

CONCOURS D'ADMISSION 2026
Composition d'Informatique B, Filière MP

Statistiques

Le tableau ci-dessous indique, pour chaque question du sujet, le pourcentage de copies l'ayant traitée.

Question	1	2	3	4	5	6	7	8
Moyenne*	0,82	0,79	0,91	0,78	0,85	0,71	0,60	0,98
Pourcentage de copies ayant traité la question	99 %	100 %	99 %	98 %	98 %	96 %	93 %	99 %

Question	9	10	11	12	13	14	15	16
Moyenne*	0,57	0,45	0,63	0,56	0,45	0,51	0,46	0,24
Pourcentage de copies ayant traité la question	93 %	89 %	76 %	87 %	78 %	60 %	54 %	18 %

* La moyenne est rapportée sur 1 point, et calculée parmi les copies qui ont traité la question.

Commentaires

Le sujet comporte peu d'écriture de codes : trois questions de programmation en Python (questions 11, 12, 14) et deux questions en SQL (questions 15, 16). Le cœur du sujet repose sur l'analyse d'algorithmes de manipulation des graphes, en lien avec le parcours en profondeur.

Il est très facile d'avoir des intuitions fausses (par exemple, s'il existe un chemin $s \rightarrow u \rightarrow v$ alors tout chemin de s à v passe par u), d'où l'importance de raisonner très rigoureusement. La grande majorité des questions se résolvent par des réponses très courtes. Les invariants sont l'outil éprouvé pour ce type de raisonnement. On insiste sur leur utilisation.

Question 1

Plusieurs copies ne donnent pas les bonnes valeurs pour le tableau sortie.

Question 2

Le raisonnement doit partir du cas général où les tableaux présent et époque sont quelconques et satisfont

l'invariant A2. Certaines copies se restreignent aux tableaux obtenus en question 1.

Question 3

La réponse doit exprimer explicitement que le parcours en profondeur $pp(s)$ visite exactement les sommets accessibles depuis s , et doit réutiliser la question 2.

Question 4

On attend une réponse qui s'appuie sur la question 3 et l'invariant A4. Plusieurs copies affirment que $sortie[v]$ sera affectée à l'intérieur de $pp(u)$, ce qui n'est pas nécessairement le cas s'il existe des chemins de s à v ne passant pas par u .

Question 5

La question est généralement bien traitée. Attention à bien argumenter que les lignes 5 à 7 sont une garde qui est nécessairement satisfaite s'il existe un chemin de s à t (par extension de A3 et A4).

Question 6

Il était possible de résoudre simplement cette question en admettant que la fonction pp préserve les invariants A2-A4. Toutefois, les copies qui l'ont démontré ont été valorisées.

Question 7

De nombreuses copies ne s'attardent pas sur la preuve des invariants A2-A4, en affirmant qu'ils sont préservés d'après la question 6. Il faut toutefois s'assurer que l'arc $u \rightarrow s$ ajouté ligne 20 satisfait également A3 et A4.

Certaines copies indiquent que $lier$ ne modifierait pas époque, ce qui n'est pas le cas puisque la fonction fait appel à pp indirectement.

Question 8

La question est systématiquement bien traitée.

Question 9

Plusieurs réponses sont possibles, notamment en définissant $N(s)$ comme le nombre de sommets accessibles

depuis s , ou comme la longueur du plus long chemin depuis s . Il faut invoquer l'acyclicité pour justifier la décroissance stricte. Certaines copies suggèrent que s'il existe un chemin $s \rightarrow t$ alors t apparaît nécessairement dans $G[s]$, ce qui n'est pas le cas.

Question 10

Cette question est rarement bien traitée.

De nombreuses copies montrent que la complexité est *au plus* exponentielle, ce qui est assez évident, mais la question porte sur la borne inférieure. Il faut pour cela exhiber un graphe sur lequel la complexité est nécessairement exponentielle.

De très nombreuses réponses proposent un graphe arborescent. Mais la complexité dans ce cas sera exponentielle en la hauteur de l'arbre, et non en la taille du graphe.

Le graphe le plus simple qui donne la complexité attendue est composé d'un sommet interne s_0 dont les deux arcs pointent vers un autre sommet interne s_1 dont les deux arcs pointent vers $s_2 \dots$ jusqu'à un sommet s_n libre. D'autres variantes sont possibles.

Question 11

Étonnamment, une proportion significative des réponses ne pense pas à la mémoïsation ou ne l'implémente pas correctement. Pour que la mémoïsation fonctionne, il faut vérifier, à chaque appel sur un sommet, s'il se trouve déjà dans le dictionnaire de mémoïsation, et sinon, calculer et stocker sa sortie avant de la retourner.

Certaines copies oublient de remplir le dictionnaire ou, plus rarement, de vérifier si la valeur cherchée s'y trouve déjà. D'autres vérifient la présence dans le dictionnaire avant d'appeler la fonction. Cela donne des codes très lourds. Il est beaucoup plus naturel de vérifier la présence dans le dictionnaire au début de la fonction. Certaines copies ne mémoïsent pas le cas de base. La complexité reste linéaire mais c'est un peu surprenant.

Question 12

Un code très simple est attendu, mais de nombreuses réponses n'ont pas été suffisamment réfléchies en amont pour donner un algorithme facile à écrire (il n'est pas demandé que la fonction soit efficace).

Il est possible, mais pas spécialement facile, de "parser" le résultat de la fonction $fn(s)$ et de calculer le nombre de sommets y figurant. Si on le fait, il ne faut pas appliquer la fonction len sur le résultat, qui n'est pas nécessairement de type tuple.

Certaines copies calculent $taille(s)$ comme la somme $\sum_{t \in G[s]} taille(t)$ ce qui est faux puisque $taille(s)$ vaut 1 quand $G[s] = []$.

Question 13

Pour cette question, la qualité de la justification est déterminante. Peu de copies apportent des arguments satisfaisants.

Question 14

Beaucoup de copies témoignent d'une mauvaise compréhension du fonctionnement de la fonction `unif`. Par exemple, il n'est pas nécessaire d'utiliser `return unif(P)` à l'intérieur puisque la boucle `while` itère déjà le comportement de la fonction. Il suffit d'indiquer :

```
if s != t:  
    return lier(s,t)
```

Question 15

La question est plutôt bien traitée. Un bon nombre de candidats et candidates maîtrisent suffisamment la syntaxe de SQL.

Question 16

Cette question a été abordée par un petit nombre de copies. Elle est assez subtile. Il ne faut pas chercher à écrire une requête SQL qui retournerait toutes les paires (s, t) , mais remarquer que la question demande simplement d'en retourner une s'il en existe. Il suffit alors d'énumérer quelques cas de base qui couvrent toutes les possibilités (par exemple deux sommets s et t distincts liés au même sommet u).