta  
 a lui  $G$ .



Fie  $G = (V, E)$  graf planar,  $M$  o harta a sa.  $M$  induce o împărțire a planului într-o multime  $F$  de părți convexe numite fete. Una dintre acestea este fata infinită (exterioră).



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

Pentru o fată  $f \in F$  definim :

$d_M(f) =$  gradul fetei

= numărul muchiilor lantului închis (frontierei) care delimită  
 ză  $f$  (câte muchii sunt percurse când traversăm frontiera)

### Observație

Hărți deosebite ale mediului: graf pot avea  
sevență gradelor diferență.

$$M = (V, E, F) \text{ harta}$$

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

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

Inducție

- Arbore parțial + muchie
- Într-un arbore
  - $|E| = |V| - 1$
  - $|F| = 1$
  - $|V| - |E| + |F| = |V| - |V| + 1 = 2 \rightarrow OK$

- Când adăugăm o muchie
  - $V$  rămâne constant
  - $E$  crește cu 1
  - $F$  crește cu 1
  - Relația rămâne validă

### Consecință

Orici hărță M a lui G are  $2 - |V| + |E|$  fețe

### Proprietăți

Fie  $G = (V, E)$  un graf planar convex, cu

$$n = |V| \geq 2 \text{ și } m = |E|$$

Atunci :

a)  $m \leq 3n - 6$

b)  $\exists x \in V$  cu  $d(x) \leq 5$

Demonstratie:

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

$$d_M(f) \geq 3$$

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

$$|V| - |E| + 2|E|/3 \leq 2$$

$$3|V| - |E| \leq 6 \Rightarrow a)$$

### Consecință

$K_{3,3}$  și  $K_5$  nu este un graf planar

### Teorema celor 6 culori

Orici graf planar convex este 6-colorabil.

Algoritm de colorare a unui graf planar cu 6 culori

colorare ( $G$ ):

dacă  $|V(G)| \leq 6$ :

    colorarea distinge vîrfurile distinct în fel  
    altfel

    alege  $x$  cu  $d(x) \leq 5$

    colorare ( $G - x$ )

    colorarea  $x$  cu o culoare din  $\{1, \dots, 6\}$   
    diferită de culorile vecinilor

### Sugestie de implementare

Determinarea iterativă a ordinii în care  
sunt colorate vîrfurile (similar BFS, sortare  
topologică)

### Teorema celor 5 culori

Orice graf planar conex este 5-colorabil

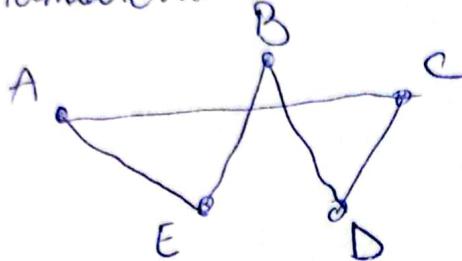
### Teorema celor 4 culori

Orice graf planar conex este 4-colorabil

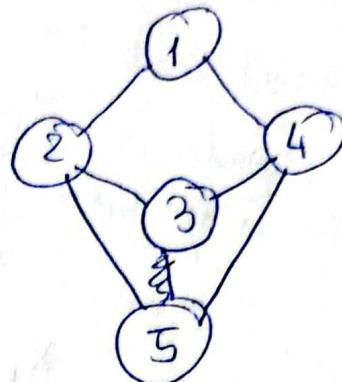
## Grafului Hamiltonian

Un ciclu hamiltonian este un ciclu care conține toate nodurile grafului.

Un graf este hamiltonian dacă admite un ciclu hamiltonian.



Ciclu ham.: A C D B E A



Zicea  
Nu avem

Proprietate (condiție necesară):

- Grafului hamiltoniene sunt biconexe (nu au noduri articice!)

Opusul nu este valabil !

- Nu toate grafulile biconexe sunt hamiltoniene

## Grafului euleriene

Condiție necesară și suficientă?

- DA, fiecare nod trebuie să aibă grad par (și graful să fie conex, exceptând nodurile izolate)

## Grafului hamiltonien

Condiții suficiente:

- Teorema lui Dirac
- Teorema lui Ore
- Teorema lui Chvatal și Erdos

- Teorema lui Goodman și Hedetniemi
- Teorema lui Duffus, Gould și Jacobson, 1981  
(similară cu Goodman)

## Teorema lui Dirac

Fie  $G$  un graf cu ordinul  $n \geq 3$

Dacă  $\delta(G) \geq n/2$ , atunci  $G$  este hamiltonian.

### Demonstratie

- Alegem un nod  $x_1$  la întâmplare (nu contează ce nod, pentru că, dacă există un ciclu hamiltonian, el poate începe din orice nod)

- Creem un lant cât mai lung, plecând din  $x_1$

obținând  $x_1 \dots x_K$

o Dacă  $K = n \rightarrow$  am obținut un lant ham.

- Dacă avem lantul  $x_1 \dots x_K$  și toti vecinii lui  $x_K$  au fost deja vizitati, putem extinde astfel:

$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ \text{click} & 1 & 2 & 8 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$

## Teorema lui Ore

Fie  $G$  un graf cu ordinul  $n \geq 3$

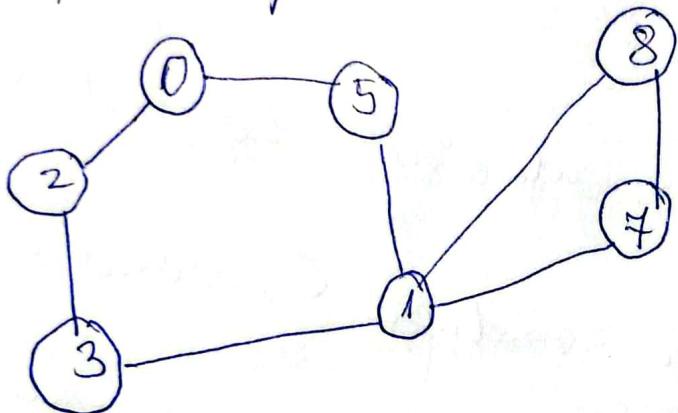
Dacă avem, pentru oricare pereche de noduri neadiacente  $\deg(x) + \deg(y) \geq n$ , atunci graful este hamiltonian.



Teorema lui Dirac este un caz particular al teoremei lui Ore, pentru că, dacă orice nod are gradul  $\geq n/2$ , atunci orice pereche de noduri are suma gradelor  $\geq n$ .

Conecțivitatea  $K(G)$  unui graf  $G$  este mărimea minimă a unei căi eturi a lui  $G$ . Cum o calculăm? Flux maxim = călătorească minimă

O multime de noduri a unui graf  $G$  este independentă dacă nu conține noduri adiacente. Numărul de independență  $\alpha(G)$  al unui graf  $G$  este mărimea cea mai mare posibilă a unei multimi independente a lui  $G$ .



3 {una din soluții este  $\{0, 3, 8\}$ }

Teorema lui Chvatal și Erdos

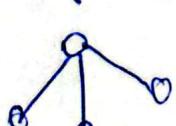
Teoremă:

Este  $G$  un graf conectat cu ordinul  $n \geq 3$ , conejivitatea  $K(G)$  și numărul de independență  $\alpha(G)$ .

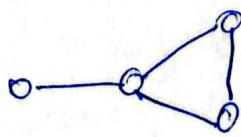
Dacă  $K(G) \geq \alpha(G)$ , atunci  $G$  este hamiltonian.

Teorema lui Goodman și Hedetniemi

Dacă  $G$  este un graf 2-conectat și liber de  $\{K_{1,3}, Z_1\}$ , atunci  $G$  este hamiltonian.



$K_{4,3}$



$Z_1$

Cum găsim un ciclu hamiltonian?

Nu este cunoscut un algoritm polynomial pentru a rezolva acestă problemă.

Varianta 1

- Facem toate permutările și le verificăm pe fiecare în parte, dacă este o soluție validă.

Complexitate?  $O(n! \cdot n)$

Varianta 2

- Folosim un algoritm exponential mai eficient:

- Vom considera matricea  $C$ , având următoarea semnificație:

$C[i][j] =$  costul minim al unui lant care începe în nodul  $i$ , se termină în nodul  $j$  și conține toate nodurile identificate cu  $L$  în configurația binară a lui  $j$  exact o singură dată

Ex : slide 1299

Complexitate?  $O(2^n \cdot n)$

## Distanță de editare

Se dau două siruri de caractere  $s_1$  și  $s_2$ :

**Distanță de editare** - numărul minim de operații (inserări, modificări, stergeri de caractere etc) necesar pentru a transforma sirul  $s_1$  în sirul  $s_2$ .

### Distanță de editare Levenshtein

Sunt permise operații de inserare, modificare și stergere.

Exemplu: de la carte la antet

care  $\xrightarrow{\text{sterge}} \text{are} \xrightarrow{\text{adă}} \text{ane} \xrightarrow{\text{inseram}} \text{ante} \xrightarrow{\text{inseram}} \text{antet}$

stergem c

adăm m

inserăm t

- La fiecare repatriere a unui caracter cu cel din destinație, avem 3 operații posibile.
- Dacă analizăm, pe rând, fiecare variantă  $\Rightarrow$  backtracking  $\Rightarrow$  inefficient

Soluție? Programare dinamică

### Principiu de optimitate:

Considerăm o transformare cu număr minim de operații:

$$x_1 x_2 \dots x_m$$

$$y_1 y_2 \dots y_m$$

Evidentiem ultima operatie

$$\boxed{x_m = y_m}$$

- problema se reduce la a transforma  $x_1 \dots x_{m-1}$  la

$$y_1 \dots y_{m-1}$$

$$\text{a)} \boxed{(x_1 \dots x_{m-1} \Rightarrow y_1 \dots y_{m-1}) + \text{pastream } x_m}$$

- o  $\boxed{x_m \text{ a fost sters}}$  - problema se reduce la a transforma  $x_1 \dots x_{m-1}$  în  $y_1 \dots y_m$  (după care stergem  $x_m$ )
  - b)  $\boxed{(x_1 \dots x_{m-1} \Rightarrow y_1 \dots y_m) + \text{stergem } x_m}$
- o  $\boxed{x_m \text{ a fost modificat în } y_m}$  - problema se reduce la a transforma  $x_1 \dots x_{m-1}$  în  $y_1 \dots y_{m-1} y_m$ 
  - c)  $\boxed{(x_1 \dots x_{m-1}) + \text{modificăm } x_m \leftrightarrow y_m}$
- o  $\boxed{\text{a fost inserat } y_m}$  - problema se reduce la a transforma  $x_1 \dots x_n$  în  $y_1 \dots y_{m-1}$ 
  - d)  $\boxed{(x_1 \dots x_n) + \text{inserăm } y_m}$

Subprobleme :

 $c[i][j] = \text{numărul minim de inserare, stergere, modificare pentru a transforma } x_1 \dots x_i \text{ în } y_1 \dots y_j$ 

Relații de recurență: a), b), c) și d) (literele sunt odată)

 $c[i][j] = \begin{cases} c[i-1][j-1], & \text{dacă } x_i = y_j \\ 1 + \min \begin{cases} c[i-1][j], & \text{stergere } x_i \\ c[i-1][j-1], & x_i \leftarrow y_j \\ c[i][j-1], & \text{inserăm } y_j \end{cases} \end{cases}$

Soluție:  $c[n][m]$

Ce valori din c stim direct:  
 $c[0][0] = 0$  (ambide curiose suntvide)

Pentru  $i=0$  sau  $j=0$  (unul din curiose estevid)

pentru  $i=0$  sau  $j=0$  (unul din curiose estevid)  
 $0 \times_1 \dots \times_{i-1} \Rightarrow$  secvență vidă prin i stergeri successive

$\circ$  secvență vidă  $\Rightarrow y_1 \dots y_j$  prin j inserări successive

$$c[0][0] = 0$$

$$c[i][0] = 1 + c[i-1][0] = i \text{ pt } i=[1, n]$$

$$c[0][j] = 1 + c[0][j-1] = j \text{ pt } j=[1, n]$$

Ordine de calcul a matricei:  
 $i = [0, n] ; j = [0, m]$