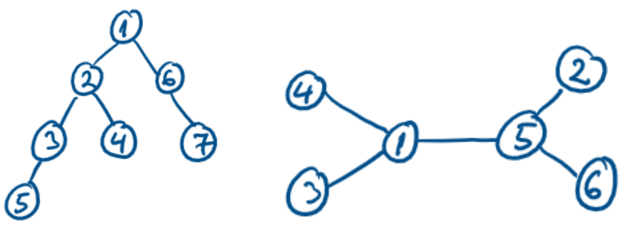
Suport seminar algoritmica grafurilor
III. Arbori

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



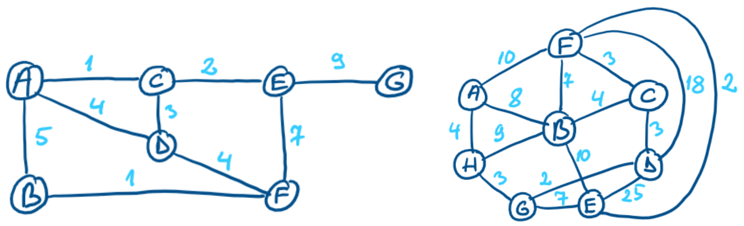
(a) (b)

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

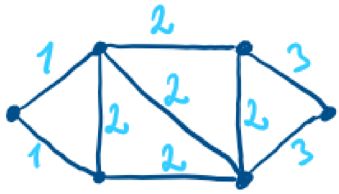


Figura 3

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:

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

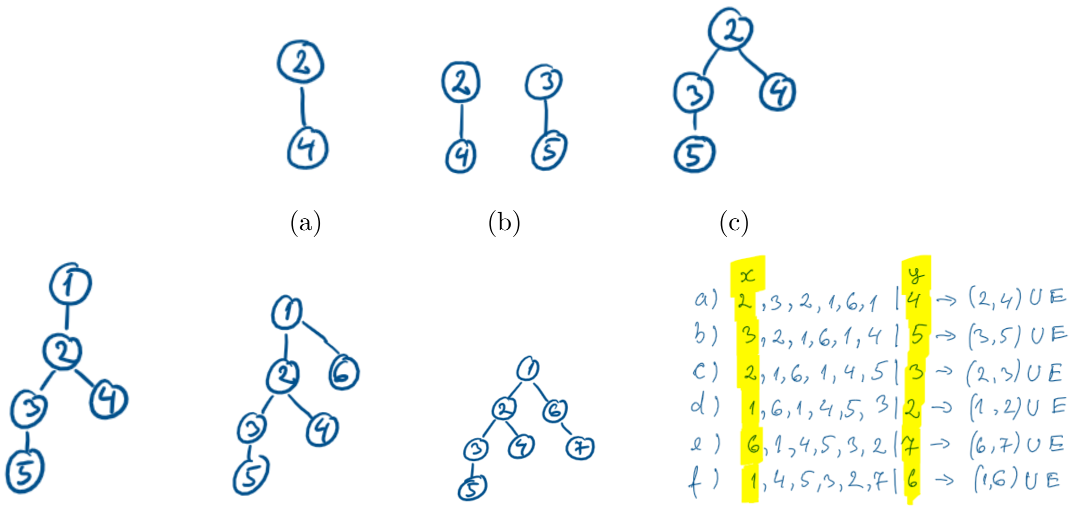
CODARE_PRUFER(F)
$K = \varnothing$
while T conține și alte vârfuri decât rădăcina **do**
  fie v frunza minimă din T
  $K \leftarrow$ predecesor(v)

$T = T \setminus \{v\}$
return K

Problema 3

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



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

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)

```
n = |C|
Q = C
for 1 ≤ i ≤ n − 1 do
    alocă un nou vârf z
    z.stang = x =EXTRACT_MIN(Q)
    z.drept = y =EXTRACT_MIN(Q)
    z.fr = x.fr + y.fr
    INSERT(Q, z)
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

$$\begin{array}{lll} a = 0 & c = 100 & e = 1101 \\ b = 101 & d = 111 & f = 1100 \end{array}$$

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

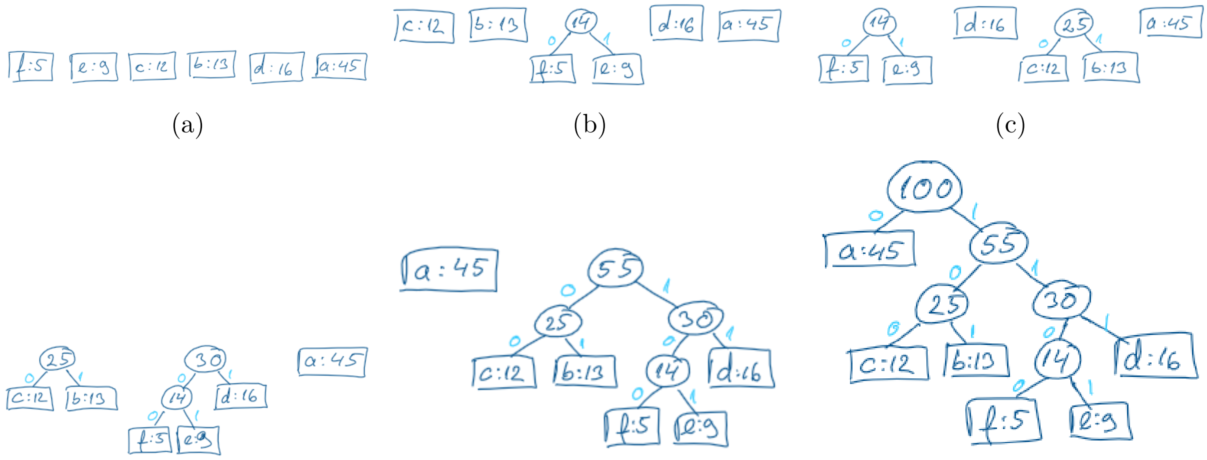
$$aaaaabbbbcccddef$$

codificată Huffman este

$$0000010110110110110010010011111111011100$$

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)

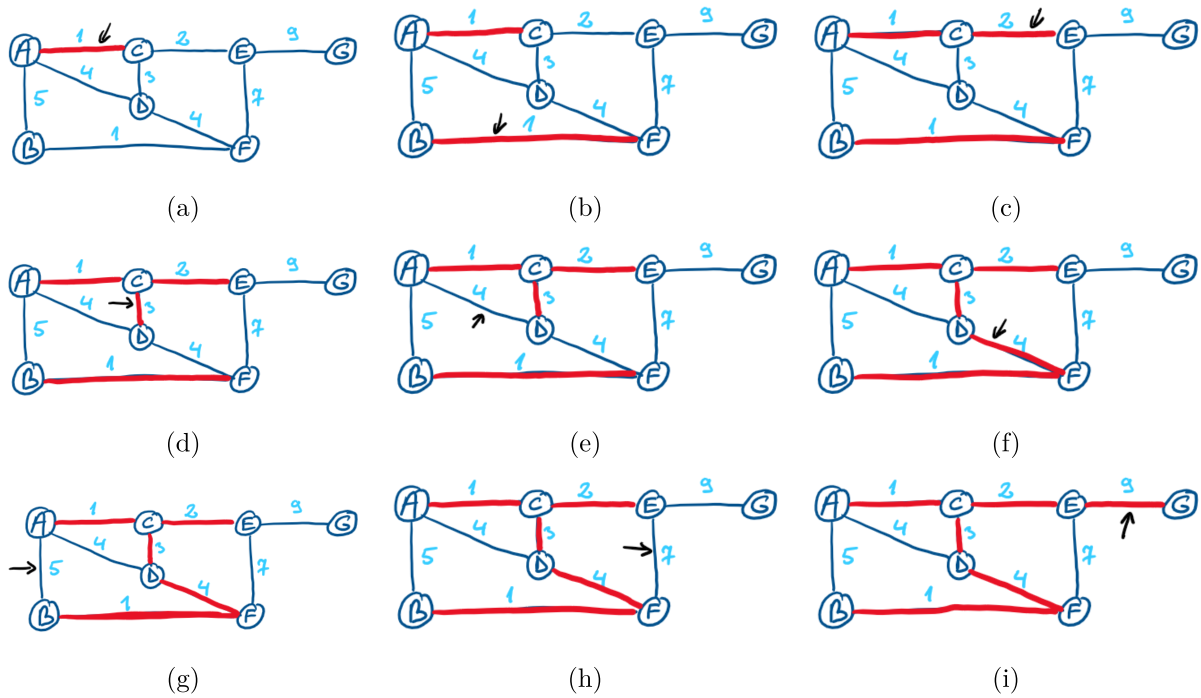
Figura 5: Paşii urmaţi de algoritmul Huffman pentru a codifica caracterele din setul C.

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)

```
A = ∅
for v ∈ V do
    make_set(v)
sortare muchii crescător după ponderea w
for (u, v) ∈ E luate crescător după w do
    if find_set(u) ≠ find_set(v) then
        A = A ∪ (u, v)
        union(u,v)
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.



(a) (b) (c) (d) (e) (f) (g) (h) (i)

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.

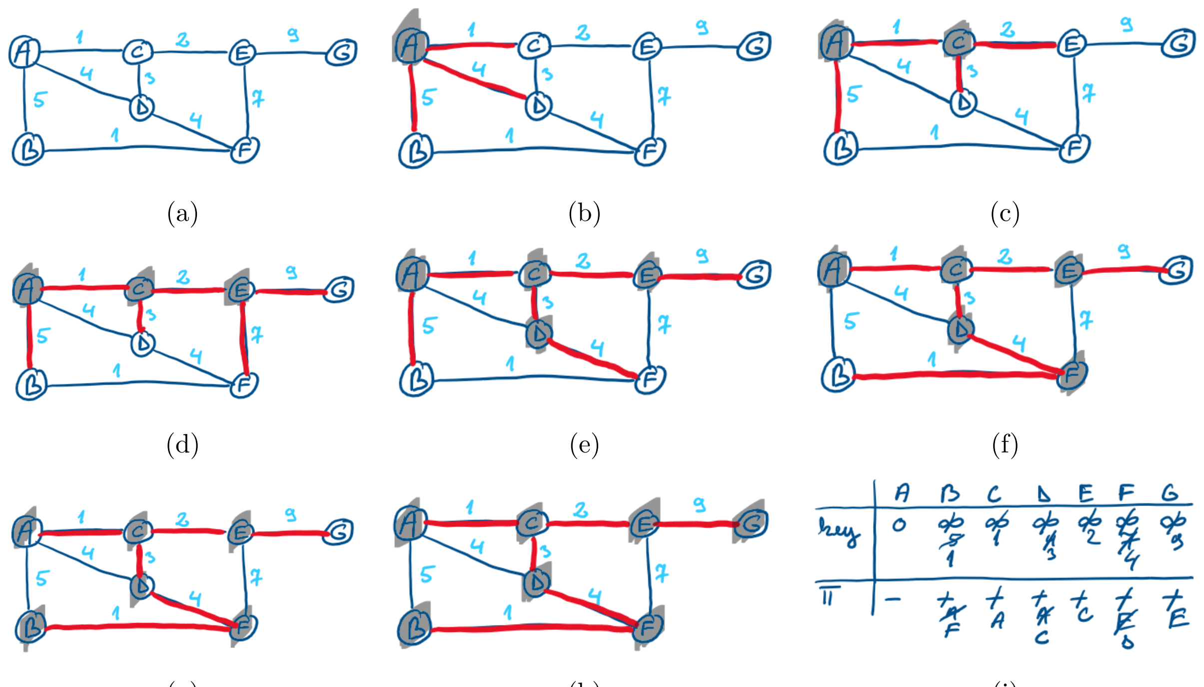
Algoritmul lui Prim găsește arborele minim de acoperire pentru $G = (V, E)$ în mod similar cu algoritmul lui Dijkstra pentru drumul de costul minim. Muchiile din setul A formează un singur arbore.

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)

for $u \in V$ **do**
    $u.key = \infty$
    $u.\pi = NIL$
$r.key = 0$
$Q = V$
while $Q \neq \varnothing$ **do**
    u = extract_min(Q)
    for $v \in Adj[u]$ **do**
        if $v \in Q$ și $w(u, v) < v.key$ **then**
            $v.\pi = u$
            $v.key = w(u, v)$

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.



(a) (b) (c) (d) (e) (f) (g) (h) (i)

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).

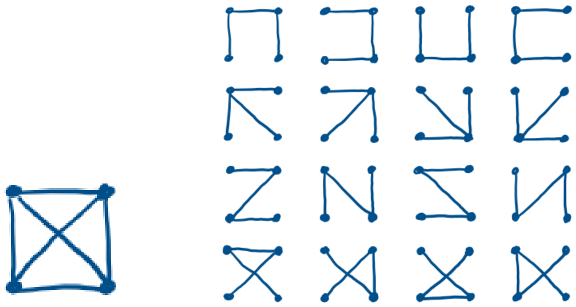
Astfel formula lui Cayley este:

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

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.

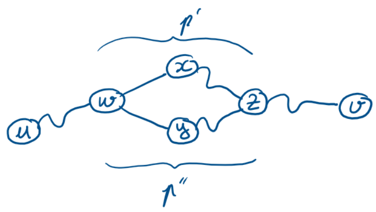


Figura 9

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. $\square$