Devoir surveillé n°1

Durée : 3h

Calculatrice non autorisée

⋆ ⋆ ⋆

Le sujet est constitué d'un exercice et d'un problème qui sont indépendants. On consacrera entre 30 et 45 minutes à l'exercice et le reste au problème.

Exercice

Pour chacun des problèmes suivants, déterminer, en justifiant, s'il est décidable puis s'il est semi-décidable.

1. `ARRÊT` :
   - **Instance** : une fonction et un argument, d'encodage $\langle f, x \rangle$.
   - **Question** : le calcul de $f(x)$ termine-t-il ?
2. $\texttt{ARRÊT}_{\texttt{fini}}$:
   - **Instance** : une fonction, d'encodage $\langle f \rangle$.
   - **Question** : le nombre d'arguments x pour lesquels $f(x)$ termine est-il fini ?
3. $\texttt{ÉQUIV}_{\exists}$:
   - **Instance** : deux fonctions, d'encodage $\langle f, g \rangle$.
   - **Question** : existe-t-il un argument x tel que $f(x)$ et $g(x)$ terminent et renvoient la même valeur ?

Problème : goulot maximal

On s'intéresse dans ce problème au goulot maximal, c'est-à-dire la recherche d'un chemin entre deux sommets qui maximise le poids de l'arête de poids minimal. Ce problème a des applications en réseau informatique (bande passante) ou pour la gestion du trafic routier.

Il est découpé en trois parties. La première s'intéresse à des généralités sur les arbres. La deuxième propose l'étude d'un algorithme de calcul d'arbre couvrant de poids minimal. Enfin, la troisième et dernière partie propose de résoudre le problème du goulot maximal sous différents angles.

Consignes aux candidates et candidats

On identifiera une même grandeur écrite dans deux polices de caractères différentes, en italique du point de vue mathématique (par exemple n) et en Computer Modern à chasse fixe du point de vue informatique (par exemple `n`).

Les candidates et candidats devront répondre aux questions de programmation en utilisant le langage OCaml et le langage C. S'ils écrivent une fonction non demandée par l'énoncé, ils devront préciser son rôle, ainsi que sa signature (son type). Les candidates et candidats sont également invités, lorsque c'est pertinent, à décrire le fonctionnement global de leurs programmes.

En OCaml, on autorisera toutes les fonctions des modules `Array` et `List`, ainsi que les fonctions de la bibliothèque standard (celles qui s'écrivent sans nom de module, comme `max`, `incr` ainsi que les opérateurs comme `@`). Sauf précision de l'énoncé, l'utilisation d'autres modules sera interdite.

En C, on supposera que les bibliothèques `stdlib.h`, `stdbool.h` et `assert.h` ont été chargées. On prendra garde à écrire du code qui n'engendre pas de fuite mémoire.

Définitions et notations

Dans l'ensemble du sujet, on considère des graphes non orientés. Un **graphe** est un couple $G = (S, A)$ tel que $A \subseteq \mathcal{P}_2(S)$. Un graphe **pondéré** est un triplet $G = (S, A, f)$ tel que (S, A) est un graphe et $f : A \to \mathbb{N}$ est une fonction dite de **pondération**.

S'il n'y a pas d'ambiguïté, une arête $\{s, t\}$ sera notée st.

La figure 1 représente un graphe pondéré G_0.

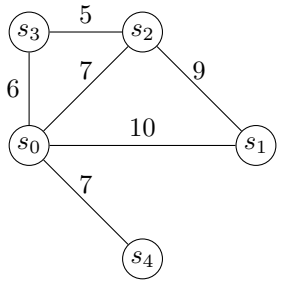

FIGURE 1 – Le graphe G_0

1 Généralités

Cette partie est à traiter en langage OCaml.

Question 1 Dessiner sans justifier un arbre couvrant de G_0 de poids minimal.

Question 2 Soit $G = (S, A)$ un graphe non orienté tel que $|S| = n$ et $|A| = k$. On pose $A = \{a_1, a_2, \ldots, a_k\}$ une numérotation des arêtes. Montrer par récurrence que pour $i \in [\![0, k]\!]$, le graphe $G_i = (S, \{a_1, \ldots, a_i\})$ possède au moins $n - i$ composantes connexes. En déduire que si G est connexe, alors $|A| \geqslant |S| - 1$.

Question 3 Avec les notations précédentes, montrer que si G_i possède au moins $n - i + 1$ composantes connexes, alors il contient un cycle. En déduire que si G est sans cycle, alors $|A| \leqslant |S| - 1$.

Question 4 Montrer l'équivalence entre les trois propriétés suivantes, pour $G = (S, A)$ un graphe non orienté :

(a) G est un arbre.
(b) G est connexe et $|A| = |S| - 1$.
(c) G est sans cycle et $|A| = |S| - 1$.

Pour les deux questions suivantes, on suppose que $G = (S, A, f)$ est un graphe pondéré tel que f est injective (toutes les arêtes ont un poids différent).

Question 5 Montrer que G possède un unique arbre couvrant de poids minimal.

Question 6 Soit $T^* = (S, B^*)$ l'unique arbre couvrant de poids minimal de G. On suppose que $a \in A$ est une arête dont le poids est maximal dans un cycle de G. Montrer que $a \notin B^*$.

Les trois questions qui suivent ont pour objectif d'implémenter un algorithme de tri efficace.

Question 7 Écrire une fonction `separation (lst: 'a list) : 'a list * 'a list` telle que si `lst` est une liste, alors `separation lst` renvoie un couple `(lst1, lst2)` tel que chaque élément de `lst` apparaît soit dans `lst1`, soit dans `lst2`, et les tailles de `lst1` et `lst2` diffèrent d'au plus 1.

Question 8 Écrire une fonction `fusion (lst1: 'a list) (lst2: 'a list) : 'a list` telle que si `lst1` et `lst2` sont deux listes triées par ordre croissant, alors `fusion lst1 lst2` renvoie une liste triée par ordre croissant contenant tous les éléments de `lst1` et `lst2`.

Question 9 En déduire une fonction `tri_fusion (lst: 'a list) : 'a list` qui prend en argument une liste et renvoie une liste triée par ordre croissant contenant les mêmes éléments. Quelle est la complexité temporelle de cette fonction ? (On ne demande pas de justifier.)

2 Algorithme de Kruskal inversé

Cette partie est à traiter en langage OCaml.

Question 10 Rappeler le principe de l'algorithme de Kruskal et sa complexité temporelle en fonction de $|S|$ et $|A|$ lorsqu'il est appliqué à un graphe pondéré connexe $G = (S, A, f)$.

L'algorithme de Kruskal inversé est une variante de l'algorithme de Kruskal qui permet de trouver un arbre couvrant de poids minimal dans un graphe pondéré connexe. En voici une description en pseudo-code.

Entrée : Graphe pondéré $G = (S, A, f)$ non orienté connexe
Début algorithme
  $B \leftarrow$ copie de A
  Trier B par ordre décroissant de poids
  Pour $a \in B$ par ordre décroissant de poids **Faire**
    Si $(S, B \setminus \{a\})$ est connexe **Alors**
      $B \leftarrow B \setminus \{a\}$
  Renvoyer (S, B)

Question 11 Appliquer l'algorithme de Kruskal inversé au graphe G_1 de la figure 2. On ne demande pas la description de l'exécution détaillée, mais simplement une représentation graphique de l'arbre obtenu.

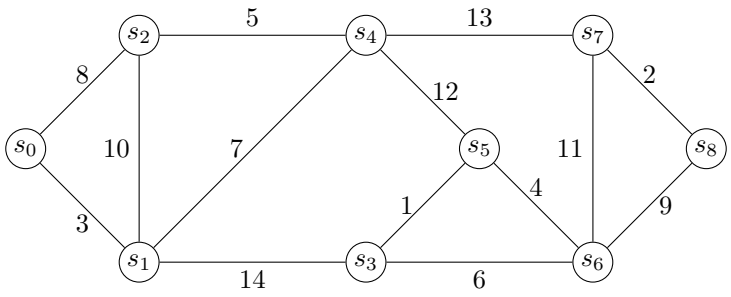

FIGURE 2 – Le graphe pondéré G_1.

On implémente un graphe pondéré par tableau de listes d'adjacences en OCaml, selon le type suivant :

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

Si $G = (S, A, f)$ est implémenté par un objet `g` de type `graphe`, alors :

- l'ensemble des sommets est $S = [\![0, n-1]\!]$, où $n = |S|$;
- `g` est un tableau de taille n tel que pour tout $s \in S$, `g.(s)` est une liste contenant des couples (t, p) tels que $st \in A$ et $f(st) = p$.

Les listes d'adjacence ne sont pas nécessairement supposée triées par ordre croissant des sommets ou des poids. Par exemple, le graphe G_0 de la figure 1 peut être créé par la commande :

```
let g0 = [|[(2, 7); (3, 6); (1, 10); (4, 7)]; [(2, 9); (0, 10)];
           [(0, 7); (1, 9); (3, 5)]; [(0, 6); (2, 5)]; [(0, 7)]|]
```

On rappelle que les opérations de comparaisons (`max`, `min`, `<`, `>`, ...) peuvent s'effectuer sur des uplets selon l'ordre lexicographique.

Question 12 Écrire une fonction `tri_aretes (g: graphe) : (int * int * int) list` telle que si `g` est la représentation d'un graphe pondéré $G = (S, A, f)$, alors `tri_aretes g` renvoie la liste des triplets $(f(st), s, t)$, triée par ordre **décroissant** des $f(st)$, tels que $st \in A$ et $s < t$. Cette liste ne devra pas contenir de doublons.

Question 13 Écrire une fonction `connectes (g: graphe) (s: int) (t: int) (pmax: int) : bool` telle que si `g` est la représentation d'un graphe pondéré $G = (S, A, f)$, $(s, t) \in S^2$ et `pmax` est un entier, alors `connectes g s t pmax` renvoie un booléen qui vaut `true` si et seulement s'il existe un chemin de s à t dans G ne passant que par des arêtes dont le poids est strictement inférieur à `pmax`. La fonction devra avoir une complexité linéaire en $|S| + |A|$ et on demande de justifier brièvement.

Question 14 En déduire une fonction `kruskal_inverse (g: graphe) : graphe` qui prend en argument un graphe $G = (S, A, f)$ et renvoie le graphe obtenu selon le principe de l'algorithme de Kruskal inversé. On supposera que le graphe G donné en argument est connexe et que la fonction de pondération f est injective.

Question 15 Montrer que si G est connexe, le graphe renvoyé par l'algorithme de Kruskal inversé appliqué à G est un arbre couvrant de G.

Question 16 Montrer que l'arbre couvrant renvoyé par l'algorithme de Kruskal inversé appliqué à un graphe pondéré connexe G est de poids minimal. On pourra commencer par traiter le cas où la fonction de pondération est injective.

Question 17 Déterminer la complexité temporelle de la fonction `kruskal_inverse`. Commenter.

3 Calcul du goulot maximal

Cette partie est à traiter en langage C.

Dans cette partie, on considère $G = (S, A, f)$ un graphe pondéré non orienté connexe.

Pour $\sigma = (s_0, s_1, \ldots, s_k)$ un chemin simple dans G, la **capacité** de σ, notée $c(\sigma)$ est le poids minimal de ses arêtes :

$$c(\sigma) = \min_{i=0}^{k-1} f(s_i s_{i+1})$$

Pour $s, t \in S^2$, le **goulot maximal** de s à t est un chemin de s à t de capacité maximale. S'il n'y a pas d'ambiguïté, on appellera également goulot maximal le poids de ce chemin.

Question 18 Déterminer, sans justifier, un goulot maximal et sa capacité de s_0 à s_8 dans le graphe G_1 de la figure 2.

3.1 Première approche

Question 19 Expliquer en français comment calculer efficacement un arbre couvrant de poids **maximal** de G. Justifier succinctement sa correction.

Question 20 Soit $(s, t) \in S^2$ et $T = (S, B)$ un arbre couvrant de poids **maximal** de G. Montrer qu'un

chemin de s à t dans T est un goulot maximal de s à t dans G.

On représente un graphe $G = (S, A, f)$ en C par le type suivant :

```
struct Graphe {
    int n;
    int* degres;
    int** voisins;
    int** poids;
};

typedef struct Graphe graphe;
```

tel que si `G` est un objet de type `graphe` représentant G, alors :

- `g.n` $= n = |S|$ et $S = [\![0, n-1]\!]$;
- `G.degres` est un tableau de taille n tel que pour $s \in S$, `G.degres[s]` $= \deg(s)$;
- `G.voisins` est un tableau de taille n tel que pour $s \in S$, `G.voisins[s]` est un tableau de taille $\deg(s)$ contenant les voisins de s ;
- `G.poids` est un tableau de taille n tel que pour $s \in S$, `G.poids[s]` est un tableau de taille $\deg(s)$ contenant les poids des arêtes entre s et ses voisins. On garantit que pour $0 \leqslant i < \deg(s)$, si $t =$ `G.voisins[s][i]`, alors $f(st) =$ `G.poids[s][i]`.

Question 21 Écrire une fonction `void liberer_graphe(graphe G)` qui libère l'espace mémoire occupé par un graphe.

Question 22 Écrire une fonction `int* predecesseurs(graphe T, int s)` qui prend en argument un graphe $T = (S, B)$ qui est un arbre et un sommet $s \in T$ et renvoie un tableau d'entiers `pred` de taille $n = |S|$ tel que `pred[s] = -1` et pour $t \neq s$, `pred[t]` est le parent de t dans l'arbre T enraciné en s.

Cette fonction devra avoir une complexité linéaire en n et on demande de le justifier.

On suppose disposer d'une fonction `graphe arbre_max(graphe)` qui prend en argument un graphe et renvoie un arbre couvrant de poids maximal de ce graphe.

Question 23 En déduire une fonction `int* goulot_max(graphe G, int s, int t, int* lc)` qui prend en argument un graphe et deux sommets s et t de ce graphe, ainsi qu'un pointeur vers un entier `lc` et renvoie un goulot maximal sous forme d'un tableau constitué des sommets de ce chemin. La fonction devra modifier l'entier pointé par `lc` pour y stocker la taille du chemin ainsi renvoyé.

3.2 Un algorithme efficace

Dans toute cette partie, on suppose que la fonction de pondération est injective. Pour $G = (S, A, f)$ un graphe non orienté pondéré et $X \subseteq S$, on appelle opération de **fusion de X dans G** l'opération qui consiste à remplacer G par le graphe $G' = (S', A', f')$ où :

- $S' = S \setminus X \cup \{x\}$, où x est un nouveau sommet ;
- $A' = (\mathcal{P}_2(S \setminus X) \cap A) \cup \{\{s, x\} \mid s \in S \setminus X, \exists t \in X, \{s, t\} \in A\}$;
- pour $a \in A'$:
  - si $a \in A$, $f'(a) = f(a)$;
  - sinon, $a = \{s, x\}$ et $f'(a) = \max\{f(s, t) \mid t \in X\}$.

Autrement dit, on fusionne les sommets de X en un seul, on laisse les arêtes qui relient X à un autre sommet, et on garde le poids maximal lorsqu'il y a plusieurs choix.

On considère l'algorithme décrit à la figure 3.

```
Entrée : Graphe pondéré G = (S, A, f) non orienté connexe, source s ∈ S et destination t ∈ S
Début algorithme
    Tant que |A| > 1 Faire
        M ← médiane de {f(a) | a ∈ A}.
        Supprimer de A les arêtes a de poids f(a) < M.
        Si il n'existe pas de chemin de s à t Alors
            Poser X_1, ..., X_k les composantes connexes de G.
            Rajouter les arêtes supprimées avant le test.
            Fusionner X_1, ..., X_k dans G.
    Renvoyer f(a) où a est l'unique arête de G.
```

FIGURE 3 – Algorithme BSP (*Bottleneck Shortest Path*).

On admet qu'un passage dans la boucle **Tant que** peut se réaliser en temps $\mathcal{O}(|A|)$, où A désigne ici l'ensemble des arêtes à l'entrée dans la boucle (qui est donc modifié d'un passage dans la boucle au suivant).

Question 24 Montrer que l'ensemble de l'algorithme a une complexité linéaire en $|A|$, où A est l'ensemble initial des arêtes.

Question 25 Montrer que l'algorithme renvoie la capacité d'un goulot maximal de s à t dans G.

Question 26 Expliquer comment calculer un goulot maximal de s à t dans G si on connaît sa capacité.

⋆ ⋆ ⋆