Informatique - MPI

Exercice A1

Un *arbre* est un graphe connexe acyclique $G = (V, E)$ où V désigne l'ensemble des nœuds de l'arbre et E l'ensemble de ses *branches* non orientées. Une *feuille* est un nœud qui n'admet qu'un seul nœud adjacent ; son degré est 1. Ces arbres sont *non enracinés* dans le sens où aucun nœud ne joue de rôle particulier, à savoir celui de la racine de l'arbre.

Étant donné un entier naturel $n \geqslant 2$, le *théorème de Cayley* affirme que le nombre d'arbres étiquetés par les entiers de $\{0, \dots, n-1\}$ est n^{n-2}. L'objet de cet exercice est de justifier ce résultat.

Soit $G = (V, E)$ un arbre à n sommets avec $V = \{0, \dots, n-1\}$. Le *codage de Prüfer* définit une séquence P de $n-2$ entiers de V par l'algorithme suivant.

Algorithme 1 : codage de Prüfer

Poser $P \leftarrow \varnothing$.
pour *il reste plus de deux sommets dans* G **faire**
    identifier le sommet i de plus petit numéro dans G ayant un degré 1
    ajouter à la fin de P le numéro du seul sommet adjacent à i et supprimer i de G

L'algorithme suivant *décode* une séquence P de $n-2$ entiers pour construire un arbre $G = (V, E)$.

Algorithme 2 : décodage

Définir $I = \{0, \dots, n-1\}$ et créer le graphe $G = (\{0, \dots, n-1\}, \varnothing)$
pour $P \neq \varnothing$ **faire**
    identifier le plus petit entier $i \in I$ non présent dans P et le supprimer de I
    dans G, relier i au premier entier de P et supprimer ce premier entier de P
relier par une arête les deux sommets portant les deux derniers numéros encore présents dans I

Question 1. Écrire une fonction `prufer_code : tree -> list` qui renvoie une liste associée à un codage de Prüfer. Pour écrire cette fonction, vous justifierez le choix des structures de données adoptées et vous évaluerez la complexité au pire de votre solution en fonction de n.

Question 2. Écrire une fonction `prufer_decode : list -> tree` qui réalise le décodage. Là encore, on attend une justification des choix de structures de données et une estimation de la complexité de la solution.

Question 3. Justifier le théorème de Cayley.

Exercice A2

Soit $G = (V, E)$ un graphe non orienté complet d'ordre n. Chaque arête $\{i, j\}$ de G est valuée par un entier naturel $c_{i,j}$ appelé coût de l'arête. On suppose que tous les coûts satisfont l'*inégalité triangulaire* : pour tout ensemble de trois sommets $\{i, j, k\}$:

$$c_{i,j} \leqslant c_{i,k} + c_{k,j}$$

On rappelle qu'un *cycle hamiltonien* est un cycle qui ne passe qu'une seule fois par chaque sommet d'un graphe. Un *cycle eulérien* ne passe qu'une seule fois par chaque arête du graphe. Le *problème du voyageur de commerce euclidien* consiste à déterminer, dans une clique valuée dont les coûts respectent les inégalités triangulaires, un cycle hamiltonien de longueur totale minimum.

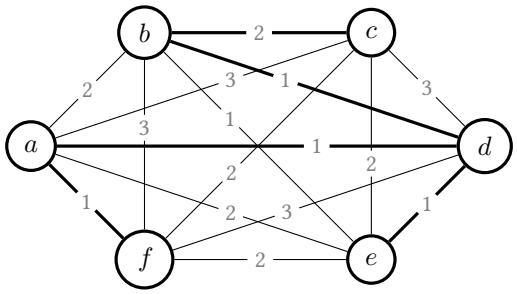
L'objet de cet exercice est de calculer une *solution approchée* à ce problème à l'aide d'un arbre couvrant de poids minimum. Soit H l'ensemble des arêtes d'un arbre couvrant de poids minimum et :

$$c(H) = \sum_{\{i,j\} \in H} c_{i,j}$$

Si $r \in H$, on note H_r l'arbre enraciné en r construit sur H. On considère l'algorithme ci-dessous. La notation H_x désigne le sous-arbre de H_r enraciné en x.

Algorithme 3 : fonction Euler

```
1 fonction Euler(x : sommet) : liste de sommets
2     L : liste
3     L ← (x)
4     pour sommet enfant s de x dans H_x faire
5         L ← L.Euler(s).(x)
```



Question 1. Quelle est la liste renvoyée par cet algorithme pour l'arbre H constitué des arêtes épaisses du graphe représenté ci-dessus, a étant pris comme racine de l'arbre engendré par H.

Question 2. Quelle la complexité de $Euler$?

Question 3. On note $L = (s_0, \dots, s_q)$ la liste obtenue à l'issue de l'exécution de l'algorithme $Euler(x)$.

□ **3.1.** Montrer que $q = 2(n-1)$.

□ **3.2.** Pour tout entier $k \in [0, \dots, q-1]$, montrer que $\{s_k, s_{k+1}\}$ est une arête de H.

□ **3.3.** Montrer que chaque arête de H apparaît deux fois dans L.

□ **3.4.** Montrer que chaque sommet de H apparaît au moins une fois dans L.

Question 4. On extrait de L la plus longue sous-liste $L' = \{s_{i_0}, \dots, s_{i_p}\}$ définie par $i_0 = 0$ et, pour tout entier $k \in [1, \dots, p]$, i_k est le plus petit indice tel que s_{i_k} n'est pas un sommet de $\{s_{i_0}, \dots, s_{i_{k-1}}\}$.

□ **4.1.** Déterminer L' pour le graphe représenté ci-dessus.

□ **4.2.** Montrer que L' est un cycle hamiltonien de G et que $c(L') \leqslant 2c(H)$.

□ **4.3.** On note c^* la longueur minimale d'un cycle hamiltonien de G. Démontrer que $c(L') \leqslant 2c^*$.

Exercice B2

Les programmes présentés dans cet exercice sont rédigés en C et reprennent les syntaxes de la librairie `<pthread.h>` pour les threads mais redéfinissent les structures de sémaphores et de mutex. Les réponses seront à écrire en C.
Soit une fonction f pour laquelle il existe une valeur i telle que $f(i) = 0$. Pour trouver une telle valeur i, l'espace de recherche est divisé en deux parties : les nombres positifs et les nombres négatifs. On propose le programme suivant.

```
1  bool found;
2  int res;
3
4  void p() {
5    int i = 0;
6    while (!found) {
7      i = i + 1;
8      found = (f(i) == 0);
9    }
10   res = i;
11 }
12
13 void q() {
14   int j = 1;
15   while (!found) {
16     j = j - 1;
17     found = (f(j) == 0);
18   }
19   res = j;
20 }
21
22 int cherche_zero() {
23   pthrea_t id[2];
24
25   pthread_create(&id[0], NULL, p, NULL);
26   pthread_create(&id[1], NULL, q, NULL);
27
28   pthread_join(id[0], NULL);
29   pthread_join(id[1], NULL);
30   return res;
31 }
```

Question 1. Pourquoi la variable `j` est-elle initialisée à `1` ?

Question 2. Montrer que le programme précédent ne termine pas pour toute exécution.

Question 3. En proposer une modification pour que la propriété de terminaison soit vérifée.

Question 4. Montrer que le programme précédent n'est pas partiellement correct.

Question 5. En proposer une modification pour qu'il soit partiellement correct.

Exercice B1

Le programme présenté dans cet exercice est rédigé en C et reprend les syntaxes de la librairie **<pthread.h>** pour les threads mais redéfinit les structures de sémaphores et de mutex. Les réponses seront à écrire en C.
Deux threads A et B doivent communiquer au travers d'un seul tableau **T** partagé de taille suffsante, appelé *tampon*. Les deux fonctions **depot** et **recuperer** permettent respectivement d'écrire un message dans le tampon et de lire le message du tampon.

```
1  char T[256];
2  void depot(char buf[]);
3  void recuperer(char buf[]);
```

A et B doivent communiquer à tour de rôle, en commençant par $A \to B$ (A envoie un message à B). On suppose donnés un type **semaphore_t** et les trois fonctions suivantes.

- **semaphore_t *init(int n);** crée un sémaphore dont la valeur initiale du compteur est **n**.
- **void P(semaphore_t *s);** décrémente la valeur du compteur du sémaphore. Si cette valeur est nulle, alors l'instruction bloque jusqu'à ce que la valeur devienne non nulle.
- **void V(semaphore_t *s);** incrémente la valeur du compteur du sémaphore.

Deux sémaphores sA et sB ont été initialisés avant le lancement des deux threads. Le code du thread A est le suivant.

```
1  {
2    char message[256];
3    char reponse[256];
4    while (true) {
5    /* Génère le message à envoyer */
6      P(sA);
7      depot(message);
8      V(sB);
9      P(sA);
10     recuperer(reponse);
11     V(sB);
12   }
13 }
```

Question 1. Quelles doivent être les valeurs initiales de **sA** et **sB** ?

Question 2. Écrire le code du thread B.

Question 3. Un troisième thread C veut communiquer avec B en utilisant le même tampon **T**. Les threads A et C se comportent de la même manière. B peut recevoir un message de A ou de C, la réponse doit être récupérée par le thread expéditeur du message. Identifier les problèmes éventuels si on ne change pas les codes de A et B et si C utilise le même code que A.

Question 4. On suppose donnés un type **mutex_t** représentant un mutex et les deux fonctions suivantes.

- **void lock(mutex_t *m);** permet de verrouiller un mutex.
- **void unlock(mutex_t *m);** permet de déverrouiller un mutex.

On suppose qu'un mutex **m** a été initialisé avant le lancement des threads. Proposer une modification du code de A qui permet de résoudre le problème.