ÉCOLE POLYTECHNIQUE CONCOURS D'ADMISSION 2024

Épreuve orale d'informatique, Filière MPI

Coefficient (en pourcentage du total des points du concours) **:** 15,7 %

L'épreuve orale d'informatique fondamentale concerne les candidats à l'école polytechnique dans la filière MPI.

Cette année, trente-neuf candidats français et deux candidats étrangers ont participé à l'épreuve. On ne peut que regretter que, sur l'ensemble de ces quarante-et-un candidats, une seule soit une fille.

Les notes des candidats sont comprises entre 7 et 20, avec une moyenne de 12,36 et un écart-type de 3,7. L'histogramme de la figure 1 présente la distribution des notes de l'ensemble des candidats.

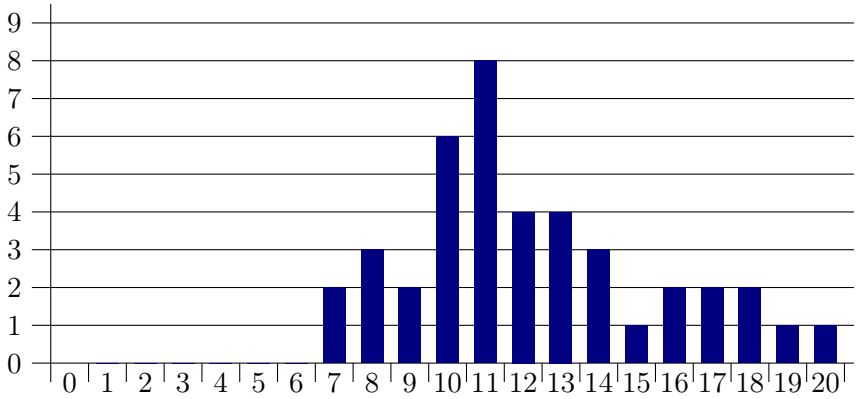


FIGURE 1 – Histogramme des notes de l'épreuve, arrondies à l'entier supérieur

Chaque interrogation durait quarante-huit minutes ; deux minutes étant utilisées par l'examinateur pour s'assurer de l'identité du candidat, lui permettre de signer la feuille d'émargement et lui rappeler les conditions de l'évaluation. Celle-ci se base principalement sur la progression des candidats dans les questions, mais tient également compte de la clarté de leur propos et de l'autonomie dont ils ont fait preuve. Rappelons ici qu'il est important d'être un minimum familier avec le format de l'épreuve, notamment en ce qui concerne l'absence de préparation avant l'oral lui-même : celui-ci commence au moment même où le candidat se voit proposer un sujet.

Le jury a proposé sept sujets originaux, dont deux exemples représentatifs sont présentés en annexe. Chaque sujet débute par un énoncé qui présente un problème d'informatique et introduit ses notations, puis comporte entre sept et onze questions de difficulté globalement croissante. Une ou deux premières questions relativement faciles d'accès permettent aux candidats de vérifier leur compréhension de l'énoncé et de se lancer dans l'interrogation orale. Les questions suivantes, progressivement plus difficiles, occupent la majeure partie du temps de l'interrogation, et visent à permettre de départager les candidats. La plupart des sujets terminent par une ou plusieurs questions considérées comme très difficiles.

En général, il n'est pas attendu des candidats qu'ils traitent l'ensemble des questions dans le temps imparti. Par ailleurs, l'évaluation du candidat tient bien sûr compte de la difficulté des questions qui lui étaient proposées. Ainsi, tel sujet conçu pour être plus difficile d'accès que tel autre est mieux valorisé ; telle question introductive simple mais chronophage est elle aussi valorisée à hauteur du temps qu'elle coûtera aux candidats.

Dans tous les cas, l'objectifs des examinateurs est de permettre au candidat d'avancer au maximum dans le sujet, quitte à lui donner des indications de temps à autre lorsque le candidat semble bloqué. Dans cette perspective, il est indispensable pour les candidats de trouver un équilibre parfois délicat : il convient à la fois d'être suffisamment précis pour convaincre l'examinateur que l'on a compris ce qui se passait, mais de ne pas passer trop de temps sur la question sauf si l'examinateur le demande. Par exemple, il ne faut pas hésiter à donner des éléments de preuve directement à l'oral plutôt que de systématiquement les écrire : si l'examinateur constate que le candidat a manifestement compris ce qu'il avance et que la solution proposée répond à la question, il pourra ainsi inviter le candidat à passer immédiatement à la question suivante.

Les sujets font appel aux différentes compétences nécessaires en science informatique : comprendre des concepts nouveaux, démontrer des résultats théoriques, et construire des solutions techniques telles que des algorithmes.

Le niveau général dont les candidats ont fait preuve pour cette seconde édition de l'épreuve orale d'informatique en voie MPI était excellent, confirmant ainsi les observations effectuées l'an dernier. Les candidats ont montré une très bonne maîtrise des algorithmes, structures de données, ainsi que d'éléments plus théoriques du programme tels que la logique et la théorie des langages. Le plus souvent, ils ont su s'appuyer sur cette maîtrise pour réfléchir de manière pertinente à des problèmes difficiles. Ainsi, les candidats ayant eu au moins 12/20, soit la moitié des candidats, ont proposé une prestation d'excellente facture.

Cette réitération du constat extrêmement satisfaisant déjà formulé en 2023 confirme tout le bien-fondé de la mise en place de la filière MPI pour l'apprentissage de l'informatique.

Enfin, voici quelques conseils méthodologiques à destination des candidats. Tout d'abord, afin de mieux manipuler les objets manipulés pendant l'oral, il est souvent très pertinent de faire des dessins et de regarder de petits cas : en sciences, intuition et rigueur sont toutes deux indispensables, et il ne faudrait pas sacrifier la première à la seconde. Ainsi, trop de candidats ont commencé à s'embourber en utilisant des notations parfois absconses, jusqu'au moment où l'examinateur leur a demandé de dessiner le graphe ou la fonction qu'ils étudiaient, ce qui les a très souvent débloqués. De même, face à une question fermée, plusieurs candidats sont partis bille en tête sur une conjecture formulée, semble-t-il, au hasard (puisqu'elle était fausse une fois sur deux en moyenne) au lieu de commencer par manipuler de petits exemples.

Par ailleurs, comme cela était déjà indiqué l'an dernier, il est évidemment nécessaire de connaître parfaitement son cours. Cette année, par exemple, plusieurs candidats se sont fourvoyés sur la complexité de l'algorithme de Dijkstra, ou encore sur le sens dans lequel doivent être effectuées les réductions visant à démontrer que tel problème est NP-difficile. De tels manquements, heureusement fort rares, ont eu pour double conséquence négative de faire perdre du temps aux candidats et de forcer les examinateurs à les sanctionner numériquement.

En annexe, nous présentons, corrigeons et commentons deux exemples de sujets représentatifs de cette année :
— Algorithme du tri lent
— Codage par couleurs

Algorithme du tri lent

On souhaite trier un tableau A de longueur n en utilisant l'algorithme de TriLent présenté ci-dessous. L'objectif de ce problème est de démontrer que l'algorithme est correct et d'évaluer certaines statistiques liées à sa complexité.

```
1 Fonction TriLent(A_{1...n}):
2   pour i = 1, 2, ..., n :
3     pour j = 1, 2, ..., n :
4       si A_i < A_j : échanger A_i et A_j
```

Question 1. Quelle est, en fonction de n, la complexité de l'algorithme de TriLent, en nombre de comparaisons, dans le pire cas ? dans le meilleur cas ?

Question 2. Un algorithme de tri est *stable* si deux objets A_i et A_j égaux pour notre fonction de comparaison et pour lesquels $i < j$ sont envoyés en deux positions u et v telles que $u < v$. Le TriLent est-il stable ?

Question 3. Soit i un entier tel que $1 \leqslant i \leqslant n$. Démontrer que, juste après avoir effectué i fois la boucle **pour** des lignes 3 et 4, l'élément A_i est l'élément maximal du tableau, et le sous-tableau formé des entrées $A_1, A_2, \ldots, A_i$ est trié dans l'ordre croissant.

Question 4. Démontrer que le TriLent permet effectivement de trier le tableau fourni en entrée.

Question 5. On suppose que les n éléments du tableau fourni en entrée sont deux à deux distincts. Donner le plus petit nombre possible d'échanges que le TriLent peut être amené à effectuer.

Question 6. Le tableau fourni en entrée est une permutation des entiers $1, 2, \ldots, n$ choisie uniformément au hasard. Démontrer que le nombre moyen d'échanges qu'effectuera le TriLent est égal à $n(n-1)/4 + 2H_n - 2$, où

$$H_n = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}$$

est le $n^{\text{ème}}$ nombre harmonique.

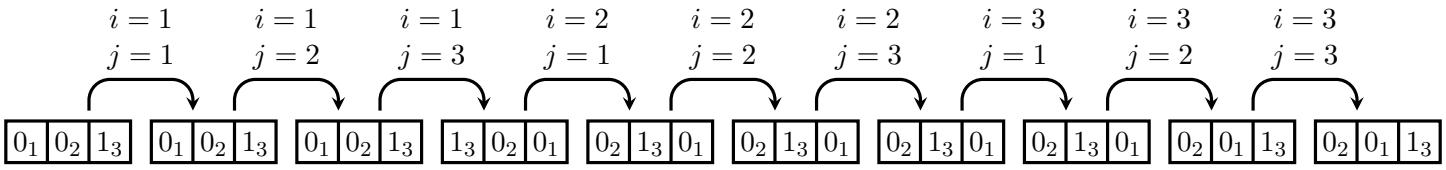
Question 7. Donner le nombre maximal d'échanges que l'algorithme de TriLent peut être amené à effectuer.

Éléments de correction et attentes de l'examinateur

Question 1. On a deux boucles imbriquées, chacune faisant prendre à sa variable (i ou j) n valeurs distinctes. À chaque fois, on effectue une comparaison. Cela fait donc n^2 comparaisons en tout, et ce quel que soit le tableau proposé en entrée.

Cette question a pour but de mettre le candidat en confiance et de lui donner une première occasion de s'approprier les notations : A_i désigne, au moment où est effectuée la comparaison $A_i < A_j$, le $i^{\text{ème}}$ élément du tableau. Elle permet aussi de vérifier que le candidat ne va pas annoncer sans avoir réfléchi au préalable que « le meilleur cas est celui où A est trié », phrase certes vraie mais qui suggère une incompréhension.

Question 2. Imaginons un tableau à trois éléments, dont deux sont égaux pour notre fonction de comparaison : il s'agit, par exemple, du tableau $0_1, 0_2, 1_3$, où l'on a indiqué à côté de chaque objet sa position initiale. Au cours de l'algorithme, en neuf occasions, on compare deux cases du tableau A, et on les échange au besoin, comme suit :

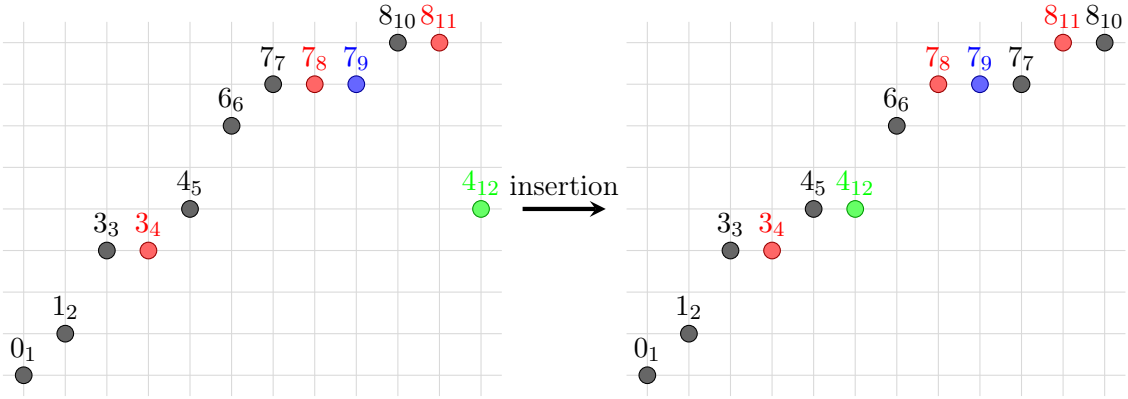


L'élément 0_2, initialement en deuxième position, est passé à gauche de l'élément 0_1, qui lui est égal pour notre fonction de comparaison mais était initialement situé en première position. Le TriLent n'est donc pas stable. [1]

Cette question a trois objectifs : tout d'abord, aider le candidat à se familiariser avec l'algorithme qu'ils devra ensuite étudier ; ensuite, s'assurer qu'il est capable d'exécuter calmement mais précisément un algorithme sur une entrée simple qu'il aura lui-même choisie ; enfin, vérifier qu'il est capable de représenter clairement les objets qu'il manipule, tels que des quantités censées être égales pour notre fonction de comparaison mais distinctes en pratique.

Question 3. Lors du premier passage dans la boucle de la ligne 2, alors que i vaut 1, on compare successivement A_1 avec toutes les cases du tableau, de manière à faire croître A_1 dès que l'on rencontre un nouveau maximum. Après ce premier passage, lors duquel on a effectué une fois la boucle **pour** des lignes 3 et 4, l'objet A_1 est bien maximal, et le tableau $A_1, \ldots, A_1$, formé d'un seul élément, est nécessairement trié dans l'ordre croissant. On dira qu'il s'agit là de la *phase 1* de l'algorithme ; toutes les opérations effectuées ensuite formeront la *phase 2*.

Par la suite, après que l'on a effectué i fois cette boucle, l'algorithme consiste à comparer chaque case du tableau avec la $i+1^{\text{ème}}$ case. Ainsi, lorsque j varie de 1 à $i+1$, on effectue en fait l'insertion de la valeur A_{i+1} dans le sous-tableau trié $A_1, A_2, \ldots, A_i$: dès que l'on repère une case du tableau, disons A_j, telle que $A_j > A_{i+1}$, on échange A_j et A_{i+1}, de sorte que la $i+1^{\text{ème}}$ case nous sert en fait de mémoire tampon pour ensuite décaler d'une case vers la droite le sous-tableau $A_j, A_{j+1}, \ldots, A_i$. Ces cases seront décalées *dans leur ensemble*, mais pas forcément individuellement, dans le cas où le tableau contient des doublons : quand plusieurs cases consécutives étaient égales et dépassaient la valeur à insérer, c'est la plus à gauche d'entre elles qui est passée à droite, comme illustré ci-dessous ; chaque objet est représenté par sa valeur et, en indice, sa position d'origine.



1. On aurait pu aller plus vite en s'arrêtant sitôt le tableau $0_2, 1_3, 0_1$ obtenu, après avoir comparé A_i et A_j lorsque $(i,j) = (2,1)$, car alors $A_1 = 0_2$ ne sera plus jamais déplacé.

Une fois cette étape franchie, A_{i+1} est la valeur maximale d'un sous-tableau trié qui contenait le maximum de A, donc on ne l'échangera plus jamais lorsque $j \geqslant i+2$. Par conséquent, une fois notre $i+1^{\text{ème}}$ exécution de la boucle **pour** effectuée, le sous-tableau $A_1, A_2, \ldots, A_{i+1}$ est bien trié dans l'ordre croissant, et se termine avec la valeur maximale du tableau A.

Cette question vise à affiner la compréhension qu'a le candidat de l'algorithme de TriLent et de vérifier comment il entreprend de se forger des intuitions qui mèneront à cette compréhension, en particulier la disjonction entre phases 1 et 2.

À défaut de la description donnée ci-dessus, il était aussi possible d'exhiber directement puis de démontrer un invariant de boucle, ce qui pouvait néanmoins s'avérer piégeux : on a tôt fait de proposer un invariant presque correct mais qui sera difficile à corriger, ou bien de répondre à la question mais sans avoir acquis de compréhension globale sur le fonctionnement de l'algorithme.

Question 4. Il suffit d'appliquer la question précédente au cas où $i = n$: une fois l'algorithme terminé, le tableau est trié.

Cette question a pour seuls buts d'aboutir à une conclusion intermédiaire raisonnable, ici la correction de l'algorithme de TriLent, et de redonner un coup de fouet aux candidats, tout heureux d'avoir résolu une question en cent fois moins de temps que la précédente.

Question 5. Sans perte de généralité, le tableau à trier est une permutation des entiers $1, 2, \ldots, n$. Juste après que la boucle **pour** des lignes 3 et 4 a été exécutée i fois, l'entier n se trouve donc en case i du tableau. En particulier, il a changé au moins $n-1$ fois de case.

Réciproquement, si l'on est parti du tableau $n, 1, 2, \ldots, n-1$, notre étude effectuée en question 3 indique qu'aucun échange n'est effectué lors de la phase 1, tandis que tous les échanges ultérieurs ont lieu juste après que l'on a comparé A_{i-1} et A_i. Ainsi, l'algorithme de TriLent s'est effectivement contenté de $n-1$ échanges pour trier notre tableau.

Il s'agit là de la première question manifestement ouverte, pour laquelle le candidat n'a, *a priori*, aucune idée de la réponse, si ce n'est qu'elle sera comprise entre 0 et n^2. Y répondre nécessite non seulement d'avoir compris comment l'élément maximal du tableau se déplacerait, mais aussi de regarder des exemples de tableaux susceptibles d'être simples, tels que le tableau déjà trié dans l'ordre croissant. De manière peut-être surprenante, celui-ci n'est pas le tableau optimal, car on effectue $n-1$ échanges inutiles lors de la phase 1, même si la phase 2 est ensuite très économe. C'est alors que le candidat doit être amené à remplacer le tableau trié par le tableau obtenu à l'issue de la phase 1.

Question 6. Il convient ici de considérer séparément les phases 1 et 2, puis de trouver un invariant qui nous permettra d'estimer le nombre d'échanges effectués ; on va en fait démontrer que le nombre d'inversions du tableau A, c'est-à-dire le nombre de couples (k, ℓ) tels que $k < \ell$ et $A_k > A_\ell$, augmente de 1 à chaque échange de la phase 1, et diminue de 1 à chaque échange de la phase 2.

Regardons, par exemple, ce qui se passe en phase 2. Lorsque l'on échange deux cases A_i et A_j et que $i \geqslant 2$, on insère en fait la case A_i au sein du sous-tableau trié $A_1, A_2, \ldots, A_{i-1}$. Par conséquent, si l'on note A le tableau avant l'échange et A' le tableau après l'échange, les inversions de A' sont :

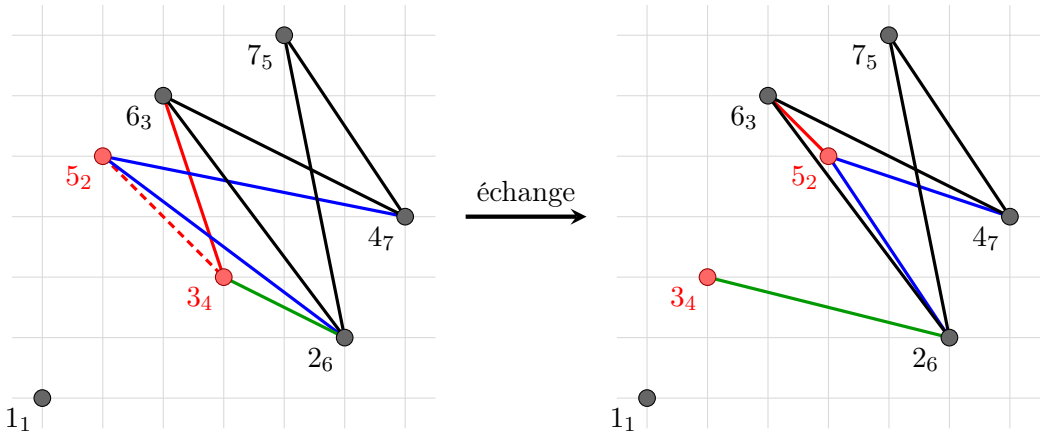
- ▷ les couples (u, i) tels que $j < u < i$; il y en a a ;
- ▷ les couples (i, v) tels que $i < v$ et $A_j = A'_i > A'_v = A_v$; il y en a b ;
- ▷ les couples (j, v) tels que $i < v$ et $A_i = A'_j > A'_v = A_v$; il y en a c ;
- ▷ les couples (u, v) tels que $\{u, v\} \cap \{i, j\} = \emptyset$, $u < v$ et $A_u > A_v$; il y en a d.

Par ailleurs, les inversions de A sont :

- ▷ les couples (u, i) tels que $j \leqslant u < i$; il y en a $a+1$;
- ▷ les couples (i, v) tels que $i < v$ et $A_i > A_v$; il y en a c ;
- ▷ les couples (j, v) tels que $i < v$ et $A_j > A_v$; il y en a b ;
- ▷ les couples (u, v) tels que $\{u, v\} \cap \{i, j\} = \emptyset$, $u < v$ et $A_u > A_v$; il y en a d.

Comme annoncé, échanger A_i et A_j a donc permis de baisser de 1 le nombre d'inversions du tableau considéré ; cette situation est illustrée ci-après : les deux points échangés sont dessinés en rouge, et l'inversion qu'ils formaient,

représentée par une ligne rouge hachurée, a disparu. Les autres inversions ont subsisté, parfois en changeant de point d'accroche ; les quatre différents types d'inversions énumérés ci-dessus (il y en a respectivement a ou $a+1$, b, c et d) sont représentés en rouge, bleu, vert et noir ; chaque objet est représenté par sa valeur et, en indice, sa position d'origine.



Or, le nombre moyen d'inversions d'une permutation choisie uniformément au hasard est $n(n-1)/4$. En effet, pour toute paire (i,j) telle que $1 \leqslant i < j \leqslant n$, si l'on note $\sigma_{i,j}$ la transposition qui échange i et j, on remarque que la fonction $\pi \mapsto \pi \circ \sigma_{i,j}$ est une involution de $\mathfrak{S}_n$, qui partitionne $\mathfrak{S}_n$ en orbites de taille 2. De surcroît, $\pi(i) < \pi(j)$ si et seulement si $\pi(\sigma_{i,j}(i)) > \pi(\sigma_{i,j}(j))$. Ainsi, le couple (i,j) est une inversion pour une des deux permutations de chaque orbite $\{\pi, \pi \circ \sigma_{i,j}\}$: au total, les $n!$ permutations ont donc $n! \times \binom{n}{2}/2 = n! \times n(n-1)/4$ inversions, ce qui nous fait bien une moyenne de $n(n-1)/4$ inversions par permutation.

On étudie ensuite la phase 1 ; notre invariant se montre en utilisant un raisonnement analogue à celui utilisé en phase 2 : si on annulait notre échange, on supprimerait une inversion, donc l'échange a bien créé une inversion. Ensuite, lorsque $i = 1$ et $2 \leqslant j \leqslant n$ et que l'on compare A_1 et A_j, on sait déjà que A_1 a été redéfini comme $\max\{\pi(1), \pi(2), \ldots, \pi(j-1)\}$, tandis que $A_j = \pi(j)$. On procède donc à un échange lorsque $\pi(j) = \max\{\pi(1), \pi(2), \ldots, \pi(j)\}$, ce qui arrive avec probabilité $1/j$. Au total, lors de la phase 1, on effectue donc $1/2 + 1/3 + \cdots + 1/n = H_n - 1$ échanges en moyenne ; on devra ensuite effectuer un échange supplémentaire pour se débarrasser de l'inversion que notre échange vient de créer.

En conclusion, l'algorithme de TriLent requiert bien $n(n-1)/4 + 2(H_n - 1)$ échanges en moyenne.

Cette question n'est manifestement guère faisable sans indication de la part de l'examinateur. Elle a pour but de tester le candidat sur des questions de nature probabiliste, qui forment un pan important de l'informatique, et de voir, dans un premier temps, comment il tente de se débrouiller face à une question fort ardue en apparence (et en réalité).

En particulier, une fois donnée l'indication, en pratique indispensable, selon laquelle il fallait vérifier que chaque échange ajoutait ou supprimait une inversion du tableau à trier, il fallait ensuite modéliser aussi simplement que possible les différents phénomènes observés. Ici, avoir compris que les phases 1 et 2 différaient radicalement, mais aussi savoir utiliser simplement des bijections et la linéarité de l'espérance était nécessaire pour espérer répondre à la question. De même, pour démontrer la véracité de l'indication donnée par l'examinateur, il était beaucoup plus simple de dessiner la permutation modifiée et d'en représenter graphiquement les inversions susceptibles de disparaître ou d'apparaître, plutôt que de tenter une disjonction de cas sans aucun apport de l'intuition. Procéder ainsi permettait notamment de traiter un seul des cas, disons celui de la suppression d'une inversion, pour ensuite convaincre à l'oral l'examinateur que l'autre cas se traiterait de même.

Question 7. Sans perte de généralité, on peut toujours supposer que le tableau à trier contient des valeurs deux à deux distinctes ; si tel n'était pas le cas, et quitte à séparer artificiellement deux valeurs considérées comme égales, on ne diminuerait pas le nombre d'échanges effectués.

Une fois ce cadre posé, on aimerait bien pouvoir maximiser simultanément le nombre d'échanges effectués lors des phases 1 et 2 ; c'est impossible. En effet, réaliser les $n-1$ échanges possibles lors de la phase 1 nécessite de partir du tableau trié. Notre idéal étant inatteignable, on note donc k le nombre d'échanges effectués en

phase 1 puis on s'intéresse, l'entier k étant fixé, au nombre maximal d'inversions, disons inv_k, que notre tableau obtenu en fin de phase 1 a pu avoir : le nombre maximal d'échanges effectués dans l'ensemble des phases 1 et 2 sera $\max\{k + \mathsf{inv}_k : 0 \leqslant k \leqslant n-1\}$.

Les k valeurs qu'a prises la case A_1 du tableau au cours de la phase 1 avant que cette case ne prenne la valeur maximale se retrouvent à former une sous-suite croissante au sein du tableau obtenu en fin de phase 1, et que l'on notera B. Si l'on note I l'ensemble des numéros de cases occupés par ces valeurs au sein du tableau B, aucun couple $(i, j) \in I^2$ ne forme une inversion de B. Ainsi, le tableau B compte au plus $n(n-1)/2 - k(k-1)/2$ inversions, et l'algorithme de TriLent a effectué au plus

$$\frac{n(n-1)}{2} - \frac{k(k-1)}{2} + k = \frac{n(n-1) - k(k-3)}{2} \leqslant \frac{n^2 - n + 2}{2}$$

échanges.

Réciproquement, si l'on choisit $k = 1$, on souhaite précisément que B soit le tableau trié dans l'ordre décroissant, que l'on a pu obtenir en partant du tableau $n-1, n, n-2, n-3, \ldots, 2, 1$. En conclusion, le nombre maximal d'échanges auquel procède l'algorithme de TriLent est $(n^2 - n + 2)/2$.

Cette question a pour but de résister même aux candidats les plus aguerris ; aucune indication n'était donc prévue pour les guider vers la solution. Elle nécessite tout d'abord de se rendre compte que l'on peut légitimement supposer que l'on travaille sur une permutation, puis d'avoir bien compris que compter les inversions était la bonne manière de compter les échanges effectués par le TriLent, et enfin de renoncer à maximiser simultanément deux quantités liées, pour fixer l'une et maximiser l'autre.

Codage par couleurs

On considère des graphes orientés, supposés sans boucle. On cherchera dans ces graphes un chemin d'une longueur spécifique (sans imposer de point de départ et d'arrivée), où la *longueur* d'un chemin est le nombre de sommets qui apparaissent dans le chemin. Sauf mention explicite du contraire, on considère toujours des chemins *simples*, c'est-à-dire qui ne passent pas deux fois par le même sommet.

Question 1. Y a-t-il des graphes fortement connexes arbitrairement grands sans chemin de longueur 4 ?

Question 2. Proposer un algorithme naïf pour déterminer, étant donné un graphe G et un entier $k \in \mathbb{N}$, si G contient un chemin de longueur k. Quelle est la complexité de cet algorithme ?

Question 3. Dans cette question seulement, on s'intéresse à des chemins *non nécessairement simples*. Proposer un algorithme efficace pour déterminer, étant donné un graphe G et un entier $k \in \mathbb{N}$, si G contient un tel chemin de longueur k.

Question 4. Dans cette question seulement, on suppose que les sommets du graphe d'entrée G portent chacun une couleur parmi l'ensemble $\{1, \dots, k\}$, et on souhaite savoir si G admet un chemin *multicolore* de longueur k, c'est-à-dire un chemin $v_1, \dots, v_k$ où les couleurs des sommets sont deux à deux distinctes.
Proposer un algorithme pour résoudre ce problème, et en expliciter la complexité.

Pour améliorer le temps d'exécution de l'algorithme de la question 2, on va concevoir un algorithme *probabiliste*. Un tel algorithme a la possibilité de tirer au hasard certaines valeurs au cours de son exécution ; et on regarde, sur chaque entrée, quelle est la probabilité que l'algorithme réponde correctement, en fonction de ces tirages aléatoires.
On veut résoudre le problème suivant : étant donné un graphe G et un entier k, on veut savoir si G a un chemin de longueur k. On considère d'abord l'algorithme (1) que voici. On répète M fois l'opération suivante (où M sera déterminé ensuite) : tirer k sommets au hasard et vérifier si ces sommets forment un chemin. On répond OUI si l'un de ces tirages est réussi et NON dans le cas contraire.

Question 5. On suppose que le graphe G a n sommets et contient précisément $1 \leqslant c \leqslant n^k$ chemins de longueur k. Exprimer la probabilité que l'algorithme (1) réponde OUI, en fonction de M, de k, de n, et de c.

Question 6. Si on suppose que G n'a pas de chemin de longueur k, que répond l'algorithme (1) ?

Question 7. Pour $M = n^k$, montrer que l'algorithme répond correctement dans les deux cas avec probabilité au moins $1/2$.

Question 8. Quel est le temps d'exécution de l'algorithme (1) pour ce choix de M ? Commenter.

On considère à présent l'algorithme (2) dont le principe est le suivant. On répète M fois l'opération suivante (où M sera déterminé ensuite) : tirer un ordre total aléatoire $v_1 < \cdots < v_n$ sur les sommets de G, retirer les arêtes (u, v) où on a $u > v$, et vérifier si le graphe résultant $G_<$ a un chemin de longueur k (d'une manière qui reste à déterminer).

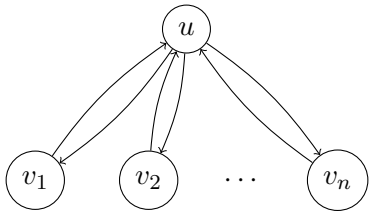
Question 9. Quelle propriété ont les graphes $G_<$ construits par cet algorithme ? Comment exploiter cette propriété pour trouver efficacement les chemins de longueur k ?

Question 10. Proposer un M qui garantisse que cet algorithme réponde correctement avec une probabilité $\geqslant 1/2$. Quelle complexité obtient-on ? Commenter.

Question 11. En utilisant les chemins multicolores, proposer un algorithme probabiliste qui répond correctement avec probabilité au moins $1/2$ et s'exécute en $O((2e)^k \times (n + m))$ sur un graphe à n sommets et m arêtes.

Éléments de correction et attentes de l'examinateur

Question 1. La réponse à cette question est positive, comme l'ont déterminé tous les candidats (plus ou moins rapidement). C'est le cas par exemple de graphes ayant une forme d'étoile :



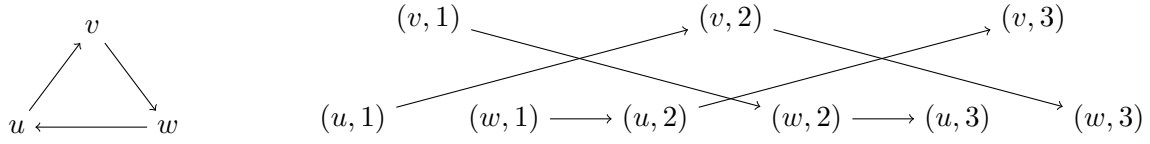
Le but de cette question est de rappeler la notion de graphes orientés fortement connexes en lien avec le sujet, et d'attirer l'attention sur la définition peu standard de la longueur des chemins que le sujet adopte (afin de simplifier certains calculs par la suite). La question permet aussi de remarquer que ce qui est étudié dans le sujet ensuite, à savoir la recherche de chemins, n'est pas une tâche triviale : les chemins tels que définis dans le sujet n'existent pas forcément, même quand le graphe est grand, et même en imposant qu'il soit fortement connexe.

Question 2. La réponse attendue est simplement de dire que l'on teste toutes les combinaisons possibles : pour n sommets et m arêtes, il y a n^k combinaisons de k sommets à tester, pour lesquelles il faut vérifier qu'il y a bien l'arête demandée. Si on suppose que le graphe est représenté par des listes d'adjacence, alors on obtient une complexité de $O(n^k \times m)$. Une autre façon de répondre à cette question est d'utiliser une approche de type retour sur trace, ou bien une exploration en largeur qui marque les sommets en cours de visite et s'interdit de les réutiliser (mais ne marque pas les sommets où la visite est finie, ceux-ci pouvant être visités à nouveau). La complexité asymptotique sera alors similaire, même si en pratique le retour sur trace est bien sûr préférable à l'énumération naïve des combinaisons : en effet il n'explore que les successeurs de chaque sommet, et il abandonne quand une solution partielle ne peut pas être étendue.

Cette question vise à vérifier que le candidat est capable de proposer rapidement un algorithme naïf pour un problème, ce qui doit être un préalable à toute tentative d'optimisation. En effet, si le graphe d'entrée est très petit, un algorithme naïf sera suffisant et sera bien plus facile à implémenter. Les candidats ne doivent pas hésiter, quand une approche naïve est demandée, à proposer des algorithmes peu efficaces : les examinateurs sauront leur préciser si c'est quelque chose de plus efficace qui est attendu.

Question 3. Cette question, qui se voulait facile, s'est avérée en réalité être très classante. L'argument que nous attendions consistait à construire une sorte de produit cartésien. Étant donné le graphe $G = (V, E)$, on crée le graphe $G \times [k]$ dont les sommets sont $V \times \{1, \ldots, k\}$ et les arêtes sont $\{((u, i), (v, i+1)) \mid (u, v) \in E, 1 \leqslant i < k\}$. On teste ensuite simplement s'il existe un chemin de la première copie à la dernière. Autrement dit, on cherche à tester s'il y a un chemin d'un sommet de la forme $\{(u, 1) \mid u \in V\}$ à un sommet de la forme $\{(u, k) \mid u \in V\}$. Ceci peut se faire avec un parcours en largeur par exemple. La complexité est en $O((n + m) \times k)$ si on suppose à nouveau que le graphe est représenté avec des listes d'adjacence.

La construction du graphe produit est illustrée ci-dessous, avec un graphe G à gauche et le graphe $G \times [3]$ à droite :



D'autres approches sont possibles en ne construisant le graphe produit qu'implicitement. Enfin, il est possible d'adopter une approche en complexité linéaire (c'est-à-dire $O(n + m)$) en distinguant deux cas : soit le graphe contient un cycle orienté (ce qui peut se vérifier en temps linéaire) et alors il y a toujours un chemin non-simple de longueur k pour n'importe quel k, soit le graphe est acyclique et on peut utiliser l'algorithme pour les graphes acycliques qui est l'objet de la question 9.

Question 4. Les candidats ont tous compris (spontanément, ou avec indication) qu'un chemin multicolore est nécessairement simple, par application du principe des tiroirs. L'objectif était alors de se rendre compte que, grâce à ce fait, on pouvait utiliser une variante de la question précédente. Là encore, cette question était finalement plus classante que prévu.

Pour répondre à la question, on crée 2^k copies du graphe $G = (V, E)$, c'est-à-dire qu'on considère le graphe $(V \times 2^k, E')$ avec E' défini comme suit : pour chaque arête $(u, v) \in E$, en notant c la couleur de v, pour chaque $S \subseteq 2^k$, on crée $((u, S), (v, S \cup \{c\}))$.

On cherche ensuite à savoir s'il existe un chemin depuis un sommet de la forme $(u, \{c\})$ pour c la couleur de u à un sommet de la forme $(v, 2^k)$. Si un tel chemin existe, c'est forcément un chemin dans G (par projection sur la première composante), et par définition des arêtes il a forcément visité un sommet de chaque couleur. En particulier le chemin est forcément simple. Donc cet algorithme identifie bien correctement si G contient un chemin multicolore.

La complexité est linéaire en la taille du graphe produit c'est-à-dire $O((n + m)2^k)$.

Question 5. Pour les candidats qui ont pu l'atteindre, cette question n'a guère posé de difficulté. Il y a n^k tirages possibles et c de ces tirages réussissent. Donc chaque tirage a une probabilité de c/n^k de réussir, et une probabilité de $1 - c/n^k$ d'échouer. On effectue M tirages indépendants, donc avec probabilité $(1 - c/n^k)^M$ aucun tirage ne réussit et l'algorithme répond NON (la réponse incorrecte) ; et avec probabilité $1 - (1 - 1/c^k)^M$ un tirage au moins réussit et l'algorithme répond OUI (la réponse correcte).

Question 6. Quand le graphe n'a pas de chemin, l'algorithme répond toujours NON (et c'est la réponse correcte).

Cette question est sans difficulté et sert juste à vérifier la compréhension du comportement de cet algorithme probabiliste.

Question 7. Si le graphe d'entrée n'a pas de chemin, il n'y a rien à montrer par la question précédente. S'il y en a un, il faut montrer que la probabilité d'échec est d'au plus 1/2. C'est-à-dire montrer que $(1 - 1/n^k)^{n^k} \leqslant 1/2$. Excluons le cas trivial où $n^k = 1$. Le logarithme du membre gauche vaut $n^k \ln(1 - 1/n^k)$, ce qui par concavité du logarithme est plus petit que $n^k \times (-1/n^k)$, donc plus petit que -1. Or le logarithme du membre droit vaut $-\ln 2$. Comme $e > 2$ on a bien $-\ln 2 \geqslant -1$. On en déduit donc que la probabilité d'échec est au plus $1/2$.

Question 8. L'algorithme (1) s'exécute en $O(M \times m)$. Donc pour le M indiqué on retrouve la même complexité qu'en question 2. On attend du candidat qu'il remarque que l'algorithme (1) n'est donc pas très intéressant en tant que tel.

Question 9. Cette question commence à nécessiter de l'initiative de la part du candidat. Il faut en effet remarquer que les graphes $G_<$ sont toujours acycliques. En effet, l'ordre $<$ peut servir de tri topologique ; autrement dit un cycle donnerait un cycle dans l'ordre total ce qui par transitivité impliquerait qu'il n'est pas asymmétrique.

Or, on peut résoudre efficacement sur des graphes acycliques le problème de savoir s'ils admettent un chemin de longueur k. En effet, on considère les sommets de $G_<$ dans l'ordre inverse de $<$, et on calcule pour chaque sommet v la longueur $l(v)$ du plus long chemin qui part de v. Si v n'a aucune arête sortante, cette longueur est de 0. Si v a des arêtes sortantes vers $w_1, \ldots, w_k$, les valeurs $l(w_1), \ldots, l(w_k)$ étant déjà calculées car $v < w_i$ pour tout i, on pose $l(v) = 1 + \max_i l(w_i)$.

Cet algorithme s'exécute en temps linéaire, c'est-à-dire en $O(n + m)$ sur un graphe à n sommets et m arêtes. On note en particulier que cette complexité ne dépend pas de k.

Question 10. Encore une fois, l'algorithme est biaisé vers le faux : si le graphe d'entrée n'a pas de chemin il répond toujours NON (correctement). Donc il suffit de majorer la probabilité que l'algorithme réponde NON alors que le graphe contient un chemin de longueur k. Or, si le graphe d'entrée a un chemin de longueur k, alors il est identifié si chacune de ses $k - 1$ arêtes sont conservées, et c'est le cas si et seulement si les sommets sont triés dans le bon ordre par l'ordre total qu'on a tiré. Ainsi chaque tirage a une probabilité de $1/k!$ d'identifier le chemin, et la probabilité d'échec est d'au plus $(1 - 1/k!)^M$.

On peut donc prendre $M = k!$ pour obtenir une probabilité d'échec constante, pour la même raison que pour l'algorithme (1). La complexité de l'algorithme est alors de $O(k! \times (n + m))$ si on suppose qu'on peut tirer un

ordre aléatoire en temps $O(m)$. C'est le cas, par exemple en $O(n)$ avec l'algorithme de Fisher-Yates pour tirer une permutation aléatoire, mais ces détails n'étaient pas demandés au candidat.

On remarque que la complexité obtenue, qui est de $O(k! \times (n+m))$ est bien meilleure que $O(n^k)$ du moment que k n'est pas trop grand.

Question 11. Dans cette dernière question, il faut davantage de créativité pour espérer répondre. L'idée est de répéter M fois l'opération suivante : tirer un k-coloriage aléatoire du graphe, et vérifier si on trouve un chemin multicolore de longueur k avec l'algorithme de la question 4, en temps $O(2^k(n+m))$.

Quand il n'y a pas de chemin de longueur k, alors il n'y a pas de tel chemin qui soit multicolore et l'algorithme répond correctement NON.

Quand il y a un tel chemin, la probabilité de le trouver est la probabilité que ce chemin soit effectivement multicolore dans le coloriage tiré. Il y a k^k coloriages possibles dont $k!$ sont corrects. Or, on sait par la formule de Stirling que $k! = \Omega((k/e)^k)$. Donc la probabilité qu'un chemin de longueur k donné soit multicolore est de $\Omega(1/e^k)$.

Ainsi, si on tire $M = e^k$ coloriages aléatoires et qu'on cherche à chaque fois un chemin multicolore, on aura à nouveau une probabilité constante de succès si le graphe contient un chemin de longueur k. On peut plus précisément garantir une probabilité de $1/2$ au moins, en multipliant par une constante le nombre de tirages. Le temps d'exécution sera alors bien de $O(e^k 2^k(n+m))$, à savoir $M = e^k$ tirages aléatoires (à une constante multiplicative près) puis $O(2^k(n+m))$ pour chaque tirage.

Cette technique du *codage par couleur*, introduite par Alon, Yuster, et Zwick, est utilisable pour d'autres problèmes : voir par exemple `https://en.wikipedia.org/wiki/Color-coding`.