Option informatique

CONCOURS CENTRALE•SUPÉLEC

MP

2021

4 heures

Calculatrice autorisée

Arbres couvrants et pavages

Le seul langage de programmation autorisé dans cette épreuve est Caml.

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` ou `incr` ainsi que les opérateurs comme `/` ou `mod`) peuvent être librement utilisés.

On utilisera également le générateur pseudo-aléatoire `int` du module `Random`. Quand `n` est un entier supérieur ou égal à 1, l'appel `Random.int n` renvoie un entier dans l'intervalle $[\![0, n[\![$ (compris entre 0 inclus et `n` exclu) en simulant une loi uniforme. Les appels successifs à cette fonction fournissent des résultats indépendants.

Les candidats ne devront faire appel à aucun autre module.

En Caml, les tableaux d'objets d'un certain type `'a` sont représentés par le type `'a array`. L'expression `tab.(i)` permet d'accéder à l'élément situé en `i`-ème position du tableau `tab`. Dans le texte, en dehors du code Caml, cet élément sera noté $Tab[i]$. Plus généralement, les objets mathématiques seront notés G, s, Adj... et représentés en Caml par `g`, `s`, `adj`... sans que cela soit systématiquement explicité.

Dans ce problème, un graphe G est un couple (S, A) où :

— S est un ensemble fini dont les éléments sont les *sommets de* G ;

— $A = (a_0, a_1, ..., a_{m-1})$ est la suite des *arêtes de* G, une arête étant une partie $a = \{s, t\}$ de S de cardinal 2. Les sommets s et t sont appelés les *extrémités* de l'arête a et on dira que a *relie* s *et* t. Si s et t sont reliés par une arête, on dit qu'ils sont *voisins* ou *adjacents*.

Ainsi, les graphes sont non orientés et il n'y a pas d'arête reliant un sommet à lui-même. Par contre, à partir de la partie V, il est possible que plusieurs arêtes aient les mêmes extrémités.

Par convention, nous noterons n (respectivement m) le nombre de sommets (respectivement d'arêtes) du graphe et nous supposerons que $S = \{0, 1, ..., n-1\} = S_n$.

Un graphe sera représenté par le type

```
type graphe = int list array
```

Si G est un graphe représenté par `g`, alors le nombre de sommets n du graphe est donné par la longueur du tableau `g`. De plus, si $s \in S$, alors `g.(s)` est une liste contenant les indices des sommets voisins de s, dans un ordre quelconque, chaque sommet apparaissant autant de fois qu'il existe d'arêtes vers s.

Les complexités temporelles demandées sont des complexités *dans le pire des cas* et seront exprimées sous la forme $\mathcal{O}(f(n, m))$, où f est une fonction la plus simple possible.

Nous utiliserons la représentation graphique habituelle des graphes. Le graphe G, défini par l'instruction

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

peut être représenté par le graphique de la figure 1.

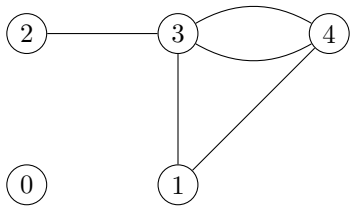

Figure 1 Exemple de graphe

Pour p, q entiers naturels non nuls, nous noterons $G_{p,q}$ le graphe, appelé *graphe quadrillage*, dont les sommets sont placés sur p colonnes et q lignes, chaque sommet étant relié à ses voisins directs. On convient de numéroter les sommets de gauche à droite et de bas en haut.

Par ailleurs, l'ordre des arêtes ayant une importance en partie V, on convient de numéroter les $p(q-1)+q(p-1)$ arêtes en commençant par les arêtes verticales, de bas en haut et de gauche à droite, puis les arêtes horizontales, de gauche à droite et de bas en haut, comme le montre la figure 2, où $p = 4$ et $q = 3$. Le *rang d'une arête* a de $G_{p,q}$ est l'indice i tel que $a = a_i$.

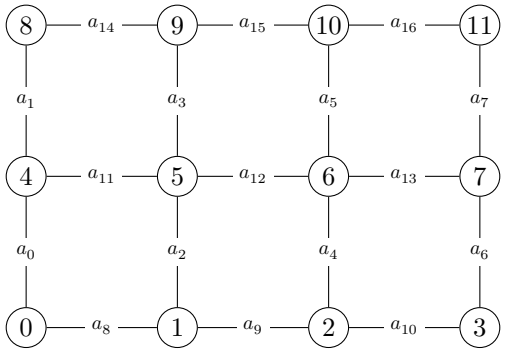

Figure 2 Le graphe $G_{4,3}$

Soit $G = (S_n, A)$ un graphe.

— Si $s \in S_n$, le *tableau d'adjacence* de s est le tableau, dans un ordre quelconque, des voisins de s, chaque sommet apparaissant autant de fois qu'il existe d'arêtes vers s. On note Adj le tableau de taille n tel que, pour tout $s \in \{0, 1, ..., n-1\}$, $Adj[s]$ est le tableau d'adjacence du sommet s. Adj est donc représenté par une variable `adj` de type `int array array`.

— Un *chemin* dans G est une suite $c = (s_0, s_1, ..., s_{j-1}, s_j, ..., s_{k-1}, s_k)$ où pour tout j compris entre 1 et k, s_{j-1} et s_j sont des sommets voisins. On dira que c est un chemin de s_0 à s_k de longueur k. Par convention, pour s sommet de G, il existe un chemin de longueur nulle de s à s.

— La *composante connexe* d'un sommet s de G, notée C_s, est l'ensemble des sommets t de G tels qu'il existe un chemin de s à t.

— On dit que G est *connexe* si pour tous sommets s et t de G, il existe un chemin de s à t.

— Un *cycle* dans G est un chemin de longueur $k \geqslant 2$ d'un sommet à lui-même et dont les arêtes sont deux à deux distinctes. On dit que G est *acyclique* s'il ne contient aucun cycle.

— Un *arbre* est un graphe connexe acyclique.

I Quelques fonctions auxiliaires

Q 1. Écrire une fonction

```
nombre_aretes : graphe -> int
```

qui, appliquée à `g` représentant un graphe $G = (S, A)$, renvoie le cardinal de A.

Q 2. Écrire une commande Caml permettant de créer le tableau des tableaux d'adjacence Adj associé au graphe $G_{3,2}$.

On rappelle que la fonction `Array.of_list : 'a list -> 'a array` prend en argument une liste $[x_0; x_1; ...; x_{k-1}]$ d'éléments de type `'a` et renvoie le tableau $[|x_0; x_1; ...; x_{k-1}|]$.

Q 3. Écrire une fonction

```
adjacence : graphe -> int array array
```

qui, appliquée à `g` représentant un graphe G, renvoie `adj` représentant le tableau des tableaux d'adjacence Adj associé à G. La fonction `adjacence` devra être de complexité $\mathcal{O}(m+n)$ et on demande de justifier cette complexité.

Q 4. Écrire une fonction

```
rang : int * int -> int * int -> int
```

telle que `rang (p, q) (s, t)` renvoie le rang de l'arête $\{s, t\}$ dans le graphe quadrillage $G_{p,q}$. Par exemple, `rang (4, 3) (6, 7)` renvoie `13`. On suppose que s et t sont adjacents et $s < t$ et on ne demande pas de le vérifier.

Q 5. Écrire de même sa fonction réciproque

```
sommets : int * int -> int -> int * int
```

telle que `sommets (p, q) i` renvoie le couple `(s, t)` tel que $a_i = \{s, t\}$ et $s < t$ dans le graphe $G_{p,q}$.

Q 6. Écrire une fonction

```
quadrillage : int -> int -> graphe
```

telle que l'instruction `quadrillage p q` renvoie le graphe représentant $G_{p,q}$.

II Caractérisation des arbres

Soit $G = (S_n, A)$ un graphe à n sommets et m arêtes, représenté par g. Les arêtes de G sont donc numérotées $A = (a_0, a_1, ..., a_{m-1})$.

II.A – Propriétés sur les arbres

Q 7. Montrer que les composantes connexes de G forment une partition de S_n, c'est-à-dire que :

— $\forall s \in S_n$, $C_s \neq \emptyset$;

— $S_n = \bigcup_{s \in S_n} C_s$;

— pour tous sommets s et t, soit $C_s = C_t$, soit $C_s \cap C_t = \emptyset$.

Q 8. Montrer que si s et t sont deux sommets de G tels que $t \in C_s$, il existe un plus court chemin de s à t et que les sommets d'un plus court chemin sont deux à deux distincts.

Pour $k \in \{0, 1, ..., m\}$, G_k désigne le graphe $(S_n, (a_0, a_1, ..., a_{k-1}))$ obtenu en ne conservant que les k premières arêtes de G.

Q 9. On suppose que G est un arbre. Montrer que pour tout $k \in \{0, 1, ..., m-1\}$, les extrémités de a_k appartiennent à deux composantes connexes différentes de G_k. En déduire que $m = n - 1$.

Q 10. Montrer que les trois propriétés suivantes sont équivalentes :

(i) G est un arbre ;

(ii) G est connexe et $m = n - 1$;

(iii) G est acyclique et $m = n - 1$.

II.B – Manipulation de partitions

Nous souhaitons écrire une fonction qui teste si un graphe est un arbre. Nous allons pour cela utiliser une structure de données permettant de manipuler les partitions de l'ensemble S_n : si $\mathcal{P} = \{X_1, X_2, ..., X_k\}$ est une partition de S_n, on choisit dans chaque partie X_i, un élément particulier r_i, appelé *représentant* de X_i. Notre structure de données doit nous permettre :

— de calculer, pour $s \in S_n$, le représentant de la partie X_i contenant s ; cet élément sera également appelé *représentant de* s ;

— pour deux entiers s et t représentant des parties distinctes X_i et X_j, de transformer $\mathcal{P}$ en réunissant X_i et X_j, s ou t devenant le représentant de la partie $X_i \cup X_j$.

Nous représenterons une partition $\mathcal{P} = \{X_1, ..., X_k\}$ de S_n par une *forêt* : chaque X_i est représenté par un arbre dont les noeuds sont étiquetés par les éléments de X_i et de racine le représentant r_i de X_i, les arcs étant orientés vers la racine. Nous noterons $h(r_i)$ la hauteur de l'arbre X_i, c'est-à-dire la longueur de sa plus longue branche. Ainsi, $\mathcal{P}_9 = \{\{0, 2\}, \{1, 5, 6, 8\}, \{3, 4, 7\}\}$ est une partition de S_9 et peut par exemple être représentée par la forêt de la figure 3.

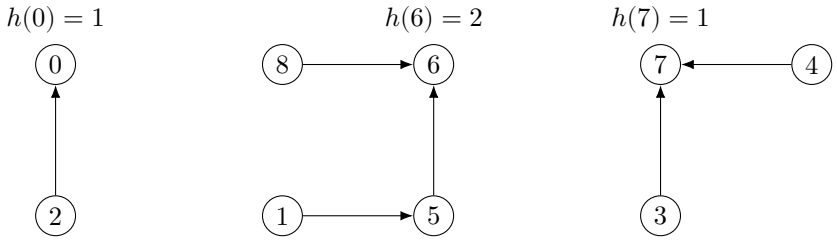

Figure 3 Une représentation de $\mathcal{P}_9 = \{\{0, 2\}, \{1, 5, 6, 8\}, \{3, 4, 7\}\}$

Le calcul du représentant d'un entier $s \in S_n$ se fait donc en remontant jusqu'à la racine de l'arbre (ce qui justifie l'orientation des arcs). Dans l'exemple, les représentants de 2, 1 et 7 sont respectivement 0, 6 et 7.

Une partition $\mathcal{P}$ de $S_n = \{0, 1, ..., n-1\}$ sera représentée par un tableau d'entiers P de taille n de sorte que pour tout $s \in \{0, 1, ..., n-1\}$, $P[s]$ est le père de s si s n'est pas un représentant, et est égal à $-1 - h(s)$ si s est un représentant. La partition $\mathcal{P}_9$ peut ainsi être représentée par le tableau

```
[| -2; 5; 0; 7; 7; 6; -3; -2; 6 |]
```

La réunion de deux parties X_i et X_j de représentants s et t distincts se fait selon la méthode suivante :

— si $h(s) > h(t)$, s est choisi pour représentant de la partie $X_i \cup X_j$ et devient le père de t ;

— si $h(s) \leqslant h(t)$, t est choisi pour représentant de la partie $X_i \cup X_j$ et devient le père de s.

Par exemple, la réunion des parties X_1 et X_3 de la partition $\mathcal{P}_9$ peut donner la nouvelle forêt représentée figure 4.

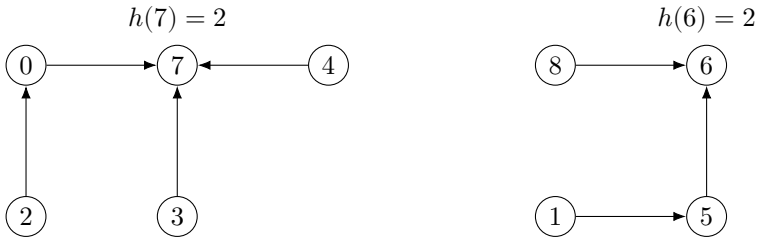

Figure 4 Une représentation de $\{\{0,2\} \cup \{3,4,7\}, \{1,5,6,8\}\}$

Q 11. Écrire une fonction

```
representant : int array -> int -> int
```

qui, appliquée à un tableau représentant une partition de S_n et à $s \in S_n$, renvoie le représentant de s.

Q 12. Écrire une fonction

```
union : int array -> int -> int -> unit
```

qui, appliquée à un tableau représentant une partition de S_n et à deux représentants s et t distincts, modifie la partition en réunissant les arbres associés à s et t, selon la méthode expliquée ci-dessus, sans oublier, si nécessaire, de modifier $h(s)$ ou $h(t)$.

On note $\mathcal{P}_n^{(0)}$ la partition de S_n où toutes les parties sont des singletons, c'est-à-dire $\mathcal{P}_n^{(0)} = \{\{0\}, \{1\}, ..., \{n-1\}\}$.

Q 13. Soit $\mathcal{P}$ une partition de S_n construite à partir de $\mathcal{P}_n^{(0)}$ par des réunions successives selon la méthode précédente. Montrer que si s est le représentant d'une partie $X \in \mathcal{P}$, alors le cardinal de X vérifie $|X| \geqslant 2^{h(s)}$.

Q 14. En déduire les complexités des deux fonctions précédentes dans le pire des cas en fonction de n pour une partition $\mathcal{P}$ construite à partir de $\mathcal{P}_n^{(0)}$.

Q 15. Écrire une fonction

```
est_un_arbre : graphe -> bool
```

qui, appliquée à un graphe `g` représentant un graphe G, renvoie `true` si G est un arbre et `false` sinon.

III Algorithme de Wilson : arbre couvrant aléatoire

Si $G = (S_n, A)$ est un graphe connexe et si $r \in S_n$, un *arbre couvrant de G enraciné en r* est un arbre $\mathcal{T} = (S_n, B)$ tel que $B \subset A$ et dont r a été choisi pour racine. On convient alors d'orienter les arêtes de l'arbre en direction de la racine r et de représenter $\mathcal{T}$ par un tableau `parent` de taille n, `parent.(s)` étant égal au père de s dans $\mathcal{T}$ si $s \neq r$, et à -1 si $s = r$.

La figure 5 représente deux arbres couvrants du graphe $G_{3,2}$, enracinés respectivement en 0 et 2, et leurs représentations.

Figure 5 Deux arbres couvrants enracinés de $G_{3,2}$ et leurs représentations

Dans cette partie, nous supposons que $G = (S_n, A)$ est un graphe connexe, représenté par `g`, que r est un sommet de G et nous cherchons à construire un arbre couvrant aléatoire de G enraciné en r.

Nous allons pour cela faire évoluer dynamiquement un arbre $\mathcal{T}$, représenté par un tableau `parent` de longueur n vérifiant :

— `parent.(r) = -1` ;

— si $s \in S_n$ n'est pas un sommet de $\mathcal{T}$, `parent.(s) = -2` ;

— si $s \in S_n$ est un sommet de $\mathcal{T}$ autre que la racine, `parent.(s)` est le père de s dans l'arbre $\mathcal{T}$.

Nous partons de l'arbre réduit à la racine r, en posant `parent.(r) = -1` et `parent.(s) = -2` pour tout sommet $s \neq r$. Pour tout sommet s, quand s n'est pas déjà dans l'arbre, on construit un chemin aléatoire c sans boucle qui part de s et qui s'arrête dès qu'il a atteint un sommet de $\mathcal{T}$. La méthode de construction est la suivante :

— au début du calcul, c est réduit à s ;

— à tout moment, c est de la forme $(s = s_0, s_1, \cdots, s_{k-1}, s_k)$ où les sommets $s_0, ..., s_k$ sont deux à deux distincts et les sommets $s_0, ..., s_{k-1}$ ne sont pas des sommets de $\mathcal{T}$;

— tant que l'extrémité s_k n'est pas un sommet de $\mathcal{T}$, on choisit aléatoirement et uniformément un voisin u de s_k et on distingue,

- si $u \in \{s_0, s_1, ..., s_k\}$, on supprime le cycle qui vient d'être formé et u devient le nouveau point d'arrivée du chemin c ;
- sinon, c devient le chemin $(s = s_0, s_1, \cdots, s_{k-1}, s_k, u)$;

— on renvoie le chemin $(s = s_0, s_1, \cdots, s_{k-1}, s_k)$ une fois le calcul terminé.

On représentera un chemin en Caml par le type :

```
type chemin = {debut : int; mutable fin : int; suivant : int array}
```

de telle sorte que si le chemin $c = (s_0, s_1, \cdots, s_{k-1}, s_k)$ est représenté par `c`, alors `c.debut` est égal à s_0, `c.fin` (qui est un champ mutable) est égal à s_k, et `c.suivant` est un tableau de taille n tel que pour $j \in \{0, ..., k-1\}$, la case d'indice s_j de `c.suivant` contient le sommet s_{j+1}. Pour tout autre sommet $t \notin \{s_0, s_1, ..., s_{k-1}\}$, `c.suivant.(t)` pourra prendre une valeur arbitraire.

On rappelle que si `c` est un objet de type chemin et `u` un entier représentant un sommet, la modification du champ mutable `c.fin` pour y mettre `u` peut se faire avec la commande :

```
c.fin <- u
```

Q 16. Déterminer, dans le graphe $G_{3,2}$, le chemin représenté par l'objet

```
{debut = 1; fin = 4; suivant = [|-5; 2; 5; 3; -1; 4|]}
```

Q 17. Que peut-on dire de la terminaison de cet algorithme ?

Q 18. Écrire une fonction

```
marche_aleatoire : int array array -> int array -> int -> chemin
```

telle que l'appel `marche_aleatoire adj parent s` renvoie l'objet `c` représentant un chemin de s à un sommet $t \in \mathcal{T}$ obtenu comme décrit précédemment. On rappelle que `adj` représente le tableau des tableaux d'adjacence Adj du graphe G, que `parent` représente l'arbre (en construction) $\mathcal{T}$ et on supposera que $s \in S_n$ n'appartient pas à $\mathcal{T}$.

L'algorithme d'évolution de $\mathcal{T}$, appelé algorithme de Wilson, est alors le suivant : à partir de l'arbre $\mathcal{T}$ réduit à r, pour s variant de 0 à $n-1$, si s n'est pas dans $\mathcal{T}$, on calcule un chemin de s à un sommet $t \in \mathcal{T}$ codé par `c` grâce à la fonction `marche_aleatoire` et on *greffe* c à $\mathcal{T}$, en ajoutant à $\mathcal{T}$ les arêtes du chemin c, orientées de s vers u, u étant le dernier sommet du chemin.

Q 19. Écrire une fonction

```
greffe : int array -> chemin -> unit
```

telle que l'instruction `greffe parent c` modifie `parent` de sorte à représenter l'arbre obtenu après la greffe du chemin c sur $\mathcal{T}$.

Q 20. Écrire une fonction

```
wilson : graphe -> int -> int array
```

tel que `wilson g r` renvoie un arbre couvrant aléatoire du graphe G de racine r.

IV Arbres couvrants et pavages par des dominos

Soient p et q deux entiers strictement supérieurs à 1. Le but des deux dernières parties est de construire des pavages aléatoires par des dominos de taille 2×1 d'un échiquier à $2p-1$ colonnes et $2q-1$ lignes privé de son coin inférieur gauche (il reste bien un nombre pair de cases à recouvrir). Nous noterons $E_{p,q}$ cet échiquier, dont les cases sont repérées par des couples de coordonnées entières (x, y), avec $0 \leqslant x \leqslant 2(p-1)$ et $0 \leqslant y \leqslant 2(q-1)$, et $E'_{p,q}$ l'échiquier $E_{p,q}$ privé de son coin inférieur gauche, c'est-à-dire de la case $(0, 0)$. Les cases de $E_{p,q}$ sont colorées en noir, gris et blanc, comme dans l'exemple de l'échiquier $E_{4,3}$ présenté figure 6.

Les cases dont les deux coordonnées sont paires sont colorées en noir, celles dont les deux coordonnées sont impaires sont colorées en gris, les autres sont colorées en blanc.

Si $\mathcal{P}$ est un pavage de $E'_{p,q}$, chaque case noire ou grise de $E'_{p,q}$ est recouverte par un domino, qui est orienté, à partir de la case noire ou grise considérée, vers le sud, l'ouest, le nord ou l'est.

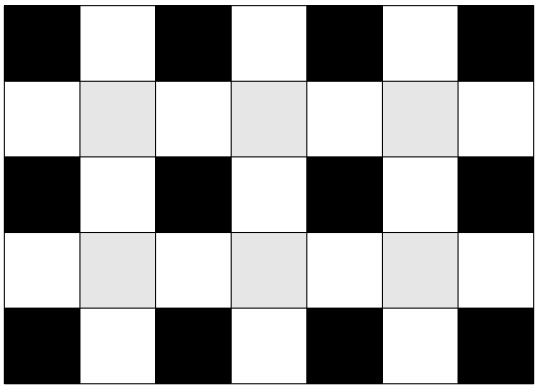

Figure 6 L'échiquier $E_{4,3}$

Nous définissons le type

```
type direction = S | W | N | E
```

et nous codons $\mathcal{P}$ par une matrice `pavage` de taille $(2p-1) \times (2q-1)$, avec, pour $i \in \{0, ..., 2p-2\}$ et $j \in \{0, ..., 2q-1\}$:

— si i et j ont la même parité et si $(i, j) \neq (0, 0)$, `p.(i).(j)` est la direction du domino qui recouvre la case (i, j) (cette case est soit noire, soit grise) ;

— sinon, `pavage.(i).(j)` prend la valeur `N` (ces valeurs ne jouent aucun rôle).

Ainsi, si `p1` est la matrice associée au pavage $\mathcal{P}_1$ de $E'_{4,3}$ représenté figure 7 (pour mieux distinguer les dominos, les cases ont été remplacées par des ronds colorés) on a par exemple `p1.(2).(4) = S`, `p1.(4).(2) = W`, `p1.(3).(3) = N`, `p1.(5).(1) = E`, comme indiqué par les sens des flèches.

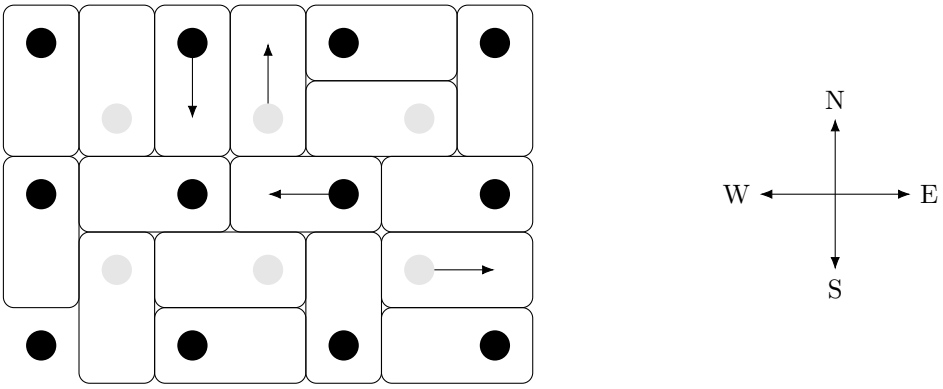

Figure 7 Le pavage $\mathcal{P}_1$ de $E^*_{4,3}$

Les cases noires de l'échiquier $E_{p,q}$ seront vues comme les sommets du graphe $G_{p,q}$ (la case noire située en bas à gauche est identifiée au sommet 0 du graphe $G_{p,q}$). Un résultat de Neville Temperley explicite une bijection canonique φ entre les pavages de $E'_{p,q}$ et les arbres couvrants de $G_{p,q}$. La construction de $\varphi(\mathcal{P})$ à partir d'un pavage $\mathcal{P}$ s'obtient en ne conservant que les dominos recouvrant une case noire.

Pour toute la suite du sujet, les valeurs de p et q seront supposées contenues dans les variables globales `p` et `q`.

On suppose de plus définie une variable globale `g`, représentant $G_{p,q}$, par l'instruction

```
let g = quadrillage p q ;;
```

IV.A – Exemples

Q 21. Dessiner l'arbre couvrant *non enraciné* $\mathcal{T}_1 = \varphi(\mathcal{P}_1)$ de $G_{4,3}$.

Q 22. Considérons inversement l'arbre couvrant $\mathcal{T}_2$ de $G_{4,3}$ décrit figure 8. Par un procédé réciproque à celui utilisé à la question précédente, dessiner le pavage $\mathcal{P}_2 = \varphi^{-1}(\mathcal{T}_2)$ de $E'_{4,3}$.

IV.B – Calcul de l'arbre couvrant associé à un pavage

Q 23. Soit $\mathcal{P}$ un pavage de $E'_{p,q}$, associé à l'arbre couvrant $\mathcal{T} = \varphi(\mathcal{P})$ de $G_{p,q}$. Décrire un procédé permettant de calculer le père d'un sommet $s \in \{1, ..., pq-1\}$ dans l'arbre $\mathcal{T}$ enraciné en 0. On ne demande pas de démontrer que la structure ainsi obtenue est bien un arbre de racine 0.

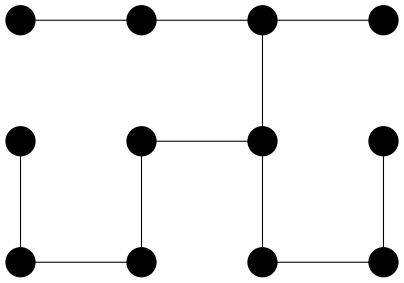

Figure 8 L'arbre couvant $\mathcal{T}_2$

Q 24. Écrire une fonction

```
coord_noire : int -> int * int
```

telle que l'instruction `coord_noire s` renvoie le couple des coordonnées de la case correspondant au sommet s du graphe $G_{p,q}$. Par exemple, dans le graphe $G_{4,3}$ représenté figure 2, `coord_noire 6` renverra `(4, 2)`.

Q 25. Écrire une fonction

```
sommet_direction : int -> direction -> int
```

telle que `sommet_direction s d` renvoie le sommet t qui se trouve directement dans la direction `d` du sommet s de $G_{p,q}$. Cette fonction renverra `-1` si un tel sommet n'existe pas. Par exemple, dans le graphe $G_{4,3}$ représenté figure 2, `sommet_direction 5 W` renverra `4`.

Q 26. Écrire une fonction

```
phi : direction array array -> int array
```

qui, appliquée à une matrice `pavage` représentant un pavage $\mathcal{P}$ de $E'_{p,q}$, renvoie le tableau `parent` représentant l'arbre $\mathcal{T} = \varphi(\mathcal{P})$ enraciné en 0 (cette représentation est définie au début de la partie III).

V Utilisation du dual pour la construction d'un pavage

V.A – Graphe dual de $G_{p,q}$

Le graphe $G_{p,q}$ est un *graphe planaire*, c'est-à-dire qu'il possède une représentation graphique dans laquelle deux arêtes distinctes ne se coupent pas, sauf peut-être en leurs extrémités. Cette *représentation planaire* sépare le plan en $(p-1)(q-1)+1$ faces, que l'on va numéroter de gauche à droite et de bas en haut, la face non bornée portant le numéro 0. Le *graphe dual* de $G_{p,q}$, noté $G^*_{p,q}$, est alors le graphe à $(p-1)(q-1)+1$ sommets (chaque sommet correspond à une des faces délimitées par la représentation de $G_{p,q}$) et qui possède autant d'arêtes que $G_{p,q}$: chaque arête a de $G_{p,q}$ donne une arête a^* entre les deux faces du plan qu'elle sépare. Ce graphe est également planaire. Attention, contrairement au graphe $G_{p,q}$, le graphe $G^*_{p,q}$ *peut posséder plusieurs arêtes entre les mêmes sommets.* Les sommets de $G_{p,q}$ ont déjà été identifiés aux cases noires de $E_{p,q}$; les cases grises seront identifiées aux sommets de $G^*_{p,q}$ distincts de 0 et les cases blanches aux arêtes de $G_{p,q}$ (et donc également aux arêtes de $G^*_{p,q}$, puisque $a \mapsto a^*$ est une bijection). La figure 9 explicite ces identifications dans le cas du graphe $G_{4,3}$. Les sommets de $G_{4,3}$ sont identifiés par des sommets gris foncés, ceux de $G^*_{4,3}$ par des sommets gris clairs, les arêtes de $G_{4,3}$ par des traits pleins et celles de $G^*_{4,3}$ par des traits hachurés. L'intersection de chaque (a, a^*) est marquée par un carré symbolisant une case blanche de $E_{4,3}$.

Ainsi, le sommet 6 de $G_{4,3}$ correspond à la case $(4, 2)$ de $E_{4,3}$, le sommet 4 de $G^*_{4,3}$ correspond à la case $(1, 3)$ et les arêtes a_9 de $G_{4,3}$ et a^*_9 de $G^*_{4,3}$ correspondent à la case $(3, 0)$.

Dans la suite du problème, nous notons :

- $n = pq$ le nombre de sommets de $G_{p,q}$;
- $n^* = (p-1)(q-1)+1$ le nombre de sommets de $G^*_{p,q}$;
- $m = p(q-1) + q(p-1)$ le nombre d'arêtes de $G_{p,q}$ et de $G^*_{p,q}$;
- A l'ensemble des arêtes de $G_{p,q}$;
- A^* l'ensemble des arêtes de $G^*_{p,q}$.

Nous supposerons définies pour la suite :

- une fonction `coord_grise : int -> int * int` qui, appliquée à un sommet de $G^*_{p,q}$ autre que 0, renvoie les coordonnées de la case grise qui correspond à ce sommet ;
- une fonction `numero : int * int -> int` qui, appliquée à un couple (x, y) représentant une case noire ou grise de l'échiquier $E_{p,q}$, renvoie le sommet du graphe $G_{p,q}$ ou $G^*_{p,q}$ associé à cette case. On supposera également que dans tous les autres cas, `numero` renvoie la valeur 0, y compris si les coordonnées sont en dehors de l'échiquier. Quand $p = 4$ et $q = 3$, nous avons par exemple : `numero (4, 2) = 6`, `numero (1, 3) = 4`, et `numero (3, 0) = 0`.

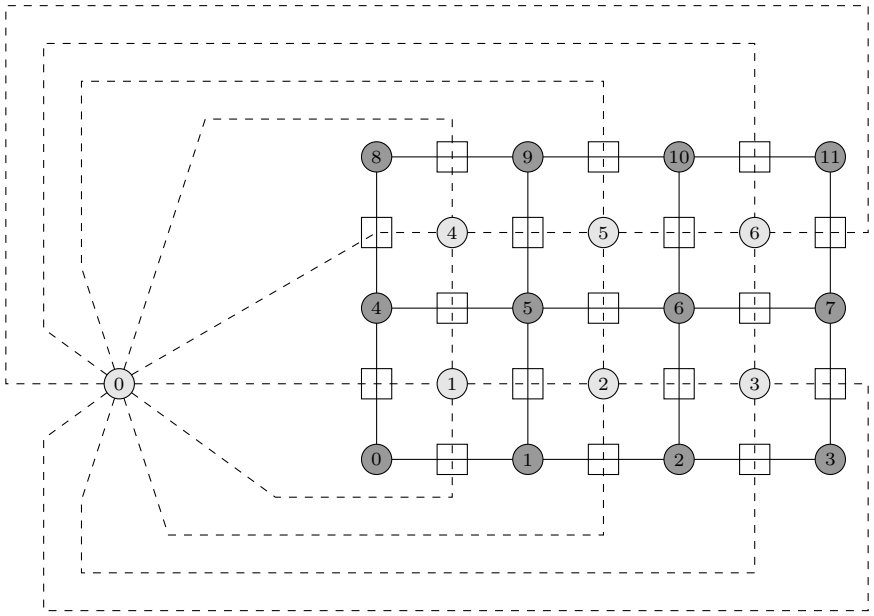

Figure 9 $G_{4,3}$ et $G^*_{4,3}$

Q 27. En utilisant les fonctions précédentes, écrire une fonction

```
dual : unit -> graphe
```

telle que l'instruction `dual ()` renvoie le graphe représentant $G^*_{p,q}$. On respectera la numérotation des sommets de $G^*_{p,q}$ décrite ci-dessus (figure 9). Par ailleurs, les listes d'adjacence du graphe dual contiendront des sommets en doublon s'il existe plusieurs arêtes entre des mêmes sommets.

On suppose désormais définie une variable globale `g_etoile` par l'instruction

```
let g_etoile = dual () ;;
```

V.B – Dual d'un arbre couvrant

Q 28. Montrer que, quitte à renuméroter les sommets, le dual de $G^*_{p,q}$ est le graphe $G_{p,q}$.

Soit B une partie de A. On note B^* la partie de A^* définie par

$$\forall i \in \{0, 1, ..., m-1\},\ a^*_i \in B^* \iff a_i \notin B.$$

Q 29. Montrer que si le graphe (S_n, B) possède un cycle, le graphe (S_{n^*}, B^*) n'est pas connexe.

Q 30. En déduire que (S_n, B) est un arbre couvrant de $G_{p,q}$ si et seulement si (S_{n^*}, B^*) est un arbre couvrant de $G^*_{p,q}$.

Si $\mathcal{T} = (S_n, B)$ est un arbre couvrant de $G_{p,q}$, nous noterons $\mathcal{T}^*$ l'arbre couvrant (S_{n^*}, B^*) de $G^*_{p,q}$.

Pour construire l'arbre $\mathcal{T}^*$, nous utiliserons une représentation différente des arbres que celle introduite en partie III : on pourra représenter un arbre couvrant $\mathcal{T} = (S, B)$ enraciné en r par un couple `(r, b)` où `b` est un tableau de booléens de longueur m représentant B, c'est-à-dire tel que `b.(i)` prend la valeur `true` si et seulement si $a_i \in B$.

La figure 10 représente les deux arbres couvrants du graphe $G_{3,2}$, présenté en figure 5, enracinés respectivement en 0 et 2, et leurs représentations par un couple.


```
(0, [| true; true; false; true; true; false; true |])
```

```
(2, [| true; true; true; true; true; false; false |])
```

Figure 10 Deux arbres couvrants enracinés de $G_{3,2}$ et leurs représentations

Q 31. Écrire une fonction

```
vers_couple : int array -> int * bool array
```

telle que l'instruction `vers_couple parent`, où `parent` est un tableau représentant un arbre enraciné de $G_{p,q}$, renvoie le couple `(r, b)` décrit ci-dessus.

Q 32. Détailler en français un algorithme permettant de construire `parent` à partir du couple (r, B) et de la représentation du graphe $G_{p,q}$ par listes d'adjacences. Cet algorithme est-il applicable à un arbre couvrant du graphe $G^*_{p,q}$? Justifier.

Q 33. Écrire une fonction

```
vers_parent : int * bool array -> int array
```

telle que l'instruction `vers_parent (r, b)`, où `b` désigne un ensemble B tel que (S_n, B) est un arbre de $G_{p,q}$ enraciné en r, renvoie le tableau `parent` représentant cet arbre.

Q 34. Déterminer les complexités de ces deux fonctions de conversion en fonction de n et m.

On supposera écrite une fonction `vers_parent_etoile : int * bool array -> int array`, ayant un fonctionnement similaire à `vers_parent`, qui prend en argument un couple `(r, b_etoile)` correspondant à un arbre couvrant $\mathcal{T}^* = (S_{n^*}, B^*)$ de $G^*_{p,q}$ et renvoie un tableau `parent_etoile` décrivant l'arbre en utilisant la représentation décrite partie III.

Q 35. Écrire une fonction

```
arbre_dual : int array -> int array
```

qui, appliquée au tableau `parent` représentant un arbre couvrant $\mathcal{T}$ de $G_{p,q}$ enraciné en 0, renvoie le tableau `parent_etoile` représentant l'arbre couvrant $\mathcal{T}^*$ de $G^*_{p,q}$ enraciné en 0.

V.C – Calcul du pavage associé à un arbre couvrant

Soit $\mathcal{T} = (S_n, B)$ un arbre couvrant de $G_{p,q}$.

Q 36. Décrire un procédé de construction, à partir des arbres $\mathcal{T}$ et $\mathcal{T}^*$ enracinés en 0, du pavage $\mathcal{P} = \varphi^{-1}(\mathcal{T})$ de $E'_{p,q}$. On expliquera en détail comment traiter les arêtes d'un sommet de $G^*_{p,q}$ vers le sommet 0. On justifiera brièvement que l'on obtient bien un pavage.

Q 37. Écrire une fonction

```
pavage_aleatoire : unit -> direction array array
```

telle que l'instruction `pavage_aleatoire ()` renvoie une matrice de taille $(2p-1)(2q-1)$ représentant un pavage aléatoire de $E'_{p,q}$ par des dominos.

Q 38. Comment adapter cette méthode à la construction de pavages aléatoires d'un échiquier à $2p$ colonnes et $2q-1$ lignes (respectivement à $2p$ colonnes et $2q$ lignes) ?

• • • FIN • • •
