Centrale Informatique optionnelle MP 2021
Corrigé

Ce corrigé est proposé par Titouan Leclercq (ENS Lyon) ; il a été relu par Vincent Puyhaubert (professeur en CPGE) et Benjamin Monmege (enseignant-chercheur à l'université).

Ce sujet porte sur la génération aléatoire de pavages d'un échiquier par des dominos. L'énoncé utilise pour cela une bijection entre ces pavages et les arbres couvrants d'un certain type de graphe. Les parties dépendent les unes des autres et le sujet est de difficulté croissante.

- Dans la partie I, on introduit les fonctions de base qui servent à manipuler et générer des graphes non orientés en forme de quadrillage. Elle comporte quelques questions de réindexation sur lesquelles il convient d'être particulièrement vigilant.
- La partie suivante introduit la notion d'arbre couvrant d'un graphe non orienté. Elle demande d'établir quelques résultats classiques qui caractérisent ces sous-graphes connexes et acycliques, et s'achève par la rédaction d'une fonction qui permet de vérifier si un sous-graphe donné est un arbre.

  On notera à cet effet plusieurs questions portant sur la structure de données *union-find* qui permet de manipuler efficacement des partitions d'ensembles.
- La partie III étudie et implémente l'algorithme de Wilson, qui permet de générer un arbre couvrant aléatoire d'un graphe quelconque. La structure utilisée pour représenter les arbres couvrants sera conservée dans tout le reste du sujet.
- Dans la partie IV, on introduit la notion de pavage d'un échiquier par des dominos rectangulaires. On s'intéresse uniquement au cas où l'échiquier est un rectangle de dimensions impaires, privé de son coin inférieur gauche. L'énoncé admet l'existence d'une bijection entre les pavages de ce dernier et les arbres couvrants d'un graphe dont la forme est celle étudiée en première partie. À l'aide d'une réprésentation simple d'un pavage en Caml, on construit une première fonction bijective associant un arbre couvrant à un pavage.
- La dernière partie, longue et difficile, s'intéresse à la génération aléatoire d'un pavage. On s'intéresse pour cela au problème réciproque de celui de la partie IV : construire un pavage à partir d'un arbre couvrant, ce qui nécessite l'introduction de la notion d'arbre dual des graphes de quadrillage. Cette dernière partie s'achève sur un code de longueur assez considérable, qui combine les fonctions des cinq parties pour aboutir à la génération aléatoire attendue.

Ce sujet est un très bon sujet de révision sur les graphes et sur les structures de données, malgré quelques imprécisions, notamment une définition du dual d'un dual peu claire à la question 28. Sur les 38 questions composant le sujet, 20 demandent d'écrire du code, de difficulté variable. Il est sans doute un peu long pour une épreuve de quatre heures et nécessite de rester concentré jusqu'à la fin.

Indications

1 Utiliser une relation entre le nombre d'arêtes et les degrés des sommets d'un graphe.

4 Différencier les cas d'une arête verticale et d'une arête horizontale, puis trouver comment situer l'arête dans le graphe.

7 Pour montrer que deux parties sont égales ou disjointes, montrer une double inclusion s'il y a un élément commun à deux parties.

8 Montrer l'existence d'un plus court chemin de s à t en considérant l'ensemble des longueurs des chemins de s à t. Montrer ensuite qu'un chemin qui passe deux fois par le même sommet peut être réduit.

9 Le caractère acyclique d'un graphe se traduit par le fait qu'un chemin au plus peut relier deux sommets, ce qui signifie que sans une arête $\{x, y\}$, les deux sommets x et y ne sont plus reliés dans G.

10 Montrer d'abord qu'ajouter une arête dans un graphe diminue le nombre de composantes connexes ou crée un cycle.

11 Un élément est un représentant quand sa case est négative, et un élément qui n'est pas un représentant pointe dans le tableau `parent` soit vers un représentant, soit vers un élément dont on peut chercher le représentant.

13 Raisonner en terme d'invariant de la partition ou par récurrence sur le nombre de réunions effectuées pour construire la partition.

15 Utiliser la caractérisation de la question 10. Le test de connexité se fait à l'aide de la structure de partition étudiée dans la partie.

20 Utiliser les fonctions précédentes pour générer un chemin aléatoire et le greffer pour chaque sommet qui n'est pas dans l'arbre.

24 Utiliser une division euclidienne.

25 Faire un pattern matching sur les directions possibles.

26 Utiliser la fonction de la question précédente pour déterminer le parent de chaque sommet autre que 0.

29 Par l'absurde, considérer un cycle de $(\mathrm{S}_n^*, \mathrm{B}^*)$ et distinguer les faces intérieures et extérieures.

30 Réutiliser la caractérisation de la question 10, en remarquant que si B a $n-1$ arêtes alors B^* a n^*-1 arêtes. Utiliser enfin le résultat de la question 28 pour passer de G^{**} à G.

31 Vérifier pour chaque arête si l'un des deux sommets est parent de l'autre.

32 Effectuer un parcours en profondeur en partant de la racine pour associer à chaque sommet son parent.

35 Convertir l'arbre en son arbre dual en utilisant le tableau de booléens, grâce aux fonctions `vers_parent` et `vers_couple`. Pour construire le dual, dans la représentation par tableau, appliquer la définition.

36 Commencer par traiter les sommets noirs, puis les gris. Distinguer les sommets gris au bord de l'échiquier et ceux dans les coins, en utilisant les directions indiquées par les sommets noirs autour lorsqu'un sommet gris a comme père le sommet 0.

I. Quelques fonctions auxiliaires

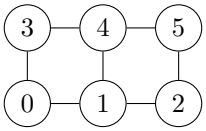
1 Pour un sommet s, notons $\mathrm{d}(s)$ le degré de s, à savoir le nombre d'arêtes dont s est une extrémité. Les graphes étant représentés par listes d'adjacence, la quantité $|\mathrm{A}|$ peut se calculer à partir des degrés des sommets à l'aide de la formule suivante :

$$2|\mathrm{A}| = \sum_{s \in \mathrm{S}} \mathrm{d}(s)$$

En effet, compter le degré des sommets pour tous les sommets revient à compter les arêtes deux fois : $\{x, y\}$ est comptée une fois lorsque x apparaît comme voisin de y, et une seconde fois lorsque y apparaît comme voisin de x. Cette formule mène au programme suivant :

```
let nombre_aretes g =
  let n = Array.length g in
  let k = ref 0 in
  for i = 0 to n - 1 do
    k := !k + List.length g.(i)
  (*List.length g.(i) représente le degré de i dans g*)
  done;
  !k /2 ;;
```

2 Commençons par dessiner le graphe $\mathrm{G}_{3,2}$:



On liste alors aisément les voisins de chaque sommet, d'où l'instruction suivante :

```
let g = [|[|1;3|];[|0;2;4|];[|1;5|];[|0;4|];[|1;3;5|];[|2;4|]|];;
```

3 Le programme consiste simplement à construire un tableau dans lequel chaque liste du tableau donné en entrée est convertie en tableau :

```
let adjacence g =
  let n = Array.length g in
  let sortie = Array.make n [||] in
  for i = 0 to n - 1 do
    sortie.(i) <- Array.of_list g.(i)
  done;
  sortie;;
```

Évaluons la complexité de cette fonction :

- les fonctions `Array.length` et `Array.make` s'exécutent en temps constant ;
- la boucle `for` contient n itérations, sa complexité est la somme des complexités des appels à `Array.of_list` ;
- la fonction `Array.of_list` a une complexité linéaire en la taille de la liste.

La complexité de la fonction `adjacence` est donc linéaire en la somme des tailles des listes de son graphe d'entrée, ce qui correspond à la somme des degrés des sommets, valant $2m$. On en conclut que

La complexité de la fonction `adjacence` est en $\mathrm{O}(n + m)$.

4 Pour déterminer le rang de $\{s,t\}$, on procède en plusieurs étapes :

- On détermine en premier lieu si l'arête est verticale. Pour ce faire, on sait déjà que s et t sont adjacents, donc $t-s$ vaut 1 ou p, et il suffit ainsi seulement de tester si leur différence vaut p.
- Si l'arête est verticale, il suffit de compter les arêtes qui se situent à gauche ou en-dessous de l'arête donnée. Notons a (resp. b) le numéro de ligne (resp. de colonne) de s en partant du bas (resp. de la gauche).
  - il y a $b\cdot(q-1)$ arêtes à gauche de $\{s,t\}$, car il y a $q-1$ arêtes par colonne ;
  - il y a a arêtes en-dessous de $\{s,t\}$.

  L'arête est donc de rang $a+(q-1)\cdot b$.
- De façon similaire, on calcule le rang d'une arête horizontale mais en comptant d'abord par ligne, puis par colonne. Il ne faut bien sûr pas oublier d'ajouter $p\cdot(q-1)$ pour tenir compte de toutes les arêtes verticales comptées avant.

Le code résultant de ce raisonnement est le suivant :

```
let rang (p,q) (s,t) =
  let a, b = s / p, s mod p in
  if t-s=p then
    a + b * (q-1)
  else
    a * (p-1) + b + p * (q-1) ;;
```

5 Comme précédemment, pour déterminer les sommets correspondant au rang i, raisonnons par étapes :

- Déterminons d'abord si l'arête cherchée est verticale ou horizontale : il suffit de regarder si son rang est plus ou moins grand que le nombre d'arêtes verticales du graphe, soit $p\cdot(q-1)$.
- Si l'arête est verticale, on sait qu'elle est précédée de i arêtes. Cherchons alors la valeur de l'indice de s, à laquelle on ajoutera p pour trouver la valeur de l'indice de t. En notant à nouveau a l'indice de ligne de s et b l'indice de colonne de s, on sait que $s=a+b\cdot p$. Pour trouver a et b, on effectue la division euclidienne de i par $q-1$ puisque la question précédente assure que $i=a+b\cdot(q-1)$ avec $0\leqslant a<q$.
- Un raisonnement analogue, en enlevant $p\times(q-1)$ au rang de l'arête, permet de trouver l'indice de l'extrémité gauche d'une arête horizontale (on ajoute alors 1 pour obtenir l'indice de l'extrémité droite).

Ce code s'en déduit directement :

```
let sommets (p,q) i =
  if i < p * (q-1) then
    let a, b = i mod (q-1), i / (q-1) in
    let s = a * p + b in
    s, s + p
  else
    let i' = i - p * (q-1) in
    let a, b = i' / (p-1), i' mod (p-1) in
    let s = a * p + b in
    s, s + 1 ;;
```