Option informatique

Présentation du sujet

Le sujet propose une méthode permettant de créer un pavage aléatoire d'un échiquier par des dominos en faisant le lien avec des arbres couvrants de graphes particuliers. La première partie demande d'écrire des fonctions générales sur les graphes. La deuxième partie étudie les propriétés sur les arbres. La troisième partie implémente la génération aléatoire d'un arbre couvrant. La quatrième partie fait le lien entre pavage et arbre couvrant. Enfin, la cinquième partie introduit la notion de graphe dual pour concrétiser la bijection précédente.

Le sujet étant très long, peu de candidats ont pu aborder les quatrième et cinquième parties. Néanmoins, certains rares candidats ont réussi à traiter correctement l'intégralité de l'épreuve.

Analyse globale des résultats

Le jury constate cette année que la programmation en Caml est globalement acquise pour une majorité de candidats, malgré une maitrise parfois superficielle de certains éléments de syntaxe. Toutefois, les raisonnements théoriques ont souvent été incomplets et des éléments de preuve importants sont passés sous silence ou jugés évidents.

Comme l'an dernier, des consignes sans ambiguïté étaient présentées en début de sujet sur les fonctions informatique autorisées pour la composition. Certains candidats ont toutefois réécrit des fonctions déjà existantes dans les librairies autorisées, soit par ignorance de ces fonctions, soit car ils n'avaient pas lu correctement les consignes. Le jury encourage donc l'utilisation de fonctions comme `List.length`, `List.rev` ou encore `Array.init` lorsqu'elles s'avèrent adéquates.

Commentaires sur les réponses apportées et conseils aux futurs candidats

De manière non exhaustive, nous citons des erreurs ou problèmes fréquents rencontrés dans les copies :

— l'utilisation de filtrage implicite alors que la présence d'un `when` est nécessaire. Par exemple, le code `match x with | y - 1 -> ...` renverra systématiquement une erreur ;

— le renommage inutile des variables, par exemple avec `match s, t with | x, y -> ...` ;

— l'utilisation maladroite des références, voire leur oubli total. Le jury constate une absence régulière du `!`, ou l'utilisation de `=` ou `<-` au lieu de `:=`. Il à ce sujet 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 ;

— l'oubli quasi-systématique des parenthèses, tant autour des arguments d'une fonction que lors de l'écriture de plusieurs instructions. Par exemple, `f n - 1` ne fait pas du tout le même calcul que `f (n - 1)`. De même, dans le code

`if` *condition* `then` *instruction1* `else` *instruction2* `;` *instruction3*

instruction3 sera toujours exécutée, indépendamment de la condition.

Si la programmation récursive et la programmation itérative sont interchangeables, 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 s'appelant `aux`, pénibles à relire et avec des cas d'arrêts erronés, sans aucune explication, là où une simple 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.

Enfin, le jury rappelle que les questions de programmation attendent rarement une réponse qui fait plus d'une dizaine de lignes, et au plus dans une ou deux questions de la deuxième moitié du sujet. Une réponse contenant une page et demie de code sans explications, avec 3 ou 4 fonctions auxiliaires aura rarement plus de la moitié des points, même si le code est correct.

En termes de présentation, il est rappelé que même si Caml n'impose pas l'utilisation de sauts de lignes et d'indentation comme Python, il est fortement recommandé de présenter sa copie en utilisant de manière pertinente ces sauts de lignes et indentations pour faciliter la lecture du code. Hélas, de nombreuses copies produisent des pavés de code où les retours à la ligne sont faits uniquement quand il n'y a plus de place pour écrire, rendant la compréhension des fonctions difficile. Le jury a valorisé les copies dont le code était bien présenté.

Concernant les questions théoriques, le jury a constaté cette année un grand nombre de preuves rédigées de manière très superficielle, avec des explications comme « on voit que », « de proche en proche », etc. Par exemple, justifier qu'un graphe connexe d'ordre n contient au moins $n-1$ arêtes ne peut pas se faire en affirmant uniquement « il faut au moins une arête par sommet ». Le jury encourage les candidats à formaliser leurs éléments de preuve lorsque c'est nécessaire, en expliquant leurs notations. Attention toutefois, une preuve formée uniquement de symboles et de flèches sans explications n'obtiendra aucun point dans la majorité des cas.

Il y a eu quelques incompréhensions sur la **Q2** car l'indication qui se trouvait entre **Q2** et **Q3** faisait référence à cette dernière. Il était attendu pour **Q4** et **Q5** une fonction en temps constant qui utilisait la division euclidienne par p ou $q-1$ selon la question. Les réponses de - **Q6** ont parfois été très lourdes en traitant séparément les 4 coins, les 4 bords et le centre du graphe. On pouvait ici se contenter d'itérer sur les arêtes, ou de vérifier pour chaque sommet s'il était sur chacun des 4 bords. Le jury souligne qu'en **Q8** il est inutile de rappeler qu'une partie de $\mathbb{N}$ doit être minorée pour admettre un minimum. En revanche, la condition d'être non vide est indispensable. Pour **Q9** et **Q10**, il était attendu de la rigueur pour chacune des implications prouvées. Certains candidats n'ont pas lu que la définition d'*arbre* était donnée par le sujet et ont travaillé implicitement avec des arbres binaires enracinés. Pour **Q12**, il fallait bien lire le sujet, qui rappelait que les arguments étaient déjà des représentants, et qu'il était nécessaire de modifier la hauteur le cas échéant. Pour **Q15**, une réponse utilisant un parcours de graphe était satisfaisante, même si elle n'utilisait pas la structure introduite. En **Q18**, très peu de candidats ont vu que la structure de données introduite dispensait de gérer la création de cycle, et ont fourni un code très long, souvent faux, pour répondre à un problème qui n'existait pas. Pour **Q28** et **Q29**, un argument géométrique suffisait.

Conclusion

Même si le jury constate cette année plus de difficultés liées à la programmation et aux raisonnements théorique, il est parfaitement conscient que les conditions d'apprentissage perturbées liées à la situation sanitaire pendant deux années en sont une cause majeure.

Il sait également qu'il est difficile d'écrire du code sur papier et recommande un entrainement régulier sur ordinateur au cours de l'année pour acquérir les bons réflexes.