BANQUE MP INTER-ENS – SESSION 2022
RAPPORT SUR L'ÉPREUVE PRATIQUE D'ALGORITHMIQUE ET DE PROGRAMMATION DU CONCOURS COMMUN DES ÉCOLES NORMALES SUPÉRIEURES

Écoles concernées : Lyon, Paris-Saclay, Rennes, Ulm

Coefficients (en pourcentage du total d'admission)
- Lyon (toutes options) : 16,9 %
- Paris-Saclay : 13,2 %
- Rennes : 17,1 %
- Ulm : 13,3 %

Jury : Louis Jachiet, Samuel Thibault, Jill-Jênn Vie, Yannick Zakowski

Contenu de ce document

Nous rappelons l'organisation de l'épreuve, annonçons une nouveauté des éditions futures, puis faisons des remarques générales sur son déroulement cette année. Nous faisons ensuite un court compte-rendu, essentiellement statistique, pour chacun des quatre sujets. Nous proposons ensuite certains de nos corrigés : certains comportent des éléments de réponse pour la partie orale, un autre une correction en OCaml.

Organisation de l'épreuve

L'objectif de cette épreuve est d'évaluer la capacité de mettre en œuvre une chaîne complète de résolution d'un problème informatique, à savoir la construction d'algorithmes, le choix de structures de données, leurs implémentations, et l'élaboration d'arguments mathématiques pour justifier ces décisions. Le déroulement de l'épreuve est le suivant : un travail sur machine d'une durée de 3 h 30, immédiatement suivi d'une présentation orale pendant 23 minutes.

Juste avant la distribution des sujets, les candidat·es disposent d'une période de 10 minutes pour se familiariser avec l'environnement informatique et poser des questions si elles ou ils rencontrent des difficultés d'ordre pratique.

Un sujet contient typiquement une dizaine de questions écrites et une dizaine de questions orales. Tous les sujets commencent par la génération pseudo-aléatoire d'entrées pour le problème étudié. Nous invitons *fortement* les candidat·es à se familiariser à l'avance avec la manière dont ces suites pseudo-aléatoires sont générées et utilisées dans les sujets précédents afin de gagner du temps le jour de l'épreuve. En particulier, les suites pseudo-aléatoires dépendent d'un u_0 qui est donné individuellement à chaque candidat·e au début de l'épreuve.

Les questions écrites attendent des réponses purement numériques. Chaque question requiert l'implémentation d'un algorithme et son utilisation sur les entrées générées au début du sujet. Une question est typiquement divisée en sous-questions pour des entrées de plus en plus grandes, ce qui permet de tester l'efficacité de l'algorithme mis en œuvre. À la fin de la partie pratique, les candidat·es remettent au jury une fiche-réponse contenant les réponses numériques aux questions écrites.

Une aide précieuse est donnée aux candidat·es sous la forme d'une fiche-réponse type pour $\widetilde{u_0}$. Cette fiche permet de vérifier l'exactitude des réponses pour une graine différente du u_0 de la candidate ou du candidat. Il est très fortement recommandé, comme indiqué dans l'introduction des sujets, de vérifier que le générateur aléatoire se comporte comme

attendu avec la graine $\widetilde{u_0}$, pour chaque question. Les examinateurs ont encore eu quelques (rares, heureusement) cas de candidat·es traitant le sujet avec un générateur faux et donc sans possibilité de diagnostiquer efficacement leurs erreurs.

Les questions orales sont de nature plus théorique et sont destinées à être présentées pendant l'oral. Le déroulement de l'oral est le suivant : nous commençons par demander de présenter le plus efficacement possible les questions orales préparées pendant la première phase et, s'il reste du temps, s'ensuit une discussion avec le jury sur les questions non traitées. La présentation orale vise à évaluer la bonne compréhension du sujet et le recul des candidat·es. Les examinateurs s'efforcent d'aborder toutes les questions préparées pendant la première étape, et, suivant le temps disponible, des extensions de ces questions ou des questions qui n'ont pas été traitées par manque de temps. Pour réaliser un bon oral, il est important de prendre le temps de réfléchir aux questions à préparer mentionnées dans le sujet et de préparer suffisamment de notes au brouillon pour être capable d'exposer clairement et efficacement les solutions, en s'aidant du tableau dans la mesure où il est utile.

La partie écrite de l'épreuve représentait cette année 60 % de la note finale. On observe dans l'ensemble, mais pas systématiquement, une bonne corrélation entre les résultats obtenus aux deux parties.

Nouveauté du concours 2022 : lecture de fichiers ou de l'entrée standard

Pour cette édition et les éditions futures du concours, nous souhaitons diversifier l'évaluation de cette épreuve. Ainsi les candidat·es pourront être amené·es à devoir lire l'entrée standard, ou bien des fichiers contenant des instances d'objets. Le cas échéant, nous fournirons des bouts de code sur la clé USB fournie. En voici quelques exemples.

Lecture d'un fichier. On suppose que les données se trouvent dans `fichier.txt` et que l'on souhaite récupérer une liste de chaînes de caractères contenant pour chaque ligne un élément dans la liste :

`fichier.py` — Code Python 3

```python
with open('fichier.txt') as f:
    lignes = []
    for ligne in f:
        lignes.append(ligne.strip())  # Enlever le \n final
    # Alternativement aux trois lignes qui précédent on peut utiliser
    # la ligne suivante :
    lignes = f.read().splitlines()
```

`fichier.ml` — Code OCaml 4.04

```
let lignes =
  let fichier = open_in "fichier.txt" in
  let str = really_input_string f (in_channel_length f) in
  close_in f;
  String.split_on_char '\n' str
```

Pour les versions précédentes d'OCaml, il est possible d'obtenir la même chose avec d'autres fonctions de la bibliothèque `Str`, à condition de l'importer avec `open Str`.

Lecture de l'entrée standard. Supposons que l'entrée standard contienne la hauteur, la largeur, et le contenu d'une grille au format ASCII.

Entrée standard

```
3
5
#####
#...#
#####
```

Note. Le contenu d'un fichier `entree.txt` peut aussi être lu via l'entrée standard en lançant le programme d'une des façons suivantes :

```
python entree.py < entree.txt
ocaml entree.ml < entree.txt
```

où `entree.py` et `entree.ml` sont les programmes lisant sur l'entrée standard suivants :

`entree.py` Code Python 3

```python
hauteur = int(input())
largeur = int(input())
grille = []
for _ in range(hauteur):
  grille.append(input())
```

`entree.ml` Code OCaml

```
let hauteur = read_int ()
let largeur = read_int ()
let grille = List.init hauteur (fun _ -> read_line ())
```

Un exemple de lecture d'entrée est donnée dans le sujet 3 : *Compression vers* $[0, 1[$.

Remarques générales

Écriture du programme. Dans certains cas, les examinateurs ont inspecté le code des candidat·es afin de lever certaines ambiguïtés lors de leur présentation de leurs algorithmes. Cela était possible uniquement pour les candidat·es qui avaient soigné la lisibilité de leur code, et seulement si les réponses aux questions étaient facilement identifiables et exécutables. Nous conseillons ainsi aux candidat·es de soigner la lisibilité de leur code. Les candidat·es peuvent s'inspirer des propositions de corrigés fournies en annexe de ce rapport.

Génération de structures. La plupart des sujets demandent de générer des objets (graphes, arbres, chaîne de caractères ou autre) qui seront manipulés ensuite dans le sujet. Bien que chaque sujet impose son propre modèle de génération de données aléatoires, il existe des étapes communes entre les diverses années. S'entraîner sur plusieurs sujets en conditions réelles prend un temps considérable (3h30 de préparation par sujet) ainsi nous recommanderions plutôt aux candidat·es de traiter les premières questions de plusieurs sujets différents afin d'être efficace sur le début du sujet. Nous conseillons aussi aux futur·es candidat·es de s'entraîner à traiter un ou deux sujets en entier et à lire des corrections

pour avoir une idée des subtilités algorithmiques qui les attendent et maîtriser les premières questions et choix de structures algorithmiques qui sont similaires d'un sujet à l'autre.

Par ailleurs, plusieurs candidat·es mélangent le u_0 commun à tous les sujets pour tester leur code et le u_0 de leur code. Pour éviter cela, nous recommandons fortement d'éviter les copier-coller avec une version du code pour le u_0 et une pour le $\widetilde{u_0}$, et plutôt de lire le u_0 dans une variable et d'exécuter tout le code avec, cf. les différents corrigés proposés.

Gestion de l'oral. La durée de l'oral étant courte relativement au nombre de questions pouvant être traitées, nous conseillons aux candidat·es de préparer une réponse précise mais intuitive plutôt que de se perdre dans une preuve laborieuse au tableau. Si le jury n'est pas convaincu par un argument simple, il sera toujours possible de le convaincre par une preuve plus détaillée sans que cela impacte la note finale. Inversement, si le jury est convaincu par un raisonnement intuitif, on dispose alors de plus de temps pour aborder des questions globalement peu traitées et donc susceptibles de rapporter beaucoup de points. Par exemple, nombre de candidat·es se lancent dans d'interminables preuves par induction alors qu'il existe parfois une explication intuitive immédiate : nous encourageons les candidat·es à favoriser la seconde. La capacité à exposer un argument formel pour répondre à une question est évaluée dans le cadre de l'épreuve d'informatique fondamentale, tandis que l'objet de l'oral ici est de s'assurer que les candidat·es font le lien entre la résolution d'un problème informatique dans un cadre formel inédit et sa mise en pratique. Le jury saura donc apprécier le recul que démontre un argument simple et intuitif par rapport à une suite d'arguments formels désincarnés.

Sur la gestion du tableau, nous invitons les candidat·es à éviter l'écueil suivant : écrire tout leur raisonnement au tableau et ainsi perdre beaucoup de temps. L'écueil inverse, de ne pas utiliser du tout le tableau est plus rare, mais il rend parfois le raisonnement difficile à suivre. Il est difficile de donner une règle générale sur l'utilisation du tableau mais quand la preuve s'explique bien par un exemple ou un dessin, alors il ne faut pas hésiter à faire le dessin au tableau (par exemple de donner un petit graphe d'exemple) et ensuite de faire la preuve dessus à l'oral. Si le ou la candidate veut se lancer dans une preuve par induction, il peut aussi être intéressant d'écrire l'hypothèse mais pas tout le raisonnement.

Présentation des algorithmes. Certaines questions orales demandent aux candidat·es de présenter leurs algorithmes et d'analyser leur complexité. Nous encourageons vivement les candidat·es à présenter leurs algorithmes de façon claire et concise. Contrairement à ce que nous avons parfois pu constater, il ne s'agit pas de recopier un programme en pseudo-Python ou pseudo-Caml au tableau. Il faut s'efforcer de présenter (uniquement) les étapes clés de l'algorithme, en langage naturel si possible, afin de supporter efficacement l'analyse de complexité par la suite. En particulier, il est essentiel d'identifier clairement la structure itérative ou récursive d'un algorithme.

Un·e candidat·e qui propose un algorithme correct peut obtenir tous les points à l'oral même si l'implémentation de l'algorithme en question n'a pas été abordée durant la partie pratique de l'épreuve. Les candidat·es ne doivent surtout pas s'interdire d'expliquer un algorithme plus efficace que celui implémenté soit en expliquant d'abord l'algorithme implémenté puis comment l'améliorer soit en expliquant directement la version optimale.

Complexité des algorithmes. Le jury décerne des points partiels aux algorithmes justes mais à la complexité non optimale. Si l'algorithme proposé est en réalité trop naïf pour traiter les instances proposées dans le sujet, le jury saura apprécier un regard critique sur l'algorithme qui exploiterait l'analyse de complexité et un ordre de grandeur sur le nombre d'opérations élémentaires qu'un ordinateur peut effectuer. Nous n'attendons pas une estimation précise du temps de calcul mais savoir qu'il est difficile d'obtenir une réponse rapidement si l'algorithme demande d'itérer des milliards de fois une boucle.

Trop souvent, des candidat·es se sont trompé·es entre une complexité en $O(n + m)$ et une complexité en $O(n \times m)$ à cause d'une présentation de l'algorithme trop confuse. En particulier, la complexité d'un parcours en profondeur en largeur est $O(n + m)$ où n est le nombre de sommets et m est le nombre d'arêtes, et nous ne devrions pas avoir à le rappeler dans ce rapport. Par ailleurs, il est utile de savoir que la somme des degrés des nœuds est égal à deux fois le nombre d'arêtes, beaucoup de candidats l'ont heureusement retrouvé en entretien.

Langages de programmation. Les sujets sont de difficulté équivalente dans les divers langages Python et OCaml. Nous notons une tendance générale à utiliser plutôt Python que OCaml. Des élèves hésitent et passent du temps à chercher le *meilleur* langage pour le sujet. Ceci est une perte de temps, les sujets sont calibrés pour être implémentables avec le même niveau de difficulté en OCaml et en Python. On conseille donc aux candidat·es de choisir à l'avance un langage qu'ils connaissent le mieux. Cela permet de s'entraîner pour bien connaître et éviter les problèmes et limitations liées au langage. Souvent, des candidat·es se lancent en Python sans se rappeler, par exemple, que dans ce langage les appels récursifs sont limités par défaut à 1000 (cf. `sys.setrecursionlimit`) et que les listes sont représentées par des tableaux.

Pour finir, nous voulons aussi conseiller aux candidat·es d'étudier les structures de données basiques pour le langage choisi. La différence en efficacité d'un programme qui utilise une liste au lieu d'un tableau est très visible dans ce type de sujet. Cela ne veut pas dire que pendant l'oral nous espérons avoir tous les détails de l'implémentation. Au contraire, les meilleur·es candidat·es se focalisent peu sur l'utilisation ou non d'une liste ou d'une table dans leur propre implémentation.

Exemple. Nous terminons par un exemple, la présentation d'un algorithme de parcours de graphe. Voici les quatre phrases que l'on s'attend typiquement à entendre pour une telle question.

- Un algorithme de parcours de graphe part d'un sommet et suit les arêtes pour visiter les sommets du graphe connectés au sommet original.
- L'ordre de traitement des arêtes est déterminé par le choix du parcours : en largeur, on traite en priorité les arêtes par distance croissante au sommet original, et en profondeur, on traite en priorité les arêtes sortant du dernier sommet visité. Ceci induit la structure de donnée utilisée : une file (FIFO) pour un parcours en largeur, une pile (LIFO) pour un parcours en profondeur.
- Dans les deux cas, chaque arête est mise dans la structure de données exactement une fois, ce qui est assuré par un tableau de booléens déterminant si un sommet a déjà été visité ou non.
- La complexité du parcours est ainsi $O(n + m)$, où n est le nombre de sommets et m le nombre d'arêtes.

Ce dernier résultat est un résultat de cours qui doit être bien intégré : un parcours bien implémenté est linéaire en la taille du graphe. L'argument clé à bien comprendre est que l'on ne parcourt chaque sommet qu'une fois et l'on ne parcourt chaque arête qu'au plus deux fois.

Sujet 1 : Ensemble dominant dans les graphes.

Ce sujet abordait différentes stratégies pour approcher le nombre dominant d'un graphe. Le sujet commence de façon classique par la génération des graphes et leur somme de contrôle. La question somme de contrôle est là pour vérifier que les candidat·es ont généré les bons graphes. Ensuite le sujet étudies divers algorithmes pour calculer des majorations puis des calculs exacts sur des petits graphes ou des classes restreintes de graphes.

Pour la partie écrite, les questions Q1 à Q3 étaient de purs exercices de programmation et ont été correctement traitées par presque tou·tes les candidat·es. La question Q4 concernait encore la génération de graphe mais les candidat·es étaient invité à sauter cette question. La question Q5 nécessitait une légère réflexion algorithmique pour obtenir les bonnes complexités et a été bien réussie par une très large majorité des élèves. Les questions Q4 et Q6 demandaient de faire un parcours de graphes. La question Q7 demandait une bonne réflexion algorithmique pour obtenir tous les points Les questions Q7 et après ont été déterminantes dans ce sujet.

Le niveau des candidat·es est un peu en deçà de ce à quoi le jury s'attendait (peut-être un effet post-pandémie ?) mais il est plutôt satisfait du résultat : les candidat·es ont, pour une large majorité, réussi à traiter Q1, Q2, Q3, Q5 et Q6, il ne s'attendait pas à beaucoup de réponses sur Q10 ou Q11 et les candidat·es sont allé chercher les points faciles sur Q7 et Q9.

Ce sujet, dans sa version distribuée aux candidat·es, comportait une erreur typographique importante : l'arbre $T(u, S)$ était défini avec les $T(v_i, S_i)$ comme sous-arbres au lieu des $T(v_i, S_{i-1})$. Le jury présente toutes ses excuses aux candidat·es qui auraient perdu du temps sur ce problème, nous savons qu'un·e candidat·e s'est manifesté et déclare avoir perdu du temps à cause de ce problème. Nous ne mettons pas en doute que cette erreur a pu faire perdre du temps mais, pour des questions d'équité, le jury a décidé de ne pas traiter le cas de ce·tte candidat·e particulièrement. En effet, de nombreux candidat·es ont été confronté à cette erreur et nous ne pouvons pas savoir lesquel·les ont perdu du temps. Par ailleurs, telle que la définition est écrite, les candidat·es pouvaient déterminer qu'il y avait une erreur dans la définition puisque les sous-arbres auraient tous été vides. De plus la correction était "logique" : l'exemple devient correct et l'on retrouve la définition "habituelle" de parcours en profondeur.

Écrit	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Tous les points	97	94	94	25	88	69	31	6	16	3	0
Réponses partielles	100	97	97	31	94	81	50	9	47	3	9

Oral	QO1	QO2	QO3	QO4	QO5	QO6	QO7	QO8	QO9
Tous les points	88	78	50	41	56	12	9	0	0
Réponses partielles	100	100	94	66	91	62	44	6	3

Table 1. Taux de candidat·es ayant répondu à chaque question du sujet 1

Sujet 2 : La musique en Braille, c'est ambigu

Ce sujet étudiait l'ambiguïté des durées des notes de musiques lorsqu'elles sont écrites en Braille. L'idée est qu'étant donné une liste (appelée *mesure*) de signes qui peuvent chacun avoir deux interprétations de durées possibles, et la durée totale à atteindre, il s'agit de trouver une interprétation de chaque signe qui permet à la somme des durées d'atteindre exactement la durée totale ciblée.

Le sujet commençait classiquement par la génération de mesures et d'interprétations utilisées comme cas de test. Les questions écrites Q1 et Q2 étaient des traductions directes du sujet, et ont été traitées correctement par presque tou·tes les candidat·es. La question orale QO1 n'a posé aucun problème.

Un premier exercice s'intéressait alors à déterminer si pour deux mesures et interprétations données, on peut les tronquer pour obtenir la même durée. Les questions écrites Q3 et Q4 pouvaient être calculées partiellement de manière naïve. Pour obtenir tous les résultats en temps raisonnable il fallait utiliser un parcours linéaire semblable au tri fusion, ce qui a été réussi pour plus de la moitié des candidat·es. Les questions orales QO2 et QO3 étant liées, elles ont été abordées ensemble à l'oral. Seule la moitié des candidat·es a pu correctement expliquer un algorithme linéaire.

L'exercice suivant s'intéressait à déterminer combien une mesure est ambigüe pour une durée donnée. Il s'agissait d'abord de compter le nombre d'interprétations possibles puis d'en déterminer une. La question écrite Q5 était accessible avec des implémentations naïves, ce qui a permis à une très grande majorité des candidat·es d'avoir tous les points. La question orale QO4 était conçue pour fournir l'indexation de la mémoisation nécessaire pour pouvoir traiter la question écrite Q6 qui portait sur de plus grands cas, mais peu de candidat·es l'ont compris. Une majorité de candidat·es a pu calculer la valeur Q6.a, mais seul·e 1 candidat·e sur 5 a pu calculer Q6.b. Les questions orales QO4 et QO5 ont également été abordées ensemble à l'oral. Les candidat·es ont en général pu expliquer au minimum un parcours récursif exponentiel, mais l'utilisation de mémoisation, et surtout l'estimation correcte de la complexité obtenue, ont posé problème à la grande majorité des candidat·es. La question écrite Q7 était relativement semblable aux Q5 et Q6 mais nécessitait de bien identifier au sein de la récursion l'interprétation ciblée. Si les trois quarts des candidat·es ont pu calculer les premières valeurs, seul·es 3 candidat·es ont pu calculer toutes les valeurs. La question orale QO6 était similaire à la QO5 mais en a accentué la difficulté. La question orale QO7 proposant de réfléchir à compter plus efficacement le nombre d'interprétations sans l'implémenter a parfois été traitée en même temps que les QO4 et QO5. Un petit tiers des candidat·es a pu proposer au moins un début d'idée, et un dizième, à être vraiment convaincant.

L'exercice suivant s'intéressait au cas où les mesures se découpent en sections comportant plusieurs parties qui sont jouées en parallèle, ce qui complique les conditions à respecter. La question écrite Q8 nécessitait d'avoir compris la synchronisation entre les

Écrit	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Tous les points	92	90	72	56	85	21	8	5	0	0
Réponses partielles	97	97	92	64	90	59	77	5	3	0

Oral	QO1	QO2+QO3	QO4+QO5	QO6	QO7	QO8	QO9
Tous les points	100	54	33	13	10	13	5
Réponses partielles	100	100	90	49	28	26	5

Table 2. Taux de candidat·es ayant répondu à chaque question du sujet 2

parties. Bien qu'un quart des candidat·es avaient abordé la question orale QO8 associée, seul un dizième a pu fournir une réponse satisfaisante, et seul·es deux candidat·es ont pu calculer correctement la valeur demandée pour Q8. La question écrite Q9 généralisait légèrement la question écrite Q8, un·e seul·e candidat·e a pu calculer la première valeur seulement.

Enfin, la dernière partie proposait de lever les ambiguïtés en ajoutant des symboles, dont il s'agit alors de minimiser le nombre. La question orale QO9 a été réussie par 2 candidat·es, mais aucun·e n'a réussi à calculer les réponses à la question écrite Q10.

Pour des candidats souhaitant s'exercer sur ce sujet en conditions réelles, on pourra par exemple utiliser les u_0 967, 972, 978, ou 987.

Épilogue. En réalité, la musique est bien plus complexe que ce que ce sujet expose : la durée des notes n'est pas entière, à cause des duolets, triolets, etc. et la durée peut être augmentée de 50%, 75%, 87.5%, etc. en utilisant des points. La durée est notes est ainsi plutôt une fraction presque quelconque. D'autre part, la hauteur de la note aussi est ambigüe car l'octave n'est pas spécifiée, elle est éventuellement précisée avec des marques que l'on cherche aussi à minimiser.

Sujet 3 : Compression vers $[0, 1[$

Ce sujet portait principalement sur le codage arithmétique dont une variante est utilisée dans la norme de codage vidéo H.264. Une première partie demandait d'implémenter le codage de Huffman et s'intéressait à sa complexité. Une deuxième partie présentait le codage arithmétique et s'intéressait à sa complexité et efficacité. Enfin, la dernière partie se concentrait sur la construction d'arbres binaires pour deviner des objets en minimisant un coût. Il s'agissait du premier sujet de cette épreuve où il fallait lire des données réelles dans un fichier, annoncé dans le rapport de l'an dernier.

Ce sujet a été relativement bien compris. Le jury a été surpris de constater que les candidat·es ne connaissent pas tou·tes la file de priorité et sa complexité (Q4–QO1), alors qu'elle est au programme d'option informatique de MP ; c'est ce qui explique la disparité des scores à la QO1. Une grande partie des candidats a été capable de bien donner la complexité de la compression Q5–QO2 et décompression du codage arithmétique Q6–QO3. Toutefois, peu de candidats ont fait part de la difficulté de représenter des nombres flottants avec une précision arbitraire, ce qui explique que peu d'entre eux ont eu tous les points à la QO3. Peu de candidats ont soigneusement résolu les QO5 et QO6, qui demandaient du recul pour comprendre l'intérêt du codage arithmétique par rapport au code de Huffman (dont ce dernier est par ailleurs est dans le programme de MPI).

Les questions Q8 et Q9 étaient relativement indépendantes du reste, mais similaires, et demandaient l'accès à un fichier, ont été essayées par peu de candidats mais traitées soigneusement en construisant un arbre de décision minimisant l'entropie (ID3, au programme de MPI).

Écrit	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Tous les points	100	97	94	59	84	72	66	16	3	0
Réponses partielles	100	100	97	72	91	72	78	22	9	3

Oral	QO1	QO2	QO3	QO4	QO5	QO6	QO7
Tous les points	59	75	19	94	19	16	6
Réponses partielles	100	100	97	97	62	22	6

Table 3. Taux de candidat·es ayant répondu à chaque question du sujet 3

Sujet 4 : Isomorphisme et similarités de graphes

Ce sujet abordait la définition de différentes relations permettant de juger de la similarité de graphes colorés, et dont les complexités temporelles sont plus raisonnables que celle du test d'isomorphisme de graphe. Le sujet commence de façon classique par la génération des graphes et de leur somme de contrôle. Afin de générer des variants de ces graphes ayant une probabilité non nulle de satisfaire les relations définies dans la suite du sujet, nous générions alors des suites de patchs à appliquer à ces graphes : nous obtenons ainsi un générateur de pairs de graphes à la structure relativement similaire. Le sujet étudie ensuite une série de méthodes pour affecter à un graphe une forme normale dont la comparaison juge de la similarité de deux graphes. Ces formes sont essentiellement ad-hoc, introduites pour le sujet, à l'exception de la similarité de Weisfeiler-Leman concluant le sujet, dont de nombreuses variantes sont utilisées dans la littérature.

À l'écrit, si le sujet ne présentait pas de difficulté algorithmique majeure, sa relative densité de notation semble avoir mis en difficulté un certain nombre de candidat·es. En particulier, la question écrite Q3 a arrêté un tiers de ceux-ci dans leur effort dès la phase de génération des graphes : l'application du patch d'échange de sommets s'est avéré plus difficile pour les candidat·es que nous ne l'avions anticipé. La première similarité, en question Q4, a sélectionné un nouveau tiers des candidats : bien que nous ayons anticipé la difficulté et pris le parti en amont de tolérer le calcul du cardinal du multi-ensemble au lieu de celui de l'ensemble, une grande partie des candidat·es ayant abordé la question l'ont soit comprise de façon complètement incorrecte, soit se sont perdu·es dans l'implémentation du calcul demandé. La question Q5 a eu un succès marginalement meilleur que Q4. Seul·es 10% des candidat·es se sont nettement distingué·es en résolvant la Section 3, et seul·es 5% ont abordé la Section 4.

La partie orale mêlait description et analyse d'algorithmes de la partie écrite à des questions plus théoriques sur les relations de similarité proposées. Cela a souvent permis d'aborder des parties non couvertes à l'écrit par les candidat·es, bien que beaucoup se soient longuement perdu·es dans QO1 du fait de leur mécompréhension de la question écrite correspondante.

Le jury a été relativement déçu du nombre de candidat·es s'étant trouvé·es en difficulté dès l'attaque du sujet. À l'oral, il est également décevant qu'un nombre impressionnant de candidat·es ne connaissent pas la complexité d'un parcours de graphe. Une belle aisance à s'approprier les relations introduites et à illustrer leurs différences grâce à des exemples de graphes bien choisis a par contre été agréablement observée chez les candidat·es ayant eu le plus de succès sur ce sujet.

Écrit	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Tous les points	100	92	32	18	28	8	0	0
Réponses partielles	100	98	90	48	42	12	2	5

Oral	QO1	QO2	QO3	QO4	QO5	QO6	QO7	QO8
Tous les points	52	42	18	28	8	8	5	0
Réponses partielles	92	82	72	40	18	8	10	0

Table 4. Taux de candidat·es ayant répondu à chaque question du sujet 4

Ensemble dominant dans les graphes.

Épreuve pratique d'algorithmique et de programmation
Concours commun des Écoles normales supérieures

Durée de l'épreuve : 3 heures 30 minutes

Juin/Juillet 2022

ATTENTION !

N'oubliez en aucun cas de recopier votre u_0
à l'emplacement prévu sur votre fiche réponse

Important.

Il vous a été donné un numéro u_0 qui servira d'entrée à vos programmes. Les réponses attendues sont généralement courtes et doivent être données sur la fiche réponse fournie à la fin du sujet. À la fin du sujet, vous trouverez en fait deux fiches réponses. La première est un exemple des réponses attendues pour un $\widetilde{u_0}$ particulier (précisé sur cette même fiche et que nous notons avec un tilde pour éviter toute confusion !). Cette fiche est destinée à vous aider à vérifier le résultat de vos programmes en les testant avec $\widetilde{u_0}$ au lieu de u_0. Vous indiquerez vos réponses (correspondant à votre u_0) sur la seconde et vous la remettrez à l'examinateur à la fin de l'épreuve.

En ce qui concerne la partie orale de l'examen, lorsque la description d'un algorithme est demandée, vous devez présenter son fonctionnement de façon schématique, courte et précise. Vous ne devez en aucun cas recopier le code de vos procédures !

Quand on demande la complexité en temps ou en mémoire d'un algorithme en fonction d'un paramètre n, on demande l'ordre de grandeur en fonction du paramètre, par exemple : $O(n^2)$, $O(n \log n)$,...

Il est recommandé de commencer par lancer vos programmes sur de petites valeurs des paramètres et de ***tester vos programmes sur des petits exemples que vous aurez résolus préalablement à la main ou bien à l'aide de la fiche réponse type fournie en annexe.***

Enfin, il est recommandé de lire l'intégralité du sujet avant de commencer afin d'effectuer les bons choix de structures de données dès le début.

1 Préliminaires

1.1 Rappel sur la récursion

La plupart des langages ont recours à une "pile d'appel" pour gérer la récursion. Cette pile d'appel a généralement une limite : une limite fixe de 1 000 appels récursifs en Python et une limite en taille de quelques méga-octets pour les langages comme OCaml ou C/C++ (ce qui généralement autorise plus de 100 000 appels récursifs). Attention, certaines questions de ce sujet manipulent des données suffisamment larges pour dépasser ces limites, surtout en Python !

1.2 Notations

On rappelle que pour deux entiers naturels a et b, $a \mod b$ désigne le reste de la division entière de a par b, c'est à dire l'unique entier r avec $0 \leq r < b$ tel que $a = k \times b + r$ pour $k \in \mathbb{N}$.
Ce sujet portant sur des graphes non orientés, chaque utilisation du mot graphe doit être entendue comme graphe non orienté sauf mention explicite du contraire. Un graphe est donc, pour ce sujet, un couple $G = \langle V, E \rangle$ où V est l'ensemble fini des sommets ou nœuds du graphe et E est l'ensemble des arêtes, chaque arête étant un ensemble de sommets de cardinal 2. Notez que cette définition interdit les boucles (un sommet n'est jamais relié à lui-même par une arête) ainsi que les arêtes multiples (deux nœuds ne peuvent être reliés que zéro ou une fois).
Par souci de simplicité, dans ce sujet, V est toujours constitué des nombres de 1 à n avec n le nombre de nœuds tandis que E est vu comme une liste de paires, chaque paire (x, y) de la liste aura $x < y$ et correspondra à l'arête $\{x, y\}$. Ainsi, un graphe est entièrement caractérisé par un nombre de nœuds et une liste d'arêtes.

1.3 Ensemble dominant d'un graphe

Étant donné un graphe $G = \langle V, E \rangle$, un sous-ensemble D des sommets ($D \subseteq V$) est dit dominant quand chaque sommet s de V est dominé, c'est à dire que s est dans D ou que s a un voisin dans D (un sommet dominé peut avoir plusieurs voisins dans D ou être dans D et avoir des voisins dans D). Dans ce sujet nous nous intéresserons au nombre dominant, c'est à dire à la taille minimale d'un ensemble dominant. L'objectif du sujet c'est de calculer ou d'approcher ce nombre dominant par des majorations.
Par exemple, dans le graphe de la figure 1, on a un nombre dominant de 5 car l'ensemble $\{1, 2, 4, 7, 10\}$ est dominant et aucun ensemble de taille 4 ne domine ce graphe. Noter que l'ensemble dominant n'est généralement pas unique, par exemple, le graphe de la figure 1 est aussi dominé, notamment, par $\{1, 2, 5, 8, 9\}$ mais le nombre dominant est unique car on s'intéresse au cardinal minimal d'un ensemble dominant.

1.4 Génération de nombres pseudo-aléatoires

Étant donné u_0, on définit la récurrence :

$$\forall t \in \mathbb{N},\ u_{t+1} = (900\,007 \times u_t) \mod 1\,000\,000\,007$$

L'entier u_0 vous est donné, et doit être recopié sur votre fiche réponse avec vos résultats. Une fiche réponse type vous est donnée en exemple, et contient tous les résultats attendus pour une valeur de u_0 différente de la vôtre (notée $\widetilde{u_0}$). Il vous est conseillé de tester vos algorithmes avec cet $\widetilde{u_0}$ et de comparer avec la fiche de résultat fournie. Pour chaque calcul demandé, avec le bon

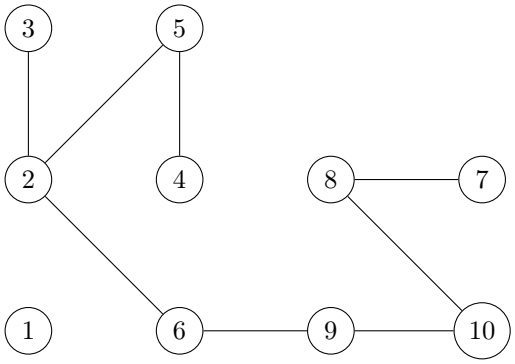


FIGURE 1 – Graphe correspondant à $G(10, 4, 4)$ pour $\widetilde{u_0}$

choix d'algorithme le calcul ne devrait demander qu'au plus de l'ordre de la seconde, jamais plus d'une minute.

Question 1 *Calculer les valeurs suivantes :*

a) $u_1 \mod 1\,000$ ***b)*** $u_{12} \mod 1\,000$ ***c)*** $u_{1234} \mod 1\,000$ ***d)*** $u_{2345678} \mod 1\,000$

```
let u0 = read_int ()

let max_u = 2345678

let u = Array.make (max_u+1)  u0
let _ =
  for i = 0 to max_u-1 do
    u.(i+1) <- (u.(i) * 900_007) mod 1_000_000_007 ;
  done
```

2 Génération pseudo-aléatoire de graphes

2.1 Listes $P(n, m, k)$ et graphes $G(n, m, k)$

Étant une liste l de paires d'entiers ($l = (x_1, y_1), \ldots, (x_n, y_n)$) et un entier k, on définit la fonction filtre $f_k(l)$ qui renvoie la liste des paires (x, y) de l telles que $u_{17\times y+x} \mod k = 0$. Étant donnés n et m, on définit $L(n, m)$ comme étant la liste de paires (x, y) avec $1 \leq x < y \leq n$ et $y \leq x+m$. On définit alors la liste $P(n, m, k)$ comme étant la liste $f_k(L(n, m))$ et le graphe $G(n, m, k)$ comme étant le graphe à n nœuds dont la liste d'arêtes est $P(n, m, k)$.

Question 2 *Calculer le nombre d'arêtes des graphes suivants :*

a) $G(123, 10, 5)$ ***b)*** $G(1\,234, 10, 5)$ ***c)*** $G(12\,345, 12, 6)$ ***d)*** $G(54\,321, 22, 8)$

```
let filtre_pair k (x,y) =  u.(17*(max x y)+(min x y)) mod k = 0 && x<>y

let graphe_ligne_filtre n m k =
  let t = Array.make (n+1) [] in
  for i = 1 to n do
    for j = max 1 (i-m) to min n (i+m) do
      if filtre_pair k (i,j)
      then t.(i) <- j::t.(i)
    done ;
    t.(i) <- List.rev t.(i) ;
  done ;
    t


let compte_aretes g = (g |> Array.map List.length |> Array.fold_left (+) 0)/2
```

Question à développer pendant l'oral 1 *Décrire la structure de donnée que vous avez choisie pour représenter les graphes. En particulier, vous devez décrire la complexité des opérations suivantes :*

- *l'occupation mémoire (en ordre de grandeur pour n nœuds et e arêtes),*
- *la complexité de récupérer la liste des voisins d'un nœud, ordonnée par indices (si a et b sont des voisins alors a apparaît avant b dans la liste quand $a < b$).*

Nous avons choisi de représenter les graphes comme des listes d'adjacence qui permettent pour chaque nœud de récupérer la liste de ses voisins en $O(1)$. Par la manière dont le graphe est construit ces listes sont créées triées donc récupérer tous les voisins en ordre est aussi $O(1)$ (puis $O(d)$ pour parcourir la liste des d voisins). L'occupation mémoire est d'un tableau du nombre de nœuds, et pour chaque nœud, d'une liste de ses voisins. Donc $O(n+E)$ où est le nombre d'arêtes.

2.2 Somme de contrôle d'un graphe

Pour un graphe $G = \langle V, E \rangle$, on définit sa somme de contrôle $SC(G)$ à l'aide de la formule suivante :

$$SC(G) = \left(\sum_{x \in V} \sum_{y \mid \{x,y\} \in E} (x+y)^2 + y \right) \mod 1\,000\,000$$

Question 3 *Calculer la somme de contrôle des graphes suivants :*

a) $G(123, 10, 5)$ ***b)*** $G(1\,234, 10, 5)$ ***c)*** $G(12\,345, 12, 6)$ ***d)*** $G(54\,321, 22, 8)$

```
let checksum g =
  let rec foo i tot =
    if i = Array.length g
    then tot
    else
      foo (i+1) ((List.fold_left (fun ac j -> ac+(i+j)*(i+j)+j) tot g.(i))
      ↪  mod 1_000_000)
  in
  foo 1 0
```

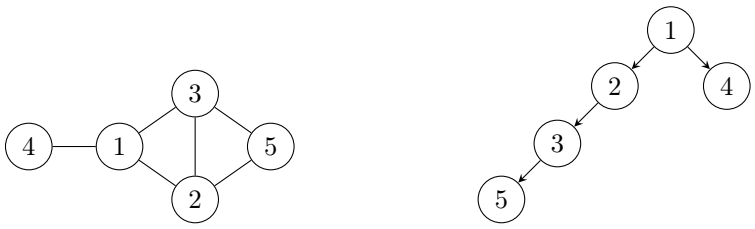
2.3 Génération d'arbre à partir de graphes

Attention, cette section 2.3 concerne la génération d'arbre et n'est utile que pour les questions 4 et 8. Si vous bloquez, n'hésitez pas à sauter cette section et à vous concentrer d'abord sur la question 5.

Étant donné un graphe G, l'ensemble $A(u, S)$ est défini comme l'ensemble des nœuds v tels que v est accessible depuis le nœud u en évitant un ensemble de nœuds S, c'est à dire qu'il y a un chemin de u à v où aucun des nœuds du chemin n'est dans S. Formellement, si $N(u)$ renvoie les voisins de u, on a :

$$A(u,S) = \begin{cases} \emptyset & \text{quand } u \in S \\ \{u\} \bigcup\limits_{v \in N(u)} A(v, S \cup \{u\}) & \text{sinon} \end{cases}$$

Soit u un nœud de G et S un ensemble de nœuds nous allons définir l'arbre $T(u, S)$ enraciné en u. Pour cela, on note $v_1, \ldots, v_k$ les voisins de u dans G avec $v_1 < \cdots < v_k$ et on pose $S_0 = S \cup \{u\}$ puis $S_{j+1} = S_j \cup A(v_{j+1}, S_j)$. Pour $u \in S$ alors $T(u, S)$ est l'arbre vide. Pour $u \notin S$, l'arbre $T(u, S)$ est l'arbre enraciné en u dont les sous-arbres sont les $T(v_i, S_{i-1})$.
Pour finir, étant donné un graphe G, on définit $\text{Arbre}(G) = T(1, \{\})$. Dans la figure suivante on voit à gauche un graphe G et à droite l'arbre $\text{Arbre}(G)$.



Notez que les arbres enracinés sont généralement vus comme des graphes orientés mais nous n'utiliserons l'orientation que pour présenter la construction des arbres et pour le calcul de la somme de contrôle. Dès qu'il s'agira de trouver un ensemble dominant pour un arbre, celui-ci sera vu comme un graphe non orienté, c'est à dire qu'un sommet sera dominé par D s'il est dans D ou si son parent est dans D ou si l'un de ses enfants est dans D.
Pour un arbre $\mathcal{T}$, la somme $SC(\mathcal{T})$ de contrôle de $\mathcal{T}$ est définie comme $\sum\limits_{x \text{ parent de } y} (x+y)^2 + y \mod 1\,000\,000$. Notez que cette formule ressemble à celle des graphes mais elle est "orientée" : dans la version graphe, une arête $\{x, y\}$ est comptée deux fois tandis qu'ici elle n'est comptée qu'une seule fois.

Question 4 *Calculer la somme de contrôle pour les arbres suivants :*

a) $Arbre(G(123, 10, 5))$

b) $Arbre(G(1\,234, 10, 5))$

c) $Arbre(G(12\,345, 12, 6))$

d) $Arbre(G(54\,321, 22, 8))$

```
type tree =
  | Noeud of int * tree list

let arbre_de g =
  let n = Array.length g  in
  let seen = Array.make n false in
  let rec foo x =
    if seen.(x) then failwith "Double visite ?" ;
    seen.(x) <- true ;
    Noeud(x, List.fold_left
               (fun ac el -> if seen.(el) then ac else (foo el)::ac)
               []
               g.(x))
  in
  foo 1

let rec checksum_arbre (Noeud(racine,fils)) =
  List.fold_left
    (fun ac (Noeud(noeud,descendant)) ->
      (ac +
         (racine+noeud)*(racine+noeud)+noeud +
         (checksum_arbre (Noeud(noeud,descendant)))
      ) mod 1000000
    )
    0
    fils

let rec taille_arbre = function
  | Noeud(x,l) -> 1+(List.map taille_arbre l |> List.fold_left (+) 0)
```

$Arbre(G)$ est l'arbre qui correspond au parcours en profondeur de notre graphe et est donc obtenu par un parcours en profondeur. Le parcours en profondeur parcourt chaque nœud une fois et de chaque nœud il regarde ses voisins et donc l'algorithme considère chaque arête deux fois ce qui donne une complexité de $O(n + E)$.

3 Majoration du nombre dominant par une heuristique

Nous étudions maintenant un algorithme pour calculer un ensemble dominant. Cet algorithme, que l'on nommera $\mathcal{A}_1$, fonctionne de la façon suivante : l'algorithme définit $D_0 = \emptyset$ et itère ensuite sur i de 1 à n. Pour chaque i, si le nœud i est dominé par D_{i-1} alors $D_i = D_{i-1}$ et sinon $D_i = \{i\} \cup D_{i-1}$. Le résultat de l'algorithme est D_n.

Question à développer pendant l'oral 2 *Justifier que cet algorithme produit un ensemble dominant mais qu'il n'est pas toujours de taille minimale. Est-ce que la taille de l'ensemble dominant produit par l'algorithme reste proche du nombre dominant pour tous les graphes ? Justifier.*

Pour montrer que c'est un ensemble dominant on peut procéder par récurrence en montrant que D_i domine tous les nœuds d'indice $\leq i$ et donc que D_n est un ensemble dominant.
Si l'on a un graphe en étoile avec tous les nœuds (sauf n) reliés uniquement à n, notre algorithme va choisir un ensemble dominant prenant tous les nœuds de 1 à $n-1$ alors qu'il suffisait de prendre le nœud n.

Question 5 *Calculer la taille de l'ensemble renvoyé par $\mathcal{A}_1$ sur les graphes suivants :*

a) $G(123, 10, 5)$ ***b)*** $G(1\,234, 10, 5)$ ***c)*** $G(12\,345, 12, 6)$ ***d)*** $G(54\,321, 22, 8)$

```
let dans_l_ordre g =
  let n = Array.length g in
  let pris = Array.make n false in
  let rec compte i =
    if i >= n
    then 0
    else
      if not pris.(i)
      then
        begin
          List.iter (fun x -> pris.(x)<-true) g.(i) ;
          1 + compte (i+1)
        end
      else compte (i+1)
  in
  compte 1
```

Question à développer pendant l'oral 3 *Décrire la complexité de votre algorithme en fonction du nombre de nœuds et du nombre d'arêtes des graphes manipulés. Pour le calcul de complexité, le temps de création du graphe ne sera pas pris en compte.*

Notre algorithme maintient un tableau où la case i stocke un booléen déterminant si le nœud i est dominé par notre ensemble en cours de construction. De cette façon il suffit d'itérer sur les nœuds, de tester en $O(1)$ s'ils sont dominé et si non de marquer les voisins comme étant dominés. Notre algorithme traverse chaque nœud une fois et chaque arête au plus une fois, la complexité est donc $O(n+E)$.

4 Majoration du nombre dominant par la décomposition en composantes connexes

La décomposition d'un graphe en composantes connexes est la partition des sommets d'un graphe en des ensembles $V_1, \ldots, V_k$ tels que deux sommets u et v sont dans le même ensemble V_i si et seulement si, il y a un chemin entre u et v. Les V_i sont les composantes connexes du graphe et chaque V_i forme un sous graphe que l'on peut étudier individuellement. L'entier k est appelé le nombre de composantes connexes (on suppose qu'aucun des V_i n'est vide). Enfin on dit qu'un sommet est isolé s'il est seul dans sa composante connexe.

Question à développer pendant l'oral 4 *Montrer que tout graphe à n nœuds sans sommet isolé admet un ensemble dominant de taille au plus $n/2$. Les propositions prouvant seulement la borne de $(n+1)/2$ obtiendront une partie des points.*

On procède par récurrence sur la forme du graphe. Si le graphe a moins de 2 nœuds, la propriété est évidente.
L'idée générale de la récurrence c'est de partir d'un graphe G sans nœuds isolés, de trouver un sommet a à enlever, d'appeler la propriété de récurrence sur un graphe G' où a et certains de ses voisins ont été enlevés. Comme G' contient (au moins) 2 sommets de moins que G on obtient la propriété que l'on voulait.
Pour que cette méthode marche, il ne faut pas créer de sommets isolés. Pour éviter cela, nous allons distinguer le cas où G contient un sommet de degré 1 ou non.
Si G contient un sommet b de degré 1, on va considérer son voisin a et appliquer la propriété en enlevant a et tous les voisins de a de degré 1. Il ne peut pas rester de sommets isolés car, à part le sommet a, les autres sommets retirés n'étaient reliés que à a.
Si G ne contient pas de sommet de degré 1, on va prendre une paire a, b de sommets voisins et l'on va retirer a, b et tous les nœuds qui n'ont que a et b comme voisins. Cela ne créé pas de sommets isolés car tous les nœuds avaient initialement un degré 2 (ou plus) et les nœuds de G ne peuvent perdre que a et b comme voisins (et ceux qui n'avaient que a et b comme voisins sont retirés).

Nous étudions alors un algorithme, que l'on nommera $\mathcal{A}_2$ pour borner le nombre dominant. L'algorithme $\mathcal{A}_2$ fonctionne de la façon suivante : il calcule d'abord les tailles $T_1 \ldots T_k$ des diverses composantes connexes puis renvoie $\sum_i t(T_i)$ où $t(1) = 1$ et $t(\ell) = \lfloor \ell/2 \rfloor$ pour $\ell > 1$.

Question 6 *Calculer la valeur renvoyée par $\mathcal{A}_2$ sur les graphes suivants :*

a) $G(123, 10, 5)$ ***b)*** $G(1\,234, 10, 5)$ ***c)*** $G(12\,345, 12, 6)$ ***d)*** $G(54\,321, 22, 8)$

```
let cout_cc cout g =
  let n = Array.length g in
  let pris = Array.make n false in
  let rec marque ac = function
    | [] -> ac
    | x::t ->
       if pris.(x)
       then marque ac t
       else
         begin
           pris.(x) <- true ;
           marque (ac+1) (g.(x)@t)
         end
  in

  let rec compte ac i =
    if i>=n
    then ac
    else
      if pris.(i)
      then compte ac (i+1)
      else
        let taille = marque 0 [i] in
         compte (ac+cout taille) (i+1)
  in
  compte 0 1


let cout_dune_composante x =
    if x = 1 then 1 else (x/2)

(* à utiliser avec cout_cc cout_dune_composante mon_graphe *)
```

Question à développer pendant l'oral 5 *Décrire la complexité de votre algorithme en fonction du nombre de nœuds et du nombre d'arêtes des graphes manipulés. Pour le calcul de complexité, le temps de création du graphe ne sera pas pris en compte.*

Notre algorithme va maintenir un tableau des nœuds déjà vus. En itérant les nœuds, si on passe un nœud qui n'est pas déjà vu, on lance un parcourt en profondeur marquant tous les nœuds de la composante connexe comme vus et on calcule sa taille. Chaque nœud ne sera donc vu qu'au plus deux fois et chaque arête aussi considérée deux fois car deux parcours différents parcourent des composantes différentes. L'algorithme sera donc en complexité $O(n + E)$.

5 Majoration du nombre dominant par une autre heuristique

L'algorithme $\mathcal{A}_1$ que nous avons vu plus haut traite les sommets dans l'ordre de leurs indices (d'abord le sommet 1, puis le 2, etc.). L'algorithme $\mathcal{A}_3$ que nous introduisons maintenant, fonctionne avec un score de dominance. Pour un ensemble D de sommets, le score $s(D)$ est le nombre de sommets dominés par D. Au lancement de l'algorithme, on a $D_0 = \emptyset$ et ensuite, tant que $s(D_i) \neq n$ l'algorithme choisit le sommet u tel que $s(D_i \cup \{u\})$ est maximal et définit $D_{i+1} = D_i \cup \{u\}$. En cas d'égalité entre plusieurs sommets qui donnent le meilleur score, l'algorithme choisit celui qui a le plus petit indice.

Question 7 *Calculer la taille de l'ensemble renvoyé par $\mathcal{A}_3$ sur les graphes suivants :*

a) $G(123, 10, 5)$ ***b)*** $G(1\,234, 10, 5)$ ***c)*** $G(9876, 1234, 6)$ ***d)*** $G(54\,321, 22, 8)$

Attention, le graphe **c)** diffère des questions précédentes !

Question à développer pendant l'oral 6 *Décrire la complexité de votre algorithme en fonction du nombre de nœuds et du nombre d'arêtes des graphes manipulés. Pour le calcul de complexité, le temps de création du graphe ne sera pas pris en compte.*

Ce que l'on veut c'est de calculer les gains de chaque nœud puis d'énumérer les nœuds par gain décroissant. Pour savoir rapidement le gain d'un nœud, on va maintenir dans un tableau la valeur courante du gain de chaque nœud. À chaque fois qu'un nœud x est sélectionné on va alors diminuer le gain de tous les voisins des voisins de x qui ne sont pas déjà dominés.
Pour récupérer le prochain nœud à traiter, on va utiliser un tableau todo. Ce tableau est initialisé avec, pour chaque d, la liste triée des nœuds dont le degré est supérieur ou égal à $d-1$ et donc qui ont, initialement, un gain de d (ou plus mais celui peut diminuer au cours de l'algorithme). En parcourant ces listes par d décroissant on sait quels sont les nœuds qui ont potentiellement un gain de d (ce que l'on peut vérifier avec notre tableau gain).
Chaque nœud ne sera marqué qu'une fois et on n'explorera donc les voisins qu'une fois, ce qui coûte $O(n+E)$. Chaque nœud sera stocké dans todo et considéré $O(d)$ fois, donc $O(E)$ fois au total. En combinant tout cela on obtient bien une complexité de $O(n+E)$.

```
let algo_meilleur_score g =
  let n = Array.length g - 1 in
  let gain = Array.map (fun l -> 1+List.length l) g in
  let pris = Array.make (n+1) false in
  let todo = Array.make (n+1) [] in

  for x = n downto 1 do
    for d = 0 to gain.(x) do
      todo.(d) <- x::todo.(d)
    done
  done ;

  let couvre i =
    let aEnlever = List.filter (fun e -> not pris.(e)) (i::g.(i)) in
    List.iter (fun e -> pris.(e) <- true) aEnlever ;
    List.iter (fun e -> List.iter (fun j -> gain.(j) <- gain.(j)-1)
    ↪  (e::g.(e))) aEnlever
  in

  let rec couvre_tout cout gainCour =
    if gainCour = 0
    then cout
    else
      match todo.(gainCour) with
      | [] -> couvre_tout cout (gainCour-1)
      | a::q ->
         todo.(gainCour) <- q ;
         let ncout =
           if gain.(a) = gainCour
           then (couvre a ; cout+1)
           else cout in
         couvre_tout ncout gainCour
  in
  couvre_tout 0 n
```

6 Calcul exact du nombre dominant

Dans cette partie, nous allons trouver le véritable nombre dominant. On note $nd(G)$ le nombre dominant du graphe G.

6.1 Cas d'un arbre

Attention, on rappelle que les arbres manipulés sont des graphes non orientés.
Pour $\mathcal{T}$ un arbre, on définit : nombreDominantAvecRacinePrise($\mathcal{T}$) la taille minimale d'un ensemble dominant pour $\mathcal{T}$ qui contient la racine et nombreDominantSaufRacine($\mathcal{T}$), la taille minimale d'un ensemble qui domine tous les nœuds de $\mathcal{T}$ sauf éventuellement la racine.

Question à développer pendant l'oral 7 *Étudier la relation entre les valeurs de $nd(\mathcal{T})$, $nombreDominantAvecRacinePrise(\mathcal{T})$, $nombreDominantSaufRacine(\mathcal{T})$ et les valeurs de ces fonc-*

tions sur les sous-arbres de $\mathcal{T}$. En déduire un algorithme pour calculer $nd(\mathcal{T})$, quelle est sa complexité ?

Dans un arbre, dans un ensemble dominant D, chaque nœud est dominé soit parce qu'il est dans D, soit parce que son parent est dans D soit parce que l'un de ses enfants est dans D.
Étant $\mathcal{T}$ dont les sous-arbres sont $\mathcal{T}_1, \ldots, T\_k$, on a :

- Si on prend la racine de $\mathcal{T}$, on domine les racines des $\mathcal{T}_i$ donc un ensemble dominant qui prend la racine de $\mathcal{T}$ peut se calculer comme la racine et un ensemble pseudo-dominant (=qui ne domine pas la racine) de chaque $\mathcal{T}_i$.
- Un ensemble qui domine $\mathcal{T}$ sauf éventuellement la racine de $\mathcal{T}$ c'est soit un ensemble dominant de $\mathcal{T}$ qui prend la racine de $\mathcal{T}$, soit un ensemble dominant pour chacun des sous-arbres.
- Un ensemble dominant pour $\mathcal{T}$, c'est soit un ensemble dominant qui prend la racine de $\mathcal{T}$; soit un ensemble dominant pour chaque sous-arbre dont au moins une prend la racine.

Pour traduire cela en algorithme efficace, il faut remarquer qu'on peut calculer les trois valeurs simultanément dans une seule récursion. De plus pour calculer la taille minimale d'un ensemble dominant composé d'une ensemble dominant pour chaque $\mathcal{T}_i$ plus un ensemble dominant pour un $\mathcal{T}_j$ qui prend la racine, cela peut se calculer comme la somme des nombres dominants de tous les $\mathcal{T}_i$ plus le delta taille d'ensemble dominant de $\mathcal{T}_j$ qui prend la racine moins nombre dominant de $\mathcal{T}_j$. L'algorithme est donc linéaire en la taille de l'arbre.

Question 8 *Calculer le nombre dominant pour les arbres suivants :*

a) $Arbre(G(123, 10, 5))$ ***b)*** $Arbre(G(1\,234, 10, 5))$

c) $Arbre(G(12\,345, 12, 6))$ ***d)*** $Arbre(G(54\,321, 22, 8))$

```
let rec optimal_sur_arbre t =
  let rec foo = function (* renvoie cout,prend_tete,couvre_tete *)
  | Noeud(x, l) ->
     let couts_enfants = List.map foo l in
     let somme_cout_enfants = List.fold_left (fun ac (cout,_,_) -> ac+cout) 0
     ↪  couts_enfants in
     let enfant_pris = List.exists (fun (_,prend,_) -> prend) couts_enfants
     ↪  in
     let enfant_non_couvert = List.exists (fun (_,_,couvert) -> not couvert)
     ↪  couts_enfants in
     if enfant_non_couvert
     then (somme_cout_enfants+1, true,true)
     else if enfant_pris then (somme_cout_enfants,false,true)
     else (somme_cout_enfants,false,false)
  in
  match foo t with | (cout,_,couvert)-> if couvert then cout else cout+1
```

6.2 Cas des petits graphes

Dans cette sous-partie, il vous est demandé d'écrire un algorithme qui calcule le nombre dominant pour un graphe quelconque. Pour cela, il faut générer tous les sous-ensembles de sommets et tester

si ceux-ci dominent ou non le graphe. Attention les cas **e)** et **f)** requièrent une exploration optimisée de l'ensemble des sous-ensembles de sommets.

Question 9 *Calculer le nombre dominant pour les graphes suivants :*

a) $G(10, 5, 2)$ ***b)*** $G(12, 3, 2)$ ***c)*** $G(14, 4, 2)$

d) $G(18, 3, 2)$ ***e)*** $G(45, 20, 2)$ ***f)*** $G(60, 25, 2)$

```
exception Trouve

let optimal g =
  let n = Array.length g in
  let pris = Array.make n 0 in

  let marque i delta =
    List.iter (fun el -> pris.(el) <- pris.(el)+delta) (i::g.(i)) ;
    List.filter (fun el->pris.(el) == 1) (i::g.(i)) |> List.length
  in

  let rec cherche i nbRestants nbCouverts =
    if nbCouverts+nbRestants >= n-1 then raise Trouve ;
    if i < n && nbRestants > 0
    then
      begin
        let ajout = marque i 1 in
        if ajout > 1 then cherche (i+1) (nbRestants-1) (nbCouverts + ajout) ;
        let _ =  marque i (-1) in
        cherche (i+1) (nbRestants) nbCouverts
      end
  in
  let rec foo i =
   try
     cherche 1 i 0 ;
     foo (i+1)
   with Trouve -> i
  in
  foo 1
```

6.3 Cas de nos graphes générés avec un petit paramètre m

Les graphes que l'on génère dans ce sujet ont une forme bien particulière quand le paramètre m est petit même si n est grand. De la même façon que dans un arbre un nœud ne peut être dominé que par son parent, par lui-même ou par l'un de ses descendants, dans nos graphes un nœud x ne peut être dominé que par un nœud y avec $x - m \leq y \leq x + m$.

Question 10 *Calculer le nombre dominant pour les graphes suivants :*

a) $G(2\,000, 8, 4)$ ***b)*** $G(3\,000, 7, 5)$

c) $G(4\,000, 6, 6)$ ***d)*** $G(5\,000, 6, 6)$

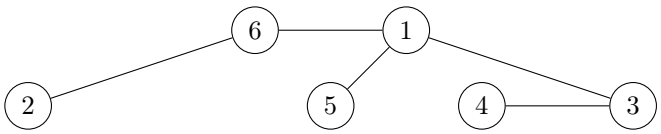
```
let opti_gen n m k =
  let g = graphe_ligne_filtre n m k in

  let p2 = Array.make (2*m+2) 1 in

  let _ =
    for i = 0 to 2*m do
      p2.(i+1) <- p2.(i)*2
    done
  in

  let nb_fenetres = p2.(2*m) in
  let vu = Array.make_matrix (n+1) nb_fenetres (-1) in

  let marque x fenetre =
    fenetre land (lnot (List.map (fun e ->  p2.(m+e-x)) (x::g.(x)) |>
    ↪  List.fold_left (lor) 0))
  in

  let rec opti i fenetre =
    if i > n
    then (if fenetre = 0 then 0 else 10000000000)
    else
      begin
        if vu.(i).(fenetre) < 0
        then
          begin
            vu.(i).(fenetre) <- 10000000000000 ;
            let nouv_fenetre = fenetre + (if i+m <= n then nb_fenetres else
            ↪  0) in
            let fenetre_prend = marque i nouv_fenetre in
            if fenetre_prend mod 2 = 0
            then
              begin
                vu.(i).(fenetre) <- 1 + opti (i+1) (fenetre_prend/2) ;
                if nouv_fenetre mod 2 = 0
                then  vu.(i).(fenetre) <- min
                                            vu.(i).(fenetre)
                                            (opti (i+1) (nouv_fenetre/2)) ;
              end
          end ;
        vu.(i).(fenetre)
      end
  in

  opti 1 (nb_fenetres-p2.(m))
```

Question à développer pendant l'oral 8 *Détailler votre algorithme et donner sa complexité.*

Par la forme du graphe, on sait qu'un nœud x ne peut dominé que des nœuds dans l'intervalle $[x-m; x+m]$. Nous allons donc travailler récursivement avec un algorithme qui prend un x et la liste des nœuds dominés dans l'intervalle $[x-m; x+m]$ (tous les nœuds en dessus de $x-m$ sont considérés dominés et ceux après $x+m$ considérés non dominés) et détermine quelle est la taille minimale d'un ensemble dominant tous les nœuds en n'ajoutant à l'ensemble que des nœuds y avec $y \geq x$.
Récursivement on a deux choix, poser x ou non, avant de passer à $x+1$ ce que l'on ne peut faire que si $x-m$ est bien dominé après avoir fait le choix de poser x.
Le nombre de possibilités pour l'intervalle $[x-m; x+m]$ de taille $2m+1$ est 2^{2m+1} et le nombre de positions est n donc l'algorithme est en $O(2^m \times n)$

6.4 Cas d'une classe spéciale de graphes

Pour cette section, nous allons générer un nouveau type de graphe. Le graphe $C(n)$ est défini comme le graphe à n nœuds dont les arêtes sont $\{i, 1 + (u_i \mod n)\}$ pour tout $1 \leq i \leq n$ avec $1 + (u_i \mod n) \neq i$.
Noter que $C(n)$ contient toujours n nœuds mais pas forcément n arêtes. Il est possible, par exemple, que $1 + (u_i \mod n) = i$ ou que l'on ait des paires i, j telles que $j = 1 + (u_i \mod n)$ et $i = 1 + (u_j \mod n)$. Pour $\widetilde{u_0}$, voici $C(6)$, qui a 5 arêtes :



Question 11 *Calculer les valeurs suivantes :*

a) $nd(C(10))$

b) $nd(C(100))$

c) $\sum_{5 \leq i \leq 1\,000} nd(C(i))$

d) $\sum_{100\,000 \leq i \leq 100\,100} nd(C(i))$

Question à développer pendant l'oral 9 *Détailler votre algorithme, donner sa complexité et justifier sa correction.*

Fiche réponse type : Ensemble dominant dans les graphes.

$\widetilde{u_0}$: 503

Question 1

- **a)** 521
- **b)** 165
- **c)** 668
- **d)** 896

Question 2

- **a)** 215
- **b)** 2398
- **c)** 24476
- **d)** 148185

Question 3

- **a)** 348194
- **b)** 362363
- **c)** 997270
- **d)** 48746

Question 4

- **a)** 340200
- **b)** 305976
- **c)** 959830
- **d)** 959765

Question 5

- **a)** 57
- **b)** 510
- **c)** 5158
- **d)** 19403

Question 6

- **a)** 63
- **b)** 626
- **c)** 6246
- **d)** 27242

Question 7

a) 36

b) 342

c) 83

d) 11800

Question 8

a) 40

b) 413

c) 4155

d) 18253

Question 9

a) 3

b) 5

c) 4

d) 6

e) 5

f) 5

Question 10

a) 473

b) 994

c) 1658

d) 2080

Question 11

a) 4

b) 35

c) 180601

d) 3638480

La musique en Braille, c'est ambigu

Épreuve pratique d'algorithmique et de programmation
Concours commun des Écoles normales supérieures

Durée de l'épreuve : 3 heures 30 minutes

Juin/Juillet 2022

ATTENTION !

N'oubliez en aucun cas de recopier votre u_0
à l'emplacement prévu sur votre fiche réponse

Important.

Il vous a été donné un numéro u_0 qui servira d'entrée à vos programmes. Les réponses attendues sont généralement courtes et doivent être données sur la fiche réponse fournie à la fin du sujet. À la fin du sujet, vous trouverez en fait deux fiches réponses. La première est un exemple des réponses attendues pour un $\widetilde{u_0}$ particulier (précisé sur cette même fiche et que nous notons avec un tilde pour éviter toute confusion !). Cette fiche est destinée à vous aider à vérifier le résultat de vos programmes en les testant avec $\widetilde{u_0}$ au lieu de u_0. Vous indiquerez vos réponses (correspondant à votre u_0) sur la seconde et vous la remettrez à l'examinateur à la fin de l'épreuve.

En ce qui concerne la partie orale de l'examen, lorsque la description d'un algorithme est demandée, vous devez présenter son fonctionnement de façon schématique, courte et précise. Vous ne devez en aucun cas recopier le code de vos procédures !

Quand on demande la complexité en temps ou en mémoire d'un algorithme en fonction d'un paramètre n, on demande l'ordre de grandeur en fonction du paramètre, par exemple : $O(n^2)$, $O(n \log n)$,...

Il est recommandé de commencer par lancer vos programmes sur de petites valeurs des paramètres et de ***tester vos programmes sur des petits exemples que vous aurez résolus préalablement à la main ou bien à l'aide de la fiche réponse type fournie en annexe***. Enfin, il est recommandé de lire l'intégralité du sujet avant de commencer afin d'effectuer les bons choix de structures de données dès le début.

1 Préliminaires

On rappelle que pour deux entiers naturels a et b, $a \bmod b$ désigne le reste de la division entière de a par b.

Suite de nombres pseudo-aléatoires

On fixe $M = 2^{31} - 1 = 2\,147\,483\,647$.
On définit la suite $\overline{u}(n)$ par récurrence :

$$\overline{u}(0) = u_0, \quad \forall n \in \mathbb{N}, \overline{u}(n+1) = (16\,807 \times \overline{u}(n) + 17) \bmod M$$

u_0 vous ayant été donné, et doit être reporté sur votre fiche réponse.
On définit alors la suite

$$u(n) = \overline{u}(n \bmod 999\,983)$$

Question 1 *Calculez* $u(n) \bmod 997$ *pour*

a) $n = 16$, ***b)*** $n = 1024$, ***c)*** $n = 1\,000\,000$.

2 Introduction

Le Braille est un système d'écriture permettant aux personnes aveugles de lire des documents. Il utilise des cellules composées de 6 points qui sont chacun soit saillant soit à plat, il n'y a ainsi que $2^6 = 64$ combinaisons possibles pour chaque cellule, l'alphabet est donc limité à 64 symboles. Parvenir à encoder le plus d'information possible avec chaque symbole est donc un enjeu très important pour limiter le nombre de pages d'un ouvrage.
Notamment, la musique est écrite en Braille en utilisant autant que possible le principe d'un seul symbole par note musicale. La combinatoire des notes possibles est cependant très grande : pour chaque note il faut à la fois encoder sa hauteur, sa durée, son expression... Il a donc été décidé de rendre l'encodage ambigu en utilisant le même symbole pour différentes notes. Par exemple, le symbole

⠒

encode une note de hauteur LA, mais qui peut durer soit 4 temps (une ronde), soit 1/4 de temps (une double croche). Il en est de même pour tous les différents symboles représentant les différentes notes : pour un symbole donné, la hauteur de la note est connue, mais la durée est ambiguë, entre deux durées possibles.
Pour lever l'ambiguïté, on utilise le fait que pour une suite de notes appelée mesure, on connait la durée totale[1]. Il faut alors lire l'ensemble des symboles de la mesure, et constater que seule une interprétation de ces symboles est possible, car c'est la seule qui respecte la durée totale de la mesure.
En pratique, les musiciens devinent assez facilement la durée des notes grâce à leur culture musicale. Un ordinateur qui est dépourvu d'une telle culture est cependant obligé d'énumérer les interprétations possibles.

1. 4 temps, typiquement

3 Modélisation

Pour simplifier la modélisation, on ignore la hauteur des notes, et l'on utilise des durées entières. On modélise ainsi le problème de la façon suivante :

- L'alphabet $\mathcal{A}$ utilisé est composé de 4 symboles représentant les notes : $\mathcal{A} = \{0, 1, 2, 3\}$.
- À chaque symbole s est associé à la fois une durée courte $d_0(s)$ et une durée longue $d_1(s)$ de la note. Voici leurs valeurs :

Symbole	Durée courte	Durée longue
0	$d_0(0) = 1$	$d_1(0) = 16$
1	$d_0(1) = 2$	$d_1(1) = 32$
2	$d_0(2) = 4$	$d_1(2) = 64$
3	$d_0(3) = 8$	$d_1(3) = 128$

- Une mesure de longueur k est une suite de symboles : $M = (s_{k-1}, s_{k-2}, \ldots, s_0)$
- Une interprétation est une suite de 0 et de 1 de même longueur que la mesure : $I = (i_{k-1}, i_{k-2}, \ldots, i_0)$ qui exprime pour chaque symbole si c'est une durée courte (0) ou si c'est une durée longue (1) qui est exprimée par l'interprétation.
- Pour une mesure M, la durée $D(M, I)$ de la mesure M avec l'interprétation I est ainsi égale à

$$D(M, I) = \sum_{n=0}^{k-1} d_{i_n}(s_n)$$

- On utilisera également pour identifier les interprétations le numéro $N(I)$ associé à une interprétation :

$$N(I) = \sum_{n=0}^{k-1} i_n \times 2^n$$

Par exemple, avec $M = (2, 2, 3)$ et l'interprétation $I = (1, 0, 0)$,

$$D(M, I) = d_1(2) + d_0(2) + d_0(3) = 64 + 4 + 8 = 76$$

$$N(I) = 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 4$$

Pour k et n donnés, on note M_k^n la mesure de k symboles suivante :

$$M_k^n = (u(n + k - 1) \mod 4, u(n + k - 2) \mod 4, \ldots, u(n) \mod 4)$$

On note I_k^n l'interprétation suivante :

$$I_k^n = (u(2n + k - 1) \mod 2, u(2n + k - 2) \mod 2, \ldots, u(2n) \mod 2)$$

Pour dénoter facilement une durée possible de la mesure M_k^n, on utilise la durée de cette interprétation, notée D_k^n :

$$D_k^n = D(M_k^n, I_k^n)$$

Par exemple, pour $\widetilde{u_0}$, on a

$$\begin{aligned}\widetilde{M_6^0} &= (1,0,0,3,3,2)\\ \widetilde{I_6^0} &= (1,0,0,1,1,0)\\ \widetilde{D_6^0} &= 294\end{aligned}$$

Question 2 *Calculez*

a) D_{16}^0, ***b)*** $N\left(I_6^0\right)$, ***c)*** $N\left(I_{16}^0\right)$, ***d)*** $D_{10\,000}^0$.

Question à développer pendant l'oral 1 *Expliquez vos implémentations et leur complexité.*

Pour calculer D, il suffit de parcourir en parallèle les tableaux `M` et `I`, ce qui donne une complexité linéaire en k.
Pour calculer N, on peut à chaque étape multiplier un accumulateur par 2, ce qui donne une complexité linéaire en le nombre de bits, donc k.

4 Même longueur ?

Lorsque l'on pioche deux paires (M, I) aléatoirement comme on l'a effectué à la section précédente, il y a très peu de chances que les deux durées correspondantes soient les mêmes. On se demande s'il suffit de tronquer un peu les mesures pour obtenir la même durée.

Notons $T(X, l)$ la troncation d'une suite à ses l premières valeurs : pour $X = (x_{k-1}, x_{k-2}, \ldots, x_0)$, $T(X, l) = (x_{k-1}, x_{k-2}, \ldots, x_{k-l})$ (avec $l \leq k$).

Par exemple, pour $\widetilde{u_0}$, on a

$$\begin{aligned}\widetilde{M_5^{10}} &= (3,2,1,3,3)\\ \widetilde{I_5^{10}} &= (0,1,0,1,1)\\ \widetilde{D_5^{10}} &= D\left(\widetilde{M_5^{10}}, \widetilde{I_5^{10}}\right) = 330\\ \widetilde{M_5^{27}} &= (2,2,1,2,0)\\ \widetilde{I_5^{27}} &= (0,1,0,0,0)\\ \widetilde{D_5^{27}} &= D\left(\widetilde{M_5^{27}}, \widetilde{I_5^{27}}\right) = 75\end{aligned}$$

Mais si l'on tronque $\widetilde{M_5^{10}}$ à ses 3 premières notes et $\widetilde{M_5^{27}}$ à ses 4 premières notes (ainsi que leurs interprétations respectives), on a

$$\begin{aligned}D\left(T\left(\widetilde{M_5^{10}}, 3\right), T\left(\widetilde{I_5^{10}}, 3\right)\right) &= 74 \quad =\\ D\left(T\left(\widetilde{M_5^{27}}, 4\right), T\left(\widetilde{I_5^{27}}, 4\right)\right) &= 74\end{aligned}$$

Et il s'agit d'ailleurs de la plus grande durée que l'on peut obtenir ainsi.
On note pour cette opération d'égalisation :

$$E(M_1, I_1, M_2, I_2) = (l_1, l_2) \quad \text{tel que } D\left(T(M_1, l_1), T(I_1, l_1)\right) = D\left(T(M_2, l_2), T(I_2, l_2)\right) = x \text{ avec } x \text{ maximal}$$

Pour $\widetilde{u_0}$ on a donc $E\left(\widetilde{M_5^{10}}, \widetilde{I_5^{10}}, \widetilde{M_5^{27}}, \widetilde{I_5^{27}}\right) = (3, 4)$

Question 3 *Calculez*

a) $E\left(M_{32}^0, I_{32}^0, M_{32}^{70}, I_{32}^{70}\right)$,

b) *le nombre d'entiers* $n \in [\![0, 99]\!]$ *tels que* $E\left(M_{32}^n, I_{32}^n, M_{32}^{n+70}, I_{32}^{n+70}\right) \neq (0, 0)$,

Question à développer pendant l'oral 2 *Expliquez comment précalculer très efficacement toutes les valeurs* $D(T(M_1, l_1), T(I_1, l_1))$ *et* $D(T(M_2, l_2), T(I_2, l_2))$ *pour les différentes valeurs de* l_1 *et* l_2. *Expliquez comment cela permet de calculer* $E(M_1, I_1, M_2, I_2)$ *très efficacement.*

Le sujet proposait de calculer toutes les valeurs intermédiaires. Cela peut se faire linéairement en k en calculant $D(T(M_1, 0), T(I_1, 0))$, $D(T(M_1, 1), T(I_1, 1))$, etc. en accumulant simplement, et de même pour M_2 et I_2. Le résultat se retrouve alors déjà trié par ordre croissant.
Le problème revient alors à trouver la valeur identique la plus grande dans les deux tableaux de résultat.

Question 4 *Calculez*

a) *le nombre d'entiers* $n \in [\![0, 9999]\!]$ *tels que* $E\left(M_{100}^n, I_{100}^n, M_{100}^{n+200}, I_{100}^{n+200}\right) \neq (0, 0)$,

b) *la plus petite valeur de* $k \geq 1$ *pour laquelle tous les entiers* $n \in [\![0, 99]\!]$ *sont tels que* $E\left(M_k^n, I_k^n, M_k^{n+2k}, I_k^{n+2k}\right) \neq (0, 0)$.

Question à développer pendant l'oral 3 *Expliquez votre implémentation du calcul de* E *et sa complexité.*

Il suffit d'effectuer un parcours en parallèle sur les deux tableaux (assez similaire au fonctionnement du tri fusion), en partant de leur fin, en remontant à chaque pas du côté du tableau qui a la plus grande valeur, jusqu'à parvenir à deux valeurs identiques, au pire deux 0 en début de tableau. Le parcours se fait ainsi en temps linéaire en k aussi.
Une approche un peu plus efficace, effectivement proposée par certain es candidat es, est de ne pas utiliser le pré-calcul proposé par le sujet, et de parcourir directement les mesures en parallèle, en accumulant les durées au fur et à mesure, ce qui évite d'avoir à calculer toutes les valeurs intermédiaires. La complexité reste la même.

5 Problème d'ambiguïté

Le problème de l'ambiguïté de la lecture du Braille musical peut s'exprimer ainsi :

Étant données une mesure M et une durée ciblée $\mathcal{D}$, trouver une interprétation I telle que la durée de l'interprétation I de la mesure M est égale à $\mathcal{D}$:

$$D(M, I) = \mathcal{D}$$

Selon la mesure et la durée données, il peut y avoir plusieurs interprétations possibles, ou bien une seule, voire aucune. On notera l'ensemble des interprétations solution $\mathbb{I}(M, \mathcal{D})$, et le nombre d'interprétations solution $|\mathbb{I}(M, \mathcal{D})|$.
Par exemple :

- Avec $M = (0, 0, 1)$ et $\mathcal{D} = 64$, l'interprétation $I = (1, 1, 1)$ donne $D(M, I) = 64$, et c'est la seule interprétation qui permet d'obtenir cette durée pour M.
- avec $M = (0, 0, 1)$ et $\mathcal{D} = 32$, il n'existe pas d'interprétation qui permet d'obtenir cette durée.
- avec $M = (0, 0, 0)$ et $\mathcal{D} = 18$, les trois interprétations $I_1 = (1, 0, 0)$, $I_2 = (0, 1, 0)$, et $I_3 = (0, 0, 1)$ permettent d'obtenir cette durée.

Lorsque plusieurs interprétations sont possibles, on retiendra celle qui est la première dans l'ordre lexicographique. Pour le dernier exemple ci-dessus $M = (0, 0, 0)$ et $\mathcal{D} = 18$, on retiendra donc l'interprétation $I_3 = (0, 0, 1)$. Il se trouve que c'est celle dont le numéro est le plus petit : $N(I_3) = 1$. On la notera $I(M, \mathcal{D})$.
Par exemple, pour $\widetilde{u_0}$, on a

$$\begin{aligned} I\left(\widetilde{M_5^0}, \widetilde{D_5^0}\right) &= (0, 0, 1, 1, 0) \\ I\left(\widetilde{M_6^0}, \widetilde{D_6^0}\right) &= (0, 1, 1, 1, 1, 0) \\ N\left(I\left(\widetilde{M_6^0}, \widetilde{D_6^0}\right)\right) &= 30 \end{aligned}$$

Question 5 *Calculez*

a) $\left|\mathbb{I}(M_6^0, D_6^0)\right|$ ***b)*** $\left|\mathbb{I}(M_{12}^0, D_{12}^0)\right|$

Question à développer pendant l'oral 4 *Expliquez pourquoi durant l'énumération des interprétations possibles, l'état du calcul ne dépend que de la position au sein de la mesure et la durée totale des notes précédant cette position.*

Lorsque l'on énumère récursivement toutes les interprétations possibles, à un instant du calcul donné on a choisi les interprétations de i notes du début de la mesure jusqu'à la position atteinte i, produisant une durée partielle d pour ce début de mesure. Pour les autres notes de la mesure, l'ensemble des interprétations possibles ne dépend que de la durée restante $\mathcal{D} - d$ pour compléter

la durée totale ciblée, indépendemment du détail du choix des interprétations des notes en début de mesure.
Autrement dit, après avoir calculé la combinatoire des interprétations possibles des notes au-delà de la position i, on peut mémoiser celle-ci dans un tableau indexé par i et d. On peut ainsi la réutiliser pour toutes les énumérations de début de mesure de i notes produisant une durée partielle d.

Question 6 *Calculez*

a) $\left|\mathbb{I}(M_{24}^0, D_{24}^0)\right|$

b) $\left|\mathbb{I}(M_{64}^0, D_{64}^0)\right| \bmod 100\,000$

Question à développer pendant l'oral 5 *Expliquez votre algorithme et sa complexité.*

On parcourt les solutions récursivement en utilisant un accumulateur pour se souvenir de la longueur déjà accumulée. En retour de la récursion, on accumule le nombre de solutions trouvées.
On obtient alors une complexité exponentielle : on effectue deux appels récursif à chaque note, on se retrouve au final 2^k fois dans le cas de base.
Pour accélérer grandement, on mémoise tel que suggéré à la question précédente.
Pour limiter la taille du tableau de mémoisation, on peut arrêter la récursion dès que l'accumulateur dépasser la longueur visée, $\mathcal{D}$.
Au final il n'est nécessaire au pire que de calculer les $k * \mathcal{D}$ valeurs du tableau, $\mathcal{D}$ étant limité à $128 * k$, et chaque calcul s'effectuant en $O(1)$.

Question 7 *Calculez*

a) $N\left(I(M_6^0, D_6^0)\right)$

b) $N\left(I(M_{12}^0, D_{12}^0)\right)$

c) $N\left(I(M_{42}^0, D_{42}^0)\right) \bmod 100\,000$

d) $N\left(I(M_{128}^0, D_{128}^0)\right) \bmod 100\,000$

Question à développer pendant l'oral 6 *Expliquez votre algorithme et sa complexité.*

Le principe est le même que la question précédente, sauf que l'on retourne dès que l'on a trouvé une solution, il n'y a donc le plus souvent pas besoin de parcourir toutes les 2^k possibilités.
Si l'on n'a vraiment pas de chance il faut réellement tout parcourir pour atteindre la dernière solution (car toutes les durées sont longues). Mais la longueur ciblée est pour une interprétation aléatoire, donc il est probable qu'environ la moitié des interprétations soient longues, et à permutation près des interprétations, elles se retrouvent "tassées" à la fin de la mesure dans l'interprétation trouvée. On aura donc plutôt parcouru de l'ordre de $2^{k/2}$ solutions.
Avec la mémoisation on accélère de nouveau grandement, car l'on choisit très rapidement environ la première moitié des interprétations comme étant courtes, et l'on énumère seulement la deuxième moitié, en pratique on a besoin de calculer environ $k/2 * \mathcal{D}$ valeurs.

Question à développer pendant l'oral 7 *Décrivez comment on pourrait calculer directement le nombre total de solutions sans avoir à les énumérer une par une (on pourra éventuellement partir d'une solution connue). On ne cherchera pas à l'implémenter.*

On peut passer d'une solution à une autre avec deux opérations de base :
- Échanger l'interprétation de deux notes ayant le même symbole mais une interprétation différente. Pour un symbole donné apparaissant n fois avec m interprétations courtes (par exemple), il s'agit donc de placer les m interprétations courtes parmi les n symboles, donc C_n^m possibilités.
- Échanger l'interprétation d'une paire de notes de même symbole avec l'interprétation d'une note de symbole ayant des durées deux fois plus longues.

Pour la première opération il n'y a pas besoin d'énumérer toutes ces interprétations, on en a directement le nombre. On peut alors parcourir l'espace des interprétations en utilisant seulement la deuxième opération.

6 Polyphonie

Il arrive parfois que les mélodies se séparent en deux parties, certains musiciens jouant l'une pendant que d'autres jouent l'autre. Souvent les différentes parties couvrent toute la mesure et leurs durées sont donc égales à celle de la mesure. Il arrive cependant souvent aussi que les différentes parties se séparent et se rejoignent à différents moments de la mesure. La mesure est alors découpée en sections comportant chacune une ou plusieurs parties. Les parties d'une même section doivent alors avoir des durées identiques[2], que l'on appelle durée de la section. La durée de la mesure doit alors être égale à la somme des durées des sections.
Il est donc à noter que la durée des sections n'est pas connue, elle est déduite, le cas échéant, de la seule possibilité d'interprétation des notes au sein des différentes parties tout en respectant l'égalité des durées des parties au sein d'une même section. Sinon plusieurs interprétations sont donc possibles.
À l'écrit, les différentes sections de la mesure sont séparées par une marque (∷ , que l'on notera 5). Au sein d'une section, les différentes parties sont séparées par une marque (∵ , que l'on notera 4).

Par exemple, la mesure $M = (1, 5, 0, 4, 3, 2, 1, 1)$ est composée d'une première section n'ayant qu'une partie (1), puis d'une deuxième section comportant deux parties (0) et $(3, 2, 1, 1)$. Cela signifie donc que le premier musicien joue $(1, 0)$ pendant que le deuxième musicien joue $(1, 3, 2, 1, 1)$. Avec une durée de mesure 48, la seule interprétation possible est $I = (1, 0, 1, 0, 0, 0, 0, 0)$, car la seule possibilité de durée commune pour les deux parties de la deuxième section est 16.
Note : les marques 4 et 5 n'ont pas besoin d'indication d'interprétation, par convention on leur donne l'interprétation 0.

On note pour $X = (x_0, x_1, \ldots, x_{k-1})$ et $Y = (y_0, y_1, \ldots, y_{l-1})$ la concaténation

$$X \cdot Y = (x_0, x_1, \ldots, x_{k-1}, y_0, y_1, \ldots, y_{l-1})$$

Attention : dans les questions suivantes, seules M_6^0 et I_6^0 de la première section dépendent de u_0 ; pour les autres sections, qui utilisent $\widetilde{M_1}, \widetilde{I_1}, \widetilde{M_2}, \widetilde{I_2}$, utilisez toujours la valeur pour $\widetilde{u_0}$, c'est-à-dire utilisez toujours :

2. Pour que les musiciens terminent leurs parties respectives en même temps.

$$\begin{aligned}
\widetilde{M_5^0} &= (0,0,3,3,2)\\
\widetilde{M_5^{414}} &= (2,1,3,2,2)\\
\widetilde{I_5^0} &= (0,0,1,1,0)\\
\text{etc.}
\end{aligned}$$

Question 8 *On note*

$$\begin{aligned}
\widetilde{M_1} &= \widetilde{M_5^0}\\
\widetilde{M_2} &= \widetilde{M_5^{414}}\\
\widetilde{I_1} &= \widetilde{I_5^0}\\
M &= M_6^0 \cdot (5) \cdot \widetilde{M_1} \cdot (4) \cdot \widetilde{M_2} \cdot (5) \cdot \widetilde{M_1} \cdot (4) \cdot \widetilde{M_2}\\
D &= D(M_6^0, I_6^0) + 2 \times D(\widetilde{M_1}, \widetilde{I_1})
\end{aligned}$$

C'est-à-dire que les deux musiciens jouent d'abord ensemble M_6^0, puis jouent séparément l'un $\widetilde{M_1}$ et l'autre $\widetilde{M_2}$, terminent ensemble, et de nouveau jouent séparément l'un $\widetilde{M_1}$ et l'autre $\widetilde{M_2}$, et terminent ensemble exactement à la fin de la mesure.
De nouveau, on notera l'ensemble des interprétations solution $\mathbb{I}(M, \mathcal{D})$, et le nombre d'interprétations solution $|\mathbb{I}(M, \mathcal{D})|$.

a) *Calculez $|\mathbb{I}(M, D)|$.*

Question à développer pendant l'oral 8 *Expliquez votre implémentation et sa complexité.*

Pour fournir une implémentation générique, l'état pendant l'énumération s'est complexifié : en plus de la position courante dans la mesure et la durée totale accumulée précédemment, l'état dépend également de la durée de la partie en cours de parcours, et éventuellement de la durée (égale) des parties précédemment parcourues de la même section.
Durant le parcours, lorsque l'on rencontre un symbole 5, on remet ces deux dernières notions à 0. Lorsque l'on rencontre un symbole 4, on initialise la durée des parties précédemment parcourue à celle de la partie que l'on vient d'achever (et l'on remet celle-ci à zéro).
Il se trouve que dans le cas proposé il n'y a que deux voix possibles, on peut donc confondre ces deux notions : lorsque l'on parcourt la première partie d'une section, on n'a pas encore de durée de parties précédentes ; lorsque l'on parcourt la deuxième et dernière partie d'une section, on n'a plus besoin de la durée de la partie précédente, on peut se contenter de la durée restante pour la partie en cours.
Cela permet ainsi d'effectuer une mémoisation dans un tableau de taille $\mathcal{D} * (\mathcal{D}+1) * k$.

Ceci étant dit, les cas que le sujet demandait à calculer étaient plutôt réduits. On pouvait ainsi simplement énumérer les durées possibles pour M_6^0, et les combiner avec les durées possibles pour $\widetilde{M_1}$ et $\widetilde{M_2}$. On obtient ainsi une complexité de $O(\mathcal{D}^2 * k)$. En précalculant les durées possibles pour $\widetilde{M_1}$ et $\widetilde{M_2}$, on peut atteindre une complexité de $O(\mathcal{D}^2)$.

Question 9 *On note*

$$\begin{aligned} M &= M_6^0 \cdot (5) \cdot \widetilde{M_1} \cdot (4) \cdot \widetilde{M_2} \\ D &= D(M_6^0, I_6^0) + D(\widetilde{M_1}, \widetilde{I_1}) \end{aligned}$$

On indique pour vérification des valeurs de $\widetilde{M_k^n}$ *et* $\widetilde{I_k^n}$ *que vous obtenez, utilisées dans les calculs demandés ci-dessous, y compris pour votre propre* u_0 *:*

$$\begin{aligned} \widetilde{M_6^0} &= [1,0,0,3,3,2] \\ \widetilde{M_6^{167}} &= [2,3,1,2,1,2] \\ \widetilde{N(I_6^0)} &= 38 \\ \widetilde{M_{15}^0} &= [3,2,1,3,3,2,2,3,3,1,0,0,3,3,2] \\ \widetilde{M_{15}^{3646}} &= [3,0,3,3,0,3,1,3,3,2,3,3,0,0,2] \\ \widetilde{I_{15}^0} &= 23\,782 \\ \widetilde{M_{30}^3} &= [2,2,2,1,2,0,0,1,2,1,0,3,1,0,1,2,0,0,3,2,1,3,3,2,2,3,3,1,0,0] \\ \widetilde{M_{30}^{658}} &= [1,2,0,1,3,2,3,2,0,0,1,1,0,1,2,0,1,1,3,3,2,2,3,0,3,1,1,0,0,3] \\ \widetilde{I_{30}^3} &= 9\,097\,587 \end{aligned}$$

a) *Calculez* $|\mathbb{I}(M,D)|$ *pour* $\widetilde{M_1} = \widetilde{M_6^0}, \widetilde{I_1} = \widetilde{I_6^0}, \widetilde{M_2} = \widetilde{M_6^{167}}$.

b) *Calculez* $|\mathbb{I}(M,D)| \bmod 100\,000$ *pour* $\widetilde{M_1} = \widetilde{M_{15}^0}, \widetilde{I_1} = \widetilde{I_{15}^0}, \widetilde{M_2} = \widetilde{M_{15}^{3646}}$.

c) *Calculez* $|\mathbb{I}(M,D)| \bmod 100\,000$ *pour* $\widetilde{M_1} = \widetilde{M_{30}^3}, \widetilde{I_1} = \widetilde{I_{30}^3}, \widetilde{M_2} = \widetilde{M_{30}^{658}}$.

7 Désambiguïsation

Pour lever l'ambiguïté et permettre donc aux musiciens d'être sûrs de l'interprétation qu'ils doivent utiliser pour jouer la mesure, on ajoute des marques qui apportent une information sur la durée.
Une première marque $\vdots\cdot$ indique que la note qui suit est de durée longue, tandis qu'une autre marque $\cdot\vdots\cdot$ indique que la note qui suit est de durée courte.
Par exemple, pour une mesure $M = (0,0,0)$ et l'interprétation voulue $I = (0,0,1)$ (et donc une durée 18), il suffit d'utiliser une marque avant la troisième note pour préciser qu'elle doit être interprétée avec une durée longue. Avec cette information, seule l'interprétation voulue est alors possible, sans avoir besoin de précisions sur les autres notes.

L'objectif est alors de minimiser le nombre de marques utilisées pour lever l'ambiguïté. On souhaite également placer ces marques le plus possible en début de mesure[3]. À noter qu'on préfèrera minimiser le nombre de marques, avant de préférer les placer plutôt en début de mesure. Ainsi, par exemple pour la mesure $M = (0,2,1,0)$ et l'interprétation voulue $I = (0,0,1,0)$ (et donc une durée 38), on aurait pu placer une marque devant la troisième note pour indiquer qu'elle doit être interprétée avec une durée longue, et les autres notes sont alors forcément interprétées

3. Pour que les musiciens soient le moins surpris possible.

avec une durée courte. Mais l'on préfèrera placer une marque devant la toute première note pour indiquer qu'elle doit être interprétée avec une durée courte. Le reste de l'interprétation est alors tout autant sans ambiguïté. On notera cela $A(M, I) = (1, 0, 0, 0)$ pour exprimer que pour lever l'ambiguïté selon ces principes, une marque a été utilisée sur la première note (signalé par la présence du 1) mais pas sur les autres (signalé par la présence des 0). Comme pour les interprétations, on utilisera $N(A)$ pour abréger leur écriture.
Ainsi, par exemple pour $\widetilde{u_0}$ on a (attention, la valeur de n est différente par rapport à la section 5) :

$$\begin{aligned} A\left(\widetilde{M_5^{14}}, \widetilde{I_5^{14}}\right) &= (0, 0, 1, 0, 0) \\ A\left(\widetilde{M_6^{14}}, \widetilde{I_6^{14}}\right) &= (0, 0, 0, 0, 1, 0) \\ N\left(A\left(\widetilde{M_6^{14}}, \widetilde{I_6^{14}}\right)\right) &= 2 \end{aligned}$$

Question 10 *Calculez*

a) $N(A(M_8^{14}, I_8^{14}))$ ***b)*** $N(A(M_{12}^{14}, I_{12}^{14}))$ ***c)*** $N(A(M_{16}^{14}, I_{16}^{14}))$

Question à développer pendant l'oral 9 *Expliquez votre implémentation et sa complexité.*

On peut énumérer tous les jeux de marques en itérant sur les nombres de 0 à $2^k - 1$, et vérifier (avec un algorithme similaire au précédent) s'il y a exactement une interprétation possible. On peut terminer plus rapidement dès que l'on constate qu'il y a plusieurs interprétations, sans besoin de parcourir absolument tout l'arbre des 2^k possibilités à chaque fois.
Il vaut mieux énumérer les jeux de marques dans l'ordre de préférence, pour s'arrêter dès que l'on trouve un A pour lesquel il n'y a qu'une interprétation. La réduction n'est cependant pas drastique, car l'on constate qu'il n'y a pas d'effet de "tassement", on est en général obligé de parcourir la plupart des cas.

Question à développer pendant l'oral 10 *Dans quelle mesure peut-on utiliser une mémoisation ? (ou une programmation dynamique)*

Si l'on énumère tous les jeux de marques avant de vérifier s'il y a une seule interprétation, la mémoisation dépend du A considéré, et cela coûte cher de repartir de zéro pour chaque valeur de A. On peut se contenter de mémoiser la fin des parcours, dans la partie où toutes les marques ont déjà été positionnées, cela améliore un peu le temps mais ce n'est pas miraculeux.
Une autre approche est d'agréger les types de notes, pour calculer directement le nombre de notes de chaque type et durée qu'il faut marquer pour éviter l'ambiguïté. La mémoisation est alors beaucoup moins coûteuse car il n'y a que 4 types de notes, et pour un type de note donné il est inutile de marquer à la fois une note d'interprétation courte et une note d'interprétation longue. On peut ainsi obtenir une complexité $O(|D|^5)$.

Fiche réponse type : La musique en Braille, c'est ambigu

$\widetilde{u_0}$: 42

Question 1

a) 781

b) 374

c) 805

Question 2

a) 979

b) 38

c) 23782

d) 417139

Question 3

a) (29, 31)

b) 69

Question 4

a) 9292

b) 113

Question 5

a) 2

b) 38

Question 6

a) 87344

b) 12704

Question 7

a) 30

b) 1503

c) 21439

d) 35807

Question 8

a) 1392

Question 9

a) 206

b) 60464

c) 78560

Question 10

a) 2

b) 514

c) 37378

Compression vers $[0, 1[$.

Épreuve pratique d'algorithmique et de programmation
Concours commun des Écoles normales supérieures

Durée de l'épreuve : 3 heures 30 minutes

Juin/Juillet 2022

ATTENTION !

N'oubliez en aucun cas de recopier votre u_0
à l'emplacement prévu sur votre fiche réponse

Important.

Il vous a été donné un numéro u_0 qui servira d'entrée à vos programmes. Les réponses attendues sont généralement courtes et doivent être données sur la fiche réponse fournie à la fin du sujet. À la fin du sujet, vous trouverez en fait deux fiches réponses. La première est un exemple des réponses attendues pour un $\widetilde{u_0}$ particulier (précisé sur cette même fiche et que nous notons avec un tilde pour éviter toute confusion !). Cette fiche est destinée à vous aider à vérifier le résultat de vos programmes en les testant avec $\widetilde{u_0}$ au lieu de u_0. Vous indiquerez vos réponses (correspondant à votre u_0) sur la seconde et vous la remettrez à l'examinateur à la fin de l'épreuve.

En ce qui concerne la partie orale de l'examen, lorsque la description d'un algorithme est demandée, vous devez présenter son fonctionnement de façon schématique, courte et précise. Vous ne devez en aucun cas recopier le code de vos procédures !

Quand on demande la complexité en temps ou en mémoire d'un algorithme en fonction d'un paramètre n, on demande l'ordre de grandeur en fonction du paramètre, par exemple : $O(n^2)$, $O(n \log n)$,...

Il est recommandé de commencer par lancer vos programmes sur de petites valeurs des paramètres et de ***tester vos programmes sur des petits exemples que vous aurez résolus préalablement à la main ou bien à l'aide de la fiche réponse type fournie en annexe***.

Enfin, il est recommandé de lire l'intégralité du sujet avant de commencer afin d'effectuer les bons choix de structures de données dès le début.

Ce sujet est composé d'une introduction ainsi que de trois parties pouvant être traitées de façon indépendante. Ainsi, nous vous invitons à bien lire tout le sujet et d'éviter de rester trop longtemps bloqué. Pour résoudre la partie 3 (sauf sa dernière question), sur votre clé se trouve un jeu de données, ainsi qu'un bout de code qui le lit.

Dans ce sujet, le **logarithme** est toujours en base 2. En Python, cette fonction est calculée par `from math import log` puis `log(x, 2)`. En OCaml, la fonction `log` calcule le logarithme en base e donc pour gagner du temps, on définira une fonction `log2` en s'appuyant sur le fait que $\log_2 x = \log_b x / \log_b 2$ pour toute base entière $b \geq 2$.

Définitions

Un **mot** w de taille $n = |w|$ sur un alphabet Σ est constitué de n lettres $w[0] \cdots w[n-1]$ de Σ.

En théorie de l'information, on définit une **source** comme un objet qui produit de façon aléatoire une séquence de symboles, ce qui nous intéresse étant la probabilité d'apparition de chaque symbole. Dans ce sujet, pour simplifier on suppose qu'une source est un mot dont les symboles sont des lettres de Σ. Ainsi la probabilité d'apparition d'un symbole de S est son nombre d'occurrences dans S, divisé par la longueur de S.

L'**entropie** d'une source S dont chaque symbole x_i apparaît avec probabilité $p_i > 0$ est donnée par $H(S) = -\sum_i p_i \log_2 p_i$ et s'exprime en bits, cf. un exemple à la table 1. L'entropie est le nombre minimal de bits pour encoder un symbole en moyenne.

Compresser est une opération qui consiste à **coder**, c'est-à-dire transformer une source S en un **code** $\mathcal{C}$ qui peut être représenté par un certain nombre de bits. Dans ce sujet on s'intéresse à une compression sans perte, ce qui veut dire qu'il est toujours possible de **décoder**, c'est-à-dire récupérer la source originale à partir du code.

Écriture positionnelle en base b Soit b un entier supérieur ou égal à 2, K un entier positif. Un réel $r \in [0, 1]$ peut s'écrire comme un mot infini $a_1 a_2 \cdots$ de symboles de $\{0, \ldots, b-1\}$ tel que

$$r = \sum_{k=1}^{\infty} a_k b^{-k}.$$

Notez que cette écriture n'est pas nécessairement unique, comme par exemple $0{,}999\ldots = 1$ en base 10. À des fins de terminaison, on s'intéressera à une écriture tronquée à des mots de K caractères, et dans leur grande mansuétude, les examinateurs accepteront toutes les écritures possibles.

TABLE 1 – La source `ABBESSES.` émet les symboles `. A B E S` avec probabilité non nulle. Son entropie vaut $H(\texttt{ABBESSES.}) = 2 \times (-1/9 \log 1/9) - 2 \times (2/9 \log 2/9) - 3/9 \log 3/9 = 2{,}197160$ bits par symbole.

Symboles par ordre croissant	`.`	`A`	`B`	`E`	`S`
Nombre d'occurrences dans la source	1	1	2	2	3
Probabilité d'apparition	1/9	1/9	2/9	2/9	3/9

Générateur de nombres pseudo-aléatoires

Étant donné u_0 on définit la récurrence suivante :

$$u(0) = u_0$$

$$\forall t \in \mathbb{N},\ u(t+1) = (28 \times u(t)) \mod 336\,529$$

L'entier u_0 vous est donné, et doit être recopié sur votre fiche réponse avec vos résultats. Une fiche réponse type vous est donnée en exemple, et contient tous les résultats attendus pour une valeur de u_0 différente de la vôtre (notée $\widetilde{u_0}$). Il vous est conseillé de tester vos algorithmes avec cet $\widetilde{u_0}$. Pour chaque calcul demandé, avec le bon choix d'algorithme le calcul ne devrait demander qu'au plus quelques secondes, jamais plus d'une minute.

Question 1 *Calculer les valeurs suivantes :*

a) $u(1) \mod 1\,000$ ***b)*** $u(42) \mod 1\,000$ ***c)*** $u(10^5) \mod 1\,000$.

Génération de source

L'alphabet comporte $M+1$ lettres de 0 à M. Dans ce sujet, $M = 29$. La source S_N est le mot de taille $N+1$ dont le t-ième symbole $S_N[t]$ pour $t = 0, \ldots, N$ vaut :

$$\begin{cases} S_N[t] = u(t) \mod M & t = 0, \ldots, N-1 \\ S_N[N] = M. \end{cases}$$

Ainsi, la source S_N termine toujours par le symbole de fin de séquence M.

Question 2 *Calculer les 5 derniers symboles de S_N pour les valeurs de N suivantes :*

a) $N = 10$ ***b)*** $N = 1\,000$ ***c)*** $N = 100\,000$.

Question 3 *Calculer l'entropie de la source S_N pour les valeurs de N suivantes, arrondie à 5 chiffres après la virgule.*

a) $N = 10$ ***b)*** $N = 1\,000$ ***c)*** $N = 100\,000$.

1 Codage de Huffman

On s'intéresse à une première méthode de compression qui s'appuie sur la construction d'un arbre binaire, dont l'arête vers le fils gauche (resp. droit) est étiquetée par 0 (resp. 1). Les feuilles représentent les symboles de la source, tandis que le chemin de la racine à une feuille est le mot binaire pour coder le symbole correspondant à cette feuille. Le code de la source est la concaténation des codes de ses symboles. Idéalement, on souhaite que les lettres fréquentes de la source soient codées avec des mots binaires courts. Un exemple est donné à la Figure 1.

L'algorithme de construction de l'arbre de Huffman s'appuie sur une structure de données $\mathcal{S}$ qui contient des arbres associés à des scores, et procède comme suit.

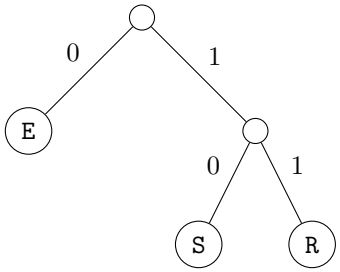


FIGURE 1 – Un exemple d'arbre de Huffman pour la source `SERREE`. `E`, lettre la plus fréquente, est codée par le bit `0` tandis que `R` et `S` sont codées sur deux bits, respectivement `10` et `11`. Ainsi le code de la source `SERREE` est `110101000` sur 9 bits.

Algorithme du codage de Huffman

1. Pour chaque symbole de la source (c'est-à-dire, apparaissant au moins une fois), insérer dans $\mathcal{S}$ une feuille étiquetée par le symbole, ayant pour score son nombre d'occurrences.
2. Tant qu'il y a plus d'un arbre dans $\mathcal{S}$
   (a) Extraire de $\mathcal{S}$ les deux arbres A_1, A_2 ayant les plus faibles scores p_1 et p_2. En cas d'égalité des scores, l'arbre ajouté le plus tôt dans $\mathcal{S}$ sort de $\mathcal{S}$ en premier.
   (b) Créer un nouvel arbre ayant pour sous-arbre gauche A_1 et sous-arbre droit A_2.
   (c) Ajouter ce nouvel arbre à $\mathcal{S}$ avec le score $p_1 + p_2$.
3. L'unique arbre restant dans $\mathcal{S}$ est l'arbre de Huffman.

Question 4 *Construire l'arbre de Huffman pour la source S_N. Si l'on code toute la source, quelle est la taille du code obtenu pour les valeurs de N suivantes ?*

a) $N = 10$ ***b)*** $N = 1\,000$ ***c)*** $N = 100\,000$.

Vous pouvez comparer vos réponses avec l'entropie à la question 3 qui correspond au nombre minimal de bits pour encoder un symbole en moyenne.

Question à développer pendant l'oral 1 *Quelle est votre complexité de construction de l'arbre de Huffman en fonction de M ? Peut-on faire mieux si l'on suppose que les symboles sont déjà triés dans $\mathcal{S}$ par nombre d'occurrences ? Comment décoder un message binaire $\mathcal{C}$ ainsi compressé et avec quelle complexité en fonction de la longueur du code $|\mathcal{C}|$?*

$O(M^2)$ si naïf, $O(M \log M)$ si file de priorité. Si les symboles sont déjà triés, on peut s'aider d'une deuxième file dans laquelle on insère les nouveaux arbres, et on défile les arbres des deux files à disposition, aboutissant à une complexité $O(M)$. Pour décoder, on part de la racine et on suit les arêtes en fonction des symboles du code. Si on atteint une feuille, on la renvoie et on repart de la racine. Ainsi la complexité est linéaire en $|\mathcal{C}|$.

2 Codage arithmétique : compression vers $[0, 1[$

On s'intéresse à une nouvelle méthode de compression appelée codage arithmétique, inspirée par Shannon et aujourd'hui utilisée dans l'encodage vidéo H.264. L'idée consiste à diviser l'intervalle dans lequel on se trouve en zones proportionnellement à la fréquence de chaque symbole. Les zones sont ordonnées par l'ordre naturel sur les symboles. On part de l'intervalle $[0, 1[$ avec $L_0 = 0$ et $D_0 = 1$. Pour chaque symbole de la source, on emprunte la zone correspondant à ce symbole et on met à jour l'intervalle $[L_i, L_i + D_i[$. La source finit par le symbole de fin de séquence M qui permet d'identifier à quel moment interrompre le codage et surtout le décodage. Le code de la source peut enfin être n'importe quel nombre qui appartient à l'intervalle $[L_n, L_n + D_n[$ où n est la longueur de la source. Un exemple est donné à la Figure 2, où l'on a omis le symbole de fin de séquence pour simplifier l'illustration.

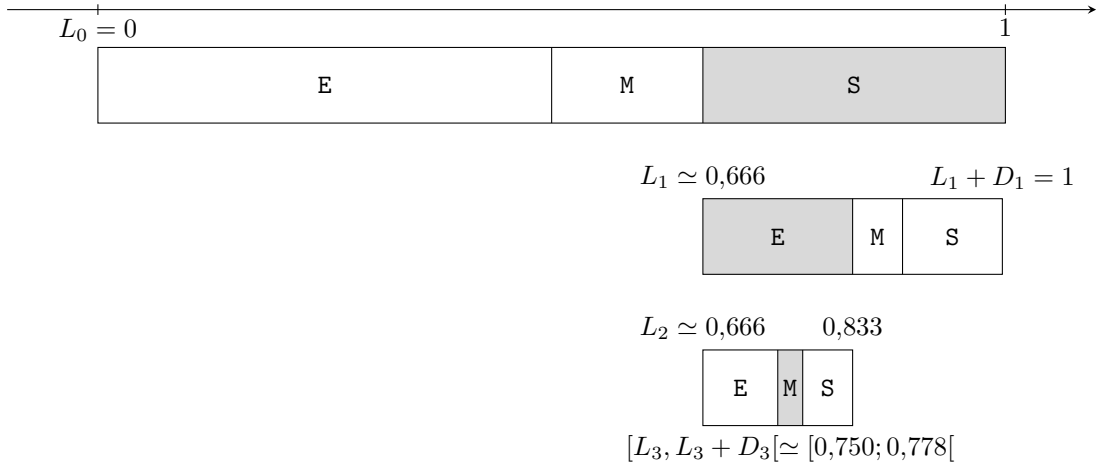


FIGURE 2 – Un exemple de codage arithmétique pour les trois premières lettres de la source `SEMEES`. Les fréquences des symboles `E M S` sont respectivement 3/6, 1/6, 2/6. À cette itération, on sait que le code de la source sera dans l'intervalle $[0{,}750; 0{,}778[$.

Question 5 *Pour cette question on attend pour chaque valeur de N un nombre compris entre 0 et 1 arrondi à N chiffres en base 10 après la virgule. Implémenter le codage arithmétique et donner la valeur de L_{N+1} obtenue en compressant la source S_N pour :*

a) $N = 7$ ***b)*** $N = 10$ ***c)*** $N = 15$.

Question à développer pendant l'oral 2 *Présenter votre algorithme de compression et sa complexité en fonction de N et de la taille de l'alphabet M.*

On peut calculer le nombre d'occurrences c_k et le nombre d'occurrences cumulatif $C_k = \sum_{j<k} c_j$ du symbole k pour se simplifier la vie, en $O(M)$. $L_{i+1} \leftarrow D_i \times C_i/n$, $D_{i+1} \leftarrow D_i \times c_i/n$. Au total la complexité est linéaire $O(N + M)$.

Question 6 *Décoder les cinq premières lettres du code u_0/N sachant qu'on sait que la fréquence de la source correspond à S_N, pour les valeurs suivantes de N. Attention, ces valeurs ne sont pas les mêmes qu'aux autres questions.*

a) $N = 100$ ***b)*** $N = 500$ ***c)*** $N = 1\,000$.

Question à développer pendant l'oral 3 *Présenter votre algorithme de décodage et sa complexité en fonction de la longueur de la source à décoder N et de la taille de l'alphabet M. Que peut-il arriver lorsque la source est longue et que proposez-vous pour y remédier ?*

À chaque niveau de profondeur, on cherche d'abord l'intervalle dans lequel se trouve le code en $O(M)$ ou $O(\log M)$ si dichotomie car le tableau est trié. Si le caractère correspondant est celui de fin de séquence, on arrête de décoder. Sinon on met à jour l'intervalle en cours et on continue. Complexité $O(NM)$ ou $O(N \log M)$ selon si dichotomie. On risque d'avoir des erreurs de précision, donc on peut représenter L_i et D_i par des fractions d'entiers pour conserver une précision arbitrairement exacte (à condition d'avoir une représentation des grands nombres).

On cherche à représenter le mot de code avec le moins de bits possible, sachant qu'on peut tronquer l'écriture d'un mot s'il ne finit que par des zéros.

Question 7 *Écrire la décomposition en base 2 du nombre L_{N+1}. Pour cette question, si l'on reprend la notation $a_1 a_2 \cdots$ pour l'écriture en base 2, alors on attend les 6 bits $a_{15}a_{16}a_{17}a_{18}a_{19}a_{20}$.*

a) $N = 7$ ***b)*** $N = 10$ ***c)*** $N = 15$.

Question à développer pendant l'oral 4 *Présenter votre algorithme et sa complexité en fonction de la taille du code souhaitée $|\mathcal{C}|$, ici $|\mathcal{C}| = 20$ bits.*

Convertir un flottant ou rationnel de la base 10 vers la base $b = 2$. Pour éviter d'avoir des problèmes d'arrondi, il vaut mieux multiplier par la base b et calculer le reste de la division par b plutôt que de soustraire des flottants de plus en plus petits, pour éviter les erreurs de précision.

Question à développer pendant l'oral 5 *Quelle est une borne supérieure convenable en bits pour écrire un nombre compris dans l'intervalle $[L_i, L_i + D_i[$, en fonction des caractéristiques de l'écriture de L_i et D_i ? Donner un algorithme pour identifier efficacement un code arithmétique de taille minimale pour une source donnée S_N, ainsi que sa complexité en fonction de l'écriture de L_i et D_i.*

L_i peut être arbitrairement long mais c'est D_i qui définit la marge de manœuvre qui permet d'arrondir L_i dans l'intervalle. Une borne supérieure est la position r du premier 1 de l'écriture de D_i : dans le pire cas, L_i et $L_i + D_i$ ont les mêmes bits jusqu'à ce chiffre. On peut naïvement parcourir l'ensemble de tous les mots de code possibles, mais il est exponentiel en la taille en bits de L_i et D_i. La position r permet grâce à L_i d'identifier un intervalle $[\ell, r]$ qui contient le candidat solution. Dans le pire cas il faudra quand même parcourir tous les bits de L_i pour identifier le code le plus court, donc complexité $O(r + |L_i|) = O(|L_i| + |D_i|)$.

Question à développer pendant l'oral 6 *Trouver un exemple simple de source pour lequel le codage arithmétique peut être arbitrairement meilleur que le codage de Huffman, c'est-à-dire qu'il nécessite moins de bits pour coder la source.*

Si l'on suppose A et B équiprobables et pas de séparateur pour simplifier, si l'on considère la chaîne BA^n de longueur $n+1$, avec Huffman il faudra toujours $2(n+1)$ bits tandis que le codage arithmétique permetta de la coder avec le nombre $1/2$ qui s'écrit sur deux bits. On peut même dire que 0 code une source infinie de A sur 0 bit.

3 Entropie et devinettes

Dans cette section on s'appuiera sur le jeu de données `data.txt` fourni avec le sujet, qui contient les mots uniques du dictionnaire dans un alphabet à 26 lettres, séparés par des retours de ligne. Si les mots sont numérotés de 0 à $D-1$, on ne considérera que les mots uniques dont les numéros sont dans l'ensemble $\{u(j), 0 \leq j < 50\,000\}$. Par exemple pour $\widetilde{u_0} = 42$ on considère seulement 48 081 mots du dictionnaire.
On cherche à deviner un mot au jeu du Pendu en un nombre minimal de coups. Pour simplifier les règles, au départ l'adversaire tire uniformément un mot du dictionnaire, et on ne connaît pas son nombre de lettres ; à chaque tour, on doit proposer une lettre et l'adversaire nous indique si elle se trouve dans le mot ou pas, sans donner sa position.
On cherche à construire un arbre binaire des états possibles : les nœuds contiennent la lettre qu'on joue, les arêtes indiquent les réponses de l'adversaire : 1, à gauche, si la lettre se trouve dans le mot à chercher ; 0, à droite, si la lettre ne s'y trouve pas. Dans un premier temps, on pose la question qui va le plus équilibrer l'espace des mots possibles, de façon à obtenir un arbre de hauteur moyenne faible, afin que le nombre de coups pour gagner soit petit, peu importe le mot choisi. En cas d'égalité entre deux lettres, on choisit la plus petite de l'alphabet. On ignore les lettres pouvant mener à un ensemble vide de mots possibles. Ainsi, si deux mots ont les mêmes ensembles de lettres, ils sont indistinguables, on considère que ça ne sert à rien de jouer davantage et qu'on a trouvé le mot.

Question 8 *Construire l'arbre binaire avec ce critère à partir du jeu de données fourni en entrée.*

a) *Quelle est la première lettre à jouer, c'est-à-dire l'étiquette de sa racine ?*

b) *Quelle est la longueur de sa plus petite et sa plus longue branche ?*

À présent on ne cherche plus à deviner le mot mais juste sa longueur, comprise entre 1 et 25. Au début de la partie, on a plus de chances de tomber juste si on devine une taille fréquente. La

quantité qu'on cherche à réduire le plus est l'entropie des réponses possibles, soit $H(L_s)$ où L_s est le mot dont le i-ème caractère est la longueur du i-ème mot possible de l'ensemble s (l'ordre n'a pas d'importance). On préfère proposer la lettre ℓ telle que, si on note s_1 les mots contenant ℓ et s_0 les autres, la valeur $p(s_1)H(s_1)+p(s_0)H(s_0)$ est minimale, où $p(s_1)$ désigne la probabilité que le mot à deviner contienne ℓ. Notons que si tous les mots d'un ensemble s ont la même longueur, alors $H(L_s)$ vaut 0 et le nœud correspondant est une feuille.

Question 9 *Construire l'arbre binaire avec ce critère à partir du jeu de données fourni en entrée.*

a) *Quelle est la première lettre à jouer, c'est-à-dire l'étiquette de sa racine ?*

b) *Quelle est la longueur de sa plus petite et sa plus longue branche ?*

Enfin, on cherche à deviner une valeur entre 1 et N choisie par l'adversaire. On propose à chaque tour un nombre k tel que $1 \leq k \leq N$ et l'adversaire nous dit si la valeur à chercher est plus grande ou plus petite que k. Mais le coût n'est plus le nombre de coups joués, mais la somme des valeurs jouées : jouer le nombre k à un moment du jeu a un coût de k, et on cherche à minimiser le coût total de la partie dans le pire cas.

Question 10 *Pour les valeurs suivantes de N, quel est le coût d'une partie optimale dans le pire cas ?*

a) $N = u_0$ ***b)*** $N = 100 + u_0$ ***c)*** $N = 5\,000 + u_0$ ***d)*** $N = 100\,000 + u_0$.

Question à développer pendant l'oral 7 *Vous présenterez votre algorithme et sa complexité en fonction de N.*

Fiche réponse type : Compression vers $[0, 1[$.

$\widetilde{u_0}$: 42

Question 1

a) 176

b) 338

c) 645

Question 2

a) [20, 4, 16, 13, 29]

b) [6, 22, 22, 16, 29]

c) [10, 19, 27, 25, 29]

Question 3

a) 2.84535

b) 4.84049

c) 4.85763

Question 4

a) 32

b) 4896

c) 492471

Question 5

a) 0.3878305

b) 0.445078646

c) 0.492487883047874

Question 6

a) [11, 25, 24, 8, 9]

b) [2, 4, 4, 23, 12]

c) [1, 0, 0, 20, 20]

Question 7

a) [0, 0, 1, 1, 0, 1]

b) [0, 0, 1, 0, 1, 0]

c) [1, 1, 1, 0, 1, 0]

Question 8

a) o

b) 10 24

Question 9

a) n

b) 8 25

Question 10

a) 129

b) 645

c) 44365

d) 1275985