université angers

FACULTÉ
DES SCIENCES
Unité de formation et de recherche

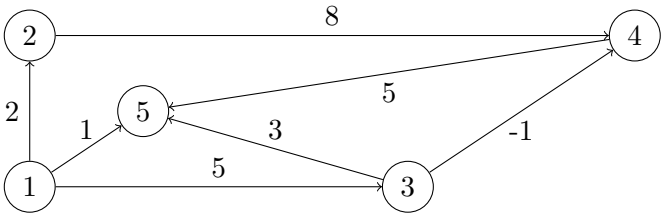
Licence 3 MPCIE
TP Graphes

TP1 - Représentation et parcours de graphe

Nous souhaitons représenter et parcourir les sommets d'un graphe. Pour cela nous allons implémenter deux algorithmes, un premier de recherche en profondeur et un second de recherche en largeur. Pour chaque exercice nous parcourons des graphes pondérés orientés ou non. Un graphe sera représenté par une matrice d'adjacence $\mathcal{M}$. Une valeur $\mathcal{M}_{ij}$ correspond au poids de l'arc entre les sommets i et j (ou à ∞ s'il n'existe pas d'arc).

Exercice 1 :

Donner la matrice d'adjacence pour le graphe suivant :



Exercice 2 :

Dans ce premier exercice, nous implémentons une structure de données permettant de représenter un graphe pondéré et orienté ou non sous la forme d'une matrice d'adjacence. Cette structure doit permettre la lecture d'un fichier en entrée (le format est décrit juste après), l'affichage de la matrice d'adjacence correspondante et le parcours du graphe.

Le format de description de graphe est le suivant :

```
c On peut faire des commentaires si la ligne commence par c (ou autre chose que o, v ou e)
c Si la ligne commence par o, elle indique si le graphe est orienté (go) ou non (autre chose)
c Si la ligne commence par v, elle indique le nombre de sommets (vertex)
c Si la ligne commence par e, elle indique un arc (edge), avec 3 éléments,
indice du sommet sortant, indice du sommet entrant et poids de l'arc
c Si le graphe est non orienté, l'arc est entre 2 sommets, qu'importe l'ordre

o go
v 5
e 1 2 2
e 1 3 5
e 2 4 8
e 1 5 1
e 3 5 3
e 4 5 2
e 3 4 -1
```

Pour utiliser la valeur ∞ en C++ on peut utiliser la bibliothèque `limits` :

```
#include <limits>
double inf = std::numeric_limits<double>::infinity();
```

Exercice 3 :

Nous souhaitons maintenant parcourir le graphe et afficher l'ensemble des sommets. Nous commençons par réaliser un parcours en profondeur, DFS (Depth-First Search). Pour vous aider voici la description de l'algorithme extrait de Wikipedia.

Description de l'algorithme DFS : Durant l'exploration, on marque les sommets afin d'éviter de re-parcourir des sommets parcourus. Initialement, aucun sommet n'est marqué.

```
explorer(graphe G, sommet s)
      marquer le sommet s
      afficher(s)
      pour tout sommet t fils du sommet s
            si t n'est pas marqué alors
                  explorer(G, t);
```

Le parcours en profondeur d'un graphe G est alors :

```
parcoursProfondeur(graphe G)
      pour tout sommet s du graphe G
            si s n'est pas marqué alors
                  explorer(G, s)
```

On notera qu'il est possible de l'implémenter itérativement à l'aide d'une pile LIFO contenant les sommets à explorer : on désempile un sommet et on empile ses voisins non encore explorés. (à implémenter si vous avez le temps).

Exercice 4 :

Nous souhaitons maintenant implémenter un autre type de parcours qui est le parcours en largeur (BFS, pour Breadth First Search). La description de l'algorithme extrait de wikipedia.

Description de l'algorithme BFS : Cet algorithme diffère de l'algorithme de parcours en profondeur par le fait que, à partir d'un nœud source S, il liste d'abord les voisins de S pour ensuite les explorer un par un. Ce mode de fonctionnement utilise donc une file dans laquelle il prend le premier sommet et place en dernier ses voisins non encore explorés.

Les nœuds déjà visités sont marqués afin d'éviter qu'un même nœud soit exploré plusieurs fois. Dans le cas particulier d'un arbre, le marquage n'est pas nécessaire.

Étapes de l'algorithme :

1 Mettre le nœud source dans la file.

2 Retirer le nœud du début de la file pour le traiter.

3 Mettre tous les voisins non explorés dans la file (à la fin).

4 Si la file n'est pas vide reprendre à l'étape 2.

Si vous avez le temps implémentez une version récursive de cet algorithme.

Rappels pour la lecture de fichiers en C++

Pour lire un fichier en C++ vous pouvez utiliser un flux avec la bibliothèque `fstream` et l'appel au constructeur de `std::ifstream`. Vous pouvez ensuite tester l'existance du fichier et récupérer chaque ligne dans une chaine de caractères avec une boucle et la fonction `getline(std::ifstream,std::string)`. Exemple ci-dessous :

```
std::ifstream fichier(input);

if (fichier) {
  unsigned int nbLine = 0;
  std::string ligne;
  while(getline(fichier,ligne)){
    ++nbLine;
    std::cout << "Ligne " << nbLine << ligne << std::endl;
  }
}
else {
  std::cout << "Erreur: impossible d'ouvrir le fichier" << std::endl;
}
```