# Capitolul 4
# PARCURGERI DE GRAFURI

Problema parcurgerii unui digraf G = (N,A), N = {1, ... , n}, A = {1, ... , m} are următoarea formulare: să se genereze mulțimea $W \subset N$ a nodurilor y pentru care există drum de la un nod sursă dat s la nodul y în digraful G. Dacă există drum, în digraful G, de la nodul sursă s la nodul y atunci se spune că nodul y este accesibil din nodul s.

Algoritmii pe care îi vom prezenta pentru rezolvarea problemei parcurgerii unui digraf G sunt metode sistematice de vizitare a nodurilor y accesibile din s. Fiecare iterație a execuției oricărui algoritm de parcurgere stabileşte pentru fiecare nod apartenența la una din următoarele trei stări:

- nevizitat;
- vizitat şi neanalizat, adică un nod vizitat ai cărui succesori au fost parțial vizitați;
- vizitat si analizat, adică un nod vizitat ai cărui succesori au fost în totalitate vizitați.

Dacă nodul x este vizitat si neanalizat, există arcul (x,y) şi nodul y este nevizitat, atunci se poate vizita nodul y. În acest caz se spune că arcul (x,y) este arc *admisibil* şi dacă nodul y este vizitat explorând arcul (x,y) se spune că nodul x este *predecesorul parcurgere* al nodului y.

Se vor prezenta următorii algoritmi pentru parcurgerea unui digraf: algoritmul BF şi algoritmul DF. Aceşti algoritmi utilizează următoarele notații comune:

U mulțimea nodurilor nevizitate;
V mulțimea nodurilor vizitate şi neanalizate;
W mulțimea nodurilor vizitate şi analizate;
p tabloul predecesor, care este unidimensional cu n elemente.

## 4.1. Parcurgerea BF a grafurilor

Parcurgerea se face "mai întâi în lățime". În engleză "breadth first" (**BF**).

Fie digraful G = (N,A) cu nodul sursă s şi $\boldsymbol{D}_x$ mulțimea drumurilor de la nodul sursa s la nodul $x \in \mathbf{N}$. Numărul de arce ce compun un drum $D_x \in \boldsymbol{D}_x$ defineşte lungimea acestui drum pe care îl notăm $l(D_x)$. Distanța de la nodul s la nodul x se defineşte în modul următor:

$$d(x) = \begin{cases} \min\{l(D_x) \mid D_x \in \boldsymbol{D}_x\}, & \boldsymbol{D}_x \neq \emptyset \\ \infty, & \boldsymbol{D}_x = \emptyset \end{cases}$$

Un drum $\hat{D}_x \in \boldsymbol{D}_x$ cu $l(\hat{D}_x) = d(x)$ se numeşte *cel mai scurt drum* de la nodul sursă s la nodul x.

**Observația 1**. Pentru orice arc $(x,y) \in A$ avem

$$d(y) \leq d(x) + 1.$$

Într-adevăr, dacă $\boldsymbol{D}_x = \emptyset$ atunci $d(x) = \infty$ şi inegalitatea se păstrează. Dacă $\boldsymbol{D}_x \neq \emptyset$ atunci evident $\boldsymbol{D}_y \neq \emptyset$. Fie $\hat{D}_x$ un cel mai scurt drum de la s la x, deci $l(\hat{D}_x) = d(x)$. Mulțimea $\boldsymbol{D}_y$ poate conține un drum $D_y$, astfel încât $l(D_y) < d(x) + 1$. Conform definiției distanței avem $d(y) \leq l(D_y)$ şi rezultă că $d(y) < d(x) + 1$. Avem egalitate când un drum cel mai scurt $\hat{D}_y$ de la s la y are lungimea

$$d(y) = l(\hat{D}_y) = d(x) + 1,$$

de exemplu

$$\hat{D}_y = \hat{D}_x \cup \{(x,y)\}.$$

În algoritmul parcurgerii BF (algoritmul PBF) se utilizează tabloul lungime l care este unidimensional şi are n elemente. Mulțimea nodurilor vizitate şi neanalizate V este organizată, ca structură de date, ca o coadă.

Algoritmul PBF este următorul:

```
(1)    PROGRAM PBF;
(2)    BEGIN;
(3)           U := N - {s} ; V := {s} ; W := ∅;
(4)           FOR toți y∈N DO p(y) := 0;
(5)           l(s) := 0;
(6)           FOR toți y∈U DO l(y) := ∞;
(7)           WHILE V ≠∅ DO
(8)           BEGIN
(9)                  se selectează cel mai vechi nod x introdus In V;
(10)                 FOR (x, y) ∈ A DO
(11)                        IF y ∈ U
(12)                              THEN U:=U-{y}; V:=V∪{y}; p(y):=x:
                                  l(y) := l(x) + 1;
(13)                 V:=V-{x}; W:=W∪{x};
(14)          END;
(15)   END.
```

**Teorema 1 .**

(1) Algoritmul PBF calculează elementele tabloului l astfel încât $d(y) \leq l(y)$, $y \in \mathbf{N}$;

(2) Dacă la iterația k oarecare a algoritmului PBF avem $V = (x_1, ..., x_r)$ în această ordine, atunci $l(x_r) \leq l(x_1) + 1$ şi $l(x_i) \leq l(x_{i+1})$, $i = \overline{1, r-1}$.

**Demonstrație.**

(1) Utilizăm inducția după k, numărul de iterații ale ciclului WHILE. Inițial l(s) := 0, l(y) := ∞ pentru $y \in U$ şi evident $d(y) \le l(y), y \in N$. Presupunem că, la iterația k avem $d(y) \le l(y)$ pentru $y \in N$. La iterația k + 1 pot exista cazurile:

(Cl ) Există arc (x,y) admisibil ($(x,y) \in A$ şi $y \in U$). În acest caz

$$l(y) = l(x) + 1$$

şi

$$d(y) \le d(x) + 1 \le l(x) + 1 = l(y)$$

(l(x) pentru $x \in V$ nu se modifică). Deci pentru toate arcele $(x,y) \in A$ şi $y \in U$ avem $d(y) \le l(y)$. Pentru celelalte noduri y, conform ipotezei inducției, avem $d(y) \le l(y)$, deoarece la iterația k + 1, l(y) nu se mai modifică.

(C2) Nu există arc (x,y) admisibil ($(x,y) \notin A$ sau $y \notin U$). În acest caz la iterația k + 1 nu se modifică nici un element l(y) şi conform ipotezei inducției $d(y) \le l(y)$ pentru $y \in N$.

(2) Utilizăm inducția după k numărul de iterații ale ciclului WHILE. Inițial V := {s}. Deci $x_1 = a$, $x_r = a$ şi l(s) < l(s) + 1, l(s) = l(s). Presupunem că la iterația k avem $l(x_r) \le l(x_1) + 1$ şi $l(x_i) < l(x_{i+1})$, pentru $i = \overline{1, r-1}$. La iterația k + 1 pot exista cazurile:

(Cl ) Există arc (x,y) admisibil ($(x,y) \in A$ şi $y \in U$). În acest caz $V = \{x_1, ..., x_r, x_{r+1}\}$, $x_1 = x$, $x_{r+1} = y$. Astfel, $l(x_{r+1}) = l(y) = l(x) + 1 = l(x_1)+1$. De asemenea, avem $l(x_r) \le l(x_r) + 1 = l(x) + 1 = l(y) = l(x_{r+1})$ şi inegalitățile $l(x_i) \le l(x_{i+1})$, $i = \overline{1, r-1}$ au rămas nemodificate.

(C2) Nu există arc (x,y) admisibil ($(x,y) \notin A$ sau $y \notin U$). În acest caz $V = \{x_2, ..., x_r\}$. Avem $l(x_r) \le l(x_1) + 1 \le l(x_2) + 1$ şi inegalitățile $l(x_i) \le l(x_{i+1})$, $i = \overline{1, r-1}$ au rămas nemodificate.

**Teorema 2.** Algoritmul PBF este convergent şi la terminarea execuției determinăm mulțimea tuturor nodurilor care sunt accesibile din nodul sursă s în digraful G = (N,A).

**Demonstrație.** Din liniile (10), (11) şi (12) ale algoritmului rezultă că toate nodurile introduse în V sunt eliminate după ce sunt analizate. Deci după un număr finit de iterații se obține V = ∅ şi execuția algoritmului se oprește. Pentru a arăta că la terminarea execuției, algoritmul determină mulțimea tuturor nodurilor care sunt accesibile din nodul sursă s în digraful G = (N,A), trebuie să arătăm că la terminarea execuției algoritmului mulțimea W este:

$$W = \{y \mid y \in N \text{ şi există drum de la s la } y\}.$$

Din liniile (10), (11) şi (12) ale algoritmului rezultă că în V sunt

introduse numai noduri y care sunt accesibile din s şi că după ce un nod $x \in V$ a fost analizat el este eliminat din V şi introdus în W. Deoarece algoritmul se opreşte când $V = \emptyset$ rezultă că W conţine toate nodurile $y \in N$ care sunt accesibile din s şi introduse în V. Să arătăm că W conţine toate nodurile $y \in N$ care sunt accesibile din s. Prin reducere la absurd să presupunem că există un drum $D = (y_1, y_2, \ldots, y_{k-1}, y_k)$ cu $y_i = s$, $y_k = y$ în G şi $y \notin W$. Rezultă că $y_k \notin V$. Deoarece $y_k \notin V$ şi $(y_{k-h}, y_k) \in A$ deducem că $y_{k-1} \notin V$, astfel $y_k$ ar fi fost introdus în V. Continuând procedeul vom deduce în final că $s = y_1 \notin V$. Aceasta contrazice faptul că în linia (3) a algoritmului iniţializăm V:={s}. Rezulta că $y \in V$, deci $y \in W$ şi teorema este demonstrată.

**Teorema 3.**

Algoritmul PBF este convergent şi determină:

(1) mulţimea tuturor nodurilor care sunt accesibile din nodul sursă s;

(2) elementele tabloului l astfel încât $l(y) = d(y)$ pentru $y \in N$.

**Demonstraţie.** Convergenţa şi punctul (1) se demonstrează la fel ca Teorema 2. Punctul (2) se demonstrează prin inducţie după k numărul iteraţiilor ciclului WHILE. Fie mulţimea $N_k = \{y \in N \mid d(y) = k\}$. Pentru $k = 0$ avem $N_0 = \{s\}$ şi deci $d(s) = l(s) = 0$. Presupunem afirmaţia adevărată pentru k. Afirmaţia pentru $k + 1$ rezultă cu uşurinţă, deoarece, în conformitate cu Teorema 1 punctul (2), un nod $y \in N_{k+1}$ este vizitat plecând de la un nod $x \in N_k$ numai după ce toate nodurile din $N_k$ sunt vizitate. Deci, dacă $y \in N_{k+1}$ şi este vizitat explorând arcul (x,y), $x \in N_k$, atunci,

$$l(y) = l(x) + 1 = d(x) + 1 = k + 1 = d(y).$$

**Teorema 4**.

Algoritmul PBF are complexitatea O(m).

**Demonstraţie.** Din liniile (10) , (11) şi (12) ale algoritmului rezultă că fiecare nod al digrafului G este introdus şi eliminat din V cel mult o dată. Deoarece execuţia algoritmului se termină când $V = \emptyset$ deducem că algoritmul execută cel mult 2n iteraţii. Fiecare arc $(x,y) \in A$ este explorat cel mult o dată pentru identificarea arcelor admisibile. Deci complexitatea algoritmului PBF este $O(m + n) = O(m)$.

În parcurgerea BF, dacă $N_p = W$ şi subgraful predecesor $G_p = (N_p, A_p)$ este o arborescenţă atunci $G_p$ se numeşte arborescenţă parcurgere BPF.

**Teorema 5**.

Algoritmul PBF determină elementele tabloului p astfel încât subgraful predecesor $G_p = (Np, Ap)$ este o arborescenţă parcurgere BPF.

**Demonstraţie.** Din liniile (10), (11) şi (12) ale algoritmului rezultă că $p(y) := x$ numai dacă y este accesibil din s. Evident că la terminarea execuţiei algoritmului avem $N_p = W$. Din modul cum sunt definite $N_p$ şi $A_p$

rezultă că $G_p$ este o arborescenţă. Deci subgraful predecesor $G_p$ este o arborescenţă parcurgere a digrafului G = (N,A).

**Observaţia 2.** Drumul unic de la nodul sursă s la un nod y din arborescenţa parcurgere BF este un cel mai scurt drum de la nodul sursă s la acelaşi nod y din digraful G, conform punctului (2) al Teoremei 3.

**Observaţia 3.** Algoritmul PBF sau PTBF se poate aplica şi grafurilor neorientate. În acest caz, subgraful predecesor $G_p = (N_p,A_p)$ este o arborescenţă sau mai multe arborescenţe.

**Observaţia 4.** Un drum unic de la nodul sursă s la un nod y din arborescenta parcurgere BF se poate determina cu următoarea procedură.

```
(1)    PROCEDURA DRUM (G,s,y);
(2)    BEGIN;
(3)           se tipăreşte u;
(4)           WHILE p (y)≠i-0 DO
(5)           BEGIN
(6)                  x := p(y); se tipăreşte x;
(7)                         IF x≠s
(8)                                THEN y := x
(9)                                ELSE EXIT;
(10)          END;
(11)   END.
```

**Observaţia 5.** Mulţimea W este în general submulţimea mulţimii nodurilor N. Pentru parcurgerea întregului digraf G = (N,A) se utilizează algoritmul parcurgerii totale generice (algoritmul PTG). Algoritmul PTG este următorul:

```
(1)    PROGRAM PTG;
(2)    BEGIN;
(3)           U := N - {s}; V :== {s}; W := ∅;
(4)           FOR toţi y∈N DO p(y) := 0;
(5)           k:= 1; o(s):= 1;
(6)           FOR toţi y∈U DO o(y) := ∞;
(7)           WHILE W≠N DO
(8)           BEGIN
(9)                  WHILE V ≠∅ DO
(10)                 BEGIN
(11)                        se selectează un nod x din V;
(12)                        IF există arc (x, y)∈A şi y∈U
(13)                               THEN   U:=U-{y}; V:=V∪{y};p(y):=x;
                                          k := k + 1; o(y) :=k
(14)                               ELSE V:=V-{x}; W:=W∪{x};
(15)                 END;
(16)                 se selectează       s∈U; U:= U - {s}; V:= {s};
                                  k := k + 1: o(S) := k;
(17)          END;
(18)   END.
```

Este evident că algoritmul PTG are complexitatea tot O(m) şi că vectorul p determină una sau mai multe arborescente parcurgeri.

**Exemplu 1.**
Se aplică algoritmul PBF digrafului din figura 1 .

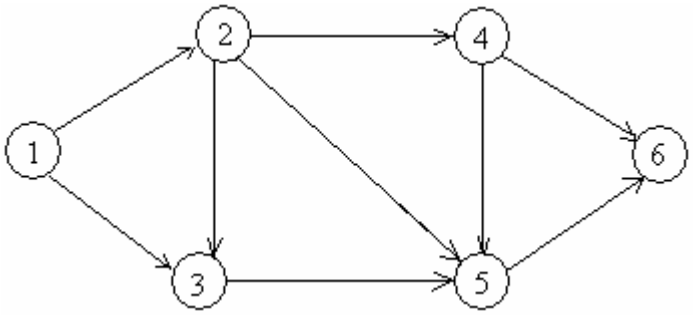


Figura 1

Inițializări:

$s = 1$, $U = \{2,3,4,5,6\}$,
$V = \{1\}$, $W = \emptyset$, $p = (0,0,0,0,0,0)$, $l = (0, \infty, \infty, \infty, \infty, \infty)$.

Iterația 1:

$x = 1$, $(1,2) \in A$, $2 \in U$: $U = \{3,4,5,6\}$,
$V = \{1,2\}$, $p = (0,1,0,0,0,0)$ ,
$l = (0,1, \infty, \infty, \infty, \infty)$; $(1,3) \in A$, $3 \in U$: $U = \{4,5,6\}$,
$V = \{1,2,3\}$, $p = (0,1,1,0,0,0)$,
$l = (0,1, 1, \infty, \infty, \infty)$; $V = \{2, 3\}$; $W = \{1\}$.

Iterația 2:

$x = 2$, $(2,4) \in A$, $4 \in U$: $U = \{5,6\}$, $V = \{2,3, 4\}$,
$p = (0,1,1,2,0,0)$, $l = (0,1,1,2, \infty, \infty)$; $(2,5) \in A$,
$5 \in U$ : $U = \{6\}$, $V = \{2,3,4,5\}$ , $p = (0,1,1,2,2,0)$ ,
$l = (0,1,1,2,2, \infty)$ ; $V = \{3, 4, 5\}$ ; $W = \{1, 2\}$.

Iterația 3:

$x = 3$, $V = \{4, 5\}$, $W = \{1, 2, 3\}$.

Iterația 4:

$x = 4$, $(4, 6) \in A$, $6 \in U$: $U = \emptyset$, $V = \{4, 5,6\}$,
$p = (0,1,1,2,2,4)$, $l = (0,1,1,2,2,3)$; $V = \{5,6\}$;
$W = \{1,2,3,4\}$.

Iterația 5:

$x = 5$, $V = \{6\}$, $W = \{1,2,3,4,5\}$.

Iterația 6:

$x = 6$, $V = \emptyset$, $W = \{1,2,3,4,5,6\}$.
$N_p = \{1,2,3,4,5,6\} = W$.

Arborescența parcurgere BF este prezentată în figura 2.

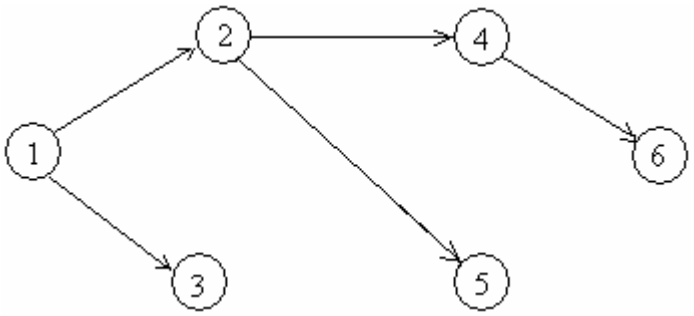

Figura 2

Drumul unic de la nodul 1 la nodul 6 se obține cu PROCEDURA DRUM în modul următor:

y = 6 este ultimul nod al drumului;
Iterația 1: x = p(6) = 4, y = 4.
Iterația 2: x = p(4) = 2, y = 2.
Iterația 3: x = p(2) = 1, y = l.
Drumul este: 1,2,4,6.

## 4.2. Parcurgerea DF a grafurilor

Parcurgerea se face "mai întâi în adâncime". În engleză "depth first" (**DF**).

În algoritmul parcurgerii DF (algoritmul PDF) se folosesc aceleaşi notații ea în algoritmul PBF cu deosebirea că, în locul tabloului unidimensional de lungime l se utilizează tablourile timp unidimensionale $t_1$ şi $t_2$ care au fiecare n elemente. Mulțimea nodurilor vizitate şi neanalizate V este organizată ca structură de date, ca stivă.

Algoritmul PDF este următorul:

```
(1)    PROGRAM PDF;
(2)    BEGIN;
(3)          U := N-{s}; V := {s}; W := ∅;
(4)          FOR toți y∈N DO p(y) := 0;
(5)          t := 1; t1(s) := 1; t2(s) := ∞;
(6)          FOR toți y∈U DO t1(y) := ∞; t2(y) := ∞;
(7)          WHILE V≠∅ DO
(8)          BEGIN
(9)                 se selectează cel mai nou nod x introdus in V;
(10)                IF există arc (x,y)∈A şi y∈U
(11)                THEN   U := U - {y}; V := V∪{y}; p (y) := x;
                           t := t + 1; t1(y) := t
(12)                ELSE   V := V-{x}; W := W∪{x}; t:=t+1;t2(x):=t;
(13)         END;
(14)    END.
```

Din liniile (10), (11), (12) ale algoritmului PDF rezultă că elementul $t_1(y)$ reprezintă momentul când y devine nod vizitat şi neanalizat şi elementul $t_2(x)$ reprezintă momentul când x devine nod vizitat şi analizat.

Se va studia în continuare algoritmul parcurgerii totale DF (algoritmul PTDF).

Algoritmul PTDF este următorul:

```
(1)     PROGRAM PTDF;
(2)     BEGIN;
(3)           U:=N-{s}; V:={s}; W:=∅;
(4)           FOR toţi y∈N DO p(y) := 0;
(5)           t:=1; t1(s):= l; t2(s):= ∞;
(6)           FOR toţi y∈U DO t1(y) := ∞; t2(y) := ∞;
(7)           WHILE W≠N DO
(8)           BEGIN
(9)                 WHILE V≠∅ DO
(10)                BEGIN
(11)                      se selectează cel mai nou nod x introdus
în V;
(12)                      IF există arc (x,y)∈A şi y∈U
(13)                            THEN   U := U - {y}; V := V∪{y};
                                       p (y) := x; t := t + 1; t1(y)
:= t;
(14)                            ELSE   V:= V - {x}; W:= W∪{x};
                                       t:= t + 1; t2 (x) := t;
(15)                END;
(16)                se selectează       s∈U; U := U - {s}; V := {s};
                                t:=t+l; t1(s):=t;
(17)          END;
(18)    END.
```

Fie mulţimea $S = \{s \,|\, s \in N$, s selectate în linia (3) şi linia (16)$\}$.

**Teorema 6.**

Algoritmul PTDF este convergent şi determină mulţimile nodurilor accesibile din s, $s \in S$.

**Demonstraţie.** Se demonstrează la fel ca Teorema 2.

**Teorema 7.**

Algoritmul PTDF are complexitatea O(m).

**Demonstraţie.** Evident că. algoritmul PTDF are complexitatea O(m).

Un digraf $\hat{G} = (\hat{N}, \hat{A})$ se numeşte pădure dacă este format din una sau mai multe arborescenţe. Un graf neorientat $\hat{G} = (\hat{N}, \hat{A})$ se numeşte pădure dacă este format din unul sau mai mulţi arbori.

În parcurgerea totală DF, dacă subgraful predecesor $G_p = (N_p, A_p)$, $N_p = \{y \,|\, p(y) \neq 0\} \cup S$, $A_p = \{(p(y), y) \,|\, y \in N_p - S\}$ este o pădure şi $N_p = W$, atunci $G_p$ se numeşte pădure parcurgere DF.

**Teorema 8.**

Algoritmul PTDF determină elementele tabloului p astfel încât subgraful predecesor $G_p = (N_p, A_p)$ este o pădure parcurgere DF.

**Demonstrație.** Se demonstrează analog ca Teorema 5.

**Exemplul 2.**

Se aplică algoritmul PTDF digrafului

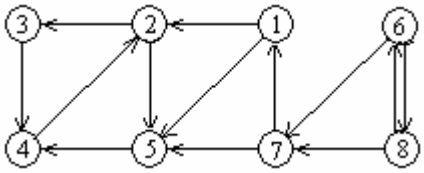


Figura 3

Inițializări:

s = 1,U = {2,3,4,5,6, ,8}, V = {1}, W = ∅ , p

p = (0,0,0,0,0,0,0,0), t = 1,

$t_1 = (1, \infty, \infty, \infty, \infty, \infty, \infty, \infty)$,

$t_2 = (\infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty)$.

Iterația 1:

$x = 1, (1,2) \in A, 2 \in U$: U = {3,4,5,6,7,8}, V = {1,2},

p = (0,1,0,0,0,0,0,0), t = 2,

$t_1 = (1,2, \infty, \infty, \infty, \infty, \infty, \infty)$.

Iterația 2:

$x = 2, (2,3) \in A, 3 \in U$: U = {4,5,6,7,8}, V = {1,2,3},

p = (0,1,2,0,0,0,0,0), t = 3,

$t_1 = (1, 2, 3, \infty, \infty, \infty, \infty, \infty)$.

Iterația 3:

$x = 3, (3,4) \in A, 4 \in U$: U = {5,6,7,8}, V = {1,2,3,4},

p = (0,1,2,3,0,0,0,0), t = 4, $t_1 = (1,2,3,4, \infty, \infty, \infty, \infty)$.

Iterația 4:

x = 4: V = {1,2, 3}, W = {4}, t = 5,

$t_2 = (\infty, \infty, \infty, 5, \infty, \infty, \infty, \infty)$.

Iterația 5:

x = 3: V = {1,2}, W = {4,3}, t = 6,

$t_2 = (\infty, \infty, 6, 5, \infty, \infty, \infty, \infty)$.

Iterația 6:

$x = 2, (2,5) \in A, 5 \in U$: U = {6,7,8},

V = {1,2,5}, p = (0,1,2,3,2,0,0,0), t = 7,

$t_1 = (1,2,3,4,7, \infty, \infty, \infty)$.

Iterația 7:

x = 5: V = {1,2}, W = {4,3,5}, t = 8,

$t_2 = (\infty, \infty, 6, 5, 8, \infty, \infty, \infty)$.

Iterația 8:

$x = 2 : V = \{1\}, W = \{4,3,5,2\}, t = 9,$

$t_2 = (\infty, 9, 6, 5, 8, \infty, \infty, \infty).$

Iterația 9:

$x = 1: V = 0, W = \{4,3,5,2,1\}, t = 10,$

$t2 = (10,9,6,5,8, \infty, \infty, \infty).$

Actualizări:

$s = 6, U = \{7,8\}, V = \{6\}, t = 11, t_1 = (1,2,3,4,7,11, \infty, \infty)$

Iterația 10:

$x = 6, (6, 7) \in A, 7 \in U : U = \{8\}, V = \{6, 7\},$

$p = (0,1,2,3,2,0,6,0), t = 12,$

$t_1 = (1,2,3,4,7,11,12, \infty).$

Iterația 11:

$x = 7 : V = \{6\}, W = \{4, 3,5,2,1,7\}, t = 13,$

$t_2 = (10,9,6,5,8,00,13, \infty).$

Iterația 12:

$x = 6, (6,8) \in A, 8 \in U: U = \emptyset, V = \{6,8\}, p = (0,1,2,3,2,0,6,6),$

$t = 14, t_1 = (1,2,3,4,7,11,12,14).$

Iterația 13:

$x = 8 : V = \{6\}, W = \{4, 3, 5, 2,1, 7,8\}, t = 15,$

$t_2 = (10,9,6,5,8, \infty, 13,15).$

Iterația 14:

$x = 6: V = \emptyset, W = \{4,3,5,2,1,7,8,6\}, t = 16,$

$t_2 = (10,9,6,5,8,16,13,15).$

## 4.3. Aplicații

### 4.3.1. Sortarea topologică

**Teorema 9.** Un digraf G = (N,A) este fără circuite dacă şi numai dacă orice parcurgere totală DF a lui G nu produce arce de revenire.

**Demonstrație.** Presupunem că digraful G este fără circuite. Prin reducere la absurd presupunem ca o parcurgere totală DF a lui G produce arce de revenire ($R \neq \emptyset$). Fie arcul $(z, x) \in R$. În acest caz nodul z este un descendent al nodului x în pădurea parcurgere DF. Deci există un drum D de la x la z în G. Atunci $\overset{\circ}{D} = D \cup \{z,x\}$ este un circuit în G şi aceasta contrazice ipoteza că digraful G este fără circuite.

Reciproc, presupunem că o parcurgere totală DF a digrafului G nu produce arce de revenire ($R \neq \emptyset$). Prin reducere la absurd presupunem că G conține un circuit $\overset{\circ}{D}$. Fie x primul nod vizitat din $\overset{\circ}{D}$ şi fie arcul $(z,x) \in \overset{\circ}{D}$. Deoarece nodurile $x, z \in \overset{\circ}{D}$ rezultă că există un drum D de la x

la z. De asemenea x fiind primul nod vizitat din $\overset{\circ}{D}$ rezultă că nodul z devine un descendent al nodului x la pădurea PDF. Deci (z,x) este un arc de revenire ce contrazice ipoteza $R = \emptyset$.

Sortarea topologică a unui digraf G = (N,A) fără circuite constă într-o ordonare liniară a nodurilor din N astfel încât dacă $(x, y) \in A$ atunci x apare înaintea lui y în ordonare.

Algoritmul sortare topologică (algoritmul ST) se obține din algoritmul PTDF făcând următoarele două completări:

(1) în partea de inițializări (liniile (3)-(6)) se inițializează o listă a nodurilor;

(2) în linia (16) după calculul lui $t_2(X)$, nodul x se introduce la începutul listei.

La terminarea algoritmului ST, lista furnizează sortarea topologică a digrafului G = (N,A) fără circuite şi nodurile x sunt plasate în listă în ordinea descrescătoare a timpilor $t_2(X)$.

## 4.3.2. Componentele conexe ale unui graf

**Definiția 1.** Un digraf G = (N,A) se numeşte conex dacă pentru oricare două noduri x, y există un lanț care are aceste două noduri drept extremități.

Noțiunea de conexitate are sens şi pentru grafuri neorientate.

**Definiția 2.** Se numeşte componentă conexă a unui digraf G = (N,A) un subgraf G' = (N',A') al lui G, care este conex şi care este maximal în raport cu incluziunea față de această proprietate (oricare ar fi $x \in \overline{N}' = N - N'$, subgraful $G'_x$ generat de $N'_x = N' \cup \{x\}$ nu mai este conex).

O componentă conexă G' = (N',A') a unui digraf G = (N,A) se poate identifica cu mulțimea N' care generează subgraful G'.

Deoarece în problema conexiunii sensul arcelor nu contează se va considera că grafurile sunt neorientate. Dacă G = (N,A) este digraf atunci i se asociază graful neorientat $\hat{G} = (\hat{N}, \hat{A})$, unde $\hat{N} = N$, $\hat{A} = \{[x, y] \mid (x, y) \in A$ şi / sau $(y, x) \in A\}$. Este evident că G şi $\hat{G}$ au aceleaşi componente conexe.

Algoritmul componentelor conexe (algoritmul CC) este o adaptare a algoritmului PTDF aplicat unui graf neorientat G = (N,A). Nu se calculează tablourile timp $t_1$, $t_2$ şi prin p notăm o variabilă scalară a cărei valoare reprezintă numărul componentelor conexe. Algoritmul CC este următorul:

```
(1)     PROGRAM CC;
(2)     BEGIN
(3)             U:= N - {s}; V:= {s}; W:= ∅; p:= 1; N' = {s};
(4)             WHILE W ≠ N DO
(5)             BEGIN
```

```
(6)                 WHILE V≠∅ DO
(7)                 BEGIN
(8)                   se selectează cel mai nou nod x introdus în V;
(9)                      IF există muchie [x,y]∈A şi y∈V
(10)                          THEN U:= U-{y}; V:=V∪{y};N':=N'∪{y}
(11)                          ELSE V:= V-{x}; W:=W∪{x};
(12)                END;
(13)                  se tipăresc p şi N';
(14)                  se selectează s∈U; U:=U-{s}; V:={s};
                                         p:= p+1;N':={s};
(15)          END;
(16)   END.
```

La terminarea algoritmului pot exista cazurile:

(Cl) se tipăreşte o singură componentă conexă şi în acest caz graful G = (N,A) este conex;

(C2) se tipăresc mai multe componente conexe şi în acest caz graful G = (N,A) nu este conex, obținându-se toate componentele conexe ale lui G.

**Teorema 10.**

Algoritmul CC determină componentele conexe ale unui graf neorientat G = (N,A).

**Demonstrație.**

La terminarea execuției ciclului WHILE se determină mulțimea N' a tuturor nodurilor accesibile printr-un lanț cu aceeaşi extremitate, nodul s. Mulțimea N' generează evident o componentă conexă G' = (N',A').

Deoarece la terminarea execuției algoritmului avem W = N rezultă că algoritmul CC determină toate componentele conexe ale lui G = (N,A).

**Teorema 11**.

Algoritmul CC are complexitatea O(m).

**Demonstrație.** Algoritmul CC are aceeaşi complexitate cu a algoritmului PTDF, adică O(m).

**Exemplul 3**.

Fie digraful din figura 4.

Pentru a determina componentele conexe ale acestui digraf se transformă într-un graf neorientat reprezentat în figura 5 căruia i se aplică algoritmul CC.

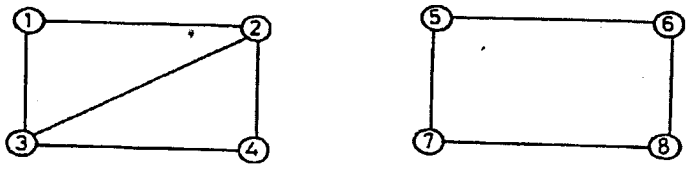


Figura 4

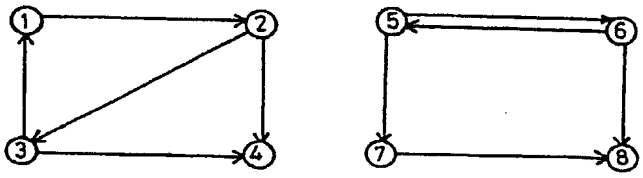


Figura 5

Inițializări:

s = 1, U = {2,3,4,5,6,7,8}, V = {1}, W = ∅ , p = 1, N'={1}.

Iterația 1:

x = 1, [1,2] ∈ A, 2 ∈ U : U = {3,4,5,6,7,8}, V = {1,2}, N'={1,2}.

Iterația 2:

x = 2, [2,3] ∈ A, 3 ∈ U: U={4,5,6,7,8}, V={1,2,3}, N'={1,2,3}.

Iterația 3:

x=3, [3,4] ∈ A, 4 ∈ U: U={5,6,7,8}, V={1,2,3,4}, N'={1,2,3,4}.

Iterația 4:

x = 4: V = {1,2,3}, W = {4}.

Iterația 5:

x = 3: V = {1,2}, W = {4,3}.

Iterația 6:

x = 2, V = {1}, W = {4,3,2}.

Iterația 7:

x = 1 : V = ∅ , W = {4, 3, 2, 1}.

Se tipăresc:

p = 1 şi N' = {1,2,3,4}

Actualizări:

s = 5, U = {6,7,8}, V = {5}, p = 2, N' = {5}.

Iterația 8:

x = 5, [5,6] ∈ A, 6 ∈ U : U = {7,8}, V = {5,6}, N' = {5,6}.

Iterația 9:

x = 6, [6,8] ∈ A, 8 ∈ U : U = {7}, V = {5,6,8}, N' = {5,6,8}.

Iterația 10:

x = 6, [8, 7] ∈ A, 7 ∈ U: U = ∅ , V = {5,6,8,7}, N'={5,6,8,7}.

Iterația 11:

x = 7 : V = {5,6,8}, W = {4,3,2,1,7}.

După încă trei iterații se obține

V = ∅ , W = {4,3,2,1,7,8,6,5}

Se tipăresc

p = 2 şi N' = {5,6,8,7}

şi execuția algoritmului se opreşte.