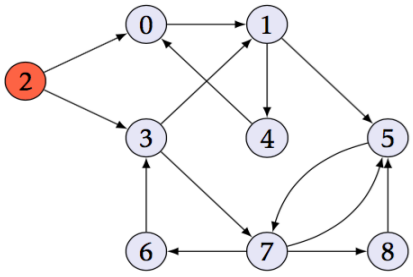
ENSA de Marrakech
M. ZRIKEM

THEORIE DFES GRAPHES
2020/2021

Série 6 de TD : Parcours de graphes

Exercice 1 :
Soit G le graphe orienté ci-dessous :

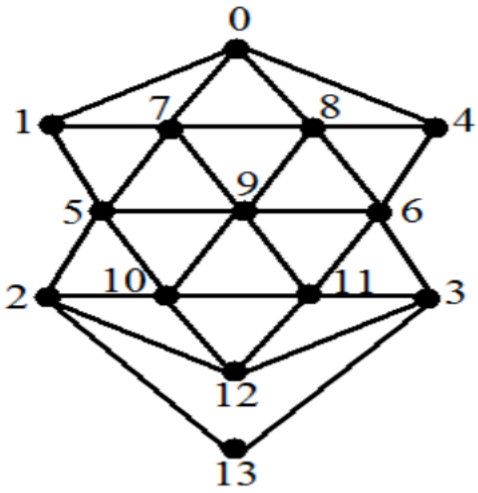


1. En partant du sommet 2, faire une exploration en largeur de ce graphe. Représenter l'évolution de la file. Dessiner l'arbre obtenu.

2. Toujours en partant du sommet 0, faire une exploration en profondeur du graphe. Représenter l'évolution de la pile. Dessiner l'arbre obtenu.

Exercice 2 :

Soit G le graphe non orienté ci-dessous :



Les voisins de chaque sommet sont supposés écrits dans l'ordre **croissant** de leurs numéros. Ainsi 0 a pour voisins 1, 4, 7, 8

3. En partant du sommet 0, faire une exploration en profondeur de ce graphe, en utilisant l'ordre de voisins tel qu'il a été défini. Dessiner l'arbre obtenu.

4. Toujours en partant du sommet 0, faire une exploration en largeur du graphe. On aura intérêt à utiliser l'évolution d'une file, afin de dessiner l'arbre final de l'exploration.

Exercice 3 :
Soit un graphe orient é G = (S, A) et soit s $\in$ S. Soit une arborescence T = (S_T , A_T) telle que A_T $\subseteq$ A et S_T est l'ensemble des sommets de G accessibles depuis s (y compris s lui-même), et telle que pour tout sommet x $\in$ S_T, le chemin de s à x dans T soit un plus court chemin de s à x dans G.

Est-il toujours possible d'obtenir T par un certain parcours en largeur de G depuis s?

Bien sûr si oui il faut le prouver et si non exhiber un contre-exemple !

Exercice 4 :
Soit G un graphe non orionté. Les deux algorithmes ci-dessous réalisent des parcours en largeur et en profondeur sur G à partir d'un sommet donné s. Rappelons que lors du parcours à partir de s, on découvre les sommets de la composante connexe de s dans G (c'est-`a-dire l'ensemble des sommets accessibles par un chemin depuis s dans G).
Lors du calcul, on utilise un ensemble Z pour stocker les sommets déjà visités, et une variable F pour stocker les sommets de la frontière, c'est-à-dire ceux ayant au moins un voisin non visité. Selon la structure de donnée qu'on utilise pour F on obtient des parcours différents.

Structures de données utilisées :

L'ensemble

On dispose des fonctions suivantes sur un ensemble Z.

- choose(Z) : renvoie un sommet x de Z (si S n'est pas vide).
- empty?(Z) : renvoie vrai ssi Z est l'ensemble vide.
- add(x,Z) : ajoute x à l'ensemble Z.
- remove(x,Z) : enlève x de l'ensemble Z (si x appartient à Z).

La file

On dispose des fonctions suivantes sur une file Q.

- checkFront(Q) : renvoie le premier sommet de F (si Q n'est pas vide).
- empty?(Q) : renvoie vrai ssi Q est la file vide.
- pushBack(x,Q) : ajoute x à l'arrière de la file Q.
- popFront(Q) : enlève le premier sommet de la file Q (si celle-ci n'est pas vide).

La pile

On dispose des fonctions suivantes sur une pile P.

- checkFront(P) : renvoie le premier sommet de P (si P n'est pas vide).
- empty?(P) : renvoie vrai ssi P est la pile vide.
- pushFront(x,P) : ajoute x au début de la pile P.
- popFront(P) : enlève le premier sommet de la pile P (si celle-ci n'est pas vide).

Algorithme 6 : Parcours en largeur

```
Données
G un graphe
s un sommet de G
Variables locales
Z un ensemble                                  /* zone connue */
F une file                                     /* la frontière */
u, v deux sommets
début
    initialisation
    add(s,Z)
    pushBack(s,F)
    répéter
        v :=checkFront (F)
        si il existe u ∉ Z adjacent à v alors
            add(u,Z)                           /* première visite de u */
            pushBack(u,F)
        sinon
            popFront(F)                        /* dernière visite de v */
        finsi
    jusqu'à empty ?(F)
fin
```

Algorithme 7 : Parcours en profondeur

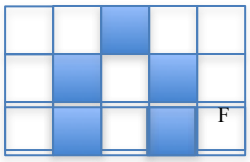
```
Données
G un graphe
s un sommet de G
Variables locales
Z un ensemble                                  /* zone connue */
F une pile                                     /* la frontière */
u, v deux sommets
début
    initialisation
    add(s,Z)
    pushFront(s,F)
    répéter
        v :=checkFront (F)
        si il existe u ∉ Z adjacent à v alors
            add(u,Z)                           /* première visite de u */
            pushFront(u,F)
        sinon
            popFront(F)                        /* dernière visite de v */
        finsi
    jusqu'à empty ?(F)
fin
```

Utiliser l'un des deux algorithmes (ou bien les deux) et mettre à jour son code pour :

1. Calculer la distance depuis la source s à chaque sommet v du graphe (longueur en nombre d'arrêtes).
2. Déterminer la première et la dernière fois où chaque sommet est visité. On peut utiliser deux tableaux Deb et Fin dans lesquels on stocke pour chaque sommet le numéro de l'itération de sa première rencontre (Deb) et sa dernière rencontre (Fin)
3. Effectuer le tri topologique des sommets de G. Cela consiste à ordonner les sommets du graphe de telle sorte que si il y a un arc (u, v) alors u vient avant v dans l'ordre calculé. Nous supposons que G et orienté et sans cycle.
4. Détecter l'existence de cycle dans G.

Exercice 5 :

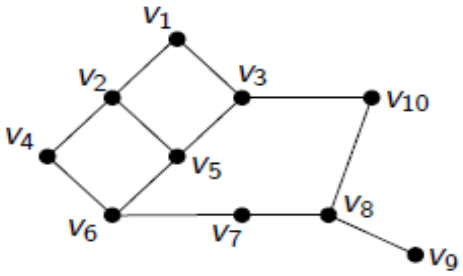
On représente un labyrinthe sous la forme d'une matrice (n, m), où les cases sont remplies par des espaces ou des murs.



Question : Peut-on aller de la case début (D) à la case fin (F) dans ce labyrinthe ?

1. Comment Transformer une matrice labyrinthe en un graphe ?
2. Donner le graphe associé au labyrinthe ci-dessous ?
3. Quel problème de la théorie des graphes faut-il résoudre pour répondre à cette question?
4. Appliquer au graphe de la question 2.

Exercice 6 :



Soit G le graphe ci-dessus

4. Déterminer une coloration des sommets de G en appliquant l'algorithme de coloration par degré
5. Est-ce que cette coloration est optimale ?
6. Déduire pour l'aspect biparti de G.
7. Effectuer un parcours en largeur de G à partir de la racine v1 et en suivant l'ordre lexicographique.
8. Comment peut-on utiliser ce parcours pour déduire à nouveau pour l'aspect biparti de G