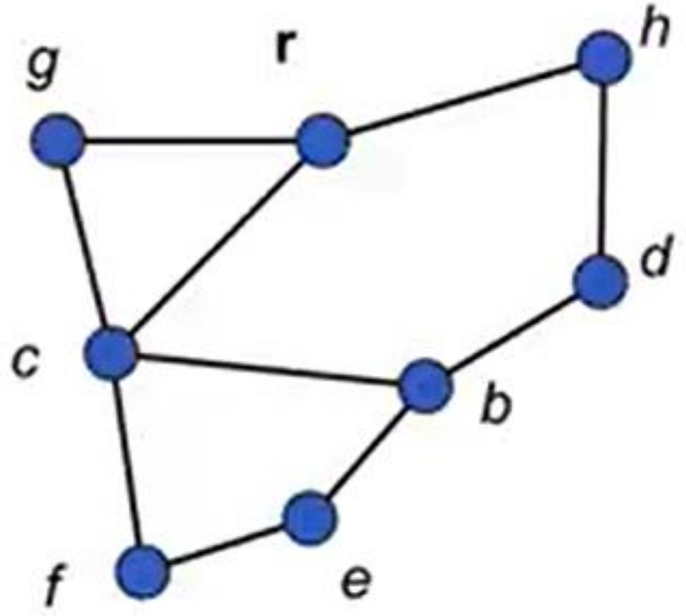
ALGORITHMES DES GRAPHES

PARCOURS EN LARGEUR

Le cœur :

- Tant que Non(FileVide(F))
  - u:=tête(F);
  - Pour tout voisin v de u faire
    - Si Couleur[v]=Blanc alors
      - Couleur[v]:=Gris;
      - Dist[v]:=Dist[u]+1;
      - Père[v]:=u;
      - Enfiler(F,v)
  - Défiler(F)
  - Couleur[u]:=Noir;

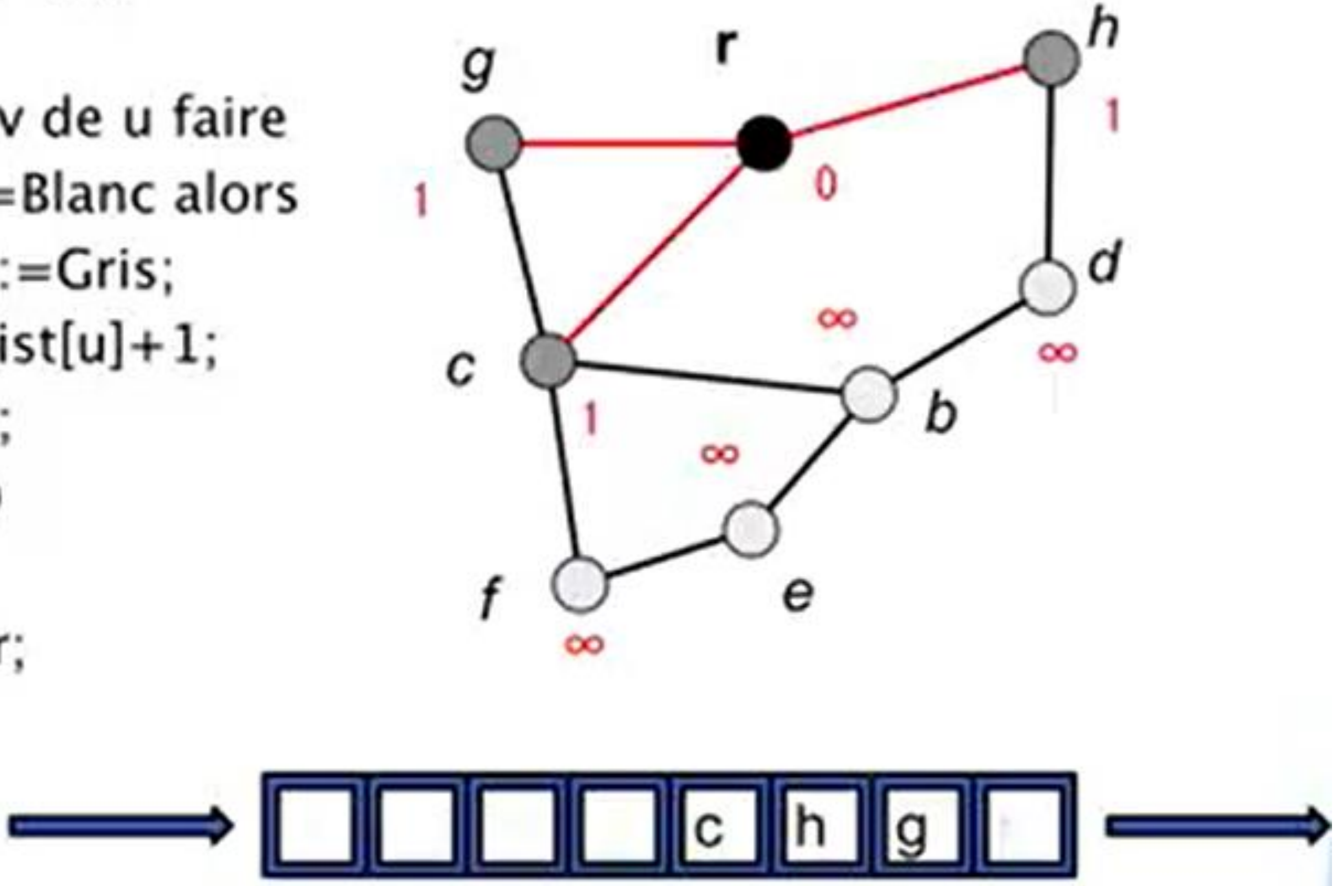


PARCOURS EN LARGEUR

Rappel du cœur :

- Tant que Non(FileVide(F))
  - u:=tête(F);
  - Pour tout voisin v de u faire
    - Si Couleur[v]=Blanc alors
      - Couleur[v]:=Gris;
      - Dist[v]:=Dist[u]+1;
      - Père[v]:=u;
      - Enfiler(F,v)
  - Défiler(F)
  - Couleur[u]:=Noir;

À la fin du 1er tour du tant que on a :

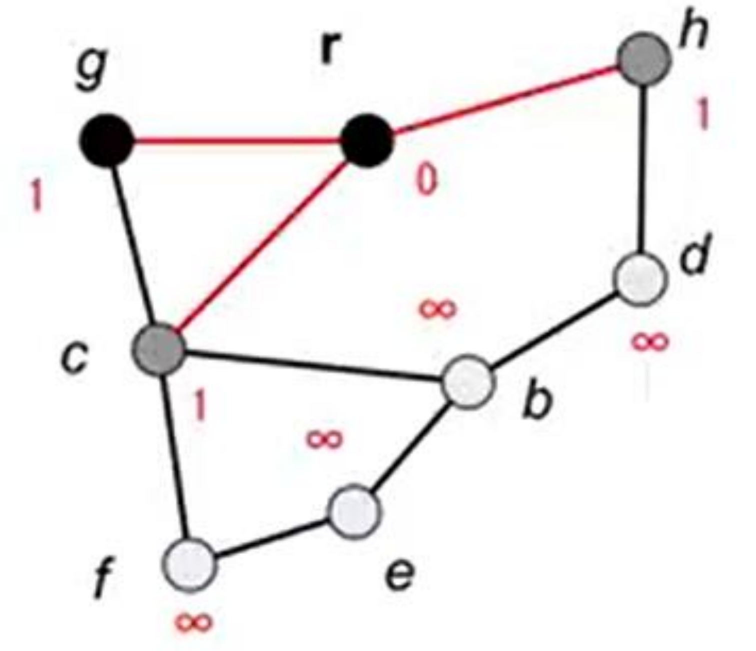


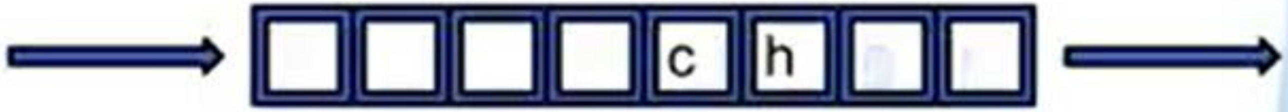
PARCOURS EN LARGEUR

Rappel du cœur :

- Tant que Non(FileVide(F))
  - u:=tête(F);
  - Pour tout voisin v de u faire
    - Si Couleur[v]=Blanc alors
      - Couleur[v]:=Gris;
      - Dist[v]:=Dist[u]+1;
      - Père[v]:=u;
      - Enfiler(F,v)
  - Défiler(F)
  - Couleur[u]:=Noir;

À la fin du 2ème tour du tant que on a :





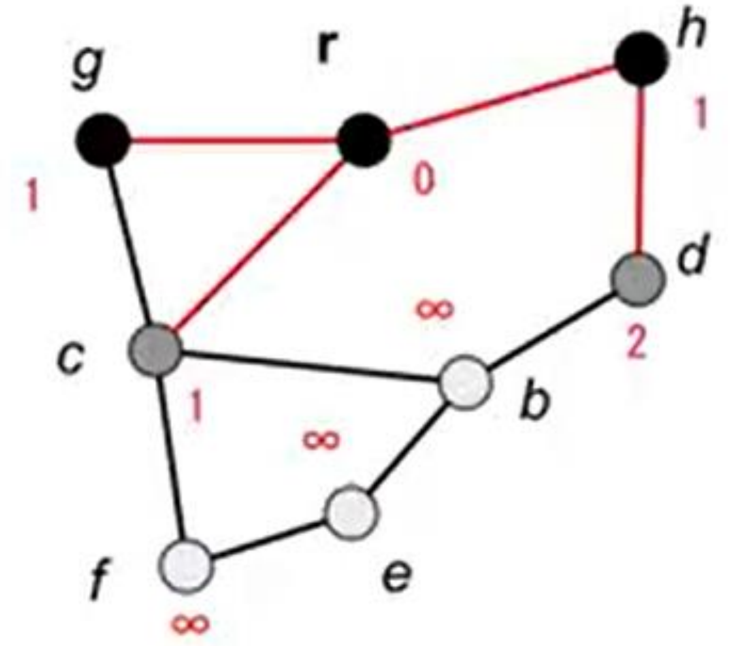
Ici l'examen de g n'a rien modifié (sauf sa couleur et il sort de la file)

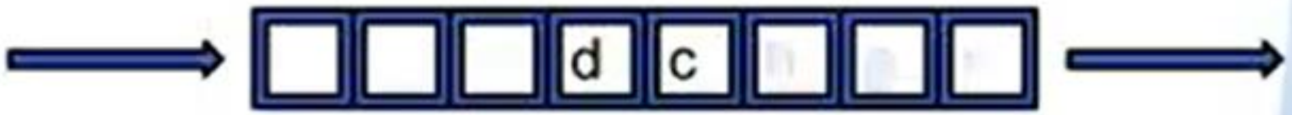
PARCOURS EN LARGEUR

Rappel du cœur :

- Tant que Non(FileVide(F))
  - u:=tête(F);
  - Pour tout voisin v de u faire
    - Si Couleur[v]=Blanc alors
      - Couleur[v]:=Gris;
      - Dist[v]:=Dist[u]+1;
      - Père[v]:=u;
      - Enfiler(F,v)
  - Défiler(F)
  - Couleur[u]:=Noir;

À la fin du 3ème tour du tant que on a :





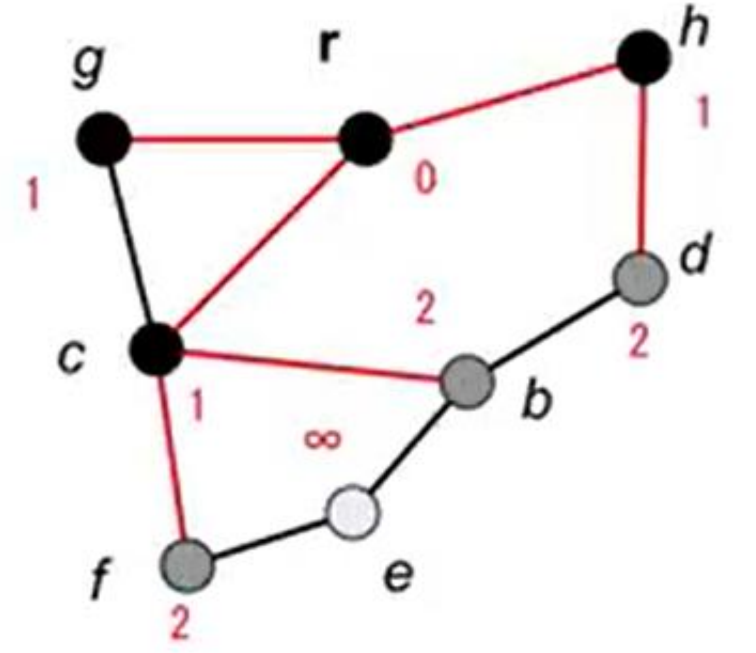
L'examen de h permet « d'atteindre » d

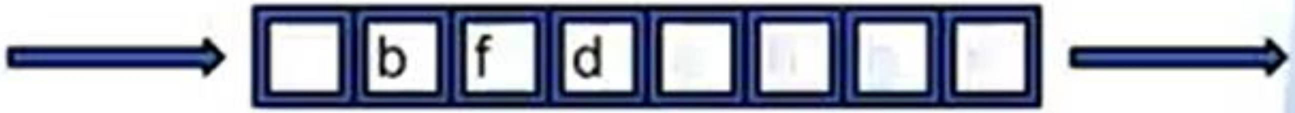
PARCOURS EN LARGEUR

Rappel du cœur :

- Tant que Non(FileVide(F))
  - u:=tête(F);
  - Pour tout voisin v de u faire
    - Si Couleur[v]=Blanc alors
      - Couleur[v]:=Gris;
      - Dist[v]:=Dist[u]+1;
      - Père[v]:=u;
      - Enfiler(F,v)
  - Défiler(F)
  - Couleur[u]:=Noir;

À la fin du 4ème tour du tant que on a :

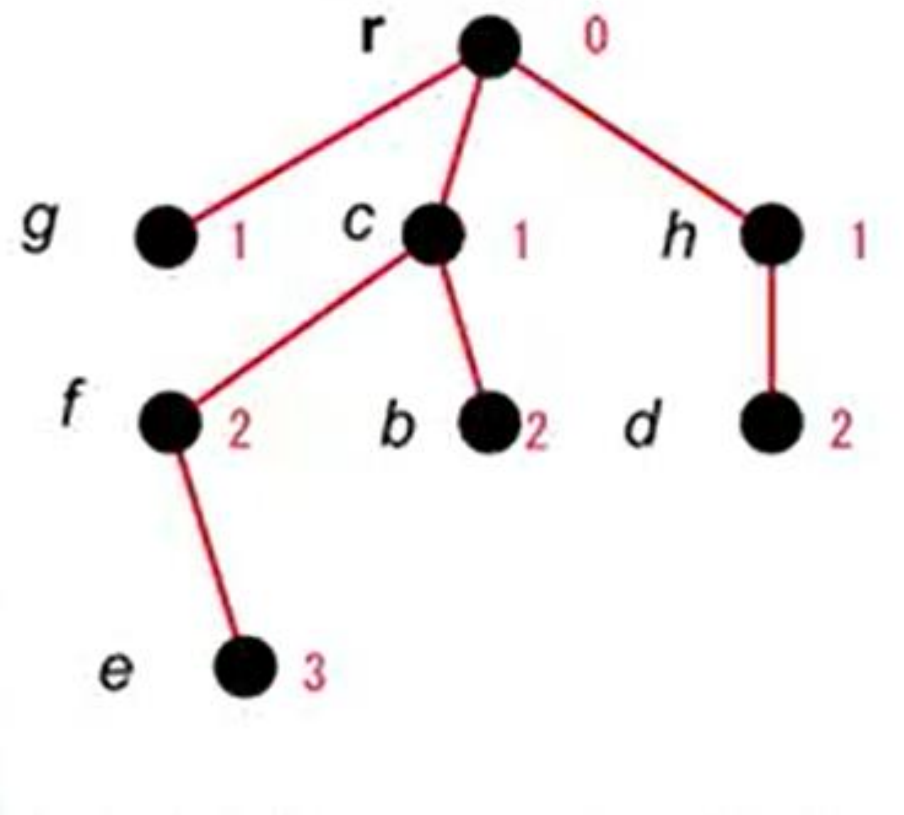




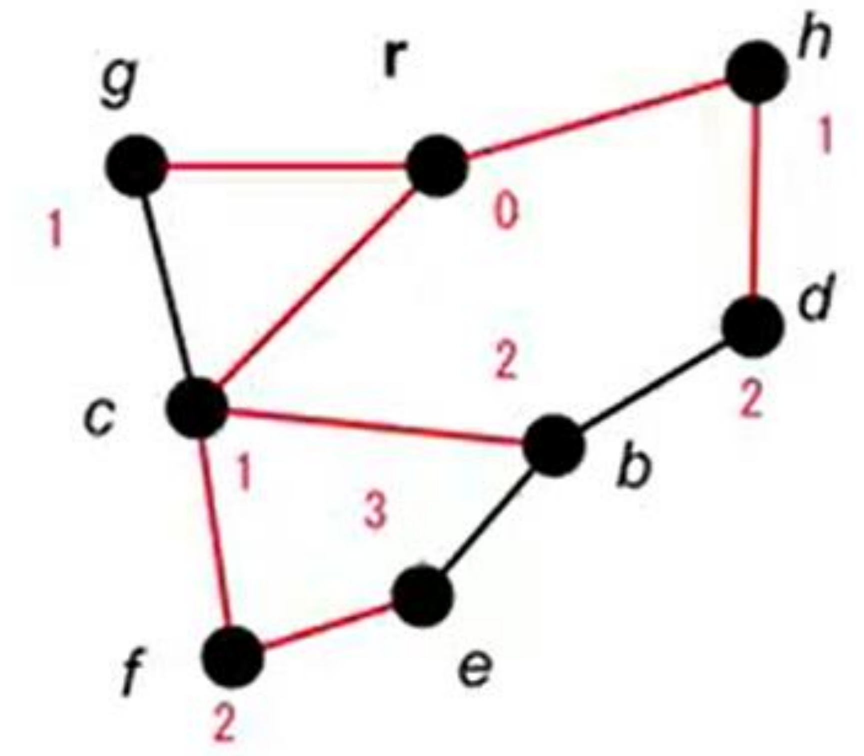
L'examen de c permet « d'atteindre » f et b

PARCOURS EN LARGEUR

Synthèse du résultat :



L'état des marques à la fin :



PARCOURS EN PROFONDEUR

Initialisation

DFS (Parcours en profondeur)

r : sommet de départ du parcours

```
Input: G = (V, E), r
Variables: Tableaux Couleur, Pere, debut, fin.
           Entier temps.
```

Chaque case de ces 4 tableaux est associé à un sommet de G

```
Pour chaque sommet u faire
    Couleur[u] := Blanc;
    Pere[u] := NIL;
temps := 0;
DFS-Rec(r,G,Couleur,Pere,debut,fin,temps);
```

Initialisation des variables

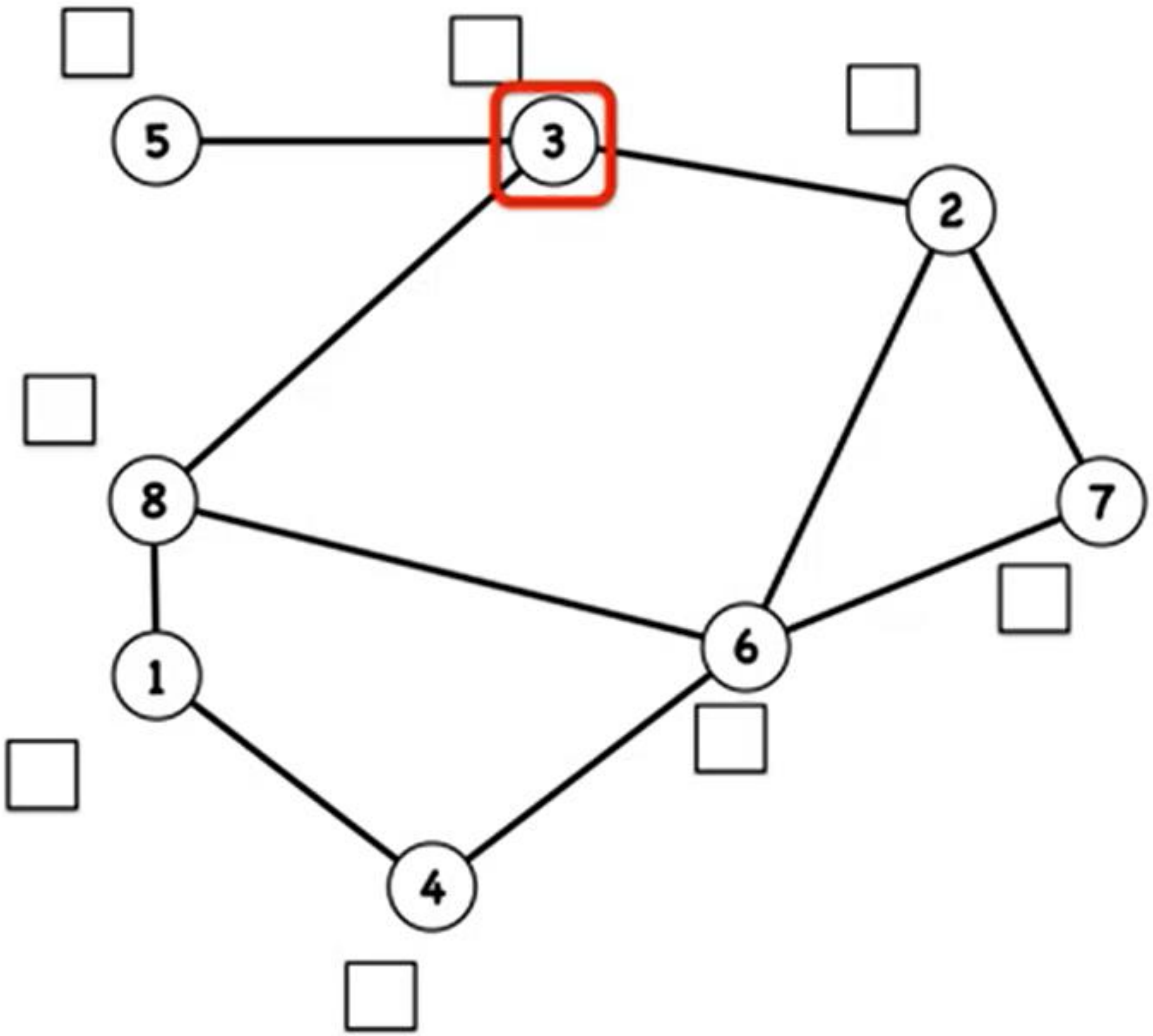
Appel à la fonction récursive

PARCOURS EN PROFONDEUR

DFS-Rec

Input: u, G = (V,E)
Input-Output: Tableaux Couleur, Pere, debut, fin. Entier temps.

```
Couleur[u] := Gris;
temps := temps + 1;
debut[u] := temps;
Pour tout voisin v de u faire
    Si Couleur[v] = Blanc alors
        Pere[v] := u;
        DFS-Rec(v);
Couleur[u] := Noir;
temps := temps + 1;
fin[u] := temps;
```

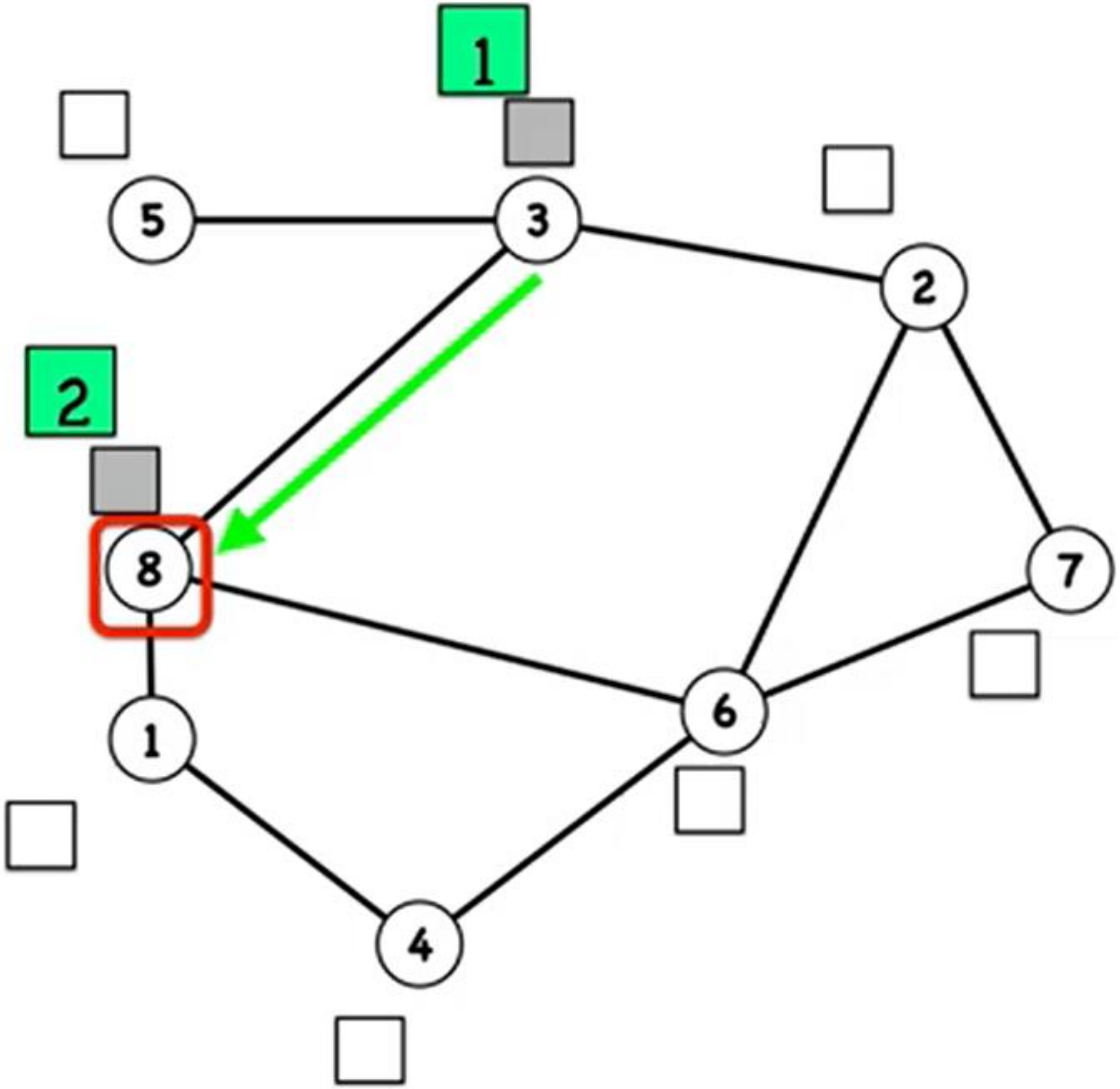


PARCOURS EN PROFONDEUR

DFS-Rec

Input: u, G = (V,E)
Input-Output: Tableaux Couleur, Pere, debut, fin. Entier temps.

```
Couleur[u] := Gris;
temps := temps + 1;
debut[u] := temps;
Pour tout voisin v de u faire
    Si Couleur[v] = Blanc alors
        Pere[v] := u;
        DFS-Rec(v);
Couleur[u] := Noir;
temps := temps + 1;
fin[u] := temps;
```

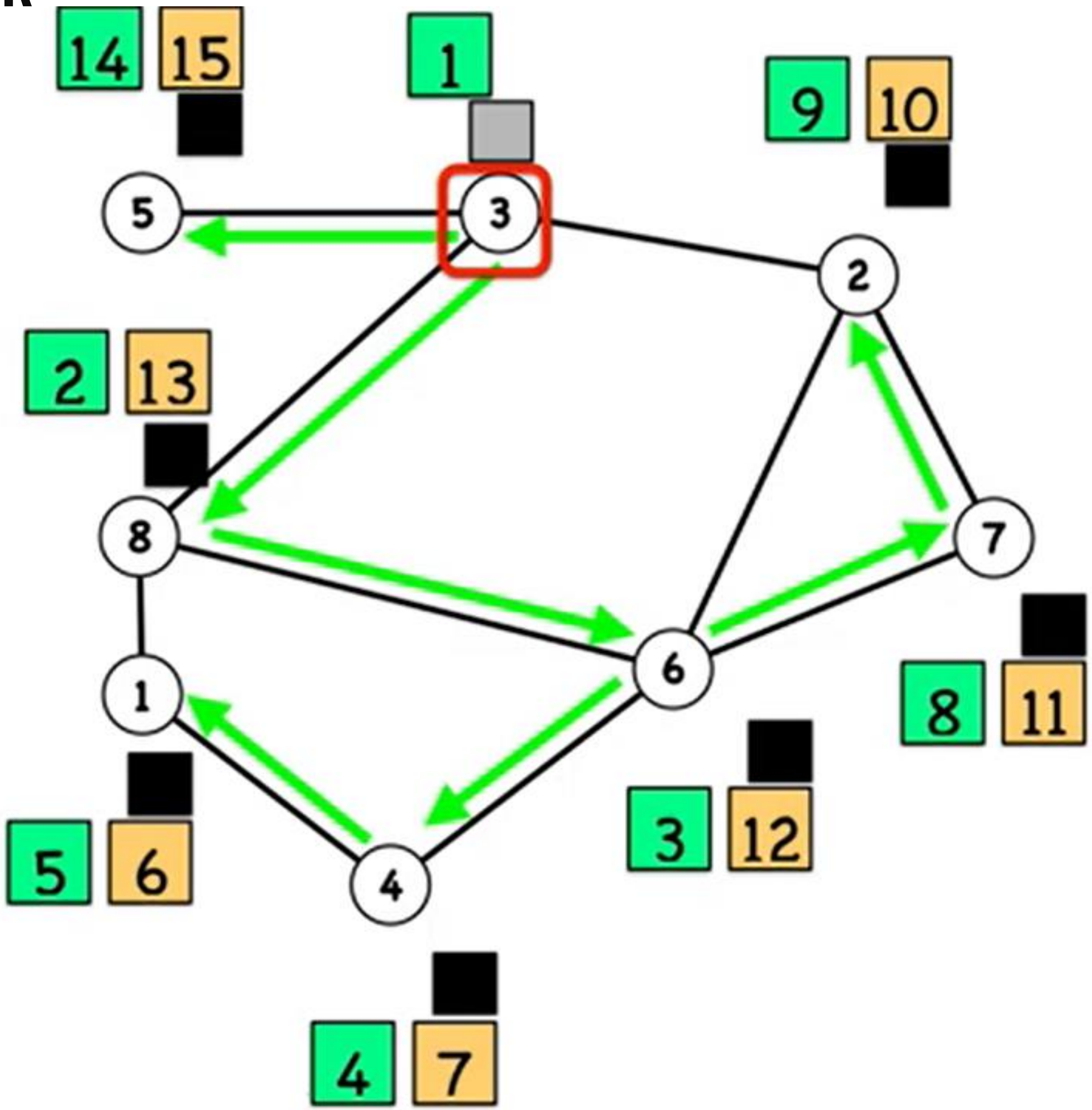


PARCOURS EN PROFONDEUR

DFS-Rec

Input: u, G = (V,E)
Input-Output: Tableaux Couleur, Pere, debut, fin. Entier temps.

```
Couleur[u] := Gris;
temps := temps + 1;
debut[u] := temps;
Pour tout voisin v de u faire
    Si Couleur[v] = Blanc alors
        Pere[v] := u;
        DFS-Rec(v);
Couleur[u] := Noir;
temps := temps + 1;
fin[u] := temps;
```



ALGORITHME DE DIJKSTRA

Les données du problème : G=(V,E,w), r

V = ensemble des sommets

E = ensemble des arcs

w = poids (ou longueur des arcs)

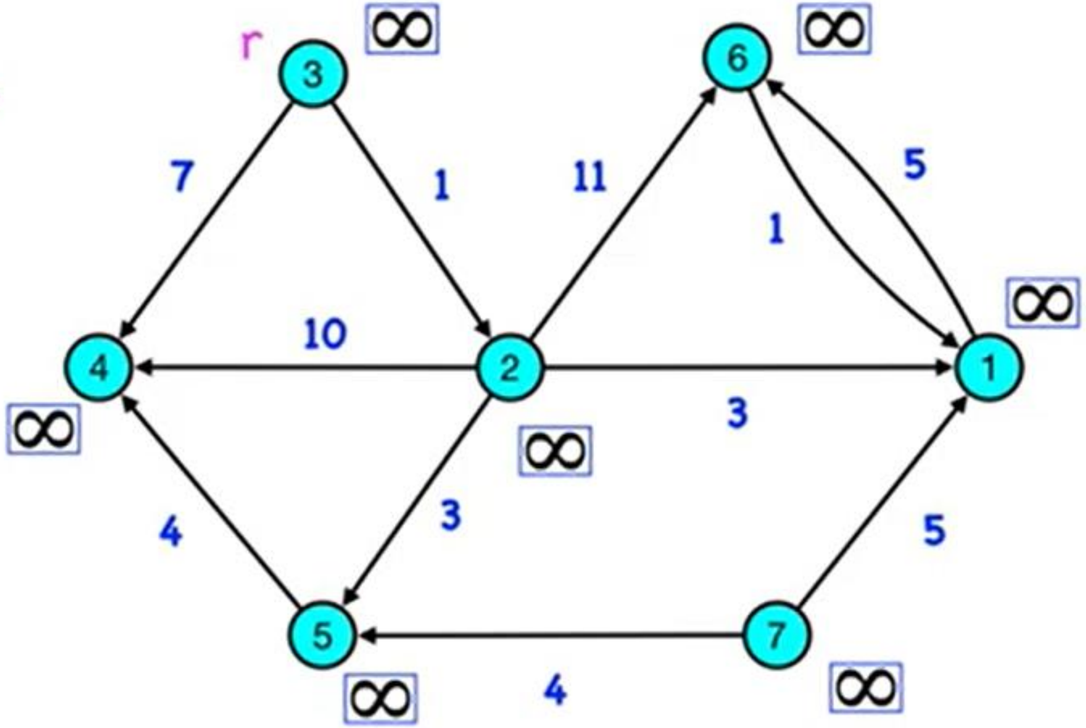
r : sommet de « départ »

Les structures de données

d[u]   Parent[u]   F

Initialisation

Pour tout u de V :
    d[u] := infini
    Parent[u] := NIL
F := V
d[r] := 0



u	1	2	3	4	5	6	7
d[u]	∞	∞	∞	∞	∞	∞	∞
Parent[u]	NIL	NIL	NIL	NIL	NIL	NIL	NIL
F	1	2	3	4	5	6	7

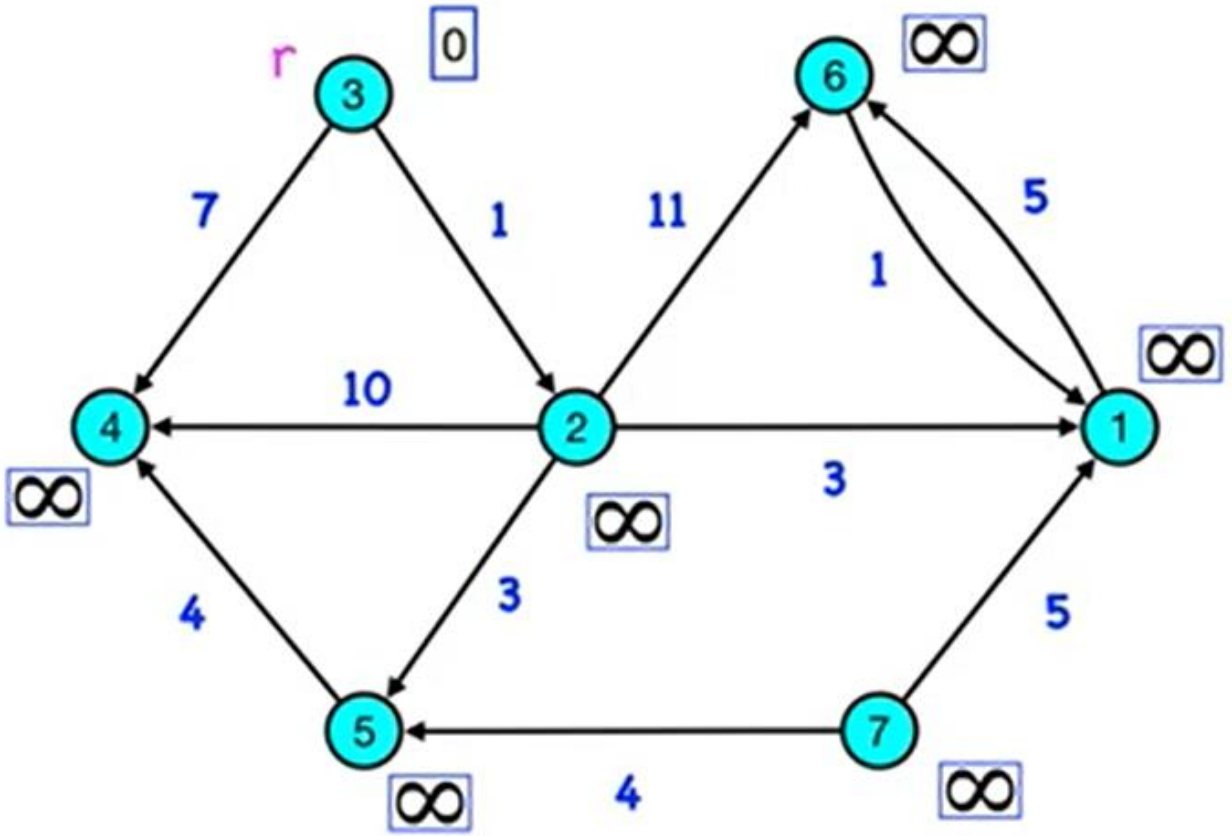
ALGORITHME DE DIJKSTRA

Relâchement (u,v)

Si d[v] > d[u] + w(u,v) alors :

  d[v] := d[u] + w(u,v)

  Parent[v] := u



Dijkstra (G, r)

Initialisation

Tant que F est non vide :

  Extraire de F le sommet u ayant le plus petit d[.]

  Pour tout arc (u,v):

    Relâchement (u,v)

Retourner d[.], Parent[.]

			u				
u	1	2	3	4	5	6	7
d[u]	∞	∞	0	∞	∞	∞	∞
Parent[u]	NIL	NIL	NIL	NIL	NIL	NIL	NIL
F	1	2		4	5	6	7

ALGORITHME DE KRUSKAL

Principe

L'algorithme construit un arbre couvrant minimum en sélectionnant des arêtes par poids croissant.

- l'algorithme considère toutes les arêtes du graphe par poids **croissant** (en pratique, on trie d'abord les arêtes du graphe par poids croissant);
- et pour chacune d'elles, il **la sélectionne si elle ne crée pas un cycle**.