Suport seminar algoritmica grafurilor
II. Algoritm pentru drumuri de cost minim de la un vârf la toate vârfurile

Pentru o problemă de drum de cost minim, se dă un graf un graf orientat și ponderat $G = (V, E)$ cu funcția de pondere $w : E \to \mathbb{R}$. Ponderea $w(p)$ drumului $p = \langle v_0, v_1, ..., v_k \rangle$ este suma ponderilor arcelor ce formează drumul:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

Se definește costul drumului minim $\delta(u, v)$ de la u la v:

$$\delta(u, v) = \begin{cases} min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{dacă există un drum de la } u \text{ la } v, \\ \infty & \text{altfel.} \end{cases}$$

Astfel, un drum de cost minim de la vârful u la vârful v în graful $G = (V, E)$ este definit ca orice drum p cu ponderea $w(p) = \delta(u, v)$.

2.1 Cost minim în grafuri neorientate

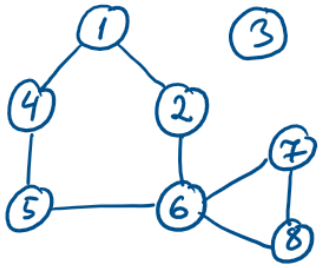
Un algoritm care determină drumul de cost minim pentru grafuri neorientate este algoritmul lui Moore.

```
MOORE(G, u)
1.  l(u) := 0
2.  for toate vârfurile v ∈ V(G), v ≠ u do
3.      l(v) := ∞
4.  Q = ∅
5.  u → Q
6.  while Q ≠ ∅ do
7.      Q → x
8.      for toți vecinii y ∈ N(x) do
9.          if l(y) = ∞ then
10.             p(y) := x
11.             l(y) := l(x) + 1
12.             y → Q
13. return l, p
```

notații:

- u - nod sursă;
- $l(v)$ - lungimea drumului până la vârful v;
- $p(v)$ - părintele vârfului v;
- Q - structură de date de tip coadă.

Problema 1. Pentru graful din figura alăturată să se aplice algoritmul lui Moore și să se determine valorile l și p pentru fiecare vârf dacă vârful de pornire este 1.



Soluție

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

2.2 Cost minim pentru grafuri orientate

Algortmul Bellman-Ford Rezolvă problema drumului de cost minim pentru cazul general (ponderile pot lua valori negative). Pentru un graf $G = (V, E)$ orientat și ponderat cu vârful sursă s și funcția de pondere $W : E \to \mathbb{R}$, algoritmul Bellman-Ford întoarce o valoare booleană care arată dacă există un circuit de pondere negativă accesibil din s. Dacă există un astfel de circuit de pondere negativ algoritmul indică faptul că nu există soluție.

Pentru a reprezenta drumul de cost minim (pentru a cunoaște vârfurile care compun drumul de cost minim) se reține pentru fiecare vârf $v \in V$ din gragul $G = (V, E)$ **predecesorul** acestuia, notat cu $v.\pi$, predecesorul poate fi un alt vârf din graf sau **NIL**.

Pentru fiecare vârf $v \in V$ din graful $G = (V, E)$ se mai reține atributul $v.d$ care reprezintă limita superioară a ponderii drumului de la vârful s la vârful v. Atributul $v.d$ se mai numește **estimarea drumului de cost minim**.

Bellman_Ford(G, w, s)
1: INITIALIZARE_S(G,s)
2: **for** $i = 1$ **la** $|V| - 1$ **do**
3: **for** fiecare arc $(u, v) \in E$ **do**
4: RELAX(u, v, w)
5: **for** fiecare arc $(u, v) \in E$ **do**
6: **if** $v.d > u.d + w(u, v)$ **then**
7: **return** FALSE
8: **return** TRUE

Inițializarea atributelor se face astfel:

INITIALIZARE_S(G, s)
1: **for** $v \in V$ **do**
2: $v.d = \infty$
3: $v.\pi = NIL$
4: $s.d = 0$

După inițializare: $v.\pi = NIL\ \forall v \in V$, $s.d = 0$ și $v.d = \infty\ \forall v \in V - \{s\}$. Cu ajutorul procedurii de relaxare se determină/testează dacă se poate îmbunătăți drumul de cost minim de la s la v dacă drumul ar trece prin nodul u, se actualizează $v.d$ și $v.\pi$ dacă drumul se îmbunătățește.

RELAX(u, v, w)
1: **if** $v.d > u.d + w(u, v)$ **then**
2: $\quad v.d = u.d + w(u, v)$
3: $\quad v.\pi = u$

Figura de mai jos prezintă două exemple de relaxare a unui arc. Deasupra fiecărui vârf se reprezintă estimarea drumului de cost minim ($v.d$).

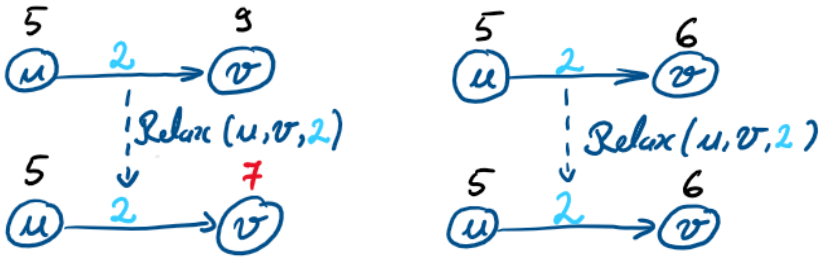
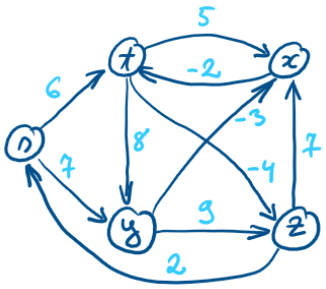


Figura 1: Pentru figura (a), deoarece $v.d > u.d + w(u, v)$, valoarea lui $v.d$ scade (drumul are un cost mai mic dacă trece prin vârful u). Pentru figura (b) $v.d \leq u.d + w(u, v)$ şi valoarea $v.d$ rămâne neschimbată.



Problema 2. Pentru graful din figura alăturată să se aplice algoritmul lui Bellman-Ford. Vârful sursă este vârful s.

Rezolvare Vârful sursă este vârful s. Atributul d este reprezentat deasupra fiecărui vârf (culoarea roșie indică schimbare în bucla respectivă). Arcele colorate în roșu arată predecesorul unui vârf, astfel dacă arcul (u, v) are culoarea roșie atunci $v.\pi = u$. Pentru acest exemplu se presupune că ordinea de parcurgere a arcelor din graf (liniile 3-4 din algoritm) este: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y). Figura 2 prezintă execuția algoritmului Bellman-Ford.

Tabela 1: Valoarea atributelor d şi π la finalul execuţiei algoritmului pentru graful dat la problema 2.

$v \in V$	s	t	x	y	z
$v.d$	0	2	4	7	-2
$v.\pi$	NIL	x	y	s	t

Algoritmul Dijkstra Algoritmul lui Dijkstra rezolvă problema drumului de cost minim pentru un graf $G = (V, E)$ orientat și ponderat pentru care toate ponderile nu sunt negative. Pentru fiecare arc $(u, v) \in E$ ponderile $w(u, v) \geq 0$.

Algoritmul menține un set S de vârfuri pentru care s-a determinat ponderea drumului minim de la vârful sursă s. Algoritmul alege vârful $u \in V - S$ pentru care estimarea costului drumului

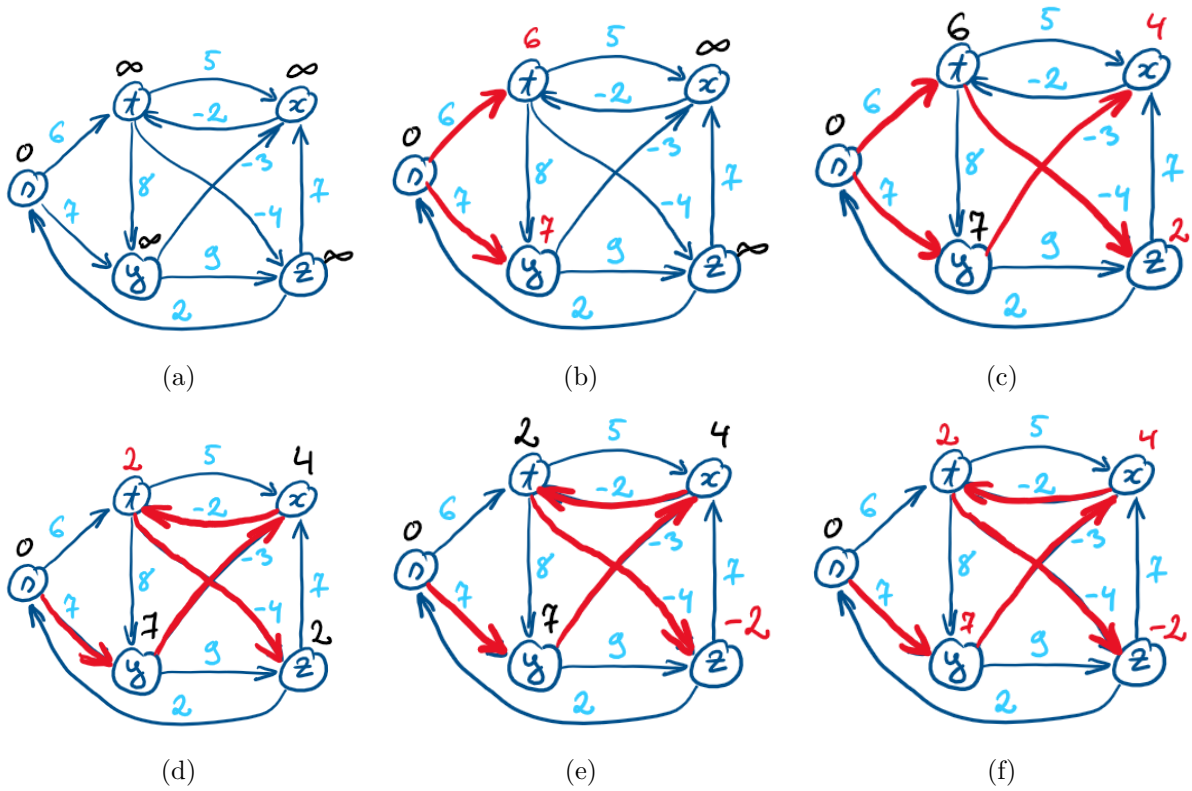
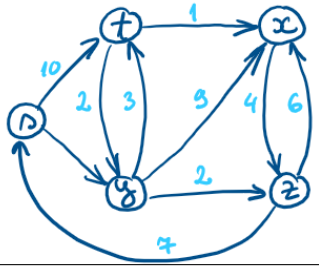


Figura 2: Execuţia algoritmului Bellman-Ford. Figura (a) reprezinta graful dat cu valoarea atributului d pentru fiecare vârf după iniţializare. Figurile (b)-(e) prezintă atributele d şi π pentru fiecare vârf după fiecare iteraţie a buclei for din algoritm (linia 2 algoritm Bellman-Ford). Figura (f) prezintă graful după terminarea execuţiei algoritmului. Pentru acest exemplu algoritmul întoarce *TRUE*.

este minimă ($\min_{u \in V-S} u.d$), adaugă u în S și relaxează toate arcele care pleacă din u. Pentru implementare se folosește o coadă minimă de priorități în care elementele sunt ordonate după atributul d.

Dijkstra(G, w, s)
1: INITIALIZARE_S(G,s)
2: $S = \varnothing$
3: $Q = V$
4: **while** $Q \neq \varnothing$ **do**
5: $\quad u =$ EXTRACT_MIN(Q)
6: $\quad S = S \cup \{u\}$
7: $\quad$ **for** $v \in G.Adj[u]$ **do**
8: $\quad\quad$ RELAX(u,v,w)

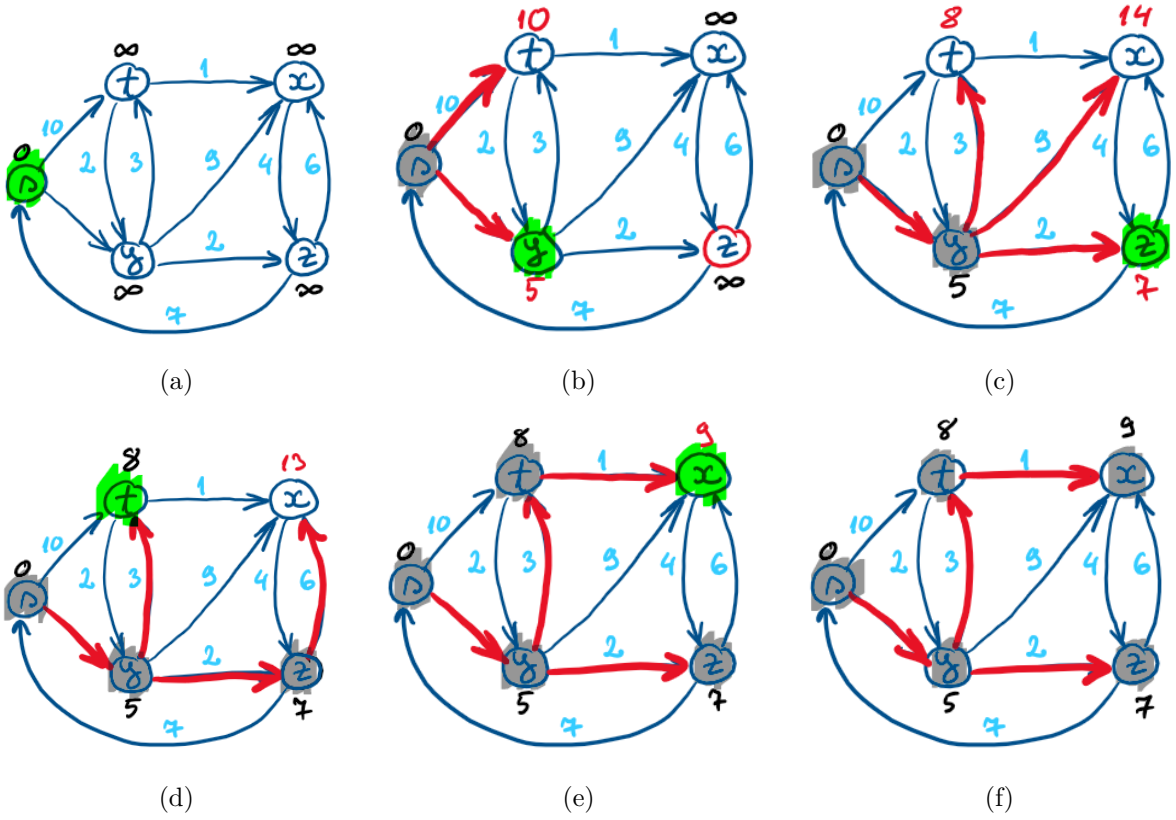
Problema 3. Pentru graful alăturat să se ruleze algoritmul lui Dijkstra și să se găsească drumul de cost minim începând din vârful s.



Rezolvare Vârful sursă este vârful s. Atributul d este reprezentat deasupra fiecărui vârf (culoarea roșie indică schimbare în bucla respectivă). Arcele colorate în roșu arată predecesorul unui vârf, astfel dacă arcul (u, v) are culoarea roșie atunci $v.\pi = u$. Culoarea verde indică vârful extras din Q, un vârf colorat în gri aparține setului S. Figura 3 prezintă execuția agoritmului Dijkstra.

Tabela 2: Valoarea atributelor d şi π la finalul execuţiei algoritmului pentru graful dat la problema 3.

$v \in V$	s	t	x	y	z
$v.d$	0	8	9	5	7
$v.\pi$	NIL	y	t	s	y



(a) (b) (c)

(d) (e) (f)

Figura 3: Execuţia algoritmului Dijkstra. Vârful sursă este vârful s, vârfurile cu fundal gri sunt în setul S iar vârfurile pe un fundal alb sunt în $Q = V - S$. Figura (a) reprezinta graful dat înainte de prima iteraţie a buclei while (liniile 4-8) cu valoarea atributului d pentru fiecare vârf după iniţializare. Vârful care are un fundal verde este cel cu valoarea minimă din Q şi este vârful u din linia 5 a algoritmului. Figurile (b)-(f) prezintă atributele d şi π pentru fiecare vârf după fiecare iteraţie a buclei while din algoritm, vârful care are un fundal verde este cel cu atributul d minim în fiecare iteraţie (vârful u din linia 5 a algoritmului).

Algoritmul Bellman Kalaba Rezolvă problema drumului de cost minim de la toate vârfurile la un vârf dat pentru un graf $G = (V, E)$ simplu orientat sau neorientat. Este un algoritm matricial, se folosește de matricea de adiacență a grafului. Se alege un vârf (o coloană din matricea de adiacență) și se determină distanțele de la toate vârfurile la acest vârf. Coloana aleasă (vârful ales) este notată

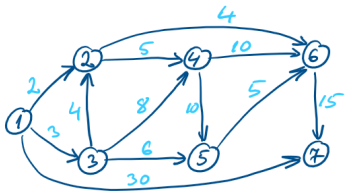
cu $V^{(1)} = (V_i^{(1)})_{i=\overline{1,n}}$. Dacă matricea de adiacența a grafului este $A = (a_{ij})_{i,j=\overline{1,n}}$, unde $a_{ij} = d_{ij}^{(0)}$. Elementele matricii de adiacența au valoarea:

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

Se determină vectorii pentru $k = 1, 2, ..., n$:

$$V_i^{(k)} = \min_{j=\overline{1,n}} \{a_{ij} + V_j^{(k-1)}\}, \text{ pentru } i = 1, 2, ..., n$$

până când $V^{(t)} = V^{(t-1)}$ pentru un t.



Problema 4. Pentru graful alăturat să se determine folosind algoritmul lui Bellman-Kalaba dostul minim pentru drumurile care duc în vârful 7.

Soluție Matricea de adiacență asociată grafului este

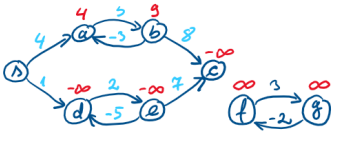
$$A = \begin{pmatrix} 0 & 2 & 3 & \infty & \infty & \infty & 30 \\ \infty & 0 & \infty & 5 & \infty & 4 & \infty \\ \infty & 4 & 0 & 8 & 6 & \infty & \infty \\ \infty & \infty & \infty & 0 & 10 & 10 & \infty \\ \infty & \infty & \infty & \infty & 0 & 5 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 15 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

iar vectorii distanță sunt

$$V^{(1)} = \begin{pmatrix} 30 \\ \infty \\ \infty \\ \infty \\ \infty \\ 15 \\ 0 \end{pmatrix}, V^{(2)} = \begin{pmatrix} 30 \\ 19 \\ \infty \\ 25 \\ 20 \\ 15 \\ 0 \end{pmatrix}, V^{(3)} = \begin{pmatrix} 21 \\ 19 \\ 23 \\ 25 \\ 20 \\ 15 \\ 0 \end{pmatrix}, V^{(4)} = \begin{pmatrix} 21 \\ 19 \\ 23 \\ 25 \\ 20 \\ 15 \\ 0 \end{pmatrix}.$$

Problema 5 Pentru algoritmul lui Dijkstra, dacă se schimbă linia 4 cu următoarea linie
while $|Q| > 1$
schimbarea face ca bucla while să se execute de $|V| - 1$ ori în loc de $|V|$ ori. Mai este acest algoritm corect (găsește drumul de cost minim de la vârful s la fiecare vârf din graf accesibil din s)?

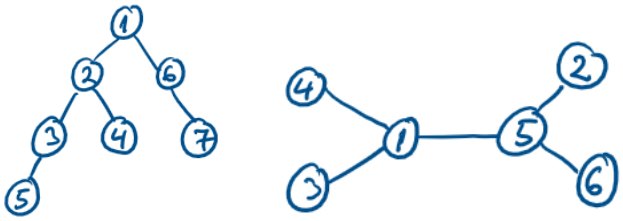
Problema 6. Modificați algoritmul lui Bellman-Ford astfel încât $v.d = -\infty$ pentru toate vârfurile v ce aparțin unui circuit de pondere negativă de pe un drum de la s la v. Graful alăturat conține un circuit negativ accesibil din vârful sursa (sursa este vârful s).



Suport seminar algoritmica grafurilor
III. Arbori

Probleme:

1. Să se demonstreze că molecula de parafină ($C_{31}H_{64}$) este un arbore.
2. Să se codifice Prüfer următorii arbori, vârful 1 este vârful rădăcină



(a) (b)

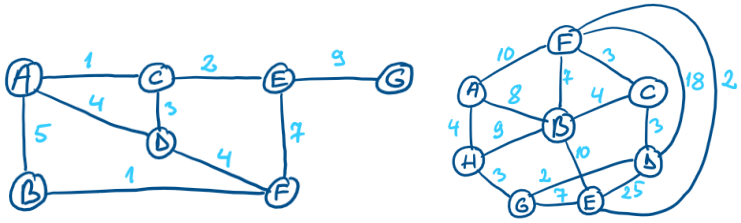
3. Să se decodifice secvențele Prüfer obținute la problema 2.
4. Ce valoare are x pentru următoarea secvență Prüfer dacă toate vârfurile din arbore au grad impar?

   secvența Prüfer: $1, 1, 5, x, 6, 6, 8$

5. Cum arată arborele pentru o secvență Prüfer dacă toate elementele secvenței au aceeași valoare?
6. Fie alfabetul format din caracterele $C = \{a, b, c, d, e, f\}$ și frecvențele de apariție pentru caracterele din C din tabelul de mai jos. Să se codifice folosind algoritmul de compresie Huffman secvența *aaaaabbbbccccddef*.

	a	b	c	d	e	f
frecv.	45	13	12	16	9	5

7. Folosind algoritmul lui Prim și Kruskal să se găsească arborele minim de acoperire pentru grafurile din figura 2.
8. Câți arbori minimi de acoperire există pentru graful din figura 3?



(a) (b)

Figura 2

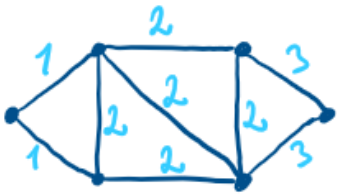


Figura 3

9. Fie un graf complet K_{1000} ponderat cu vârfurile etichetate. În graf exista un K_3 cu ponderea muchiilor 1, restul muchiilor au ponderea 2. Câți arbori minimi de acoperire sunt pentru graful dat?

10. (Proprietăți ale arborilor) Fie $G = (V, E)$ un graf neorientat, să se demonstreze că următoarele afirmații sunt echivalente:
    a. G este un arbore;
    b. oricare două vârfuri din G sunt conectate de un lanț simplu;
    c. G este conex, dacă se șterge o muchie din E graful rezultat va conține două componente conexe;
    d. G este conex și $|E| = V - 1$;
    e. G este aciclic și $|E| = V - 1$;
    f. G este aciclic dar dacă se adaugă o muchie în E graful rezultat conține un ciclu.

Soluții

Problema 2 Pentru graful din figura (a) de la cerința 2 dacă se aplică algoritmul de codare Prüfer pe graful din figura (a) secvența obținută este:

$$2, 3, 2, 1, 6, 1.$$

CODARE_PRUFER(F)
1. $K = \varnothing$
2. **while** T conține și alte vârfuri decât rădăcina **do**
3. fie v frunza minimă din T
4. $K \leftarrow$ predecesor(v)

5. $T = T \setminus \{v\}$
6. **return** K

Problema 3

DECODARE_PRUFER(K, n)
1. $T = \varnothing$
2. **for** $i = 1, 2, \ldots, n-1$ **do**
3. x primul element din K
4. y cel mai mic număr natural care nu se găsește în K
5. $(x, y) \in E(T)$, x părintele lui y în T
6. șterg x din K, adaugă y în coada secvenței K
7. **return** T

Pentru graful din figura (a) de la cerința 2 reconstrucția arborelui din secvența Prüfer $2, 3, 2, 1, 6, 1$ este prezentată în figura 4.

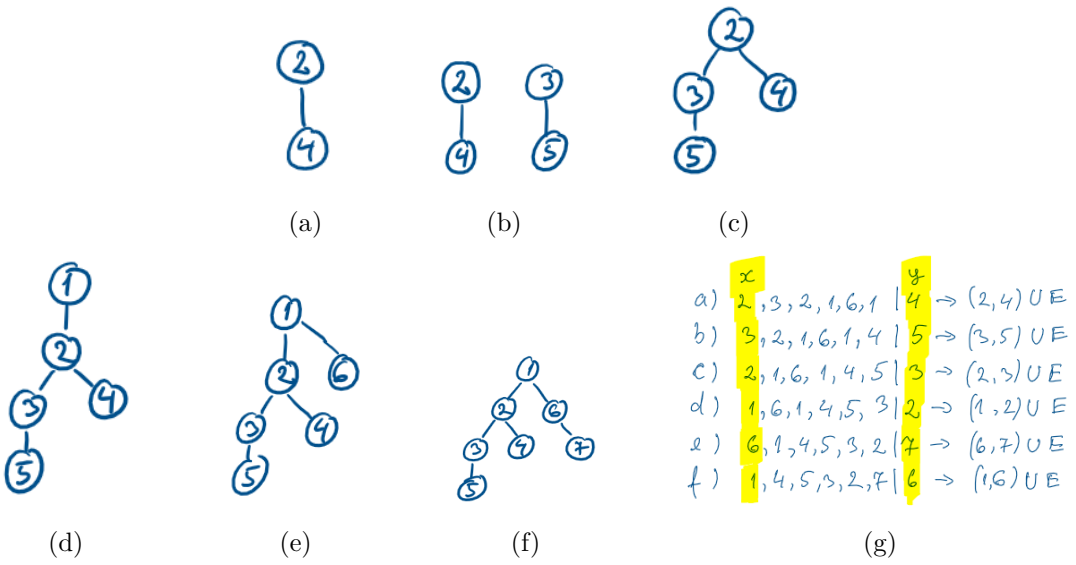


Figura 4: Reconstrucţia arborelui - decodificarea secvenţei Prüfer obţinută pentru arborele de la problema (2a). Figura (4g) prezintă paşii urmaţi pentru a reconstrui arborele, pentru fiecare pas (a)-(f) este prezentat vârful care se adaugă în arbore (vârful minim y) şi muchia (x, y).

Problema 6 Codul Huffman este utilizat pentru compresia datelor, în funcție de caracteristica datelor se poate obține o compresie între 20% și 90%. Se consideră datele ca fiind un set de caractere. Algoritmul folosește un tabel de frecvență (frecvența de apariție a fiecărui caracter din mesajul ce se vrea codat) pentru a construi reprezentarea binară optimă pentru fiecare caracter. La sfârșit fiecare caracter este reprezentat de un șir binar. Numărul de biți necesari pentru a reprezenta fiecare caracter depinde de frecvența de apariție a caracterului respectiv (se vrea ca un

caracter cu frecvență mare de apariție să fie codat cu un număr cât mai mic de biți, reducând astfel lungimea totală a mesajului inițial).

Pentru a putea decodifica mesajul la destinație fiecare cuvânt de cod nu trebuie să fie un prefix pentru alt cuvânt de cod.

Pentru algoritmul prezentat mai jos, C este un set de caractere ce se vrea a fi codificate Huffman. Fiecare caracter are asociat atributul fr care memorează frecvența de apariție a caracterului respectiv. Algoritmul construiește un arbore binar: pornește de la un set de $|C|$ frunze și face $|C|-1$ pași în care "unește" frunzele într-un arbore binar. Algoritmul folosește o coadă minimă de priorități Q pentru a identifică frunzele cu frecvența minimă de apariție. În fiecare pas de "unire", două frunze cu frecvența minimă de apariție din Q se unesc într-un nou vârf care are frecvența de apariție suma frecvențelor de apariție a frunzelor sale.

HUFFMAN(C)

1: $n = |C|$
2: $Q = C$
3: **for** $1 \leq i \leq n-1$ **do**
4: alocă un nou vârf z
5: $z.stang = x$ =EXTRACT_MIN(Q)
6: $z.drept = y$ =EXTRACT_MIN(Q)
7: $z.fr = x.fr + y.fr$
8: INSERT(Q, z)
9: **return** EXTRACT_MIN(Q)

Figura 5 prezentă pașii urmați de algoritm pentru a codifica setul de caractere C. Fiecare pas prezintă coada de priorități cu elementele ordonate crescător după atributul fr. Se poate observa că în fiecare pas arborii cu părintele de frecvență minimă de apariție sunt uniți într-un singur arbore. Frunzele arborilor sunt prezentate în dreptunghiuri care conțin caracterul și frecvența sa de apariție. Vârfurile prezentate în cerc țin suma frecvenței de apariție a frunzelor. Muchia care leagă un părinte de frunza stângă are ponderea 0 iar muchia care leagă un părinte de frunza dreaptă are ponderea 1. Codul obținut pentru fiecare caracter este secvența ce reprezintă ponderile muchiilor dacă se parcurge arborele de la rădăcină până la frunza ce se vrea codificată.

Codul obținut pentru fiecare caracter din C este

$$a = 0 \qquad c = 100 \qquad e = 1101$$
$$b = 101 \qquad d = 111 \qquad f = 1100$$

unde se pot observa următoarele:

- lungimea unui cod depinde de frecvența de apariție a caracterului respectiv;
- un cod nu este prefix pentru alt cod.

Astfel secvența

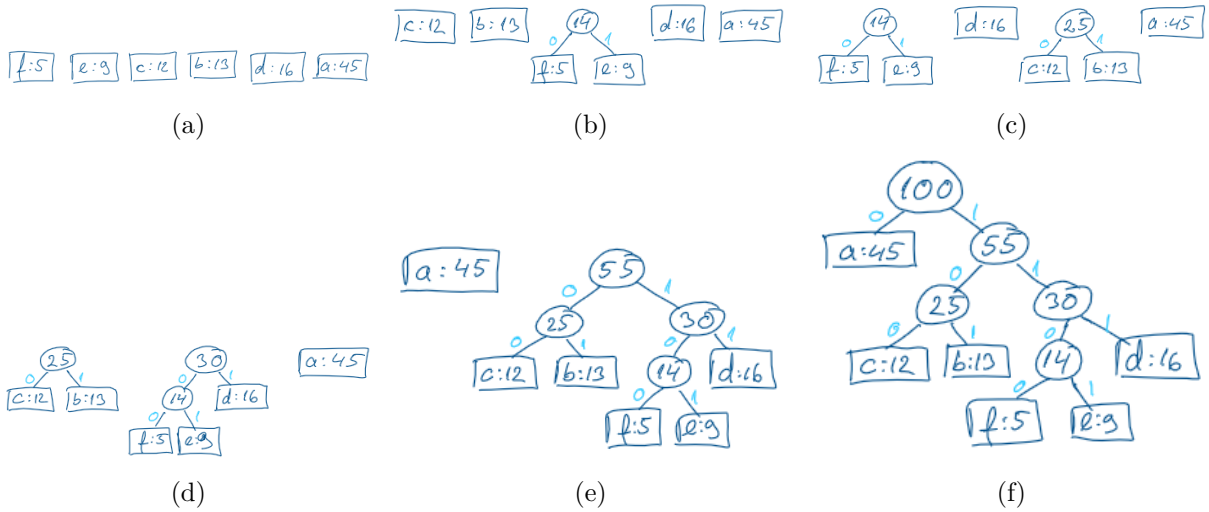
$$aaaaabbbbcccddef$$

codificată Huffman este

$$00000101101101101100100100111111111011100$$

unde numărul de biți necesari pentru a transmite mesajul inițial (dacă fiecare caracter este codificat ASCII) este $8 \cdot 16 = 128$ iar numărul de biți necesari pentru a transmite mesajul codificat este 40.

La recepție, pentru a decodifica mesajul, se parcurge arborele de la rădăcină după biții din mesaj. Deoarece acest cod se aplică asupra unor date pentru care se cunosc caracteristicile se consideră că arborele de codificare/decodificare este cunoscut la destinație (ex. pentru un mesaj transmis în limba română se cunosc frecvențele de apariție ale caracterelor alfabetului pentru limba română la ambele capete ale canalului de comunicație).



(a) (b) (c) (d) (e) (f)

Figura 5: Paşii urmaţi de algoritmul Huffman pentru a codifica caracterele din setul C.

Problema 7 Pentru un graf $G = (V, E)$ algoritmul lui Kruskal găsește arborele minim de acoperire. Algoritmul găsește o muchie sigură cu pondere minimă care unește doi arbori din pădure. Implementarea folosește tipul abstract de data *disjoint set* pentru a menține seturi disjuncte de vârfuri (arbori).

mst_kruskal(G, w)
1: $A = \varnothing$
2: **for** $v \in V$ **do**
3: make_set(v)
4: sortare muchii crescător după ponderea w
5: **for** $(u, v) \in E$ luate crescător după w **do**
6: **if** find_set(u) $\neq$ find_set(v) **then**
7: $A = A \cup (u, v)$
8: union(u,v)
9: **return** A

Figura 6 prezintă execuția algoritmului lui Kruskal pentru graful din figura 2a. Muchiile roșii aparțin pădurii A care crește în arbore de acoperire. Algoritmul consideră fiecare muchie din graf în ordine crescătoare după pondere. Săgeata indică muchia analizată în pasul respectiv. Dacă o muchie unește doi arbori disjuncți din pădure atunci muchia respectivă este adăugată pădurii, unind astfel cei doi arbori.

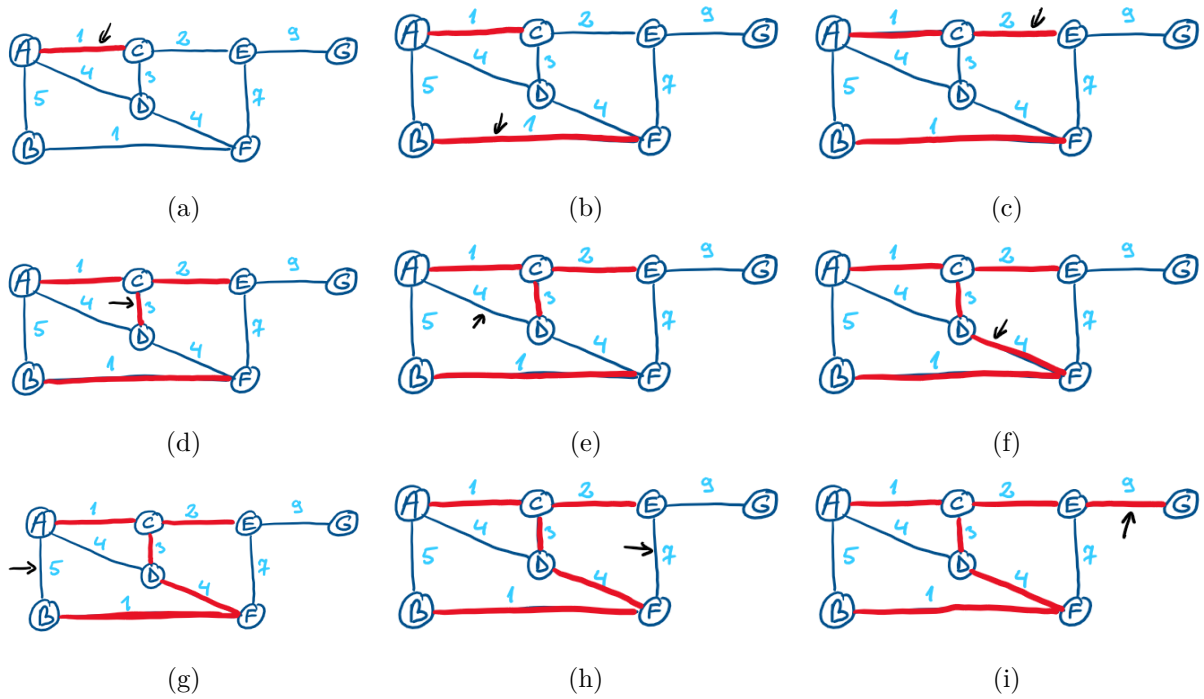


Figura 6: Execuţia algoritmului Kruskal. Muchiile roşii aparţin pădurii A care creşte în arbore de acoperire. Algoritmul consideră fiecare muchie din graf în ordine crescătoare după pondere. Săgeata indică muchia analizată în pasul respectiv. Dacă o muchie uneşte doi arbori disjuncţi din pădure atunci muchia respectivă este adăugată pădurii, unind astfel cei doi arbori.

Algoritmul lui Prim găsește arborele minim de acoperire pentru $G = (V, E)$ în mod similar cu algoritmul lui Dijkstra pentru drumul de costul minim. Muchiile din setul A formează un singur arbore.

Pentru algoritm, r este vârful rădăcină (vârful de unde începe construcția arborelui minim de acoperire). Algoritmul folosește o coadă de priorități Q, vârfurile care nu sunt în arbore sunt în coadă ordonate după atributul key.

mst_prin(G,w,r)

1: **for** $u \in V$ **do**
2: $\quad u.key = \infty$
3: $\quad u.\pi = NIL$
4: $r.key = 0$
5: $Q = V$
6: **while** $Q \neq \varnothing$ **do**
7: $\quad$ u = extract_min(Q)
8: $\quad$ **for** $v \in Adj[u]$ **do**
9: $\quad\quad$ **if** $v \in Q$ și $w(u, v) < v.key$ **then**
10: $\quad\quad\quad v.\pi = u$
11: $\quad\quad\quad v.key = w(u, v)$

Figura 7 prezintă execuția algoritmului lui Prim pentru graful din figura 2a. Tabelul din figura 7i prezintă evoluția atributelor key și π pentru fiecare nod pe parcursul execuției algoritmului iar

tabelul 1 prezintă valoarea finală a atributelor *key* și π pentru fiecare vârf.

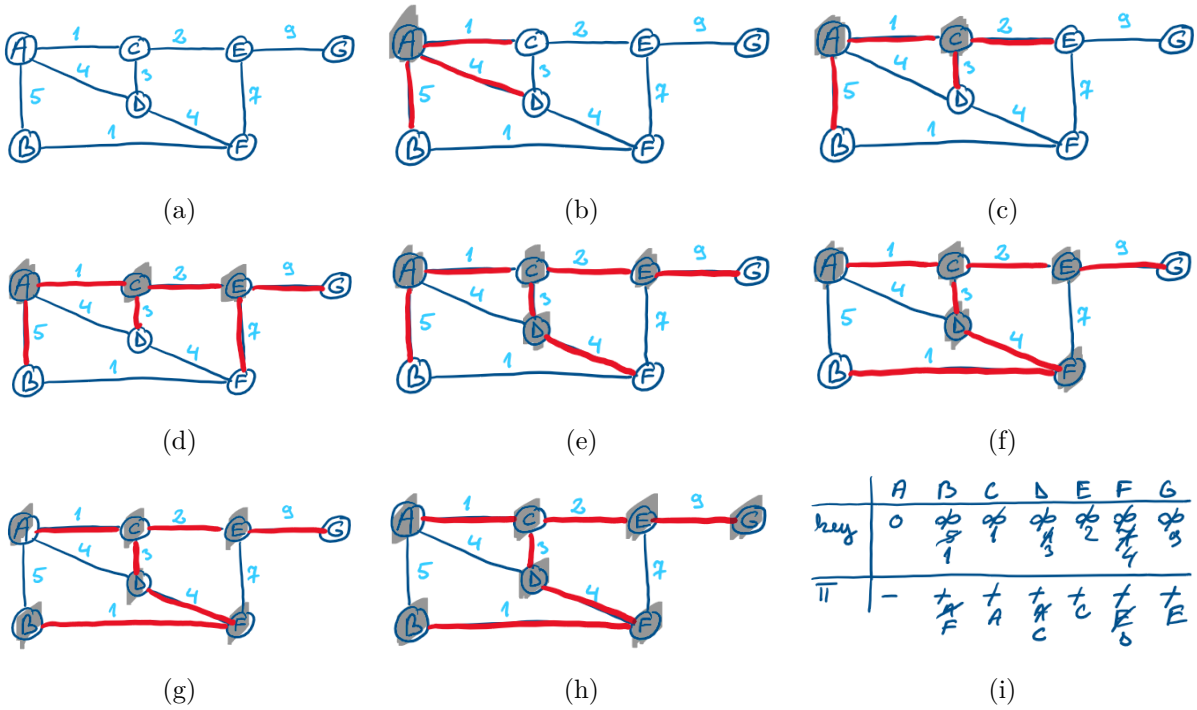


Figura 7: Execuţia algoritmului Prim pentru graful din figura 2a. Vârful rădăcină este vârful A, muchiile roşii şi vârfurile pe fundal gri aparţin arborelui construit.

Tabela 1: Valoarea finală a atributelor *key* şi π pentru fiecare nod din graf după execuţia algoritmului lui Prim, vârful A este vârful sursă.

	A	B	C	D	E	F	G
key	0	1	1	3	2	4	9
π	-	F	A	C	C	D	E

Problema 9 Pentru a determina numărul de arbori se utilizează formula lui Cayley. Rezultatul prezentat de Cayley indică câți arbori se pot construi cu n vârfuri (sau dacă se începe de la un graf complet K_n și se elimină muchii până se obține un arbore formula lui Cayley spune în câte feluri diferite se poate obține un arbore din graful complet K_n).

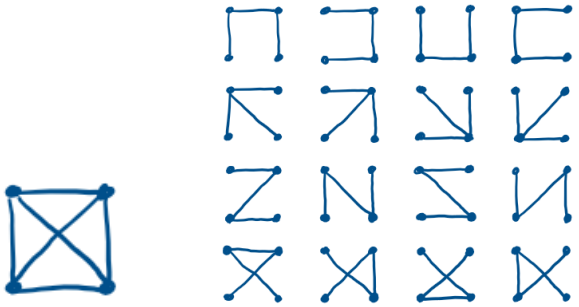
Astfel formula lui Cayley este:

$$|T| = n^{n-1}.$$

Fie graful complet K_4, figura 8 prezintă câți arbori distincți se pot obține din K_4 (se consideră etichetate vârfurile grafului). Pentru K_4 se pot construi 4^2 arbori.

Pentru graful K_{1000} din cerință există un subgraf K_3 pentru care fiecare muchie are ponderea 1 iar restul muchiilor din K_{1000} au ponderea 2. Pentru a determina câți arbori minimi de acoperire se pot construi se țin cont de următoarele:

- din K_3 se pot construi 3 arbori de acoperire;
- subgraful K_3 se poate considera ca un singur vârf în graful inițial, astfel graful inițial devenind un K_{1000} unde $|V| = 1000 - 2$;



(a) (b)

Figura 8: Graful complet K_4 şi arborii obţinuţi din K_4.

- numărul de arbori ce se pot construi pentru graful din cerința 9 este

$$3 \cdot 998^{996}.$$

Problema 10

Demonstraţie. $(a) \Rightarrow (b)$: deoarece un arbore este conectat, oricare două vârfuri sunt conectate de un lanţ simplu.

Prin contradicţie, se presupune că vârfurile u şi v sunt conectate de două lanţuri simple distincte p_1 şi p_2 (a se vedea figura 9). Fie w vârful unde cele două lanţuri diverg (w este primul vârf din lanţurile p_1 şi p_2 pentru care succesorul în p_1 este x şi succesorul în p_2 este y, $x \neq y$) şi z primul vârf în care lanţurile reconverg (z este următorul vârf după w care este comun în p_1 şi p_2). Fie p' un sub-lanţ în p_1 de la w la z care trece prin x şi p'' un sub-lanţ în p_2 de la w la z care trece prin y. Lanţuri p' şi p'' au în comun doar vârfurile extreme.

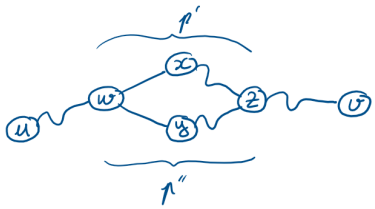


Figura 9

Se poate obţine un *ciclu* dacă se concatenează lanţul p' şi lanţul p'' inversat. Acest lucru contrazice presupunerea iniţială: G este un arbore. Astfel, dacă G este un arbore, poate exista cel mult un lanţ simplu între oricare două vârfuri din arbore.

$(b) \Rightarrow (c)$: dacă oricare două vârfuri din G sunt conectate de un lanţ simplu atunci G este conex. Fie $(u, v) \in E$ o muchie, (u, v) este un lanţ de la u la v, (u, v) trebuie să fie un lanţ unic în G. Dacă se elimină (u, v) din G atunci graful va avea două componente conexe.

$(c) \Rightarrow (d)$: se presupune că G este conex şi $|E| \geq |V| - 1$, se demonstrează prin inducţie că $|E| \leq |V| - 1$. Un graf conex cu $n = 1$ sau $n = 2$ vârfuri are $n - 1$ muchii. *Se presupune* că G are $n \geq 3$ vârfuri şi că toate grafurile G ce satisfac (c) şi au mai puţin de n vârfuri satisfac şi $|E| \leq |V| - 1$. Ştergerea unei muchii din G împarte graful în $k \geq 2$ componente conexe (mai exact $k = 2$). Fiecare componentă satisface (c) altfel G nu ar satisface (c). Dacă fiecare componentă conexă V_i are setul muchiilor E_i este un arbore conex, deoarece fiecare componentă are mai puţin de $|V|$ vârfuri atunci $|E_i| \leq |V_i| - 1$ (inducţie). Deci, numărul muchiilor din toate componentele conexe este cel mult $|V| - k \leq |V| - 2$. Dacă se adaugă şi muchia ştearsă se obţine $|E| \leq |V| - 1$.

$(d) \Rightarrow (e)$: se presupune că G este conex şi că $|E| \leq |V| - 1$. Trebuie arătat că G este aciclic. *Se presupune* că G conţine un ciclu simplu de k vârfuri (nu conţine de două ori aceeaşi muchie). Fie $G_k = (V_k, E_k)$ subgraful din G care conţine ciclul, atunci $|V_k| = |E_k| = k$. Dacă $k < |V|$, deoarece G este conex trebuie să existe un vârf $v_{k+1} \in V - V_k$ care este adiacent unui vârf $v_i \in V_k$. Se defineşte $G_{k+1} = (V_{k+1}, E_{k+1})$ ca şi subgraful lui G cu $V_{k+1} = V_k \cup \{v_{k+1}\}$ şi $E_{k+1} = E_k \cup \{(v_i, v_{k+1})\}$, $|V_{k+1}| = |E_{k+1}| = k + 1$. Dacă $k + 1 < |V|$ se poate defini G_{k+2} în acelaşi mod, şi tot aşa, până se obţine $G_n = (V_n, E_n)$ unde $n = |V|$, $V_n = V$ şi $|E_n| = |V_n| = |V|$. Deoarece G_n este un subgraf a lui G atunci $E_n \subseteq E$ şi $|E| \geq |V|$ ceea ce contrazice presupunerea că $|E| \leq |V| - 1$. G este aciclic.

$(e) \Rightarrow (f)$: se presupune că G este aciclic şi $|E| \geq |V|$. Fie k numărul componentelor conexe din G. Fiecare componentă conexă este un arbore, deoarece (a) implică (e), suma muchiilor tuturor componentelor conexe din graf este $|V| - k$. În consecinţă $k = 1$ şi G este un arbore. Deoarece (a) implică (b), oricare două vârfuri din graf sunt conectate printr-un lanţ simplu. Astfel adăugând o muchie în G se formează un ciclu.

$(f) \Rightarrow (a)$: se presupune că G este aciclic şi prin adăugarea unei muchii în G se formează un ciclu. Trebuie arătat că G este conex. Fie $G = (V, E)$ şi $u, v \in V$ alese aleator. Dacă vârfurile u şi v nu sunt adiacente, prin adăugarea muchiei (u, v) se creează un ciclu care conţine muchii din G şi muchia (u, v). Astfel, ciclul fără muchia (u, v) trebuie să conţină un lanţ de la vârful u la vârful v, deoarece u şi v au fost alese aleator graful G trebuie să fie conex. □

Suport seminar algoritmica grafurilor
IV. Cicluri Euleriene și Hamiltoniene

Probleme:

1. Sunt următoarele grafuri Euleriene?

(a) (b)

2. Fie un graf complet K_n. Ce valoare are n astfel încât graful conține un ciclu Eulerian? Ce valoare are n astfel încât graful conține un lanț Eulerian? Fie un graf bipartit complet $K_{n,m}$, ce valoare are n și m astfel încât graful conține un ciclu Eulerian?

3. Vârfurile unui graf neorientat sunt numerotate de la $1, \ldots, 100$. Între vârfurile i și j există o muchie dacă $|i - j| \leq 2$. Conține acest graf un lanț Eulerian? Dar un ciclu Eulerian?

4. Să se găsească un ciclu Eulerian în graful din figura 2.

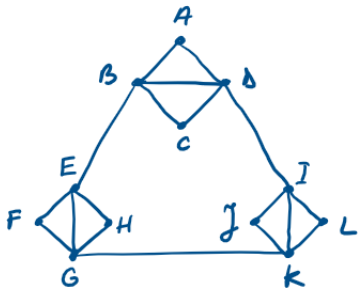


Figura 2

5. Să se găsească un ciclu Eulerian în graful din figura 3 pentru care suma ponderilor este minimă.

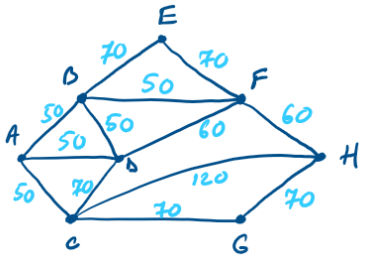


Figura 3

6. Vârfurile unui graf neorientat sunt numerotate de la $1, \ldots, 100$. Între vârfurile i și j există o muchie dacă $|i - j| \leq 2$. Conține acest graf un lanț Hamiltonian? Dar un ciclu Hamiltonian?

7. Să se deseneze un graf 3 regular care să nu conțină un ciclu Hamiltonian.

8. Alin cumpără 100 de mărgele roșii, 150 de mărgele albastre și 200 de mărgele verzi. Să se demonstreze că Alin poate face un colier cu toate mărgelele astfel încât să nu existe mărgele de aceeași culoare vecine în colier.

9. La un hotel se cazează 100 de persoane. La început toți vizitatorii sunt prieteni între ei. În fiecare seară la cină se așează toți vizitatorii la o masă rotundă, din păcate cu ocazia fiecărei mese rotunde persoanele vecine se ceartă între ele și prietenia se termină. Înainte de cină toată lumea se așează la masă astfel încât să nu aibă ca și vecini vizitatori cu care s-au certat la mesele precedente, dacă acest lucru nu este posibil toți pleacă acasă în acea zi. Să se demonstreze că minim 25 de nopți vor fi cazate în hotel cele 100 de persoane.

10. Fie activitățile dintr-un proiect prezentate în tabelul de mai jos împreuna cu durata de execuție a fiecărei activități. Să se determine activitățile critice din proiect. (Dacă apar întârzieri în execuția activităților critice durata proiectului se va extinde peste planificarea inițială.)

Tabela 1: Activităţile, interdependenţă între activităţi şi timpul de execuţie.

activitate	predecesor	durata
A	-	3
B	A	4
C	A	2
D	B	5
E	C	1
F	C	2
G	D, E	4
H	F, G	3

Soluții

Problema 3 Fie un graf simplu, neorientat $G = (V, E)$:

- un lanț eulerian este un lanț simplu care conține toate muchiile din G;
- un ciclu eulerian este un lanț simplu care conține toate muchiile din G și extremitățile lanțului coincid;

- un graf eulerian este un graf simplu care conține un ciclu eulerian.

Graful din cerință este reprezentat în figura 4, se observă că doar două vârfuri au grad impar ⇒ graful conține un lanț eulerian dar nu conține un ciclu eulerian.

Corolar 6.3 un graf conex $G = (V, E)$ conține un lanț eulerian dacă și numai dacă are cel mult două vârfuri de grad impar.

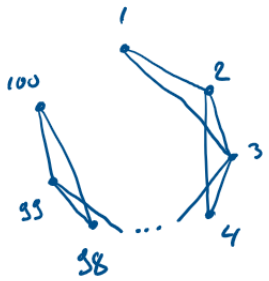


Figura 4

Problema 4 Pentru a rezolva problema se va aplica algoritmul lui Fleury care caută un ciclu eulerian în graf. Algoritmul primește la intrare graful și se presupune că există un ciclu eulerian în graf.

Fleury(G)

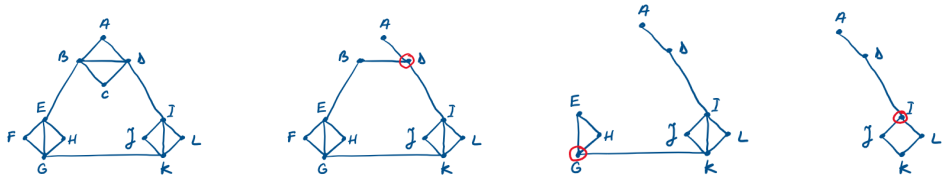
1: $x = u \in G$
2: $A = E$
3: $L = \varnothing$
4: **while** $A \neq \varnothing$ **do**
5: alege $e \in A$ astfel încât e incidentă lui x ($x \in g(e)$) și dacă se poate e să nu fie punte
6: $L = L \cup \{e\}$
7: **if** $|g(e)| = 2$ (dacă s-a parcurs muchia) **then**
8: $x = v \in g(e) \setminus \{x\}$ (se merge la extremitatea nevizitată a muchiei e)
9: $A = A \setminus \{e\}$
10: **return** L

Figura 5 prezintă evoluția grafului dacă se aplică algoritmul lui Fleury pe graful din figura (5a). Se presupune că algoritmul pornește de la vârful A și găsește lanțul $\langle AB, BC, CD\rangle$ (figura (5b)), algoritmul nu alege în continuare muchia AD deoarece aceasta este o muchie punte (dacă se elimină muchia AD din graf graful va avea două componente conexe). Se presupune că lanțul eulerian format până în figura (5c) este format din muchiile $\langle AB, BC, CD, DB, BE, EF, FG\rangle$, în continuare algoritmul nu alege muchia GK deoarece aceasta este o muchie punte (similar cu situația din figura (5d) unde muchia ID este muchie punte și nu poate fi încă adăugată ciclului eulerian). La final ciclu eulerian este $\langle AB, BC, CD, DB, BE, EF, FG, GE, EH, HG, GK, KI, IJ, JK, KL, LI, ID, DA\rangle$.

O altă variantă pentru a găsi un ciclu eulerian într-un graf este algoritmul lui Hierholzer.

Hierholzer

1: Se identifică un ciclu simplu R_i în G și se marchează muchiile lui R_i.
2: dacă R_i conține toate muchiile lui G algoritmul se oprește, R_i este eulerian.



(a) (b) (c) (d)

Figura 5: Algoritmul lui Fleury aplicat pe graful de la problema 4.

3: R_i cu conține toate muchile grafului, fie v_i un vârf al lui R_i incident la o muchie nemarcată e_i.
4: Se construiește un ciclu simplu Q_i pornind de la $v_i$din muchiile nemarcate pornind de la e_i. Se marchează muchiile lui Q_i.
5: Se creează R_{i+1} înlănțuind Q_i în R_i la vârful v_i.
6: $i = i + 1$ si se sare la pasul 2.

Problema 5 Problema de a găsi un ciclu Eulerian de pondere minimă mai este cunoscută sub numele de "problema poștașului chinezesc".

Formal problema este definită astfel: pentru un graf conex G să se găsească un ciclu de pondere minimă care conține toate muchiile din G. Problema a apărut ca și răspuns la următoarea întrebare: un poștaș trebuie să își parcurgă ruta asignată înainte de a se întoarce la sediu, poștașul vrea să meargă pe jos cât mai puțin, trebuie găsit traseul de lungime minimă pentru poștaș.

Se consideră graful $G = (V, E)$ cu funcția de pondere asociată $\omega : E \to \mathbb{R}^+$, ponderea unei muchii $w(e)$ poate fi interpretată ca lungimea muchiei (a, b) (a străzii ce trebuie parcursă de poștaș), durata necesară parcurgerii muchiei (a,b), costul necesar parcurgerii muchiei/străzii cu un vehicul, etc. Pentru $H \subseteq G$ și o secvență de muchii $W = e_1, \ldots, e_r$ în G

$$\omega(W) = \sum_{e \in H} \omega(e),\ \omega(W) = \sum_{i=1}^{r} \omega(e_i).$$

Problema poștașului chinezesc cere găsirea unui ciclu W în G de pondere minimă $\omega(W)$ (găsirea unui ciclu eulerian de pondere minimă - poștașul vrea să parcurgă fiecare stradă o singură dată).

Dacă G este eulerian se poate folosi algoritmul lui Fleury sau Hierholzer pentru a găsi un ciclu eulerian în G.

Dacă G nu este eulerian cum se poate găsi un ciclu eulerian de lungime minimă? Se dublează muchii în G astfel încât lungimea ciclului eulerian rezultat să fie minimă. Poștașul va trebui să traverseze de două ori aceeași strada, stradă de lungime (timp, cost) minimă astfel încât traseul său să aibă lungimea minimă.

Un algoritm pentru problema poștașului chinezesc:

poștaș_chinezesc(G)

1: Se identifică vârfurile de grad impar.
2: Se formează toate perechile de vârfuri de grad impar.
3: Pentru fiecare pereche se caută muchiile de cost minim care conectează vârfurile.
4: Se caută perechea pentru care suma ponderilor este minimă.
5: Se dublează muchiile identificate la pasul 4 în graful inițial.
6: Costul drumului poștașului este suma ponderilor muchiilor din graf plus ponderile muchiilor dublate în pasul 5.
7: Se caută un ciclu eulerian în graf.

Algoritmul transformă un graf într-un graf eulerian.
Exemplu Fie graful din figura 6, se identifică vârfurile de grad impar $\{A, B, D, D\}$.

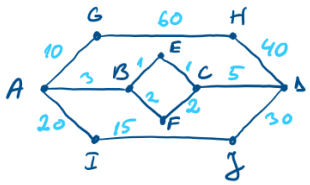


Figura 6

Se formează toate perechile de vârfuri de grad impar $\{\{AB, CD\}, \{AC, BD\}, \{AD, BC\}\}$ și se determină suma ponderilor muchiilor, tabelul 2.

Tabela 2: Perechile de vârfuri de grad impar şi suma ponderilor acestora. Pentru perechea AC lanţul de cost minim care leagă vârfurile A şi C este $\langle AB, BE, EC\rangle$ iar suma ponderilor acestui lanţ este $3 + 1 + 1 = 5$.

AB = 3	AC = 5	AD = 8
CD = 5	BD = 7	BC = 2
$\omega(AB) + \omega(CD) = 8$	$\omega(AC) + \omega(BD) = 12$	$\omega(AD) + \omega(BC) = 10$

Se alege grupul de perechi pentru care suma ponderilor este minimă, în acest caz $\{AB, CD\}$. În graful din figura 6 se dublează muchiile $(A, B), (C, D)$, costul ciclului eulerian pentru poștaș este $\sum_{e\in E} \omega(e) + \omega(AB) + \omega(CD) = 179 + 3 + 5 = 187$.

Problema 6 Fie un graf simplu, neorientat $G = (V, E)$:

- un lanț hamiltonian este un lanț simplu care conține toate vârfurile din G;
- un ciclu hamiltonian este un lanț simplu care conține toate vârfurile din G și extremitățile lanțului coincid;
- un graf hamiltonian este un graf simplu care conține un ciclu hamiltonian.

Un graf cu $n = 5$ vârfuri și muchiile conform cerinței poate fi reprezentat că în figura 7. Se observă că acest graf conține un lanț hamiltonian și un ciclu hamiltonian, un posibil ciclu hamiltonian $\langle 13, 35, 54, 42, 21\rangle$.

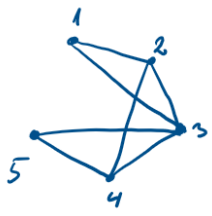


Figura 7

Problema 7 Graful care respectă cerință este reprezentat în figura 8.

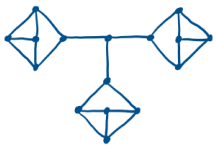


Figura 8

Problema 8 Cu mărgelele cumpărate se poate forma un graf $G = (V, E)$ simplu neorientat și neponderat, $|V| = 450$, în care între două muchii sunt conectate dacă extremitățile au culori diferite.

Pentru graful format gradul vârfurilor este:

$$d(v_{rosii}) = d(v_{verzi}) + d(v_{albastre}) = 350,$$
$$d(v_{verzi}) = d(v_{rosii}) + d(v_{albastre}) = 250,$$
$$d(v_{albastre}) = d(v_{verzi}) + d(v_{rosii}) = 300.$$

Pe baza teoremei lui Dirac graful este hamiltonian ($\delta(G) = 250, \frac{n}{2} = 225, 250 \geq 225$) și se poate construi colierul care să respecte cerința (se poate găsi un ciclu hamiltonian pentru care vârfurile adiacente au culori diferite).

Teorema lui Dirac
Fie G un graf de ordinul $n \geq 3$. Dacă $\delta(G) \geq \frac{n}{2}$ atunci G este hamiltonian.
sau
Fie G un graf de ordinul $n \geq 3$. Dacă $\forall u \in V, d(u) \geq \frac{n}{2}$ atunci G este hamiltonian.

Problema 9 Fie i ziua curentă în care persoanele sunt cazate la hotel, $i = 1, \ldots$. În prima zi toate persoanele cazate sunt prietene, se poate forma un graf complet din care să se aleagă un ciclu hamiltonian care reprezintă schema de așezare a persoanelor la masa. Gradul vârfurilor în prima zi este, în ziua i gradul unui vârf este $99 - 2(i - 1)$.

Pe baza teoremei lui Dirac numărul minim de mese organizate este 25.

Problema 10 Problema se mai numește problema ordonanței sau problema drumului critic și presupune programarea activităților unui proiect. Pentru realizarea unui proiect trebuie realizate un număr mai mare sau mai mic de activități. Deoarece între aceste activități exista o interdependență și o anumită coordonare este foarte importantă programarea desfășurării lor și cunoașterea punctelor sensibile pentru ca obiectivul propus să se realizeze în condiții de eficiență (timp, cost, etc.).

Pentru problema 10 se consideră doar condițiile ce țin de timp, unele activități pot începe mai devreme sau imediat după terminarea altor activități bine precizate. Pentru fiecare activitate se cunoaște durata sa în timp și activitățile ce trebuie realizate înainte ca ea să fie abordată. Drumului critic în graful asociat proiectului permite determinarea timpului minim necesar de realizare a proiectului.

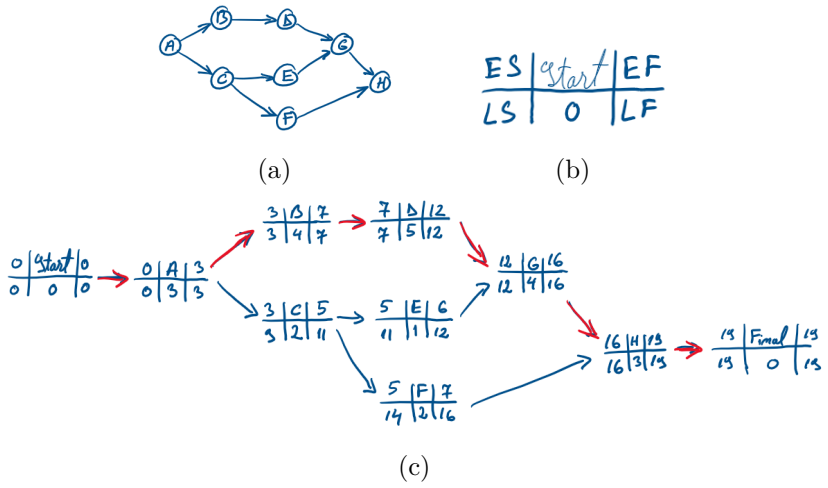
Pentru a determina drumul critic se construiește un graf orientat în care vârfurile reprezinta activitățile proiectului și arcele indică dependență între activități. Figura (9a) prezintă graful asociat proiectului. Pe lângă activitățile descrise în proiect graful va mai conține două vârfuri care reprezintă începutul proiectului și sfârșitul acestuia, figura (9b).

Pentru a începe o activitate trebuie ca toate activitățile care se află pe un drum de la debutul proiectului (vârful start) până la vârful corespunzător activității respective să fie finalizate. Se notează cu ES (*Early Start*) timpul cel mai devreme de începere a activității i și se determină ca valoarea maximă a drumurilor de la vârful start la vârful i (suma ponderilor arcelor ce compun drumul este

maximă). Timpul cel mai devreme de terminare a activității i, EF (*Early Finish*), se determină ca valoarea maximă a drumurilor de la vârful start la vârful i la care se adaugă timpul de execuție a activității respective ($ES +$ timp_execuție_ activitate_i). După ce se determină momentele cele mai timpurii de începere a activităților se pot determina și momentele cele mai târzii de începere a activităților care nu conduc la o anumită amânare a proiectului. Timpul cel mai târziu de finalizare, LF (*Late Finish*), se determină ca valoarea minimă a drumului de la vârful $final$ la activitatea i. Timpul cel mai târziu de începere a activității, LS (*Late Start*), se determină ca valoarea LF din care se scade timpul de execuție a activității i ($LF -$ timp_execuție_ activitate_i). Pentru vârful *Start* cei patru timpi au valoarea 0 ($ES = EF = LS = LF = 0$), pentru vârful *Final* timpul de execuție este 0.

Figura (9b) prezintă vârful *Start* și notația folosită pentru timpii ES, EF, LS, LF, timpul de execuție pentru activitatea *Start* este 0.

Figura (9c) prezintă valoarea timpilor ES, EF, LS, LF pentru fiecare vârf și drumul critic format din vârfurile (activitățile) pentru care acești timpi au aceeași valoare. Drumul critic indică activitățile din proiect care nu trebuie amânate pentru a putea termina proiectul la termen. Activitățile care nu se afla pe drumul critic pot fi pornite în intervalul dat de $[ES, LS]$ și terminate în intervalul $[EF, LF]$ și proiectul se va termina la termen.



(a) (b)

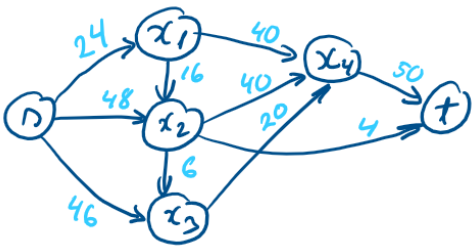
(c)

Figura 9: Drumul critic pentru proiectul descris în tabelul 1.

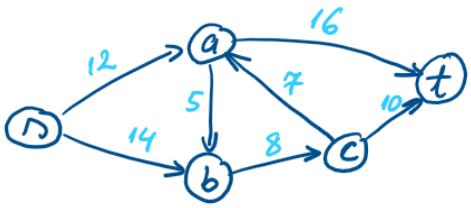
Suport seminar algoritmica grafurilor
V. Rețele de flux

Probleme:

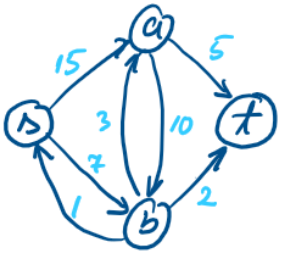
1. Folosind algoritmul Ford-Fulkerson să se determine fluxul maxim pentru graful din figura (1a).
2. Folosind algoritmul de pompare preflux să se determine fluxul maxim pentru graful din figura (1b).
3. Folosind algoritmul de pompare topologică să se determine fluxul maxim pentru graful din figura (1c).



(a)



(b)



(c)

Figura 1

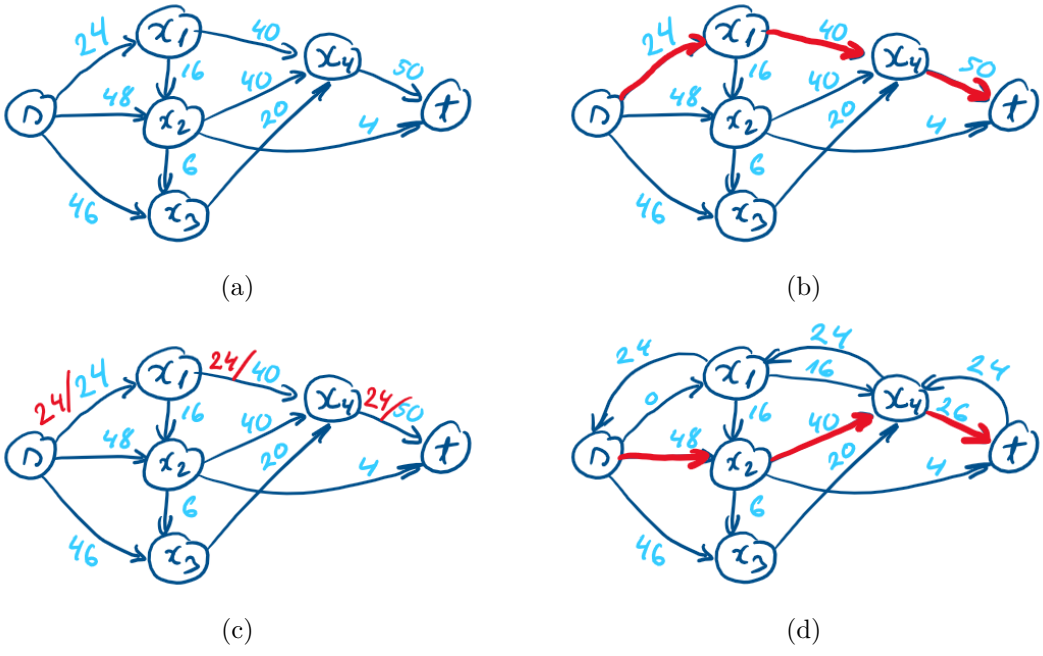
Rezolvări

Problema 1 Pentru a rezolva problema se aplică algoritmul Ford-Fulkerson. În fiecare iterație a algoritmului se găsește o cale reziduală p și se folosește p pentru a modifica fluxul f. Se înlocuiește f cu $f \uparrow f_p$ și se obține un nou flux cu valoarea $|f| + |f_p|$.

Algoritmul nu specifică cum se găsește calea reziduală. Dacă f^* reprezintă fluxul maxim din rețeaua de transport, în cel mai rău caz algoritmul execută liniile $1-9$ de cel mult $|f^*|$ ori deoarece fluxul crește în fiecare iterație cel puțin cu o unitate.

FORD_FULKERSON(G, s, t)
1: **for** $(u, v) \in E$ **do**
2: $\quad (u, v).f = 0$
3: **while** există o cale reziduală p de la s la t în graful rezidual G_f **do**
4: $\quad c_f(p) = min\{c_f(u, v)|(u, v) \in p\}$
5: $\quad$ **for** fiecare arc $(u, v) \in p$ **do**
6: $\quad\quad$ **if** $(u, v) \in E$ **then**
7: $\quad\quad\quad (u, v).f = (u, v).f + c_f(p)$
8: $\quad\quad$ **else**
9: $\quad\quad\quad (v, u).f = (v, u).f - c_f(p)$
10: **return** f

Figura 2 prezintă rezultatul fiecărei iterații pentru algoritm, în cazul căilor reziduale indicate. Figurile (2a)-(2g) prezintă iterațiile succesive ale buclei **while**. Coloana dreaptă prezintă grafurile reziduale G_f din linia 3 a algoritmului și calea reziduală (formată din arcele colorate în roșu). Coloana stângă prezintă noul flux din rețea. Pentru a indica capacitatea unui arc și fluxul transmis pe acel arc se folosește notația *flux/capacitate* (cu roșu se prezintă valoarea fluxului și cu albastru valoarea capacității). În prima iterație graful rezidual este identic cu rețeaua de flux (figurile (2a) și (2b)).



(a) (b)

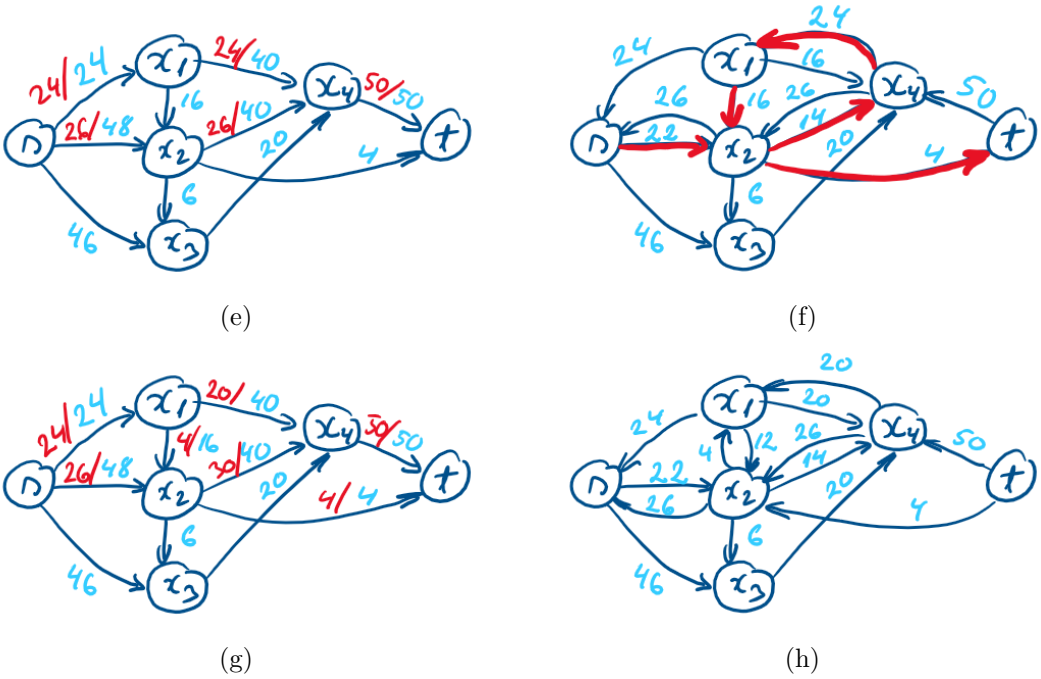
(c) (d)

Figura 2: Algoritmul Ford-Fulkerson aplicat grafului din figura (1a).

De exemplu, pentru graful rezidual din figura 2b) se găsește calea reziduală $s \to x_1 \to x_4 \to t$ cu capacitatea $c_f = min\{24, 40, 50\}$. Pentru această cale reziduala fluxul maxim în rețeaua de transport crește la valoarea $|f| = 24$ și va fi transmis pe calea reziduala $s \to x_1 \to x_4 \to t$ (figura (2c)). Graful rezidual (2d) prezintă rețeaua de transport pentru fluxul $|f| = 24$ și calea reziduală

identificată.

Algoritmul se oprește când în graful rezidual nu se mai găsește o cale rezidual, un drum de la vârful s la vârful t (figura (2h)). Fluxul maxim în graf are valoarea $|f| = 54$.



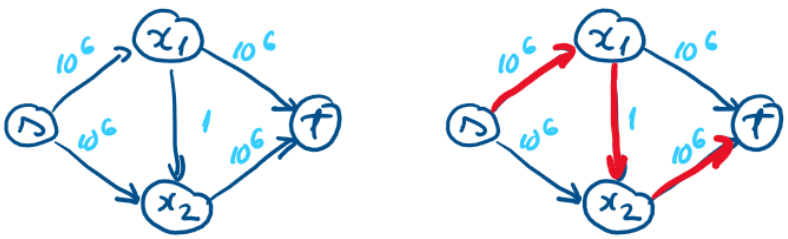
(e) (f)

(g) (h)

Figura 2: (continuare) Algoritmul Ford-Fulkerson aplicat grafului din figura (1a).

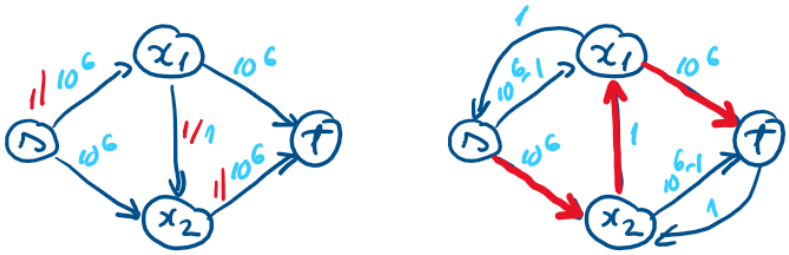
Pentru valori întregi ale capacităților și pentru un flux maxim $|f^*|$ mic, timpul de execuție pentru algoritmul Ford-Fulkerson este bun. Figura (3a) prezintă un caz extrem, ce se poate întâmpla în cel mai rău caz, pentru o rețea de flux mică pentru care $|f^*|$ este mare. Pentru această rețea fluxul maxim are valoarea $2 \cdot 10^6$ unde 10^6 unități de flux traversează drumul $s \to x_1 \to t$ și 10^6 unități de flux traversează drumul $s \to x_2 \to t$.

Dacă algoritmul Ford-Fulkerson găsește prima cale rezidual $s \to x_1 \to x_2 \to t$ (figura (3b)), fluxul maxim în rețea după prima iterație are valoarea 1. Graful rezidual rezultat este prezentat în figura (3d). Dacă în a doua iterație se găsește calea rezidual $s \to x_2 \to x_1 \to t$, ca în figura (3d), fluxul în rețea va avea valoarea 2. Dacă în iterațiile impare se alege calea rezidual $s \to x_1 \to x_2 \to t$ și în iterațiile pare se alege calea rezidual $s \to x_2 \to x_1 \to t$ se vor executa $2 \cdot 10^6$ iterații, fluxul crește în fiecare iterație cu o unitate.



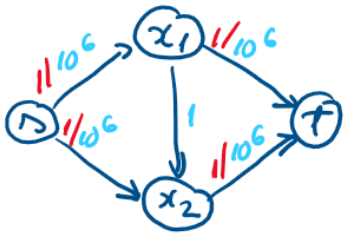
(a) (b)

Figura 3: Un exemplu pentru care algoritmul Ford-Fulkerson poate lua $\theta(E|f^*|)$ timp, f^* reprezintă fluxul maxim în graf. Pentru această reţea fluxul maxim este $|f^*| = 2 \cdot 10^6$.

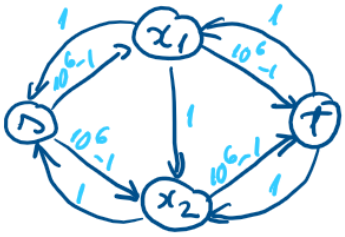


(c)

(d)



(e)



(f)

Figura 3: (continuare) Un exemplu pentru care algoritmul Ford-Fulkerson poate lua $\theta(E|f^*|)$ timp, f^* reprezintă fluxul maxim în graf. Pentru această reţea fluxul maxim este $|f^*| = 2 \cdot 10^6$.

Pentru a îmbunătăți algoritmul Ford-Fulkerson se vor cauta căile reziduale (linia 3) folosind un algoritm de parcurgere în lățime, algoritmul rezultat se numește Edmonds-Karp. Pentru rețeaua de transport din figura (3a) fluxul maxim se va determina în două iterații.

Problema 2 Algoritmii de pompare rezolva problema fluxului maxim din rețea într-un mod mai localizat/concentrat față de abordarea Ford-Fulkerson. Aceștia se "uită" la un moment dat doar doar la un vârf și la vecinii acestuia din graful rezidual, nu caută o îmbunătățire sub forma unui drum în tot graful rezidual.

Algoritmii de pompare nu respectă proprietatea de conservare a fluxului pe parcursul execuției. Fluxul se poate acumula într-un nod, pe parcursul execuției, sub numele de exces de flux $u.e = \sum_{v \in V}(v,u).f - \sum_{v \in V}(u,v).f$.

Intuitiv, funcționarea algoritmilor de pompare este similară cu curgerea lichidelor printr-un sistem de conducte de diverse capacități care leagă puncte aflate la diverse înălțimi. Inițial vârful sursă s al rețelei este cel mai înalt, celelalte vârfuri fiind la înălțimea 0. Destinația t rămâne permanent la înălțimea 0. Prefluxul este inițializat încărcând la capacitate maximă toate conductele care pornesc din s. În cursul funcționării, se poate întâmpla ca fluxul (lichidul) strâns într-un vârf u din conductele ce alimentează vârful să depășească posibilitatea de eliminare prin conductele de scurgere la care este conectat vârful (restricția de conservare a fluxului nu mai este îndeplinită). Excesul de flux este stocat într-un rezervor $u.e$ al vârfului, cu capacitatea nelimitată teoretic. Pentru a echilibra rețeaua și a elimina supraîncărcarea vârfurilor, sunt efectuate două operații de bază:

- Mărirea înălțimii unui vârf. Atunci când un vârf supraîncărcat u are o conductă de scurgere orientată spre un vecin v, care nu este încărcat la capacitate maximă, u este înălțat mai sus decât v, astfel încât fluxul să poată curge din u în v prin conducta (u, v).

- Pomparea fluxului unui vârf. Un vârf u supraîncărcat, aflat la o înălțime mai mare decât un vecin v și conectat cu v printr-o conductă subîncărcată, pompează flux din rezervorul $u.e$ spre v prin conducta (u, v).

Treptat, înălțimile vârfurilor cresc monoton și pot depăși înălțimea sursei s. În acest moment, fluxul în exces din rețea se pompează sursei. Numărul de operații de mărire a înălțimii vârfurilor și de pompare a fluxului este limitat iar atunci când nu mai este posibilă o operație de acest fel, fluxul din rețea este maxim.

Algoritmul de pompare preflux și rutinele de inițializare, pompare și înălțare sunt prezentate mai jos.

INITIALIZARE_PREFLUX(G,s,t)

\\ inițializare $f(u,v)$ și $h(u,v)$, $\forall u,v \in V$
for fiecare $v \in V$ **do**
    $v.h = 0$
    $v.e = 0$
for fiecare $(u,v) \in E$ **do**
    (u,v).f=0
$s.h = |V|$
for fiecare $v \in s.Adj$ **do**
    $(s,v).f = c(s,v)$
    $v.e = c(s,v)$
    $s.e = s.e - c(s,v)$

POMPARE(u,v)

\\ condiție de aplicare: $u \notin \{s,t\} \wedge u.e > 0 \wedge c_f(u,v) > 0 \wedge u.h = v.h + 1$
\\ acțiune: pompează cantitatea de flux $\Delta_f(u,v) = min(u.e, c_f(u,v))$
$\Delta_f(u,v) = min(u.e, c_f(u,v))$
if $(u,v) \in E$ **then**
    $(u,v).f = (u,v).f + \Delta_f(u,v)$
else
    $(v,u).f = (v,u).f - \Delta_f(u,v)$
$u.e = u.e - \Delta_f(u,v)$
$v.e = v.e + \Delta_f(u,v)$

ÎNĂLȚARE(u)

\\ condiție de aplicare: $u \notin \{s,t\} \wedge u.e > 0 \wedge [u.h \leq v.h | \forall v \in V, (u,v) \in E_f]$
\\ acțiune: mărește înălțimea $u.h$
$u.h = 1 + min\{v.h | (u,v) \in E_f\}$

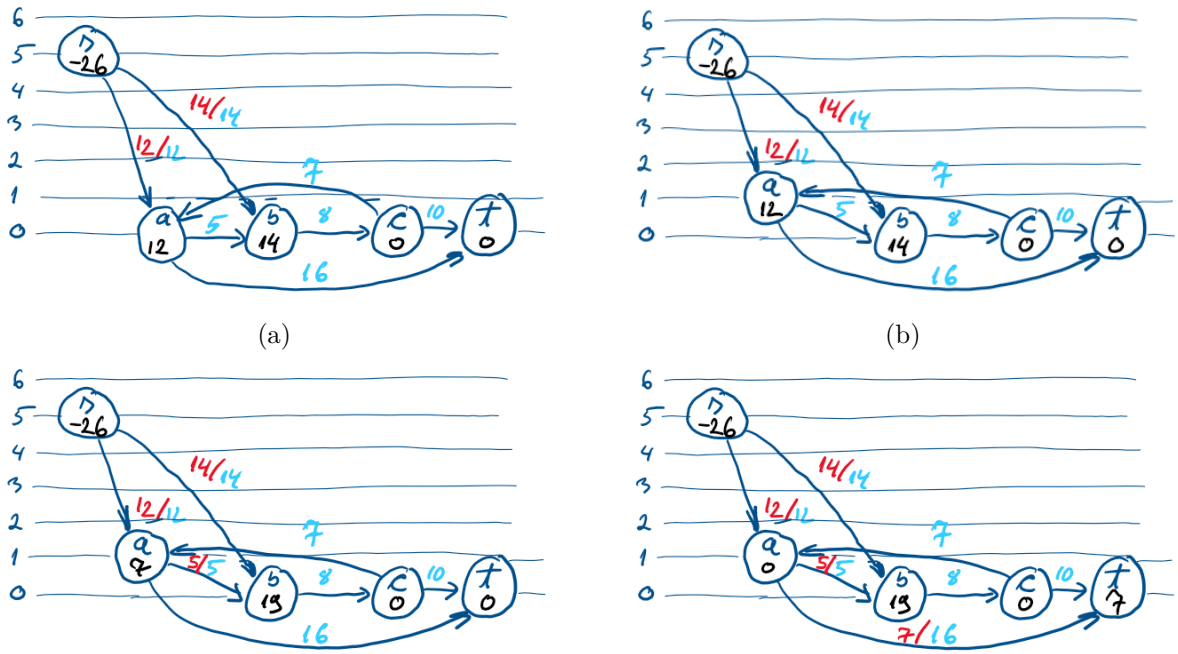
POMPARE_PREFLUX(G,s,t)

INITIALIZARE_PREFLUX(G,s,t)
while TRUE **do**
    if $\exists u \notin \{s,t\} \wedge u.e > 0 \wedge c_f(u,v) > 0 \wedge u.h = v.h + 1$ **then**
        POMPARE(u,v)
        continue
    if $\exists u \notin \{s,t\} \wedge u.e > 0 \wedge [u.h \leq v.h | \forall v \in V, (u,v) \in E_f]$ **then**
        INALTARE(u)

8: continue
9: break

Figura (4a) prezintă rețeaua de flux din figura (1b) după procedura de inițializare. Vârful sursă are atributul înălțime $s.h = 5$ și pompează fluxul maxim posibil vârfurilor vecine. Restul vârfurilor se află la înălțimea 0. Excesul de flux este notat sub eticheta vârfului (numerele colorate cu negru), excesul pentru vârfurile adiacente cu s este $a.e = 12$ și $b.e$=14.

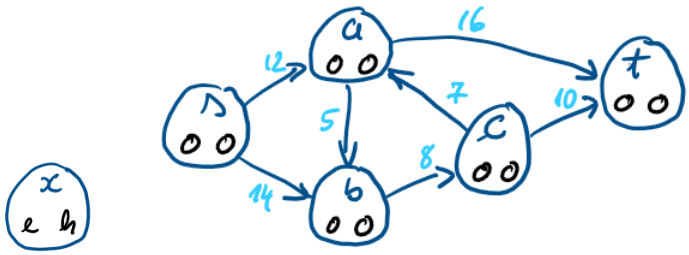
După procedura de inițializare nu se poate descărca excesul de flux din vârfurile $u \in V \setminus \{s, t\}$ și se aplică procedura de înălțare pentru vârful a, figura (4b). După acest pas $a.h = 1$. În figura (4c) se descarcă excesul de flux din vârful a în vârful b, după acest pas $b.e = 19$ și $a.e = 7$. În pasul următor (figura (4d)), restul de exces de flux din vârful a este descărcat în vârful destinație t, $a.e = 0$ și $t.e = 7$.



(a) (b) (c) (d)

Figura 4: Un exemplu de pompare preflux.

Pentru simplitatea prezentării se va adopta notația din figura (5a) unde sub eticheta unui vârf se va trece valoarea atributului exces și înălțime. (5b) prezintă rețeaua de flux cu noua notație.



(a) (b)

Figura 5: Pompare preflux. Sub eticheta vârfului se trece valoarea atributelor exces şi înălţime.

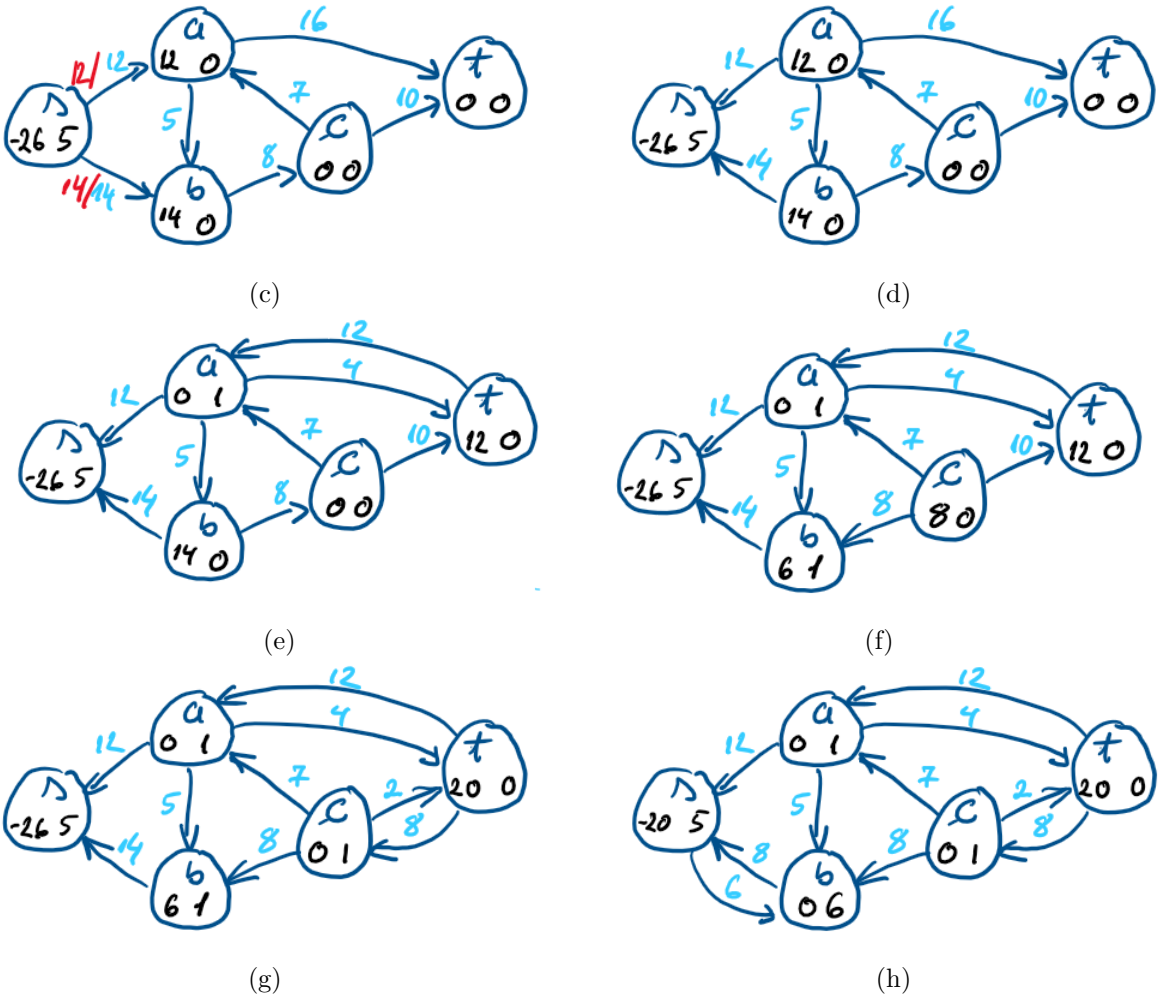


Figura 5: (continuare) Pompare preflux.

Figurile (5c)-(5h) prezintă pașii algoritmului de pompare preflux pentru a determina fluxul maxim în rețea. În figura (5c) se inițializează prefluxul. Figurile (5d)-(5h) prezintă graful rezidual și valoarea atributelor e și h după operații de înălțare și pompare. Astfel după inițializare se înalță vârful a și se pompează excesul de flux în vârful t, figura (5e). După acest pas se înalță vârful b și se pompează 8 unități de flux în vârful c, figura (5f). În următorii doi pași se înalță vârful c și excesul său de flux se pompează în vârful t, figura (5g). Singurul vârf din rețea diferit de sursă și destinație care mai are exces de flux este vârful b, în următorii pași asupra acestuia se aplică procedura de înălțare până când se poate pompa excesul de flux înapoi în vârful sursă nemaiexistând alt drum în afară de (b, s) pe care să se poată pompa excesul de flux. Înălțimea vârfului b, după acești pași, este $b.h = 6$ și excesul de flux $b.e = 0$, figura (5h). Algoritmul se oprește, nu mai există vârfuri asupra cărora să se aplice procedurile de pompare sau înălțare. Fluxul maxim în graf este $|f| = 20$.

Problema 3 Algoritmul de pompare topologică este o variantă a pompării prefluxului. Spre deosebire de algoritmul de pompare preflux, pomparea topologică impune o disciplină strictă de secvențiere a operațiilor elementare de pompare a prefluxului prin rețeaua de transport G și de înălțare a vârfurilor rețelei. Pașii algoritmului sunt descriși în curs. Mai jos este prezentat algoritmul de pompare topologică și procedura de descărcare a excesului de flux.

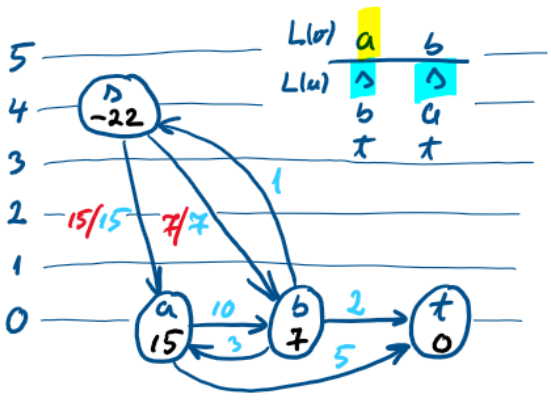
Algoritmul este o variantă a celui de pompare preflux. În algoritmul de pompare preflux, opera-

țiile de înălțare și pompare pot fi executate în orice ordine, inclusiv în ordinea impusă de regulile pompării topologice.

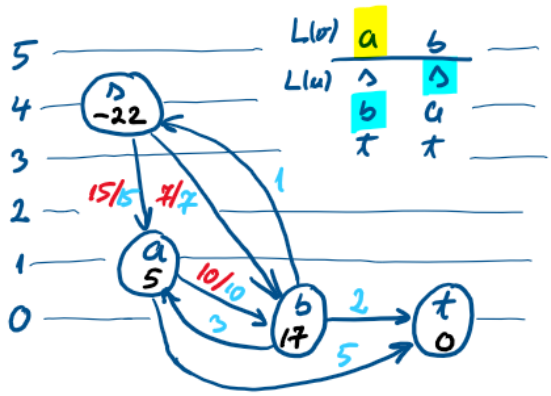
```
POMPARE_TOPOLOGICA(G,s,t)
1: INITIALIZARE_PREFLUX(G,s,t)
2: L = V \ {s,t}
3: for fiecare u ∈ V \ {s,t} do
4:     u.curent = u.N.head
5: u = L.head
6: while u ≠ NIL do
7:     înălțime_veche = u.h
8:     DESCARCARE(u)
9:     if u.h > înălțime_veche then
10:        mută u în capul listei L
11:    u.next
```

```
DESCARCARE(u)
1: while u.e > 0 do
2:     v = u.curent
3:     if v == NIL then
4:         INALTARE(u)
5:         u.curent = u.N.head
6:     else if c_f(u,v) > 0 ∧ u.h == v.h + 1 then
7:         POMPARE(u,v)
8:     else
9:         u.curent = u.urmatorul_vecin
```

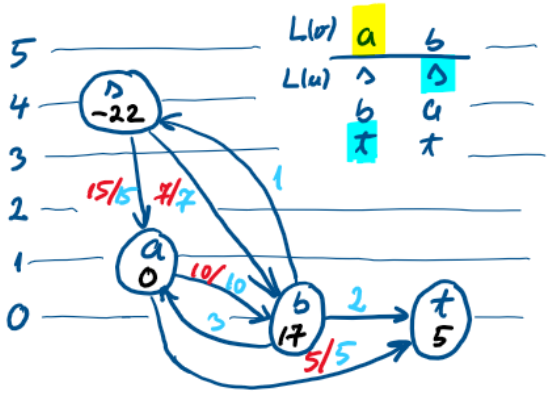
Pentru a ilustra funcționalitatea algoritmului de pompare topologică, numărul dintr-un vârf (colorat cu negru) reprezintă excesul de flux al acestuia, în dreapta rețelei de flux este prezentată lista $L(V)$ și listele $L(u)$ pentru $u \in V \setminus \{s,t\}$.



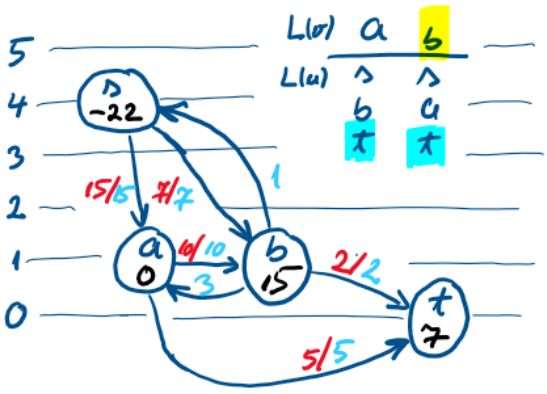
Pentru graful din cerință (figura (1c)), starea rețelei după inițializare este prezentată în figura de mai sus. $L(a) = (s\ b\ t)$, $L(b) = (s\ a\ t)$, $L(V) = (a\ b)$, arcele (s,a) și (s,b) au flux maxim și excesul vârfurilor a și b este $a.e = 15$, $b.e = 17$. În lista $L(V)$ cu galben este marcat vârful curent, în lista $L(u)$ cu albastru este marcat vârful curent.



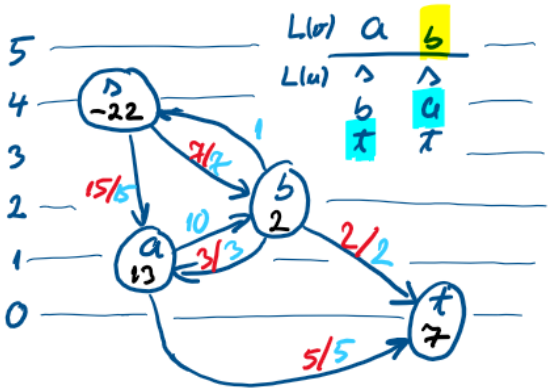
Este apelat algoritmul $DESCARCARE(a)$. Se parcurge lista $L(a)$ fără a se întâlni un arc admisibil (prin care să se poată pompa excesul de flux). În momentul când se parcurge toată lista $L(a)$ are loc operația $INALTARE(a)$. Reîncepe parcurgerea listei $L(a)$. Prin arcul (a, b) se transmit 10 unități de flux, $(a, b).f = 10$ (fluxul transmis printr-un arc este indicat în figură prin culoarea roșie, numerele colorate cu albastru reprezintă capacitatea arcelor). Supraîncărcarea vârfului a scade iar supraîncărcarea vârfului b crește



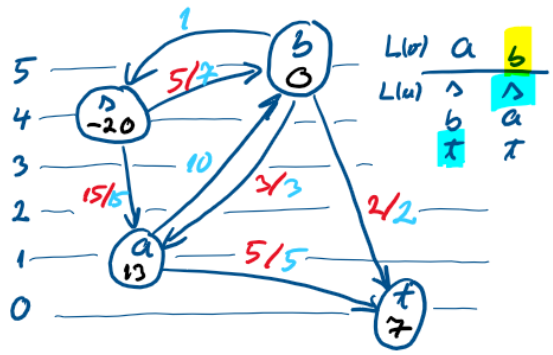
Continuă parcurgerea liste $L(a)$. Prin arcul (a, t) se transmit 5 unități de flux, $(a, b).f = 10$. Deoarece $a.e = 0$, operația $DESCARCARE(a)$ se termină. În ciclul central al algoritmului (linia 9) se constată că înălțimea actuală a lui a este mai mare decât înălțimea lui a când s-a întrat în algoritmul $DESCARCARE(a)$ și atunci vârful a este deplasat în vârful listei $L(V)$.



Se reia parcurgerea lui $L(V)$ cu succesorul vârfului a și se apelează $DESCARCARE(b)$. Începe parcurgerea listei $L(b)$ fără a descoperi un arc prin care să se poată pompa excesul de flux din b. Când se ajunge la capătul listei $L(b)$ are loc $INALTARE(b)$ și $b.h = 1$. Se reîncepe parcurgerea listei $L(b)$ și se găsește arcul (b, t) ca fiind admisibil (se poate pompa excesul de flux). Are loc pomparea fluxului $(b, t).f = 2$.

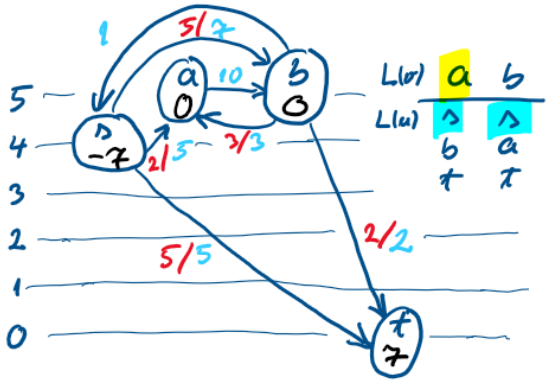


Este continuată parcurgerea listei $L(b)$, pentru că se parcurge toată lista fără să se mai găsească un arc admisibil are loc operația $INALTARE(b)$. Lista $L(b)$ este parcursă de la vârf și arcul (b, a) este găsit admisibil: $c_f(b, a) = 13 \wedge b.h = a.h + 1$. Pe arc se pompează 13 unități de flux. Ca efect, fluxul $(a, b), f$ este **anulat** și există flux $(b, a).f = 3$.



Pentru că $b.e > 0$ se continuă parcurgerea listei $L(b)$. Toate arcele $(b, x), x \in L(b)$ nu sunt admisibile (nu se mai poate pompa excesul de flux din b în alt vârf). Astfel se ajunge la finalul listei $L(b)$ și are loc $INALTARE(b)$. Pasul de parcurgere al listei $L(b)$ și $INALTARE(b)$ se repetă până când $b.h = 5$, până când înălțimea vârfului b depășește înălțimea sursei s.

Atunci când $b.h = 5$ se reia parcurgerea listei $L(b)$ de la vârf. Arcul (b, s) este admisibil: $c_f(b, s) = 8 \wedge b.h = s.h + 1$. Pe arc se pompează 2 unități de flux. Ca efect $(s, b).f = 5$. Deoarece $b.e = 0$, operația $DESCARCARE(b)$ se termină iar b este mutat în vârful listei $L(V)$.



Parcurgerea listei $L(V)$ se reia cu vârful a. Se execută $DESCARCARE(a)$ în ciclul central al algoritmului și, implicit, reîncepe parcurgerea listei $L(a)$ din punctul în care a fost lăsată la ultima revenire din $DESCARCARE(a)$. Toate arcele $(a, x), x \in L(a)$ nu sunt admisibile (nu se mai poate pompa excesul de flux din a în alt vârf), vârful a este înălțat până când $a.h = 5$.

Se reia parcurgerea listei $L(a)$ de la vârf. Arcul (a, s) este admisibil: $c_f(a, s) = 15 \wedge a.h = s.h+1$. Pe arc se pompează 13 unități de flux. Ca efect, fluxul de pe arcul (s, a) devine $(s, a).f = 2$. Deoarece $a.e = 0$, operația $DESCARCARE(a)$ se termină iar a este mutat în vârful listei $L(V)$.

Parcurgerea listei $L(V)$ continuă cu vârful b. Operația $DESCARCARE(b)$ se termină imediat și nu are nici un efect, pentru că $b.e = 0$. În final, parcurgerea listei $L(V)$ se termină. Se observă că toate vârfurile din $v \setminus \{s, t\}$ au excesul de flux $a.e = b.e = 0$. Fluxul maxim determinat în rețea este $|f| = 7$.

Soluția obținută nu este singura posibilă. Dacă ordinea vârfurilor în lista $L(V)$ sau în listele $L(u)$ diferă atunci soluția obținută poate fi alta, fluxul maxim va avea tot valoarea 7.

Algoritmul poate lucra direct pe rețeaua de flux G, nu este necesară menținerea explicită a rețelei reziduale.

Suport seminar algoritmica grafurilor
VI. Grafuri planare, colorarea grafurilor

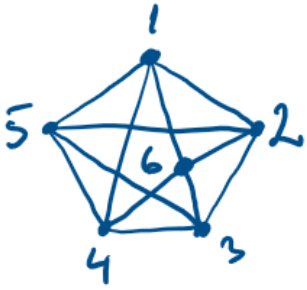
Probleme:

1. Să se verifice formula lui Euler pe următorul graf:

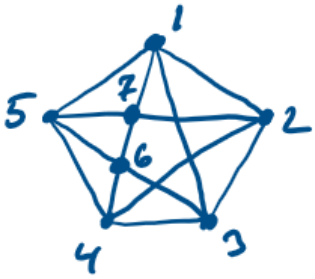
2. Fie mulțimea vârfurilor $V = \{1, 2, \ldots, n\}$, în $G = (V, E)$ există o muchie între i și j dacă i este divizibil cu j sau j este divizibil cu i. Pentru $n \geq 16$ este G planar?

3. Fie $G = (V, E)$ un graf neorientat și neponderat. Care este numărul minim de vârfuri astfel încât G să fie 4-regular simplu și planar?

4. Sunt următoarele grafuri planare?

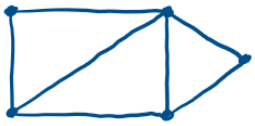


(a)



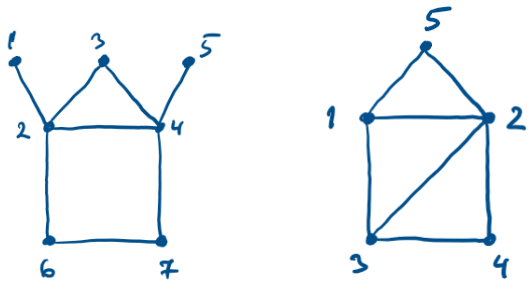
(b)

5. Să se determine dualul grafului de mai jos.



6. Fie $G = (V, E)$ un graf r-regular cu n noduri să se demonstreze că $\chi(G) + \chi(\bar{G}) \leq n + 1$, unde $\bar{G}$ este complementarul grafului G.

7. Să se determine polinomul cromatic și numărul cromatic pentru următoarele grafuri.



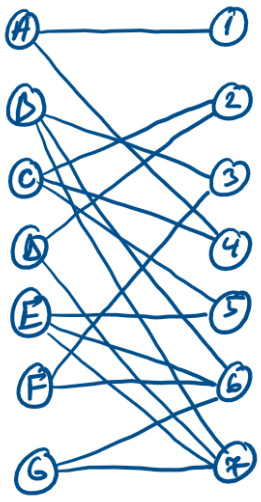
(a) (b)

8. Într-un graf cu $n = 128$ de vârfuri există muchie între vârfurile i și j dacă $cmmdc(i, j) \neq 1$. Care este numărul cromatic pentru un astfel de graf?

9. Să se deseneze un graf 3−regular al cărui număr cromatic de muchie este mai mare sau egal cu 4.

10. Firma Transport S.R.L. are 4 camioane în următoarele orașe: Carei, Cluj, Oradea, Brașov. Firma a primit simultan 4 comenzi, toate cele 4 camioanele trebuie să se prezinte în următoarele orașe: Satu-Mare, Timișoara, București, Iași. Distanțele dintre orașe se cunosc. Să se determine cu metoda maghiară destinația fiecărui camion astfel încât un camion să parcurgă distanța minimă.

11. Să se determine cuplajul maxim pentru graful de mai jos cu algoritmul Hopcroft-Karp.

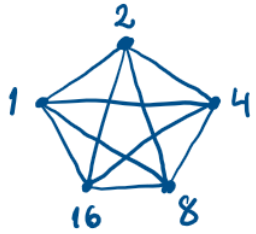


Soluții

Problema 1

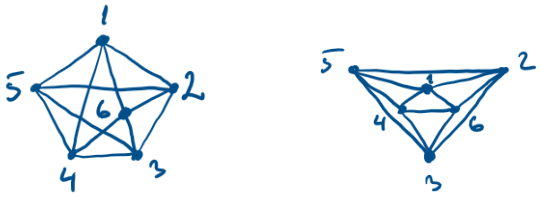
Formula lui Euler: $n - m + r = 2$, pentru graful respectiv $n = 4$, $m = 6$, $r = 4$.

Problema 2 Graful nu este planar deoarece se poate găsi ca și subgraf graful complet K_5 format din vârfurile $1, 2, 4, 8, 16$ (figura de mai jos).



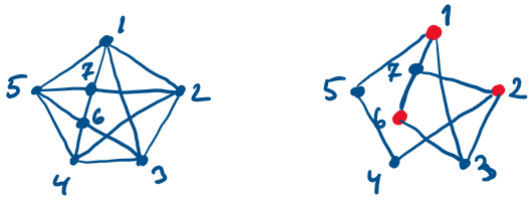
Problema 3 Dacă graful este 4-regular și simplu → are minim 5 vârfuri, K_5 este 4-regular, dar nu este simplu, de aceea trebuie să aibă minim 6 vârfuri. Desenați un astfel de graf.

Problema 4 Graful din figura (4a) este planar (figura 3) iar graful din figura (4b) nu este planar deoarece se poate găsi graful k_3 (figura 4).



(a) (b)

Figura 3: Graful din cerinţa (4a) şi reprezentarea planară a acestuia.



(a) (b)

Figura 4: Graful din cerinţa (4b) şi subgraful K_3.

Problema 5 Dacă pentru un graf se poate găsi dualul acestuia atunci graful este planar. Un graf dual se determină în felul următor:

1. fiecare regiune din graful inițial reprezintă un vârf în graful dual;
2. fiecare muchie din graful inițial trebuie traversată de o muchie din graful dual.

Figura 5 prezintă graful inițial și dualul acestuia.

Problema 6 Știm că $\chi(G) \leq \Delta(G) + 1$, deoarece graful G este $r-$regular $\rightarrow \Delta(G) = r$.
Pentru garful complementar $\Delta(\bar{G}) = n - r - 1$.

(a) (b)

Figura 5: Graful din cerinţa (5) şi dualul acestuia (graful dual este colorat în roşu).

Atunci:

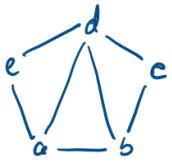
$$\frac{\begin{array}{r}(\chi(G) \leq r+1)\\ +\left(\chi(\bar{G}) \leq n-r-1+1\right)\end{array}}{\chi(G)+\chi(\bar{G}) \leq n+1}$$

Problema 7 Se aplică pașii descriși în cursul 10 pentru determinarea polinomului cromatic.

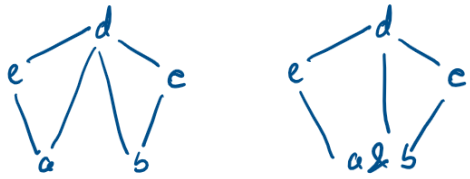
Calculul polinomului cromatic Pentru a determina polinomul cromatic al unui graf $G = (V, E)$ neorientat există două metode recursive:

1. $c_G(k) = c_{G-e}(k) - c_{G/e}(k)$: se reduce G eliminând pe rând câte o muchie $e \in E$ până când se obțin grafurile E_n, T_n sau K_n;
2. $c_G(k) = c_{\bar{G}}(k) - c_{\bar{G}/e}(k)$: se extinde G adăugând pe rând muchii e care lipsesc din G, $\bar{G} = G + e$.

Fie graful de mai jos, se utilizează ambele variante pentru a determina polinomul cromatic al grafului.

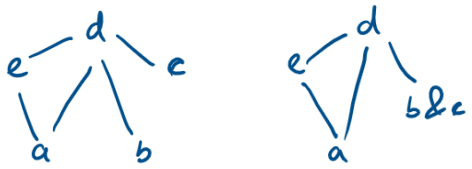


Metoda 1 Graful se descompune în $c_G(k) = c_{G_1}(k) - c_{G_2}(k)$



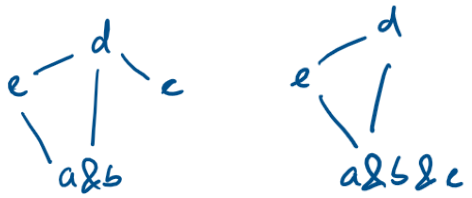
(a) G_1 (b) G_2

Primul termen c_{G_1} se descompune în $c_{G_1}(k) = c_{G_{11}}(k) - c_{G_{12}}(k)$, unde $G_{11} = G_1 - (b/c)$ și $G_{12} = G_1/(b, c)$.



(a) G_{11} (b) G_{12}

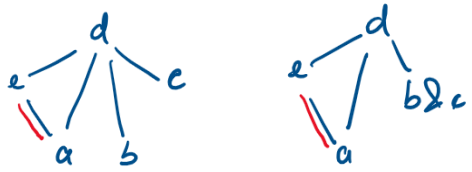
Termenul c_{G_2} se descompune în $c_{G_2} = c_{G_{21}}(k) - c_{G_{22}}(k)$, unde $G_{21} = G_2 - (a\&b, c)$ și $G_{22} = G_2/(a\&bb, c)$.



(a) G_{21} (b) G_{22}

Grafurile G_{12} și G_{21} sunt izomorfe, graful G_{22} este un graf complet K_3, $c_{K_3}(k) = k(k-1)(k-2)$).Până acum

$$c_G(k) = c_{G_{11}}(k) - 2c_{G_{12}}(k) + c_{K_3}(k).$$

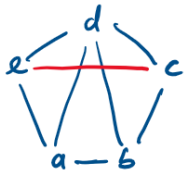


(a) G_{11} (b) G_{12}

Se observă că

$$c_{G_{11}}(k) = c_{T_5}(k) - c_{T_4}(k) = k(k-1)^4 - k(k-1)^3,$$
$$c_{G_{12}}(k) = c_{T_4}(k) - c_{T_3}(k) = k(k-1)^3 - k(k-1)^2.$$
$$\Rightarrow c_G(k) = k(k-1)^4 - k(k-1)^3 - 2[k(k-1)^3 - k(k-1)^2] + k(k-1)(k-2)$$
$$= k^5 - 7k^4 + 18k^3 - 20k^2 + 8k$$

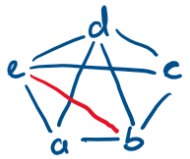
Metoda 2 Graful se descompune în $c_G(k) = c_{G_1}(k) + c_{G_2}(k)$, unde $G_1 = G + (c, e)$ și $G_2 = G/(c, e)$.



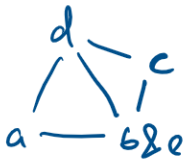
(a) G_1

(b) G_2

$G_2 = K_4 \Rightarrow c_{G_2}(k) = k(k-1)(k-2)(k-3)$, $C_{G_1}(k) = c_{G_{11}}(k) + c_{G_{12}}(k)$, unde



(a) G_{11}



(b) G_{12}

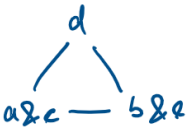
$C_{G_{11}}(k) = c_{G_{111}}(k) + c_{G_{112}}(k) = c_{K_5}(k) + c_{K_4}(k)$, unde

(a) G_{111}

(b) G_{112}

$C_{G_{12}}(k) = c_{G_{121}}(k) + c_{G_{122}}(k) = c_{K_4}(k) + c_{K_3}(k)$, unde

(a) G_{121}



(b) G_{122}

$$\begin{aligned} \Rightarrow c_G(k) &= c_{G_1}(k) + c_{G_2}(k) \\ &= \ldots \\ &= c_{K_5}(k) + 3c_{K_4}(k) + c_{K_3}(k) \\ &= k^5 - 7k^4 + 18k^3 - 20k^2 + 8k \end{aligned}$$

Problema 8 Numerele pare $2, 4, \ldots, 128$ formează un graf complet $\rightarrow \chi(G) \geq 64$.

Trebuie să se arate că ajung 64 de culori: vârfurile impare se pot colora cu culoarea lui i+1, de exemplu vârful 1 cu culoarea vârfului 2, vârful 3 cu culoarea vârfului 4, ș.a.m.d.

Problema 9 Colorare de muchii: un vârf din graf nu leagă două muchii de aceeași culoare. Figura 14 prezintă un astfel de graf.

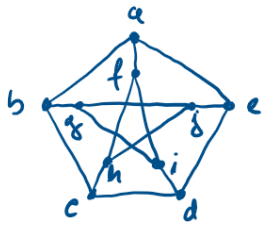


Figura 14: Soluţia pentru problema 8.

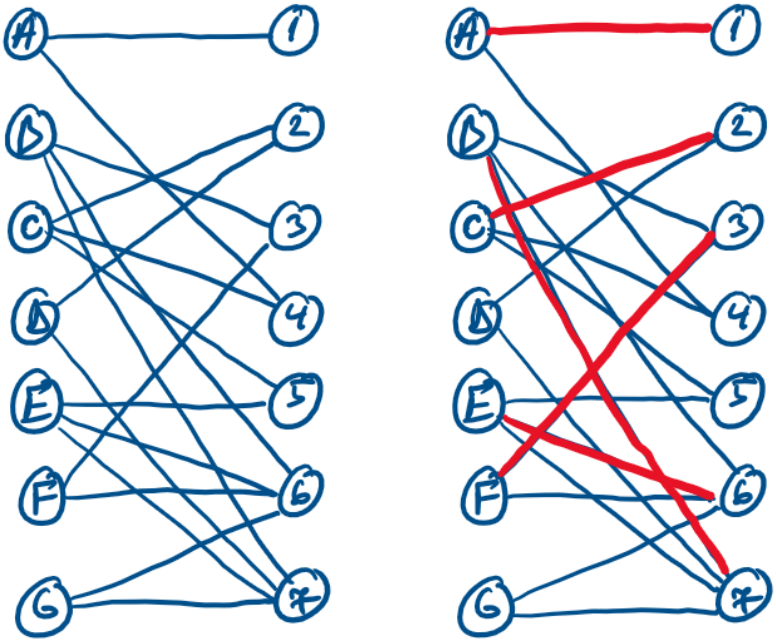
Problema 11 Algoritmul $Hopcroft - Karp(G)$ determină un cuplaj maxim. Algoritmul este prezentat mai jos.

Hopcroft_Karp(G)

1: $M = \varnothing$ (cuplaj maxim nul)
2: **while** există lanț de creștere în G **do**
3: folosește BFS pentru a construi un graf alternant cu rădăcina în vârf nesaturat
4: îmbunătățește cuplajul M cu DFS
5: **return** M

Figura 15 prezintă graful inițial și inițializarea grafului (găsirea unui cuplaj în graf - figura 15b). Cuplajul inițial se găsește după primul pas BFS. După primul pas, vârfurile $D, G, 4$ și 5 nu fac parte din cuplaj.

În pașii următori se rulează BFS pe graful din figura 16 și se obțin lanțurile de creștere $4-C-2-D$ și $5-E-6-G$. După acești doi pași, cuplajul maxim este format din $(A, 1), (C, 4), (D, 2), (E, 5)$ și $(G, 6)$.



(a) (b)

Figura 15: Graful din cerinţă şi iniţializarea unui cuplaj în graf.

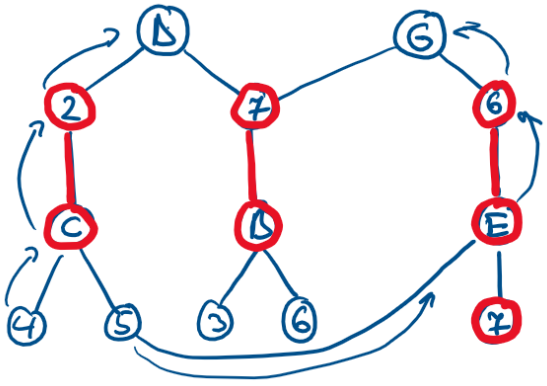


Figura 16