Partie II - Traversée de rivière

Cette partie comporte des questions nécessitant un **code OCaml**.

Dans une vallée des Alpes, un passage à gué fait de cailloux permet de traverser la rivière. Deux groupes de randonneurs arrivent simultanément sur les berges gauche et droite de cette rivière et veulent la traverser. Le chemin étant très étroit, une seule personne peut se trouver sur chaque caillou de ce chemin (**figure 1**). Un randonneur sur la berge de gauche peut avancer d'un caillou (vers la droite sur la **figure 1**) et sauter par dessus le randonneur devant lui (un caillou à droite) si le caillou où il atterit est libre. De même, chaque randonneur de la berge de droite peut avancer d'un caillou (vers la gauche sur la **figure 1**) et sauter par dessus le randonneur devant lui, dans la mesure où le caillou sur lequel il atterit est libre. Une fois engagés, les randonneurs ne peuvent pas faire marche arrière. De plus, pour simplifier, on suppose qu'une fois tous les randonneurs sur le chemin, il ne reste qu'un caillou de libre.

Figure 1 - Les randonneurs et le chemin de cailloux

Le chemin de cailloux est défini par un tableau d'entiers :

```
type chemin_caillou = int array
```

Dans ce tableau, un randonneur venant de la berge de gauche est représenté par un 1, un randonneur issu de la berge de droite par un 2 et un caillou libre par un 0.

Q14. Écrire une fonction de signature `caillou_vide : chemin_caillou -> int` qui détermine la position du caillou inoccupé.

Q15. Écrire une fonction de signature `echange : chemin_caillou -> int -> int -> chemin_caillou` qui permute les valeurs codées sur deux cailloux. Le tableau d'entiers initial représentant le chemin n'est pas modifié. On pourra utiliser ici la fonction `copy` du module `Array`.

Q16. Écrire une fonction de signature `randonneurG_avance : chemin_caillou -> bool` qui teste si, parmi les randonneurs venant de la berge de gauche, il en existe un qui puisse avancer (vers la droite).

Q17. Écrire une fonction de signature `randonneurG_saute : chemin_caillou -> bool` qui teste si, parmi les randonneurs venant de la berge de gauche, il en existe un qui puisse sauter (vers la droite) au-dessus d'un randonneur.

On supposera dans la suite les fonctions de signature `randonneurD_avance : chemin_caillou -> bool` et `randonneurD_saute : chemin_caillou -> bool` écrites de manière similaire pour les randonneurs venant de la berge de droite.

Q18. Écrire une fonction de signature `mouvement_chemin : chemin_caillou -> chemin_caillou list` qui, en fonction de l'état du chemin, calcule l'état suivant après les opérations suivantes (si elles sont permises) :

(i). déplacement d'un randonneur venant de la berge de gauche,

(ii). déplacement d'un randonneur venant de la berge de droite,

(iii). saut d'un randonneur venant de la berge de gauche,

(iv). saut d'un randonneur venant de la berge de droite.

On donne la syntaxe OCaml pour créer une liste de N entiers i : `List.init N (fun x->i)`.
Par exemple, `List.init 5 (fun x->2);;` renvoie [2 ; 2 ; 2 ; 2 ; 2].

Q19. Écrire une fonction de signature `passage : int -> int`, utilisant la question précédente, telle que l'appel `passage nG nD` résout le problème de passage de `nG` randonneurs venant de la berge de gauche et `nD` randonneurs venant de la berge de droite. Par exemple, `passage 3 2` permet de passer de [1 ;1 ;1 ;0 ;2 ;2] à [2 ;2 ;0 ;1 ;1 ;1].

Partie III - Calcul d'une coupe minimum d'un graphe

Un agent secret a pour mission de perturber le réseau de communications ennemi en coupant certains fils dans le réseau. Ayant peu de moyens, l'agent a pour consigne de couper le moins de fils possibles pour accomplir cette tâche. Le réseau de communications est modélisé à l'aide d'un *multigraphe* non orienté $G = (S, A)$, où S est l'ensemble des sommets du graphe et A l'ensemble de ses arêtes. Résoudre le problème de l'agent secret, c'est rechercher une *coupe minimum* dans G.

Définition 4 (Multigraphe)
Un multigraphe est un graphe dans lequel des couples de mêmes sommets peuvent être reliés par plus d'une arête.

Dans toute la suite, on considérera les multigraphes sans arêtes du type (s, s) (boucles).

Définition 5 (Coupe)
Une coupe d'un multigraphe non orienté $G = (S, A)$ est une partition non triviale (S_1, S_2) de S, c'est-à-dire telle que $S_1 \cup S_2 = S, S_1 \cap S_2 = \emptyset$ et $S_1, S_2 \neq \emptyset$. On dira que les arêtes reliant S_1 à S_2 sont les arêtes de la coupe.

La *taille* d'une coupe (S_1, S_2) est définie par $|(S_1, S_2)| = |\{(s, t) \in A, s \in S_1, t \in S_2\}|$. Autrement dit, c'est le nombre d'arêtes de G qui relient S_1 et S_2.

Définition 6 (Coupe minimum)
Une coupe minimum est une coupe de taille minimale.

Pour trouver une coupe minimum d'un multigraphe $G = (S, A)$, on pourrait énumérer l'ensemble des coupes possibles (il y en a $2^{|S|}$...), ou encore utiliser un algorithme de flot (le plus efficace étant en $O(|S|^3)$). Nous proposons dans la suite un algorithme probabiliste, l'algorithme de Karger, basé sur la notion de contraction d'arête.

III.1 - Contraction d'arête

Soient $G = (S, A)$ un multigraphe et $a = (s, t) \in A$ une arête de G de sommets s et t. *Contracter* l'arête a consiste à :

(i). créer un nouveau sommet $u = (st)$ dans S. u est un *supersommet* du nouveau graphe,

(ii). pour toute arête (r, s) ou (r, t), $r \in S$, ajouter une arête (r, u) à A. Dans le cas où (r, s) et (r, t) existent, deux arêtes reliant r à u sont créées.

(iii). Supprimer de A toutes les arêtes ayant s ou t comme extrémité.

(iv). Supprimer s et t de S.