4.1.3 Conseils aux futurs candidats

Nous conseillons aux futurs candidats une lecture attentive du rapport du jury afin d'éviter les erreurs récurrentes dans les copies.

Nous conseillons de se familiariser avec le programme officiel d'informatique commune afin de vérifier la maîtrise des points exigibles aux concours.

4.2 Informatique option MP

4.2.1 Généralités

Le sujet de cette année porte sur les langages rationnels et propose de construire un automate reconnaissant un langage L donné à partir des réponses fournies par un oracle qui sait répondre aux questions

- un mot w appartient-il au langage L ?
- un automate $\mathcal{A}$ reconnaît-il le langage L et, si la réponse est non, donner un contre-exemple, c'est-à-dire un mot appartenant à L mais non reconnu par $\mathcal{A}$ ou un mot reconnu par $\mathcal{A}$ mais n'appartenant pas à L.

Partie 1. Cette partie permet de s'approprier les premières notions du sujet avec quelques questions de cours et plusieurs fonctions simples à écrire en `OCaml`, certaines donnent un cadre pour guider l'écriture du code.

Partie 2. Le sujet introduit la notion de séparabilité de deux mots par un mot et fait travailler sur cette notion et la relation d'équivalence associée.

Partie 3. Le sujet utilise la séparation de mots dans un arbre de décision et étudie des propriétés de ceux-ci.

Partie 4. Cette partie donne la définition d'un automate à partir de certains arbres par itération de constructions. La dernière question montre qu'on a construit un automate minimal.

On remarque que l'oracle peut être défini à partir d'un automate reconnaissant un langage L : la construction permet donc de minimiser un automate.

12 des 28 questions demandent d'écrire des fonctions dans le langage `OCaml`.

Une analyse détaillée des questions est présentée dans l'annexe R.

4.2.2 Commentaires généraux

- Les correcteurs ont été surpris par le nombre de candidats, de l'ordre du tiers des copies, qui écrivent

  ```
  if condition then true else false
  ```

  Bien entendu cette construction donne le bon résultat mais elle laisse entendre un manque de compréhension des booléens.

- L'usage d'une fonction récursive auxiliaire est une bonne pratique quand on a besoin de faire intervenir des variables supplémentaires mais cela devient inutile et même contre-productif quand la fonction auxiliaire reprend exactement les variables de la fonction.

```
let f a b c =
  let rec aux a b c =
    ...
  in
aux a b c
```

  On a le droit d'écrire directement une fonction récursive.

- De même un raisonnement par l'absurde permet d'écrire des preuves plus simplement. Cependant ce type de raisonnement obscurcit les choses quand il ne sert que d'enveloppe à un raisonnement direct. Une part non négligeable de candidats, pour prouver par exemple une égalité $a = b$, écrivent

  Je suppose qu'on a $a \neq b$. *Or je prouve que* $a = b$ *donc j'aboutis à une contradiction. J'en déduis que* $a = b$.

On a une sorte de mise en abyme qui n'aide pas à la compréhension.

4.2.3 Conseils aux futurs candidats

- Les réponses aux questions de programmation montre un manque de solidité dans l'écriture des codes chez certains candidats. Il est utile aussi de continuer le codage en dehors des heures de cours afin de maîtriser le langage.

- Quand on écrit des fonctions dans le langage de programmation, il est indispensable qu'elles soient compréhensibles : donnez des noms explicites aux variables et aux fonctions auxiliaires, expliquez à quoi correspondent les variables, expliquez ce que fait une fonction auxiliaire, ne surchargez pas les lignes par des ratures et des ajouts, écrivez une fonction sur une seule page et de manière linéaire sans renvoi fléché vers un bout de code, ...
  Un bon test est de se relire après quelques minutes : si vous ne comprenez pas ce que vous avez fait, comment le correcteur pourrait-il comprendre ?

- Les questions de dénombrements ne sont pas faciles même (surtout ?) lorsque le résultat semble évident. Vous ne perdrez pas votre temps en écrivant une description précise des ensembles et des fonctions bijectives, injectives ou surjectives que vous utilisez.

- Comme dans toutes les épreuves, il est indispensable de bien lire l'énoncé. Bien des réponses infructueuses sont le fruit d'une lecture défaillante. Lisez l'énoncé, re-lisez le.

- Soignez votre lisibilité, il ne suffit pas de comprendre, il faut aussi faire comprendre que vous avez compris. Évaluer une copie dont on doit décrypter les lignes qui ressemblent à des vaguelettes est impossible.

4.2.4 Conclusions

Même si les remarques et conseils ci-dessus ont mis l'accent sur les problèmes que les correcteurs ont rencontrés, il faut souligner l'investissement de la plupart des candidats. Ils se sont appropriés l'énoncé avec ses notations et ses concepts nouveaux, ils ont avancé dans les questions et apporté des réponses souvent pertinentes. La compréhension de la matière est visible chez la majorité des candidats.

4.3 Informatique 1 filière MPI

4.3.1 Généralités et présentation du sujet

Le sujet s'intéresse à différentes façons d'implémenter en langage C les tableaux associatifs. Il est composé de quatre parties indépendantes comprenant au total 34 questions.

La première partie s'intéresse aux arbres binaires de recherche équilibrés. Les questions de programmation sont pour la plupart des questions proches du cours, et ont été globalement réussies. Les quelques questions théoriques n'ont pas eu une grande réussite.

La seconde partie s'intéresse à l'implémentation d'un ramasse-miettes. Elle ne comporte quasiment que des questions de programmation. Cette partie a été globalement réussie.

La troisième partie traite de la structure de skip list. La moitié des questions sont des questions de programmation, l'autre moitié des questions théoriques. Cette partie, plus difficile que les autres, a mis en difficulté beaucoup de candidats.

La dernière partie concerne l'analyse syntaxique d'une chaîne de caractères encodant un tableau associatif. Elle ne comportait que des questions théoriques, liées aux grammaires et les automates. Les candidats qui ont abordé cette partie l'ont fait souvent avec succès.

Une analyse détaillée des questions est présentée dans l'annexe S.

4.3.2 Commentaires généraux

- Les codes attendus sont pour la plupart courts. Écrire des fonctions dépassant la dizaine de lignes augmente le risque d'écrire du code incorrect et doit être perçu comme un signe de complexité excessive.
- En C, il ne faut pas imbriquer des fonctions les unes dans les autres « à la OCaml », cela n'est pas correct.
- Lorsqu'une fonction doit renvoyer une référence vers une structure qui doit être allouée par la fonction, il faut allouer la mémoire dans le tas (en utilisant la fonction `malloc`) et non dans la pile (en déclarant une variable locale de type structure).
- Lorsqu'une variable est de type référence, il n'est pas forcément nécessaire de faire une allocation mémoire, cela peut simplement être un passage de paramètre.
- Une attention toute particulière a été apportée à la validité du code C présenté.
- Numéroter les questions est essentiel. Numéroter les sections n'est pas nécessaire.
- Mettre en valeur le code ou les résultats principaux est apprécié par le correcteur.
- Certains codes sont très difficiles à comprendre parce qu'ils contiennent trop de ratures ou de corrections.

- Certaines copies sont presque illisibles, rendant la correction difficile.
- Il est important de lire les annexes éventuelles données dans le sujet avant de commencer.
- Lorsque le sujet propose plusieurs parties indépendantes, il est souvent intéressant de lire rapidement le sujet dans sa globalité pour éviter de passer trop de temps sur une partie difficile, et ne pas avoir le temps de répondre à un grand nombre de questions plus abordables.

4.4 Informatique 2 filière MPI

4.4.1 Généralités

Le sujet est composé d'un problème unique s'intéressant au jeu de Shannon, un jeu qui se joue sur un graphe : deux joueurs s'affrontent en sélectionnant des arêtes, l'un pour construire un chemin, l'autre pour l'en empêcher. Il est divisé en trois sections et comporte 33 questions. A travers diverses approches du jeu de Shannon, le sujet permet de tester les candidats sur différentes parties du programme dont la programmation en C, les structures de données usuelles, l'écriture de requêtes SQL, la théorie des graphes et l'étude des jeux.

La première section s'intéressait d'une part à l'implémentation du jeu en C et d'autre part à l'étude de tournois en SQL. Le sujet propose aux candidats d'écrire un certain nombre de fonctions pour s'approprier les définitions du jeu en l'implémentant. Cette première sous-partie était essentielle pour bien comprendre le sujet. La sous-partie SQL était quant à elle indépendante.

Dans la deuxième section, il s'agit des définir certaines conditions suffisantes à l'existence d'une stratégie gagnante pour un joueur. Elle était divisée en deux sous-parties : la première proposait au candidat des questions de cours, ainsi que des questions assez générales sur l'existence de stratégie gagnante pour un joueur. La deuxième guidait le candidat sur la démonstration d'une condition suffisante plus fine pour l'existence d'une stratégie gagnante pour un joueur.

La troisième section introduit des notions autour des arbres couvrants d'un graphe, notamment celle d'arête principale afin de permettre au candidat de montrer une autre condition suffisante pour l'existence d'une stratégie gagnante. Cette section, divisée en deux sous-parties, était plus théorique et invitait les candidats à construire des raisonnements plus complexes.

Le sujet comporte des questions de programmation en C de difficultés variables, des questions de cours et des questions d'applications directes, permettant ainsi d'évaluer différentes compétences. La plupart des candidats ont compris le jeu de Shannon et ont réussi à répondre à plusieurs des questions théoriques. Une grande partie d'entre eux a su aborder les questions proches du cours. Le jury a particulièrement valorisé les candidats n'ayant pas fait l'impasse sur les question de SQL. Quelques uns ont réussi à aborder le sujet dans sa quasi-totalité.

Une analyse détaillée des questions est présentée dans l'annexe T.

4.4.2 Mise en forme des copies

Les programmes présentés par une bonne majorité des candidats respectent les règles d'indentation avec des retours à la ligne facilitant la lecture du code. Plusieurs copies restent malgré tout mal présentées voire même illisibles pour le correcteur, avec des indentations peu marquées, de grosses ratures, des renvois avec des flèches en bas de page, etc. Ces copies sont très difficiles à corriger. Certains candidats utilisent une couleur différente pour l'écriture des codes et de commentaires. Cela n'a facilité la lecture que dans le cas de copies peu soignées par ailleurs et ce n'est pas une priorité. En

particulier, il faut éviter d'utiliser des couleurs trop pâles, même pour les commentaires de code. Le vert est souvent illisible.

Le jury rappelle aux candidats de la filière MPI que pour éviter d'écrire plusieurs fois le même morceau de code, il est possible de séparer sa réponse à une question en plusieurs fonctions. Les commentaires « Ctrl+C, Ctrl+V » ou encore « de même pour le second cas » ont été pénalisés.

Les commentaires permettent aux correcteurs de mieux évaluer la compréhension du candidat et sont valorisés. En outre les noms de variables et de fonctions manipulés doivent avoir être suffisamment explicites pour permettre au correcteur de vérifier qu'elles sont utilisées à bon escient.

Certaines copies présentent beaucoup de ratures, voire des pages entières barrées. Même si cela est parfois réalisé de manière à ne pas gêner la lisibilité, au vu de la quantité de texte barré dans certaines copies, les correcteurs jugent utile de rappeler aux candidats que des feuilles de brouillon sont mises à leur disposition pour la phase de réflexion.

Les preuves, comme les programmes, doivent être écrites avec soin. Les raisonnements écrits sans connecteurs logiques, sans introduction, sans ponctuation, sans retour à la ligne, sans séparation des hypothèses et des conclusions, rendent la preuve inintelligible pour les correcteurs. De plus, des notations sont introduites dans le sujet, il s'agit de les respecter plutôt que d'utiliser des notations personnelles, qui mènent généralement à des erreurs de raisonnement.

Enfin, l'orthographe et la rédaction ne sont pas à négliger. Une copie de concours n'est pas un brouillon personnel, les abréviations et les fautes de français multiples n'ont pas leur place. Ces points ont été pénalisés.

4.4.3 Commentaires généraux

Programmation. Les programmes proposés par les candidats sont généralement de qualité correcte, sur la forme et le fond. Les correcteurs ont toutefois noté certaines erreurs récurrentes :

- Les noms des variables, des structures et de leurs champs doivent absolument être respectés.
- La spécification des fonctions demandées dans l'énoncé doit absolument être respectée. Le nom des fonctions a été globalement respecté, le type des paramètres aussi, mais plusieurs candidats ne vérifient pas que le type de sortie est cohérent.
- La fonction `malloc` est à utiliser pour allouer de la mémoire sur le tas : pour les tableaux, les maillons dans les listes chaînées. Le retour est de type `void*` et ne peut donc pas être utilisé pour des variables qui ont un type qui n'est pas un type de pointeur.
- Il faut veiller à libérer la mémoire avec l'utilisation de `free`, notamment lors de la suppression de maillons dans une liste chaînée.
- Beaucoup de candidats ne savent pas quand utiliser `->` ou `.`. Pour une variable qui est un pointeur vers une structure, on utilise `->` pour accéder aux champs et pour une variable qui n'est pas un pointeur (comme `gm` dans le sujet), on utilise `.` pour accéder à ses champs. Par exemple, on écrit `gm.n`.
- Lorsque le type de retour est `void` en C, on écrit `return ;` ou rien mais `return () ;` montre une confusion avec le langage OCaml.
- L'indentation et certaines accolades ne sont pas indispensables pour la compilation d'un code en C. Néanmoins, elles sont vivement conseillées pour rendre lisible le code pour le correcteur. Les correcteurs peuvent choisir de ne pas donner la totalité des points pour une question de code si le code n'est pas suffisamment clair. Le code SQL doit également être indenté et les mots clés sont plus lisibles en majuscule.

R Informatique option MP

Q1 - Une question de programmation simple mais pour laquelle l'erreur d'utiliser une variable comme motif a parfois été commise. **Q2 -** L'ordre lexicographique ne semble pas être connu par une partie des candidats. **Q3 -** La réponse à cette question ne saurait être de proposer l'utilisation du module `Hashtbl` : ce n'est pas une structure de données. De plus les fonctions du module indiquent que la structure de donnée utilisée est mutable, ce qui ne répond pas à la question de l'énoncé. **Q4 -** Il s'agissait ici d'utiliser les fonctions définies dans l'énoncé et non de donner la description de dictionnaires par des ensembles de couples. **Q5 -** Pour cette question certains ont bien vu le langage reconnu mais ont donné des expressions régulières qui ne le dénotaient pas. D'autres ont donné une bonne expression régulière, sans doute en appliquant un algorithme du cours, mais n'ont pas su donner une description du langage reconnu. **Q6 -** Beaucoup de candidats ne semblent pas savoir ce qu'est un automate de Glushkov. **Q7 -** Peu de candidats ont lu dans l'énoncé que `final` devait être une fonction. **Q8 -** Question souvent bien traitée. **Q9 -** Question souvent bien traitée. **Q10 -** Une réponse qui fait que `separated_by u v w` renvoie `false` lorsque u et v sont séparés par w ne saurait être correcte. **Q11 -** Cette question semble avoir désarçonné une partie des candidats, peut-être parce qu'elle est "trop simple". **Q12 -** Dans cette question, il fallait justifier que, si deux mots parviennent à un même état, alors, quel que soit le suffixe qu'on leur ajoute à ces mots, les transitions associées arrivent toujours dans un même état. Parfois ce fait est juste énoncé au milieu de raisonnements inutiles, la question n'apporte alors pas de points. **Q13 -** Demander de citer un théorème est sans doute une idée folle : il y a eu de nombreux noms proposés, parfois proches (Klein, Quine, Klein-Gordon) parfois moins (lemme de l'étoile). Donner le nom ne suffit bien entendu pas : *Le théorème de Kleene relie les langages reconnaissables et les langages rationnels* n'est pas une réponse satisfaisante. La démonstration de la finitude est trop souvent brouillonne. Cela se retrouve dans les questions 20. et 28. **Q14 -** Cette question demande une complexité linéaire. Beaucoup de codes proposés ne respectent pas cette complexité tout en fournissant une "preuve" de la linéarité. **Q15 -** Question facile mais quelques candidats n'ont pas compris le criblage. **Q16 -** Question souvent réussie. **Q17 -** Pour répondre simplement il fallait supposer que la fonction μ était croissante, ce qui ne semble pas avoir posé de problème. Quelques candidats calculent la complexité d'une succession d'instructions avec un produit. **Q18 -** Dans cette question il y a eu parfois des raisonnements compliqués et faux. **Q19 -** C'est la contraposée de la question précédente. **Q20 -** Une partie des candidats prouve en fait que le nombre de feuilles accessibles est borné : il est nécessaire de rappeler que toutes les feuilles sont accessibles. **Q21 -** Question souvent bien traitée. **Q22 -** Même remarque qu'à la question 7. pour `finals`. **Q23 -** L'arbre doit être un crible : ce n'est pas souvent respecté. La contrainte du nombre de feuilles n'a pas toujours été comprise. **Q24 -** Très peu de candidats ont su donner une preuve correcte. **Q25 -** Beaucoup de candidats ont répondu à la question en essayant d'écrire des fonctions de découpages de listes. C'est une méthode pour arriver au résultat mais, trop souvent, le manque d'explications du code rend celui-ci illisible. **Q26 -** Très peu de candidats ont vu qu'il y avait 2 cas de construction dans le cas d'une feuille et 2 cas d'appel récursif dans un seul des deux fils d'un nœud. **Q27 -** Cette question rassemblait de nombreux résultats antérieurs : quelques candidats ont réussi à les ordonner correctement. **Q28 -** Dernière question peu traitée.

↑RETOUR

S Informatique 1 MPI

Q1 - Question globalement bien faite. La représentation des chaînes de caractères comme un tableau de `char` est évoquée dans une très grande majorité des copies. La correspondance entre un `char` et un octet ne pose pas de problème à cette question, même si ce sera moins le cas par la suite pour allouer la bonne quantité de mémoire. Cependant, trop de candidats oublient le caractère de fin de chaîne `'\0'`.

Q2 - Question bien faite en général. A quelques exceptions près, l'ensemble des candidats ont compris qu'il s'agissait de convertir le nombre 97 en écriture binaire, puis en écriture hexadécimale. La représentation en binaire ne pose souvent aucun problème, celle en hexadécimale est moins maîtrisée.

Q3 - Question plutôt bien faite. Liée à la première question, de nombreux candidats n'ont pas alloué la bonne quantité de mémoire ou n'ont pas copié le caractère de fin de chaîne. Par ailleurs, certains candidats ne pensent pas à utiliser la fonction `strcpy`, rappelée dans l'annexe. Parmi eux, quelques uns ont réussi néanmoins à répondre correctement à cette question mais ont indéniablement perdu du temps.

Q4 - Question bien faite la plupart du temps. Après avoir alloué la bonne quantité de mémoire, l'ensemble des champs de la structure devait être rempli, en pensant à utiliser la fonction précédente `copie_chaine` pour remplir le champ `clef`.

Q5 - Question moyennement réussie. La définition donnée dans l'énoncé de la valeur d'équilibre d'un arbre impliquait l'écriture d'une fonction `hauteur`. On rappelle qu'en langage C, il n'est pas autorisé d'écrire le code d'une fonction dans une autre fonction. Le cas de base de cette fonction récursive est bien l'arbre vide, et non un arbre à un nœud, comme écrit dans certaines copies, rendant la fonction inopérante dans certains cas. Une fois cette fonction écrite, écrire la fonction principale, à savoir `abr_equilibre` n'a posé aucun problème.

Q6 - Question bien faite. L'énoncé n'imposait pas de démontrer le calcul de la complexité, mais juste d'écrire qu'il s'agissait d'une complexité linéaire en le nombre de nœuds. Par ailleurs un grand nombre de candidats ont pensé à l'ajout d'un champ hauteur dans la structure.

Q7 - Question très mal traitée. La plupart des candidats ayant répondu à cette question ont confondu arbre complet et arbre équilibré. Les rares copies ayant trouvé la relation de récurrence $N(h) = N(h-1) + N(h-2) + 1$, et ayant fait le lien avec le nombre d'or ont eu l'ensemble des points à cette question.

Q8 - Question très mal traitée, en rapport avec la question précédente. Les copies ayant traité le cas des arbres complets n'ont pas eu de point à cette question.

Q9 - Question plutôt bien traitée. A nouveau, certaines copies n'ont pas utilisé la fonction `strcmp`, donnée dans l'annexe du sujet, et ont, au mieux perdu beaucoup de temps, la plupart du temps n'ont pas pu valider cette question. Par ailleurs, rien n'indique que les seules valeurs renvoyées par `strcmp` soient 1, 0 ou -1. Enfin, quelques copies n'exploitent pas le contexte des arbres binaires de recherche, ce qui a été pénalisé.

Q10 - Question bien traitée dans l'ensemble. Comme l'ordre de grandeur pour la hauteur d'un arbre complet et d'un arbre équilibré sont identiques, la réponse donnée a souvent été juste, et a été validée.

Q11 - Question bien traitée pour ceux l'ayant abordée.

Q12 - Question très bien traitée pour sa première partie. Par contre, la plupart des copies ont évoqué une complexité linéaire ou en $O(n)$ sans préciser de quelle valeur de n il s'agissait. Les valeurs n'étaient pas les mêmes pour les deux algorithmes.

Q13 - Question plutôt bien traitée. L'ajout d'un terme en tête d'une liste chaînée relève de la question de cours, maîtrisée par presque tous les candidats. Quelques uns ont voulu rajouter l'élément à la fin de la liste, ce qui ne respectait pas la complexité imposée dans l'énoncé.

Q14 - Question plutôt bien traitée, mais peu de candidats ont bien initialisé le tableau du champ

`arbres`.

Q15 - Question moyennement réussie. Le cas où le nœud donné `element` est égal à `NULL` a été bien traité, ainsi que le côté récursif de cette fonction. Cependant, trop peu de candidats ont pensé à arrêter le marquage dès que l'on tombe sur un nœud déjà marqué. Sans cette précaution, le complexité de cette fonction devenait catastrophique.

Q16 - Question moyennement réussie. Il y avait trois zones de mémoires à libérer, dépendantes entre elles. Cette question demandait donc de la précision pour bien faire les choses dans l'ordre, ce qui a été bien fait bien souvent. Malheureusement, certaines zones mémoire n'ont pas été libérées, ou au contraire, certains candidats ont libéré récursivement les zones mémoire occupées par les descendants du nœud concerné.

Q17 - Question plutôt bien traitée. La réponse attendue était l'ensemble des nœuds atteignables à partir de la structure `rm`. Une certaine tolérance a été accordée aux copies évoquant une spécificité de moins grande importance, comme le fait que l'ensemble de ces nœuds sont non marqués à la fin de l'exécution de cette fonction.

Q18 - Question moyennement traitée. Voir que le champ `objet_conserve` n'est pas modifié dans le code proposé n'a pas posé de problème, le corriger correctement beaucoup plus. L'utilisation de la fonction `rm_inserer_tete` entraîne une fuite mémoire, et a donc été légèrement pénalisée.

Q19 - Question rarement bien réussie. Cette question technique demandait de bien comprendre la structure et l'algorithme. De nombreuses copies ont fourni un code non abouti. D'autres n'ont pas exploité la structure de skip list, et ont fourni un code dont la complexité était moins bonne que celle d'une liste chaînée classique.

Q20 - Question bien traitée dans l'ensemble même si les réponses manquaient parfois de précision. La valeur renvoyée ne suit pas tout à fait une loi géométrique, 0 pouvant être renvoyé, et la valeur ne pouvant dépasser `MAX_NIVEAU`.

Q21 - Question rarement très bien traitée. Les candidats ayant réussi la question 19 ont souvent réussi cette question, dont la structure de code était proche. Sans valoir zéro, les candidats parcourant l'ensemble des niveaux en partant de la sentinelle pour trouver la bonne place ont été pénalisé, cette stratégie n'ayant pas une complexité satisfaisante.

Q22 - Question souvent mal comprise. Deux choses étaient à faire ici : compléter le tableau `nœuds_a_mettre_a_jour` en y plaçant `list_sentinelle` pour les nouveaux niveaux, puis actualiser le champ `niveau_actuel` de la skip list.

Q23 - Question moyennement bien traitée. L'insertion d'un élément dans une liste chaînée à une place donnée semble acquise pour un bon nombre de candidats, mais la compréhension du contexte général a parfois fait défaut. Par ailleurs, la cohérence de la réponse avec les réponses données aux deux questions prédécentes a été prise en compte dans la notation.

Q24 - Question bien traitée par la plupart des candidats, même si certains n'ont pas saisi ce qui différencie ces deux types d'algorithmes probabilistes.

Q25 - Question rarement bien traitée. Il fallait tout d'abord montrer que le nombre de niveaux était logarithmique en le nombre d'éléments contenus dans la skip list. Puis voir que le parcours de chaque niveau se fait en temps constant, pour aboutir à une complexité logarithmique.

Q26 - Question rarement traitée. Après avoir établi la probabilité p_k qu'un élément donné soit présent au niveau k, on pouvait repérer que la question se ramenait à l'espérance d'une loi binomiale de paramètre n (le nombre d'éléments dans la skip list) et p_k.

Q27 - Question rarement traitée. Cette question très mathématique demandait d'utiliser la fonction de répartition pour établir la loi du maximum entre les n variables aléatoires mutuellement indépendantes, qui suivent, aux réserves près énoncées plus haut, une loi géométrique tronquée. Puis d'établir le lien entre l'espérance d'une loi et la fonction de répartition. Moins d'une dizaine de copies ont réussi à le faire de façon satisfaisante.

Q28 - Question rarement traitée et encore plus rarement avec la rigueur suffisante.

Q29 - Seules quelques copies ont abordé cette question, qui était une synthèse des deux questions précédentes. Cela a permis de valoriser les candidats ayant réussi les questions précédentes.

Q30 - Question très bien traitée. Tous les candidats, ou presque, ayant essayé cette question l'ont réussi.

Q31 - Question plutôt bien traitée. Les règles de production non déterministes n'ont pas été acceptées. Tout l'enjeu était de gérer le cas du zéro à part du cas d'un nombre non nul, dont l'écriture ne devait pas commencer par un zéro.

Q32 - Question plutôt bien traitée dans l'ensemble. Certains candidats ne font pas l'effort de dessiner l'ensemble de l'arbre de dérivation.

Q33 - Question bien traitée pour ceux qui l'ont abordée. Seuls les automates déterministes ont été acceptés, comme précisé dans le sujet. A nouveau, il fallait gérer le cas des valeurs nulles à part.

Q34 - Question correctement traitée. Cette question demandait de la précision pour obtenir l'ensemble des points. Cette question valorisait bien souvent les candidats ayant donné un bon automate. Cependant, les candidats ayant répondu de façon honnête par rapport à leur automate, en indiquant clairement qu'il y avait donc un problème dans leur réponse précédente, ont obtenu quelques points, l'enjeu de cette question étant justement de tester l'automate.

↑RETOUR

T Informatique 2 MPI

Q1 - Pour cette question, il fallait penser à bien allouer la mémoire pour les deux tableaux et les initialiser. Il fallait aussi bien remplir les autres champs. Néanmoins, il n'était pas nécessaire de déclarer `gm` qui est une variable globale, ni de lui allouer de la mémoire. Initialement, aucune arête n'est marquée et toutes les listes sont vides donc égales à `NULL`. Peu de candidats ont pensé à tout faire correctement et la question a été discriminante.

Q2 - Il s'agissait d'ajouter des maillons aux deux listes chaînées correspondant à u et v et d'incrémenter le nombre d'arêtes non marquées. Il n'était pas nécessaire de vérifier que l'arête n'étaient pas déjà présente. Beaucoup de candidats ont fait des erreurs dans la façon d'ajouter un maillon dans une liste chaînées : ici le plus simple était l'ajout en début de liste et l'utilisation de `malloc` était nécessaire. Cette deuxième question a été également très discriminante.

Q3 - Cette question a été très peu réussie. Rares sont les candidats qui ont pensé à libérer la mémoire. La suppression d'un maillon dans une liste chaînée devrait être une compétence maîtrisée dès la MP2I.

Q4 - Cette question et la suivante permettait de vérifier que les candidats connaissait une implémentation simple de la structure Unir & Trouver. Celle-ci a été généralement bien réussie par les candidats. Même si c'était tentant, la question imposait de ne pas modifier `A` et donc l'heuristique de compression de chemin ne devait pas être utilisée ici.

Q5 - Cette question a été bien réussi par les candidats connaissant leur cours. Il fallait bien faire attention et travailler avec les représentants des classes des deux éléments.

Q6 - Q7 - Il suffisait de faire le lien entre la structure Unir & Trouver et la structure de graphe. Cela a été bien réussi. Les erreurs les plus fréquentes étaient sur l'utilisation de `->` à la place d'un point.

Q8 - Cette question a été bien réussie par les candidats ayant compris les conditions de victoire du jeu et comment utiliser la structure Unir & Trouver.

Q9 - Il n'était pas suffisant de vérifier que Positus n'a pas encore gagné, il fallait aussi vérifié qu'il ne resté par d'arêtes non marquées.

Q10 - Q11 - Ces deux questions ne posaient pas beaucoup de difficultés. Il fallait bien penser à utiliser une seule fois `gm_remove`.

Q12 - Les questions de SQL ont été peu traitées par les candidats. Les bonnes réponses ont été fortement valorisées. Cette première question a été bien réussi par les candidats ayant travaillé leur chapitre de base de données. Il fallait bien prendre une clé primaire après l'utilisation de `GROUP BY`.

Q13 - Cette question était plus difficile. Il fallait bien comprendre qu'un tournoi est gagné par le joueur ayant le plus de matchs gagnés. Les réponses partielles ont été valorisées.

Q14 - Dans cette question, il fallait prendre en compte la situation où le joueur A a le rôle de Positus et la situation où le joueur B a le rôle de Positus, c'est-à-dire les matchs où le joueur A a le rôle de Minus. Ces deux situations pouvaient être traitées dans des requêtes imbriquées. On pouvait ensuite utilisées une jointure ou une opérations ensemble pour combiner puis sommer les résultats. Les réponses partielles ont été valorisées.

Q15 - Le variant a généralement été identifié par les candidats : il fallait ensuite bien justifier, ce qui n'a pas toujours été fait. Il ne fallait pas oublier de répondre à la deuxième partie précisément.

Q16 - Cette question de cours n'a pas été souvent réussie. Les correcteurs ont accepté toutes les réponses cohérentes. Une « suite de coups à effectuer pour gagner » n'est pas une stratégie gagnante.

Q17 - La plupart des candidats ont donné des réponses peu rigoureuses affirmant que le coup supplémentaire ne peut être qu'à l'avantage du joueur. Ceci n'étant pas vrai dans les jeux en général, ces réponses n'ont rapporté aucun point. Les candidats ayant mené un raisonnement rigoureux en posant une stratégie adaptée étaient peu nombreux. La question a été très discriminante.

Q18 - Cette question était longue et demandait une réponse en plusieurs partie. Il fallait d'abord

justifier que pour n'importe quel instance du jeu, un des deux joueurs disposait d'une stratégie gagnante. Il fallait ensuite justifier qu'il n'existait que les trois types d'instances proposées. Enfin, on attendait un exemple simple ou rigoureusement justifié pour chacune des trois instances. Peu de candidats ont donné des réponses complètes. Les exemples ne devaient pas être des multigraphes.

Q19 - Cette question centrale a été très discriminante. Il fallait repérer que $T \setminus \{\alpha\}$ induisait un graphe à deux composantes connexes et que T' induisait un graphe connexe. Beaucoup de candidats ont fait des erreurs de raisonnement et confondent arbre couvrant et ensemble d'arêtes couvrant les sommets.

Q20 - Les quelques candidats ayant abordé cette question après avoir compris la précédente ont bien réussi. Il fallait penser à exploiter à nouveau les fonctions de la structure Unir & Trouver.

Q21 - Très peu de candidats ont réussi cette question. Les réponses partielles et les schémas explicatifs clairs ont été récompensés. Il s'agissait d'une question difficile.

Q22 - La question a été bien réussie. Il fallait utiliser le résultat de la question 19.

Q23 - Cette question ne s'abordait par exactement comme la précédente. Il fallait vérifier l'existence et la pertinence des arêtes envisagées.

Q24 - Il y avait deux parties dans la question : une récurrence et un raisonnement sur la fin de partie. Les candidats qui ont affirmé que l'ensemble de toutes les arêtes sont marquées en fin de partie ont été pénalisé pour la deuxième partie de la question.

Q25 - Cette question a généralement été mal comprise. Il ne fallait pas confondre les graphes G et $\hat{G}$.

Q26 - Cette question était plus facile. La plupart des candidats ayant traité la question ont identifié les arbres à considérer. Il fallait néanmoins justifier les choix.

Q27 - Cette question, similaire à la précédente, a été moins bien traitée par les candidats. Il ne suffisait pas de plaquer le même raisonnement, il fallait l'adapter intelligemment.

Q28 - Dans cette question, le deuxième cas a été souvent correctement abordée. Le premier cas n'a été correctement traité que par très peu de candidats.

Q29 - La question était difficile. Un couple d'arbres couvrants éloignés n'est pas nécessairement d'intersection vide.

Q30 - L'algorithme de Kruskal n'était pas la meilleur réponse car les graphes ne sont pas pondérés. Les correcteurs ont valorisé les candidats utilisant un simple parcours.

Q31 - Q32 - Q33 - Très peu de candidats ont abordé sérieusement les trois dernières questions.

↑RETOUR