

## TD/TP: Algorithmique et Programmation avancée

## 5: Algorithmes gloutons

**1: Problème de choix d'activités**

Ecrire pseudo code et coder.

**2: Arbre couvrant minimal (de Prim)**

On considère un graphe  $G$  connexe non-orienté et pondéré: chaque arête possède un poids qui représente le coût de cette arête (par exemple sa longueur). Dans un tel graphe, un arbre couvrant est un sous-graphe connexe sans cycle qui contient tous les sommets du graphe. Le poids d'un tel arbre est la somme des poids des arêtes qui le compose. Un arbre couvrant minimal est un arbre couvrant dont le poids est inférieur ou égal à celui de tous les autres arbres couvrants du graphe. Le problème est de trouver un tel arbre.

On pourra représenter un graphe par sa matrice d'adjacence pondérée  $A$ :

$$A_{ij} = \begin{cases} 0 & \text{si les sommets } i \text{ et } j \text{ ne sont pas liés} \\ w(i, j) & \text{si les sommets } i \text{ et } j \text{ sont liés par une arête de poids } w(i, j) \end{cases}$$

On considère l'algorithme suivant: L'arbre démarre d'un sommet racine  $a$  arbitraire puis croît jusqu'à être couvrant. Chaque étape ajoute une arête minimale à l'arbre  $E$  pour relier celui-ci à un sommet isolé.

Pour implémenter efficacement l'algorithme de Prim, l'important est de faciliter la sélection de la nouvelle arête à ajouter. Pendant l'exécution, tous les sommets qui n'appartiennent pas à l'arbre peuvent être stockés avec un attribut *poids* et *parent*, où *v.poids* est le poids minimal d'une arête reliant  $v$  à l'arbre  $E$  et *v.parent* est le sommet de  $E$  correspondant à cette arête.

1. Effectuer à la main, l'algorithme sur le graphe représenté par:

$$A = \begin{pmatrix} 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \\ 4 & 0 & 8 & 0 & 0 & 0 & 0 & 0 & 11 \\ 0 & 8 & 0 & 7 & 0 & 4 & 0 & 2 & 0 \\ 0 & 0 & 7 & 0 & 9 & 14 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 4 & 14 & 10 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 6 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 6 & 0 & 7 \\ 8 & 11 & 0 & 0 & 0 & 0 & 1 & 7 & 0 \end{pmatrix}$$

2. Proposer un pseudo-code, déterminer une borne sup sur la complexité (varie selon les structures de données utilisées), et coder en Python3.

Vous pourrez: utiliser judicieusement la structure de donnée "tas" en lui ajoutant quelques fonctions; utiliser une classe Sommet ou Graphe, ainsi que des attributs utiles à minimiser le temps d'exécution.

**3: Arbre couvrant minimal graphique**

Afficher l'arbre couvrant minimal du graphe complet sur un ensemble de points aléatoire uniforme de  $[0, 1]^2$  en utilisant la classe Point du chapitre précédent et le module matplotlib. Le poids des arêtes correspond à la distance euclidienne.

**4: Plus court chemin depuis une source (Dijkstra)**

On s'intéresse au problème des plus courts chemins à partir d'une source donnée dans un graphe. L'algorithme de Dijkstra que l'on va étudier fonctionne sur les graphes où les poids sont positifs.

Soit  $G(V, E, w)$  un graphe orienté, dont les sommets sont numérotés  $V = a, b, \dots$ , et dont la source est le sommet  $a$ . L'algorithme, entièrement glouton, consiste à ajouter les sommets de  $V$  un par un dans l'ordre de leur distance au sommet  $a$ .