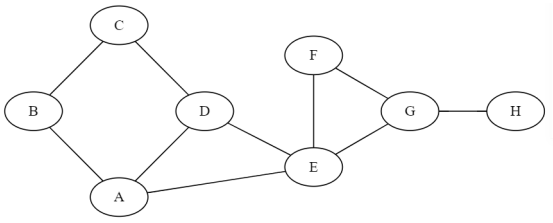
"Parcours en largeur et en profondeur de graphes / Recherche d'un chemin "

Table des matières

Parcourir un graphe consiste à lister tous ses sommets une seule fois.

On peut *parcourir un graphe* :
– soit pour lister simplement tous ses sommets ;
– soit pour trouver un chemin pour relier un sommet à un autre.

Considérons le graphe **(G_Test)** ci-dessous.



Exercice 0
Créer le dictionnaire représentant le graphe **(G_Test)**.

I Parcours en largeur

L'algorithme du **parcours en largeur** repose sur l'utilisation d'une **file F** et retourne la liste **Lsommets** contenant tous les sommets du graphe en le parcourant à partir d'un sommet donné :

– on part d'un sommet que l'on enfile dans **F** ;
– on visite ses voisins et on les enfile dans **F** s'ils ne sont pas déjà présents dans **F** et s'ils n'ont pas déjà été traités (c-a-d qui ne sont pas dans **Lsommets**) ;
– on défile **F** dans **Lsommets** ;
– on recommence à partir du point deux tant que la file n'est pas vide.

Voici la traduction en pseudo-code du parcours en largeur d'un graphe décrit ci-dessus.

F *est une file vide*
Lsommets
On enfile un sommet dans*est une liste vide*

On enfile un sommet dans ***F***
Tant que ***F*** *n'est pas vide*
..... *S=Tête(**F**)*
..... *On enfile les voisins de S qui ne sont pas déjà présents dans* ***F*** *et qui n'ont pas été déjà défilés (c-a-d qui ne*
..... *sont pas dans* ***Lsommets**)*
..... *On défile F dans* ***Lsommets***
Fin Tant Que
On retourne ***Lsommets***

Exercice 1
Créer une fonction nommée **parcours_en_largeur*(graphe,sommet)*** qui parcourt en *largeur* le graphe **graphe**, représenté par un dictionnaire, à partir du sommet **sommet**.
Cette fonction retournera la liste des sommets parcourus si le sommet **sommet** appartient bien au graphe et None sinon.

⇝ Rq. *On pourra coder en utilisant les fonctions relatives aux files et/ou non.*

Exercice 2
Tester la fonction **parcours_en_largeur** avec **(G_Test)** et chacun de ses sommets tour à tour.

II Parcours en profondeur

L'algorithme du **parcours en profondeur** repose sur l'utilisation d'une **pile P** et retourne la liste **Lsommets** contenant tous les sommets du graphe en le parcourant à partir d'un sommet donné :

– on part d'un sommet que l'on empile dans **P** ;
– on dépile **P** et on marque le sommet dépilé comme traité dans **Lsommets** ;
– on empile dans **P** chacun des voisins du sommet dépilé qui ne sont pas déjà dans la pile et qui n'ont pas été déjà été traités (c-a-d qui ne sont pas dans **Lsommets**) ;
– on recommence à partir du point deux tant que la pile n'est pas vide.

Voici la traduction en pseudo-code du parcours en profondeur d'un graphe décrit ci-dessus.

P *est une pile vide*
Lsommets *est une liste vide*
On empile un sommet dans **P**
Tant que **P** *n'est pas vide*
..... *S=depile(***P**)
..... *On marque S dans* **Lsommets**
..... *On empile les voisins de S qui ne sont pas déjà présents dans* **P** *et qui n'ont*
..... *pas été déjà dépilés (c-a-d qui ne sont pas dans* **Lsommets**) ;
Fin Tant Que
On retourne **Lsommets**

Exercice 3
Créer une fonction nommée **parcours_en_profondeur*(graphe,sommet)*** qui parcourt en *profondeur* le graphe **graphe**, représenté par un dictionnaire, à partir du sommet **sommet**.
Cette fonction retournera la liste des sommets parcourus si le sommet **sommet** appartient bien au graphe et None sinon.

⇝ Rq. *On pourra coder en utilisant les fonctions relatives aux piles et/ou non.*

Exercice 4
Tester la fonction **parcours_en_profondeur** avec **(G_Test)** et chacun de ses sommets tour à tour.

III Recherche d'un chemin

L'objectif est de rechercher un *chemin* (s'il existe) entre deux sommets (**depart** et **arrivee**) dans un graphe à partir de l'algorithme (basé sur le parcours en profondeur) suivant :

P ⟵ *pile vide*
listesnouveauxsommetsvoisins ⟵ *liste vide*
Empiler le couple (**depart**, [**depart**]) *dans* **P**
Tant que **P** *n'est pas vide faire*
..... (**sommet**, **chemin**) ⟵ *tête de P (on dépile)*
.....**listesnouveauxsommetsvoisins** ⟵ *liste des sommets voisins à* **sommet** *qui ne sont pas dans* **chemin**
.....*Pour* **1sommet** *dans* **listesnouveauxsommetsvoisins**
.... *Si* **1sommet** = **arrivee** *alors*
....*retourner* **chemin** + [**1sommet**]
.... sinon
.... *empiler* (**1sommet**, **chemin** + [**1sommet**])
.... *FinSi*
..... *FinPour*
FinTantQue
Retourner **chemin**

Exercice 5
Créer une fonction nommée **chemin*(graphe,départ,arrivée)*** qui recherche un chemin reliant un sommet **départ** au sommet **arrivée** dans le graphe **graphe**, représenté par un dictionnaire. Cette fonction retournera la liste des sommets parcourus pour aller du **départ** jusqu'au **arrivée** lorsque cela est possible, et None sinon.

IV Graphe connexe

Exercice 6

1. Ecrire une fonction **ConnexeD(G)** d'argument un graphe simple non orienté **G** représenté par un dictionnaire, retournant un booléen répondant à la question "Le graphe est-il connexe ?".
2. La tester avec le graphe **(G_Test)**.

Exercice 7

1. Ecrire une fonction **ConnexeL(G)** d'argument un graphe simple non orienté **G** défini par sa liste d'adjacence, retournant un booléen répondant à la question "Le graphe est-il connexe ?".
2. La tester avec le graphe **(G_Test)**.

Exercice 8

1. Ecrire une fonction **ConnexeM(M)** d'argument une matrice **M** d'adjacence d'un graphe simple non orienté, retournant un booléen répondant à la question "Le graphe est-il connexe ?".
2. La tester avec le graphe **(G_Test)**.