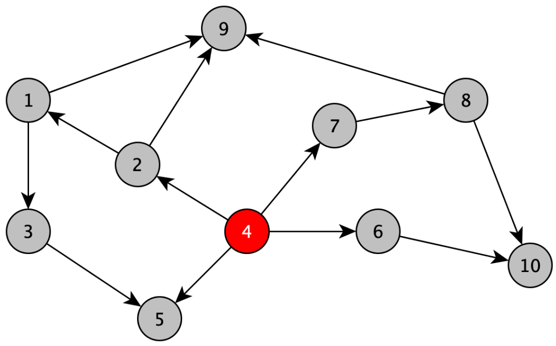
PLAN

PARCOURS DANS UN GRAPHE

Comment parcourir les sommets d'un graphe ?



PARCOURS EN LARGEUR

<u>Principe</u> : visiter tous les sommets du graphe à partir d'un sommet « a » en suivant l'ordre des générations (les fils de « a » sont d'abord visités, puis les fils des fils de « a », …).

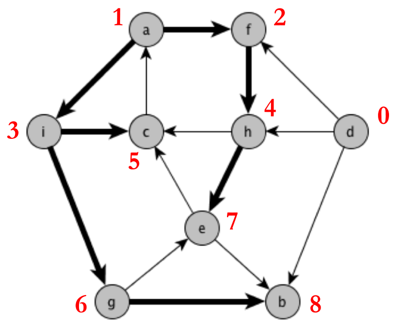
Le premier atteint est le premier traité (gestion d'une file d'attente).

Un sommet « x » est marqué (marquer(x)=vrai) lorsque l'algorithme atteint « x ». Il le met alors dans la file d'attente.

Un sommet « x » est traité (traiter(x)$\neq$0) lorsque tous ses successeurs ont été marqués. Il sort de la file d'attente.

Le tableau *traiter(.)* renvoie la numérotation des sommets indiquant leur ordre de visite par l'algorithme.

PARCOURS EN LARGEUR



Largeur(G=(X,U), a, traiter) ;
{ F = $\varnothing$; p=1 ;
  pour tout x$\in$X **faire**
  { marquer(x)=faux ; traiter(x)=0 ; }
  marquer(a)=vrai ;
  mettre a dans F ;
  tant que F non vide **faire**
  { prendre x dans F ;
    tant qu'il existe y successeur de x **et** marquer(y)=faux **faire**
    { marquer(y)=vrai ; mettre y dans F ; }
    traiter(x)=p ;
    enlever x de F ;
    p=p+1 ;
  }
}

$\boxed{O(n+m)}$

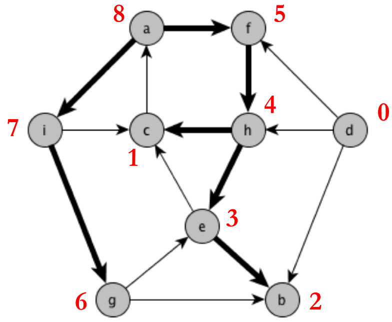
Principe : visiter tous les sommets de G à partir d'un sommet « a » en commençant par les plus « lointains » successeurs (DFS Depth First Search).

Le dernier atteint est le premier traité (gestion d'une pile).

Un sommet « x » est marqué (marquer(x)=vrai) lorsque l'algorithme atteint « x ». Il le met alors dans la file d'attente.

Un sommet « x » est traité (traiter(x)≠0) lorsque tous ses successeurs ont été marqués. Il sort de la file d'attente.

Le tableau *traiter(.)* renvoie la numérotation des sommets indiquant leur ordre de visite par l'algorithme.



Profondeur(G=(X,U), a, traiter) ;
{ P = ∅ ; p=1 ;
pour tout x∈X **faire**
{ marquer(x)=faux ; traiter(x)=0 ; }
marquer(a)=vrai ;
empiler a dans P ;
tant que P non vide **faire**
{ **tant qu'il existe** y successeur de SommetPile **et** marquer(y)=faux **faire**
{ marquer(y)=vrai ; empiler y dans P ; }
traiter(SommetPile)=p ;
depiler ;
p=p+1 ;
}
}

$\boxed{O(n+m)}$

PARCOURS EN PROFONDEUR

```
Profondeur_récursif(G=(X,U), a, traiter) ;
{     p=1 ;
      pour tout x∈X faire
      {     marquer(x)=faux ; traiter(x)=0 ; }
      Rechercher(a,p) ;
}


Rechercher(x,p)
{     marquer(x)=vrai ;
      pour tout y successeur de x et marquer(y)=faux faire rechercher(y,p) ;
      traiter(x)=p ;
      p=p+1 ;
}
```

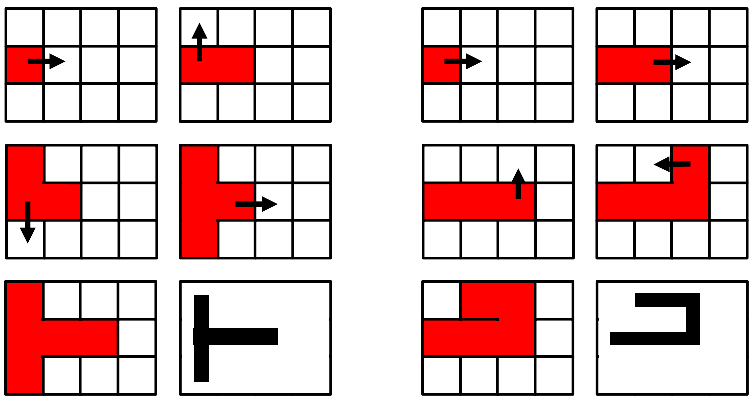
$\boxed{O(n+m)}$

DIFFÉRENCE ENTRE LES PARCOURS

<u>Objectif</u> : illustrer les différences entre les 2 parcours

Génération de labyrinthe

Parcours en largeur   Parcours en profondeur

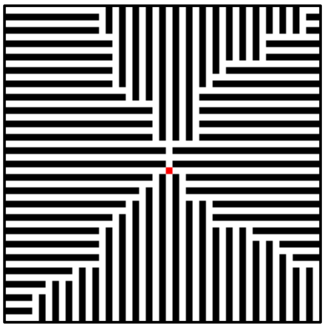


DIFFÉRENCE ENTRE LES PARCOURS

Objectif : illustrer les différences entre les 2 parcours

Génération de labyrinthe

Parcours en largeur



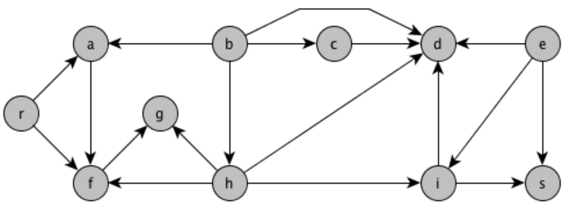
Parcours en profondeur

DÉTECTION DE CIRCUIT

Théorème

Un graphe orienté G=(X,U) est sans circuit si et seulement si :

$$\exists x \in X \text{ tel que } \begin{cases} d^{-}(x)=0 \\ \text{et} \\ \text{le graphe G auquel on enlève x est sans circuit} \end{cases}$$



Adapter l'algorithme de parcours en largeur en utilisant de théorème ci-dessus pour concevoir un algorithme de détection de circuit.

DÉTECTION DE CIRCUIT

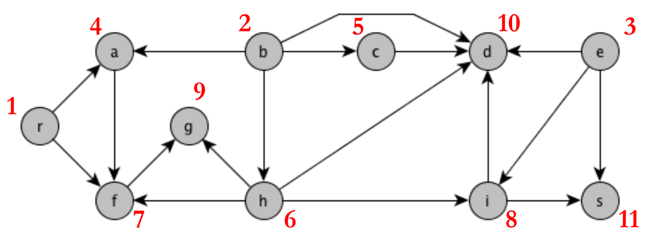
Circuit(G=(X,U)): boolean ;
{ F = ∅ ; p=0 ;
  pour tout x∈X tel que $d^-(x)=0$ **faire**
  { mettre x dans F ; p=p+1 ;
  }
  tant que F non vide **faire**
  { prendre x dans F ;
    tant qu'il existe y successeur de x **faire**
    { $d^-(y)=d^-(y)-1$;
      si $d^-(y)=0$ **faire** { mettre y dans F ; p=p+1 ; }
    }
    enlever x de F ;
  }
  si p=n **faire** **return**(faux) ;
  sinon **return**(vrai) ;
}

$\boxed{O(n+m)}$

NUMÉROTATION TOPOLOGIQUE

Un graphe orienté G=(X,U) admet une numérotation topologique des sommets s'il existe une bijection topo(.) : $X \to \{1,2,3, \dots, n\}$
$x \to topo(x)$

vérifiant : $\forall(x,y)\in U$, $topo(x) < topo(y)$



6

NUMÉROTATION TOPOLOGIQUE

Propriété

Un graphe admet une numérotation topologique si et seulement si c'est un graphe sans circuit.

Adapter l'algorithme de parcours en largeur pour concevoir un algorithme de numérotation topologique.

NUMÉROTATION TOPOLOGIQUE

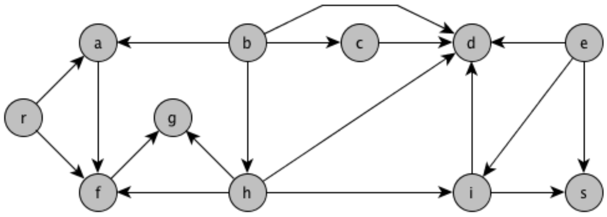
Num_topologique(G=(X,U),topo) ;
{ F = $\varnothing$; p=1 ;
    pour tout $x \in X$ tel que $d^-(x)=0$ **faire**
    { mettre x dans F ;
        topo(x)=p ;
        p=p+1 ;
    }
    tant que F non vide **faire**
    { prendre x dans F ;
        tant qu'il existe y successeur de x **faire**
        { $d^-(y)=d^-(y)-1$;
            si $d^-(y)=0$ **faire** { mettre y dans F ; topo(y)=p ; p=p+1 ; }
        }
        enlever x de F ;
    }
}

$\boxed{O(n+m)}$

PARTITION EN NIVEAUX

Un graphe orienté G=(X,U) admet une partition en niveaux s'il existe une partition de X en X_0, …, X_p , telle que :

- X_0 est l'ensemble des sources de G
- $\forall i \geq 1$, X_i est l'ensemble des sources du sous-graphe engendré par X-$\{X_0 \cup \ldots \cup X_{i-1}\}$



$X_0 = \{r,b,e\}$ $X_1 = \{a,c,h\}$ $X_2 = \{f,i\}$ $X_3 = \{d,g,s\}$

PARTITION EN NIVEAUX

<u>Propriété</u>
Un graphe admet une partition en niveaux si et seulement si c'est un graphe sans circuit.

Adapter l'algorithme de détection de circuit pour concevoir un algorithme de partition en niveaux.

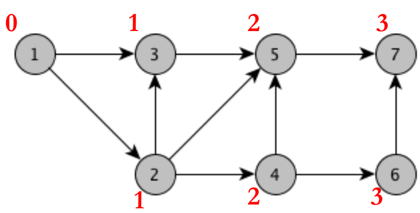
PARTITION EN NIVEAUX

```
NiveauxCircuit(G=(X,U),{X_i}_i≥0): boolean ;
{     X_0=∅ ; p=0 ; i=0 ;
      pour tout x∈X tel que d^-(x)=0  faire
      {     p=p+1 ; X_0 = X_0∪{x} ;    }
      tant que (p<n et X_i≠∅) faire
      {     X_i+1=∅ ;
            pour tout x de X_i faire
            {     tant qu'il existe y successeur de x faire
                  {     d^-(y)=d^-(y)-1 ;
                        si  d^-(y)=0  faire {   p=p+1 ; X_i+1 = X_i+1∪{y} ;   }
                  }
            }
            i=i+1 ;
      }
      si p=n faire      return(faux) ;
      sinon             return(vrai) ;
}
```

$\boxed{O(n+m)}$

PLUS COURT CHEMIN

Adapter l'algorithme de parcours en largeur pour concevoir un algorithme de calcul de la longueur du plus court chemin (en terme de nombre d'arcs) d'un sommet (passé en paramètre) à tous les autres.



```
PlusCourtCheminNbArcs(G=(X,U), a, long) ;
{     F=∅ ;
      pour tout x∈X faire
      {     marquer(x)=faux ; long(x)=+∞ ; }
      marquer(a)=vrai ;
      mettre a dans F ;
      long(a)=0;
      tant que F non vide faire
      {     prendre x dans F ;
            tant qu'il existe y successeur de x et marquer(y)=faux faire
            {     marquer(y)=vrai ;
                  long(y)=long(x)+1 ;
                  mettre y dans F ;
            }
            enlever x de F ;
      }
}
```

O(n+m)