CONCOURS CENTRALE•SUPÉLEC

Option informatique

MP

4 heures Calculatrice autorisée

2020

Un système de vote

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 librairie standard (celles qui s'écrivent sans nom de module, comme `max` ou `incr` ainsi que les opérateurs comme `@`) peuvent être librement utilisées. Les candidats ne devront faire appel à aucun autre module.

En Caml, les matrices d'entiers sont représentées par des tableaux de tableaux, c'est-à-dire par le type `int array array`. L'expression `m.(i).(j)` permet d'accéder au coefficient de la `i`-ième ligne et de la `j`-ième colonne de la matrice `m`. Dans le texte, en dehors du code Caml, ce coefficient sera noté $m[i,j]$. On rappelle les définitions suivantes :

— `Array.make_matrix : int -> int -> 'a -> 'a array array` est telle que `Array.make_matrix n p v` renvoie une matrice de `n` lignes et `p` colonnes dont toutes les cases contiennent la valeur `v` ;

— `Array.length : 'a array -> int` est telle que `Array.length tab` renvoie le nombre d'éléments du tableau `tab`. Si `tab` est une matrice, c'est le nombre de lignes de `tab` ;

— `min : 'a -> 'a -> 'a` renvoie le minimum des deux valeurs en argument ;

— `max : 'a -> 'a -> 'a` renvoie le maximum des deux valeurs en argument ;

— l'opérateur `@` concatène deux listes. Par exemple, `[2; 0] @ [4; 2]` renvoie `[2; 0; 4; 2]`.

Dans ce problème, les graphes ont un ensemble de sommets de la forme $\{0, 1, ..., n-1\}$ et sont orientés complets et pondérés par des entiers : pour tout couple (i, j) de sommets distincts, il existe un arc de i à j de poids $m_{i,j} \in \mathbb{Z}$. Le graphe est représenté par sa matrice d'adjacence $M = (m_{i,j})_{0 \leqslant i,j \leqslant n-1}$, avec la convention que $m_{i,i} = 0$ pour tout sommet i (il n'y a pas d'arc d'un sommet vers lui-même).

I Vote par préférence

Nous considérons une élection, à laquelle se présentent n candidats. Chaque électeur inscrit sur son bulletin l'ensemble des candidats (tous les candidats sont classés), par ordre de préférence. L'ensemble des bulletins est rassemblé dans une urne. Le nombre de votants est noté p, l'urne contient donc p bulletins.

Les trois types de données suivants sont utilisés pour représenter un vote :

— `type candidat = int ;;` chaque candidat est désigné par un numéro de 0 à $n-1$;

— `type bulletin = candidat list ;;` un bulletin de vote est une liste (ordonnée) de candidats ;

— `type urne = bulletin list ;;` une urne est un ensemble de bulletins de vote.

Par exemple, s'il y a trois candidats et qu'un électeur préfère le candidat 2, puis le candidat 0 et enfin considère que le candidat 1 est le moins souhaitable, son bulletin de vote sera `[2; 0; 1]`.

Une fois le vote effectué, on compare les résultats de deux candidats particuliers i et j en comptant le nombre de bulletins qui classent le candidat i avant le candidat j. On exprime le résultat de la comparaison entre les candidats i et j en calculant la différence entre le nombre de bulletins de vote qui placent i avant j et le nombre de ceux qui placent j avant i.

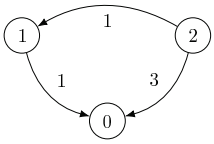
I.A – Premier exemple

On considère une élection avec trois candidat et trois votants : $n = 3$, $p = 3$. Les bulletins sont `[2; 0; 1]`, `[2; 1; 0]` et `[1; 2; 0]`. La comparaison entre le candidat numéro 0 et le candidat numéro 1 donne -1 car le candidat 0 est placé une fois avant le candidat 1 et deux fois après.

Q 1. Écrire une fonction `duel : candidat -> candidat -> urne -> int`, telle que `duel i j u` renvoie la comparaison entre le candidat `i` et le candidat `j` à partir des bulletins contenus dans l'urne `u`.

On peut alors synthétiser le contenu d'une urne `u` en construisant le *graphe de préférence* des votants : ses sommets correspondent aux candidats et l'arc du sommet `i` vers le sommet `j` est pondéré par la comparaison entre le candidat `i` et le candidat `j`, c'est-à-dire la valeur de `duel i j u`.

La figure 1 donne le graphe de préférence obtenu à partir du vote de l'exemple 1 ainsi que sa matrice d'adjacence M. Afin d'alléger le schéma, seuls les arcs avec un poids strictement positif sont représentés.



$$M = \begin{pmatrix} 0 & -1 & -3 \\ 1 & 0 & -1 \\ 3 & 1 & 0 \end{pmatrix}$$

Figure 1

I.B – Deuxième exemple

On considère une élection avec trois candidats et 4 votants : $n = 3$, $p = 4$. À l'issue du vote, le contenu de l'urne est `[[0; 1; 2]; [1; 2; 0]; [0; 2; 1]; [0; 2; 1]]`.

Q 2. Tracer le graphe de préférence de l'urne (en ne dessinant que les arcs ayant un poids strictement positif) et donner sa matrice d'adjacence.

I.C – Construction du graphe de préférence

Q 3. Expliquer pourquoi la matrice d'adjacence d'un graphe de préférence est antisymétrique et pourquoi tous ses coefficients non-diagonaux ont la même parité.

Étant donné une urne U, on note $\mathrm{Mat}(U)$ la matrice du graphe de préférence associée à cette urne.

Q 4. Écrire une fonction `depouillement : int -> urne -> int array array` qui prend en paramètres le nombre de candidats n et une urne U et renvoie la matrice $\mathrm{Mat}(U)$.

I.D – Théorème de McGarvey

Le but de cette sous-partie est de démontrer le théorème de McGarvey : « Pour toute matrice antisymétrique à coefficients pairs M, il existe une urne U telle que $M = \mathrm{Mat}(U)$. »

Soient i, j et n trois entiers naturels tels que $i < n$, $j < n$ et $i \neq j$. On note $E_{i,j,n}$ la matrice carrée de taille n dont tous les coefficients sont nuls sauf les coefficients d'indices (i, j) et (j, i) qui valent respectivement 2 et -2.

Q 5. Soit n un nombre de candidats et i et j deux entiers naturels strictement inférieurs à n. Montrer qu'il existe une urne $U_{i,j,n}$ contenant deux votes telle que $\mathrm{Mat}(U_{i,j,n}) = E_{i,j,n}$.

Q 6. On considère deux urnes U_1 et U_2. Exprimez $M_3 = \mathrm{Mat}(U_1 \cup U_2)$ en fonction de $M_1 = \mathrm{Mat}(U_1)$ et de $M_2 = \mathrm{Mat}(U_2)$.

Q 7. Démontrer le théorème de McGarvey.

Q 8. Écrire une fonction `mcgarvey : int array array -> urne` prenant en paramètre une matrice antisymétrique M de coefficients tous pairs et renvoyant une urne U telle que $M = \mathrm{Mat}(U)$.

Q 9. Estimer la complexité de la fonction `mcgarvey` en fonction de n, la taille de la matrice (c'est-à-dire le nombre de candidats), et de q, le maximum des coefficients de la matrice.

II Recherche du vainqueur

L'objectif de cette partie est de déterminer le vainqueur, ou les vainqueurs *ex aequo*, d'un vote par préférence.

II.A – Vainqueur de Condorcet

On appelle *vainqueur de Condorcet* tout sommet tel que, dans le graphe de préférence, les arcs sortant de ce sommet ont tous un poids positif ou nul. Ainsi, dans le premier exemple de la partie I, le candidat 2 est un vainqueur de Condorcet.

Q 10. Expliquer pourquoi un vainqueur de Condorcet peut être qualifié de « vainqueur » de l'élection.

Q 11. Écrire une fonction `condorcet : int array array -> candidat list` prenant en paramètre la matrice d'adjacence d'un graphe de préférence et renvoyant la liste des vainqueurs de Condorcet.

Q 12. En se plaçant dans le cas $n = 3$ et $p = 3$, construire une urne pour laquelle il n'existe pas de vainqueur de Condorcet. Tracer le graphe de préférence correspondant.

II.B – Graphe intermédiaire de Schultze

À la fin du XX$^{\text{e}}$ siècle, Markus Schulze imagina une méthode permettant de déterminer un vainqueur à l'issue de n'importe quel vote par préférence.

On appelle *poids d'un chemin* du graphe de préférence, le **minimum** des poids des arcs constituant ce chemin. Ainsi, dans le premier exemple de la partie I, le poids du chemin $2 \to 0 \to 1$ est -1.

Le *graphe intermédiaire de Schulze* est un graphe orienté pondéré complet dont les sommets sont les candidats et dont l'arc du sommet i vers le sommet j (distinct de i) est pondéré par le **maximum** des poids de tous les chemins allant de i à j dans le graphe de préférence. Sa matrice d'adjacence est notée I.

Q 13. Pour i et j, candidats distincts, démontrer que $I[i,j]$ est aussi le maximum des poids des chemins sans boucle de i à j, c'est-à-dire des chemins $i = i_0 \to i_1 \to \cdots \to i_k = j$ avec $i_0, i_1, \ldots, i_k$ deux à deux distincts.

Q 14. Démontrer que $\min(I[i,j], I[j,k]) \leqslant I[i,k]$ pour tout triplet (i,j,k) de sommets distincts.

Q 15. En adaptant l'algorithme de Floyd-Warshall, programmer une fonction `intermediaire : int array array -> int array array` prenant en paramètre la matrice d'adjacence d'un graphe de préférence et renvoyant la matrice d'adjacence du graphe intermédiaire de Schulze correspondant.

Q 16. Estimez la complexité de la fonction `intermediaire` en fonction de n, le nombre de candidats.

Q 17. Serait-il pertinent d'utiliser l'algorithme de Dijkstra au lieu de l'algorithme de Floyd-Warshall ? Argumenter la réponse.

II.C – Graphe de préférence de Schulze

Le *graphe de préférence de Schulze* est défini à partir du graphe intermédiaire de Schulze. Si I est la matrice d'adjacence du graphe intermédiaire de Schulze, alors la matrice d'adjacence S du graphe de préférence de Schulze est définie par $S[i,j] = I[i,j] - I[j,i]$ pour tous entiers naturels i et j strictement inférieurs à n.

Q 18. Montrer que la matrice d'adjacence d'un graphe de préférence de Schulze est antisymétrique et que tous ses coefficients sont pairs.

Q 19. Écrire une fonction `graphe_schulze : int array array -> int array array` qui prend en paramètre un graphe intermédiaire de Schulze et qui renvoie le graphe de préférence de Schulze correspondant.

II.D – Vainqueur de Schulze

Un *vainqueur de Schulze* est un vainqueur de Condorcet dans le graphe de préférence de Schulze.

Q 20. Écrire une fonction `schulze : int -> urne -> candidat list` qui prend en paramètres le nombre de candidats et un ensemble de bulletins de vote et renvoie la liste des vainqueurs de Schulze.

Q 21. Estimer la complexité de la fonction `schulze` en fonction du nombre de candidats n et du nombre de votants p.

À partir d'un graphe de préférence de Schulze représenté par la matrice S, on définit la relation R_S entre les candidats comme suit : $i \, R_S \, j$ si et seulement $S[i,j]$ est strictement positif.

Q 22. Montrer que la relation R_S est transitive, c'est-à-dire que pour tous candidats i, j et k, si $i \, R_S \, j$ et $j \, R_S \, k$ alors $i \, R_S \, k$.

Si I désigne la matrice d'adjacence du graphe intermédiaire, on pourra distinguer les cas $I[i,j] \leqslant I[j,k]$ et $I[i,j] > I[j,k]$.

Q 23. Montrer que quelle que soit l'urne non vide considérée, il existe toujours au moins un vainqueur de Schulze.

III Satisfiabilité d'une formule de logique propositionnelle

Étant donné une variable propositionnelle x, on appelle *littéral*, les formules x et $\neg x$ ($\neg x$ est la négation de x).

On appelle *clause* une disjonction de littéraux, par exemple $x \vee \neg y \vee z$ est une clause ($\vee$ signifie « ou »).

On appelle *conjonction de clauses*, une formule qui est la conjonction entre plusieurs clauses. Par exemple $(x \vee \neg y \vee z) \wedge (x \vee \neg y)$ est une conjonction de clauses ($\wedge$ signifie « et »).

On appelle *interprétation* une fonction qui à chaque variable propositionnelle associe une valeur de vérité (vrai ou faux).

On dit qu'une conjonction de clauses est *satisfiable* s'il existe une interprétation qui la rend vraie, c'est-à-dire qui rend vrai toutes ses clauses, autrement dit qui rend vrai au moins un littéral de chacune de ses clauses.

Le problème SAT consiste à déterminer si une conjonction de clauses est satisfiable :

Entrées : Une conjonction de clauses φ.
Sortie : Un booléen. Vrai si φ est satisfiable. Faux sinon.

Q 24. Montrer qu'il est possible de résoudre le problème SAT avec une complexité en $O(2^n m)$ où n est le nombres de variables distinctes et m le nombre de littéraux de l'entrée (comptés avec leurs répétitions).

On considère un vote par préférence pour lequel on a réparti les candidats en trois groupes :

— un candidat c, appelé *champion* ;

— un ensemble A de candidats dits *automatiques* ;

— un ensemble B de candidats *optionnels*.

Le problème appelé CONTROL-ADD-ALT consiste à déterminer s'il est possible d'éliminer un certain nombre de candidats optionnels de façon à ce que c soit un vainqueur de Schulze de l'élection :

Entrées : Le candidat c. L'ensemble A. L'ensemble B. Le graphe de préférence G du vote sur l'ensemble des candidats $\{c\} \cup A \cup B$. Un budget $k \in \mathbb{N}$.
Sortie : Un booléen. Vrai, s'il est possible de choisir k candidats distincts $b_1, ..., b_k$ dans B tels que c est un vainqueur de Schulze de l'élection en considérant uniquement l'ensemble de candidats $\{c\} \cup A \cup \{b_1, ..., b_k\}$. Faux sinon.

Pour obtenir le graphe de préférence de l'élection avec l'ensemble de candidats $\{c\} \cup A \cup \{b_1, ..., b_k\}$, on prend le graphe de préférence sur l'ensemble des candidats $\{c\} \cup A \cup B$ et on supprime les candidats de B qui n'ont pas été choisis (ainsi que les arrêtes entrantes et sortantes de ces candidats).

On appelle *instance* de CONTROL-ADD-ALT une entrée du problème CONTROL-ADD-ALT et instance de SAT une entrée du problème SAT.

On considère l'algorithme « Transformation d'une formule en élection » qui, étant donné une instance de SAT (c'est-à-dire une conjonction de clauses), construit une instance de CONTROL-ADD-ALT :

Entrées : Une conjonction de clauses φ
Sortie : Un candidat c (le champion), un ensemble de candidats automatiques A, un ensemble de candidats optionnels B, un budget k, un graphe de préférence G sur l'ensemble de candidats $\{c\} \cup A \cup B$
$\psi \leftarrow \varphi$
pour tout x variable propositionnelle apparaissant dans φ **faire**
    $\psi \leftarrow \psi \wedge (x \vee \neg x)$
fin pour
Créer un nouveau candidat c
Créer un graphe G avec un seul sommet c
$A \leftarrow \emptyset$, $B \leftarrow \emptyset$
pour tout cl clause de ψ **faire**
    Créer un nouveau candidat a_{cl} associé à cl et l'ajouter à G et à A
    Ajouter à G un arc de poids $+4$ de a_{cl} vers c et un arc de poids -4 en sens inverse.
fin pour
pour tout x variable propositionnelle apparaissant dans ψ **faire**
    Créer un nouveau candidat b_x associé au littéral x et l'ajouter à G et à B
    Créer un nouveau candidat $b_{\neg x}$ associé au littéral $\neg x$ et l'ajouter à G et à B
    Ajouter à G deux arcs de poids $+6$ de c vers b_x et vers $b_{\neg x}$ puis ajouter deux arcs de poids -6 en sens inverse.
fin pour
pour tout couple (li, cl) avec li un littéral de ψ et cl une clause de ψ contenant le littéral li **faire**
    Ajouter à G un arc de poids $+6$ de b_{li} vers a_{cl} et un arc de poids -6 en sens inverse.
fin pour
Compléter le graphe G avec des arrêtes de poids nul pour le rendre complet.
k ←le nombre de variables propositionelles qui apparaissent dans ψ.
renvoyer c, A, B, G, k.

Q 25. Donner le graphe de préférence créé à partir de la formule $(x \vee x) \wedge (\neg x \vee \neg x)$. Est-il possible de choisir $k = 1$ candidat parmi x et $\neg x$ de sorte que c gagne ?

Q 26. Montrer que si on peut choisir k candidats optionnels de sorte à faire gagner le champion, alors, pour toute variable propositionnelle x, on a choisi x ou $\neg x$ mais on n'a pas choisit x et $\neg x$.

Q 27. Justifier que φ est satisfiable si et seulement si l'instance de CONTROL-ADD-ALT créée par cet algorithme à partir de φ donne une réponse positive.

On dit qu'un algorithme est en temps polynomial en un paramètre m, si la complexité de cet algorithme est majorée par un polynôme en m.

Conjecture. Il n'existe pas d'algorithme en temps polynomial en nombre de littéraux de l'entrée qui résout SAT.

Q 28. Montrer que, si la conjecture précédente est vraie, alors il n'existe pas d'algorithme en temps polynomial en nombre de candidats qui résout CONTROL-ADD-ALT.

• • • FIN • • •