Faculté des Sciences
Aix*Marseille Université

Année universitaire 2018/2019

Site : ☒ Luminy ☒ St-Charles ☐ St-Jérôme ☐ Cht-Gombert ☒ Aix-Montperrin ☐ Aubagne-SATIS
Sujet de : ☒ 1er semestre ☐ 2ème semestre ☐ Session 2 Durée de l'épreuve : 2h
Examen de : L3 Nom du diplôme : Licence d'Informatique
Code du module : SIN5U03 Libellé du module : Algorithmique 2
Calculatrices autorisées : NON Documents autorisés : OUI, notes de Cours/TD/TP

Les questions sont pour la plupart indépendantes et ne sont pas classées par ordre de difficulté. Elles peuvent donc être traitées dans n'importe quel ordre. Une estimation de la difficulté de chaque question est donnée (⋆ facile, ⋆⋆ moyenne, ⋆⋆⋆ difficile), et pour certaines une borne maximum sur la longueur de la réponse, en nombre de caractères. Les réponses correctes dépassant excessivement *cette limite seront considérées fausses.*

1 Parcours de graphes

1. (⋆, 300 caractères) Quelle est la principale différence entre les parcours en largeur et en profondeur, en terme de structures de données utilisées pour les implémenter de manière non récursive ?

   Solution : Le parcours en largeur utilise une file pour mémoriser les arcs découverts mais non parcouru. Le parcours en profondeur utilise une pile.

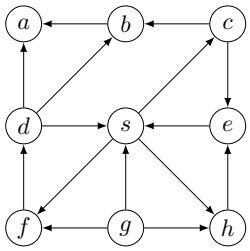
2. (⋆) Donner la complexité asymptotique de ces deux algorithmes.

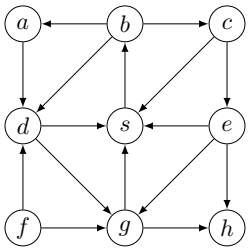
   Solution : La complexité des deux algorithmes est en $O(|E| + |V|)$ sur un graphe $G = (V, E)$, dans les deux cas.

3. (⋆, 200 caractères) Donner une application pour chacun de ces algorithmes.

   Solution : Le parcours en largeur est utilisé pour le calcul de plus courts chemins depuis une source. Le parcours en profondeur est utilisé pour la recherche de composantes fortement connexes.

4. (⋆) Effectuer un parcours en largeur dans le graphe de gauche et un parcours en profondeur dans le graphe de droite, en partant du sommet s : on traitera les arcs sortants dans l'ordre alphabétique croissant. Sur votre copie, dessiner les arborescences de parcours renvoyées par les algorithmes et, dans le cas du parcours en profondeur, donner les ordres d'entrée et de sortie.



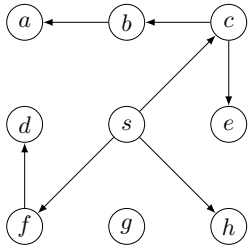


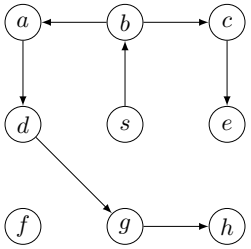
Solution : Pour le graphe de gauche, on visite les arcs dans l'ordre suivant : (s,c), (s,f), (s,h), (c,b), (c,e), (f,d), (h,e), (b,a), (e,s), (d,a), (d,b), (d,s). On obtient l'arborescence à gauche ci-dessous.
Pour le graphe de droite, on visite les arcs dans l'ordre suivant : (s,b), (b,a), (a,d), (d,g), (g,h), (g,s), (d,s), (b,d), (b,c), (c,e), (e,g), (e,h), (e,s), (c,s). On obtient l'arborescence à droite ci-dessous. L'ordre d'entrée est donc

$$s <_e b <_e a <_e d <_e g <_e h <_e c <_e e$$

et l'ordre de sortie

$$h <_s g <_s d <_s a <_s e <_s c <_s b <_s s$$





2 Algorithmes dans des arbres AVL

1. ($\star$, 200 caractères) Quelle structure de données implémente les arbres binaires de recherche (et donc aussi les arbres AVL) ?

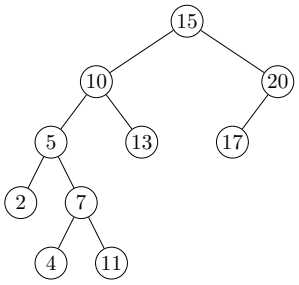
   Solution : Un ABR permet d'implémenter la structure de dictionnaire, permettant de stocker un ensemble de clés dans lequel on peut rechercher, insérer et supprimer une clé.

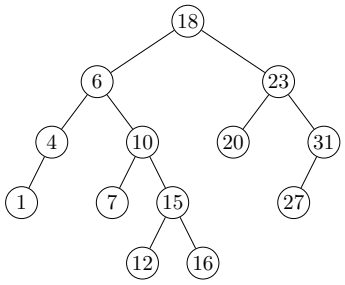
2. ($\star$) Établir la liste de toutes les violations d'invariants d'arbres AVL dans l'arbre de gauche ci-dessous.

   Solution :
   - À la racine, la hauteur du sous-arbre droit est deux de moins que le sous-arbre gauche.
   - Au nœud 10, la hauteur du sous-arbre droit est deux de moins que le sous-arbre gauche. De plus, le sous-arbre gauche contient la clé 11 qui est strictement supérieure à 10.
   - Au nœud 5, le sous-arbre droit contient la clé 4 qui est strictement inférieure à 5.

3. ($\star$, 300 caractères) Pour quelle(s) raison(s) préfère-t-on utiliser des arbres AVL plutôt que des arbres binaires de recherche quelconques ?

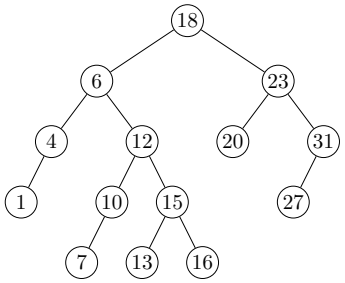
   Solution : Les arbres AVL sont des ABR équilibrés dont la hauteur est logarithmique en fonction de la taille : cela permet d'implémenter les trois opérations d'un dictionnaire avec une complexité $O(\log n)$.





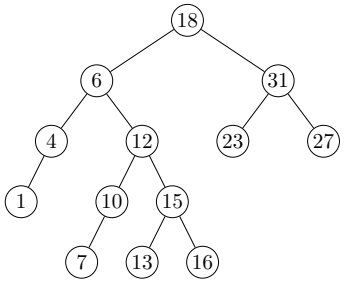
4. ($\star\star$) À partir de l'arbre AVL de droite ci-dessus, insérer la clé 13 (en maintenant la propriété des arbres AVL).

Solution : On insère 13 à une feuille, puis on effectue une rotation double.



5. (⋆⋆) À partir de l'arbre obtenu dans la question précédente, supprimer la clé 20 (en maintenant la propriété des arbres AVL).

Solution : On supprime la clé, puis on effectue une rotation simple.



3 Modélisation et flots

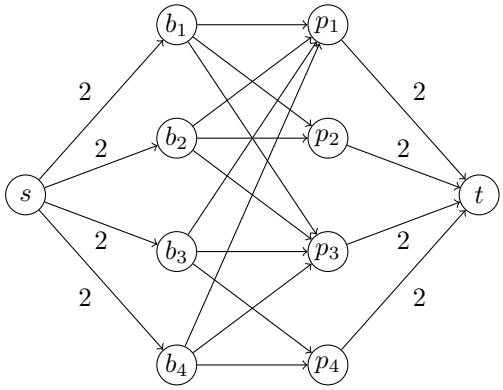
Vous êtes impliqué dans une expérience scientifique nécessitant des relevés de mesures atmosphériques à large échelle. L'expérience nécessite de connaître précisément un ensemble $P = \{p_1, p_2, \ldots, p_n\}$ de n propriétés atmosphériques (telles que le niveau d'ozone et de dioxyde de carbone à différents endroits). Vous avez à votre disposition un ensemble de m ballons que vous pouvez lancer pour faire ces mesures. Une fois lancé, chaque ballon peut mesurer seulement au plus deux propriétés. De plus, il y a des ballons de plusieurs types, chacun ne pouvant mesurer qu'un sous-ensemble des propriétés : pour chaque $i \in \{1, \ldots, m\}$, le ballon i ne peut mesurer que les propriétés $P_i \subseteq P$. Enfin, pour permettre d'obtenir des mesures plus fiables, on souhaite que chaque propriété p_i soit mesurée par au moins k ballons (évidemment, un ballon ne peut pas effectuer deux fois la même mesure).

Considérons un exemple avec $k = 2$, $n = 4$ propriétés p_1, p_2, p_3, p_4 et $m = 4$ ballons tels que les ballons 1 et 2 peuvent mesurer les propriétés $P_1 = P_2 = \{p_1, p_2, p_3\}$ et les ballons 3 et 4 peuvent mesurer les propriétés $P_3 = P_4 = \{p_1, p_3, p_4\}$. Une façon de faire en sorte que chaque propriété soit mesurée par au moins $k = 2$ ballons est la répartition suivante :

— le ballon 1 mesure les propriétés p_1 et p_2 ;
— le ballon 2 mesure les propriétés p_2 et p_3 ;
— le ballon 3 mesure les propriétés p_3 et p_4 ;
— le ballon 4 mesure les propriétés p_1 et p_4.

1. (⋆⋆) Proposer une modélisation du problème en terme de flot en précisant les sommets et les arêtes du réseau, ainsi que les capacités des arêtes. Illustrer la modélisation sur l'exemple précédent. Quel algorithme vu en cours permet alors de résoudre le problème en temps polynomial ?

Solution : Modélisons le problème avec un réseau de flot maximum, dans lequel le flot représente les mesures qu'on effectue. Le réseau contient un sommet par ballon $b_1, b_2, \ldots, b_m$, un sommet par propriété $p_1, p_2, \ldots, p_n$ ainsi qu'une source s et un puits t. On relie la source à chaque ballon avec un arc de capacité 2 modélisant la contrainte que chaque ballon ne peut servir qu'à mesurer deux propriétés. On relie chaque propriété au puits avec un arc de capacité k modélisant la contrainte qu'on souhaite mesurer chaque propriété au moins (et donc exactement, inutile de faire du zèle) k fois. Finalement, on ajoute les arcs (b_i, p_j) si le ballon b_i est équipé d'un capteur pour la propriété p_j : tous ces arcs ont capacité 1. On résout alors le problème de st-flot maximum sur ce réseau en temps polynomial à l'aide de l'algorithme d'Edmonds-Karp : le problème de mesures admet une solution si et seulement si la valeur du st-flot maximum renvoyé par l'algorithme est kn, voulant dire que la st-coupe minimum sépare le puits des autres sommets et donc que chaque propriété est bien mesurée par deux ballons. Sur l'exemple précédent, le réseau de flot est le suivant (dans lequel on a omis les capacités 1 sur les arcs du milieu pour la lisibilité) :

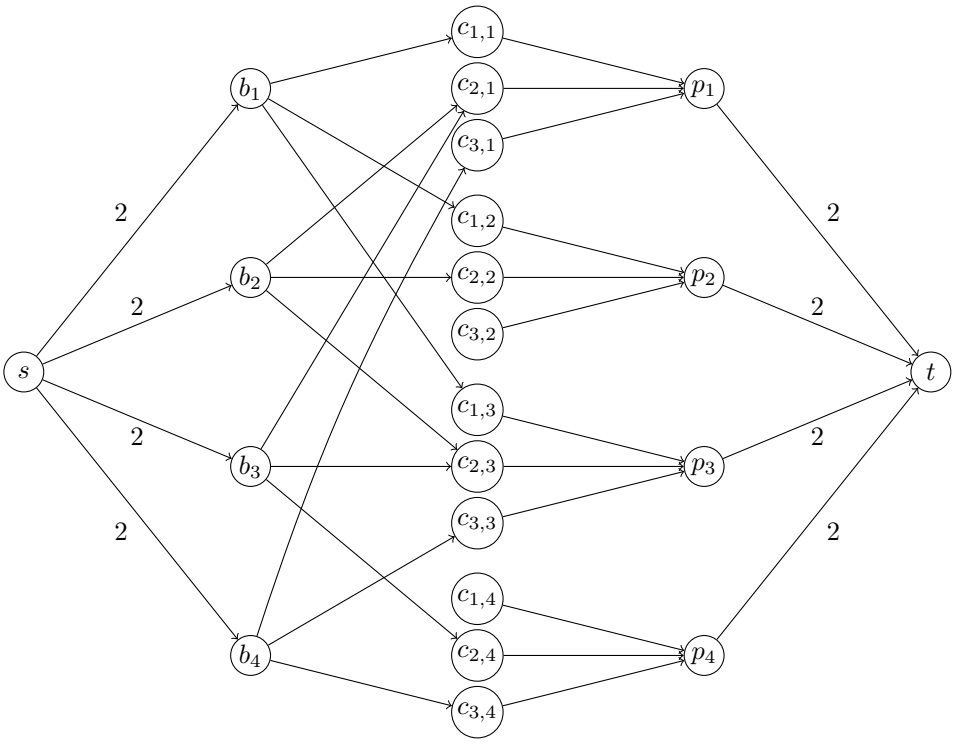


Vous montrez la solution que vous avez obtenue à l'aide de votre algorithme à votre supérieur hiérarchique, mais il ne la trouve pas satisfaisante. En effet, le cahier des charges ne mentionnait pas une requête supplémentaire que la répartition des ballons doit satisfaire. Chaque ballon est construit par une des trois sociétés qui construisent de tels ballons atmosphériques. La requête supplémentaire à satisfaire est de faire en sorte qu'aucune propriété n'est mesurée uniquement (c'est-à-dire les k mesures) par des ballons du même constructeur.

Dans l'exemple précédent, supposons que le ballon 1 provienne du premier constructeur, les ballons 2 et 3 du second constructeur et le ballon 4 du troisième constructeur. Alors, la répartition proposée n'est plus correcte puisque la propriété p_3 n'est réalisée que par des ballons du second constructeur. Cependant, on pourrait utiliser les ballons 1 et 2 pour mesurer les propriétés p_1 et p_2, et les ballons 3 et 4 pour mesurer les propriétés p_3 et p_4.

2. ($\star\star\star$) Expliquer comment modifier votre modélisation afin de trouver un nouvel algorithme en temps polynomial pour résoudre le problème.

Solution : On ajoute dans le réseau des sommets $c_{\ell,j}$ représentant le fait que le constructeur $\ell \in \{1,2,3\}$ soit utilisé pour mesurer la propriété p_j. On ajoute des arcs $(b_i, c_{\ell,j})$ si le ballon b_i est fabriqué par le constructeur ℓ et permet de mesurer la propriété p_j, de capacité 1. On ajoute également des arcs $(c_{\ell,j}, p_j)$ de capacité $k-1$ modélisant le fait que chaque propriété doit être mesurée au plus $k-1$ fois par le même constructeur. On pose la même question qu'avant et on vérifie qu'il existe un st-flot de valeur kn si et seulement si le problème admet une solution.
Sur l'exemple précédent, voici le nouveau réseau de flot :



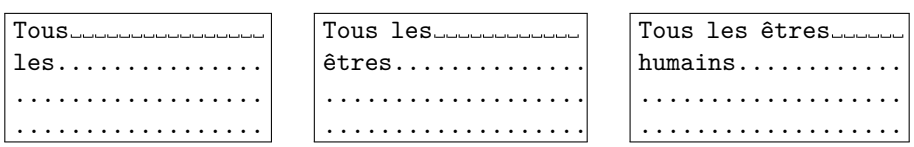
4 Formatage équilibré d'un paragraphe

Intéressons-nous au problème du formatage équilibré d'un paragraphe. On se donne ainsi une suite de n mots $m_1, \ldots, m_n$ de longueurs $\ell_1, \ell_2, \ldots, \ell_n$ et on souhaite imprimer ce paragraphe de manière équilibrée sur des lignes ne pouvant contenir qu'un maximum de M caractères chacune : on supposera donc que $\ell_i \leq M$ pour tout i. Le critère d'équilibre qu'on choisit est le suivant : si une ligne contient les mots de m_i à m_j inclus et qu'on laisse un caractère d'espacement entre chaque mot, le nombre de caractères d'espacements supplémentaires à la fin de la ligne est

$$f(i,j) = M - (j-i) - \sum_{k=i}^{j} \ell_k$$

L'objectif est de minimiser la somme des cubes des nombres de caractères d'espacement présents à la fin de chaque ligne, hormis la dernière ligne (qui peut donc être presque vide si nécessaire). On appelle *déséquilibre de la ligne* la quantité $f(i,j)^3$.

Par exemple, on a trois possibilités pour formater la première ligne de la Déclaration universelle des droits de l'homme, en lignes de 20 caractères (puisque le mot suivant est trop long pour pouvoir tenir sur celle-ci) :



Dans le premier cas, le déséquilibre de la première ligne est $f(1,1)^3 = (M-\ell_1)^3 = 16^3$; dans le second cas, il est égal à $f(1,2)^3 = (M-1-\ell_1-\ell_2)^3 = 12^3$; dans le troisième cas, il est égal à $f(1,3)^3 = (M-2-\ell_1-\ell_2-\ell_3)^3 = 6^3$.

Un premier algorithme de formatage équilibré de paragraphe est la stratégie gloutonne consistant à remplir les lignes les unes après les autres en mettant à chaque fois le plus de mots possibles sur la ligne en cours.

1. (⋆) Montrer sur un exemple simple que la stratégie gloutonne ne produit pas nécessairement le formatage optimal.

Solution : Considérons $M = 10$ et $\ell_1 = 6$, $\ell_2 = 3$, $\ell_3 = 4$ et $\ell_4 = 6$. Alors l'algorithme glouton produit le paragraphe comprenant les mots m_1 et m_2 sur la première ligne (sans espace en fin de ligne), puis le mot m_3 sur la seconde ligne (avec 6 espaces en fin de ligne) et le mot m_4 sur la dernière ligne. Son déséquilibre est de $6^3 = 216$.
On peut trouver un paragraphe moins déséquilibré en plaçant le mot m_1 seul sur la première ligne (avec 4 espaces en fin de ligne), les mots m_2 et m_3 sur la seconde ligne (avec 2 espaces en fin de ligne) et le mot m_4 sur la dernière ligne. Son déséquilibre est $4^3 + 2^3 = 72 < 216$. L'algorithme glouton n'est donc pas optimal.

On applique désormais un algorithme de programmation dynamique. On note donc $d(j)$ la valeur minimale du déséquilibre total occasionné par le formatage du paragraphe des mots $m_1, \ldots, m_j$.

2. (⋆⋆) Trouver une formule récursive pour calculer $d(j)$.

Solution : On note $\alpha(j)$ le plus petit entier i pour lequel $\ell_i + \ell_{i+1} + \cdots + \ell_j + j - i \leq M$: $\alpha(i)$ est l'indice j minimal pour lequel on peut mettre sur la même ligne les mots d'indice $i, i+1, \ldots, j$. On considère chacun de ses indices $i \in \{\alpha(j), \alpha(j)+1, \ldots, j\}$ et on mesure le déséquilibre total par récurrence, en n'oubliant pas le cas de base où tout le paragraphe peut tenir sur la dernière ligne de déséquilibre nul :

$$d(j) = \begin{cases} 0 & \text{si } \alpha(j) = 1 \\ \min\limits_{\alpha(j) \leq i \leq j} \left(f(i,j)^3 + d(i-1) \right) & \text{sinon} \end{cases}$$

3. (⋆⋆) En déduire un algorithme de programmation dynamique calculant le déséquilibre minimal qu'on puisse obtenir pour une valeur de M et des longueurs $\ell_1, \ldots, \ell_n$ données (qu'on suppose toutes inférieures à M). On supposera écrite une fonction calculant $f(i,j)$ pour un tableau de longueurs de mots et des indices i et j donnés.

Solution : On calcule donc les valeurs de $d(j)$ successivement par valeur de j croissante :

```
1   fonction formatage(tableau d'entiers l, entier M) : entier =
2     soit n := longueur(l)
3     soit d := tableau[1..n] d'entiers
4     d[1] ← 0
5     pour tout j de 2 à n faire
6       soit i := j et L := l[j] et d[j] ← f(j,j)^3 + d[j − 1]
7       tant que i > 1 et L ≤ M faire
8         i := i − 1
9         L := L + l[i]
10        d[j] ← min(d[j], f(i,j)^3 + d[i − 1])
11      si i = 1 alors d[j] ← 0
12    retourner d[n]
```

4. (⋆) Quelle est la complexité de l'algorithme précédent ?

Solution : On remplit chaque case du tableau `d`, le calcul d'une case nécessitant dans le pire cas de considérer tous les mots précédents. Au total, on est donc en $O(n^2)$.

5 Élimination de boules

On se donne une séquence de n boules colorées. On peut éliminer une boule de la séquence, au prix d'un euro. On peut aussi éliminer deux boules voisines gratuitement si elles ont la même couleur. L'objectif est de calculer le prix minimum à payer pour éliminer toutes les boules de la séquence.

Par exemple, en partant de la séquence de boules $RBBVRBVBR$ (rouge, bleu et vert), si on élimine la première boule verte, puis les deux boules rouges voisines, puis deux des trois boules bleues voisines, puis la boule verte, puis les deux boules bleues voisines, puis les deux boules rouges voisines, on a la suite d'élimination suivante :

$$RBBRVRBVBR \longrightarrow RBBRRBVBR \longrightarrow RBBBVBR \longrightarrow RBVBR \longrightarrow RBBR \longrightarrow RR \longrightarrow \varepsilon$$

Son prix total est de 2 euros.

Notons $c(i,j)$ le cout minimal d'élimination des boules comprises entre la boule d'indice i et la boule d'indice j (incluses). On admet en ce début d'exercice (cf question 5) que **si les boules i et j ont la même couleur, alors** $c(i,j) = c(i+1, j-1)$.

1. ($\star\star$) Donner une formule récursive permettant de calculer $c(i,j)$.

Solution : On note $\kappa(i)$ la couleur de la boule i.
Il faut connaître la valeur de $c(i,j)$ lorsque les boules i et j n'ont pas la même couleur. Lorsque $i = j$, il faut nécessairement payer un euro pour éliminer cette dernière boule. Sinon, on est sûr que i et j sont deux boules qu'on ne peut pas éliminer en les rendant voisines : on peut donc subdiviser le problème en deux sous-problèmes, en les séparant à l'aide d'un indice k quelconque dans $\{i, i+1, \ldots, j-1\}$. Cela fournit la formule de récurrence suivante, où on n'oublie pas le cas de base où $i = j$ et même $i > j$ au cas où on n'en ait besoin dans les appels récursifs (vu le $c(i+1, j-1)$ on aura besoin du cas où $i = j-1$ typiquement) :

$$c(i,j) = \begin{cases} 0 & \text{si } i > j \\ 1 & \text{si } i = j \\ c(i+1, j-1) & \text{si } \kappa(i) = \kappa(j) \\ \min\limits_{i \leq k < j} \big(c(i,k) + c(k+1, j)\big) & \text{sinon} \end{cases}$$

2. ($\star\star$) Écrire le pseudocode de l'algorithme pour résoudre le problème initial renvoyant le coût optimal d'élimination de la séquence de boules.

Solution : On remplit donc la matrice ligne par ligne de bas en haut, chaque ligne étant remplie de gauche à droite en commençant par le coefficient diagonal :

```
1   fonction élimination(tableau de couleurs couleur[1..n]) : entier =
2     soit c = tableau[1..n][1..n] d'entiers initialisés à 0
3     pour i de n à 1 avec un pas de −1 faire
4       c[i][i] ← 1
6       pour j de i + 1 à n faire
8         si couleur[i] = couleur[j] alors c[i][j] ← c[i + 1][j − 1]
9         sinon
10          c[i][j] ← 1 + c[i + 1][j]
11          pour k de i + 1 à j − 1 faire
12            si couleur[i] = couleur[k] alors
13              c[i][j] ← min(c[i][j], c[i + 1][k − 1] + c[k + 1][j])
14    retourner c[1][n]
```

3. ($\star$) Quelle est sa complexité ?

Solution : Remplir le coefficient (i,j) de la matrice nécessite un nombre d'opérations élémentaires linéaire en n. Ainsi, cet algorithme a une complexité en $O(n^3)$.

4. ($\star$) Utiliser l'algorithme pour trouver l'optimum pour la séquence $RBBVRBVBR$.

Solution : On remplit diagonale par diagonale la matrice de coût suivante, pour trouver que l'optimum de la séquence vaut 3 :

	R	B	B	V	R	B	V	B	R
R	1	2	1	2	1	2	3	2	3
B	0	1	0	1	2	3	2	3	2
B	0	0	1	2	3	2	3	2	3
V	0	0	0	1	2	3	2	3	2
R	0	0	0	0	1	2	3	2	1
B	0	0	0	0	0	1	2	1	2
V	0	0	0	0	0	0	1	2	3
B	0	0	0	0	0	0	0	1	2
R	0	0	0	0	0	0	0	0	1

5. ($\star\star\star$) Montrer la propriété de c admise en début d'exercice, à savoir que si les boules i et j ont la même couleur, alors $c(i,j) = c(i+1, j-1)$. *Indication : on pourra montrer qu'il existe toujours une stratégie d'élimination optimale des boules i à j qui rend voisines les boules i et j pour les éliminer gratuitement.*

Solution : Si les boules i et j ont la même couleur, alors on obtient une stratégie d'élimination de coût $c(i+1, j-1)$ en éliminant d'abord toutes les boules entre $i+1$ et $j-1$, avant d'éliminer gratuitement les deux boules i et j qui sont devenues voisines. Ainsi $c(i,j) \leq c(i+1, j-1)$.
Réciproquement, montrons que $c(i+1, j-1) \leq c(i,j)$, c'est-à-dire qu'il existe une stratégie d'élimination optimale des boules i à j qui commence par éliminer les boules $i+1$ à $j-1$, pour rendre voisines les boules i et j, et ainsi les éliminer gratuitement à la fin. Considérons donc une stratégie d'élimination quelconque des boules i à j de coût $c(i,j)$.

- Si les boules i et j y sont toutes deux éliminées en payant un euro, on obtient une stratégie pour éliminer les boules $i+1$ à $j-1$ de coût $c(i,j)-2$, d'où $c(i+1, j-1) \leq c(i,j) - 2$ ce qui contredit l'inégalité prouvée avant.
- Si une des deux boules, disons i, est éliminée en payant un euro, mais que la boule j est éliminée gratuitement à l'aide de la boule $k \in \{i+1, \ldots, j-1\}$, alors les boules i, k et j sont de même couleur, donc on obtient une solution de même coût en éliminant la boule k en payant un euro, puis en finissant par éliminer les boules i et j rendues voisines.
- Si les deux boules sont éliminées gratuitement, la boule i avec une boule $k \in \{i+1, \ldots, j-1\}$ et la boule j avec une boule $\ell \in \{k+1, \ldots, j-1\}$, les quatre boules ont la même couleur. On obtient donc une solution de même coût en éliminant les boules k et ℓ en les ayant rendues voisines, puis en éliminant in fine les boules i et j rendues voisines.

On a donc obtenu une stratégie d'élimination optimale des boules i à j finissant par éliminer les boules i et j rendues voisines. Cela fournit donc une stratégie d'élimination des boules $i+1$ à $j-1$ de coût $c(i,j)$, d'où $c(i+1, j-1) \leq c(i,j)$.