Metode de explorare a grafurilor.

Explorarea (sau *traversarea*) unui graf este o metodă sistematică de parcurgere, prin examinarea muchiilor şi vârfurilor. O traversare eficientă are loc în timp liniar **O(n+m)**.

Traversarea în adâncime a unui graf (DFS- Depth First Search).

In cazul grafurilor neorientate, traversarea în adâncime este o metodă eficientă pentru:

- găsirea unei căi între două vârfuri
- a determina dacă un graf este conex
- determinarea arborelui de acoperire a unui graf conex

Prin traversarea DFS a unui graf neorientat se obține un *arbore de acoperire în adâncime*.

In acest scop:

- se porneşte dintr-un vârf **s** (*vârf de start*). Inițial *vârful curent* **u** va fi vârful de start. Fie **(u, v)** o muchie incidentă în vârful curent **u**. Sunt posibile 2 situații:
  - vârful **v** nu a fost încă explorat (vizitat), caz în care:
    - ✓ se marchează vârful ca vizitat
    - ✓ se continuă explorarea din **v**
  - vârful **v** a fost deja vizitat, situație în care se revine în **u** şi se încearcă vizitarea unui alt vârf adiacent lui **u**
  - dacă s-au vizitat toate vârfurile adiacente lui **u**, se face un pas înapoi, alegând alt vârf curent **u**, cu muchii neexplorate, Procesul se încheie când ne întoarcem în **s**.

Muchia explorată din **u**, care conduce la un vârf nedescoperit este o *muchie de arbore*.
O muchie din **u**, care conduce la un vârf deja vizitat, este o *muchie de revenire*. Ea indică prezența unui ciclu.

Fie o traversare DFS într-un graf neorientat **G**, pornind din vârful de start **u**:

- ❑ traversarea vizitează toate vârfurile componentei conexe a lui **s**
- ❑ muchiile de arbore formează un arbore de acoperire a componentei conexe a lui **s**.

In traversare, funcția DFS este apelată o singură dată pentru fiecare vârf, a.î. fiecare muchie este examinată de 2 ori, câte o dată pentru fiecare extremitate. Complexitatea algoritmului va fi aşadar **O(n+m)**.

Următorii algoritmi se bazează pe traversare în adâncime (DFS) şi au aceeaşi complexitate:

1) test conexitate graf
2) determinare arbore de acoperire pentru graf conex
3) determinare componente conexe ale grafului
4) determinarea unei căi între două vârfuri
5) determinarea unui ciclu (sau determinarea inexistenței ciclurilor).

Traversarea în adâncime a unui graf neorientat clasifică muchiile grafului în:

- muchii de arbore
- muchii de revenire

algoritm DFS(u)

```
for  fiecare muchie e incidentă în u
  v = cealalta extremitate a lui e;
  if  v nevizitat
    e->tip = arbore;
    DFS(v);
  else
    e->tip = revenire;
```

Arborele de acoperire în adâncime este păstrat prin subgraful predecesor $G_p = (V, E_p)$ în care:

$E_p = \{(pred(v), v): v \in V \cap pred(v) \neq 0\}$

Se observă că E_p păstrează muchiile de arbore.

```
void DFSV(Graf G, Varf u){
  Varf v;
  G_SetCol(G, u, gri);
  for(v=primV(G); !dultV(G); v=avansV(G,v))
    if(G_IsV(G,v) && G_IsA(G,u,v) && G_GetCol(G,v)==alb){
      G_SetPred(G, v, u);
      DFSV(G, v);
    }
}
void DFS(Graf G){
  Varf u;
  for(u=primV(G); !dultV(G); u=avansV(G,u))
    if(G_IsV(G,u)){
      G_SetCol(G, u, alb);
      G_SetPred(G,u, 0);
    };
  for(u=primV(G); !dultV(G); u=avansV(G,u))
    if(G_IsV(G,u) && V_GetCol(u)==alb)
      DFSV(G, u);
}
```

Spre deosebire de grafurile neorientate, la care în cursul traversării un vârf poate fi vizitat sau nevizitat, pentru grafuri orientate, se disting 3 stări, identificate prin culori:
`0 = alb` pentru vârf nevizitat
`1 = gri` pentru vârf vizitat, a cărui listă de succesori nu a fost în întregime explorată
`2 = negru` pentru vârf vizitat, cu listă de succesori explorată

Pentru fiecare vârf u se marchează două momente:
`start[u]` = momentul descoperirii vârfului
`stop[u]` = momentul în care s-a explorat lista de succesori ai vârfului **`u`**

Intr-un graf orientat, arcele accesibile din **`s`** se clasifică în:

- *arce de arbore* – care conduc la descoperirea de vârfuri noi
- *arce de revenire* – care leagă un vârf cu un strămoş în arborele DFS
- *arce de înaintare* – care leagă un vârf cu un descendent în arborele DFS
- *arce de traversare* – care leagă vârfuri din arbori DFS diferiți (nu se încadrează în categoriile precedente).

```
int start[2*M], stop[2*M];
int t;
void DFS(Graf G){
  Varf u;
  for(u=primV(G); !dultimV(G); u=avansV(G,u))
    if(G_IsV(G,u)){
      G_SetCol(G, u, alb);
      G_SetPred(G, u, 0);
    };
  t = 0;
  for(u=primV(G); !dultimV(G); u=avansV(G,u))
    if(G_IsV(G,u) && G_GetCol(G, u)==alb)
      DFSV(G, u);
}
```

```
void DFSV(Graf G, Varf u){
  Varf v;
  G_SetCol(G, u, gri);
  start[u] = ++t;
  for(v=primV(G); !dultimV(G); v=avansV(G,v))
    if(G_IsV(G,v) && G_IsA(G,u,v) && G_GetCol(G,v)==alb){
      G_SetPred(G, v, u);
      DFSV(G, v);
    };
  G_SetCol(G, u, negru);
  stop[u] = ++t;
}
```

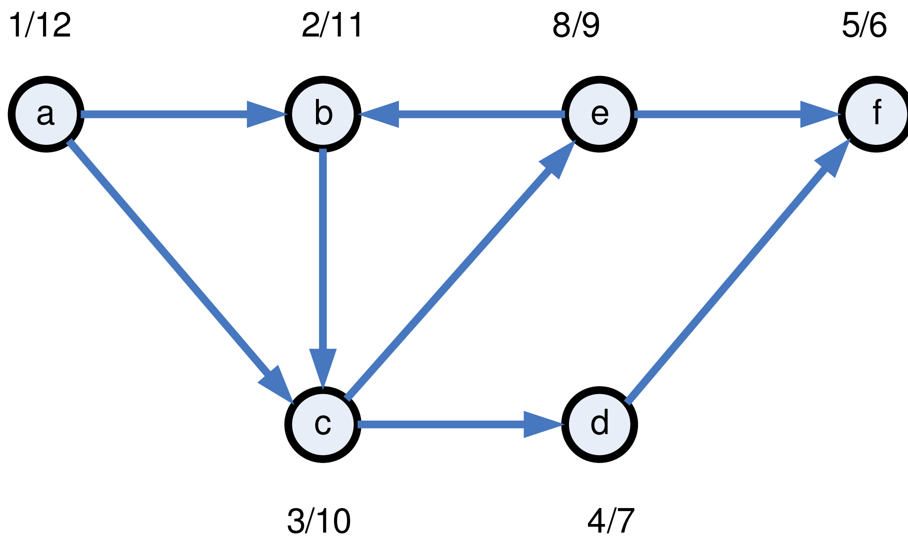
O funcție DFS nerecursivă foloseşte o stivă:

```
DFS(G, s){
  pune nodul de start in stivă;
  cât timp(stivă nevidă){
    scoate din stivă în x;
    marcare x;
    pune în stivă succesorii nevizitați ai lui x;
  }
}
```

```
void DFSV(Graf G, Varf u){
  Stiva S = S_New();
  Push(S, u);
  Varf x, v;
  while(!S_Empty(S)){
    x =*(Varf*)Pop(S);
    if(G_GetCol(G, x)==alb){
      G_SetCol(G, x, gri);
      for(v=primV(G); !dultimV(G); v=avansV(G,v))
        if(G_IsV(G,v) && G_IsA(G,x,v) && G_GetCol(G,v)==alb){
          Push(S, v);
          G_SetPred(G, v, u);
        }
    }
  }
}
```

Traversarea DFS poate folosi la clasificarea arcelor. Astfel dacă:

- `colorat[v] = alb` ⇒ `(u, v)` *arc de arbore*
- `colorat[v] = gri` ⇒ `(u, v)` *arc de revenire*
- `colorat[v] = negru`
  - `start[u] < start[v]` ⇒ `(u, v)` *arc de înaintare*
  - `start[u] > start[v]` ⇒ `(u, v)` *arc de traversare*



virf	Lista succesori
a	b c
b	c
c	d e
d	f
e	b f
f	

virf	pred	colorat	start	stop
a	0	0 1 2	1	12
b	0 a	0 1 2	2	11
c	0 b	0 1 2	3	10
d	0 c	0 1 2	4	7
e	0 c	0 1 2	8	9
f	0 d	0 1 2	5	6

Traversarea în lățime (BFS – Breadth First Search).

Prin traversarea în lățime a unui graf **G = (V, E)** se crează un arbore de parcurgere în lățime:

$$G_p = (V_p, E_p)$$
$$V_p = \{ v \in V : pred[v] \neq 0\} \cup \{s\}$$
$$E_p = \{(pred[v],v) \in E : v \in V_p - \{s\}\}$$

Folosit pentru afişarea căii de la vârful de start **s** la un vârf **v**:

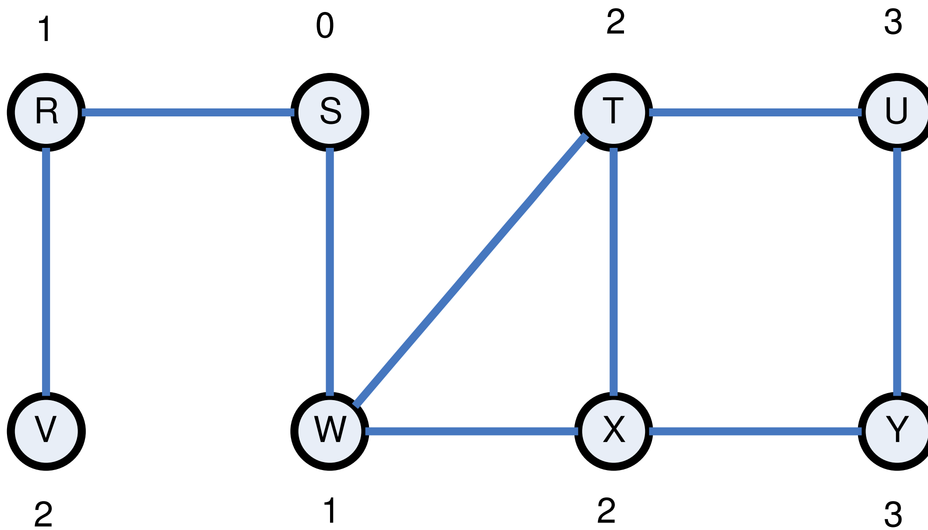
```
void afiscale(Graf G, Varf s, Varf v){
  if(v==s)
    printf("%s\n", V_GetEt(v));
  else
    if(G_GetPred(G, v)==0)
      printf("nu exista drum\n");
    else{
      afiscale(G, s, G_GetPred(G, v));
      printf("%s\n", G_GetEt(G, v));
    }
}
```

Prin traversarea în lățime, pornind din vârful sursă **s**:

- se calculează distanța (în număr de muchii) de la sursă la fiecare vârf

- pentru orice vârf v, accesibil din sursa s,calea în arbore corespunde celui mai scurt drum de la s la v

```
void BFS(Graf G, Varf s){
  Varf u,v;
  Queue Q=Q_New();
  for(u=primV(G); !dultimV(G); u=avansV(G,u)){
    if(G_IsV(G,u)){
      G_SetCol(G, u, alb);
      G_SetDist(G, u, INF);
      G_SetPred(G, u, NULL);
    }
  }
  G_SetCol(G, s, gri);
  G_SetDist(s, 0);
  Enq(Q, s);
  while(!Q_Empty(Q)){
    u = Front(Q);
    for(v=primVAd(G,u);!dultVAd(G,u); v=avansVAd(G,u,v)
      if(G_GetCol(G, v)==alb){
        G_SetCol(G, v, gri);
        G_SetDist(G, v, G_GetDist(G,u) + 1);
        G_SetPred(G, v, u);
        Enq(Q, v);
      }
    Deq(Q);
    G_SetCol(G, u, negru);
  }
}
```



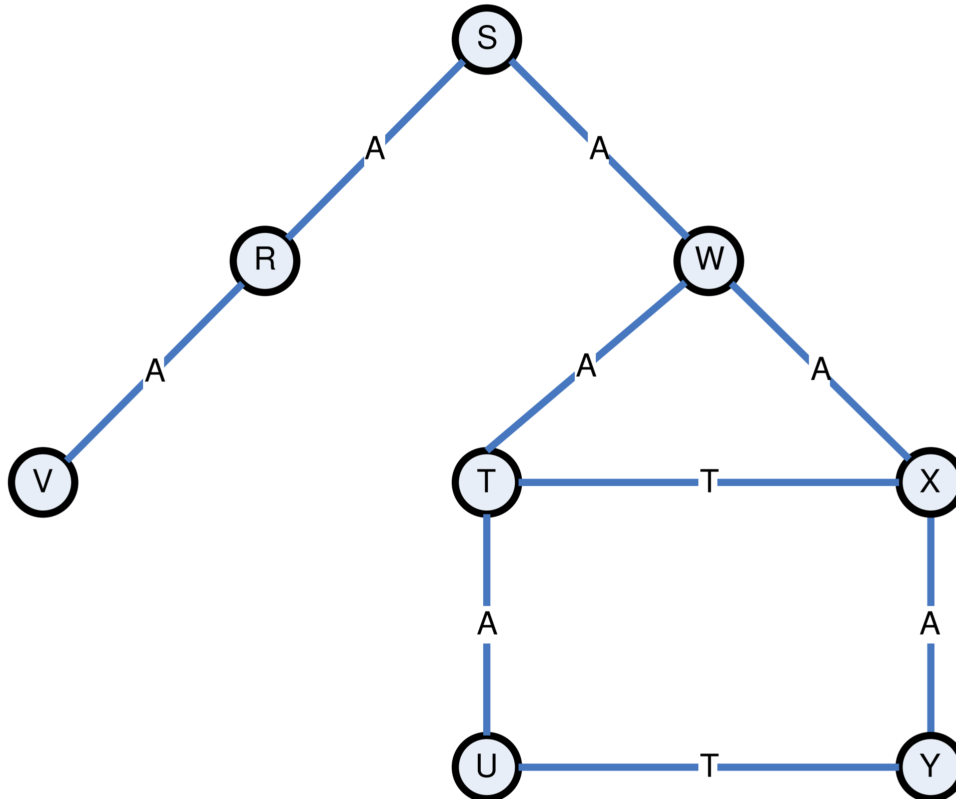
virf	pred	colorat	d
s	0	01 2	0
r	0 s	0 1 2	ω 1
t	0 w	0 1 2	ω 2
u	0 t	0 1 2	ω 3
v	0 r	0 1 2	ω 2
w	0 s	0 1 2	ω 1
x	0 w	0 1 2	ω 2
y	0 x	0 1 2	ω 3

Evoluția cozii pe parcursul execuției algoritmului BFS este:

Q Q Q   Q Q   Q Q   Q   Q   Q Q   Q Q   Q Q Q.
s sr srw rw rwv wv wvt wvtx vtx tx txu xu xuy uy y

Prin traversarea în lățime a unui graf neorientat apar *muchii de arbore* şi *muchii de traversare*.
La traversarea în lățime a unui graf orientat apar: *arce de arbore*, *arce de traversare* şi *arce de revenire.*

Arborele de parcurgere în lățime este:



Sortare topologică.

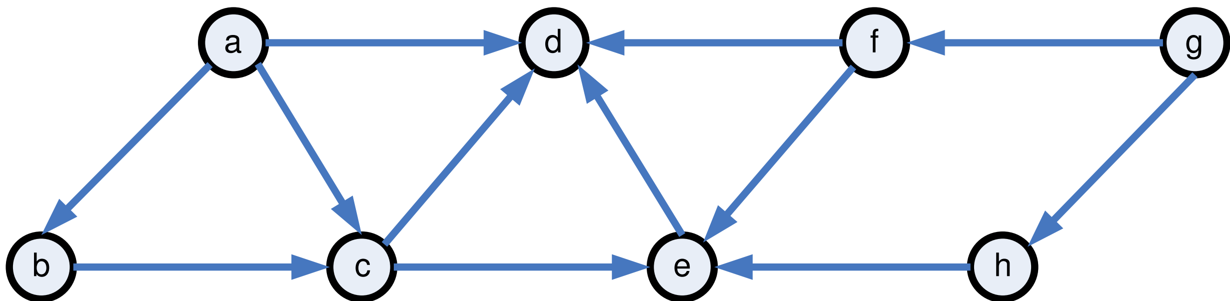
Intr-un graf orientat prezența unui arc **`(u, v)`** poate fi privită ca o relație de precedență:
"`u precede v`". Reciproc, mai multe elemente între care există relații de precedență pot fi reprezentate printr-un graf orientat. Dacă nu există cicluri, este posibilă găsirea unei relații de ordine pe ansamblul tuturor vârfurilor grafului.
O *sortare topologică* a unui graf orientat aciclic este o ordonare liniară de vârfuri în care **`u`** precede **`v`**, dacă există arcul **`(u, v)`**.
Un algoritm de sortare topologică ar considera mai întâi vârfurile care nu sunt precedate (condiționate) de alte vârfuri, adică vârfurile sursă (cu grad interior nul), după care urmează vârfurile care succed pe cele dintâi ş.a.m.d.

```
do
    gaseste un vârf v cu indeg[v]=0
    pune v în coadă
    şterge vârful v şi arcele incidente în vârf
while mai sunt vârfuri)
```

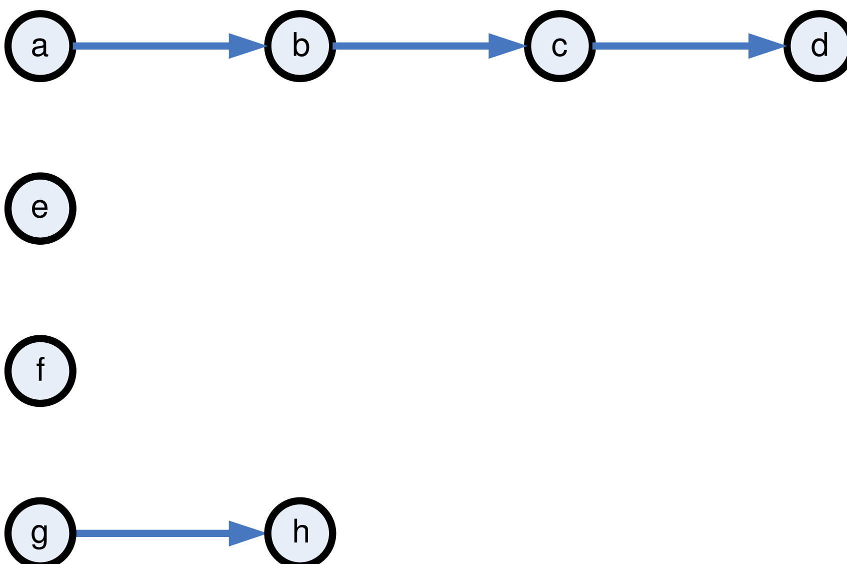
Sortarea topologică se poate realiza printr-o traversare în adâncime (DFS). Un vârf care nu mai are succesori (colorat negru) are grad de ieşire 0, deci apare în dreapta şirului de vârfuri sortate topologic; el va fi pus în stivă, a.î. în vârful stivei vor apare vârfurile cu grad de intrare 0.

```
void topsort(Graf G, Varf u){
  Stiva S=S_New ();
  Varf v;
  G_SetCol(G, u, gri);
  for(v=primV(G); !dultimV(G); v=avansV(G,v))
    if(G_IsV(G,v) && G_IsA(G, u, v) && G_GetCol(G,v)==alb)
      topsort(G, v);
  Push(S, u);
}
```

Pentru graful de mai jos:



Pădurea de acoperire în adâncime este:



Elementele sunt puse în stivă în ordinea: **`d c b a e f h g ,`** deci sortarea topologică este: **`g h f e a b c d`**

Se constată că există mai multe variante de sortare topologică, deoarece vârfurile independente, având la un moment dat gradul de intrare 0 pot fi considerate în orice ordine.

Determinarea componentelor tare conexe.

O *componentă tare conexă* a unui graf orientat **`G = (V, E)`** este un set maximal de vârfuri **`U`**$\subseteq$**`V`** a.î. pentru $\forall$ **`u, v`** $\in$ **`U`** avem **`u`** $\rightarrow$ **`v`** şi **`v`** $\rightarrow$ **`u`** (vârfurile u şi v sunt accesibile unul din celălalt).

Se formează *graful transpus*, inversând direcţia arcelor.

$G^T = (V, E^T)$

$E^T = \{(u, v) : (v, u) \in E\}$

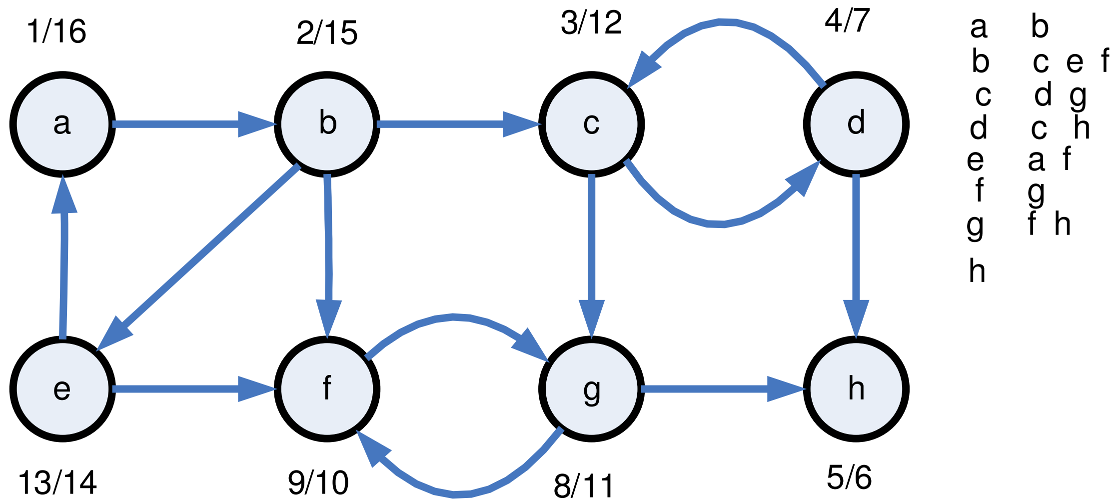
Grafurile **G** *şi* G^T *au aceleaşi componente tare conexe.*

Algoritmul pentru calculul componentelor tare conexe presupune două traversări în adâncime: în prima traversare a grafului **G** se calculează timpii terminării **`stop[u]`** pentru fiecare vârf **`u`**. A doua traversare

se face asupra grafului transpus $\mathbf{G^T}$ în ordinea vârfurilor dictată de vectorul **`stop[]`**. Fiecare arbore din pădurea de acoperire în adâncime a celei de a doua parcurgeri reprezintă o componentă tare conexă.

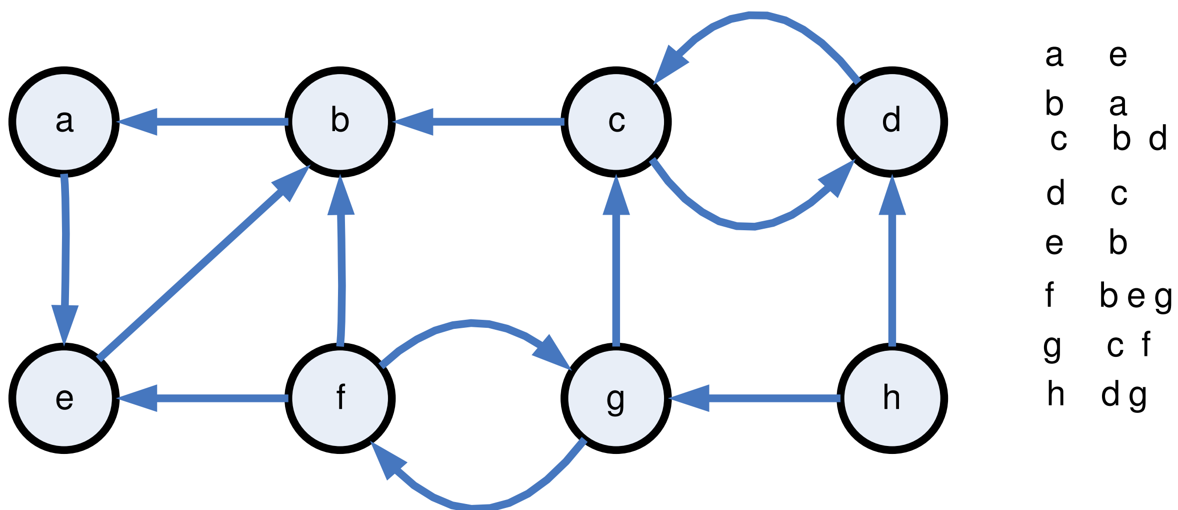
`CTC(G)`

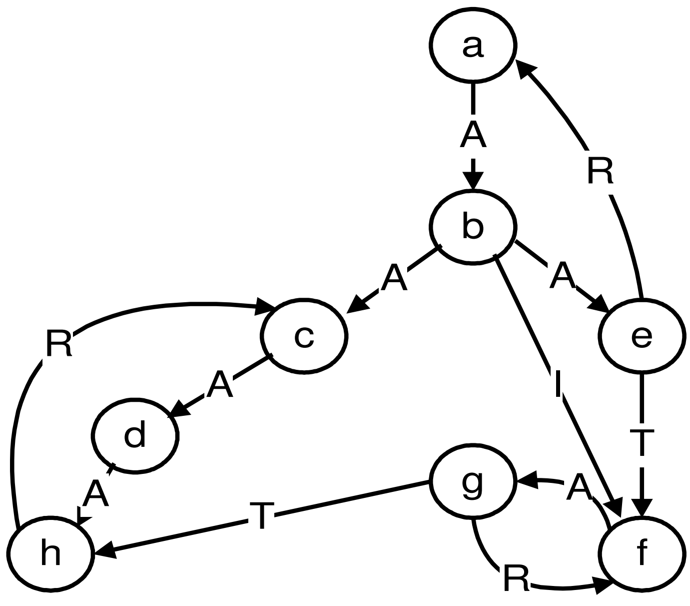
```
DFS(G)
Calcul G^T
DFS(G^T) în ordinea dictata de stop[]
```



varf	pred	color	start	stop1
a	0	0 1 2	1	16
b	0 a	0 1 2	2	15
c	0 b	0 1 2	3	12
d	0 c	0 1 2	4	7
e	0 b	0 1 2	13	14
f	0 g	0 1 2	9	10
g	0 c	0 1 2	8	11
h	0 d	0 1 2	5	6

Graful transpus este:





varf	stop1	pred	color	start	stop h
a	16	0	0 1 2	1	6
b	15	0 e	0 1 2	3	4
e	14	0 a	0 1 2	2	5
c	12	0	0 1 2	7	10
g	11	0	0 1 2	11	14
f	10	0	0 1 2	12	13
d	7	0	0 1 2	8	9
h	6	0	0 1 2	15	16