Rapport des correcteurs - Composition d'informatique 2h, filière PSI

1 Commentaires généraux

Le sujet portait sur la résolution de jeu de Röckse. Etant donnée une grille avec des pénalités, une case de départ, une case d'arrivée et un ensemble de mouvements possibles, il fallait trouver un chemin qui minimise la somme des pénalités. Des cases bonus ajoutent, une fois atteintes, un ou plusieurs mouvements possibles. La partie I portait sur les primitives pour manipuler les sauts et les chemins. La partie II s'intéressait à la résolution par recherche exhaustive. La partie III portait sur la résolution avec un algorithme glouton. Enfin, la partie IV portait sur la résolution par programmation dynamique.

L'énoncé commençait par un ensemble de consignes clairement détaillées, en particulier sur les opérations Python autorisées.

1.1 Présentation et lisibilité des réponses

Le jury rappelle que le code demandé est relu, il est donc nécessaire de produire du code lisible. En particulier les noms de variables doivent être judicieux et le code doit être commenté. Par exemple, les noms de variable monolettre `i`, `j`, `x`, `y` sont à proscrire. Par ailleurs, le code demandé ne dépasse *jamais* une demi-page. Dans de nombreuses copies, l'indentation n'est pas toujours claire, ce qui est une source de confusion, notamment quand le code est sur deux pages. Par ailleurs, beaucoup de copies sont raturées au point d'être difficilement lisibles. On rappelle que les réponses doivent être préparées au brouillon. Parfois, les candidats présentent plusieurs réponses à une même question sans indiquer celle qui est retenue. Enfin, certains candidats lisent mal l'énoncé et font du hors-sujet.

1.2 Maîtrise du langage Python

Dans l'ensemble, les candidats montrent une meilleure maîtrise du langage Python que les années précédentes. Cela dit, on note encore de nombreuses maladresses syntaxiques. Par exemple, il est souvent préférable d'utiliser `for (x,y) in L` puis `x` et `y` ; que `for i in range(len(L))` puis `L[i][0]` et `L[i][1]`. Dans le même registre, le fonctionnement de l'opération `range` semble confus pour divers candidats. Si un programme appelle `for i in range(4)`, la variable `i` va prendre successivement les valeurs 0, 1, 2 et 3.

Plus grave, beaucoup de candidats *ignorent le concept de donnée mutable*, pourtant central dans le langage Python, ce qui a été une grande source d'erreurs dans les copies, notamment dans la question 12.

1.3 Maîtrise de l'algorithmique

Dans l'ensemble, les candidats ont écrit des algorithmes syntaxiquement et sémantiquement cohérents. Cela dit, de nombreuses erreurs émaillent les copies. Par exemple, les cas de bords sont très souvent ignorés (typiquement en question 3). D'autre part, la récursion est presque toujours mal écrite et sa complexité est rarement comprise, ce qui mène à des réponses où les candidats appellent plusieurs fois une fonction au lieu de loger le résultat dans une variable à consulter plusieurs fois.

1.4 Bilan général

Globalement, le niveau est meilleur que l'année précédente mais reste très lacunaire. Celà étant, beaucoup de candidats maîtrisent mal la récursion et les calculs de complexité et se montrent très maladroits dans l'utilisation du langage Python. La notion de donnée *mutable* n'est presque jamais comprise, ce qui est inquiétant. Les enseignements devraient peut-être insister davantage sur ce point.

2 Commentaires détaillés

Partie 1

Question 1 Cette question a été bien traitée dans l'ensemble.

Question 2 Cette question a été bien traitée dans l'ensemble.

Question 3 Beaucoup de copies calculent un saut sans vérifier l'existence des cases de départ et d'arrivée. Souvent, les candidats modifient le paramètre `saut` sans comprendre qu'il est mutable et donc entrant/sortant. Enfin, les cases bonus sont très souvent mal gérées.

Question 4 La plupart des candidats ont développé $\neg\delta \gg 0$, alors que la condition pouvait être écrite directement. Parfois ces développements sont faux, car les lois de De Morgan ne sont pas maîtrisées ; ce qui est inquiétant.

Partie 2

Question 5 Très peu de bonnes réponses, la plupart des candidats ne comprennent pas que les sauts sont à trouver.

Question 6 La récursion est souvent mal écrite, certains candidats utilisent plusieurs appels récursifs (un pour recuperer le poids, un pour récupérer les sauts) sans conscience de la complexité. Certains tentent de construire tous les chemins, puis de sélectionner le meilleur, ce qui est désastreux en complexité spatiale. La complexité est presque toujours fausse et mal justifiée.

Question 7 Très peu de candidats ont vu que la clé doit contenir les bonus rencontrés.

Partie 3

Question 8 Beaucoup de réponses fausses. En particulier, très peu de candidats ont vu l'effet d'horizon.

Question 9 Le cas où aucun chemin n'est trouvé n'est presque jamais traité. Certains candidats n'utilisent pas trouver_complet_rec alors que c'était explicité dans l'énoncé.

Partie 4

Question 10 Un nombre significatif de candidats ne comprennent pas l'énoncé et retournent une chaîne de caractères avec un code binaire. Quand le code est calculé, il est souvent donné en base 10 alors qu'on demandait en base 2. Parfois le candidat se contente d'incrémenter un compteur à chaque True dans le masque, ce qui ne donne pas la réponse attendue. Beaucoup de confusion, également, dans le sens de la somme (poids faible/poids fort).

Question 11 Très peu traité, mais plutôt bien compris dans l'ensemble ; parfois le nouveau code bonus est non explicité et juste écrit `code_bonus'`.

Question 12 Très peu traité, quelques excellentes réponses. Attention aux données mutables : lorsqu'on écrit `l.append(m)`, avec `m` une liste, toute modification postérieure de `m` se répercute dans `l`, ce qui est une importante source d'erreurs.

Question 13 Très peu traité, mais avec de bonnes réponses.

Question 14 Très peu traité, mais avec de bonnes réponses.

Question 15 Très peu traité, mais avec de bonnes réponses.