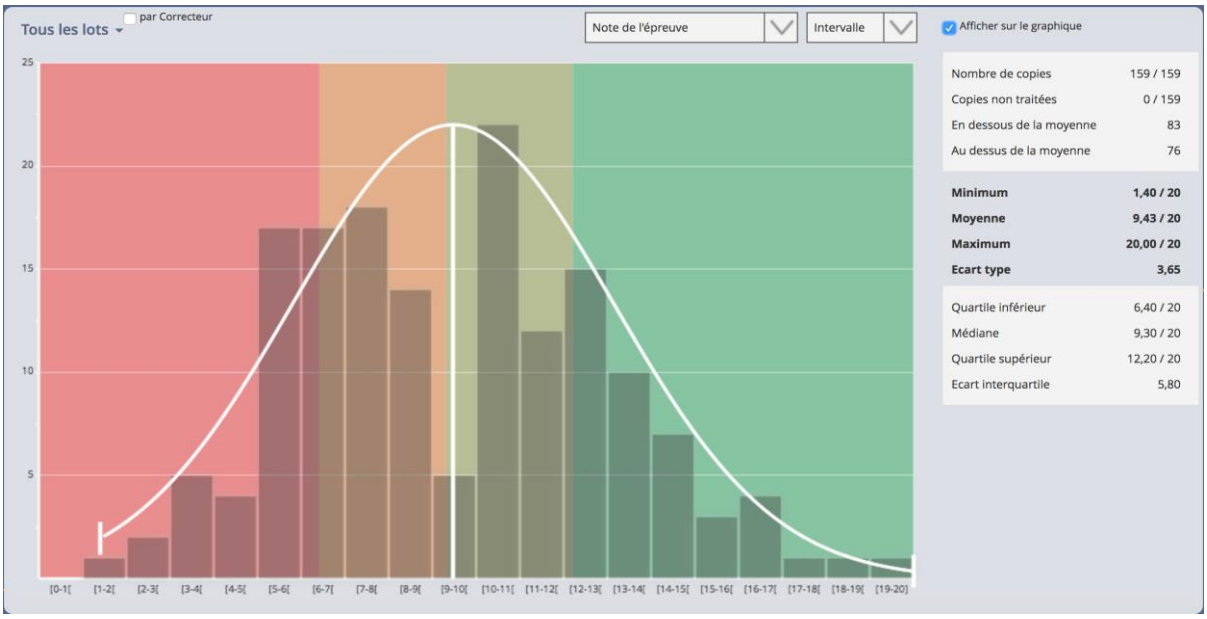
Rapport sur l'épreuve d'Algorithmique et Informatique

Il y avait cette année 159 candidats. Les notes de l'épreuve s'étalent de 1,4 à 20 sur 20 (un candidat ayant réussi brillamment l'épreuve dans sa totalité), la moyenne se situe à 9,43, la médiane à 9,30 et l'écart-type à 3,65.

Les modalités de l'épreuve ont été inchangées par rapport aux années précédentes. D'une durée de 45 minutes, l'épreuve consiste au codage en python d'un algorithme, via l'usage de Q.C.M., de code à compléter ou corriger, ainsi que de code à concevoir.

La réussite à l'épreuve a été inégale : un bon nombre de copies se sont avérées de très bon niveau, pointant notamment que le sujet, malgré sa difficulté, était accessible et sa longueur adaptée. Mais à contrario, de nombreuses copies se sont révélées insuffisantes, mettant en évidence une nette hétérogénéité dans le niveau ou la préparation des candidats.



Le sujet était ambitieux ; il portait sur la recherche d'un chemin dans un labyrinthe. L'approche, un classique du domaine, s'effectuait par un parcours en profondeur (en termes techniques ; en anglais backtracking), dont le principe était détaillé par l'énoncé et l'implantation illustrée par des exemples éclairants ; elle était rendue accessible grâce à l'usage de questions de type QCM ou de code à compléter, ainsi que de la progressivité dans les questions posées.

Les compétences suivantes ont été évaluées par l'épreuve : <<Concevoir un algorithme répondant à un problème algorithmique clairement posé >>, <<Comprendre un code>>, <<modifier un code donné>> et <<compléter un code donné>>.

La principale difficulté du sujet résidait d'abord dans la compréhension de l'approche utilisée et de l'implantation du problème, qui était expliquée par l'énoncé, puis dans sa retranscription en code Python.
Du point de vue technique, l'implantation manipulait des tableaux, que ce soit pour représenter le labyrinthe à l'aide d'une matrice carrée d'entiers (par un tableau numpy bidimensionnel), ou un chemin par une liste des coordonnées des positions empruntées dans le labyrinthe (liste de listes de deux entiers). L'implantation nécessitait le parcours de tableaux, l'accès, la modification à un élément d'un tableau, ainsi que l'ajout et la suppression d'éléments en fin de liste. Les commandes nécessaires aux tableaux numpy étaient données et illustrées sur des exemples.
La réussite à l'épreuve nécessitait une lecture attentive de l'énoncé pour la compréhension de la stratégie employée et de son implantation, et sur le plan du code, la manipulation correcte des types de données liste et booléens, ainsi que le bon usage des fonctions, des 3 structures de contrôles (boucles ***for*** et ***while***, et branchement conditionnel ***if***), et des connecteurs logiques ***and***, ***or***, ***not***.

Les copies les moins réussies ont pointé des lacunes sur les compétences de compréhension de code, ou sur l'écriture de code répondant à un problème algorithmique clairement posé, et dans le code, sur la manipulation de liste, comme l'accès à un élément, suppression, ajout d'un élément, ou parcours de liste, ou encore dans l'usage des connecteurs logiques.

Dans le détail :

La première partie testait la compréhension de l'implantation employée : représentations du labyrinthe à l'aide d'une matrice et d'un chemin à l'aide d'une liste.

Question 1.a. Q.C.M. sur la modélisation du labyrinthe. Question bien réussie ; il s'agissait surtout de vérifier les positions de l'entrée et de la sortie. L'énoncé comportait une coquille (un 0 ayant été changé en 1) ; les candidats ayant relevé l'erreur ou donné une réponse cohérente ont eu tous les points à la question.

Question 1.b. Q.C.M. sur la modélisation d'un chemin. Question bien réussie.

Question 1.c. Ecriture du code d'une fonction retournant les coordonnées de l'entrée et de la sortie du labyrinthe. Question relativement mal réussie. Il s'agissait de parcourir le tableau représentant le labyrinthe à l'aide de deux boucles ***for*** imbriquées, pour mémoriser les positions de l'entrée et de la sortie, et finalement les renvoyer.

La deuxième partie du sujet concevait l'algorithme de recherche d'un chemin de l'entrée à la sortie du labyrinthe.

Question 2.a. Il s'agissait de modifier un code erroné. Question facile, pourtant mal réussie. Il suffisait de modifier la condition du test ; beaucoup de candidats ont répondu de manière semble-t-il un peu aléatoire, démontrant ne pas avoir clairement compris, l'implantation, ou avoir été perturbé par l'usage des connecteurs logiques, la nécessité de tester que les coordonnées se situent bien dans le tableau, ou de l'instruction *lab[x,y]* ***not in*** *[0,2]* ; rappelons que lorsque *L* est une liste et x une donnée, l'expression booléenne *x* ***not in*** *L* vaut ***True*** si *x* n'est pas un élément de *L*, et vaut ***False*** sinon.

Question 2.b. Code à compléter. Question assez facile qui nécessitait l'appel à la fonction *vide()* modifiée dans la question précédente. Elle a été assez moyennement réussie. Assez peu de candidats ont pensé à utiliser la fonction *vide()* ; sans perdre de point, le candidat perd ainsi un temps non négligeable. Faisons remarquer au passage, que chaque fonction écrite dans les questions intermédiaires d'un sujet a pour vocation d'être réemployée dans la suite du sujet.

Question 2.c. Question facile sur la manipulation de liste. Le plus simple était d'utiliser l'instruction *chemin.**pop()***, mais on pouvait aussi utiliser *chemin.**pop(len**(chemin)-1)*** ou ***del*** *chemin[**len**(chemin)-1]*. Etonnement mal réussie ; peu de candidats semblent connaître la méthode ***pop()*** des listes, pourtant bien utile. Notons que la correction ne sanctionne pas de petites erreurs de syntaxe (que ce soit dans cette question ou dans tout le reste du code).

Question 2.d. Code à compléter. C'était la question principale du sujet, celle permettant de déterminer le chemin recherché par une recherche en profondeur, et aussi surement la plus difficile. Sans surprise elle a été très inégalement réussie ; une poignée de candidats a réussi parfaitement la question, ce qui est positivement remarquable. La notation de la question était très progressive et relativement indulgente, valorisant surtout une compréhension au moins partielle de la stratégie employée.

La troisième et dernière partie du sujet proposait une implantation plus efficiente d'un chemin, sous forme « compressée », c'est-à-dire, ici, en n'y conservant que les positions des extrémités et des changements directionnels. Elle permettait notamment au candidat ayant mal compris la stratégie de recherche d'un chemin, d'y montrer ses compétences en compréhension et en complétion de code. La plupart des candidats ont pris le temps d'aborder cette partie

Question 3.a. Question de compréhension sur l'implantation d'un chemin compressé. Question bien réussie.

Question 3.b. Ecriture de fonction. Question bien réussie seulement dans les meilleures copies.

Question 3.c. Complétion de code de la fonction pratiquant la compression du chemin. Question mal réussie, souvent remplie aléatoirement pour glaner des points, à l'exception des quelques toutes meilleures copies.

En conclusion :

- Cette année encore, le sujet a semblé adapté à la durée de l'épreuve ; de nombreux candidats l'ont terminé, dont un candidat exceptionnel qui l'aura parfaitement traité.
- La difficulté résidait dans la compréhension de la stratégie employée et dans sa bonne implantation. Il était nécessaire d'être concentré pour lire attentivement et comprendre l'énoncé.
- Une bonne réussite nécessitait aussi le bon usage des listes de Python. Des lacunes ont été constatées sur ce domaine, notamment dans le parcours de liste ou tableaux bidimensionnels, ou dans l'ajout d'élément en fin de liste (pour laquelle la méthode ***append()*** des listes est la mieux adaptée).
- Des erreurs algorithmiques liées au langage *Python* ont été relevées :
  - Les erreurs de typage, comme la confusion entre le booléen ***True*** et la chaine de caractère *« True »*.
  - Des dépassements d'indice dans l'accès aux éléments d'une liste ; notamment l'instruction *Liste[**len**(Liste)] = x* ne permet pas de rajouter l'élément *x* en fin de la liste *Liste* ; il faut employer l'instruction *Liste.**append**(x)*.
  - Le parcours d'un tableau bidimensionnel à l'aide de deux boucles ***for*** imbriquées n'est pas maîtrisé ; il est pourtant essentiel dans l'implantation de matrices, et devrait être mieux préparé.
  - Rappelons que les listes sont itérables : on peut les parcourir directement dans une boucle ***for*** ; ainsi le code :

    ***for** x **in** Liste :*
    …

    est parfois préférable, car plus simple, au code analogue :

    ***for** k **in range(len**(Liste)) :*
    x = Liste[k]
    …

    on a rencontré en effet trop souvent les erreurs de parcours suivantes :

    ***for** k **in range(len**(Liste)-1) :* ou
    ***for** k **in range(len**(Liste)+1) :*

  - Les erreurs logiques (notamment sur les connecteurs logiques et autres négations) ou sur les structures de contrôle (boucles et tests) sont lourdement sanctionnées.
  - La confusion entre la fonction ***print()*** et ***return*** est fréquente ; il est essentiel d'en distinguer les différences.
  - Dans le cas d'une boucle dont le nombre d'itération est connu à l'avance, il est approprié, bien que non nécessaire, de privilégier une boucle *for* à une boucle *while*, évitant le plus souvent les manquements aboutissant à la non-terminaison de la boucle.

Conseils aux candidats :

- Pour une bonne réussite à l'épreuve, le candidat doit bien maîtriser les notions exigibles d'algorithmique en python : types de données, structures de contrôle, fonctions, etc. L'usage des listes en Python est incontournable. Cela ne peut s'acquérir que par une pratique investie et régulière de la programmation durant les 2 ou 3 années de préparation.

- La correction est indulgente face aux petites erreurs de syntaxe (oubli des « : », orthographe des fonctions, etc.) mais pas face aux erreurs algorithmiques (typage, expressions booléennes, structure de contrôle, fonctions, indentation, indexation, etc.)

- Les fonctions écrites dans les questions intermédiaires ont pour vocation d'être appelées dans les questions suivantes. Le candidat devrait naturellement se questionner lorsqu'il n'a pas utilisé dans la suite du sujet ces fonctions intermédiaires.

- L'énoncé doit être lu très attentivement ; il contient la description de l'algorithme à réaliser, et des exemples de code non exigible ou rappels de code exigible à employer. Ne surtout pas négliger sa lecture approfondie.

- Il est improductif de remplir de manière aléatoire du code à compléter ; tenter d'y glaner quelques points n'en rapporte en général aucun. Mieux vaut se consacrer de manière plus investie à un nombre plus restreint de questions.

- Le sujet comporte en général plusieurs parties souvent indépendantes ; une première lecture rapide préliminaire de la totalité de l'énoncé permettra de repérer les parties plus favorables au niveau du candidat. Notons aussi que la première partie est toujours la plus largement accessible, et qu'on ne peut se passer de la lecture de la partie préliminaire. Dans une même partie, la difficulté des questions est généralement progressive.