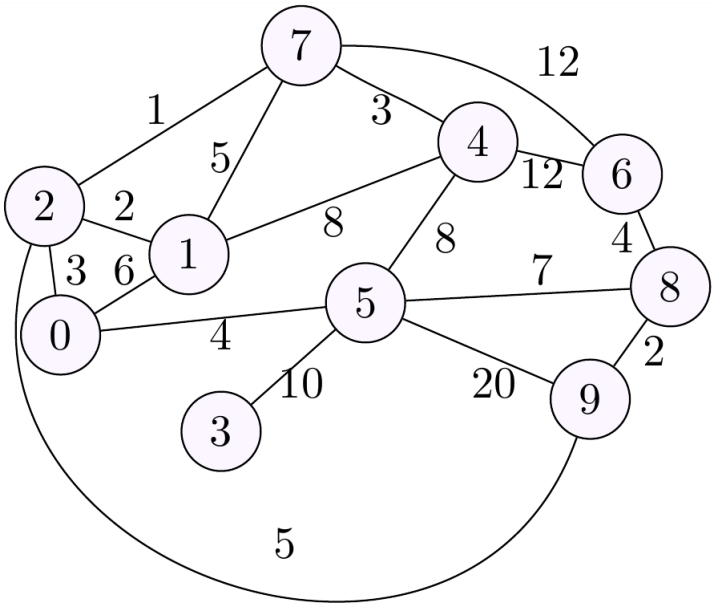
TP n°9 – Arbres couvrants minimaux

1 Un exemple

- **Q1.** Effectuer l'algorithme de Kruskal sur l'exemple suivant et dessiner l'arbre couvrant résultant.



2 Implémentation de Kruskal

Dans ce TP on considère ses graphes connexes, non-orientés et pondérés par des entiers. On représente un tel graphe à n sommets en Ocaml par le type `(int*int) list array` : un tableau de n cases contenant les listes des couples (poids de l'arête, voisin).

Remarque : On choisit de mettre d'abord le poids dans le couple pour faire usage de l'ordre lexicographique.

Par exemple si `adj[0]` contient `(25, 3)` alors il existe une arête de poids 25 entre le noeud 0 et le noeud 3.

- **Q2.** Implémenter le tri fusion pour une liste de couples. On triera selon l'ordre des premiers éléments des couples.
- **Q3.** Écrire une fonction qui crée la liste des arêtes et la trie en fonction des poids.
- **Q4.** En utilisant une implémentation d'union-find (reprendre celle du TP précédent), implémenter l'algorithme de Kruskal
- **Q5.** Stocker en mémoire le graphe de la partie 1 et tester l'algorithme dessus.

3 Unicité des arbres couvrants

Dans cette section on s'intéresse à la question suivante : "existe-t-il un unique arbre couvrant minimal ?"

- **Q6.** Trouver un exemple de graphe qui admet plusieurs arbres couvrants minimaux.

Soit $G = (S, A, p)$ un graphe non orienté pondéré connexe avec plus de 3 arêtes. On note T un arbre couvrant minimal de G.

On suppose qu'il n'y a pas deux arêtes de même poids. On veut montrer que sous cette hypothèse il y a un seul arbre couvrant minimal pour G.

- **Q7.** Soit T' un arbre couvrant minimal de G. Montrer que $T = T'$.

4 Spécificités de certaines arêtes

Supposons d'abord que le poids le plus petit ne soit porté que par une arête : il existe une (et une seule) arête de poids minimal.

- **Q8.** Montrer qu'un arbre couvrant minimal contient nécessairement cette arête.
- **Q9.** En rajoutant l'hypothèse que le deuxième poids le plus petit ne soit également porté que par une seule arête, montrer que la deuxième plus petite arête est également dans n'importe quel arbre couvrant minimal.

- **Q10.** La troisième plus petite arête (à nouveau on suppose qu'elle est la seule à porter son poids) est elle nécessairement dans tout arbre couvrant de poids minimal ?

En rapport avec la réponse à la question précédente, on peut prouver la propriété suivante :

- **Q11.** Montrer que pour chaque cycle dans le graphe, les arbres couvrants minimaux ne contiennent pas l'arête de poids maximal du cycle (à nouveau on suppose qu'elle est unique).
- **Q12.** Un arbre couvrant minimal contient-il l'arête minimale de chaque cycle ?

5 L'algorithme de Prim

L'algorithme de Prim est un autre algorithme pour calculer un arbre couvrant minimal. Comme Kruskal il s'agit d'un algorithme glouton.

Le code d'une file de priorité minimum vous est fourni dans le fichier file.ml. Copiez-le dans le même dossier que votre fichier de TP et mettez la ligne suivante en haut de votre fichier de TP : `#use "file.ml"`.

Si vous êtes sur VScode, lorsque vous interprétez, utilisez la comande **`ocaml tp09.ml file.ml`**. Sous emacs interprétez comme d'habitude.

Vous pouvez désormais utiliser les fonctions suivantes (la file est implémentée avec des tas c'est pour cela qu'il y a des arbres binaires) :

- `creer : 'a array -> 'a arbre_binaire` pour créer la file à partir d'un tableau,
- `ajoute : 'a arbre_binaire->'a->'a arbre_binaire`,
- `retire : 'a arbre_binaire -> 'a*'arbre_binaire`,
- `regarde : 'a arbre_binaire -> 'a`,
- `est_vide : 'a arbre_binaire -> bool` .

L'algorithme est le suivant :

- On part d'un arbre vide T et on garde en mémoire l'ensemble des sommets déjà ajoutés à l'arbre. La sélection de la prochaine arête se fera grâce à une file de priorité.
- On ajoute un sommet s quelconque à l'arbre T pour démarrer. On ajoute à la file de priorité toutes les arêtes sortantes de s.
- Tant qu'il reste des arêtes accessibles depuis T non encore considérées on en choisit une de poids minimal. (on en récupère une dans la file de priorité donc)
  Si une des deux extrémités u de cette arête n'est pas dans T alors on ajoute u à T et toutes les arêtes qui vont avec (pensez à gérer les répétitions d'arêtes intelligemment) et on boucle.

- **Q13.** Implémenter l'algorithme de Prim.
- **Q14.** Tester sur l'exemple de la partie 1 pour vérifier.