Rapport des correcteurs - Composition d'informatique 2h, filière PSI, session 2023

1 Commentaires généraux

Le sujet portait sur la réalisation d'un système de gestion de versions de grands textes et plus particulièrement la notion de *différentiel* entre textes. La partie 1 portait sur les différentiels entre textes de même taille. Puis, la partie 2 étudiait les différentiels entre textes de tailles différentes et les algorithmes pour la gestion de versions. Enfin, la partie 3 étudiait le lien entre le calcul d'un différentiel entre textes et la recherche d'un plus court chemin dans un graphe.

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

1.1 Des réponses illisibles

Le jury rappelle que le code demandé sera *lu*. 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 à une lettre i,j,x,y sont à proscrire, au moins dès qu'on dépasse le cas simple de l'indice d'une boucle seule. Ce n'est pas une épreuve de mathématiques mais d'informatique ! Par ailleurs le code demandé ne dépasse *jamais* une demi-page. Il est *inadmissible* que la quasi-totalité des candidats méconnaissent ces règles élémentaires ; ou soient dans l'incapacité de les respecter. On ne saurait qu'encourager les enseignants de classe préparatoire à *insister* sur ces points quitte à sanctionner par la note 0 tout code qui ne respecte pas ces principes de base.

Sur des questions délicates comme la construction ou l'application d'un différentiel, beaucoup de candidats n'ont pas su répondre et se sont enfoncés dans la construction confuse d'un automate d'états à base de variables drapeau. L'absence de commentaire condamne souvent ces tentatives laborieuses à la note 0 dès que le code n'est pas exact. L'important est de produire un code *court* et *clair*.

1.2 Du code incohérent

De nombreuses copies commettent des incohérences élémentaires comme utiliser des arguments de façon incompatible avec leur type, ou renvoyer une donnée du mauvais type. Par exemple, la fonction inverse de la question 6 doit retourner un différentiel, pas un texte. Vérifier la cohérence des types permet de repérer tôt et facilement de nombreuses erreurs, ce qui permet parfois de corriger des incompréhensions profondes, de même que vérifier que les accès à un tableau sont dans les bornes. Le jury rappelle que l'énoncé doit être lu avec soin avant de commencer la composition.

1.3 De l'algorithmique de base non maîtrisée

Nombre de copies montrent une incompréhension inquiétante de l'utilisation des boucles. Dans certaines copies, des boucles `while` et `for` imbriquées étaient utilisées pour tenter de reproduire le fonctionnement d'un `while`. Les erreurs de complexité algorithmiques sont aussi communes : la complexité de la composition de deux fonctions n'est en général pas la composition de leurs complexités, et la complexité de deux boucles imbriquées de tailles n et m n'est pas $O(n+m)$. Il est par ailleurs très inquiétant que le passage par adresse du langage Python ne soit quasiment jamais compris. À nouveau, le jury recommande aux enseignants de classe préparatoire de bien insister sur tous ces points. À un niveau supérieur, les quelques candidats qui ont atteint la partie 3 montrent une méconnaissance de l'algorithme de Dijkstra et de sa complexité, pourtant au programme.

1.4 Bilan général

Globalement, le niveau est assez faible, beaucoup de candidats ne semblent pas du tout maîtriser l'algorithmique et le langage Python, même pour les algorithmes simples ; et se montrent incapables d'écrire un code propre et lisible. C'est extrêmement inquiétant pour un concours de ce niveau. Il est par ailleurs significatif que la majorité des copies n'aient traité que la moitié de l'énoncé.

2 Commentaires détaillés

Partie 1

Question 1 L'épreuve commençait avec une simple comparaison de textes. La plupart des candidats ont bien répondu, ce qui est heureux. Toutefois, dans la précipitation, beaucoup de candidats ont perdu des points en oubliant de tester l'égalité des longueurs.

Question 2 Cette question demande une fonction pour calculer la distance entre deux textes. Cette question, également très simple, a été globalement bien traitée. Certaines copies utilisent deux dictionnaires de manière impropre ; on ne saurait que recommander d'aller au plus simple. Quelques copies ont parfois présenté des incongruités comme `if(texte1[i] == texte2[i]) d += 0`.

Question 3 Cette question demande une fonction qui teste l'absence de caractère commun entre deux textes. Malgré la simplicité de la question, nombre de candidats ont échoué à répondre correctement. Certains n'ont pas utilisé de dictionnaire, malgré la consigne explicite. D'autres se sont perdus dans une imbrication de boucles avec une complexité inutilement élevée. Une copie donne une réponse – fausse et parfaitement illisible – d'une page entière, alors que la solution tient en quelques lignes, ce qui a également été sanctionné. Quant aux réponses qui se contentent de mettre un drapeau à `False`, au lieu d'interrompre et de retourner le résultat, elles ont été acceptées ; bien que la complexité puisse être facilement réduite.

Question 4 Cette question demande de calculer le différentiel de deux textes de même longueur. Très peu de candidats ont donné une réponse exacte à cette question, certes plus technique mais tout de même assez simple. Certains ont cru devoir réutiliser la fonction `aucun_caractère_commun` de la question 3, ce qui a augmenté inutilement la complexité. Comme d'habitude, les *corner cases* ont rarement été traités correctement. Typiquement, lorsque les candidats se contentent d'insérer une tranche dès que `texte1[i] == texte2[i]` en oubliant la dernière tranche. Beaucoup de candidats réinventent inconsciemment les automates d'états, mais sans le moindre commentaire ; ce qui rend la réponse particulièrement illisible – et la marge de tolérance du correcteur particulièrement faible.

Question 5 Dans cette question, on demandait d'écrire une fonction pour appliquer un différentiel à un texte. Beaucoup de candidats n'ont pas utilisé correctement les sélecteurs mis à leur disposition pour manipuler un différentiel. Certains ont même jugé utile de construire leur propre structure de données, qui s'avère la plupart du temps mal choisie et inutilement complexe. A nouveau, les *corner cases* ont très rarement étés traités. Typiquement, l'oubli du texte qui suit la dernière tranche.

Question 6 Cette question demande d'inverser le différentiel. Il s'agit d'une simple inversion de relation, qui n'a généralement pas posé de difficultés ; sauf manipulation impropre de la structure de données différentiel.

Question 7 Cette question demande d'écrire les primitives de manipulation d'un texte versionné. Il s'agissait surtout de manipuler avec soin une pile de différentiels, tout réutilisant les fonctions des questions précédentes. Généralement, les candidats ont plutôt bien répondu. On note quelques erreurs, dont une incompréhension de la copie par référence utilisée par Python (exemple : historique affecté à une variable intermédiaire inutile) ; ou encore la simple lecture du sommet de la pile sans dépilage.

Partie 2

Question 8 Dans cette question, on demande d'écrire une fonction qui calcule le poids d'un différentiel. Il suffisait de lire l'énoncé et de traduire la définition d'un poids ; ce qui n'a globalement pas posé de difficultés aux candidats qui ont répondu.

Question 9 Cette question demande une récurrence pour calculer la distance d'édition entre deux textes. Il s'agit en fait d'une question de cours, la distance de Levenstein étant au programme. Malgré cela, le cas `texte1[i+1] != texte2[j+1]` n'est presque jamais traité correctement. Il fallait bien sur répondre `M[i,j]+2`, puisqu'il faut supprimer l'ancien caractère et insérer le nouveau.

Question 10 Cette question demande ensuite d'implémenter la récurrence trouvée en question 9. Bien que ce soit une question de cours, le peu de copies qui ont répondu ont souvent donné un code faux, avec, comme toujours, des *corner cases* mal gérés : une initialisation mal écrite, un tableau de mauvaise taille, des accès hors des bornes de la matrice d'édition.

Question 11 Dans cette question, on demande d'exploiter la matrice d'édition pour trouver le différentiel de deux textes de taille quelconque. Cette question demandait une bonne compréhension du fonctionnement de la matrice d'édition. Seule une poignée de candidats a su utiliser la matrice d'édition pour retrouver le différentiel.

Question 12 Dans cette question, on se propose de déterminer si deux différentiels sont en conflit. Il suffisait d'implémenter la formule donnée dans l'énoncé, ce qui ne devait normalement pas poser de problème. Sur le faible nombre de candidats qui ont atteint cette question, très peu ont eu le réflexe d'implémenter une fonction intermédiaire qui décide si deux intervalles intersectent, ce qui a donné des réponses particulièrement illisibles.

Question 13 Cette question demande d'écrire une fonction pour fusionner deux différentiels sans conflits. La seule difficulté était de translater proprement les tranches, ce qui a très rarement été fait.

Partie 3

Question 14 Dans cette question, on demande de construire la fonction successeur du graphe d'édition entre deux textes. On demande également de montrer une correspondance entre le graphe ainsi décrit et la matrice d'édition. Très peu de candidats ont vu qu'il était inutile de créer un arc $(i, j) \to^2 (i{+}1, j{+}1)$ quand `texte1[i] != texte[j]`. Par ailleurs, la quasi-totalité des réponses ne donnent pas la preuve demandée.

Question 15 Dans cette question, il s'agissait d'expliquer en quoi l'algorithme de Dijkstra permet de résoudre le problème. Cette question demandait une connaissance fine de l'algorithme de Dijkstra et n'a presque jamais été traitée.

Question 16 Dans cette question, on s'intéresse à la complexité de l'approche par plus court chemin, comparativement à l'approche par programmation dynamique. Parmi les réponses, très peu de candidats connaissent la complexité de l'algorithme de Dijkstra. Et quand bien même, très peu ont été capable d'expliquer en quoi l'approche par plus court chemin peut être plus intéressante bien que sa complexité soit en apparence plus élevée.

Question 17 Enfin, cette question cherche à utiliser l'algorithme A^* et demande d'exhiber une heuristique admissible. Un très petit nombre de candidats a eu l'idée de proposer une fonction linéaire en i et j. Aucune copie n'a proposé de solution correcte.