Exercice 1. Implémentation de l'algorithme de Berge

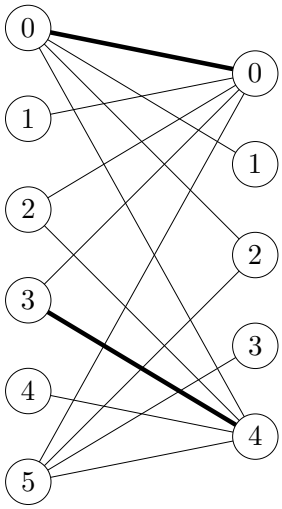

FIGURE 1

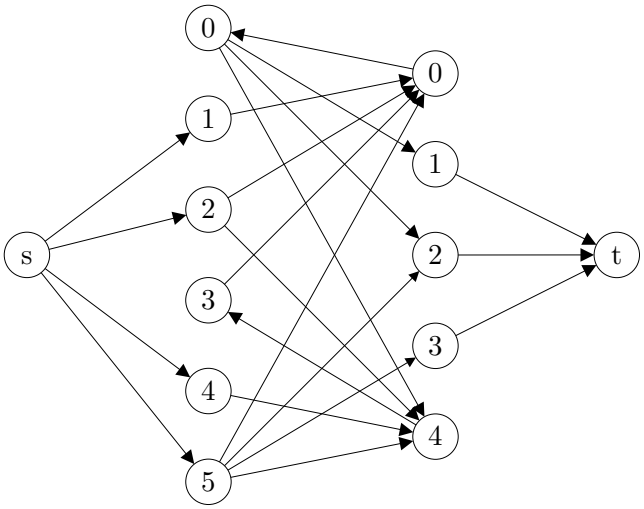

FIGURE 2

Le but de l'exercice est d'implémenter l'algorithme de Berge en OCaml. Soit $G = (S_0 \cup S_1, A)$ le graphe biparti dont on cherche un couplage maximal, soit $n_0 = |S_0|$ et $n_1 = |S_1|$. On suppose que les sommets de S_0 sont numérotés de 0 à $n_0 - 1$ et que les sommets de S_1 sont numérotés de 0 à $n_1 - 1$.

```
type graphe_bip = {
    n0: int;
    n1: int;
    aretes: (int*int) list;
  };;
```

Le graphe biparti est représenté par une variable de type `graphe_bip`. Le champ `aretes` contient une liste dont chaque couple est de la forme (i, j) avec i le numéro d'un sommet de S_0 et j un numéro d'un sommet de S_1. Par exemple, le graphe de la figure 1 correspond à :

```
let g_bip_fig1 = {
    n0 = 6; n1 = 5;
    aretes = [(0,0); (0,1); (0,2); (0,4); (1,0);
              (2,0); (2,4); (3,0); (3,4);
              (4,4); (5,0); (5,2); (5,3); (5,4)]
  };;
```

1. Écrire une fonction « `berge: graphe_bip -> (int*int) list` » qui exécute l'algorithme de Berge sur le graphe biparti donné en entrée et renvoie un couplage maximal sous forme d'une liste de couples. Si besoin, on pourra lire les indications ci-dessous.

Indications (essayez de résoudre l'exercice sans lire ce qui suit). La principale difficulté est, à partir d'un couplage M, de déterminer une chaîne M-augmentante dans G. Comme expliqué dans le cours, ceci est équivalent à déterminer un chemin dans un graphe orienté bien choisi noté $\alpha(G, M)$. Par exemple, si G est le graphe de la figure 1 et M correspond aux arêtes en gras, alors $\alpha(G, M)$ est le graphe de la figure 2. Pour représenter $\alpha(G, M)$, on utilisera le type :

```
type graphe = {
    mat_adj: int array array;
    s_adj: bool array;
    t_adj: bool array;
  };;
```

Soit `g` de type `graphe`. Dans les explications qui suivent, on note $\texttt{u}_\texttt{i}$ le sommet de S_0 numéroté par `i` et $\texttt{v}_\texttt{j}$ le sommet de S_1 numéroté par `j`. Alors :

- ⋆ `g.mat_adj` est un tableau de tableau de taille $n_1 \times n_2$ tel que :
  - `g.mat_adj.(i).(j)` vaut `1` s'il y a un arc du sommet $\texttt{u}_\texttt{i}$ au sommet $\texttt{v}_\texttt{j}$.
  - `g.mat_adj.(i).(j)` vaut `-1` s'il y a un arc du sommet $\texttt{v}_\texttt{j}$ au sommet $\texttt{u}_\texttt{i}$.
  - `g.mat_adj.(i).(j)` vaut `0` sinon.
- ⋆ `g.s_adj` est un tableau de taille n_0 tel que `g.s_adj.(i)` indique s'il y a un arc entre le sommet `s` et $\texttt{u}_\texttt{i}$.
- ⋆ `g.t_adj` est un tableau de taille n_1 tel que `g.t_adj.(j)` indique s'il y a un arc entre le sommet $\texttt{v}_\texttt{j}$ et `t`.

Par exemple, le graphe de la figure 2 correspond à :

```
let g_fig2 = {
    mat_adj = [|[|-1; 1; 1; 0; 1|];
                [| 1; 0; 0; 0; 0|];
                [| 1; 0; 0; 0; 1|];
                [| 1; 0; 0; 0;-1|];
                [| 0; 0; 0; 0; 1|];
                [| 1; 0; 1; 1; 1|]|];
    s_adj = [|false; true; true; false; true; true|];
    t_adj = [|false; true; true; true; false|]
  };;
```

On peut se convaincre assez facilement que le fait de connaître le graphe $\alpha(G, M)$ est suffisant pour connaître le couplage M. Ainsi, M ne sera jamais représenté explicitement en OCaml ; on se contentera de manipuler une variable de type `graphe` représentant $\alpha(G, M)$.

2. Écrire une fonction « `make_graphe : graphe_bip -> graphe` » qui prend en entrée un graphe biparti G et renvoie le graphe $\alpha(G, \varnothing)$. Pour exploiter la liste du champ `aretes`, on utilisera la fonction `List.iter` qui prend en entrée une fonction `f` ainsi qu'une liste `[e1; e2; e3; ...; en]` et exécute la suite d'instructions :

   ```
   f e1; f e2; f e3; ...f en;
   ```

3. Écrire une fonction « `get_couplage : graphe -> (int * int) list` » qui prend en entrée un graphe $\alpha(G, M)$ et renvoie le couplage M. Par exemple (l'ordre des éléments de la liste n'a pas d'importance) :

   « `get_couplage g_fig2` » vaut `[(3,4); (0,0)]`,

   « `get_couplage (make_graphe g_bip_fig1)` » vaut `[]`.

4. Supposons avoir à notre disposition un graphe orienté $\alpha(G, M)$ représenté par « `g: graphe` » ainsi qu'une chaîne M-augmentante donnée par une liste de sommets « `c: int list` ». Dans l'algorithme de Berge, on obtient alors un nouveau couplage $M' = M \Delta A$ où A est l'ensemble des arêtes apparaissant dans la chaîne M-augmentante.

   Écrire une fonction `maj_graphe` qui prend en entrée `g` ainsi que `c`, et modifie `g` afin qu'il représente $\alpha(G, M')$.

5. (a) Écrire une fonction « `list_init : int -> (int -> bool) -> int list` » qui prend en entrée un entier `n` ainsi qu'un prédicat « `pred: int -> bool` », et renvoie une liste contenant tous les entiers `i` $\in [\![0; n]\!]$ tels que « `pred i` » vaut `true`.
   (b) Écrire une fonction « `get_CMA : graphe -> int list` » qui prend en entrée $\alpha(G, M)$, et renvoie une chaîne M-augmentante. On utilisera autant que possible la fonction `list_init`.
6. Répondre à la question 1.

Exercice 2. 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 chaînes $\gamma_0, \gamma_1, \cdots, \gamma_{k-1}$ définis par :

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

$$M_i \Delta A_i = (M_i \cup A_i) \setminus (M_i \cap A_i).$$

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

1. Soit M un couplage de G, soit γ une chaîne M-augmentante 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 :

- Une chaîne est dite ***élémentaire*** si elle ne repasse pas deux fois par le même sommet.
- 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 une chaîne 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 une chaîne de G' et a_1, a_2 deux arêtes consécutives dans γ. Montrer que :

$$a_1 \in M_1 \Rightarrow a_2 \in M_2 \qquad \text{et} \qquad a_2 \in M_1 \Rightarrow a_1 \in M_2$$

   (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 une chaîne élémentaire de longueur strictement positive contenant plus d'arêtes de M_2 que d'arêtes de M_1.
   (c) En déduire que G contient une chaîne M_1-augmentante.

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

   M est un coupage maximal $\Leftrightarrow$ il n'existe pas de chaîne M-augmentante 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).