



## 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  $i$ -ème 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  $\{0, 1, \dots, n-1\}$  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  $M = (m_{i,j})_{0 \leq i, j \leq n-1}$ , 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  $i$ .

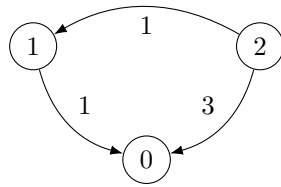
### I.A – Premier exemple

On considère une élection avec trois candidats 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  $\text{Mat}(U)$  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  $\text{Mat}(U)$ .

### 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  $M = \text{Mat}(U)$ . »

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  $\text{Mat}(U_{i,j,n}) = E_{i,j,n}$ .

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

**Q 7.** Démontrer le théorème de McGarvey.

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

**Q 9.** Estimer la complexité de la fonction `mccgarvey` 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 Schulze

À la fin du  $xx^e$  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  $2 \rightarrow 0 \rightarrow 1$  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  $i = i_0 \rightarrow i_1 \rightarrow \dots \rightarrow i_k = j$  avec  $i_0, i_1, \dots, i_k$  deux à deux distincts.
- Q 14.** Démontrer que  $\min(I[i, j], I[j, k]) \leq I[i, k]$  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` 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 si  $S[i, j]$  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  $I[i, j] \leq I[j, k]$  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  $\varphi$ .

**Sortie :** Un booléen. Vrai si  $\varphi$  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 nombre 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  $b_1, \dots, b_k$  dans  $B$  tels que  $c$  est un vainqueur de Schulze de l'élection en considérant uniquement l'ensemble de candidats  $\{c\} \cup A \cup \{b_1, \dots, b_k\}$ . Faux sinon.

Pour obtenir le graphe de préférence de l'élection avec l'ensemble de candidats  $\{c\} \cup A \cup \{b_1, \dots, b_k\}$ , 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  $\varphi$

**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  $\varphi$  **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  $\psi$  **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  $\psi$  **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  $\psi$  et  $cl$  une clause de  $\psi$  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.

$k \leftarrow$  le nombre de variables propositionnelles qui apparaissent dans  $\psi$ .

**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 choisi  $x$  et  $\neg x$ .

**Q 27.** Justifier que  $\varphi$  est satisfiable si et seulement si l'instance de CONTROL-ADD-ALT créée par cet algorithme à partir de  $\varphi$  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 • • •

---

# Option informatique

## Présentation du sujet

Le sujet porte sur des systèmes de vote à effectuer lors d'une élection. La première partie traite du vote par préférence et évoque le théorème de McGarvey sur sa relation avec les matrices. La deuxième partie présente les vainqueurs de Condorcet, ainsi que la méthode de sélection d'un vainqueur de Schulze. La dernière partie présente un problème de décision (`Control-Alt-Add`) et propose de montrer sa NP-complétude.

Les candidats ont bien abordé l'ensemble du problème et les meilleurs d'entre eux ont traité toutes les questions.

## Analyse globale des résultats

Le sujet a globalement été bien compris et le jury est globalement satisfait d'une majorité des candidats. Les questions de programmation sont plutôt bien traitées, mais comme chaque année, nous constatons des candidats qui ont une maîtrise très superficielle de la syntaxe `CamL`, la mélangent avec celle de `Python`, et ne comprennent pas les différences fondamentales entre ces deux langages.

Certaines questions théoriques ont perturbé une partie des candidats de par leur composante mathématique forte. De même, la dernière partie sur la NP-complétude en a déstabilisé plus d'un.

Cette année, des consignes sans ambiguïté étaient présentées en début de sujet sur les fonctions informatiques autorisées pour la composition. Certains candidats ont toutefois réécrit des fonctions déjà existantes dans les bibliothèques autorisées, soit par ignorance de ces fonctions, soit car ils n'avaient pas lu correctement les consignes.

Malgré ces remarques, il reste beaucoup de candidats qui rendent des copies exemplaires, avec des justifications détaillées pour expliquer le rôle des fonctions auxiliaires qui interviennent et le fonctionnement global des algorithmes mis en place.

## Commentaires sur les réponses apportées et conseils aux futurs candidats

De manière non exhaustive, nous citons des erreurs fréquentes rencontrées dans les copies :

- l'oubli que les variables dans les filtrages sont muettes et que l'usage du `when` est parfois nécessaire. Par exemple, `match x with y -> ...` doit souvent être remplacé par une syntaxe de la forme `match x with _ when x = y -> ...` ;
- l'utilisation maladroite des références, voire leur oubli total. Nous constatons une absence régulière du `!`, ou l'utilisation de `=` au lieu de `:=`. Nous rappelons qu'une liste est un objet non modifiable. Ainsi, le code suivant

```
let l = [] in
for i = 0 to n - 1 do
  l @ [i]
done
```

ne fait absolument rien : une nouvelle liste est créée à chaque passage dans la boucle, puis oubliée au passage suivant ;

- la confusion perpétuelle entre listes et tableaux, dans un sens comme dans l'autre, même en fin de deuxième année de classe préparatoire. Si  $u$  est une liste,  $u.(i)$  n'a aucun sens. De même, la commande `List.make` n'existe pas. Une lecture attentive de l'énoncé peut souvent permettre à ceux qui connaissent ces différences d'éviter de perdre des points.

Par ailleurs, les questions de complexité sont parfois mal justifiées, et en particulier, les candidats oublient que la complexité temporelle de  $u @ v$  n'est pas nécessairement la même que celle de  $v @ u$ . Nous rappelons qu'en utilisant la notation  $\mathcal{O}$ , il est largement préférable de conclure qu'une complexité est un  $\mathcal{O}(n^2p)$  plutôt qu'un  $\mathcal{O}(\frac{1}{2}n(n-1)p)$ . Enfin, il est nécessaire de comprendre que la complexité d'un appel à `f (g x)` n'est pas le produit, mais la *somme* des complexités de `f` et `g`.

Aussi, les candidats qui écrivent plusieurs fonctions auxiliaires appelées `aux1`, `aux2`, etc. ne peuvent se le permettre que s'ils apportent des explications claires sur le rôle de ces fonctions et de leurs arguments. Les membres de jury ne sont pas (encore ?) des ordinateurs qui compilent automatiquement le code qu'ils lisent pour le comprendre.

Si la programmation récursive et la programmation itérative sont interchangeable, certaines fonctions sont plus facile à écrire et à lire lorsqu'elle le sont avec des boucles. Certains candidats ont mal écrit des fonctions récursives, pénibles à relire et avec des cas d'arrêts erronés, là où une double boucle `for` faisait tout aussi bien le travail. À ce sujet, nous rappelons que dans la syntaxe `for i = a to b do`, les bornes `a` et `b` sont toutes les deux atteintes.

Pour en terminer avec les remarques générales, il y a un réel effort de présentation pour la très grande majorité des copies. Cela rend d'autant plus visible les quelques copies rendues quasiment sous forme de brouillon (résultats soulignés ou encadrés sans règle, ratures omniprésentes, corrections écrites en petit et en travers au milieu des ratures, etc.). De tels candidats ont bien entendu été pénalisés. Par ailleurs, nous rappelons aux candidats que l'usage d'un français correct est de rigueur : respect de l'orthographe, des règles de grammaire et de conjugaison. Enfin, les abréviations et le style télégraphique sont à proscrire.

Il y a eu quelques incompréhensions sur la question **Q1** du sujet, car plusieurs candidats ont compris qu'il fallait écrire la fonction dans le cas de l'exemple qui était donné (la sous-partie en question s'appelle effectivement « Premier exemple »). À cette question, plusieurs candidats ont considéré que  $u$  était en fait une matrice et non une liste de listes. Le cas des coefficients diagonaux a souvent été omis à la question **Q4**, alors que la fonction `duel` ne traitait pas de manière particulière le cas d'égalité entre  $i$  et  $j$ . Nous avons souvent vu des preuves utilisant des bases d'espaces vectoriels pour la question **Q7**, mais rappelons que  $\mathbb{Z}$  n'est pas un corps ! Plusieurs problèmes de complexité ont été rencontrés à la question **Q8**. Le jury rappelle que même si aucune complexité n'est imposée à une question, il est recommandé d'écrire une fonction avec la meilleure complexité possible tant que cela ne nécessite pas une réécriture complète de l'algorithme. Ainsi, on préférera `i :: 1 à 1 @ [i]` en termes de complexité.

La confusion entre listes et tableaux s'est à nouveau présentée à la question **Q11** où une part non négligeable a travaillé avec des tableaux de listes au lieu de matrices. Les questions **Q13**, **Q14** et **Q15** ont posé problème aux candidats qui n'ont pas réussi à s'adapter au fait que la notion de distance entre les sommets n'était pas la notion usuelle. Nous rappelons également que l'algorithme de Floyd-Warshall est au programme de deuxième année et ceux qui ont montré qu'ils connaissaient leurs cours ont été récompensés. La question **Q22** était difficile, mais les candidats qui ont correctement utilisé l'indication fournie par l'énoncé ont réussi à justifier correctement la transitivité.

Si un grand nombre de candidats ont indiqué qu'il existait  $2^n$  distributions de vérité pour la question **Q24**, très peu ont justifié correctement qu'il était possible de les générer avec une complexité  $\mathcal{O}(2^n)$ . La justification que le test d'une distribution se faisait en  $\mathcal{O}(m)$  (qui était le nombre de littéraux et non le nombre de clauses) était également rarement présente. Les candidats qui, à la question **Q25**, ont tracé le

graphe en ne représentant que les arcs strictement positifs ont été appréciés, en comparaison aux schémas illisibles qui ont parfois été rencontrés. Les questions **Q26**, **Q27** et **Q28** ont été peu traitées par les candidats.

## **Conclusion**

Malgré des conditions de travail inhabituelles et difficiles pour cette fin de deuxième année, le jury est satisfait de voir qu'un grand nombre de candidats ont un niveau tout à fait satisfaisant en informatique, tant pour la rédaction des questions théoriques que l'écriture de programmes clairs et compréhensibles.

Nous sommes tout à fait conscients qu'il est difficile d'écrire du code sur papier, mais le jury recommande toutefois de s'entraîner sur ordinateur au cours de l'année pour acquérir les bons réflexes.