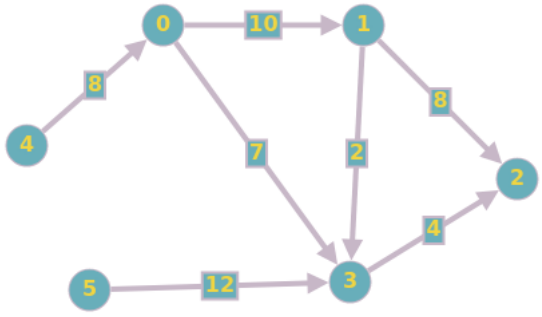
2022/2023 Université Paris 8
Algorithmique avancée Licence 3 Informatique

# TP n°3 : Graphes: parcours en largeur et algorithme de Dijkstra.

Vous testerez toutes vos fonctions sur le graphe suivant. Quels sont les accessibles depuis le sommet 0, les plus courts chemins depuis 0 ?




**Exercice 1:** *Avec les matrices d'adjacence.* Vous trouverez la structure de graphe utilisant une matrice d'adjacence dans le fichier *matriceadj.c*.

1. En utilisant la définition et les fonctions sur les files, écrire une fonction *accessibles* qui parcourt un graphe en largeur et renvoie un tableau indexé par tous les sommets, contenant 1 si le sommet est accessible depuis un sommet $s$, et 0 sinon.

2. En utilisant la fonction *minDistance* permettant de trouver le sommet non-encore parcouru situé à plus courte distance, écrivez une fonction *dijkstra_mat* implémentant l'algorithme de Dijkstra avec la version matricielle.

3. Mesurer le temps de calcul de l'algorithme de Dijkstra (version matricielle) sur des échantillons de graphes avec 100, 1000 et 10000 sommets générés aléatoirement avec la fonction *remplirgraphe*.

**Exercice 2:** *Avec les listes d'adjacence.* Vous trouverez la structure de graphe utilisant une liste d'adjacence dans le fichier *listeadj.c*.

1. Adapter la fonction *accessibles_mat* de l'exercice précédent pour écrire une fonction produisant le même résultat avec la structure de listes d'adjacence.

2. En utilisant la définition et les primitives de la structure de Tas du fichier *listeadj.c*, implémenter l'algorithme de Dijkstra utilisant une file de priorité pour aller chercher l'élément de la liste d'adjacence avec le plus petit poids.

3. Mesurer le temps de calcul de l'algorithme de Dijkstra (version liste) sur des échantillons de graphes avec 100, 1000 et 10000 sommets générés aléatoirement avec la fonction *remplirgraphe*.

**Exercice 3:** *Avec une matrice compressée ou représentation par triplets.*
Cette implémentation ressemble à la méthode d'implémentation de la matrice d'adjacence, en oubliant tous les 0. Pour ce faire, on va juste garder un vecteur contenant les triplets $(i, j, w)$ dès lors qu'il y a une arête de $i$ vers $j$ de poids $w$.

```
struct triplet {
   int i;
   int j;
   float poids;
} ;

typedef struct triplet triple_t;
struct gramaco {
   int nbs; /* nombre de sommets*/
   int nba; /* nombre d aretes*/
   triple * ares; /* vecteur d aretes*/
}
typedef struct gramaco gramaco_t;
```

1. Écrire les fonctions *newgraph*, *add_edge*, *print_graph* et *remplirgraphe* pour cette structure.

2. Implémenter le parcours en largeur et l'algorithme de Dijkstra pour cette structure.