### 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.

Par ailleurs, Les correcteurs souhaitent ajouter quelques commentaires supplémentaires pour les futurs candidats :

- Les candidats doivent faire attention à la complexité de leurs fonctions, tout particulièrement lorsqu'il existe une manière élémentaire de l'améliorer. Par exemple, ne pas rappeler plusieurs fois une fonction sur les mêmes arguments.
- Lorsqu'un candidat souhaite réutiliser une partie de son code plusieurs fois : il peut soit le recopier, soit écrire une fonction supplémentaire et l'appeler sur différents arguments. Les réponses comportant des commentaires « idem pour le deuxième cas » ou « Ctrl+C, Ctrl+V » ont été pénalisées.
- Lorsque cela est pertinent, les candidats peuvent illustrer leurs raisonnements et leurs algorithmes à l'aide de schémas. Ils doivent être propres, soignés et annotés. Dans ce cas, ils sont fortement valorisés mais ne se substituent pas à un raisonnement complet.
- Il est important d'annoncer clairement les raisonnements classiques utilisés : variants, invariants, récurrences et sur quelle valeur, induction et objet. Il faut ensuite dérouler le raisonnement : les mots « immédiat », « trivial » ou « évident » ne rapportent pas de points.
- Un raisonnement ne saurait être présenté comme un bloc de texte. Pour faciliter la lecture et montrer au correcteur que le raisonnement est bien mené, il est pertinent d'indenter une preuve.
- Enfin, il est rappelé au candidat qu'il faut écrire le bon numéro de question devant une réponse.