Ecole Polytechnique
ENS Lyon, Paris-Saclay, Rennes

Concours d'admission 2021

Composition d'Informatique B (XLCR), Filière MP

Bilan général

Pour cette épreuve, 176 copies des candidats français de l'Ecole polytechnique ont été rendues. La note moyenne est 7,43/20, l'écart-type 290.

Commentaires

Le sujet portait cette année sur un système de gestion de la mémoire, permettant à un utilisateur de réserver des blocs de mémoire et de les libérer, avec l'objectif de réutiliser au maximum les espaces libérés tout en gardant une complexité efficace pour la recherche de zones disponibles. La première partie se concentrait sur une version naïve sans libération possible. La seconde partie ajoutait la gestion de la libération, mais uniquement avec des blocs de taille fixée. Dans la troisième partie on gérait la libération de zones de taille quelconque, et leur fusion avec les zones libres adjacentes. Enfin dans la quatrième partie un système de liste chaînée des zones libres était introduit de façon à accélérer la recherche de zones disponibles.

Le sujet n'était pas particulièrement long (une large majorité des candidats a répondu à plusieurs questions de la partie IV), mais demandait une lecture très attentive des contraintes et des outils mis à disposition dans chaque partie.

Le langage Python était cette année relativement bien maîtrisé. On rappelle les points d'attention généraux lorsque l'on écrit du code "sur papier": faire bien attention aux indentations, en particulier au moment des sauts de pages. L'idéal est de parfaitement suivre le quadrillage des copies, mais des lignes verticales indiquant la portée de chaque bloc peuvent aider si nécessaire (attention dans ce cas à ne pas introduire d'ambiguïté supplémentaire…). La syntaxe du Python doit être parfaitement respectée: parenthèses aux appels de fonction, deux-points pour les if, for et def, respect des majuscules, etc.

La principale pierre d'achoppement pour de nombreux candidats a été le suivi des contraintes dictées par le sujet pour accéder à la mémoire. En fonction des parties, des

fonctions telles que *est_libre(p), lire_taille(p),* etc. étaient données pour accéder à la mémoire. Ces fonctions devaient être utilisées à chaque fois qu'elles étaient utiles, et *uniquement* pour effectuer l'action "logique" correspondant à leur description. Par exemple, dans la partie II, même si d'après leur implémentation lire(0,0) renvoie exactement la même valeur que lire_prochain(), ces fonctions n'étaient pas interchangeables: lire ne pouvait être utilisée que pour récupérer une valeur à l'intérieur d'un bloc, pas pour accéder aux en-têtes des blocs ou de la mémoire.

Commentaires détaillés

Pour chacune des questions, sont indiquées les statistiques suivantes:

- Moyenne: note moyenne pour cette question, ramenée sur 1. Cette moyenne tient compte uniquement des notes des candidats qui ont traité la question.
- 0,]0 ; 0,5[, [0,5; 1[, 1 : pourcentage de copies dont la note est dans l'intervalle correspondant
- Question non traitée: pourcentage de candidats qui n'ont pas traité la question.

Partie I : Implémentation naïve

Question 1

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,69	4%	9%	42%	45%	0%

Ici une simple boucle suffisait, idéalement avec la fonction ecrire() pour remplir la mémoire (en utilisant les bons indices pour le numéro de bloc et la position dans le bloc).

Question 2

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,62	20%	5%	33%	41%	0%

Dans cette fonction il n'y avait besoin *que* d'initialiser mem[0], puisqu'il était explicitement dit que mem était initialisée avant l'appel à demarrage().

Question 3

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,61	2%	22%	65%	10%	0%

Pour cette question il fallait faire attention à bien vérifier la disponibilité de la mémoire, mettre à jour le pointeur vers la prochaine case, et initialiser la zone grâce à la fonction dédiée à cet effet.

Partie II : Réservations de blocs de tailles fixes

Question 4

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,52	25%	7%	36%	30%	2%

A nouveau il ne fallait pas réinitialiser entièrement la mémoire pour cette question, seule la fonction ecrire_prochain était utile pour bien initialiser mem[0].

Question 5

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,53	0%	39%	61%	0%	0%

Deux cas pouvaient se présenter pour cette fonction: soit un bloc précédemment occupé a été libéré et peut être réutilisé (on utilise une boucle pour détecter un tel bloc), soit on utilise un nouveau bloc après le dernier bloc utilisé (dans ce cas, on doit bien vérifier la disponibilité de la mémoire et mettre à jour le pointeur correspondant).

Le calcul de la complexité était très souvent erroné: TAILLE_BLOC étant considéré comme une constante, l'initialisation d'un bloc (ou d'une partie d'un bloc) avait un coût constant (et non O(n): si on demande un bloc de taille n=1000 dans une mémoire de taille 10, il n'y a pas besoin de 1000 opérations pour renvoyer None).

Question 6

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,85	4%	3%	18%	74%	1%

La solution la plus parcimonieuse pour cette question consistait simplement à appeler marque_libre: il n'était pas utile de réinitialiser le bloc libéré.

Partie III : Portions avec en-tête et pied de page

Question 7

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,92	1%	1%	16%	81%	1%

Il s'agissait pour cette question de bien comprendre et expliquer comment étaient écrits et lus un entier pair (la taille) et un booléen (est_libre) sur un seul entier: le booléen est stocké sur le bit de poids faible, donc l'écriture se fait directement avec une somme. Pour la lecture, on utilise le reste de la division entière pour le booléen, et l'arrondi à l'entier pair inférieur pour retrouver la taille.

Question 8

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,59	11%	30%	21%	34%	3%

Pour cette question il est nécessaire d'initialiser les 5 premières cases mémoire, servant d'en-tête, de prologue et d'épilogue. L'en-tête est initialisé avec lire_epilogue(), les cases suivantes avec deux appels à marque_libre() (qui correspondent bien aux deux portions décrites dans les spécifications).

Question 9

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,47	3%	48%	45%	0%	4%

Cette question était la plus complexe du sujet, avec de nombreuses difficultés à prendre en compte. Tout d'abord, la valeur de n en entrée n'étant pas nécessairement paire (cette condition n'entrant pas dans les hypothèses de bon usage), il fallait réaliser un arrondi à l'entier pair supérieur. Ensuite, pour la recherche d'une zone libre, il s'agit de bien prendre en compte toutes les zones qui sont au moins aussi grandes que n (et pas forcément exactement n). Cette recherche est beaucoup plus naturelle si on utilise l'adresse de la portion, plutôt que celle de son en-tête. Si cette recherche échoue, il fallait également s'assurer que la place restante était suffisante pour écrire le nouveau bloc. Enfin, dans le cas où on utilisait une portion libre plus grande que n, il fallait bien traiter le recyclage du reste d'un bloc: ce recyclage est inutile s'il reste au plus 4 cases mémoire disponibles (sinon on ne pourrait pas y insérer de prologue, d'épilogue et de données supplémentaires), et dans ce cas il faut augmenter la taille de la portion créée pour bien correspondre aux limites existantes et ne pas créer de décalage. Finalement, une fois que la position et la taille du

bloc créé ont été correctement calculées, on termine avec marque_reservee() et initialiser(). Cette fonction nécessite au pire O(TAILLE_MEMOIRE), la valeur de n n'ayant pas à apparaître dans cette complexité.

Question 10

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,53	4%	30%	39%	8%	18%

Pour cette question il s'agit de fusionner le bloc libéré avec son prédécesseur et son successeur si ceux-ci sont libres. Il est inutile d'utiliser une boucle, puisque la mémoire n'a jamais deux portions libres consécutives. En fonction de l'approche utilisée, cette question en théorie relativement simple pouvait rapidement devenir très compliquée voire illisible. Il était important d'utiliser des variables soit pour les tailles de bloc soit pour la position des blocs voisins pour éviter les lignes avec plusieurs appels à lire_taille imbriqués. Enfin, il était beaucoup plus simple de réaliser indépendamment les fusions à gauche et les fusions à droite, plutôt que de tester toutes les combinaisons de cas possibles**.**

Partie IV : Chaînage explicite des portions libres

Question 11

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,47	5%	33%	20%	5%	37%

Pour une insertion dans une liste doublement chaînée, il est important d'une part d'initialiser les pointeurs vers les prédécesseurs et successeurs de l'élément, mais aussi de mettre à jour le pointeur de l'ancien premier élément (s'il existe) et celui de tête de liste.

Question 12

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,50	3%	18%	26%	8%	45%

Pour la suppression, il faut d'abord récupérer les numéros du prédécesseur et du successeur, et les faire pointer l'un vers l'autre. Il s'agit de bien traiter les cas au bord, où l'élément supprimé n'a pas de prédécesseur et/ou de successeur.

Question 13

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,62	5%	12%	14%	18%	51%

Cette question est très similaire à la question 8, la différence consistant à initialiser la liste (avec ecrire_entree()).

Question 14

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,44	4%	16%	17%	2%	61%

Cette question posait les mêmes difficultés que la question 9. Il était d'autant plus important de bien utiliser l'adresse du bloc plutôt que son en-tête pour manipuler les pointeurs vers les blocs précédent et suivants. La recherche d'un bloc libre ne posait pas de difficulté particulière. En cas de recyclage d'une fraction de bloc libre, la solution la plus directe consistait à retirer l'ancien bloc de la chaîne et de ré-insérer le nouveau bloc (le cas échéant): tenter de mettre à jour les pointeurs des voisins manuellement posait inévitablement des difficultés importantes pour bien gérer les cas particuliers.

Question 15

Moyenne	0	]0;0.5[	[0.5;1[	1	Question non traitée
0,55	2%	8%	7%	4%	79%

À nouveau cette question était très similaire à la question 10. En plus d'insérer le nouveau bloc libre dans la chaîne, il fallait bien penser au préalable à retirer les éventuels blocs adjacents avec lequel il fusionne.