Exercice 1. Étude de l'algorithme de Berge

Soit $G = (S_1 \cup S_2, A)$ un graphe biparti sur lequel on applique l'algorithme de Berge afin d'en déterminer un couplage maximal. L'exécution fournit donc des ensembles $M_0, M_1, \ldots, M_k \subset A$ ainsi que des chemins $\gamma_0, \gamma_1, \cdots, \gamma_{k-1}$ définis par :

- $M_0 = \varnothing$
- Pour chaque $i \in [\![0, k-1]\!]$:
  - γ_i est un chemin M_i-augmentant de G.
  - $M_{i+1} = M_i \Delta A_i$ avec A_i l'ensemble des arêtes de γ_i.

Terminaison et complexité de l'algorithme de Berge. Pour rappel, l'algorithme de Berge s'arrête lorsqu'il n'existe plus de chemin M_i-augmentant dans G.

1. Soit M un couplage de G, soit γ un chemin M-augmentant et $a_1, a_2, \ldots, a_\ell$ les arêtes de γ (ℓ est donc la longueur de γ). On pose :
$$M' = M\Delta\{a_1, a_2, \ldots, a_\ell\}.$$
   (a) Montrer que $|M'| = |M| + 1$.
   (b) Montrer que M' est un couplage de G.
   (c) En déduire que l'algorithme de Berge termine.
2. Quelle est la complexité de l'algorithme ?

Correction de l'algorithme de Berge. Voici quelques définitions qui seront utiles pour la preuve de correction :

– Étant donné $G_0 = (S_0, A_0)$ un graphe et $R \subset S_0$ un sous-ensemble des sommets de G_0. On appelle ***sous-graphe induit par*** R et on note $I(G_0, R)$ le graphe (R, A') où :
$$A' = \Big\{\{u_1, u_2\} \in A_0 : u_1 \in R, u_2 \in R\Big\}.$$

– Soit $G_0 = (S_0, A_0)$ un graphe. On dira que G_0 est de type 1, 2 ou 3 dans les cas suivants :
  (type 1) G_0 contient un unique sommet et aucune arête.
  (type 2) Les sommets et les arêtes de G_0 forment un chemin élémentaire de longueur strictement positive.
  (type 3) Les sommets et les arêtes de G_0 forment un cycle (de longueur strictement positive).

Dans la suite, on considère $G = (S_1 \cup S_2, A)$ un graphe biparti et M_1, M_2 deux couplages sur G. On définit un nouveau graphe biparti G' par :
$$G' = (S_1 \cup S_2, M_1 \Delta M_2).$$

3. (a) Soit γ un chemin élémentaire de G' et a_1, a_2 deux arêtes consécutives dans γ. Montrer que :
$$a_1 \in M_1 \Rightarrow a_2 \in M_2 \qquad \text{et} \qquad a_1 \in M_2 \Rightarrow a_2 \in M_1$$
   (b) Soit $R \subset S_1 \cup S_2$ une composante connexe de G'. Montrer que $I(G', R)$ est de type 1, 2 ou 3.
4. Supposons $|M_2| > |M_1|$.
   (a) Montrer que G' contient strictement plus d'arêtes issues de M_2 que d'arêtes issues de M_1.
   (b) Montrer qu'il existe $R \subset S_1 \cup S_2$ une composante connexe de G' telle que les sommets et les arêtes de $I(G', R)$ forment un chemin élémentaire de longueur strictement positive contenant plus d'arêtes de M_2 que d'arêtes de M_1.
   (c) En déduire que G contient un chemin M_1-augmentant.

5. Prouver le lemme suivant (Claude Berge, 1957).
   Soit G un graphe biparti et M un couplage sur G. Alors :

$$M \text{ est un couplage maximal} \Leftrightarrow \text{il n'existe pas de chemin } M\text{-augmentant dans } G.$$

6. À l'aide d'un invariant de boucle, montrer la correction de l'algorithme de Berge (il faut donc montrer qu'il renvoie un couplage maximal du graphe biparti donné en entrée).

Exercice 2. Preuve de validité de l'algorithme de Kruskal

Le but de cet exercice est de montrer que l'algorithme de Kruskal appliqué à un graphe $G = (S, A)$ renvoie bien un arbre couvrant de poids minimal.

Algorithme de Kruskal.

Entrées: $G = (S, A, \omega)$ un graphe pondéré connexe.
Sortie: T un arbre couvrant de poids de minimal de G

Initialement : $T = (S, \varnothing)$
Soit L la liste des arêtes de G triées par ordre croissant des poids
pour chaque arête $\{u, v\}$ dans L **faire**
  si u et v sont dans la même composante connexe **alors**
    On ne fait rien
  sinon
    On ajoute $\{u, v\}$ dans T
  fin si
fin pour

Dans un premier temps, on aura besoin de montrer deux lemmes concernant les graphes.

Définition 1. Soit G un graphe et u un sommet. On note $\mathscr{C}(G, u)$ la composante connexe de u dans G, c'est à dire :

$$\mathscr{C}(G, u) = \Big\{ v \in S : \text{il existe un chemin de } u \text{ à } v \text{ dans } G \Big\}$$

Lemme 2. *Soit $G_1 = (S, A_1)$ un graphe connexe, soit $\{u_1, v_1\}$ une arête de G_1 et $G_2 = (S, A_2)$ le graphe G_1 privé de l'arête $\{u_1, v_1\}$. On choisit $u_2 \in \mathscr{C}(G_2, u_1)$, $v_2 \in \mathscr{C}(G_2, v_1)$, et on pose $G_3 = (S, A_3)$ le graphe G_2 auquel on a ajouté l'arête $\{u_2, v_2\}$ (notez que $G_3 = G_2$ si $\{u_2, v_2\}$ est déjà une arête de G_2). Alors G_3 est connexe.*

Lemme 3. *Soit $G_1 = (S, A_1)$ un graphe sans cycle et u_1, v_1 deux sommets appartenant à deux composantes connexes différentes de G_1. Soit $G_2 = (S, A_2)$ le graphe G_1 auquel on a ajouté l'arête $\{u_1, v_1\}$. Alors G_2 ne contient pas de cycle.*

1. Montrer les deux lemmes ci-dessus.

Pour montrer la correction de l'algorithme de Kruskal, on utilise un invariant de boucle. On note $T_0, T_1, \ldots, T_k$ les graphes construits par l'algorithme. Ainsi, $T_0 = (S, \varnothing)$; pour tout i, T_{i+1} est le graphe T_i auquel on a ajouté une arête ; et T_k est l'arbre couvrant renvoyé par l'algorithme. Pour chaque i, on note A_i l'ensemble des arêtes de T_i. On pose :

$$(\mathcal{P}_i) : \text{Il existe } T = (S, B) \text{ un arbre couvrant de poids minimal de } G \text{ tel que } A_i \subset B$$

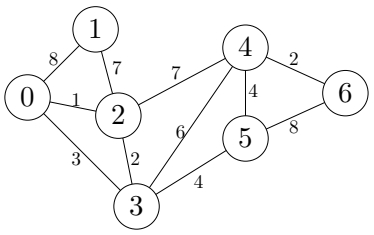
2. (a) Montrer $(\mathcal{P}_0)$.
   (b) Soit $i \in [\![0; k-1]\!]$. Montrer que $(\mathcal{P}_i)$ implique $(\mathcal{P}_{i+1})$.
   (c) En déduire la validité de l'algorithme de Kruskal.

Exercice 3. Algorithme de Prim

On considère $G = (S, A, \omega)$ un graphe connexe dont les sommets sont étiquetés par $0, 1, \ldots, n-1$ et dont les arêtes sont pondérées par des poids strictement positifs. En Caml, un graphe est représenté par matrice d'adjacence :

```
type graphe1 = int array array;;
```

Voici un exemple de graphe et sa représentation OCaml :




```
[| [|0;8;1;3;0;0;0|];
   [|8;0;7;0;0;0;0|];
   [|1;7;0;2;7;0;0|];
   [|3;0;2;0;6;4;0|];
   [|0;0;7;6;0;4;2|];
   [|0;0;0;4;4;0;8|];
   [|0;0;0;0;2;8;0|] |];;
```

Comme l'algorithme de Kruskal, l'algorithme de Prim permet de construire un arbre couvrant de poids minimal d'un graphe connexe $G = (S, A)$. En voici le principe :

- ⋆ Soit r un sommet quelconque de G. Initialement, le graphe T contient uniquement le sommet r, c'est à dire que $T = (\{r\}, \varnothing)$.
- ⋆ Soit S' l'ensemble des sommets de T. Pour chaque $s \in S \setminus S'$, on pose :

$$D(s) = \Big\{\omega(s,t) : t \in S' \text{ et } \{s,t\} \text{ est une arête de } G\Big\}.$$

où $\omega(s,t)$ est le poids de l'arête $\{s,t\}$. On définit alors :

$$d(s) = \begin{cases} +\infty & \text{si } D(s) = \varnothing, \\ \min(D(s)) & \text{sinon.} \end{cases}$$

- ⋆ Tant que $S' \neq S$:
  - Soit $s_0 \in S \setminus S'$ tel que $d(s_0)$ est minimum ($d(s_0) \neq +\infty$ car G est connexe). Soit $t \in S'$ l'un des sommets tels que $\omega(s_0, t) = d(s_0)$.
  - On ajoute le sommet s_0 et l'arête $\{s_0, t\}$ au graphe T.

Implémenter l'algorithme de Prim en OCaml (si besoin, vous pouvez lire les indications ci-dessous).

Indications (essayez de résoudre l'exercice sans lire ce qui suit). Tout au long de l'algorithme, on va manipuler une variable « `fp: file_priorite` » :

```
type voisin_plus_proche = int option;;
```

```
type file_prio =
  (int * voisin_plus_proche) list ref;;
```

À chaque étape, la liste `fp` est de taille card$(S \setminus S')$. De plus, pour chaque sommet `s` de $S \setminus S'$, la file `fp` contient un couple `(s,t0)` où :

→ Si `s` n'a pas de voisin dans S' alors `t0` vaut `None`.

→ Sinon, `t0` vaut « `Some t` » où `t` est le sommet de S' qui minimise ω`(s, t)`.

À chaque étape de l'algorithme, il suffit de récupérer l'élément de `fp` correspondant à un poids minimal. Cet élément correspond à une arête $\{s_0, t\}$, qui est l'arête à ajouter à T.