Devoir Non Surveillé 1

À rendre le 4 novembre 2024

INFORMATIQUE MPI/MPI*

Vue d'ensemble du sujet

Ce sujet est composé de 3 parties indépendantes, utilisant les langages de programmation C et OCaml.

Les différentes parties sont indépendantes et peuvent être traités dans n'importe quel ordre. Il n'est pas nécessaire d'avoir répondu aux questions d'une partie pour répondre aux questions d'une autre partie.

- La partie I se concentre sur l'étude des couplages de cardinal maximum, leur lien avec les couvertures par sommet, puis propose des jeux à 2 joueurs sur ce thème.
- La partie II s'intéresse aux tests de primalité probabilistes, et à leur implémentation en C.
- La partie III s'intéresse à l'analyse et l'implémentation d'un algorithme de Tarjan pour trouver des plus petits ancêtres communs dans un arbre.

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat. En langage C, il est inutile de rappeler que les entêtes `<assert.h>`, `<stdbool.h>`, etc. doivent être inclus.

Quand l'énoncé demande de coder une fonction, sauf indication explicite de l'énoncé, il n'est pas nécessaire de justifier que celle-ci est correcte ou de tester que des préconditions sont satisfaites.

Le barème tient compte de la clarté des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle. Lorsqu'une réponse en pseudo-code est permise, seule la logique de programmation est évaluée, même dans le cas où un code en C a été fourni en guise de réponse.

Partie I. Couplages dans les graphes bipartis

On s'intéresse au problème de la *couverture minimale par sommets* (ou problème du transversal minimum) : étant donné un graphe $G = (S, A)$, déterminer le nombre de sommets minimal d'un ensemble $C \subset S$ tel que $\forall \{x, y\} \in A, x \in C \vee y \in C$. Un tel ensemble est appelé une couverture par sommets du graphe G.

☐ 1 – Démontrer que le nombre de sommets d'une couverture minimale par sommets est toujours supérieur ou égal au nombre d'arêtes d'un couplage de cardinal maximum.

Dans la suite de cette partie, on s'intéresse à un graphe biparti non orienté $G = (S, A)$, avec $S = X \cup Y$, $X \cap Y = \emptyset$ et $A \subset \{ \{x, y\} |\ x \in X, y \in Y\}$.

Soit M un couplage de cardinal maximum pour G. On note L l'ensemble des sommets accessibles par un chemin alternant [1] depuis un sommet libre de X.

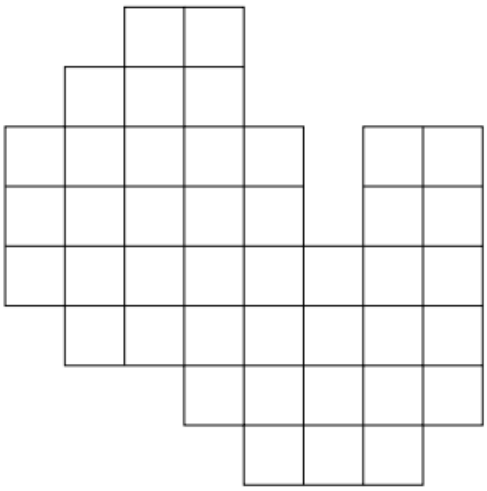
☐ 2 – Montrer que $C = (X \setminus L) \cup (Y \cap L)$ constitue une couverture par sommets de G.

☐ 3 – Justifier que L ne contient aucun sommet libre de Y.

☐ 4 – Montrer qu'il n'existe pas d'arête de M reliant un sommet de $X \setminus L$ et un sommet de $Y \cap L$.

☐ 5 – Démontrer le théorème de König (1931) :
Pour tout graphe biparti, le cardinal maximum d'un couplage est égal au cardinal minimum d'une couverture par sommets.

☐ 6 – Peut-on paver la figure suivante à l'aide de dominos ? Donner un pavage ou une preuve concise qu'aucun tel pavage ne peut exister.



☐ 7 – Démontrer le théorème de Hall (1935) :
G possède un couplage de cardinal $|X|$ si et seulement si pour tout $P \subset X$, on a $|V(P)| \geq |P|$
où $V(P) = \{y \in Y \mid \exists x \in P, \{x, y\} \in A\}$ est le voisinage de P.

☐ 8 – On considère le jeu à 2 joueurs suivant sur un graphe (pas nécessairement biparti) $G = (S, A)$. Les joueurs 1 et 2 choisissent chacun leur tour une arête pour former un chemin élémentaire avec les arêtes déjà sélectionnées (on rappelle qu'un chemin élémentaire ne peut pas passer plusieurs fois par le même sommet). Le premier joueur qui ne peut pas choisir d'arête a perdu. Montrer que si G possède un couplage parfait, le joueur 1 possède une stratégie gagnante.

☐ 9 – On considère le jeu à 2 joueurs suivant sur un graphe biparti. On commence avec un couplage vide $M := \emptyset$. Chacun leur tour, les joueurs choisissent un chemin augmentant I pour le couplage M, et on met à jour $M := M \oplus I$, où $\oplus$ est la différence symétrique. Si un joueur ne peut plus jouer, il perd et l'autre joueur gagne. Déterminer si un joueur possède une stratégie gagnante, et si oui lequel.

1. On appelle *chemin* alternant un chemin qui alterne les arêtes dans M et les arêtes dans $A \setminus M$

Partie II. Test de primalité (C)

Dans cette partie, on utilisera le langage C en supposant qu'il n'y a pas de dépassement de capacité sur les entiers, et que la fonction `rand()` renvoie un nombre aléatoire entre 0 et M, où M est suffisamment grand pour que `rand() % k` renvoie un nombre uniformément aléatoire entre 0 et $k-1$.

Test de primalité de Fermat

Si un nombre n est premier, alors tout nombre a premier avec n vérifie $a^{n-1} = 1 \mod n$. Le test de primalité de Fermat est un algorithme probabiliste basé sur cette observation.

☐ 10 – Écrire une fonction `int mod_exp(int a, int b, int n)` qui calcule $a^b \mod n$ en complexité en temps $O(\log(b))$, en supposant que l'addtion et la multiplication d'entiers s'effectue en temps constant.

☐ 11 – Écrire une fonction `bool fermat_witness(int n, int a)` qui renvoie `true` si a est un témoin de primalité de Fermat de n (i.e., $a^{n-1} = 1 \mod n$) et `false` sinon.

☐ 12 – Écrire une fonction `bool fermat_primality_test(int n, int k)` qui tire aléatoirement k nombres entre 2 et $n-1$ inclus et qui renvoie `true` si tous ces nombres sont des témoins de primalité de Fermat de n, et `false` si au moins l'un d'entre eux ne l'est pas.

Cet algorithme se révèle très efficace, car la moitié des nombres a entre 2 et $n-1$ sont des témoins de non-primalité si n n'est pas premier, sauf pour une famille de nombres particuliers : les nombres de Carmichael. Le test suivant permet d'éviter ces faux nombres premiers.

Test de primalité de Miller-Rabin

Soit p un nombre premier impair. Notons que 1 et -1 sont les seules racines carrées de 1 modulo p. Soit s non nul et d impair deux entiers vérifiant $p-1 = 2^s d$. Alors pour tout entier a qui n'est pas un multiple de p, on a : $a^{p-1} = 1 \mod p$. Donc

$$a^d = 1 \mod p \quad \vee \quad \exists r \in [\![0, s-1]\!], \quad a^{2^r d} = -1 \mod p.$$

Donc (par contraposée) pour tout entiers s non nul, d impair, $n = 2^s d + 1$ et $1 < a < n$,

$$\left(a^d \neq 1 \mod n \wedge \forall r \in [\![0, s-1]\!], a^{2^r d} \neq -1 \mod n\right) \to n \text{ est composite.}$$

Dans ce cas, on appelle a un témoin de non-primalité de Miller pour n.

☐ 13 – Écrire une fonction `void decompose(int n, int *s, int *d)` qui à partir d'un nombre n donné, calcule les valeurs de s et de d et change les valeurs pointées par `s` et `d`.

☐ 14 – Écrire une fonction `bool miller_witness(int n, int a)` qui étant donné n et a renvoie `true` si a est un témoin de non-primalité de n, et `false` sinon.

Si n est un nombre impair composé, au moins 3/4 des entiers entre 2 et $n-1$ inclus sont des témoins de non-primalité de Miller pour n. On peut donc en déduire un algorithme de type Monte-Carlo dont la probabilité de correction est supérieure à 0.95.

☐ 15 – Écrire une fonction `bool miller_rabin95(int n)` qui renvoie `true` si n est premier, et qui renvoie `false` avec une probabilité supérieure à 0.95 si n est composé.

Ce test de primalité à été proposée par Michael Rabin comme variante de l'algorithme proposé par Gary L. Miller dont la preuve de correction repose sur l'hypothèse de Riemann généralisée. Il s'agit dans la version originale de tester les entiers compris entre 2 et $O(\log(n)^2)$.

Partie III. Tarjan's off-line lowest common ancestors algorithm

Cet algorithme, du à Robert Tarjan[2], sert à calculer dans un arbre $\mathcal{T}$, pour une ensemble de paires de nœuds $P = (\{u, v\})$ les plus petit ancêtre communs de u et v. Il est donné par le pseudo-code suivant :

```
function TarjanOLCA(u) is
    MakeSet(u)
    u.ancestor := u
    for each v in u.children do
        TarjanOLCA(v)
        Union(u, v)
        Find(u).ancestor := u
    u.color := black
    for each v such that {u, v} in P do
        if v.color == black then
            print "Tarjan's Lowest Common Ancestor of " + u +
                  " and " + v + " is " + Find(v).ancestor + "."
```

On cherche à le comprendre, et à l'implémenter en OCaml, avec le type suivant pour les arbres (on se restreint à des arbres binaires localement complets non vides) :

```
type 'a tree = Leaf of 'a | Node of ('a tree * 'a * 'a tree)
```

☐ 16 – Proposer un type adapté pour représenter l'ensemble P et justifier votre choix.

☐ 17 – Quel type de parcours d'arbre peut-on reconnaître dans cet algorithme ? Justifier.

☐ 18 – Pour représenter les champs `ancestor` et `color` présentés dans le pseudo-code, nous souhaiterions utiliser des tableaux. Or, nous avons besoin pour cela que les nœuds possèdent une étiquette unique entre 0 et $n-1$, où n est la taille de l'arbre.
Écrire une fonction `label (tree : 'a tree) : (int * 'a) tree` qui prend en entrée un arbre $\mathcal{T}$ et qui renvoie un arbre $\mathcal{T}'$ qui contient les mêmes nœuds que $\mathcal{T}$ mais dont l'étiquette contient en plus un unique identifiant entier entre 0 et $n-1$, où $n-1$ est la taille de l'arbre.

☐ 19 – Étudier la complexité de cet algorithme.

Étudions sa correction

☐ 20 – Démontrer que chaque paire (u, v) apparaît une et une seule fois dans l'affichage (`print`) proposé par l'algorithme décrit en pseudo-code.

☐ 21 – Démontrer que tant que le plus petit ancêtre commun à u et v n'est pas noir, il est l'ancêtre (`ancestor`) du représentant (`Find`) du premier des deux nœuds rencontrés parmi u et v.

☐ 22 – Proposer une implémentation de cet algorithme de Tarjan.

Fin de l'épreuve

2. Un autre algorithme porte le nom de Tarjan, et sert à calculer les composantes fortement connexes dans un graphe orienté.