Dans ce TP nous allons nous intéresser au problème du voyageur de commerce. Dans un premier temps nous allons étudier l'algorithme de Prim qui permet d'obtenir un arbre couvrant minimal.

Les graphes considérés seront des graphes non orientés et pondérés. Ils seront implémentés par des listes d'adjacences de couples :

```
type graphe = (int*int) list array
```

I - L'algorithme de Prim

On considère un graphe non orienté supposé connexe $G = (V, E)$ muni d'une fonction de pondération $w : E \to \mathbb{R}$.

Le but de l'algorithme de Prim est alors de déterminer un sous-arbre couvrant minimal, c'est-à-dire un sous-graphe $A = (V', E')$ de G où :

- le sous-graphe A est couvrant c'est-à-dire $V' = V$;
- le sous-graphe A est un arbre (c'est-à-dire qu'il est connexe et acyclique) ;
- il est minimal ce qui signifie que si $A_1 = (V_1, E_1)$ est un sous-arbre couvrant de G alors

$$\sum_{e \in E'} w(e) \leq \sum_{e \in E_1} w(e)$$

L'algorithme est le suivant :

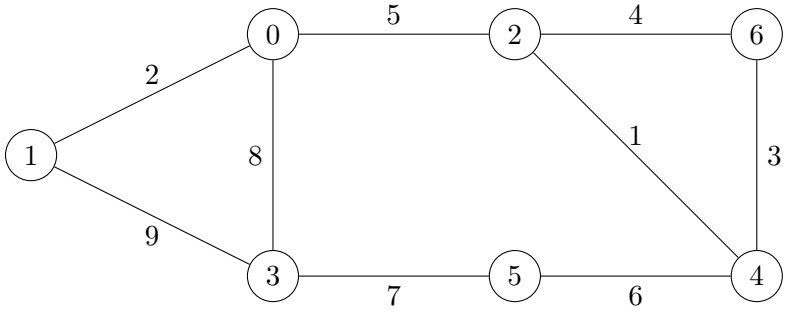
Algorithme 1 : Algorithme de Prim

Données : Graphe pondéré $G = (S, A)$
Choisir un sommet $s \in S$
$V' \leftarrow \{s\}$; $E' \leftarrow \emptyset$; $F = \{$les arêtes issues de $s\}$
tant que $V' \neq V$ **faire**
    Choisir une arête e dans F de poids minimal
    Enlever e de F
    si *l'arête ne rejoint pas deux sommets de* V' **alors**
        Ajouter e à E'
        Ajouter à V' le sommet u qui est source ou destination de e et qui n'est pas dans V'
        Ajouter à F toutes les arêtes issues de u
    fin
fin
Résultat : Le graphe $A = (V', E')$

Nous admettrons que cet algorithme renvoie bien un arbre couvrant minimal.

Quand on veut implémenter cet algorithme on utilise une file de priorité pour l'ensemble F des arêtes afin de pouvoir efficacement choisir une arête de poids minimal. Pour faire plus simple dans ce TP nous utiliserons une liste triée par les poids de type `(int*int*int) list`.

1) On considère le graphe suivant



Déterminer un sous-arbre couvrant minimal.

2) Écrire une fonction `insere : ('a * 'b * 'c) list -> 'a * 'b * 'c -> ('a * 'b * 'c) list` qui permet d'insérer à sa place un élément `(u,v,w)` dans une liste triée. Le tri étant fait sur le troisième élément du triplet.

3) Écrire une fonction `prim : (int * int) list array -> (int * int) list array` qui renvoie un sous-arbre couvrant de poids minimal. Vérifier votre fonction avec le graphe donné à la question 1)
L'ensemble V' pourra être implémenté par un tableau de booléen ; l'ensemble E' pourra être implémenté par un tableau de listes d'adjacence de type `(int*int) list array` ; l'ensemble F pourra être implémenté par un référence de liste.

II - Le problème du voyageur de commerce

On considère un ensemble X de n points $X = \{v_0, \dots, v_{n-1}\}$ et on suppose que l'on dispose d'une distance $d : X \times X \to \mathbb{R}_+$. A chaque permutation σ de S_n on associe la longueur $\ell(\sigma)$ du chemin

$$v_{\sigma(0)} \to v_{\sigma(1)} \to \cdots \to v_{\sigma(n-1)} \to v_{\sigma(0)}$$

On a donc en notant $\sigma(n) = \sigma(0)$:

$$\ell(\sigma) = \sum_{i=0}^{n-1} d(v_{\sigma(i)}, v_{\sigma(i+1)})$$

On cherche à déterminer la permutation σ telle que $\ell(\sigma)$ soit minimal.

Dans la suite, on supposera que les points appartiennent à $\mathbb{Z}^2$ et on utilisera comme distance

$$d((x_1, y_1), (x_2, y_2)) = |x_2 - x_1| + |y_2 - y_1|$$

Notre ensemble de points sera stocké sous la forme d'un tableau `points` de type `int*int array` tel que `t.(i)` contienne les coordonnées des v_i. Pour les exemples on prendra
`points = [|(-2, 5); (6, -6); (-10, -7); (-1, 0); (5, 2); (7, 1); (7, 7); (-7, -10)|]`

Une permutation σ de S_n sera donnée par un tableau `sigma` de type `int array` tel que `sigma.(i)` soit la valeur de $\sigma(i)$.

4) Écrire une fonction `distance : int*int -> int*int -> int` qui associe à un couple de points la distance entre les deux points.

5) Écrire une fonction `longueur_chemin : int*int array -> int array -> int` tel que `longueur_chemin points sigma` renvoie la valeur de $\ell(\sigma)$ où les points sont codés par le tableau `points` et σ par `sigma`.

Pour déterminer une « bonne permutation » nous allons utiliser l'algorithme suivant :

- On construit le graphe complet $G = (X, E)$ où $E = X \times X \setminus \{(v, v)\ ,\ v \in X\}$. On pondère ce graphe par la fonction d.
- On construit un arbre couvrant minimal.
- On parcourt en profondeur cet arbre (en partant de n'importe quel sommet). On considère alors les points de X dans l'ordre dans lequel ils sont parcourus.

6) Écrire la fonction `construction_graphe : (int*int) array -> (int*int) list array` telle que `construction_graphe points` renvoie les listes d'adjacence du graphe défini ci-dessus.

7) En utilisant la fonction `prim` en déduire une fonction `voyageur : (int*int) array -> int array` telle que `voyageur points` renvoie la permutation renvoyée par l'algorithme ci-dessus.

8) Tester votre fonction sur les points de l'exemple. La permutation obtenue est-elle optimale ?

9) Montrer que si on note σ_0 la permutation renvoyée, $\ell(\sigma_0) \leq 2 \min_{\sigma \in S_n} \ell(\sigma)$.

10) Écrire une fonction `voyageur_exhaustif : (int*int) array -> int array` telle que `voyageur_exhaustif points` renvoie la permutation optimale pour le problème. On réalisera une recherche exhaustive.