Probleme de drum în (di)grafuri.

Parcurgeri sistematice ale (di)grafurilor.

Dat G = (V, E) un (di)graf cu mulțimea vârfurilor V = {1, . . ., n} şi s ∈ V, se cere să se genereze "eficient" mulțimea S a vârfurilor accesibile din s pe drumuri în G.

Soluția pe care o vom da va impune un proces de "vizitare" succesivă a vecinilor lui ***s*** în ***G***, apoi a vecinilor (încă nevizitați) ai acestora, ş.a.m.d., până se va construi mulțimea ***S***.

Algoritmul va implementa o metodă sistematică de "vizitare", utilă în probleme de optimizare pe grafuri, dar mai ales în problemele în care grafurile sunt folosite ca structuri (explicite sau implicite) de date.

Vom presupune că (di)graful ***G*** este dat prin intermediul listelor de adiacență (în cazul grafurilor se poate considera că fiecare muchie generează o pereche de arce simetrice şi reprezentarea folosită la sortarea topologică rămâne valabilă). În algoritm vom folosi două mulțimi:

S = mulțimea vârfurilor (deja) vizitate;

$\overline{S} \subseteq S$ astfel încât $\forall v \in S - \overline{S}$ $vw \in E \Rightarrow w \in S$ (complementara în ***S*** a lui $\overline{S}$ conține vârfuri ai căror vecini au fost deja vizitați).

Algoritmul se va termina atunci când $\overline{S} = \varnothing$.

Inițial, $S = \overline{S} = \{s\}$. În pasul curent al algoritmului, se alege un vârf din $\overline{S}$ şi se va "explora", considerând "următorul" vecin din lista sa de adiacență (încă neexaminat). Dacă lista sa de adiacență a fost parcursă, atunci se poate "scoate" acest vârf din $\overline{S}$. Dacă există acel următor vecin şi n-a fost încă vizitat, el se adaugă la ***S*** şi la $\overline{S}$. "Poziția" următorului vecin neexaminat din lista de adiacență a unui vârf ***i*** se va gestiona cu ajutorul unui pointer ***p[i]***, care inițial are valoarea ***cap[i]*** şi va fi avansat în timpul parcurgerii listei vecinilor lui ***i***.

Descriem această "vizitare" a vârfurilor (di)grafului în procedura nerecursivă ***exploraredin(s)***.

```
type    pointer = ^arc;
arc =   record
        vârf : 1 . . n;
        prec : pointer;
        end;
var cap, p : array[1 . . n] of pointer;

procedure exploraredin (s : 1 . . n);
  { înaintea apelului, se va inițializa p[i] := cap[i], i = 1, n şi S := Ø }
begin
        S := S ∪ {s};
        S̄ := {s};
```

```
while S̄ ≠ Ø do
    begin
        alege v ∈ S̄
        if p[v] = nil then
            şterge v din S̄
        else
            begin
                w := p[v]^.vârf;
                p[v] := p[v]^.prec;
                if w ∉ S then
                    begin
                        adaugă w la S;
                        adaugă w la S̄;
                    end;
            end;
    end;
end;
```

Presupunând că operaţiile subliniate necesită timpul de **O(1)**, atunci se observă că aplicarea lui ***exploraredin(s)*** va necesita timpul $\mathbf{O}(n_s + e_s)$ unde, $n_s = |\{ v \mid v$ accesibil din s pe un drum în $G \}|$ şi e_s = numărul arcelor (muchiilor) cu ambele extremităţi în mulţimea finală ***S***. Acest ordin de complexitate rezultă din faptul că la fiecare iteraţie a ciclului **while**, în timp constant, sau se adaugă un nou vârf la ***S*** ($\bar{S}$), sau se progresează în lista de adiacenţe a unui vârf din $\bar{S}$.

Pentru ca operaţiile subliniate să se execute în **O(1)** va trebui ca ***S*** să fie reprezentată cu ajutorul unui tablou boolean [***S*** : **array**[1 . . n] **of boolean**; iniţializarea lui ***S*** (pe fals) înaintea apelului lui ***exploraredin***, necesită **O(*n*)** operaţii; testul **if** $w \notin S$ este **if not** ***S*(*w*)**, deci **O(1)**; atribuirea $S := S \cup \{s\}$ este ***S*(*s*) := *true***; (adaugă ***w*** la ***S***) este ***S*(*w*) := true**] iar $\bar{S}$ ca o listă înlănţuită:

1. Dacă $\bar{S}$ va fi o ***stivă*** cu ***top* : *pointer*** capul ei, atunci:

$\bar{S}$ **:=** ***s*** $\rightarrow$ **new(*top*)**; ***top*^.*vârf* := *s***; ***top*^.*prec* := nil**.

$\bar{S} \neq \emptyset \rightarrow$ ***top*** $\neq$ **nil**.

alege ***v*** $\in \bar{S} \rightarrow$ ***v* := *top*^.*vârf*** (nu se modifică ***top***)

şterge ***v*** din $\bar{S} \rightarrow$ ***top* := *top*^.*prec***

adaugă ***w*** la $\bar{S} \rightarrow$ **new(*l*)**; ***l*^.*vârf* := *w***; ***l*^.*prec* := *top***; ***top* := *l***.

În acest caz , de fiecare dată, va fi ales pentru explorare ultimul vârf introdus în $\bar{S}$ şi parcurgerea se face "***mai întâi în adâncime***", ***depth first search*** (***dfs***).

2. Dacă $\bar{S}$ va fi o ***coadă*** cu ***cap, spate* : *pointer*** începutul şi sfârşitul ei, atunci:

$\bar{S}$ **:=** ***s***; $\rightarrow$ **new(*l*)**; ***l*^.*vârf* := *s***; ***l*^.*prec* := nil**; ***cap* := *l***; ***spate* := *l***;

$\bar{S} \neq \emptyset \rightarrow$ ***cap*** $\neq$ **nil**;

alege $v \in \overline{S} \rightarrow$ ***v := cap^.vârf*** (nu se modifică nici ***cap*** nici ***spate***).
şterge ***v*** din $\overline{S} \rightarrow$ ***cap := cap^.prec***; (**if *cap* = nil then *spate* := nil**)
adaugă ***w*** la $\overline{S} \rightarrow$ **new(*l*)**; ***l^.vârf := w***; ***spate^.prec := l***; ***spate := l***; (***l^.prec* := nil**)
În acest caz, de fiecare dată, va fi ales spre explorare cel mai "vechi" vârf introdus în $\overline{S}$ şi un vârf va fi scos din $\overline{S}$ abia după ce i se introduc toţi vecinii neexploraţi încă. Parcurgerea va fi deci "***mai întâi la lăţime***" ***breadth first search*** (***bfs***).

Implementarea lui $\overline{S}$ ca o stivă (deci ***dfs***) se poate face explicit, aşa cum am arătat mai sus, dar şi implicit scriind o formă recursivă a lui ***exploraredin(s)***. Vom face acest lucru în aplicaţia următoare, în care vom lista componentele conexe ale unui graf reprezentat cu ajutorul listelor de adiacenţă în **O(*n* + *e*)**, ***n*** = | ***V*** |, ***e*** = | ***E*** |.

```
var     cap, p : array[1 . . n] of pointer;
        i, nrcconexa : integer;
        l, topc : pointer;

procedure dfs(v : 1 . . n);
var     p1 : pointer;
        w : 1 . . n;
begin
        p1 := p[v];
        while p1 ≠ nil do
                begin
                        w := p1^.vârf;
                        p1 := p1^.prec;
                        if not S[w] then
                                begin
                                        new(l);
                                        l^.vârf := w;
                                        l^.prec := topc;
                                        topc := l;
                                        S[w] := true;
                                        dfs(w)
                                end;
                end;
end;

begin {program principal}
        for i := 1 to n do S[i] := false;
        for i := 1 to n do p[i] := cap[i];
        nrcconexa := 0;
        for i := 1 to n do
                if not S[i] then
                        begin
                                nrcconexa := nrcconexa + 1;
                                S[i] := true;
```

```
                    new(topc);
                    topc^.vârf := i;
                    topc^.prec := nil;
                    dfs(i); {determină componenta conexă la care ∈ i}
                    writeln('componenta conexă ', nrcconexa);
                    while topc ≠ nil do
                        begin
                            write(topc^.vârf);
                            topc := topc^.prec;
                        end;
                end;
end.
```

Vom face în continuare, o analiză a comportării logice a parcurgerii ***dfs*** a unui (di)graf, pentru a pune în evidență modul de accesare a vârfurilor, precum şi o partiție a mulțimii muchiilor (arcelor), relevante în aplicații (care necesită parcurgerea sistematică a (di)grafului în scopul calculării recursive a unor funcții definite pe mulțimea vârfurilor sale, în ordinul **O(*n* + *m*)**).

În aceleaşi ipoteze referitoare la tipurile de date şi variabilele folosite mai sus, considerăm modulul apelant:

```
begin
    for v := 1 to n do
        begin
            p[v] := cap[v];
            S[v] := false;
        end;
    nr1 := 0;
    nr2 := 0;
    {T := Ø ; F := Ø; C := Ø }
    for v := 1 to n do
        if not S[v] then
            begin
                nr1 := nr1 + 1;
                viz[v] := nr1;
                S[v] := true;
                dfs(v);
                nr2 := nr2 + 1;
                sfv[v] := nr2
            end;
end.
```

şi procedura dfs:

```
procedure dfs(v : V);
var   p1 : pointer;
```

```
        w : V;
begin
        p1 := p[v];
        while p1 ≠ nil do
                begin
                        w := p1^.vârf;
                        p1 := p1^.prec;
                        if not S[w] then
                                begin
                                        {T := T ∪ {vw};}
                                        nr1 := nr1 + 1;
                                        viz[w] := nr1;
                                        S[w] := true;
                                        dfs(w);
                                        nr2 := nr2 + 1;
                                        sfv[v] := nr2
                                end
                        else
                                begin {w a mai fost vizitat}
                                        if viz[w] > viz[v] then {F := F ∪ {vw}}
                                        else
                                                if ∃ drum de la w la v în (V, T) then
                                                        {B := B ∪ {vw}}
                                                else {C := C ∪ {vw}}
                                end
                end;
end;
```

S-au utilizat ***nr*1** şi ***nr*2** pentru construirea tablourilor ***viz*** şi ***sfv***, având semnificaţia:

- ***viz*[*v*]** = numărul de ordine al întâlnirii vârfului ***v*** de către procedura ***dfs*** (apelul ***dfs*(*v*)** este cel de-al ***viz*[*v*]**-lea, în traversare (di)grafului);
- ***svf*[*v*]** = numărul de ordine al sfârşitului vizitării vârfului ***v*** de către procedura ***dfs***.

De asemenea, s-au construit submulţimile ***T***, ***B***, ***F***, ***C*** ale lui ***E*** (în operaţiile subliniate) care au următoarele semnificaţii, în ipoteza că ***G*** este digraf:

- ***T*** – mulţimea arcelor utilizate de procedura ***dfs*** pentru parcurgerea digrafului; se verifică imediat că **(*V*, *T*)** este o reuniune de arborescenţe (***arborescenţele dfs***).
- ***F*** – mulţimea arcelor "forward", orientate de la o extremitate iniţială vizitată înaintea extremităţii finale;
- ***B*** – mulţimea arcelor "backward", orientate de la o extremitate iniţială vizitată după extremitatea finală şi extremitatea iniţială se află în subarborescenţa ***dfs*** cu rădăcina în extremitatea finală; mai precis, $vw \in B \Leftrightarrow vw \in E$, $viz[v] > viz[w]$ şi $sfv[v] < sfv[w]$.

- ***C*** – mulțimea arcelor "cruce", orientate de la o extremitate inițială, aflată într-o arborescență ***dfs*** construită după cea în care se află extremitatea finală; mai precis, $vw \in C \Leftrightarrow vw \in E$, $viz[v] < viz[w]$ şi $sfv[v] > sfv[w]$.

Se observă că ***B***, ***F***, sau ***C*** pot fi vide şi că $E = T \cup B \cup F \cup C$ şi toate aceste mulțimi sunt disjuncte. Dacă ***G*** este graf, atunci $B = F$ şi $C = \varnothing$, interpretările fiind evidente.

Vom exemplifica utilitatea parcurgerii sistematice a unui digraf, pentru a depista eficient componentele sale tari-conexe (modificări minore ale metodei pe care o vom descrie permit determinarea blocurilor (componentelor biconexe) ale unui graf).

Fie $G = (V, E)$ un digraf. Considerăm, pe mulțimea vârfurilor sale, relația binară: $v \rho w \Leftrightarrow$ ***există drum de la v la w şi drum de la w la v în digraful G***. Se verifică imediat că ρ este relație de echivalență. Clasele de echivalență în raport cu relația ρ se numesc ***componentele tari-conexe*** ale digrafului ***G*** (submulțimi maximale (în raport cu incluziunea) de vârfuri cu proprietatea că induc subdigrafuri tari-conexe).

Ideia algoritmului este următoarea:

- se efectuează o parcurgere ***dfs*** a digrafului, construind tablourile ***viz*** şi ***sfv***.
- Dacă ***C*** este o componentă tare conexă a lui ***G***, definim

$$f(C) = \max_{v \in C} sfv[v] .$$

Fie C_0 astfel încât $f(C_0) = \min_C f(C)$, deci C_0 este prima componentă tare conexă pentru care se termină complet vizitarea.

Dacă $f(C_0) = sfv[v_0]$, atunci, notând cu T_{v_0} subarborescența ***dfs*** cu rădăcina în v_0, avem $C_0 = V(T_{v_0})$. În adevăr, $C_0 \subseteq V(T_{v_0})$, întrucât în T_{v_0} avem vârfurile accesibile prin drumuri în ***G*** (şi orice vârf al lui C_0 are această proprietate). Dacă $V(T_{v_0}) - C_0 \neq \varnothing$, se contrazice alegerea lui C_0, întrucât $\forall\, x \in V(T_{v_0}) - C_0$ satisface $sfv[x] < sfv[v_0]$ (***x*** este vizitat de un apel recursiv generat de $dfs(v_0)$).

Rezultă că la terminarea vizitării vârfului v_0 s-a depistat C_0.

- Recunoaşterea vârfului v_0 este simplă: v_0 este primul vârf al digrafului cu proprietatea că din nici unul din descendenții săi ***dfs*** nu există arce de întoarcere (***B***) cu extremitatea finală ***w*** având $viz[w] < viz[v_0]$.
- Pentru $G - C_0$ se repetă acelaşi raționament.

Dacă vom considera $b[v]$ = cel mai mic $viz[w]$ al unui arc $uw \in B$ cu $u \in T_v$, dacă există un astfel de arc, altfel $b[v] = viz[v]$, atunci se observa ca întreg planul de lucru descris mai sus se realizează la o singura traversare a digrafului , întrucât $b[v]$ se pot calcula recursiv, iar considerarea subdigrafului $G - C_0$ înseamnă simpla continuare a traversării.

Detaliile sunt descrise mai jos (modulul apelant şi procedura ***dfs*** adaptata problemei):

```
begin
    for v:=1 to n do
        begin
            p[v] = cap[v];
            S[v] := false
        end;
    nr:=0;
```

```
        L:= ∅; {L= mulţimea vârfurilor curente vizitate şi neincluse încă într-o componenta tare conexa}
        for v:=1 to n do
                if not S[v] then
                        begin
                                nr := nr+1;
                                viz[v] := nr;
                                S[v] :=true;
                                adaugă v la L;
                                dfs(v);
                        end
end.

procedure dfs(v : V);
var p1 : pointer; w : V;
begin
        p1 :=p[v];
        b[v] := viz[v];
        while p1 ≠ nil do
                begin
                        w := p1 ↑.virf;
                        p1 := p1 ↑.prec;
                        if not S[w] then
                                begin
                                        nr := nr +1;
                                        viz[w]:= nr;
                                        S[w] := true;
                                        adaugă w la L;
                                        dfs(w);
                                        if b[w]<b[v] then
                                                b[v]:= b[w]
                                end
                        else
                                if  w ∈ L and b[v] > viz[w] then
                                        b[v] :=viz[w]
                end;
        if b[v] = viz[v] then
                extrage din L c.t.c. cu rădăcina v formata din vârfurile w din L cu viz[w] ≥ viz[v].
end.
```

Pentru realizarea eficienta a operaţiilor subliniate, se va implementa ***L*** ca o pereche formata dintr-un tablou boolean (pentru testarea apartenenţei în ***O*(1)**) şi ca o lista dublu înlănţuită (pentru extragerea eficienta a vârfurilor unei componente tare conexe; notam ca la inserţia în ***L***, elementele vin în ordinea crescătoare a lui *viz*, iar extragerea se face de la celalalt capăt). Se verifica imediat ca în acest fel întreg algoritmul are complexitatea ***O*(|*V*| + |*E*|)**.

Probleme de drum minim.

Fie $G = (V, E)$ un digraf cu mulțimea vârfurilor $V = \{1,\ldots,n\}$. Consideram dată o funcție $a : E \to \mathbf{R}$ cu interpretarea : $\forall\ ij \in E$ $a(ij)$ = costul arcului ij (ponderea, lungimea, etc.). Dacă digraful G este reprezentat cu ajutorul listelor de adiacenta, atunci costul arcului ij este un câmp în articolul ce reprezintă acest arc în lista de adiacenta a lui i. Pentru comoditatea notațiilor vom folosi reprezentarea digrafului G cu ajutorul matricii de cost-adiacenta $A = (a_{ij})_{n \times n}$ cu

$$a_{ij} = \begin{cases} a(ij) & \text{daca } ij \in E \\ \infty & \text{altfel} \end{cases}$$

Notam ca ∞ desemnează un număr real mare în raport cu celelalte costuri (de exemplu $\infty > n \times \max_{ij \in E} a(ij)$) şi vom presupune în plus ca $\infty \pm a = \infty$, $\infty + \infty = \infty$.
(Este posibil, de asemenea , ca ∞ sa semnifice acces terminat cu insucces în structura de date în care se reprezintă matricea A).

Dacă $i, j \in V$, vom nota cu $\mathbf{D}_{ij} = \{ D_{ij} \mid D_{ij}$ drum în G de la i la $j\}$. Pentru $D_{ij} \in \mathbf{D}_{ij}$

$$D_{ij} : (i =) v_0, v_0v_1, v_1, \ldots, v_{r-1}, v_{r-1}v_r, v_r (= j)$$

mulțimea vârfurilor este $V(D_{ij}) = \{v_0, v_1, \ldots, v_r\}$
şi mulțimea arcelor $E(D_{ij}) = \{v_0v_1, v_1v_2, \ldots, v_{r-1}v_r\}$.

Orice vârf $k \neq i, j$ al lui D_{ij}, determina pe D_{ij} doua drumuri $D_{ik} \in D_{ik}$ şi $D_{kj} \in D_{kj}$. Vom nota $D_{ij} = D_{ik} \circ D_{kj}$.

Costul unui drum $D_{ij} \in \mathbf{D}_{ij}$ se defineşte

$$a(D_{ij}) = 0 + \sum_{lk \in E(D_{ij})} a_{lk}.$$

În particular, $a(D_{ii}) = 0$. Principalele probleme de drum (de cost) minim care apar în aplicații practice (sau sunt utile în rezolvarea altor probleme de optimizare combinatorie) sunt:

(**P1**) Date: G digraf, $a : E(G) \to \mathbf{R}$, $s, t \in V(G)$ $s \neq t$.

Să se determine $D^*_{st} \in \mathbf{D}_{st}$ astfel încât

$$a(D^*_{st}) = \min\{a(D_{st}) \mid D_{st} \in \mathbf{D}_{st}\}$$

(**P2**) Date: G digraf, $a : E(G) \to \mathbf{R}$, $s \in V(G)$.

Să se determine $D^*_{si} \in \mathbf{D}_{si}\ \forall\ i \in V(G)$, astfel încât

$$a(D^*_{st}) = \min\{a(D_{si}) \mid D_{si} \in \mathbf{D}_{si}\}\ i \in V(G)$$

(**P3**) Date: G digraf, $a : E(G) \to \mathbf{R}$.

Să se determine $D^*_{ij} \in \mathbf{D}_{ij}\ \forall\ i, j \in V(G)$, astfel încât

$$a(D^*_{ij}) = \min\{a(D_{ij}) \mid D_{ij} \in \mathbf{D}_{ij}\}.$$

Observații: 1. Cu convenția folosita în reprezentarea matricilor de cost adiacenta, se poate considera ca $\mathbf{D}_{ij} \neq \varnothing\ \forall\ i, j \in V$. Dacă $a(D_{ij}) < \infty$ atunci D_{ij}, este drum în G de la i la j iar dacă $a(D_{ij}) = \infty$, atunci D_{ij} este drum în digraful complet simetric obținut din G prin adăugarea arcelor lipsa, cu ponderea ∞. Rezulta ca toate mulțimile, pe care se considera

minimele în problemele precedente, sunt nevide şi, cum digrafurile considerate sunt finite, rezulta ca aceste mulţimi sunt finite (în fiecare drum vârfurile sunt distincte), deci minimele considerate există.

2. Algoritmii de rezolvare a problemei (**P1**) se obţin din algoritmii de rezolvare a problemei (**P2**) adăugânduli-se un test suplimentar (evident) de oprire. Problema (**P3**) se poate rezolva iterând un algoritm de rezolvare a problemei (**P2**). Sunt posibile insa soluţii mai eficiente.

Ne vom ocupa în continuare de rezolvarea problemei (**P2**).

Teorema 1. Fie $G = (V, E)$ digraf, $V = \{1, \ldots, n\}$, $s \in V$ şi $a : E \to \mathbf{R}$, astfel încât

(I) $\forall\, C$ circuit în G, $a(C) > 0$.

Atunci $(u_1, \ldots, u_n)$ este o soluţia sistemului

(*)
$$\begin{cases} u_s = 0 \\ u_i = \min\limits_{j \neq i}(u_j + a_{ji}) \ \forall i \neq s \end{cases}$$

dacă şi numai dacă $\forall\, i \in V$, $\exists\, D^*_{si} \in \mathbf{D}_{si}$ astfel încât $a(D^*_{si}) = u_i$ şi $a(D^*_{si}) = \min\{a(D) \mid D \in \mathbf{D}_{si}\}$.

Demonstraţie : "$\Leftarrow$" Fie D^*_{si} ($i \in V$) soluţii ale problemei (**P2**) cu $a(D^*_{si}) = \min\{a(D) \mid D \in \mathbf{D}_{si}\}$. Notam cu $u_i = a(D^*_{si})$ ($i \in V$).

Să observam ca ipoteza **(I)** asigura faptul ca $u_s = 0$; Pentru $i \neq s$ drumul D^*_{si} are un penultim vârf j. Dacă D_{sj} este drumul de la s la j determinat pe D^*_{si} de vârful j, avem: $u_i = a(D^*_{si}) = a(D_{sj}) + a_{ji} \geq a(D^*_{sj}) + a_{ji} = u_j + a_{ji}$.

Presupunem ca $u_i > u_j + a_{ji}$, adică $a(D_{sj}) > a(D^*_{sj})$.

Avem 2 cazuri posibile :

1. $i \notin V(D^*_{sj})$. Atunci $D^1 = D^*_{sj} \circ (j, ji, i) \in \mathbf{D}_{si}$ şi $a(D^1) = a(D^*_{sj}) + a_{ji} < a(D_{sj}) + a_{ji} = a(D^*_{si})$, ceea ce contrazice alegerea drumului D^*_{si} .

2. $i \in V(D^*_{sj})$. Fie $D^*_{si} = (D_{sj}) \circ (D_{ij})$ cele doua drumuri determinate pe D^*_{sj} de vârful i. Atunci circuitul $C = (D_{ij}) \circ (j, ji, i)$ are costul $a(C) = a(D_{ij}) + a_{ji} = a(D^*_{sj}) - a(D_{sj}) + a_{ji} = u_j + a_{ji} - a(D_{sj}) \leq u_j + a_{ji} - a(D^*_{sj}) = u_j + a_{ji} - u_i < 0$, contrazicând ipoteza **(I)**

Rezulta că presupunerea că u_i nu satisface **(*)** este falsa şi suficienta teoremei este demonstrata.

Notăm că de fapt am dovedit mai sus că $a(D_{sj}) = a(D^*_{sj})$ adică, dacă j este vârful dinaintea lui i pe un drum de cost minim de la s la i atunci şi porţiunea de drum de la s la j este de cost minim de la s la j. Inductiv rezulta că: (Principiul optimalităţii al lui Bellman) dacă D^*_{si} este drum de cost minim de la s la i atunci $\forall\, j \in V(D^*_{si})$ dacă $D^*_{si} = D_{sj} \circ D_{ji}$ atunci D_{sj} (respectiv D_{ji}) sunt drumuri de cost minim de la s la j (respectiv de la j la i).

"$\Rightarrow$". Dovedim că dacă $(u_1, \ldots, u_n)$ este o soluţie a lui **(*)** atunci:

a) $\exists\, D_{si} \in \mathbf{D}_{si} : u_i = a(D_{si})\ \forall\, i \in V$

b) ***Pentru orice*** $i \in V$, $u_i = \min\{a(D) \mid D \in \mathbf{D}_{si}\}$ $(= a(D^*_{si}))$

a) Dacă $i = s$ atunci $u_s = 0$ şi drumul D_{ss} satisface $a(D_{ss}) = 0 = u_s$. Dacă $i \neq s$, consideram următorul algoritm :

```
begin
    v:=i;
    k:=0;
    while v ≠ s do
        begin
            determină w astfel încât u_v = u_w + a_wv ;
            {există w pentru care u_V satisface (*)}
            i_k:=v;
            k:=k+1;
            v:=w;
        end;
    i_{k+1}:=s
end.
```

Să observam că algoritmul determina drumul :

$$D : (s =)\ i_{k+1}, i_{k+1}i_k, \ldots, i_1, i_1i_0, i_0\ (= i)$$

cu $D \in D_{si}$ satisfăcând $a(D) = a(i_{k+1}i_k) + \ldots + a(i_1i_0) = (u_{i_k} - u_{i_{k+1}}) + (u_{i_{k-1}} - u_{i_k}) + \ldots + (u_{i_0} - u_{i_1}) = u_{i_0} - u_{i_{k+1}} = u_i - u_s = u_i$.

Nu este posibil ca într-o iteraţie oarecare ca $w \in \{i_0, \ldots, i_{k-1}\}$, căci atunci s-ar obţine un circuit C de cost total 0, contrazicând ipoteza **(I)**.

Din construcţie se observă că $u_i = u_{i_1} + a_{i_1 j}$.

b) Fie $\bar{u}_i = a(D^*_{si})\ \forall\ i \in V$. Conform primei părţi a demonstraţiei $\bar{u}_i$, $i = \overline{1,n}$, satisfac sistemul **(*)**. Presupunem că $u = (u_1, \ldots, u_n) \neq \bar{u} = (\bar{u}_1, \ldots, \bar{u}_n)$. Cum $u_s = \bar{u}_s = 0$, rezultă că există $i \neq s$ astfel încât $u_i \neq \bar{u}_i$ şi $\forall\ j \in V(D_{si}), j \neq i, u_j = \bar{u}_j$, unde D_{si} este drumul construit la (a) pentru $\bar{u}_i$. Atunci avem:

$$u_i > \bar{u}_i = \bar{u}_{i_1} + a_{i_1 i} = u_{i_1} + a_{i_1 i} \quad (\textit{din alegerea lui } i)$$
$$\geq u_i \quad (\textit{pentru că } u_i \textit{ satisface } (*))$$

Observaţii 1. Din demonstraţie rezulta ca pentru rezolvarea problemei **P2** este suficient sa obţinem o soluţie a sistemului **(*)**. Drumurile corespunzătoare se obţin ca la (a). Algoritmii pe care ii vom prezenta se vor ocupa de rezolvarea sistemului **(*)**. Totuşi, dacă avem $u_i = u_k + a_{ki}$ atunci aşa cum am văzut, k este vârful dinaintea lui i de pe drumul minim de la s la i de cost u_i. Rezultă că dacă în algoritmul de rezolvare a lui **(*)** construim un vector *înainte* = **array**[1..n] **of** V cu interpretarea finală "*înainte*[i] = vârful dinaintea lui i de pe drumul minim de la s la i", atunci vârfurile acestui drum pot fi determinate în **O(n)** construind şirul i, *înainte*[i], *înainte*[*înainte*[i]], …, până se depistează vârful s.

2. Dacă algoritmii de rezolvare a lui **(*)** vor evita (prin modul de actualizare a vectorului înainte) apariția circuitelor de cost total **0**, atunci se observă că, deşi nu mai are loc unicitatea soluției sistemului **(*)**, problema (**P2**) este rezolvată. Rezultă că aceşti algoritmi vor rezolva problema (**P2**) în condiția:

(I') $\forall\ C$ circuit în G, $a(C) \geq 0$.

3. În cazul grafurilor, rezolvarea problemelor (**P1**) – (**P3**) corespunzătoare se poate face utilizând algoritmii pentru digrafuri, prin înlocuirea fiecărei muchii cu o pereche de arce simetrice de acelaşi cost ca şi muchia pe care o înlocuiesc. Dificultatea unei astfel de abordări rezulta din introducerea pentru muchii de cost negativ a unor circuite de lungime **2** de cost negativ. Deci, în cazul grafurilor, algoritmii pentru digrafuri sunt valabili doar dacă toate costurile sunt nenegative.

4. Având în vedere că mulțimile D_{ij} sunt finite se pot considera probleme analoage problemelor (**P1**) – (**P3**) înlocuind **min** cu **max**. Utilizarea ideii uzuale,

$$\max_{x \in A} x = -(\min_{x \in A}(-x))$$

prin înlocuirea costurilor a_{ij} cu $-a_{ij}$ este posibilă doar în cazul digrafurilor în care pentru orice circuit C avem $a(C) \leq 0$. În particular, această abordare este posibilă în cazul digrafurilor fără circuite (ca în aplicațiile b) şi c) prezentate). Dacă digraful inițial are circuite, problemele de drum de cost maxim se pot dovedi uşor (prin reducerea polinomială la probleme hamiltoniene) a fi **NP** – dificile.

Rezolvarea problemei (P2) în cazul digrafurilor fără circuite

În acest caz, dupa sortarea topologica a digrafului, care oferă o numerotare aciclica, sistemul **(*)** se poate rezolva prin substituție :

```
begin
    1.  Sortează topologic G; {O(n+e) operații}
    2.  u1 := 0; înainte[1] := 0;
        for i:=2 to n do
            begin
                ui :=∞;
                înainte[i]:=0;
                for j:=1 to i –1 do
                    if ui > uj + aji then
                        begin
                            ui := uj + aij;
                            înainte[i]:=j;
                        end;
            end;
end.
```

Complexitatea pasului **2** este, evident $\mathbf{O(1+2+\ldots+n-1)=O(n^2)}$

Notăm că în enunțul algoritmului am presupus că vârful s în problema (**P2**) devine în numerotarea aciclica vârful **1**, (altfel se rezolva problema **P2** doar în subdigraful indus de vârfurile i, care vor fi în numerotarea aciclica mai mari sau egale decât s).

Rezolvarea problemei (P2) în cazul costurilor nenegative.

Dacă $a_{ij} \geq 0 \ \forall \ ij \in E$ atunci condiția **(I)** este îndeplinita şi o soluție a sistemului **(*)** se poate obține cu ajutorul următorului algoritm (Dijkstra, 1961) .

Se considera $S \subseteq V$ astfel încât pe tot parcursul algoritmului are loc

$$\textbf{(D)} \quad \begin{cases} \forall i \in S & u_i = \min\{a(D_{si}) \mid D_{si} \in \mathbf{D}_{si}\} \\ \forall i \in V \setminus S & u_i = \min\{a(D_{si}) \mid D_{si} \in \mathbf{D}_{si}, V(D_{si}) \setminus S = \{i\}\} \end{cases}$$

Dacă se reuşeşte construirea lui S astfel încât $S = V$, atunci problema este rezolvata. Inițial, se va considera $S = \{s\}$ şi în $n - 1$ paşi se adaugă la S câte un vârf nou din V

Algoritmul lui Dijkstra

begin

1. S:=s; u_S :=0;înainte[s]:=0;

for i ∈ V \ {s} **do**

begin

u_i := a_{Si}; înainte[i]:=s;

end;

{se observă ca după aceste inițializari (D) are loc }

2. **while** S ≠ V **do**

begin

determină j*∈V\S a.i. u_{j*}=min{ u_j | j∈V\S };

S:=SU{j*};

for j ∈ V \ S **do**

if $u_j > u_{j*} + a_{j*j}$ **then**

begin

$u_j := u_{j*} + a_{j*j}$;

înainte[j]:=j*;

end;

end;

end.

Corectitudinea algoritmului va rezulta dacă vom arata că, dacă înaintea unei iterații din pasul **2** are loc **(D)**, atunci, după execuția acelei iterații, **(D)** are loc de asemenea .

Arătam mai întâi că în ipoteza că **(D)** are loc, atunci adăugarea lui j^* la S nu contrazice **(D)**. Deci trebuie dovedit că dacă $u_{j^*} = \min\{ u_j \mid j \in V \setminus S\}$ atunci $u_{j^*} = \min\{ a(D_{sj^*}) \mid D_{sj^*} \in D_{sj^*}\}$. Presupunem că dacă există $D^1_{sj^*} \in \mathbf{D}_{sj^*}$ astfel încât $a(D^1_{sj^*}) < u_{j^*}$. Fie k primul vârf al drumului $D^1_{sj^*}$ (în parcurgerea sa din s), astfel încât $k \notin S$, atunci $a(D^1_{sj^*}) = a(D^1_{sk}) + a(D^1_{kj^*})$.

Din alegerea lui k, avem $V(D^1_{sk}) \setminus S = \{k\}$ şi cum **(D)** are loc, avem $a(D^1_{sk}) = u_k$. Obținem $u_{j^*} > a(D^1_{sk^*}) = u_k + a(D^*_{kj}) \geq u_k$ (costurile sunt nenegative), ceea ce contrazice alegerea lui j^*. Contradicția obținută arata că, după atribuirea $S := SU\{j^*\}$, prima parte a condiției **(D)** are loc.

Pentru ca şi cea de-a doua parte a condiţiei **(D)** să aibă loc după această atribuire, să observăm că $\forall\ j \in V \setminus (S \cup \{j^*\})$ avem $\mathbf{min}\{a(D_{sj}) \mid D_{sj} \in D_{sj},\ V(D_{sj}) \setminus (S \cup \{j^*\}) = \{j\}\} = \mathbf{min}(\mathbf{min}\{a(D_{sj}) \mid D_{sj} \in D_{sj},\ V(D_{sj}) \setminus S = \{j\}\},\ \mathbf{min}\{a(D_{sj}) \mid D_{sj} \in D_{sj},\ V(D_{sj}) \setminus S = \{j, j^*\}\})$.

Cum **(D)** are loc, primul dintre cele două minime de mai sus este u_j. Fie α_j valoarea celui de-al doilea minim şi fie D^1_{sj} drumul pentru care se realizează. Cum $j^* \in V(D^1_{sj})$ avem $\alpha_j = a(D^1_{sj^*}) + a(D^1_{j^*j})$.

Întrucât $S \cup \{j^*\}$ satisface prima parte a lui **(D)**, avem $a(D^1_{sj^*}) = u_{j^*}$ (altfel s-ar contrazice alegerea lui D^1_{sj} înlocuind în D^1_{sj}, porţiunea $D^1_{sj^*}$ cu un drum de cost mai mic). Deci $\alpha_j = u_{j^*} + a(D^1_{j^*j})$.

Dacă drumul $D^1_{j^*j}$ este de lungime **1** atunci avem $\alpha_j = u_j + a(D^1_{j^*j})$. Altfel, considerând k vârful dinaintea lui j de pe drumul D^1_{sj} avem $k \neq j$, k ? S şi $\alpha_j = a(D^1_{sk}) + a_{kj}$. Cum $S \cup \{j^*\}$ satisface prima parte a lui **(D)**, obţinem $\alpha_j = u_k + \mathbf{a}_{kj}$.

Întrucât S satisface **(D)**, u_k este costul unui drum minim de la s la k cu vârfurile conţinute în S deci α_j este costul unui drum de la s la j cu vârfurile conţinute în S. Rezultă că $\alpha_j \geq u_j$ căci S satisface **(D)**. Am obţinut ca singurul caz când $\alpha_j < u_j$ este atunci când $\alpha_j = u_{j^*} + a_{j^*j}$, situaţie testata în ciclul **for** al pasului **2**.

Rezultă că **(D)** are loc pe tot parcursul algoritmului şi deci valorile finale ale variabilelor u_i reprezintă soluţia sistemului **(*)**. Evident, tabloul *înainte* este actualizat pentru memorarea implicita a drumurilor de cost minim. Complexitatea algoritmului, în descrierea dată este $\mathbf{O(n^2)}$ datorita selectării minimelor de la pasul **2**.

Observaţii :

1. Este posibila organizarea unei cozi cu prioritate pentru memorarea valorilor u_i, $i \in V \setminus S$, astfel încât extragerea minimului să se facă în **O(1)**, iar actualizările necesare în pasul **2** să se facă în timp total de **O(m log n)** unde **m** = | E |.
2. Dacă se doreşte rezolvarea problemei (**P1**) cu ajutorul algoritmului lui Dijkstra, atunci la introducerea lui t în S, se poate opri algoritmul. Complexitatea, în cazul cel mai nefavorabil, rămâne aceeaşi. Totuşi, în situaţii practice concrete există posibilitatea de a grăbi introducerea lui t în S utilizând o funcţie de dirijare a procesului de construcţie a lui S.

O funcţie $g : V \rightarrow \mathbf{R}_+$ se numeşte estimator consistent dacă :

ι (i) $\forall\ i \in V\ u_i + g(i) \leq \mathbf{min}\ \{a(D_{st}) \mid D_{st} \in D_{st}$ şi $i \in V(D_{st})\}$

ιι (ii) $\forall\ ij \in E\ g(i) \leq a_{ij} + g(j)$.

Să observăm că $g(i) = 0$, $\forall\ i$ este un estimator consistent. Dacă $V(G)$ este o mulţime de puncte din plan, atunci $g(i)$ = distanta (euclidiana) de la i la t este un estimator consistent, dacă sunt satisfăcute condiţiile (ii). Dacă g este un estimator consistent atunci se poate modifica alegerea lui j^* în algoritm astfel: $u_{j^*} + \mathbf{g(j^*)} = \mathbf{min}\{u_j + g(j) \mid j \in V \setminus S\}$. Algoritmul rămâne valabil (demonstraţia este identica situaţiei $g(i) = 0$, $\forall\ i$ şi se foloseşte (ii) repetat). Avantajul este acela că se vor introduce în S vârfuri care să ne apropie de t.

3. În implementarea care rezulta din descrierea algoritmului lui Dijkstra, s-a presupus (aşa cum am precizat) că se dispune de matricea de cost-adiacenta a digrafului. În cazul digrafurilor cu multe vârfuri (în care, de exemplu, listele de adiacenţă sunt memorate în memoria secundara), sau în cazul digrafurilor date funcţional (se dispune de o procedura care construieşte pentru un vârf dat, lista sa de adiacenta, ca de exemplu în aplicaţia c)) această implementare este neeficientă, respectiv neaplicabilă. O

implementare care nu are aceste deficiente este următoarea dată de Glover, Klingman şi Philips (1985) (Partition Shortest Path algorithm, PSP algorithm)

```
begin
    1.  u_S:=0; înainte[s] := 0; S := 0; NOW := s; NEXT :=Ø ;
    2.  while NOW ∪ NEXT ≠ Ø do
        begin
            while NOW ≠ Ø do
                begin
                    Extrage i din NOW;
                    S := S ∪ {i};
                    L := N+G(i); {se generează în lista L, lista de adiacență şi costurile corespunzătoare.}
                    for j ∈ L \ S do
                        if j ∉ NOW ∪ NEXT then
                            begin
                                uj:= ui+ aij;înainte(j):=i;
                                introdu j în NEXT
                            end
                        else
                            if uj > ui + aij then
                                begin
                                    uj:= ui+ aij;
                                    înainte(j):=i;
                                end
                end;
            if NEXT ≠ Ø then
                begin
                    determină d = min{ui | i ∈ NEXT};
                    transfera ∀ i ∈ NEXT cu ui = d în NOW
                end
end.
```

Rezolvarea problemei (P2) în cazul general.

Dacă există ***ij*** **∈** ***E*** astfel încât $a_{ij} < 0$ algoritmul lui Dijkstra nu mai este valabil în general (introducerea lui ***j**** în ***S*** poate conduce la violarea condiției **(D)**). Considerând îndeplinita condiția **(I')** vom rezolva sistemul **(*)** prin aproximații succesive.

Considerăm ∀ ***i*** **∈** ***V*** şi ∀ ***m*** **= 1, …, *n*-1**

(BM) $u_i^m = \min\{a(D) \mid D \in D_{si}$, numărul arcelor lui D este $\le m\}$

Cum orice drum în ***G*** are cel mult ***n*** **– 1** arce rezulta că dacă reuşim construcția lui $u^1 = (u_1^1, ..., u_n^1)$, $u^2 = (u_1^2, ..., u_n^2)$, …, $u^{n-1} = (u_1^{n-1}, ..., u_n^{n-1})$, atunci u^{n-1} este soluția sistemului **(*)**. Algoritmul care rezulta este următorul:

Algoritmul lui Bellman, Ford, Moore (aproximativ 1960)

begin

1. u_s^1:=0; **for** i ∈ V \ {s} **do** u_i^1:= a_{si} ;
   {evident **(BM)** are loc }
2. **for** m = 1 **to** n – 2 **do**
   for i = 1 **to** n **do**
   u_i^{m+1} := min(u_i^m, $\min_{j \neq i}(u_j^m + a_{ji})$)

end.

Pentru a demonstra corectitudinea algoritmului, arătăm că dacă $\boldsymbol{u^m}$ ($\boldsymbol{m \geq 1}$) satisface **(BM)** atunci şi $\boldsymbol{u^{m+1}}$ o satisface. Fie $\boldsymbol{i \in V}$ şi consideram mulţimile de drumuri:

$\boldsymbol{A}$ = {$\boldsymbol{D}$ | $\boldsymbol{D \in D_{si}}$, numărul arcelor lui $\boldsymbol{D \leq m+1}$}.

$\boldsymbol{B}$ = {$\boldsymbol{D}$ | $\boldsymbol{D \in D_{si}}$, numărul arcelor lui $\boldsymbol{D \leq m}$}.

$\boldsymbol{C}$ = {$\boldsymbol{D}$ | $\boldsymbol{D \in D_{si}}$, numărul arcelor lui $\boldsymbol{D = m+1}$}.

Atunci $\boldsymbol{A = B \cup C}$ şi $\boldsymbol{\min\{a(D) \mid D \in A\} = \min(\min\{a(D) \mid D \in B\}, \min\{a(D) \mid D \in C\})}$.

Cum $\boldsymbol{u^m}$ satisfac **(BM)** rezultă că

$\boldsymbol{\min(a(D) \mid D \in A) = \min(u_i^m, \min\{a(D) \mid D \in C\})}$. Fie $\boldsymbol{\min\{a(D) \mid D \in C\} = a(D^0)}$, $\boldsymbol{D^0 \in C}$. Dacă $\boldsymbol{j}$ este vârful ce-l precede pe $\boldsymbol{i}$ în $\boldsymbol{D^0}$ (există, întrucât $\boldsymbol{D^0}$ are măcar **2** arce) atunci $\boldsymbol{a(D^0) = a(D^0_{sj}) + a_{ji} \geq u_j^m + a_{ji}}$ întrucât $\boldsymbol{D^0_{sj}}$ are $\boldsymbol{m}$ arce şi $\boldsymbol{u^m}$ satisface **(BM)**.

Rezultă că $\boldsymbol{\min\{a(D) \mid D \in A\} = \min\{u_i^m, \min_{j \neq i}(u_j^m + a_{ji})\}}$ valoare care în algoritm se atribuie lui $\boldsymbol{u_i^{m+1}}$. Observăm că algoritmul are complexitate de $\mathbf{O(n^3)}$ dacă determinarea minimului din pasul doi necesita **O(n)** operaţii. Determinarea drumurilor minime se face menţinând vectorul *înainte*, iniţializat în mod evident în pasul **1** şi actualizat în mod corespunzător, la stabilirea minimului din pasul **2**.

Observaţii:

1. Dacă la algoritm se adaugă şi pasul 3:
2. **if** ($\boldsymbol{\exists\, i \in V}$ astfel încât $\boldsymbol{u_i^{n-1} > \min_{j \neq i}(u_i^{n-1} + a_{ji})}$) **then**
   "Există circuit de cost negativ" .

se obţine posibilitatea testării în $\mathbf{O(n^3)}$ a existentei unui circuit de cost negativ în digraful $\boldsymbol{G}$. În adevăr, dacă

(1) $\boldsymbol{u_i^{n-1} > \min_{j \neq i}(u_j^{n-1} + a_{ji}) = u_k^{n-1} + a_{ki}}$,

atunci, din demonstraţia corectitudinii algoritmului, rezultă că singura posibilitate este ca $\boldsymbol{u_k^{n-1} = a(D^0)}$, unde $\boldsymbol{D^0 \in D_{sk}}$ are exact $\boldsymbol{n-1}$ arce şi deci vârful $\boldsymbol{i \in V(D_{sk})}$. Deci $\boldsymbol{u_k^{n-1} = a(D^0) = a(D^0_{si}) + a(D^0_{ik}) = u_i^{n-1} + a(D^0_{ik})}$ ($\boldsymbol{u^{n-1}}$ este soluţie a sistemului **(*)**, şi se aplică principiul optimalităţii a lui Bellman).

Circuitul $\boldsymbol{C = D^0_{ik} \circ (k, ki, i)}$ are costul $\boldsymbol{a(C) = a(D^0_{ik}) + a_{ki} = u_k^{n-1} - u_i^{n-1} + a_{ki} < 0}$, din ipoteza **(1)**. Depistarea circuitului $\boldsymbol{C}$ se face simplu (**O(n)**) utilizând vectorul *înainte*: ($\boldsymbol{k}$, *înainte*($\boldsymbol{k}$), *înainte*(*înainte*($\boldsymbol{k}$)), …, $\boldsymbol{i}$, $\boldsymbol{k}$) sunt vârfurile sale în ordinea inversa a parcurgerii).

2. dacă $\boldsymbol{\exists\, k < n-1}$ astfel încât $\boldsymbol{u^k = u^{k+1}}$ atunci algoritmul se poate opri.

Descriem în continuare o implementare a acestui algoritm, care are complexitatea **O(nm)** (**m** = | $\boldsymbol{E}$ |). Vom folosi o coada $\boldsymbol{UQ}$ în care se vor păstra vârfurile $\boldsymbol{i}$ cărora li se modifica $\boldsymbol{u_i}$ curent (se va renunţa, evident, la memorarea tuturor aproximaţiilor succesive). Prezenţa unui vârf în coada se va face utilizând un vector boolean $\boldsymbol{UB}$. Vom utiliza de asemenea un

tablou ***nr*** = **array**[1..n] **of integer** care contorizează pentru fiecare vârf ***i*** numărul modificărilor variabilei u_i corespunzător lui ***i***. Atunci când ***nr(i)*** $\geq$ ***n*** suntem în situația din observația 1 precedenta şi putem opri algoritmul cu mesajul existentei circuitului negativ. Descrierea algoritmului :

```
begin
    1: us=0; înainte(s) := 0; UQ = {s}; UB[s] := true; nr[s] := 1;
        for i ∈ V \ {s} do
            begin
                ui := infinit; UB[i] := false; nr[I] := 0;
            end;
    2: while UQ<>0 do
        begin
            extrage primul element i din UQ
            nr[i] := nr[i] + 1
            if nr[i] >= n + 1 then
                begin
                    "există circuit de cost negativ ce trece prin vârful i"
                    goto 3;
                end;
            UB[I] := false;
            for j ∈ A(i) do
                if ui + aij < uj then
                    begin
                        uj =ui + aij;
                        if not UB[j] then
                            begin
                                adaugă j la UQ;
                                UB[j] := true;
                            end;
                    end;
    end;  {while}
3 : end.
```

Nu am mai introdus operațiile necesare întreținerii tabloului *înainte* întrucât sunt evidente şi ar îngreuna lecturarea textului.

Rezolvarea problemei (P3).

Consideram $u_{ij} = \min\{a(D_{ij}) \mid D_{ij} \in D_{ij}\}$, $\forall\ i,j \in V$. Problema se reduce la determinarea matricii $U = (u_{ij})_{n\times n}$, atunci când se cunoaşte A, matricea de cost – adiacenta. Drumurile de cost minim vor fi obținute în **O(n)** dacă odată cu determinarea matricii U se va construi matricea $Înainte = (înainte(i, j))_{n\times n}$ cu elementele având semnificația *înainte*(***ij***) = vârful dinaintea lui ***j*** de pe drumul de cost minim de la ***i*** la ***j*** în ***G***.

Să observăm că dacă $a_{ij} \geq 0\ \forall\ i, j$ atunci, iterând algoritmul lui Dijkstra pentru $s \in \{1,.., n\}$, se obține un algoritm de complexitate $\mathbf{O(n^3)}$.

Dacă ***G*** nu conține circuite de cost negativ, dar există şi arce de cost negativ, iterând algoritmul lui Bellman-Ford pentru ***s* = 1..*n*** se obține un algoritm de complexitate $\mathbf{O(n^4)}$. Arătam în continuare ca se poate proceda şi mai eficient.

Soluţia I[a] . Fie $\alpha : V \to \mathbf{R}$ a. î. $\forall\, i, j \in E \;\; \alpha(i) + a_{ij} \geq \alpha(j)$. Consideram $a^*: E \to \mathbf{R}_+$ dată de

$a^*_{ij} = a_{ij} + \alpha(i) - \alpha(j)$, $\forall\, i, j \in E$. Avem $a^*_{ij} \geq 0$ şi, în plus, $\forall\, D_{ij} \in D_{ij}$,

(2) $a^*(D_{ij}) = a(D) + \alpha(i) - \alpha(j)$.

Rezultă că se poate itera algoritmul Dijkstra pentru obţinerea drumului de cost a^* minim şi din relaţia **(2)** se observă că un drum este de cost a^* minim dacă şi numai dacă este drum de cost a minim. Rezulta următorul algoritm :

1. determină α şi construieşte A^* .
2. rezolvă (**P3**) pentru A^* construind U^* şi *Înainte.*
3. determina U ($u_{ij} = u^*_{ij} - \alpha(i) + \alpha(j), \forall ij$).

Pasul 2 al algoritmului necesita $\mathbf{O(n^3)}$ operaţii prin iterarea algoritmului lui Dijkstra. Pasul 1 se poate realiza în $\mathbf{O(n^3)}$ operaţii, fixând $s \in V$ şi rezolvând (**P2**) cu algoritmul Bellman – Ford. În adevăr, dacă u_i, $i \in V$ este soluţia lui (**P2**) atunci u_i, $i \in V$ este soluţie a sistemului **(*)**, deci $u_j = \min_{i \neq j}\{u_i + a_{ij}\}$ adică $\forall\, ij \in E$, $u_j \leq u_i + a_{ij}$, deci $a_{ij} + u_i - u_j \geq 0$. Prin urmare se poate considera $\alpha(i) = u_i$, $\forall\, i \in V$.

Soluţia a II-a.

Fie $u_{ij}^{m} = \min\{a(D_{ij}) \mid D_{ij} \in D_{ij},\, V(D_{ij}) \setminus \{i, j\} \subseteq \{1, 2, \ldots, m\text{-}1\}\}$ $\forall\, i, j, m \in \{1, 2, ..., n\}$

Atunci, evident $u_{ij}^{1} = a_{ij}$ $\forall\, i, j \in V$ (presupunem, din nou matricea A având elementele diagonale egale cu **0**). În plus,

$u_{ij}^{m+1} = \min\{u_{ij}^{m}, u_{im}^{m} + u_{mj}^{m}\}$ $\forall\, i, j \in V, \forall\, m = 1, 2, ..., n$

Această ultimă relaţie se poate justifica inductiv: un drum de cost minim de la i la j care nu are vârfuri interioare $\geq m + 1$ poate să nu conţină vârful m, şi atunci are costul u_{ij}^{m}, sau poate conţine vârful m, şi atunci, din principiul optimalităţii al lui Bellman şi ipoteza inductiva, este $u_{im}^{m} + u_{mj}^{m}$.

Evident, dacă se obţine $u_{ii}^{m} < 0$ atunci digraful conţine un circuit de cost negativ C care trece prin vârful i, cu $V(C) \setminus \{i\} \subseteq \{1, \ldots, m\text{-}1\}$. Aceasta soluţie a problemei (**P3**) este cunoscuta ca algoritmul lui **Floyd -Warshal** şi poate fi descris astfel:

```
begin
    1:  for i:=1 to n do
            for j:=1 to n do
                begin
                    înainte(i,j):=i;
                    if i = j then
                        begin
                            a_ii := 0; înainte(i, i):=0
                        end
                end;
2:  for m:=1 to n do
            for i:=1 to n do    for j:=1 to n do
                if a_ij>a_im+a_mj then
                    begin
                        a_ij:=a_im+a_mj;
                        înainte(i,j):=înainte(m,j)
                        if (i=j and a_ij<0) then
```

```
                                    begin
                                        "circuit negativ" ;
                                        goto 3
                                    end
                            end;
3: end.
```

Evident, complexitatea algoritmului este de $\mathbf{O(n^3)}$.

Observație. Dacă digraful nu conține circuite de cost negativ, atunci inițializând a_{ii} **:=0**, valorile finale ale elementelor diagonale dau costul minim al câte unui circuit ce trece prin vârful corespunzător.

Teorema lui Menger şi aplicații ale acesteia.

Definiție. Fie $G = (V, E)$ (di)graf şi $X,Y \subseteq V$. Numim ***XY – drum*** în G orice drum D în G de la un vârf $x \in X$ la un vârf $y \in Y$, astfel încât

$$V(D) \cap X = \{x\} \text{ şi } V(D) \cap Y = \{y\}.$$

Vom nota cu $\Delta(X, Y; G)$ mulțimea tuturor XY-drumurilor în G .

Să observăm că dacă $x \in X \cap Y$ atunci drumul de lungime $\mathbf{0}$, $D = \{x\}$, este XY-drum.

Vom spune că drumurile D_1 şi D_2 sunt disjuncte dacă $V(D_1) \cap V(D_2) = \emptyset$.

Probleme practice evidente, din rețelele de comunicație, dar şi unele probleme legate de conexiunea garfurilor şi digrafurilor, necesita determinarea unor mulțimi de XY – drumuri disjuncte şi cu număr maxim de elemente. Vom nota cu $p(X, Y; G)$ numărul maxim de XY – drumuri disjuncte în (di)graful G. Teorema care precizează acest număr a fost stabilita de Menger în 1927 şi constituie unul din rezultatele fundamentale din teoria garfurilor.

Definiție. Fie $G = (V, E)$ un digraf şi $X, Y \subseteq V$. Numim ***mulțime XY – separatoare*** în G o mulțime $Z \subseteq V$ astfel încât $\forall D \in \Delta(X, Y; G)$, $V(D) \cap Z \neq \emptyset$.

Notam cu $S(X, Y; G) = \{Z \mid Z \ XY$ – separatoare în $G\}$ şi $k(X, Y; G) = \mathbf{min}\{|Z|; Z \in S(X, Y; G)\}$. Din definiție, rezultă următoarele proprietăți imediate ale mulțimilor XY – separatoare:

(a) Dacă $Z \in S(X, Y; G)$ atunci $\forall D \in \Delta(X, Y; G)$ D nu este drum în $G – Z$.
(b) $X, Y \in S(X, Y; G)$
(c) Dacă $Z \in S(X, Y; G)$ atunci $\forall A$ astfel încât $Z \subseteq A \subseteq V$ avem $A \in S(X, Y; G)$.
(d) Dacă $Z \in S(X, Y; G)$ si $T \in S(X, Z; G)$ sau $T \in S(Z , Y; G)$ atunci $T \in S(X, Y; G)$

Teorema 1. Fie $G = (V, E)$ (di)graf şi $X, Y \subseteq V$. Atunci

$$p(X, Y; G) = k(X, Y; G).$$

Demonstrație: 1^0. Dacă $p = p(X, Y; G)$ şi $D_1, D_2, \ldots, D_p$ sunt XY – drumuri disjuncte în G, atunci $\forall Z \in S(X, Y; G)$ avem $Z \cap V(D_i) \neq \emptyset$ şi cum D_i sunt disjuncte ($i = \mathbf{1}, p$): $|Z| \geq |Z \cap \cup_{i=1}^{p} V(D_i)| =$

$= \sum_{i=1,p} |Z \cap V(D_i)| \geq \sum_{i=1,p} \mathbf{1} = p$.

Deci $\forall Z \in S(X, Y; G)$ $|Z| \geq p$; în particular $k(X, Y; G) \geq p(X, Y; G)$.

2^0. Arătam prin inducție după $a(G) = |V| + |E|$ că

(*) $\forall G = (V, E), \forall X, Y \subseteq V, \exists k(X, Y; G)$ XY – drumuri disjuncte în G.

(Evident din **(*)** rezultă că $p(X, Y; G) \geq k(X, Y; G)$ şi deci teorema e demonstrată). Cum **(*)** se verifica pentru (di)grafuri G cu $a(G) = 1, 2$, consideram în pasul inductiv că **(*)** are loc pentru orice (di)graf G' şi orice $X', Y' \subseteq V(G')$, cu $a(G') < a(G)$. Pentru a exclude cazurile banale, vom presupune că $X \not\subset Y$, $Y \not\subset X$ şi $k = k(X, Y; G) > 0$.

Cazul 1. Există $Z \in S(X, Y; G)$ a. î. $|Z| = k$, $Z \neq X, Y$.

Considerăm $V_{XZ} = \{v \mid \exists D \in D(X, Z; G) : v \in V(D)\}$ şi $V_{ZY} = \{v \mid \exists D \in D(Z, Y; G) : v \in V(D)\}$.

Să observăm că $V_{XZ} \cap V_{ZY} = Z$ (dacă există $v \in V_{XZ} \cup V_{ZY} - Z$, atunci se obţine că Z nu este XY – separatoare; dacă există $z \in Z$ a. î. $z \notin V_{XZ} \cap V_{ZY}$ atunci $Z - \{z\}$ este XY – separatoare, contrazicând $|Z| = k(X, Y; G)$). Pe de altă parte, există $x \in X - Z$ (dacă $X \subseteq Z$, atunci cum $X \in S(X, Y; G)$ şi $|Z| = k(X, Y; G)$ rezulta $X = Z$, contrazicând ipoteza cazului 1) şi evident $x \notin V_{ZY}$ (altfel, Z nu ar fi XY – separatoare). Rezultă $|V_{ZY}| < |V|$. În mod similar $|V_{XZ}| < |V|$.

Fie $G_{XZ} = [V_{XZ}]_G$ şi $G_{ZY} = [V_{ZY}]_G$. Din observaţiile precedente: $a(G_{XZ}), a(G_{ZY}) < a(G)$.

Să observam că $k(X, Z; G_{XZ}) = k$ şi $k(Z, Y; G_{ZY}) = k$ (Z este XZ – separatoare în G_{XZ}, respectiv ZY – separatoare în G_{ZY} şi are cardinalul k; dacă în unul din cele doua grafuri, ar există o mulţime T separatoare de cardinal $< k$, atunci, utilizând observaţia (d), se contrazice definiţia lui k pentru G, X şi Y).

Din ipoteza inductivă, rezultă că există k X Z – drumuri disjuncte în G_{XZ} şi k Z Y – drumuri disjuncte în G_{ZY}. Cum $V_{XZ} \cap V_{ZY} = Z$ şi $|Z| = k$, rezultă că aceste $2k$ drumuri se pot concatena două câte două în G şi deci **(*)** are loc.

Cazul 2. $\forall Z$ X Y – separatoare a. î. $|Z| = k$ avem $Z = X$ sau $Z = Y$.

Presupunem, pentru precizarea notaţiilor, $Z = X$. Cum $X \not\subset Y$, $\exists x \in X \setminus Y$. $X - \{x\}$ nu este XY – separatoare (are mai puţin de k elemente). Există deci un XY – drum în G. Fie $e = xy$ prima muchie (arc) a acestui drum (există!). Să observăm că $y \notin X$. Considerăm $G' = G - e$. Avem $a(G') < a(G)$, deci **(*)** are loc pentru G', X şi Y.

Dacă $k(X, Y; G') = k$, atunci cele k XY – drumuri disjuncte din G' sunt XY – drumuri disjuncte şi în G deci **(*)** are loc pentru G, X şi Y.

Dacă $k(X, Y; G') < k$, atunci în G' există Z' XY – separatoare cu $|Z'| = k - 1$ (se aplică, eventual proprietatea (c)). Z' nu este XY – separatoare în G ($|Z'| < k$). Singurele XY – drumuri pe care Z nu le intersectează sunt cele care au drept primă muchie (arc) pe e. Din definiţia lui k, rezulta ca $x \notin Z'$, $y \notin Z'$ şi $|Z' \cup \{x\}| = |Z' \cup \{y\}| = k$. Din alegerea lui x, y avem $Z' \cup \{x\} \neq Y$ şi $Z' \cup \{y\} \neq X$. Din ipoteza cazului 2, rezultă atunci că $Z' \cup \{x\} = X$ şi $Z' \cup \{y\} = Y$. Cele k drumuri din **(*)** sunt în acest caz $\{z\}_{z \in Z'}$ şi (x, xy, y).

Cu acestea , teorema este demonstrată.

Observaţii :

1^0 Egalitatea min-max din enunţul teoremei este interesantă şi conduce, aşa cum vom vedea, la rezultate importante, în cazuri particulare.

2^0. Teorema se poate demonstra şi algoritmic ca o consecinţă a teoremei fluxului maxim-secţiunii minime.

Forma echivalenta în care a fost enunţată şi demonstrată iniţial de Menger (1927) teorema 1 este:

Teorema 1'. Fie $G = (V, E)$ un (di)graf şi $s, t \in V$, astfel încât $s \neq t$, $st \notin E$. Există k drumuri intern disjuncte de la s la t în graful G dacă şi numai dacă îndepărtând mai puţin

de k vârfuri diferite de s şi t, în graful rămas există un drum de la s la t.

Notăm că două drumuri sunt intern disjuncte dacă nu au vârfuri comune cu excepția extremităților. Se observă că dacă se considera $X = N_G(s)$ şi $Y = N_G(t)$ (respectiv, $N^+_G(s)$ şi $N^-_G(t)$ în cazul digrafurilor) teorema 1' se obține imediat din teorema 1. Reciproc, o construcție inversă celei de mai sus asupra tripletului G, X, Y din teorema 1, arată că teorema 1 se obține din teorema 1'.

Am definit un graf G p – conex ($p \in N^*$) dacă $G = K_p$ sau dacă $|G| > p$ şi G nu poate fi deconectat prin îndepărtarea a mai puțin de p vârfuri. Utilizând teorema 1' obținem.

Consecința. Un graf G este p – conex dacă $G = K_p$ sau $\forall\ st \in E(\overline{G})$ există p drumuri intern disjuncte de la s la t în G.

Determinarea numărului $k(G)$ de conexiune a grafului G (cea mai mare valoare a lui p pentru care G este p–conex) se reduce deci la determinarea lui $\min_{st \in E(\overline{G})} p(\{s\}, \{t\}; G)$ problemă care vom dovedi că se poate rezolva în timp polinomial.

Un caz particular interesant al teoremei 1, se obține atunci când G este un graf bipartit iar X şi Y sunt cele doua clase ale bipartiției :

Teorema 2. (Konig,1931) Dacă $G = (S, R; E)$ este un graf bipartit, atunci cardinalul maxim al unui cuplaj este egal cu cardinalul minim al unei mulțimi de vârfuri incidente cu toate muchiile grafului.

Demonstrație: Evident, cardinalul maxim al unui cuplaj în G este $p(S, R; G)$, care este egal, conform teoremei 2, cu $k(S, R; G)$. Teorema rezultă imediat dacă observăm că o mulțime de vârfuri este SR – separatoare dacă şi numai dacă este incidentă cu orice muchie a grafului.

O aplicație, fundamentală în numeroase raționamente combinatorii, a acestei teoreme este teorema lui Hall (1935).

Definiție : Fie I şi S mulțimi finite nevide. Numim familie de submulțimi ale lui S (indexată după I) orice aplicație $A : I \to 2^S$. Vom nota familia $A = (A_i; i \in I)$ şi vom folosi notația funcțională uzuală $A(J) = U_{j \in J} A_j$ (pentru $J \subseteq I$).

Dacă $A = (A_i; i \in I)$ este o familie de submulțimi ale lui S, o funcție $r_A : I \to S$ cu proprietatea ca $r_A(i) \in A_i$, $\forall\ i \in I$ se numeşte funcție de reprezentare pentru familia A. În acest caz, $(r_A(i); i \in I)$ formează un sistem de reprezentanți a familiei A. Dacă funcția de reprezentare r_A este injectivă atunci $r_A(I) \subseteq S$ se numeşte sistem de reprezentanți distincți ai familiei A, sau transversală.

Problema centrală în teoria transversalelor este aceea de a caracteriza familiile A care admit transversale (eventual cu anumite proprietăți). Prima teorema de acest tip a fost stabilita de Hall în 1935 :

Teorema 3. Familia $A = (A_i; i \in I)$ de submulțimi ale lui S admite o transversală dacă şi numai dacă

(H) $\qquad |A(J)| \geq |J| \qquad \forall\ J \subseteq I.$

Demonstrație: Necesitatea este evidentă : dacă A admite o funcție r_A de

reprezentare injectivă atunci $\forall\ J \subseteq I$, $r_A(J) \subseteq A(J)$ şi deci $|\ A(J)\ | \geq |\ r_A(J)\ | \geq |\ J\ |$ (întrucât r_A este injectiva).

Suficienţa. Consideram graful bipartit $G_A = (I, S; E)$ unde am presupus $I \cap S = \emptyset$ (altfel, se considera copii izomorfe disjuncte) iar $E = \{is \mid i \in I, s \in S \wedge s \in A_i\}$. Se observă ca $N_{G_A}(i) = A_i$ şi ca A are o transversala dacă şi numai dacă G_A are un cuplaj de cardinal $|I|$. În ipoteza că **(H)** are loc, arătam că orice mulţime de vârfuri incidentă cu toate muchiile lui G_A are măcar $|\ I\ |$ elemente, ceea ce dovedeşte existenta cuplajului de cardinal $|\ I\ |$ (utilizând teorema 2).

Fie $X = I' \cup S' \subset I \cup S$ o mulţime de vârfuri incidentă cu toate muchiile. Rezultă că $N_{G_A}(I - I') \subseteq S'$, adică $A(I - I') \subseteq S'$. Atunci, $|\ X\ | = |\ I'\ | + |\ S'\ | \geq |\ I'\ | + |A(I - I')|$. Folosind condiţia **(H)** obţinem în continuare: $|\ X\ | \geq |\ I'\ | + |A(I - I')| \geq |\ I'\ | + |I - I'| = |\ I\ |$.

Structura garfurilor p-conexe

Lema 1. Fie $G = (V, E)$ p-conex, $|\ V\ | \geq p + 1$, $U \subseteq V$, $|\ U\ | = p$ şi $x \in V - U$. Există în G p xU – drumuri cu singurul vârf comun x.

Demonstraţie: Consideram graful $G' = (V \cup \{z\}, E')$, unde $E' = E \cup \{zy \mid y \in U\}$. G' este p-conex . În adevăr, oricare A cu $|\ A\ | \leq p - 1$ $G' - A$ este conex (dacă $z \in A$, acest lucru este evident din p-conexiunea lui G; dacă $A \subseteq V$ atunci $G' - A$ este conex întrucât $G - A$ este conex şi $\exists\ y \in U$ cu $zy \in E(G' - A)$).
Lema rezultă, aplicând teorema 1' grafului G' şi perechii x, z.

Lema 2. Dacă $G = (V, E)$ este un graf p-conex $p \geq 2$, atunci $\forall$ doua muchii e_1 şi e_2 şi $p - 2$ vârfuri $x_1, \ldots, x_{p-2}$ există un circuit în G care le conţine.

Demonstraţie: Inducţie după p.

Dacă $p = 2$, trebuie să dovedim că în orice graf **2**-conex, prin orice două muchii trece un circuit. Consideram G' obţinut din G prin inserţia câte unui vârf pe muchiile $e_1(a)$ şi $e_2(b)$. Noul graf este tot **2**-conex, deoarece orice vârf am scoate, nu se pierde conexiunea. Există deci în G' două ab – drumuri disjuncte, care în G oferă un circuit ce conţine e_1 şi e_2.

Fie $p > 2$ şi presupunem afirmaţia adevărată pentru orice graf k-conex, $k < p$. Fie G p-conex.

Putem presupune ca extremităţile muchiilor e_1 şi e_2 nu sunt printre $x_1, \ldots, x_{p-2}$, deoarece altfel, afirmaţia ar rezulta prin inducţie. Graful $G - x_{p-2}$ este $(p - 1)$-conex. Conform ipotezei inductive există un circuit C ce conţine $x_1, \ldots, x_{p-3}$ şi e_1, e_2. Fie Y mulţimea vârfurilor lui C, $|\ Y\ | \geq p$. Folosind lema 2, există în G p $x_{p-2}y$ drumuri cu $y \in Y$, disjuncte. Putem presupune că pentru un drum $x_{p-2}y$, y este primul vârf din Y întâlnit, aşa că aceste drumuri au câte un singur vârf comun cu Y. Dăm o orientare circuitului şi numerotăm vârfurile sale conform acestei orientări. Avem deci drumurile
$D_{x_{p-2}y_1}, D_{x_{p-2}y_2}, \ldots, D_{x_{p-2}y_p}$. Vârfurile $y_1, \ldots, y_p$ descompun circuitul în drumurile $D_{y_1y_2}, D_{y_2y_3}, \ldots, D_{y_{p-1}y_p}, D_{y_py_1}$.

Există un drum dintre acestea, în care nu este conţinut nici unul de elementele $x_1, \ldots, x_{p-3}, e_1, e_2$. Fie acest drum $D_{y_1y_2}$; atunci

$D_{x_{p-2}y_2}, D_{y_2y_3}, \ldots, D_{y_{p-1}y_p}, D_{y_px_{p-2}}$ este un circuit ce conţine $x_1, x_2, \ldots, x_{p-2}, e_1$

şi e_2, şi lema este demonstrată.

Teorema 4. (Dirac 1953) Dacă $G = (V, E)$ este un graf p-conex $p \geq 2$, atunci prin orice p vârfuri ale sale trece un circuit.

Demonstraţie. Fie $x_1, \ldots, x_{p-1}, x_p$ p – vârfuri oarecare a lui G. Deoarece graful G este conex, există $e_1 = xx_{p-1}$ şi $e_2 = yx_p$ şi aplicam lema 3.

Aplicam această teoremă precum şi idea utilizată în demonstraţia lemei 2, pentru a demonstra o condiţie suficienta de hamiltonietate interesantă datorată lui Erdos şi Chvatal (1972).

Teorema 5. Fie G p-conex. Dacă $\alpha(G) < p$ atunci G este hamiltonian.

Demonstraţie: Presupunem că G nu este hamiltonian. Vom obţine o contradicţie. Cum G este p-conex , $\exists$ un circuit de lungime cel puţin p (conform teoremei lui Dirac de mai sus). Fie C un circuit de lungime maxima în G. Dacă G nu este hamiltonian, $\exists$ $v \notin C$. Întrucât $|C| \geq p$, conform lemei 2, există p vC-drumuri disjuncte (cu excepţia lui v) fie ele $D_{vv_1}, D_{vv_2}, \ldots, D_{vv_p}$ (numerotarea vârfurilor este în ordinea întâlnirii lor într-o parcurgere fixată a circuitului). Notăm, pentru fiecare v_i cu w_i vârful succesor al lui v_i în parcurgerea lui C.

Atunci, $vw_i \notin E$, căci altfel am avea circuitul vw_i, w_i, $C - w_iv_i$, D_{v_iv} de lungime mai mare decât C. Cum $\alpha(G) \leq p$, atunci mulţimea $\{v, w_1, w_2, \ldots, w_p\}$ nu este stabila. Deci, există $w_sw_t \in E$. Dar atunci: D_{vv_s} , drumul (invers) pe C de la v_s la w_t , muchia w_tw_s , drumul (invers) pe C de la w_s la v_t şi D_{v_tv} este un circuit de lungime mai mare decât C, contrazicând ipoteza că C este de lungime maxima.