

8 Épreuve d'Informatique – Filière MPI

Ce chapitre présente le compte rendu de la session 2024 de l'épreuve de TP d'informatique.

Son objectif est de présenter les modalités du TP et de donner des conseils pour que les candidats puissent au mieux préparer les prochaines années.

Beaucoup de « défauts » sont listés, mais le jury précise qu'il a été satisfait de la prestation des candidats. En 2024, le jury a trouvé que les candidats étaient mieux préparé qu'en 2023, année où le jury avait déjà trouvé que le niveau était bon, à la fois sur le programme et les technique que sur la manipulation des outils.

8.1 Déroulement de l'épreuve.

Accueil des candidats.

Chaque session de l'épreuve du TP d'informatique commençait par un accueil des candidats où les examinateurs rappelaient les consignes générales :

- les candidats peuvent manger et boire dans les salles (mais en faisant très attention aux ordinateurs !) ; ils peuvent aller aux toilettes (en demandant avant !) ;
- pendant l'oral, les candidats peuvent n'avoir avec eux que pièce d'identité, convocation, stylos et éventuellement nourriture et boisson. S'ils ont d'autres affaires (comme des sacs, des téléphones ou des montres), ils peuvent les poser dans un coin de la salle. Du brouillon et un compte rendu vierge leur sont fournis ;
- les candidats peuvent poser toutes leurs questions pendant l'oral et cela n'affecte pas la notation (mais le jury se réserve le droit de répondre ou pas) ;
- il est demandé aux candidats de tester leurs codes et la notation prend en compte ces tests ;
- les candidats doivent rédiger un compte rendu ;
- les épreuves ayant lieu sur machine, il est possible d'avoir des pannes. Dans ce cas, les candidats doivent prévenir au plus vite les examinateurs qui résoudront le problème en compensant la perte de temps par du temps supplémentaire à la fin ou dans la notation. Il est attendu des candidats qu'ils enregistrent régulièrement.

Installation dans les salles.

Après l'introduction générale, les candidats sont ensuite répartis dans les diverses salles. Cette année, il y avait maximum 4 candidats par salle. Les candidats posent leurs sacs dans un coin de la salle puis s'installent à un des postes disponibles. Les candidats ont eu le droit à quelques minutes de familiarisation avec la machine.

Épreuve principale.

Vient ensuite l'épreuve. Les examinateurs donnent le top départ à une heure précise et à ce moment les candidats doivent charger une page web où le sujet est présenté.

Même si le sujet est souvent fourni en intégralité dès le début de l'épreuve, le jury déconseille de le lire en intégralité car ce serait une perte de temps, en revanche ce n'est pas une mauvaise idée de

lire quelques questions à l'avance (surtout au début de l'épreuve) pour mieux comprendre ce qui est attendu.

Au cours des 3h30, les examinateurs passent régulièrement voir les candidats. Ceux qui ont une question peuvent appeler les examinateurs sans attendre que ceux-ci ne passent.

Attention, certains candidats tentent de sauter certaines parties sans demander à l'examineur. Cette technique est à proscrire lors d'un oral, elle exhibe de toute façon les lacunes du candidat (peur de la technicité, impasse, ou autre).

Fin de l'épreuve.

L'épreuve est prévue pour durer exactement 3h30. Quelques minutes avant la fin les examinateurs rappellent l'imminence de la fin de l'épreuve pour que les candidats s'assurent que le compte rendu est bien à jour et les examinateurs passent une dernière fois pour noter où en sont les candidats. Il n'est pas demandé de mettre leurs fichiers à un endroit spécifique à la fin de l'épreuve.

Précisions quant aux tenues vestimentaires.

De nombreuses matières du Concours Commun Mines Ponts insistent dans les rapports sur la tenue correcte exigée. Entre le stress sur 3h30 d'épreuve et les salles peu ou pas climatisées les candidats peuvent avoir chaud... Les candidats peuvent venir en tenue légère si des températures chaudes sont prévues et ceux qui viennent en costume-cravate peuvent retirer la veste ou la cravate pendant l'épreuve.

Précisions quant à l'écriture du compte rendu.

À l'inverse des autres TPs où le compte rendu a un rôle central dans le suivi des expériences et la compétence de remplissage d'un compte rendu est attendue, dans l'épreuve de TP informatique le compte rendu sert principalement à suivre la progression des candidats durant l'épreuve ainsi qu'à résumer ce qui a été fait.

Quand les examinateurs passent voir les candidats, ils commencent généralement par lire le compte rendu, voient ce qui a changé depuis leur dernier passage puis, en fonction de cela, posent éventuellement des questions ou regardent ce que le candidat est en train de faire. Ceci permet d'économiser les interactions orales qui feraient perdre du temps au candidat mais aussi pourraient gêner les autres candidats dans la salle. Le compte rendu doit donc être plus détaillé qu'une succession de "Q 8 : faite" mais ce n'est pas non plus la peine de détailler les réponses autant que dans une copie écrite. En général, même les questions les plus compliquées ne requièrent pas d'écrire plus de 5 lignes.

Savoir remplir le compte rendu n'est pas une compétence attendue, le jury ne retire pas de points sur la façon dont les candidats remplissent les compte rendus, en revanche il peut signaler à des candidats qu'ils peuvent être plus concis dans leurs réponses (pour ne pas perdre de temps) ou, au contraire, qu'il faut détailler plus leurs réponses quand ils trouve celle-ci insuffisante. Les réponses fausses (mauvais calcul de complexité ou algorithme faux par exemple) peuvent être prises en compte dans la notation mais si le jury demande de compléter une réponse trop courte ou lorsqu'il demande à être plus concis, cela n'est pas pris en compte dans la notation.

Voici quelques précisions sur la façon de remplir le compte rendu :

- pour toute question posée dans le sujet ou à l'oral, le jury attend une réponse écrite sur le compte rendu ;
- les sujets sont majoritairement donnés en format PDF avec des questions numérotés. Ce n'est donc pas la peine recopier les questions, il suffit de donner le numéro de la question répondue ;

- certaines questions sont vraiment très simples (comme écrire une fonction qui somme deux vecteurs 2D) dans ce cas le candidat peut se contenter d’un “Q1 : faite” écrit sur le compte rendu ;
- pour les questions qui n’attendent pas du code, on demande une réponse brève sur le compte rendu et de cette façon l’examineur peut lire la réponse sans déranger le candidat (ou les autres candidats). Le jury insiste sur le fait que les candidats n’ont pas à écrire des réponses longues, si l’examineur trouve la réponse trop laconique, il peut toujours demander au candidat de préciser ;
- enfin, pour les questions d’algorithmique ou de code non triviales, le jury attend une description générale de l’algorithme, une complexité (sans justifier) et les tests effectués. Par exemple, dans le cas où l’algorithme est un parcours de graphe, il convient d’expliquer si le parcours est en largeur ou en profondeur, dans quel graphe quand le graphe est implicite, etc. puis d’indiquer brièvement quels graphes ont été testés et que la complexité est $O(n + m)$ avec n le nombre de noeud et m le nombre d’arêtes. Le compte-rendu peut servir de “brouillon” pour préparer les tests qui sont parfois plus lisibles sous forme dessin (par exemple pour un graphe) que sous forme de code.

8.2 Commentaires généraux sur la méthode de programmation.

Le jury a constaté que les défauts relevés dans le rapport du concours 2023 étaient moins présents en 2024. Cette partie va néanmoins va beaucoup répéter le contenu du rapport 2023 car ces défauts persistent chez certains candidats.

Architecture du code.

Beaucoup de candidats voient chaque question comme un bloc unitaire et semblent avoir du mal à voir une question comme la combinaison de plusieurs sous-problèmes qui peuvent être implémentés dans plusieurs fonctions avec des tests pour chaque. De manière plus générale, le jury trouve que, même quand les candidats ont la bonne idée, ils ont du mal à décrire et donc à réfléchir sur leur solution avant de se lancer dans le code.

Cela pénalise souvent les candidats qui perdent du temps à ne pas tester chaque partie indépendamment quand il y a des bugs, qui écrivent souvent des bouts de code inutiles qu’ils doivent ensuite retravailler et parfois qui se rendent compte qu’au bout de 15 minutes que leur solution ne marche pas. Pour toutes les questions non triviales, le jury conseille de passer quelques minutes à élaborer rapidement la solution (par exemple sous forme de pseudo code ou juste en donnant les grandes étapes) puis à réfléchir à comment simplifier la solution ou la décomposer en plusieurs sous-problèmes. Ce temps perdu sera souvent regagné ensuite. . .

Tout ceci fait que les candidats sont en majorité bien plus à l’aide sur les sujets très guidés avec beaucoup de questions intermédiaires, et sont parfois déroutés sur les sujets plus ouverts. Il serait intéressant d’être en mesure, à la fin des années de CPGE, de prendre l’initiative sur un problème complexe. Plusieurs sujets étaient assez ouverts et les jurys continueront de poser des sujets ouverts les années prochaines, ces sujets posent particulièrement problèmes aux élèves qui ont du mal à prendre des initiatives ou à avoir une posture réflexive par rapport à ce qu’il faudrait coder.

Tests.

Pour les questions d'algorithmique non triviales, il était attendu des candidats qu'ils testent leurs programmes. Cette consigne était donnée avant l'oral et les examinateurs le rappelaient régulièrement aux candidats. Il n'est pas attendu de faire des tests complets qui pourraient presque garantir une absence de bugs mais simplement vérifier qu'il n'y a pas d'erreur manifeste. Cette demande de tester les programmes remplit plusieurs objectifs :

- cela permet aux examinateurs de valider la compétence test qui figure au programme de la MPI, notamment dans le fait d'écrire des tests pertinents qui essaient de couvrir les différents cas possibles ;
- cela permet ensuite aux examinateurs de vérifier plus facilement la correction des programmes écrits par les candidats. En effet, les examinateurs lisent souvent rapidement les programmes écrits par les candidats et les tests (s'ils sont bien écrits) renforcent la confiance dans la correction ou non du code ;
- enfin les tests permettent aux candidats d'avoir confiance (de façon justifiée !) dans ce qu'ils écrivent. Certains candidats semblent penser que tester un programme est une perte de temps voire un aveu de faiblesse. Le jury préfère toujours un candidat prudent qui teste son code à un candidat trop sûr de lui qui ne le teste pas, même quand le code est correct. Un candidat qui refuse de tester un code que le jury sait faux fait une toute particulièrement mauvaise impression. Les candidats devraient savoir que tout programmeur peut faire des bugs et c'est encore plus vrai pour des candidats sortants de MPI qui n'ont que peu d'expérience.

Il est évident que la quantité de tests à effectuer dépend de la complexité du code de la difficulté d'écrire des tests et de la précision des tests. Une fonction qui renvoie une information booléenne (si un graphe est connexe ou non par exemple) a plus de chances de renvoyer la mauvaise réponse au hasard qu'une fonction qui renvoie quelque chose de précis (par exemple si cela affiche pour chaque composante la liste des nœuds). De la même manière une fonction très simple (par exemple la somme de deux vecteurs 2D) a moins de chances d'être fausse que l'implémentation d'un algorithme compliqué (p.ex. une file à priorité).

Il est parfois utile de réfléchir à comment écrire des tests rapidement ou intelligemment. Si on ne peut pas se contenter de tester un algorithme que sur des exemples triviaux, ces exemples peuvent tout de même servir de test car ils testent bien les cas de base. S'il est demandé, par exemple, d'écrire une fonction qui décompose un graphe en ses composantes connexes, on peut facilement tester sur 5 graphes de la façon suivante : on a 2 graphes triviaux (un graphe avec 2 nœuds, connectés ou non) puis 3 graphes plus compliqués (par exemple un graphe à 8 nœuds et sa copie avec une puis deux arrêtes en plus dans la copie ce qui fait passer le graphe de 3 à 1 composante connexe).

Voici plusieurs conseils pour les candidats :

- il est important de calculer à l'avance le résultat des tests. À plusieurs reprises, le jury a vu des candidats écrire un unique test compliqué et prendre le résultat de leur fonction comme étant le standard pour leur test alors que la fonction renvoyait un résultat faux ;
- des candidats utilisent un éditeur pour leur code et testent avec un shell (comme `utop`) et donc leurs tests "disparaissent" et ils perdent beaucoup de temps à les réécrire pour chaque petit

changement de leur fonction, ce qui pousse les candidats à n'écrire que peu de tests. Le jury ne déconseille pas d'utiliser `utop` mais recommande d'écrire les tests dans le code principal. Si, par exemple, il faut tester des algorithmes de graphe, on peut définir des graphes dans le code principal et les utiliser à chaque question pour tester. Il est aussi conseillé d'utiliser des `assert`, de cette façon on garde une trace des tests utilisés (pour les montrer au jury), on peut plus facilement faire beaucoup de tests d'un coup et ces tests sont facilement copiables d'une question à la suivante ;

- enfin, il est parfois possible de tester plusieurs fonctions en même temps quand les questions sont données à l'avance. S'il est demandé, par exemple, de faire des fonctions somme, produit puis évaluation pour des polynômes, on peut tester très rapidement chaque fonction puis faire des tests plus compliqués qui combinent ces trois fonctions en testant que $((P + Q) * R)(42) = (P(42) + Q(42)) * R(42)$ pour plusieurs polynômes P , Q et R .

Gestion des bugs.

Les outils adaptés au débogage (comme `ocamldebug` ou `trace` en OCaml et `gdb` en C) sont peut-être un peu compliqués pour les candidats mais voici quelques conseils faciles à mettre en œuvre pour s'attaquer méthodiquement au débogage :

- parfois un premier problème engendre un second plus visible (par exemple un calcul renvoie le mauvais résultat et à cause de cela on fait un accès hors des cases du tableau). Il faut bien penser à chercher détecter le moment où le premier problème apparaît. Pour cela la programmation défensive et l'utilisation d'`assert` permet de gagner du temps. De la même manière, il est souvent intéressant de chercher à simplifier l'exemple où l'algorithme bugue car c'est plus simple de suivre le déroulé de l'algorithme ;
- les candidats pensent souvent à faire des `printf` mais pas toujours à l'utilisation des `assert`. Même si le `assert` est moins informatif et donc qu'il est parfois plus utile de déboguer en utilisant aussi `printf`, lire des dizaines de lignes prend plus de temps qu'un simple `assert`... il faut savoir combiner les deux. Il ne faut pas hésiter à mettre des dizaines d'`assert` dans un code ;
- quand on manipule une structure de données un peu compliquée (comme un tas ou un arbre rouge-noir) il peut être intéressant de faire une fonction qui vérifie que cette structure a le format attendu (pour un tas ou un arbre rouge-noir, par exemple, la fonction de vérification est assez simple comparé au reste de l'algorithme). Lors de la phase de test on peut par exemple vérifier la structure après chaque modification (ce qui renforce la confiance en la correction de l'algorithme car les tests testent bien mieux). Il peut être utile d'utiliser une variable `debug` pour déclencher ou non ces tests ;
- quand le candidat écrit une fonction qui attend un argument qui a une forme précise (par exemple un entier positif ou un tableau trié), il ne faut pas hésiter (quand c'est pertinent) à mettre un `assert` pour le vérifier et ainsi détecter plus rapidement les bugs plus loin dans le programme (si cette fonction est réutilisée) ;
- quand un programme a plusieurs fonctions et sous-fonctions on peut tester chaque fonction indépendamment ;

- enfin, il est presque toujours recommandé d'utiliser les warnings du compilateur. Les options de compilation recommandées en C sont `-Wall` `-Wextra` et `-fsanitize=address` (mais un candidat qui n'a jamais testé ces options risque d'être décontenancé par les warnings).

Environnement de développement.

Cette année le concours n'a pas utilisé de machine virtuelle mais une image était fournie sur le site du concours et l'environnement du concours était assez proche en terme de configuration. Nous encourageons vivement les candidats à tester la machine virtuelle proposé sur le site du concours, ne serait-ce que 10 min, pour ne pas être déstabilisé face à l'environnement qu'ils rencontreront le jour du concours.

Cette année, les candidats avaient quelques minutes avant le début de l'épreuve où ils pouvaient utiliser la machine. Certains candidats, pris par le stress commencent à taper des bouts de code en C ou en OCaml, le jury n'est pas persuadé que cela les aide beaucoup d'autant que les candidats ne savent à ce moment pas quel sera le langage ni si un patron de code est fourni.

Le jury essaie de fournir, dans la mesure du possible, les logiciels libres et populaires, et on a vu de nombreux logiciels utilisés comme VScodium, emacs, gedit, nano, vim. Le jury continuera à proposer un large choix d'éditeurs de texte ainsi que plusieurs outils adaptés (des "modes" ocaml dans ces éditeurs et des shells adaptés comme `utop`). Cela étant, il est dommageable que beaucoup de candidats ne savent programmer qu'avec un outil très spécifique et sont déroutés à l'oral lorsque l'environnement ne leur propose pas ou que ce n'est pas compatible avec un sujet.

Par exemple, lorsque rappelée, l'utilisation de `ocamlopt` ou de `ocamlc` peut se révéler laborieuse et le manque d'aisance fait perdre du temps aux candidats. De la même manière, il était recommandé dans un sujet de ne pas taper manuellement l'entrée à chaque test mais de rediriger l'entrée standard (la syntaxe `./monProgramme < monFichier` était donnée dans le sujet) et très peu d'élèves ont utilisé cette commande... qui était pourtant là pour aider les candidats !

Pour finir rappelons que les candidats vont passer un certain temps dans le terminal. Sans que le jury n'attende une grande familiarité avec l'outil, certains candidats gagneraient beaucoup de temps en connaissant les raccourcis de base comme flèche du haut pour remonter dans l'historique, `ctrl+R` pour rechercher, `ctrl+C` pour arrêter un processus, etc.

Nom des variables, fonctions et commentaires.

Les examinateurs examinant le code des candidats, il est attendu des candidats qu'ils produisent des codes lisibles. Le jury n'attend pas des candidats des choses élaborées ou le suivi de conventions particulières mais simplement que les candidats utilisent des noms de fonctions ou de variables pertinents et des commentaires aux endroits qui le nécessitent.

Quand le nom de la fonction et de ses arguments ne suffisent pas à comprendre son objectif, ou si la fonction est longue avec plusieurs blocs qui accomplissent plusieurs choses différentes, le jury attend aussi un commentaire rapide qui décrit ce que fait le bout de code considéré.

Pour des variables locales à une fonction, les candidats peuvent se contenter de noms simples de variables mais pour les variables globales, pour les noms de fonction et éventuellement pour les variables nommant les arguments de ces fonctions, il est demandé de donner des noms qui précisent ce qu'elles représentent. Le jury n'a pas envie de chercher quels sont les rôles respectifs de `aux`, `aux1`, `aux2` et `aux1bis` ni ce que signifient les arguments `a`, `b` et `c` tous de type `int`.

Notons au passage que les candidats passent souvent autant de temps à relire ou déboguer leur code qu'à l'écrire, il est donc probable que le temps économisé à mettre des variables à une seule lettre soit

souvent perdu dans la relecture qui est plus difficile.

8.3 Commentaires liés au programme.

Il est difficile de se prononcer sur la compréhension du programme par les candidats car, bien que l'intégralité du programme a été traité dans les sujets posés, chaque point du programme n'était généralement traité que dans un petit nombre de sujets et chaque sujet n'était posé qu'à un petit nombre de candidats.

Par rapport à 2023, le jury a rencontré moins de candidats qui avaient fait des impasses complètes sur un pan du programme. Le conseil du jury est qu'il est important de bien maîtriser et de savoir implémenter rapidement tous les algorithmes de base (parcours en largeur et en profondeur, tas, etc.) mais qu'il faut aussi connaître toutes les définitions et tous les algorithmes du programme (sans forcément les avoir implémentés ou en maîtriser toutes les subtilités). Il est très dangereux de faire une impasse complète sur un point du programme car un sujet entier peut porter sur un point du programme.

Documentation.

Le jury fournissait une documentation accessible via le navigateur. La documentation de référence en C et OCaml étaient disponibles ainsi qu'une "cheatsheet" de sqlite3 et un polycopié de C.

Les noms et prototypes des fonctions de la bibliothèque standard qui étaient nécessaires dans certains sujets (pour ouvrir des fichiers ou prendre des mutex) étaient généralement directement rappelés dans les sujets soit par de la documentation soit par des exemples.

8.4 Commentaires liés au langage SQL.

Cette année, la plupart des candidats arrivaient à résoudre les questions de SQL mais certains le font beaucoup plus rapidement que d'autres et donc la distinction entre candidats se fait plus sur la vitesse à résoudre les questions. Nous conseillons donc aux candidats de s'entraîner à résoudre rapidement des problèmes de SQL.

Tous les sujets qui manipulaient du SQL utilisaient le moteur de requête SQLite3 (dont une brève documentation était fournie) mais le jury se réserve le droit d'utiliser d'autres moteurs de requêtes dans le futur. Il n'est pas attendu de connaissance spécifique aux légères variations qui peuvent exister entre les divers moteurs de requêtes et en particulier il n'est pas attendu des candidats qu'ils connaissent les options de chaque client SQL. Le jury fournissait la commande à lancer et indiquait qu'il était recommandé de taper les commandes `.header on` et `.mode column` pour avoir des résultats de commandes plus lisibles.

Enfin le jury conseille de connaître la construction `COUNT(DISTINCT v)` car elle peut souvent simplifier l'écriture des requêtes (qui pouvaient s'écrire autrement dans tous les sujets posés).

8.5 Commentaires liés au langage C.

Le jury a, de nouveau, été favorablement surpris par la maîtrise du langage C qu'ont en moyenne les candidats. Voici quelques précisions pour une meilleure préparation.

Gestion de la mémoire. Certains candidats ont du mal avec `malloc`. Dans les erreurs récurrentes que les candidats ont commises : oubli de `sizeof` (et donc mémoire allouée trop petite), tentatives d'appels à `malloc` en dehors de toute fonction (pour des variables globales), quelques candidats qui ne savent pas allouer un tableau 1D, d'autres, plus nombreux, qui ont des problèmes avec les tableaux 2D (que ce soit des tableaux de tableaux ou des tableaux linéarisés).

Compilation. Comme décrit plus haut dans la section gestion des bugs, le jury recommande fortement aux candidats de compiler avec les options `-Wall -Wextra` et `-fsanitize=address` car cela permet d'attraper de diverses erreurs et facilite le débogage. Quand il n'y a qu'un seul fichier à compiler, le jury déconseille de compiler en deux étapes (fichier objet puis exécutable) car cela fait perdre du temps aux candidats, et ce d'autant plus que certains candidats retapent entièrement chaque commande dans le shell (plutôt que de rechercher avec `ctrl+r` ou de relancer une commande précédente avec flèche du haut).

Libération de la mémoire et valeurs de retour des fonctions de la bibliothèque.

Comme annoncé dans le rapport 2023, le jury ne peut pas facilement vérifier que la mémoire allouée est bien libérée et donc, même si ce n'est pas une bonne pratique de programmation, le jury ne demande pas de libérer la mémoire et il ne faut pas que les candidats perdent du temps à le faire (sauf demande explicite dans le sujet ou par l'examineur). De la même manière la vérification des valeurs de retour des fonctions (comme `fopen`) n'est pas attendue.

À l'inverse, le jury a pris en compte, pour une petite partie de la note, la qualité du code (pertinence des noms de variables et fonctions, commentaires, tests et `assert`, lisibilité) et continuera à la prendre en compte pour les prochaines années.

8.6 Commentaires liés au langage OCaml.

La maîtrise du langage OCaml par les candidats est assez bonne mais le jury a tout de même quelques conseils et remarques pour les candidats.

Utilisation simple de la bibliothèque standard. Peu de candidats utilisent les fonctions "de base" sur la manipulation de listes (comme `filter`, `map`, `iter`, `exists`) qui sont au programme. Le jury ne pénalise pas leur non-usage mais bien les connaître permet souvent d'écrire du code plus court donc plus rapide à écrire et plus simple à relire. Le jury a aussi peu vu d'utilisation des tables de hachage (par exemple pour détecter des doublons).

Utilisation avancée de la bibliothèque. Seule une petite partie de la bibliothèque standard OCaml est au programme et seule cette partie est exigible mais cela ne veut pas dire que les candidats doivent s'interdire toute autre fonction. Les candidats ayant une installation standard d'OCaml avec la documentation complète, le jury souhaite leur laisser accès à ces bibliothèques pour ne pas pénaliser ceux qui en ont l'habitude mais il ne souhaite pas non plus que la connaissance des fonctionnalités avancées de la bibliothèque donne un véritable avantage.

Il y a, par exemple, dans la bibliothèque standard les `Set`. Ces `Set` peuvent servir d'arbres binaires équilibrés ou de files de priorité. Si une question demande d'écrire un arbre binaire de recherche équilibré et qu'un candidat propose de répondre avec `Set`, il serait probablement demandé au candidat de recommencer sans (ou alors il n'obtiendrait qu'une partie des points sur cette question). Ce cas ne s'est pas présenté, mais, si un candidat veut utiliser de telles fonctions avancées, il convient de d'abord demander à l'examineur.

Le jury a constaté l'utilisation de nombreuses fonctions de la bibliothèque hors programme comme `List.mapi`, `List.assoc`, `Array.of_list` ou même `String.split_on_char` et le jury a considéré toutes ces utilisations acceptables du moment que les candidats pourraient rapidement les ré-implementer (et que ce n'était pas le cœur du problème posé). Le jury ne conseille pas spécialement l'apprentissage de ces fonctions car il ne pense pas que l'utilisation de ces fonctions apportent un véritable avantage aux candidats. Cela étant, connaître, par exemple, la structure de liste associative ou l'idée de décomposer un problème de parsing en un découpage de la chaîne en token pour ensuite traiter chaque token peuvent aider les candidats et les candidats vraiment très à l'aise avec la bibliothèque standard gagnent du temps.

Attention tout de même à l'utilisation de la bibliothèque. De multiples candidats qui ont un style de programmation impératif et utilisent beaucoup, par exemple, `List.nth` sans faire attention (ou même parfois connaître) la complexité de cette opération ! Même quand les candidats utilisent une fonction de la bibliothèque dans un algorithme, ils doivent être capables de donner la complexité de l'algorithme.

Certains candidats ont utilisé la documentation pour voir ce qui existait. Par exemple, pour un exercice de parsing certains ont regardé la documentation du module `String`. Tant que les candidats n'utilisent pas des fonctions trop avancées, le jury n'a aucun de problème avec cette pratique. Au contraire, c'est plutôt un bon réflexe et comme lire la documentation n'est pas forcément évident et il est donc conseillé d'avoir déjà de l'expérience avec la documentation OCaml. En effet, des sujets peuvent nécessiter l'utilisation de bibliothèques, auxquels cas les candidats auront à lire la documentation.

Utilisation du tri de la bibliothèque.

Le cas du tri est un peu particulier. Les candidats pensent souvent à des algorithmes qui utilisent des tris même quand ceux-ci ne sont pas strictement nécessaires. C'est un réflexe algorithmique louable que de se ramener à un problème plus simple en utilisant un algorithme connu mais cela pose un problème : `List.sort` n'est pas au programme (et donc pas toujours connu), le tri fusion (assez facile à implémenter) n'est pas explicitement au programme et l'implémentation d'un tas est un exercice long et difficile pour beaucoup de candidats. Bien qu'il ne soit pas au programme et qu'il ne rentre pas dans la case "rapidement implémentable", le jury autorise l'utilisation et conseille fortement de connaître `List.sort` car il est difficile pour le jury de donner des points à l'implémentation d'un tri qui n'était pas requis... et certains perdent beaucoup de temps à implémenter un simple tri fusion.

Style de programmation. Certains candidats connaissent les `ref` mais pensent qu'il ne faut surtout pas les utiliser en vertu d'une programmation fonctionnelle pure. La pureté du programme écrit n'étant pas notée, c'est dommageable pour les candidats qui perdent du temps à cause de cela ; par exemple, en voulant utiliser une boucle `for` mais sans `ref` ou pour faire un calcul simple sur un tableau.

À l'inverse, certains candidats ne programment qu'en style impératif, et c'est parfois plus compliqué. Par exemple, si le sujet demande d'écrire une fonction qui somme deux nombres en base B représentés sous forme de listes et que les candidats utilisent des boucles `for` et `List.nth` alors cela risque d'avoir un impact sur la lisibilité, la concision, voire la complexité du code résultant.

Pour toutes ces raisons, le jury rappelle que le style de programmation est libre mais conseille d'être capable d'un peu de souplesse sur ce style de programmation en s'adaptant au sujet (par exemple en utilisant plutôt des fonctions récursives sur les listes et plutôt des boucles et des `ref` sur les tableaux).

8.7 Évolutions envisagées pour l'édition 2025.

Le format de l'édition 2025 devrait être très similaire à celui de 2024, la principale évolution souhaitée par le jury est l'utilisation d'outils qui vérifient le code des candidats de façon automatique à l'aide de tests. Aucune connaissance sur de tels systèmes de test ne sera nécessaire mais les élèves qui veulent s'entraîner à ce type d'épreuves peuvent utiliser des plateformes comme France-IOI.

