Composition d'Informatique n°1

Durée : 3 heures

L'utilisation de la calculatrice **n'est pas autorisée** pour cette épreuve.

⋆ ⋆ ⋆

Ce sujet est constitué d'un premier problème sur l'indécidabilité (à faire en 45 minutes environ) et d'un deuxième problème sur les graphes (à faire en 2h15 environ) qui sont indépendants.

Problème 1 : théorème de Rice

Dans l'ensemble de ce problème, on manipule des fonctions écrites dans un langage de programmation, par exemple OCaml. On distinguera la notation f, correspondant à la fonction elle-même, ou l'algorithme, et la notation $\langle f \rangle$, désignant le **code source** de la fonction f. Ainsi, si on considère le code :

```
let rec f lst = match lst with
    | []     -> 0
    | _ :: q -> 1 + f q
```

Alors on distingue l'objet `f`, de signature `'a list -> int`, et l'objet $\langle$`f`$\rangle$, de signature `string`, correspondant à la chaîne de caractères :

```
"let rec f lst = match lst with | [] -> 0 | _ :: q -> 1 + f q"
```

Par ailleurs, si $\langle f \rangle$ est le code source d'une fonction et x est une chaîne de caractère représentant un argument pour f, on note $\langle f, x \rangle$ une chaîne de caractère représentant de manière unique le couple $(\langle f \rangle, x)$. On suppose que $\langle f, x \rangle$ est facilement constructible à partir de $\langle f \rangle$ et x, on peut par exemple imaginer qu'il s'agit de la concaténation de $\langle f \rangle$ et x, séparés par un double point virgule `;;`.

Pour simplifier l'étude, on suppose que l'ensemble des fonctions manipulées sont de type `string -> bool`. On suppose également disposer d'une fonction `universel : string -> bool` telle que l'appel à la fonction `universel` $\langle$`f, x`$\rangle$ simule l'exécution de `f x`. On notera Σ^* l'ensemble des chaînes de caractères.

Si f est une fonction, on note $L(f)$, appelé **langage de** f, l'ensemble des arguments pour lesquels la fonction renvoie `true`, c'est-à-dire :

$$L(f) = \{x \in \Sigma^* \mid f(x) \text{ termine et renvoie } \texttt{true}\}$$

Question 1 Montrer que le problème `Appartient` suivant est indécidable et semi-décidable :

- **Instance** : un code source $\langle f \rangle \in \Sigma^*$ et un argument $x \in \Sigma^*$.
- **Question** : est-ce que $x \in L(f)$?

On attend une preuve directe, sans utiliser de réduction.

On appelle **propriété sémantique** tout sous-ensemble de $\mathcal{P}(\Sigma^*)$. Une propriété sémantique est une propriété qui permet de décrire le langage d'une fonction. Par exemple, « les arguments x pour lesquels $f(x)$ termine et renvoie `true` sont les chaînes de taille inférieure ou égale à 3 ».

Si $P \subseteq \mathcal{P}(\Sigma^*)$, on associe à P le problème de décision suivant, qu'on notera Π_P :

- **Instance** : un code source $\langle f \rangle \in \Sigma^*$.
- **Question** : est-ce que $L(f) \in P$?

Ainsi, avec la notation ensembliste des problèmes de décision, on a l'équivalence $\langle f \rangle \in \Pi_P \Leftrightarrow L(f) \in P$.

Question 2 Pour chacun des deux problèmes suivants, indiquer s'ils sont associés ou non à une propriété sémantique :

a) * **Instance** : un code source $\langle f \rangle \in \Sigma^*$.
   * **Question** : est-ce que le code source de f contient une boucle `for` ?

b) * **Instance** : un code source $\langle f \rangle \in \Sigma^*$.
   * **Question** : est-ce que $|L(f)| \leqslant 10$?

Une propriété sémantique P est dite **triviale** si $\Pi_P = \Sigma^*$ ou $\Pi_P = \emptyset$, c'est-à-dire si tout code source vérifie la propriété, ou si aucun code source ne vérifie la propriété.

Pour la suite, on cherche à montrer le théorème de Rice :

Théorème : Rice

Si P est une propriété sémantique non triviale, alors Π_P est indécidable.

Pour ce faire, considérons P une propriété sémantique non triviale. On suppose, pour commencer, que $\emptyset \notin P$.

Question 3 Justifier qu'il existe $\langle g \rangle \in \Sigma^*$ tel que $L(g) \in P$.

Soit alors $\langle f \rangle \in \Sigma^*$ et $x \in \Sigma^*$. On définit la fonction suivante :

```
let h y =
    universel <f, x> && universel <g, y>
```

Question 4 En considérant la fonction `h`, montrer que `Appartient` $\leqslant_m \Pi_P$, puis que Π_P est indécidable.

Question 5 En traitant le cas où $\emptyset \in P$, montrer le théorème de Rice.

Question 6 Pour chacun des deux problèmes de la question 2, déterminer s'il est décidable ou non. Justifier. On utilisera le théorème de Rice si c'est pertinent.

Problème 2 : voyageur de commerce

Les questions de programmation doivent être traitées en langage C. On supposera que les bibliothèques `stdlib.h` et `stdbool.h` ont été chargées. On apportera une attention particulière à la gestion de la mémoire et à sa libération.

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`).

Sans précision supplémentaire, lorsqu'une question demande la complexité d'une fonction, il s'agira de la complexité temporelle dans le pire des cas. La complexité sera exprimée sous la forme $\mathcal{O}(f(n, m))$ où n et m sont les tailles des arguments de la fonction, et f une expression la plus simple possible. Les calculs de complexité seront justifiés succinctement.

Présentation du problème

Dans ce sujet, on s'intéresse au problème du voyageur de commerce, un problème d'optimisation de chemin dans un graphe pondéré. La première partie aborde le problème sous un angle naïf. La deuxième partie améliore la complexité par un algorithme glouton. La troisième partie présente l'algorithme de Prim permettant de trouver un arbre couvrant minimal d'un graphe pondéré non orienté connexe. La quatrième et dernière partie

utilise l'arbre couvrant minimal pour approximer une solution au problème du voyageur de commerce sous certaines conditions.

Définition

Soit $G = (S, A)$ un graphe non orienté. Un **cycle hamiltonien** de G est un cycle passant une fois et une seule par chaque sommet. Un cycle hamiltonien dans un graphe d'ordre n sera noté simplement $(s_0, \ldots, s_{n-1})$ au lieu de $(s_0, \ldots, s_{n-1}, s_0)$.

On définit le `Problème du voyageur de commerce (PVC)` :

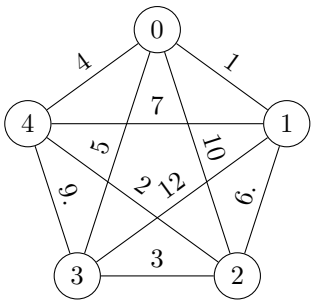
* **Instance** : $G = (S, A, f)$ un graphe pondéré non orienté.
* **Solution** : un cycle hamiltonien $c = (s_0, \ldots, s_{n-1})$.
* **Optimisation** : Minimiser le poids du cycle $f(c) = f(s_0, s_{n-1}) + \sum_{i=0}^{n-2} f(s_i, s_{i+1})$.

Si $G = (S, A)$ est un graphe, on supposera dans l'ensemble du problème que $S = \{0, 1, \ldots, |S| - 1\}$.

1 Approche naïve

Question 7 On suppose que G est un graphe complet d'ordre n. Déterminer le nombre de cycles hamiltoniens différents dans G. On supposera que deux cycles hamiltoniens sont différents s'ils ne sont pas constitués des mêmes arêtes.

Pour la suite de cette partie, on suppose qu'un graphe $G = (S, A, f)$ pondéré, non orienté et complet est représenté par sa matrice d'adjacence implémentée en C dans un tableau unidimensionnel d'entiers, les lignes de la matrice étant consécutives dans le tableau. Le graphe étant supposé complet, la valeur ∞ n'apparaitra pas dans la matrice. Par exemple, le graphe G_0 de la figure 1 pourra être représenté par le code suivant :



$$M_0 = \begin{pmatrix} 0 & 1 & 10 & 5 & 4 \\ 1 & 0 & 6 & 12 & 7 \\ 10 & 6 & 0 & 3 & 2 \\ 5 & 12 & 3 & 0 & 9 \\ 4 & 7 & 2 & 9 & 0 \end{pmatrix}$$

```c
int G0[25] = {0, 1, 10, 5, 4,
              1, 0, 6, 12, 7,
              10, 6, 0, 3, 2,
              5, 12, 3, 0, 9,
              4, 7, 2, 9, 0};
```

FIGURE 1 – Le graphe G_0, sa matrice d'adjacence et sa représentation en C

Question 8 Écrire une fonction `int f(int* G, int n, int s, int t)` qui prend en argument un tableau correspondant à un graphe $G = (S, A, f)$, un entier $n = |S|$ et deux entiers $(s, t) \in S^2$ et renvoie la valeur $f(s, t)$ correspondant au poids de l'arête $\{s, t\}$.

On choisit de représenter un cycle hamiltonien $c = (s_0, \ldots, s_{n-1})$ d'un graphe G d'ordre n par un tableau de taille n contenant les sommets du cycle.

Question 9 Écrire une fonction `int poids_cycle(int* G, int* c, int n)` qui prend en argument un graphe $G = (S, A, f)$, un cycle c de G et un entier $n = |S|$ et renvoie $f(c)$ tel que défini précédemment. Par exemple, si `c` est défini par `int c[5] = {0, 2, 4, 3, 1};`, alors `poids_cycle(G0, c, 5)` renverrait `34`.

On remarque que toute permutation de $[\![0, n-1]\!]$ forme un cycle hamiltonien de G. On propose de parcourir toutes les permutations par ordre lexicographique pour conserver celle de poids minimal. Par exemple, la permutation qui suit $(0, 2, 4, 3, 1)$ dans l'ordre lexicographique est $(0, 3, 1, 2, 4)$.

On suppose disposer d'une fonction `bool permut_suivante(int* p, int n)` qui prend en argument une permutation p de taille n et modifie p en place pour la transformer en la permutation suivante selon l'ordre lexicographique. Cette fonction renvoie `false` si p était la dernière permutation selon l'ordre lexicographique (c'est-à-dire la permutation $(n-1, n-2, \ldots, 1, 0)$) et `true` sinon.

Question 10 Écrire une fonction `int* PVC_naif(int* G, int n)` qui prend en argument un graphe $G = (S, A, f)$ et un entier $n = |S|$ et renvoie un tableau contenant un cycle hamiltonien de G de poids minimal.

Question 11 Déterminer la complexité temporelle de `PVC_naif` en fonction de $n = |S|$. On admettra que `permut_suivante` a une complexité amortie en $\mathcal{O}(1)$.

2 Heuristique du plus proche voisin

L'heuristique du plus proche voisin est un algorithme glouton permettant de trouver un cycle hamiltonien dans un graphe G pondéré complet en espérant obtenir un poids total faible. Pour ce faire, on utilise un tableau `c` correspondant au cycle en cours de construction, ainsi qu'un tableau `vus` de booléens, permettant de savoir quels sommets ont déjà été vus et rajoutés au cycle.

Question 12 Écrire une fonction `int plus_proche(int* G, bool* vus, int n, int s)` qui prend en argument un graphe $G = (S, A, f)$, un tableau `vus` de booléens, un entier $n = |S|$ et un sommet $s \in S$ et renvoie un sommet t vérifiant :

- `vus[t]` vaut `false` ;
- $f(s, t)$ est minimal parmi les sommets t non vus.

Par exemple, si `vus` est défini par le tableau `bool vus[5] = {true, false, true, true, false};`, alors l'appel à `plus_proche(G0, vus, 5, 2)` renverra 4 car les seuls sommets non vus sont 1 et 4, et $f(2, 1) = 6 > f(2, 4) = 2$.

L'heuristique du plus proche voisin consiste à partir d'un sommet puis, à chaque étape, de choisir comme voisin suivant le sommet le plus proche du dernier sommet choisi parmi les sommets non encore choisis.

Question 13 Déterminer le cycle renvoyé par l'heuristique du plus proche voisin appliquée au graphe G_0, en commençant par le sommet 0 comme sommet initial.

Question 14 Écrire une fonction `int* PVC_glouton(int* G, int n)` qui prend en argument un graphe $G = (S, A, f)$ et un entier $n = |S|$ et renvoie un pointeur vers un tableau contenant un cycle hamiltonien de G construit selon l'heuristique du plus proche voisin. On commencera par le sommet 0 comme sommet initial.

Question 15 Déterminer la complexité temporelle de `PVC_glouton` en fonction de $n = |S|$.

Question 16 Déterminer et représenter graphiquement un graphe pondéré complet d'ordre 4, dont tous les poids d'arêtes sont distincts, tel que l'heuristique du plus proche voisin ne renvoie jamais de cycle hamiltonien de poids minimal, quel que soit le sommet de départ.

3 Algorithme de Prim

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

Question 17 Rappeler le principe de l'algorithme de Kruskal et donner, sans justifier, sa complexité temporelle.

On s'intéresse à l'algorithme de Prim suivant, permettant de calculer un arbre couvrant minimal de G.

Entrée : Graphe $G = (S, A, f)$ pondéré non orienté connexe
Début algorithme
  Poser $B = \emptyset$.
  Poser $R = \{s_0\}$ un sommet choisi arbitrairement.
  Poser $\mathcal{F} = \{(s_0, t) \mid t \text{ voisin de } s_0\}$ une file de priorité selon les poids des arêtes.
  Tant que $R \neq S$ **Faire**
    Retirer de $\mathcal{F}$ l'arête (s, t) de poids minimal.
    Si $t \notin R$ **Alors**
      Ajouter t à R.
      Ajouter $\{s, t\}$ à B.
      Pour u voisin de t **Faire**
        Ajouter (t, u) à $\mathcal{F}$.
  Renvoyer (S, B)

Question 18 Déterminer le graphe renvoyé par l'algorithme de Prim appliqué au graphe G_1 représenté figure 2, en supposant $s_0 = 0$.

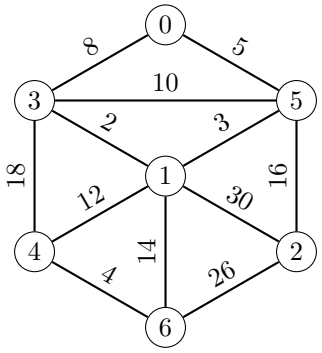


FIGURE 2 – Le graphe G_1

Question 19 Déterminer la complexité temporelle de l'algorithme de Prim en fonction de $|S|$ et $|A|$. On détaillera les structures de données utilisées et les choix d'implémentation nécessaires. En particulier, on indiquera la représentation choisie pour implémenter les graphes.

Question 20 Montrer que l'algorithme de Prim renvoie un arbre couvrant de G.

On cherche à montrer que l'arbre $T = (S, B)$ renvoyé par l'algorithme de Prim est bien un arbre couvrant minimal de G. Pour ce faire, on veut montrer que la propriété P :

« Il existe un arbre couvrant minimal $T^* = (S, B^*)$ de G tel que $B \subseteq B^*$ »

est un invariant de boucle. Pour simplifier les raisonnements, on note R_i et B_i les ensembles R et B à l'entrée dans la i-ème boucle (en commençant à zéro). Ainsi, $R_0 = \{s_0\}$ et $B_0 = \emptyset$. On note également $a_i = \{s_i, t_i\}$ l'arête considérée à la i-ème boucle.

Question 21 On suppose qu'il existe $T^* = (S, B^*)$ tel que $B_i \subseteq B^*$, et que $a_i \in B_{i+1} \setminus B^*$. Montrer qu'il existe, dans T^*, un chemin de s_i à t_i qui contient une arête reliant un sommet de R_i à un sommet de $S \setminus R_i$.

Question 22 En déduire que la propriété P est bien un invariant de boucle, puis que l'algorithme de Prim renvoie un arbre couvrant minimal.

4 Approximation de PVC dans le cas métrique

Sous certaines hypothèses, il est possible d'approximer efficacement une solution du problème du voyageur de commerce. Nous proposons ici d'étudier un algorithme qui, pour un graphe pondéré complet $G = (S, A, f)$, fournit un cycle hamiltonien de poids au plus le double du poids minimal, en supposant que la fonction de poids vérifie l'inégalité triangulaire, c'est-à-dire :

$$\forall s, t, u \in S^3, f(s, u) \leqslant f(s, t) + f(t, u)$$

Pour la suite, on note c^* un cycle hamiltonien de poids minimal de G et $T = (S, B)$ l'arbre renvoyé par l'algorithme de Prim. On étend de manière naturelle la fonction f aux sous-graphes de G et aux chemins dans G comme la somme des poids des arêtes qui les composent.

Question 23 Montrer que pour toute arête $a \in A$, $f(a) \geqslant 0$.

Question 24 Montrer que $f(T) \leqslant f(c^*)$.

Question 25 On pose c_T un parcours en profondeur préfixe de T. Montrer que $f(c_T) \leqslant 2f(c^*)$.

Comme on souhaite appliquer un tel algorithme à un graphe initialement complet, on suppose disposer d'une fonction `int* prim(int* G, int n)` qui prend en argument une matrice d'adjacence représentant un graphe complet, ainsi que l'ordre du graphe et renvoie un arbre couvrant minimal de ce graphe, donné lui-aussi sous forme de matrice d'adjacence, mais correspondant à un graphe non pondéré, donc ne contenant que des 0 et des 1.

Question 26 Écrire une fonction `int* PVC_approx(int* G, int n)` qui prend en argument un graphe $G = (S, A, f)$ et un entier $n = |S|$ et renvoie un tableau contenant un cycle hamiltonien de G dont le poids est au plus le double du cycle hamiltonien de poids minimal.

On peut montrer que l'existence d'un algorithme d'approximation polynomial sans l'hypothèse métrique entraînerait en fait $\mathsf{P} = \mathsf{NP}$ *(égalité qui est un problème ouvert à ce jour).*

⋆ ⋆ ⋆