COMPOSITION D'INFORMATIQUE n°1

Sujet 1 (Durée : 4 heures)

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

⋆ ⋆ ⋆

Ce sujet est constitué d'un exercice (à faire en 30 minutes environ) et d'un problème (à faire en 3h30 environ) qui sont indépendants.

Exercice

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

1. ARRÊT :
   - **Instance** : le code source d'une fonction f et un argument x.
   - **Question** : le calcul de $f(x)$ termine-t-il ?
2. $\text{ARRÊT}_{\texttt{fini}}$:
   - **Instance** : le code source d'une fonction f.
   - **Question** : le nombre d'arguments x pour lesquels $f(x)$ termine est-il fini ?
3. $\text{ARRÊT}_{\geqslant 10}$:
   - **Instance** : le code source d'une fonction f.
   - **Question** : le nombre d'arguments x pour lesquels $f(x)$ termine est-il supérieur ou égal à 10 ?

Problème : le voyageur de commerce

Les questions de programmation doivent être traitées en langage OCaml ou C selon ce qui est demandé par l'énoncé. 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` et `stdbool.h` ont été chargées.

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 étudie un algorithme de programmation dynamique permettant d'améliorer la complexité temporelle de l'algorithme naïf. La quatrième partie présente l'algorithme de Prim permettant de trouver un arbre couvrant minimal d'un graphe pondéré non orienté

connexe. La cinquième partie utilise l'arbre couvrant minimal pour approximer une solution au problème du voyageur de commerce sous certaines conditions. Enfin, la dernière partie démontre que sans hypothèses particulières, il n'est pas possible de trouver d'approximation en temps polynomial.

Les questions de programmation des parties 1 et 2 seront traitées en C, celles de la partie 3 seront traitées en OCaml.

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)` :

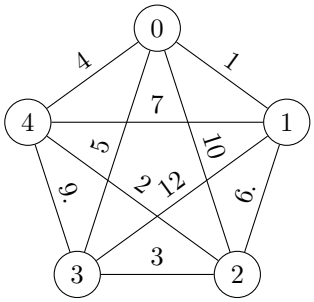
* **Instance** : $G = (S, A, f)$ un graphe pondéré non orienté.
* **Solution** : un cycle hamiltonien $c = (s_0, \ldots, s_{n-1})$.
* **Optimisation** : Minimiser le poids du cycle $f(c) = f(s_0, s_{n-1}) + \sum_{i=0}^{n-2} f(s_i, s_{i+1})$.

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 1 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 :



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

```
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 2 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 3 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 $(0, 3, 1, 2, 4)$.

Si $p = (p_0, \ldots, p_{n-1})$ est une permutation de $[\![0, n-1]\!]$, on définit les indices suivants :

- $j = \max\{i \in [\![0, n-2]\!] \mid p_i < p_{i+1}\}$, et $j = -1$ si cet ensemble est vide ;
- $k = \max\{i \in [\![j+1, n-1]\!] \mid p_j < p_i\}$ et $k = n$ si $j = -1$.

Question 4 En utilisant les indices j et k, décrire en français ou en pseudo-code un algorithme permettant de modifier p en place pour obtenir la permutation suivante dans l'ordre lexicographique et renvoyer « faux » si p était la dernière permutation (selon l'ordre lexicographique) et « vrai » sinon. On supposera pour la suite qu'une telle fonction est implémentée par une fonction `bool permut_suivante(int* p, int n)`.

Question 5 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 pointeur vers un tableau contenant un cycle hamiltonien de G de poids minimal.

Question 6 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 7 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 `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 8 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 9 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 10 Déterminer la complexité temporelle de `PVC_glouton` en fonction de $n = |S|$.

Question 11 Déterminer et représenter graphiquement un graphe pondéré complet d'ordre 5 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 Held-Karp

L'algorithme de Held-Karp est un algorithme de programmation dynamique permettant de résoudre le problème du voyageur de commerce. Son principe utilise des solutions aux sous-problèmes de la forme : « quel est le chemin de poids minimal d'un sommet s_0 à un sommet s_k passant par les sommets intermédiaires $\{s_1, \ldots, s_{k-1}\}$? ». On cherche alors à trouver le chemin de poids minimal du sommet 0 au sommet 0 passant par $\{1, 2, \ldots, n-1\}$, où $n = |S|$.

Si $s \in S$, et $X \subseteq S \setminus \{0, s\}$ est un ensemble de sommets, on note $P_{\min}(s, X)$ le poids minimal d'un chemin du sommet 0 au sommet s passant une et une seule fois par chaque sommet de X et aucun autre sommet. On note également $pred(s, X)$ le sommet qui précède s dans un tel chemin. Par convention, on posera $pred(0, \emptyset) = 0$.

Question 12 En envisageant chaque valeur possible pour $pred(s, X)$, donner une formule de récurrence vérifiée par $P_{\min}(s, X)$ en fonction des $P_{\min}(t, Y)$, pour $t \in X$ et Y bien choisi. Comment peut-on alors déterminer la valeur de $pred(s, X)$?

Comme précédemment, on choisira de représenter un ensemble de sommets $X \subseteq S$ par un tableau de booléens de taille $|S|$. Pour éviter de faire les mêmes calculs plusieurs fois et gagner en complexité, on choisit de mémoïser les résultats dans une table de hachage.

Question 13 Expliquer brièvement pourquoi il ne faut pas utiliser un objet mutable (comme un tableau) comme clé dans une table de hachage.

Pour pouvoir stocker les résultats déjà calculés dans la table de hachage, il est nécessaire de transformer les valeurs en des objets non mutables, comme par exemple des entiers ou des tuples d'objets non mutables.

Question 14 Écrire une fonction `encodage (s: int) (tab_X: bool array)` qui prend en argument un entier s et un tableau de booléens représentant un sous-ensemble $X \subseteq S$ et renvoie un objet représentant de manière unique s et X et pouvant servir de clé dans une table de hachage. On précisera le type choisi pour la valeur de retour.
On pourra assimiler le tableau de booléens à la décomposition binaire d'un entier compris entre 0 *et* $2^n - 1$.

On choisit en OCaml de représenter un graphe par matrice d'adjacence. Ainsi, le graphe G_0 sera représenté par :

```
let g0 = [| [|0; 1; 3; 5; 4|];
            [|1; 0; 6; 12; 7|];
            [|3; 6; 0; 2; 10|];
            [|5; 12; 2; 0; 9|];
            [|4; 7; 10; 9; 0|] |];;
```

On rappelle que les tables de hachage peuvent être manipulées en OCaml avec les fonctions suivantes :

- `Hashtbl.create : int -> ('a, 'b) Hashtbl.t` prend en argument un entier m et crée une table vide occupant un espace mémoire de taille proportionnelle à m (on pourra choisir $m = 1$ en pratique) ;
- `Hashtbl.add : ('a, 'b) Hashtbl.t -> 'a -> 'b -> unit` prend en argument une table, une clé et une valeur et rajoute une association ;
- `Hashtbl.mem : ('a, 'b) Hashtbl.t -> 'a -> bool` teste si une table contient une clé donnée ;
- `Hashtbl.find : ('a, 'b) Hashtbl.t -> 'a -> 'b` prend en argument une table et une clé et renvoie la valeur associée.

On suppose créées en variables globales :

- une matrice d'entiers `g` correspondant à un graphe $G = (S, A, f)$ d'ordre n ;
- une table de hachage par la commande `let tabh = Hashtbl.create 1;;`.

Question 15 Écrire une fonction `chemin_min (s: int) (tab_X: int array) : int * int` qui prend

en argument un sommet $s \in S$ et un tableau de booléens de taille n décrivant un sous-ensemble $X \subseteq S$ et renvoie le couple $(P_{\min}(s, X), pred(s, X))$.

Question 16 En déduire une fonction `pvc_dynamique () : int array` qui renvoie un tableau correspondant à un cycle hamiltonien de poids minimal du graphe G, calculé selon l'algorithme de Held-Karp.

Question 17 Déterminer les complexités temporelles et spatiales de `pvc_dynamique` en fonction de $n = |S|$.

4 Algorithme de Prim

Dans cette partie, on considère $G = (S, A, f)$ un graphe pondéré, non orienté et connexe (mais pas nécessairement complet). 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.
  Tant que $R \neq S$ **Faire**
    Poser $\{s, t\} \in A$ l'arête de poids minimal telle que $s \in R$ et $t \notin R$.
    Ajouter t à R.
    Ajouter $\{s, t\}$ à B.
  Renvoyer (S, B)

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

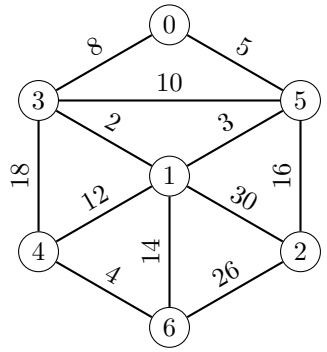


FIGURE 2 – Le graphe G_1

Question 19 Déterminer la complexité temporelle de l'algorithme de Prim en fonction de $|S|$ et $|A|$. On détaillera le choix des structures de données utilisées et les détails d'implémentation nécessaires.

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, supposons qu'il ne le soit pas, et soit $T^* = (S, B^*)$ un arbre couvrant minimal. Notons $(a_1, a_2, \ldots, a_{n-1})$ les arêtes ajoutées à B au cours de l'algorithme de Prim, dans cet ordre. Par hypothèse, $T \neq T^*$, donc il existe $i \in [\![1, n-1]\!]$ minimal tel que $a_i \notin B^*$. Considérons T^* comme l'arbre couvrant minimal qui maximise cette valeur de i. On pose R l'ensemble des sommets juste avant l'ajout de a_i à B au cours de l'algorithme.

Question 21 Montrer que dans un chemin de s à t dans T^*, il existe une arête $a^* \in B^*$, reliant un sommet de R et un sommet de $S \setminus R$.

Question 22 Montrer que $f(a_i) \leqslant f(a^*)$ et que cela entraîne une contradiction, puis conclure.

5 Approximation de PVC dans le cas métrique

Sous certaines hypothèses, il est possible d'approximer 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 En considérant un parcours en profondeur préfixe de T, en déduire l'existence d'un cycle hamiltonien c_T, calculable en temps polynomial en la taille de G, tel que $f(c_T) \leqslant 2f(c^*)$.

6 Approximer PVC est difficile dans le cas général

On montre ici que l'hypothèse d'inégalité triangulaire est indispensable dans l'approximation de la partie précédente. Pour ce faire, on propose de faire une réduction depuis un problème connu comme difficile, dont on ne connait pas d'algorithme de complexité polynomiale, à savoir une variante plus simple : le problème de décision `Cycle hamiltonien` :

* **Instance** : un graphe $G = (S, A)$ non orienté.
* **Question** : existe-t-il un cycle hamiltonien dans G ?

On suppose dans cette partie qu'il existe un algorithme de complexité polynomiale qui, en prenant un graphe $G = (S, A, f)$ pondéré, non orienté et complet, renvoie un cycle hamiltonien c de G tel que si c^* est un cycle hamiltonien de poids minimal, alors $f(c) \leqslant \alpha f(c^*)$, avec $\alpha \geqslant 1$ est une constante fixée.

Pour $G = (S, A)$ un graphe non orienté, on considère le graphe pondéré non orienté complet $K_G = (S, S^2, f)$ défini par :

- pour $a \in A$, $f(a) = 1$;
- pour $a \notin A$, $f(a) = \alpha|S|$.

Question 26 Montrer que $G = (S, A)$ possède un cycle hamiltonien si et seulement si K_G possède un cycle hamiltonien de poids inférieur ou égal à $\alpha|S|$.

Question 27 En déduire qu'on peut résoudre `Cycle hamiltonien` en temps polynomial.

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

⋆ ⋆ ⋆