MINISTÈRE
DE L'ÉDUCATION
NATIONALE
ET DE LA JEUNESSE

Liberté
Égalité
Fraternité

Rapport du jury

Concours : agrégation externe

Section : Informatique

Session 2024

Rapport de jury présenté par :
Sylvie Boldo, directrice de recherche, présidente du jury.

Table des matières

Chapitre 1

Déroulement et statistiques

Ce document est le rapport de la troisième édition de l'agrégation externe d'informatique, qui s'est déroulée en 2024. Il a plusieurs objectifs : c'est d'une part une analyse et un bilan de cette édition (statistiques sur les candidates et candidats, réussite aux épreuves, erreurs fréquentes, conseils…) et d'autre part un document à l'attention des futurs candidates, candidats, préparatrices et préparateurs (attentes du jury, écueils à éviter…).

1.1 Déroulement du concours

L'agrégation externe section informatique est régie par l'arrêté MENH2112666A du 17 mai 2021 [1] qui en définit en particulier le programme et les épreuves.

Le **programme des épreuves d'admissibilité** se compose des programmes d'enseignement de la spécialité « numérique et sciences informatiques » (NSI) du cycle terminal de la voie générale du lycée (première et terminale), de ceux des classes préparatoires scientifiques aux grandes écoles « mathématiques, physique, ingénierie et informatique » (MP2I) et « mathématiques, physique, informatique » (MPI), auxquels s'ajoute un programme complémentaire. Ce programme complémentaire a été élaboré par le jury. Il a pour but de lui permettre d'évaluer le recul des candidates et candidats sur les notions enseignées et a été conçu par « adhérence » des programmes sus-cités. Rappelons que, pour l'ensemble du programme, il est attendu des candidates et candidats un recul correspondant au niveau master.

Pour la session 2024, les **épreuves écrites** se sont déroulées du 20 au 22 février, organisées par les académies :

- Composition d'informatique : 20 février 2024 de 9 heures à 14 heures,
- Étude d'un problème informatique : 21 février 2024 de 9 heures à 15 heures,
- Épreuve spécifique (selon l'option choisie : étude de cas informatique ou fondements de l'informatique) : 22 février 2024 de 9 heures à 15 heures.

Les **épreuves orales** se sont déroulées du 17 au 22 juin 2024 au lycée Guy Mollet à Arras, organisées par le jury. Chaque admissible a été convoqué pour les trois épreuves orales sur trois jours consécutifs :

- Leçon d'informatique (4 heures de préparation, 1h d'épreuve),
- Travaux pratiques de programmation (5 heures de préparation, 1h d'épreuve),
- Modélisation (4 heures de préparation, 1h d'épreuve).

Pour plus de précisions, nous renvoyons les lectrices et lecteurs aux descriptifs de ces épreuves se trouvant dans l'arrêté MENH2112666A[1]. Nous signalons aussi le site web du jury `https://agreg-info.org`, régulièrement mis à jour. Il contient en particulier des sujets des années précédentes pour les épreuves écrites et orales.

1. `https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000043648279`

1.2 Statistiques

En résumé et comme les années précédentes, ce concours a été attractif, avec un très grand nombre de candidates et candidats. De plus, le niveau final des agrégées et agrégés a été très bon. Le jury a pourvu sans hésiter tous les postes. Les agrégées et agrégés ont montré de très belles qualités, autant disciplinaires que pédagogiques. Le jury les a volontairement testés sur des domaines différents de l'informatique et des langages différents avec succès et a pu sélectionner des informaticiennes et informaticiens complets, avec une vision large de la discipline.

Voici tout d'abord le nombre de candidatures lors des différentes phases du concours. Pour 2024 et comme en 2023, les épreuves écrites furent communes avec l'agrégation d'informatique du Maroc. Les candidates et candidats des deux concours ont passé les épreuves écrites en même temps. Leurs copies ont ensuite été corrigées par les mêmes correcteurs et correctrices, anonymement et sans connaissance de la nationalité.

Inscrits (dont Maroc)	Présents (dont Maroc)	Inscrits (France)	Présents (France)
483	175	393	142

Admissibles	Admis liste principale	Liste complémentaire
48	22	0

Le chiffre des inscrits est en légère baisse par rapport à 2023. Mais le nombre de personnes présentes est lui en baisse importante par rapport à 2023, sachant que les épreuves écrites se déroulaient cette année pendant des vacances scolaires. Malgré cette baisse, l'agrégation d'informatique reste parmi les sections très sélectives du concours de l'agrégation externe, avec environ 8 candidats présents pour un poste.

Pour ce qui concerne les épreuves écrites, la moyenne des candidats admissibles est de 12,33/20, avec une **barre d'admissibilité à 9,11/20**.

Pour ce qui concerne les épreuves orales uniquement, la moyenne des candidats admis est de 12,98/20. Sur l'ensemble des épreuves, la **barre d'admission est de 10,50/20**. L'ensemble de ces chiffres montre une agrégation restant toujours très sélective.

Les statistiques suivantes ne concernent que les candidates et candidats à l'agrégation française.

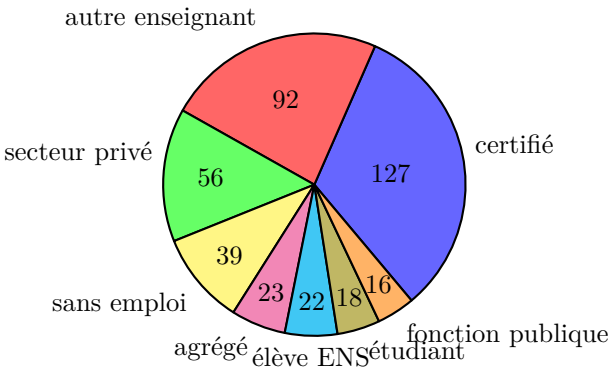
1.2.1 Statistiques par statut

Il est important de rappeler que ce tableau se base sur les statuts indiqués, de manière purement déclarative, par les candidates et candidats lors de l'inscription. Aucune vérification n'est opérée sur ce point, qui fait uniquement l'objet de la présente exploitation statistique.

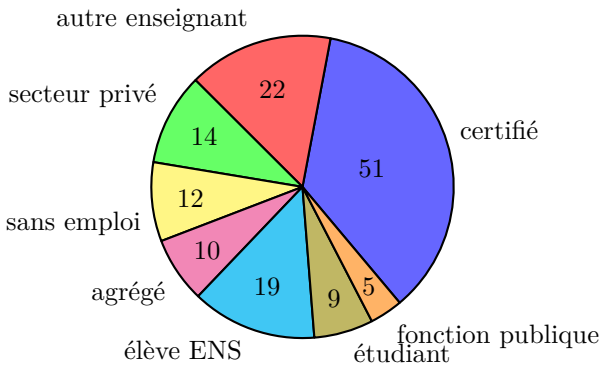
Les profils sont extrêmement divers, même si cette diversité tend à se réduire au cours du concours, pour favoriser progressivement les candidats bénéficiant de l'environnement d'une préparation.

inscrits	présents aux écrits	admissibles	admis	profession
127	51	10	4	Certifié
43	6	1	0	Personnel enseignant de la fonction publique
39	12	6	1	Sans emploi
33	8	3	0	Cadres secteur privé
23	10	2	0	Agrégé
22	19	17	14	Élève d'une ENS
21	6	0	0	Contractuel 2nd degré
17	3	2	0	Salariés secteur tertiaire
15	5	0	0	Personnel de la fonction publique
11	3	0	0	Étudiant hors inspe (sans prépa)
8	2	1	1	Enseignant du supérieur
6	3	0	0	Professions libérales
6	2	1	0	Vacataire
6	2	0	0	Formateurs dans secteur privé
5	3	1	0	Maître contr. et agréé rem tit
4	3	0	0	Étudiant en inspe en 2eme année
3	3	3	2	Étudiant hors inspe (prépa mo.univ)
3	1	1	0	Contractuel enseignant supérieur
1	0	0	0	Militaire

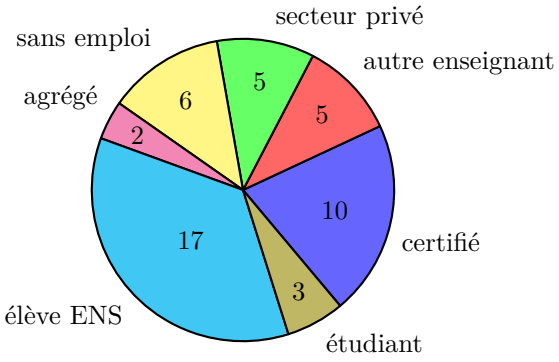
TABLE 1.1 – Répartition des candidats par profession.



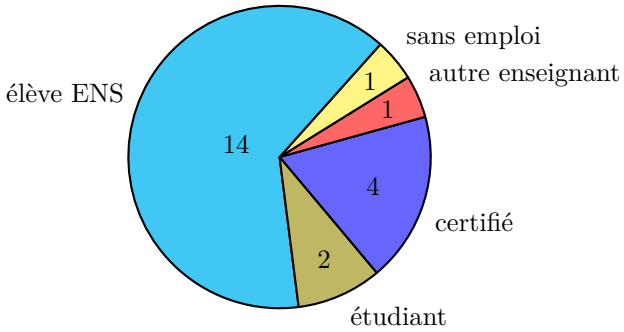
Inscrits par statut.



Présents par statut.



Admissibles par statut.

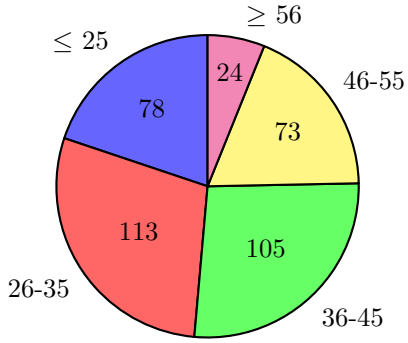


Admis par statut.

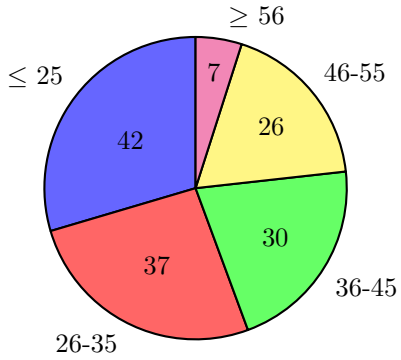
1.2.2 Statistiques par âge

Âge	inscrits	présents aux écrits	admissibles	admis
≤ 25	78	42	26	17
26-35	113	37	16	4
36-45	105	30	4	0
46-55	73	26	2	1
≥ 56	24	7	0	0

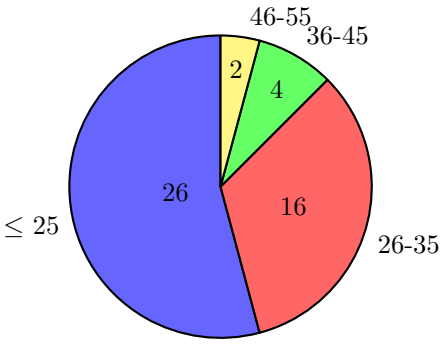
TABLE 1.2 – Répartition des candidats par âge.



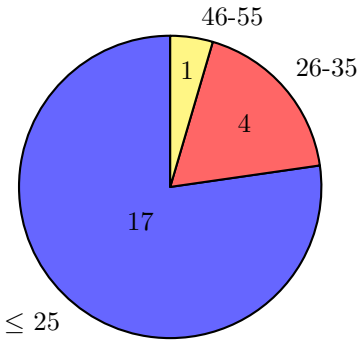
Inscrits par âge.



Présents par âge.



Admissibles par âge.



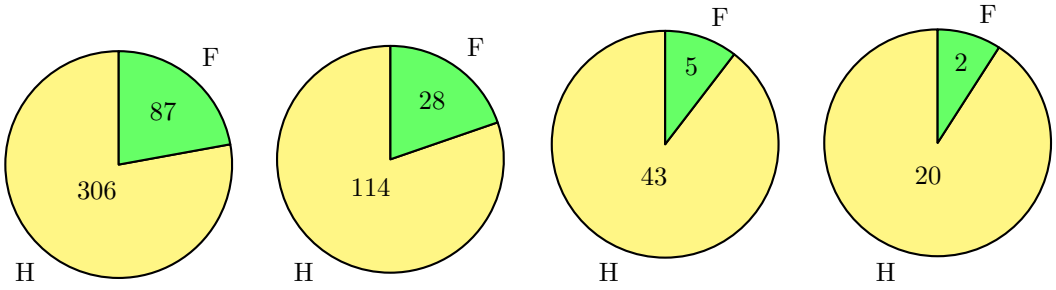
Admis par âge.

1.2.3 Statistiques par genre

Rappelons que le genre, également déclaratif, ne permet à l'inscription que les deux choix M. ou Mme.

inscrits	présents aux écrits	admissibles	admis	genre
306	114	43	20	M
87	28	5	2	F

TABLE 1.3 – Répartition des candidats par genre.



Inscrits par genre. **Présents par genre.** **Admissibles par genre.** **Admis par genre.**

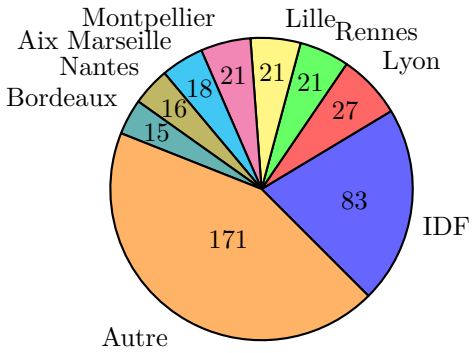
La représentation des femmes en sciences et en informatique en particulier reste très faible. Le taux de femmes inscrites est assez faible (environ 22%), les femmes se présentent en moindre proportion, et elles ont plutôt moins bien réussi les épreuves écrites (sachant que les copies sont anonymes). Pour les épreuves orales, il est délicat de faire des analyses sur un échantillon aussi réduit, probablement corrélé avec d'autres facteurs statistiques mais ce point reste une préoccupation du jury.

1.2.4 Statistiques par académie

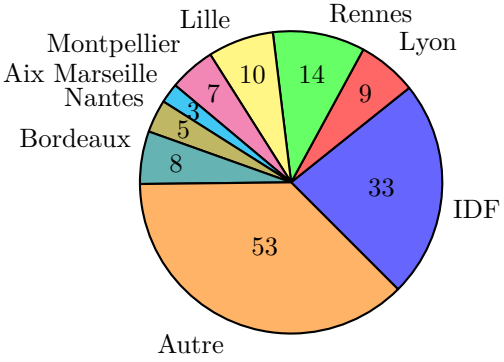
inscrits	présents aux écrits	admissibles	admis	académie
83	33	10	4	Académies de Créteil Paris Versailles
43	1	0	0	Hors académie
27	9	7	4	Académie de Lyon
21	14	12	8	Académie de Rennes
21	10	1	0	Académie de Lille
21	7	1	0	Académie de Montpellier
18	3	1	0	Académie d'Aix Marseille
16	5	2	0	Académie de Nantes
15	8	2	1	Académie de Bordeaux
13	6	2	0	Académie de Nice
13	6	0	0	Académie de Normandie
13	5	1	0	Académie de la Réunion
11	5	3	1	Académie de Nancy-Metz
11	2	0	0	Académie de Grenoble
10	5	1	0	Académie de Strasbourg
10	1	0	0	Académie de Poitiers
7	4	2	1	Académie d'Orléans-Tours
7	3	0	0	Académie d'Amiens
7	2	1	1	Académie de Toulouse
6	4	1	1	Académie de Reims
6	4	0	0	Académie de la Nouvelle Calédonie
2	2	0	0	Académie de la Martinique
2	1	1	1	Académie de Limoges
2	1	0	0	Académie de Corse
2	1	0	0	Académie de la Guyane
2	0	0	0	Académie de Besançon
2	0	0	0	Académie de Mayotte
1	0	0	0	Académie de la Guadeloupe
1	0	0	0	Académie de Dijon

TABLE 1.4 – Répartition des candidats par académie.

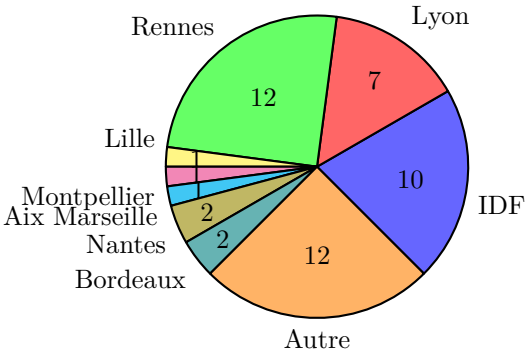
Rappelons que les seules académies (à ce jour) à accueillir une préparation à l'agrégation externe d'informatique sont les académies de Créteil-Paris-Versailles, Lyon et Rennes, même si la première préparation peut se suivre partiellement à distance.



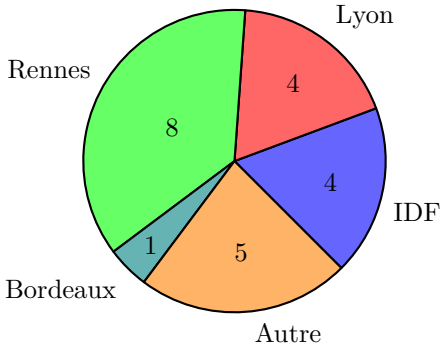
Inscrits par académie.



Présents par académie.



Admissibles par académie.



Admis par académie.

1.3 Remerciements

Le jury du concours 2024 souhaite remercier les personnels de la DGRH pour tout leur appui logistique. En particulier, un sincère merci à la gestionnaire de notre concours pour sa disponibilité sans faille et son aide précieuse.

Les épreuves orales de 2024 se sont tenues pour la première fois à Arras. Le jury tient à remercier très sincèrement les personnels techniques, administratifs et de direction du lycée Guy Mollet. Votre efficacité, disponibilité et gentillesse nous ont permis de tenir cette session dans d'excellentes conditions.

Merci également aux personnels de la DEC de Lille pour la gestion des malles, ainsi que pour avoir trouvé et convoqué les appariteurs et apparitrices.

La présidence du jury remercie également très chaleureusement les membres du jury pour leur travail dans la création de sujets pour ces très nombreuses épreuves, la qualité de la correction et de l'interrogation, leur ouverture d'esprit et leur implication.

Chapitre 2

Épreuves écrites

Pour alléger le texte, ce chapitre est uniquement écrit au masculin [1], mais cela doit être pris dans le sens du neutre : l'ensemble de ce texte, sans exception, concerne uniformément candidates et candidats.

L'architecture générale de l'écrit de l'agrégation externe d'informatique est définie par l'arrêté du 17 mai 2021. Dans les limites de la maquette définie par cet arrêté, le jury souhaite affirmer sa volonté d'utiliser l'ensemble des épreuves à sa disposition pour recruter des *informaticiens complets*, capables à la fois de réflexes sains dans un ensemble de domaines assez larges de l'informatique (épreuve 1), et dotées d'une solide maîtrise de la programmation et de l'algorithmique (épreuve 2). L'épreuve 3 est l'occasion de démontrer des connaissances approfondies dans un domaine au choix des candidates, soit plutôt l'ingénierie logicielle, soit plutôt les fondements de l'informatique.

Certaines remarques s'appliquent à toutes les épreuves. En particulier, on attend d'un candidat aux fonctions de professeur qu'il sache rédiger une copie. En conséquence, un passage rayé ou hachuré ne sera jamais lu. De plus, quand plusieurs versions sont proposées par le candidat, le correcteur ne prend pas la meilleure. C'est au candidat de choisir ce qu'il présente à l'évaluation du jury, pour le meilleur et pour le pire.

Il est déconseillé d'écrire hors du cadre de la copie. Le scan des copies est souvent bon mais ce n'est pas garanti que ce qui est écrit hors du cadre arrive au correcteur.

Une préoccupation du jury est de valoriser les candidats précis sur les parties élémentaires du sujet plus que ceux qui bâcleraient le début pour avancer plus vite vers les parties plus ambitieuses. Le jury recherche des candidats solides et précis sur les bases de la discipline, afin de pouvoir transmettre ces dernières.

Il est attendu, pour l'écrit comme pour l'oral, que les candidats maîtrisent le vocabulaire des mathématiques nécessaire à l'étude des éléments au programme de l'agrégation d'informatique, dont par exemple les algorithmes probabilistes et l'analyse de complexité.

Le jury tient compte dans la notation des épreuves de la maîtrise écrite et orale de la langue française (vocabulaire, grammaire, conjugaison, ponctuation, orthographe) [2]. Le jury attend donc une certaine tenue dans l'écriture, la présentation et l'orthographe. Lesdites attentes sont modérées, mais réelles. Le jury fait la part des choses entre ce qui est explicable par le contexte "concours" et ce qui risque d'handicaper un future agrégé dans sa pratique professionnelle et la transmission des savoirs. Les travers relevant de la seconde catégorie sont pris en compte, aux côtés de la maîtrise disciplinaire, dans l'évaluation d'une copie.

Le jury a apprécié de lire des réponses construites, c'est-à-dire des phrases. Rappelons qu'une phrase doit toujours commencer par une majuscule et se terminer par un point, même quand elle est très courte et constitue à elle seule la réponse à une question.

1. Le chapitre 3 est écrit au féminin.
2. `https://www.legifrance.gouv.fr/loda/id/LEGIARTI000046287651/2022-09-01/#LEGIARTI000046287651`

2.1 Épreuve 1

Cette épreuve était composée de trois parties, balayant largement plusieurs domaines de l'informatique : gestion de ressources, structure de donnée, base de données et plusieurs langages : C, OCaml, SQL et Python.

Le jury a été déçu par les réponses à cette épreuve, car de nombreux candidats ont peu abordé certaines parties, par exemple la programmation en OCaml.

La lisibilité des copies est un problème qui semble empirer. On rappelle aux candidats que leurs copies sont faites pour être lues par d'autres personnes. Si une question est incompréhensible car le correcteur ne peut deviner quels sont les mots écrits, aucun point ne sera accordé.

On demande aux candidats d'indiquer très clairement la question à laquelle ils répondent, par exemple 2.5 et pas seulement 5, puisque l'épreuve est composée de plusieurs parties. De plus, réécrire sur le numéro de question (lorsqu'on décide de ne pas traiter une question par exemple) rend ce numéro illisible par le correcteur qui a alors du mal à savoir quelle question est répondue. Lorsque le candidat ne répond pas aux questions dans l'ordre, le numéro doit être parfaitement lisible, voire mis en valeur.

Il est déconseillé de critiquer les questions posées. Il est possible de ne pas les comprendre ou d'avoir plusieurs interprétations, mais la critique explicite (et fausse) est peu appréciée par le correcteur, qui peut être le concepteur du sujet.

Données statistiques Pour chaque épreuve, et en plus des données de base, un tableau statistique est fourni. Il contient à la fois les quartiles et la proportion des candidats ayant une note supérieure à 5, 10 et 15.

— 141 présents
— Meilleure note : 16,30/20
— Moyenne : 6,95 ; écart-type : 3,79.

≥ 4.4	75%
≥ 5	63,8%
≥ 6.32	50%
≥ 9.68	25%
≥ 10	24,1%
≥ 15	2,1%

2.1.1 Partie I : robot de dépôt de solutions chimiques

Cette partie de l'épreuve se focalise sur la gestion concurrente de ressources. Il s'agissait principalement d'explorer les notions d'exclusion mutuelle, et plus généralement de synchronisation entre threads.

Cette partie a été assez souvent abordée. La notion de base d'exclusion mutuelle a souvent bien été abordée. Les notions plus évoluées telles que la gestion de plusieurs ressources, les processus légers (threads), ou les synchronisations un peu avancées (gestion de n ressources), ont été traitées de manière correcte par une minorité de copies.

Enfin, lorsqu'une question attend une réponse close (de type oui ou non), une certaine précision est attendue. En effet une longue explication abordant les deux côtés sans réponse définitive ne correspond pas à la question. En plus des attendus pédagogiques, le jury attend, sur les questions closes, une réponse explicite et claire.

Question 1.1 La question a été bien comprise dans l'ensemble. On regrettera seulement la rareté des explications basées sur l'utilisation d'un état (par exemple variable) partagé. Certaines copies relèvent le problème d'utiliser le même i (alors que cette variable n'est pas partagée) mais semblent ne pas être gênées par l'utilisation en simultané de la fonction aller() par deux threads en même temps. Vu qu'il s'agit du même bras cela pose un problème.

Question 1.2 On rappelle que la réponse aux questions doit être claire. Ici le code proposé ne marche pas. Une réponse de type "la solution n'est pas optimale", ou "cela ne semble pas fonctionner", ne répond pas à la question.

Question 1.3 De même, ici le code proposé ne marche pas. Une réponse de type "ce code n'est pas satisfaisant", "on perd le bénéfice des threads" ne répond pas à la question.

Question 1.4 L'utilisation de la fonction `encoder` indique clairement qu'elle ne produit pas d'effets de bords. Il est donc possible de la déplacer, et plus particulièrement avant la fonction critique. Par contre les propositions visant à séparer la section critique en deux, une pour `libre`, une pour `aller/injecter`, ne garantissent pas le bon fonctionnement du bras.

Question 1.5 Pour prouver qu'une solution ne fonctionne pas comme ici, il suffit d'exhiber un contre-exemple.

Question 1.6 On s'attend à ce que la notion de système d'exploitation (plus particulièrement la gestion des processus et processus légers) et ses grandeurs caractéristiques (temps de changement entre processus) soient connues au moins d'un point de vue ordre de grandeur.

Question 1.7 Cette question a reçu un grand nombre de réponses vagues de type "il peut fonctionner". On s'attend à avoir une réponse précise : oui, non.

De plus, plusieurs copies ont utilisé le terme "boucle infinie". Une boucle infinie est une boucle qui par construction ne termine jamais. Ici ce terme n'est pas adapté car les boucles terminent toujours, du fait du comportement des autres processus légers.

Question 1.8 La question demandait explicitement quel était l'impact du programme décrit sur les autres processus et processus légers. On s'attendait à obtenir des éléments sur l'impact sur les performances, plus particulièrement sur les autres processus ne participant pas à la gestion du bras.

Question 1.9 Plusieurs propositions de solutions étaient proche d'une solution fonctionnelle basée sur un sémaphore à N jetons, par contre en oubliant de relâcher la sémaphore dans la fonction `récupérer`. De plus, il était tout de même nécessaire d'utiliser une section critique pour protéger les accès aux variables partagées.

Question 1.10 La principale différence dans le cadre de la question entre processus et processus légers tient au partage de données. Ici les données sont autant les données directes (par exemple l'état des tubes), que les données indirectes (les structures de données associées aux sémaphores par exemple).

Question 1.11 Lorsqu'il est demandé de modifier un code, il est attendu un bon degré de précision. Par exemple un "au début" n'est pas particulièrement clair, la définition des variables faisant partie ou pas du code. Les énoncés sont fournis avec des numéros de lignes de façon à pouvoir écrire : ajouter la ligne suivante entre les lignes 7 et 8 par exemple. De même le à la "fin de la fonction" lorsqu'un return est présent est ambigu.

Enfin, plusieurs copies utilisent de manière étonnante deux mutex bloqués de manière simultanée entre les lignes 2 et 3, et débloqués entre les lignes 6 et 7.

Question 1.12 Plusieurs copies proposent d'utiliser une fonction allerXY. Il était attendu d'utiliser un thread (ou deux) pour paralléliser l'utilisation de X et Y de manière explicite.

Question 1.13 Cette question a été relativement bien traitée dans la majorité des copies abordant la fin de ce sujet.

Question 1.14 Plusieurs propositions se sont concentrées sur la structure de données permettant de gérer les positions des demandes sans aborder la synchronisation ni les exclusions mutuelles. Les copies proposant aussi des moyens de gestion de la synchronisation sont très rares.

2.1.2 Partie II : tableaux flexibles

Dans ce problème, on étudie une structure de données de tableaux flexibles, qui peuvent être agrandis ou raccourcis aux deux extrémités, tout en permettant un accès efficace au n-ième élément. La clé de cette structure de données est un arbre binaire, équilibré par construction, où les éléments sont répartis dans les sous-arbres selon l'indice qui leur correspond.

Ce problème n'utilise que des notions élémentaires et au programme sur les arbres binaires, de même qu'il n'utilise que des constructions simples du langage OCaml.

Il est à noter qu'un nombre significatif de copies (84 en tout) n'obtiennent aucun point sur cette partie de l'épreuve 1, l'immense majorité d'entre elles n'ayant même pas abordé le problème.

Commentaires généraux Certaines copies considèrent à tort que l'on peut accéder aux composantes d'un constructeur OCaml comme s'il s'agissait d'un tableau (par exemple `t.(1)` pour accéder à la seconde composante `b` d'une valeur de la forme `N(a,b,c)`). Certaines copies ajoutent à cela une syntaxe farfelue `t[1]`.

Le paradigme de la programmation fonctionnelle n'empêche pas de faire des tests de type `if ... then ... else ...` qui sont dans beaucoup de situations bien plus naturels que des `match ... with ...`.

Un parenthésage correct est indispensable pour les appels de fonction en OCaml où les arguments ne sont pas, comme dans un grand nombre de langages, écrits entre parenthèses et séparés par une virgule. Ainsi, écrire `get i / 2 l` sera lu `(get i) / 2 l` et non `get (i / 2) l` du fait des priorités dans les expressions.

Question 2.1 Il s'agit là d'une question visant à aider à la compréhension de la structure de tableau flexible. C'est une question plutôt bien traitée dans l'ensemble, mais beaucoup de copies placent tout de même incorrectement dans l'arbre les trois derniers éléments du tableau.

Question 2.2 Dans les réponses à cette question, et plus généralement dans les réponses aux questions impliquant une descente récursive dans l'arbre en maintenant un indice ou une taille, on observe de très nombreuses erreurs arithmétiques de type « erreur de décalage unitaire ».

Question 2.3 Plusieurs copies parviennent à faire la preuve que la hauteur de l'arbre est logarithmique *sans jamais* utiliser la définition d'un tableau flexible. Il devrait pourtant être connu que des arbres binaires quelconques n'ont pas nécessairement une hauteur logarithmique.

D'autres copies utilisent dans leur preuve (parfois même implicitement) l'idée que l'arbre est presque complet, alors que c'est justement là ce qu'on est en train de chercher à démontrer.

On rappelle que cela n'a pas de sens d'écrire que « 0 est bien en $O(\log n)$ ».

Question 2.4 Une question facile, conséquence directe de la question précédente. Les correcteurs apprécient une référence explicite à la question précédente.

Question 2.5 Pour la correction, de nombreux candidats justifient uniquement que la valeur retirée l'a été au bon endroit, mais négligent de montrer que les éléments présents dans l'arbre résultat sont les bons. Certaines copies se contentent même de montrer que le tableau flexible renvoyé a un élément de moins.

Plus généralement, si l'on veut prouver que la fonction est correcte, il faut précisément formaliser ce que l'on entend par cela. Dans de trop nombreuses copies, le candidat indique qu'il prouve la correction sans préciser ce qu'il veut effectivement montrer. C'est comme faire, en mathématiques, une démonstration par récurrence sans préciser l'hypothèse de récurrence.

Question 2.6 Cette question est plutôt traitée. À noter que l'énoncé contenant une typo, demandant une complexité $O(n)$ au lieu de $O(\log n)$. Une seule copie l'a noté.

Question 2.7 Cette question est bien traitée. Cependant, beaucoup de copies oublient de justifier la complexité, ce qui était explicitement demandé.

Question 2.8 Une question bien traitée, où l'échange des sous-arbres gauche et droit a en général été bien identifié.

Question 2.9 Une question peu traitée, ce que l'on peut regretter car l'indication fournie en faisait une question plutôt simple.

Question 2.10 Cette dernière question est plus difficile que les précédentes, et notamment moins guidée que la précédente. La clé de cette dernière question réside dans le partage des sous-arbres, ce que très peu de copies parviennent à saisir.

Certains candidats ont donné une solution avec deux appels récursifs en n/2. Malheureusement, une mauvaise analyse de complexité leur fait croire, à tort, que la fonction est donc de complexité logarithmique, alors que la relation de récurrence

$$c_n = 2c_{n/2} + O(1)$$

a pour solution $c_n = O(n)$.

D'autres copies ont bien compris l'idée de faire un unique appel récursif utilisé deux fois, mais utilisent `cons` ou `liat` pour compléter l'un des sous-arbres, obtenant alors une mauvaise complexité.

Enfin, certaines copies ont l'idée d'utiliser de la programmation dynamique, ce qui est pertinent, mais l'utilisation d'un tableau ne permet pas d'obtenir la bonne complexité.

2.1.3 Partie III : gestion d'un concours de recrutement

Cette partie avait pour but la gestion d'un concours de recrutement, avec une partie SQL et une partie en Python. Elle a été beaucoup traitée et souvent raisonnablement. Le SQL a souvent été bien traité, même si certaines questions, plus difficiles, ont été moins bien réussies. La partie Python, située à la toute fin de l'épreuve 1, a été moins abordée. On peut noter que certaines copies écrivent des fonctions Python vraiment très longues et complexes alors que les solutions faisaient toutes moins d'une douzaine de lignes (sauf 3.21, un peu plus longue).

La lisibilité était un souci dans cette partie. En particulier, les correcteurs ont parfois eu du mal à distinguer `aid` et `cid`.

Il est demandé aux candidats de répondre à la question posée et pas à sa simplification : seulement une partie de la question est traitée en oubliant la difficulté. Par exemple en 3.4, beaucoup de candidats oublient "et pas au CAPES" et en 3.16, beaucoup oublient "sur 3 jours différents".

Même si l'indentation n'est pas significative en SQL, le correcteur préfère un code indenté, qui est beaucoup plus lisible et rapide à corriger. Certains candidats écrivent le SQL comme un paragraphe de texte, ce qui est particulièrement difficile à corriger.

Certains candidats ont une vision assez approximative de la syntaxe SQL. En particulier, AND ne permet pas de juxtaposer des colonnes.

La manipulation des opérateurs d'agrégation est toujours approximative en particulier dans le cadre de filtre sur des valeurs obtenues par une agrégation. Il est important dans ce cadre de savoir quand utiliser l'opérateur HAVING. De plus, les correcteurs ont été assez surpris de voir des copies faisant des SUM, des MAX ou des AVG sur les cid.

Certains candidats utilisent des constructions plus complexes comme LEFT JOIN ou NATURAL JOIN. Ils en ont le droit, mais ils doivent alors en connaître parfaitement la sémantique. Les copies de cette année ont montré que c'était rarement le cas. Des LEFT JOIN ont été utilisés pour faire

des EXCEPT ou des NOT IN mais sans tester à NULL. Des NATURAL JOIN en chaîne, tel que entre Académie, AgregxtInfo et Candidat, ne fonctionneront pas comme espéré car Académie et Candidat ont une colonne "nom" : les NATURAL JOIN ne donneront donc que les candidats ayant un nom d'académie.

Le SELECT permet de choisir les colonnes renvoyées par une requête SQL. Il est évidemment pertinent de lire la question pour savoir quoi afficher. Mais il est aussi important de bien faire cette sélection avant un EXCEPT, au risque d'avoir une requête qui ne renvoie jamais rien.

De façon générale, il y a encore une trop grande confusion entre le fait qu'une valeur v n'est jamais associée à une autre valeur u et le fait que la valeur v est associée avec une valeur différente de u.

Comme en 2022, l'algèbre relationnelle a été peu traitée et a donc parfois permis de différencier des candidats.

Lorsque du code Python est demandé, il faut bien indiquer le nom de la fonction demandée, d'une part pour faire du code correct, mais aussi pour que le correcteur sache bien quelle question est traitée. De plus, il est conseillé de commencer une fonction à gauche de la page et pas trop bas. Certains candidats commencent au milieu de la page (après le numéro de la question et beaucoup d'espace) et se trouvent gênés en fin de ligne ou ont une fonction qui s'étale sur deux pages, ce qui rend l'appréciation de l'indentation complexe.

Il est apprécié des correcteurs lorsque le candidat utilise des traits verticaux pour marquer l'indentation (au changement de page ou pas). Certains candidats importent des bibliothèques externes (telle que panda ou de la gestion d'expression régulières), ce qui a été sanctionné. D'une part, le correcteur ne peut connaître toutes les bibliothèques Python existantes et d'autre part, cela n'est pas équitable entre candidats sur les algorithmes à mettre en oeuvre.

Dans plusieurs questions (3.16 et 3.18 notamment), les candidats ont voulu trier le tableau, ce qui peut donner un algorithme plus simple. Mais il faut alors connaître la syntaxe avec un lambda. La plupart des tels tris vus dans les copies sont syntaxiquement faux.

Même si cet extrait de code est correct

```
if Bool == True:
  return True
else:
  return False
```

il est bien peu pédagogique et les correcteurs ne souhaitent plus le voir.

Le Python trouvé dans les copies est parfois bien peu idiomatique, notamment sur les itérateurs. Les correcteurs ont souvent lu :

```
for i in range (len(X)):
    l = X[i]
```

avant un usage seulement de la variable `l`.

Il est commun en Python d'avoir un nom proche pour l'itéré, comme par exemple : for horaire in horaires Sur papier, cela peut poser quelques soucis au correcteur, notamment lorsque la copie est mal écrite et que le rapporteur ne sait pas si un mot finit ou pas par un s. Une copie a même écrit horaire[horaire].

Il est conseillé aux candidats de bien lire le sujet. En particulier, la phrase "Chacune [fonction de vérification] devra renvoyer un booléen indiquant si le fichier est correct ou non. Si non, elle devra écrire un message d'erreur explicatif." semble avoir été peu lue.

Pour la partie en Python qui suit une partie en SQL, les correcteurs ont été surpris de vois que des copies ont utilisé le nom et prénom du candidat pour en faire des clés de dictionnaire, alors qu'un identifiant unique cid était fourni. Le risque d'homonyme est certes faible, mais c'est une mauvaise pratique.

Question 3.1 On attendait ici surtout l'unicité de cid. Les correcteurs attendaient quelques lignes ("brièvement"), les copies allant de quelques mots à une trentaine de lignes.

Question 3.2 De trop nombreuses copies oublient le GROUP BY.

Question 3.3 Plusieurs copies ne donnent pas le nom des deux académies. Une difficulté ici était le besoin du renommage de tables, à la fois en SQL et en algèbre relationnelle.

Question 3.4 Trop de copies oublient d'enlever les candidats au CAPES. De nombreuses possibilités ont été proposées par les candidats, basées sur NOT IN ou EXCEPT et ont été acceptées.

Question 3.5 Il se trouve que SQL refuse une faire une division entre les valeurs de deux requêtes. Le correcteur attendait donc une requête commençant par SELECT.

Question 3.6 En SQL, trop de copies font une jointure simple avec un aménagement différent de 'Tiers-Temps', alors qu'il fallait bien une soustraction. En algèbre relationnelle, comme en SQL, il faut faire attention à bien faire une projection avant de faire une soustraction.

Question 3.7 Cette question est plus difficile qu'il n'y paraît. De nombreuses solutions correctes ont été proposées, avec des NOT IN ou sélectionnant avec un COUNT nul (sans oublier le GROUP BY dans ce cas).

Question 3.8 Cette requête un peu longue a été souvent mal traitée. En particulier, on ne peut pas faire de MAX (COUNT X). Il était faux de trier les académies par nombre de salles isolées et de n'afficher que la première ligne puisqu'on voulait "les noms des académies [...]".

Question 3.9 Cette question facile a été souvent bien traitée, même si quelques copies ont oublié la limite à 42.

Question 3.10 On rappelle que la moyenne en SQL se dit AVG.

Question 3.11 Les réponses à cette question ont souvent été décevantes. Beaucoup de copies ne permettent pas de passer plusieurs concours. Certaines autorisent le même candidat à passer le même concours dans plusieurs académies. D'autres imposent que tous les candidats passent un concours dans une seule académie.

Question 3.12 Cette question a été peu traitée, mais souvent correctement quand la question précédente était raisonnable.

Question 3.13 Un grand nombre de copies considèrent que la taille de `t_oral` doit être N. Mais sachant que chaque candidat a 3 épreuves, la taille de `t_oral` doit être de 3 * N. Certaines copies vérifient que toutes les lignes de `t_oral` sont de la même longueur alors qu'on voulait vérifier que la longueur est exactement 10.

Question 3.14 Cette question assez facile a été bien traitée. On peut noter quelques erreurs plus fréquentes qu'attendu. Certains vérifient seulement la première ligne. D'autres ont une variable booléenne qui teste la correction d'une ligne, mais elle est réécrite à chaque ligne et renvoyée à la fin, donc le résultat ne reflète que la dernière ligne du tableau.

Question 3.15 Cette question a été mal comprise et mal traitée. Il fallait calculer "pour chaque horaire de chaque jour" le nombre de jurys nécessaires, donc le nombre de candidats qui commencent une interrogation à cette heure. Beaucoup de copies vérifient seulement selon l'heure de début sans tenir compte du jour de convocation. Les correcteurs ont accepté que le calcul se fasse selon l'heure de convocation (ou de préparation) plutôt que d'interrogation, on pouvait supposer que cette vérification vient après celle de 3.14.

Question 3.16 Cette question était difficile et a été peu traitée de façon correcte. Les solutions étaient souvent bien trop longues et complexes alors que l'utilisation des set de Python permettait de simplifier le code. Les copies ne testent souvent que le fait d'avoir 3 épreuves. Pour chaque candidat, il fallait déterminer ses épreuves et les jours de celles-ci. Une solution possible est de trier suivant le cid mais beaucoup de candidats ont utilisé un dictionnaire de façon plus ou moins judicieuse. Une erreur très commune était de ne pas vérifier s'il y a deux fois la même épreuve pour un candidat. C'est le cas de ceux qui avaient une variable par date d'épreuve qu'ils écrasent même si cette valeur est déjà remplie.

Question 3.17 Quand traitée, cette question a été bien traitée. On peut noter quelques copies qui oublient le prénom.

Question 3.18 Mêmes remarques pour la question 16. S'il était judicieux de remarquer que du code est commun entre 3.16 et 3.18, il fallait alors indiquer précisément les changements.

Question 3.21 Cette dernière question était un peu plus longue, même si peu difficile. Pour la transformation des dates et des heures, il était souhaitable de faire des fonctions auxiliaires pour la lisibilité. De plus, on rappelle que `split` est au programme.

2.2 Épreuve 2 : les modèles de Markov cachés

— 141 présents
— Meilleure note : 20/20
— Moyenne : 8,86 ; écart-type : 4,45.

≥ 5	82,9%
≥ 5.75	75%
≥ 8.08	50%
≥ 10	34,7%
≥ 11.92	25%
≥ 15	12,7%

Remarques générales

L'objet de cette épreuve est d'étudier un modèle probabiliste classiquement utilisé en analyse de séquences, les modèles de Markov cachés. Elle est composée de quatre parties : la partie I permet de prendre en main les définitions et propriétés de ces modèles, la partie II étudie leur représentation et implémentation en langage Python, la partie III s'intéresse aux algorithmes classiques qui sous-tendent le fonctionnement des modèles de Markov cachés, s'appuyant sur le principe de programmation dynamique, et enfin la partie IV propose des adaptations de ces algorithmes pour rendre les modèles plus réalistes.

Cette épreuve mobilise les connaissances des candidats en matière de programmation, algorithmique et complexité pour comprendre le fonctionnement d'un modèle probabiliste s'appuyant sur un graphe orienté. Les notions non explicitement au programme sont rappelées et font l'objet de questions d'application directe, ou de déroulés d'algorithme permettant de prendre en main ces notions.

La partie I sollicite en priorité les capacités des candidats à étudier un modèle inédit et à inférer et prouver des propriétés à partir des définitions données. Les connaissances sur les graphes, et notamment les graphes orientés, sont requises et évaluées au travers des preuves de certaines de ces propriétés. Des connaissances élémentaires de probabilités sont attendues, les propriétés plus complexes étant rappelées dans l'énoncé.

La partie II fait appel aux capacités des candidats à mobiliser la programmation objet en Python et d'en maîtriser la mise en oeuvre, notamment en ce qui concerne les constructeurs. Le parti pris de ne pas utiliser de librairies complémentaires de type numpy impose également une réflexion sur la mise en oeuvre notamment de la comparaison de flottants ou les concepts d'égalité et d'identité sur des structures de données imbriquées. La partie sur l'adaptation à une famille de séquences via un alignement multiple met en lumière les capacités des candidats à organiser un code sur la base du cahier des charges, et la partie sur la représentation du modèle implique une réflexion autour des représentations de graphes sous forme de matrice d'adjacence ou de listes d'adjacence.

La partie III s'intéresse à deux types d'algorithmes de programmation dynamique liés aux modèles de Markov cachés, l'algorithme de décodage (Viterbi) qui permet d'inférer une marche de probabilité maximale dans le modèle étant donnée une séquence, et les algorithmes d'évaluation (Forward et Backward) qui calculent la probabilité d'une séquence de longueur donnée dans le modèle. La plupart des questions guident les candidats vers la compréhension des algorithmes, par le biais de déroulement sur un exemple simple, preuve de correction ou analyse de complexité. Il était attendu des candidats qu'ils maîtrisent les concepts du programme en lien avec l'algorithmique, et les techniques usuelles de preuves.

La partie IV s'articule en deux sous-parties, l'une porte sur l'adaptation des conditions initiales d'un modèle de Markov caché construit à partir d'un alignement. Elle permet d'évaluer les capacités des candidats à analyser les modifications à apporter à un code et à mettre en oeuvre cette adaptation tout en conservant les propriétés des modèles. La seconde sous-partie est également une adaptation des modèles, portant cette fois sur le graphe sous-jacent. Il s'agit de détecter, puis de traiter, les colonnes mal alignées dans l'alignement initial. La partie détection est découpée en algorithmique et implémentation en Python, tandis que la partie traitement fait l'objet d'une dernière question assez longue d'implémentation moins guidée.

Remarques générales La première partie a été plutôt bien réussie et abordée par une majorité de candidats, à l'exception de la question 16 qui demandait une preuve par récurrence. La deuxième partie a également été abordée en majorité, mais les questions portant sur la représentation n'ont pas toujours été traitées, et la maîtrise attendues du langage Python pas toujours au rendez-vous. De nombreuses copies se contentent d'effleurer les questions simples sur ces deux parties sans rentrer dans la réflexion et le formalisme. Dans la troisième partie, abordée par une majorité de copies, les questions sur la complexité n'ont pas toujours été traitées, ce qui est fort dommage car elles ne présentaient pas de difficultés particulières. Une minorité de copies ont traité la quatrième partie, qui était légèrement plus difficile que les autres en terme de compréhension de l'attendu. Elle nécessitait une bonne compréhension des notions abordées précédemment dans l'énoncé et scinde les copies en deux catégories : celles qui ont tenté de traiter les questions en les effleurant, avec très peu succès, et celles qui démontraient de vraies qualités d'analyse de candidats.

Présentation Il est attendu des candidats un minimum de propreté dans leurs copies. Un trait sur la largeur de la page doit être tracé à la règle.

Il est attendu que les questions soient traitées dans l'ordre, quitte à laisser des pages vides au milieu de la copie.

Sauf éventuellement quand la question consiste à représenter graphiquement un schéma, il est attendu un minimum de rédaction pour chaque question. L'utilisation de $\Rightarrow$ avec une signification approximative en dehors de formules logiques est à proscrire. On ne met pas de quantificateurs ou d'opérateurs mathématiques au milieu d'une phrase pour éviter d'écrire un mot. Par exemple, "car $\forall$ une séquence" est à éviter, ainsi que "la $\sum$ des probabilités".

Quand des notations non présentes dans l'énoncé sont utilisées, il est nécessaire d'expliquer leur signification.

L'usage de points de suspension ne pose pas de problème lorsqu'il n'y a pas d'ambiguïté sur ce qui doit compléter ces points de suspension. Par exemple, lorsqu'on écrit $a_1a_2...a_n$, il est clair que le ... doit être remplacé par la concaténation des a_i, pour $i \in [\![2, n-1]\!]$. Certaines copies utilisent cependant des expressions comme $AA...A$ sans que la taille du mot ne soit claire (ni même si on considère un seul mot, ou l'ensemble de tous les mots de A^*). On rappelle que la notation A^k désigne la concaténation de k fois la lettre A.

Le jury reconnaît que l'écriture de code sur papier est un exercice difficile. Il est toutefois nécessaire que l'indentation soit marquée de manière précise et ne laisse pas de place au doute.

Programmation Pour les questions de programmation, il peut être pertinent d'écrire des fonctions auxiliaires non demandées par l'énoncé, mais ces fonctions doivent systématiquement être expliquées sur la copie (un simple résumé des arguments et de la sortie peut souvent suffire).

Tous les enseignants recrutés seront amenés à enseigner la programmation, au moins en Python. Il n'est pas acceptable que certains candidats ne connaissent pas la syntaxe de ce langage, ni qu'ils évitent systématiquement les questions de programmation.

Les éléments de syntaxe propres à Python sont parfois mal maîtrisés. On rappelle que contrairement au C, il faut le mot clé `def` pour définition une fonction. De plus, `else if` n'existe pas, c'est `elif` qu'il faut utiliser. Enfin, les booléens sont `True` et `False`, pas `true` et `false`.

Comme dans l'épreuve 1, le jury s'étonne de retrouver des morceaux de code de la forme :

```
if expression_booléenne:
    return True
else:
    return False
```

qui, même s'ils ne sont pas faux, montrent un manque de recul sur la manipulation des booléens, par rapport à :

```
return expression_booléenne
```

Le jury rappelle qu'on ne peut pas, en Python, accéder à une case inexistante dans une table. Ainsi, le morceau de code suivant est incorrect :

```
L = []
L[i] = 3  # incorrect !!!
```

Preuves et raisonnements Il est important de savoir repérer quand l'énoncé demande de prouver un résultat sur un exemple particulier, ou un résultat général. Traiter une question générale uniquement sur un exemple ne rapportera généralement aucun point.

Quand l'énoncé demande un seul exemple ou une seule construction, le candidat doit faire son choix et présenter une seule réponse. Ce n'est pas au correcteur de choisir, d'autant plus quand certaines réponses sont correctes et pas d'autres.

Remarques par question

Question 1 Il était inutile de perdre trop de temps à calculer les résultats. Une réponse sous forme de fraction (pas nécessairement irréductible) était parfaitement acceptée.

On rappelle que $0,8 \times 0,8 \times 0,8$ n'est pas égal à $3 \times 0,8$.

Question 2 Il y a eu un problème fréquent dans l'ordre entre transition et émission : pour que la marche 13 puisse produire la séquence AC, il faut que l'état 3 émette C, pas que l'état 1 émette C.

Question 3 De nombreuses copies n'ont pas utilisé pour cette question la définition qui était donnée explicitement juste avant la question.

Question 4 Il est attendu un minimum de justification pour la condition donnée.

Question 5 Un exemple suffit.

Question 6 Même pour les questions calculatoires, le jury souhaite juger les capacités de raisonnement du candidat, plutôt que ses capacités de calcul. Ainsi un calcul symbolique posé, même s'il est suivi d'une erreur de calcul peut être considéré favorablement, tandis qu'il est difficile de juger un résultat numérique sans explication (d'autant plus quand il est faux).

Question 7 Il n'était pas attendu de calculs dans cette question.

Question 8 La question demande la production d'un graphe et la vérification de certaines contraintes. Il faut bien entendu répondre aux deux parties de la question.

Question 9 La proximité des questions 9 et 10 a parfois occasionné des confusions et des réponses communes. Lorsque la fusion des réponses aux deux questions a été clairement identifiée et justifiée, cela n'a pas été sanctionné.

Question 13 Il faut bien répondre à l'ensemble de la question et ne pas s'arrêter au milieu.

Question 14 Même si le lien avec les automates finis semble naturel, cela n'a pas de sens de parler d'état acceptant pour ce modèle de calcul.

Question 15 Certaines copies mentionnent la nécessité d'avoir des boucles (transitions d'un état vers lui-même), ce qui est faux dans le cas général. C'est la présence d'un *cycle* qui est nécessaire (et garanti par le modèle).

Affirmer qu'un graphe avec n arêtes pour n sommets contient un cycle n'est correct que dans un graphe *non orienté* et faux dans le cas général dans un graphe orienté.

Il est cocasse de voir une réponse rédigée avec des "il faut" quand la question demande une condition suffisante.

Question 16 Il fallait prendre en compte les différentes marches permettant d'obtenir une même séquence. Cette question demandait un résultat dans un modèle de Markov *quelconque*, pas uniquement dans H_0. Une propriété P_k qu'on désire prouver par récurrence sur k ne peut pas commencer par "Pour tout $k \geq 0$, ...".

Question 17 Inutile ici de s'étendre sur les différentes méthodes possibles, mais celle à ne pas oublier était le constructeur.

Question 18 En accord avec l'énoncé, seuls le constructeur et l'affichage étaient demandés. L'écriture de toute autre méthode n'a pas été valorisée et n'a pu que représenter une perte de temps.

On attendait un affichage basique. Pas besoin de perdre du temps pour un affichage sophistiqué, avec un alignement vertical particulier.

Question 19 L'utilisation de test avec une valeur par défaut égale à `None` a été valorisée, en opposition avec l'utilisation de valeurs par défaut mutables (ce qui entraîne un risque lors d'une modification).

Question 21 Pour cette question et la suivante, il a été valorisé de comparer les flottants par une proximité plutôt qu'une égalité, pour anticiper les erreurs d'arrondis.

Question 22 Pour vérifier que la matrice de transitions/émission est valide, il faut vérifier que chaque ligne (ou colonne selon le modèle choisi) a une somme égale à 1, pas que la somme de tous les coefficients de la matrice vaut 1.

Question 23 Il manque souvent la transition de l'état 6 vers lui-même. Le vecteur des probabilités de départ doit être un vecteur de taille 6, comme le nombre d'états, pas de taille 4, comme le nombre de lettres différentes.

Question 24 Parler de « matrice » est ici imprécis. Il faut expliciter, en parlant par exemple de liste de listes, ou de liste de chaînes de caractères. Comme clairement explicité dans l'énoncé, une justification était attendue. Elle manque souvent de précision (”meilleure” n'est pas suffisant).

Question 25 Créer une matrice de la forme :

```
M = [[0] * n] * m
```

entraîne de la liaison de données entre les lignes de la matrice : la modification d'une des lignes entraîne la même modification dans toutes les autres lignes.

Il est dommage d'écrire des tests répétés de la forme :

```
if align[i][j] == 'A':
    d['A'] += 1
elif align[i][j] == 'C':
    d['C'] += 1…
```

au lieu d'écrire simplement :

```
if align[i][j] != '-':
    d[align[i][j]] += 1
```

Question 26 Une réponse de la forme « découle directement de la définition » ne rapportait aucun point. Il ne s'agit pas ici de prouver qu'on obtient un modèle de Markov caché, mais que l'objet obtenu vérifie les conditions (1), (2) et (3).

Question 27 Même si ce n'était pas explicitement précisé dans l'énoncé, il était attendu une majoration la plus précise possible. Majorer cette grandeur par le nombre d'entrées de la matrice n'apportait aucun point.

Question 28 De même, majorer une probabilité par 1 n'apporte aucun point.

Attention, le nombre d'entrées à considérer est p^2 (taille de la matrice de transitions), pas $p \times q$ (taille de la matrice d'alignement).

Question 29 Il était attendu une représentation par liste de listes d'adjacence ou liste de dictionnaires d'adjacence.

Question 30 La connaissance des complexités de l'accès aux principales structures de données était attendue.

Question 31 Si l'idée était souvent comprise, les preuves manquaient de formalisme.

Même si certaines copies ont réussi à l'utiliser correctement, il n'était pas nécessaire de procéder par récurrence, une preuve par contraposition suffisait.

Question 32 Des détails formels sont attendus ici. Redonner les formules de l'algorithme de Viterbi avec une explication succincte était parfaitement accepté.

Question 34 Il fallait ici faire référence à la question 32.

Question 35 Une justification était attendue.

Question 39 De trop nombreuses copies manquent de clarté dans les explications des différences entre les deux algorithmes.

Question 41 On attendait des justifications.

Question 42 La gestion des indices n'est pas toujours correcte.

Question 43 On attendait ici un minimum de formalisme.

Question 45 On pouvait se contenter ici de réécrire la partie qui concerne les probabilités d'émission.

Question 47 Peu de copies ont traité cette question. L'utilisation de Python pour exprimer l'algorithme n'a pas été pénalisée, en revanche il était attendu des détails sur la façon dont on fusionne les colonnes.

Question 48 La question demandait une traduction de l'algorithme précédent, le point de vigilance concernait donc notamment la syntaxe et la gestion des indices.

Question 49 Le code à produire pouvant s'avérer long, il est conseillé de commenter le code pour en faciliter la lecture.

2.3 Épreuve 3

2.3.1 Étude de cas informatique

— 83 présents
— Meilleure note : 16/20
— Moyenne : 7,14 ; écart-type : 3,7.

≥ 4.32	75%
≥ 5	65%
≥ 6.96	50%
≥ 9.6	25%
≥ 10	21,6%
≥ 15	1,2%

Commentaires généraux Au travers de l'étude de cas proposée, l'épreuve 3A vise à évaluer les compétences des candidats à modéliser un système logiciel dans sa globalité, à la fois en incluant et en dépassant les compétences algorithmiques qui attendent des réponses précises et efficaces à des problèmes clairement circonscrits. Cette compétence de modélisation requiert donc une compréhension longitudinale de ce qu'est un système logiciel et de ses interactions avec les couches matérielles et réseaux d'une part mais aussi de ses impacts environnementaux et légaux.

Par ailleurs, cette compétence de modélisation doit nécessairement conduire les candidats à réaliser une analyse du domaine métier couvert par le sujet, en proposant notamment des types de données (structures, classes, etc.) appropriés pour répondre aux questions posées, mais aussi formaliser les pré- et post-conditions nécessaires à la bonne exécution des applications modélisées. Ce travail d'analyse et de modélisation est essentiel pour cadrer le code développé, ce qui peut

conduire à proposer des algorithmes simplifiés en tenant compte des conditions dans lesquelles ils peuvent être exécutés.

Nous rappelons que tout code proposé doit être lisible *et* clairement documenté. Cette notion de code couvre non seulement les algorithmes et programmes du domaine métier du sujet mais aussi tous les tests logiciels qui peuvent être sollicités.

De manière générale, lorsqu'un candidat estime avoir déjà répondu à une question du fait de l'exhaustivité de sa réponse à une question antérieure, il est attendu de reprendre et de mettre en valeur les éléments probants dans sa réponse à la question posée plutôt que d'indiquer "question déjà traitée précédemment" en guise de réponse. Le jury cherche à recruter de bons enseignants, c'est-à-dire des personnes capables d'argumenter et d'expliquer des points précis, plutôt que de bons développeurs, dont le travail s'arrêterait à produire une fois pour toute un code parfait.

Quand une question attend une réponse OUI ou NON, il est rappelé au candidat de commencer par répondre simplement OUI ou NON avant de se lancer dans une explication longue et élaborée dans laquelle le parti choisi n'est pas clair.

Question 1 Il était attendu *a minima* que les réponses identifient l'apport du protocole HTTPS à la sécurité des échanges en ligne, notamment par la mise en place d'un certificat et le chiffrement des échanges.

Cette question visait également à s'assurer que les candidats comprennent la notion de port réseau et la réservation de certains ports à des protocoles standards, comme le port 443 pour HTTPS (80 pour HTTP, 21 pour FTP, 22 pour SSH, etc.)

Question 2 Le jury attendait que la réponse évoque différents aspects d'intérêt pour le protocole HTTPS. De nombreuses copies se sont contentées de reformuler la réponse à la question 1, ce qui n'était pas attendu. Le jury rappelle qu'on parle de chiffrement des données et non pas de cryptage.

Question 3 Il était attendu que les réponses évoquent l'ensemble des couches du modèle OSI en partant de la couche supérieure (application) et en allant jusqu'aux couches basses (physique). La réponse devait explicitement indiquer que le protocole HTTPS se place au niveau application.

Question 4 La réponse attendue était le corps d'une requête GET. Certaines copies confondent URL et requête GET. Plus surprenant, on a trouvé une copie proposant un lien hypertexte.

Il est inutile de présenter l'ensemble des autres requêtes possibles.

Question 5 Le sujet insistait sur la nécessité de disposer d'une adresse propre à chaque compétition. Utiliser des paramètres de requête ne répondait par conséquent pas à la question. Une URL *complète* était attendue et non pas une simple portion.

Question 6 Les réponses à cette question ont été parfois surprenantes et compliquées. Il était attendu une réponse mentionnant le principe de négociation de contenu propre à HTTP, et plus spécifiquement la notion de type MIME qui permet d'indiquer les préférences de formats acceptables par un client.

Question 7 Une bonne réponse doit faire la différence entre du code compilé et le code interprété par les navigateurs. Dans le cas de code compilé, il fallait mettre en avant non seulement des systèmes d'exploitation différents mais également évoquer les architectures matérielles et les jeux d'instructions distincts. Dans le cas d'un code interprété, il s'agissait d'évoquer que les codes HTML/CSS/JS manipulés par une application web étaient, eux, indépendants du matériel.

Question 8 Cette question vise à proposer une comparaison du point de vue de la vitesse d'exécution directement liée à la quantité de données échangées sur le réseau, en tenant compte du cycle de vie d'une application (depuis son installation, jusqu'à ses multiples utilisations). Des copies ont choisi de présenter les avantages et inconvénients sous forme de tableau 2 x 2, ce qui démontre une volonté de clarté, appréciée par le jury.

Question 9 Plusieurs solutions étaient possibles. Le jury attendait de retrouver une structure HTML de document globalement correcte dans sa logique et dans sa constitution.

Une majorité de réponses ont fourni la structure demandée sans donner des adresses (URL) correctes complètes.

Question 10 Peu de copies ont su expliquer l'association d'une balise à une classe identifiante. Les copies qui ont justement remarqué que les deux équipes devaient être gérées différemment étaient encore moins nombreuses.

Question 11 La réponse attendue doit expliciter l'adresse de la requête et mettre à jour les parties de la page concernant les deux équipes. Les réponses faisant usage d'extensions web hors programme ont été considérées irrecevables.

Question 12 La question invitait les candidats à rentrer dans les détails. Le simple fait d'écrire "authentification" ne suffisait pas à obtenir l'ensemble des points de la question : on pouvait évoquer un protocole à base de jeton ou de cookie en détaillant le fonctionnement applicable.

Question 13 Une fonction effectuant une requête de type POST ou PUT était attendue de manière à enregistrer le nouveau score sur le serveur d'applications. De nombreuses copies n'ont pas précisé comment la fonction `marquerPoint()` était appelée.

Question 14 Le jury a accepté des réponses basées sur la soumission d'un score en dur. Néanmoins, il attendait une réponse évoquant un délai gelant toute action sur le score.

Question 15 La réponse attendue devait désactiver les boutons localement pour éviter toute erreur de manipulation d'un utilisateur qui cliquerait sur l'un des 2 boutons, mais également les réactiver après un certain délai. Pour les autres utilisateurs connectés à l'application web, on pouvait tirer parti de la mise à jour périodique du score pour désactiver les boutons pendant un laps de temps donné.

Question 16 Une majorité de copies ont bien répondu sur le fait que le serveur est l'entité principale qui gère l'état de la compétition et qu'il vaut mieux modifier son code, par exemple pour tenir compte du délai entre 2 requêtes de mise à jour du score.

Question 17 Une bonne réponse devait surcharger les deux méthodes ___eq___ et ___hash___. Un grand nombre de copies n'ont surchargé que la méthode ___str___. Certaines copies faisaient l'erreur de penser que la méthode ___str___ renvoie par défaut l'attribut name, ce qui n'est pas le cas.

C

Par ailleurs, il est regrettable de lire des `if (test d'égalite): true else false`.

Question 18 Une bonne réponse devait faire émerger une réflexion structurée sur l'architecture de la classe `GestionMatch` et la dépendance du comportement des méthodes en fonction de l'état du match. De nombreuses copies oublient de gérer une horloge pour identifier si le match est en cours ou non.

Question 19 Il n'était pas nécessaire d'imbriquer des sous-requêtes pour répondre à cette question. Peu de copies traitent la question entièrement : elles oublient de trier les résultats et de garder le premier d'entre eux. Le jury a accepté la formulation SELECT TOP mais s'attendait à l'utilisation de la construction LIMIT 1.

Question 20 Une bonne réponse décrit le comportement de bout en bout en partant de l'ajout d'un bouton et en détaillant précisément le comportement : envoi d'une requête, action de suppression dans l'historique, etc.

Question 21 La majorité de copies ont su traiter cette question en calculant le nombre maximum de matchs dans un tournoi, la plus petite taille de poule admissible et en générant les différentes configurations envisageables jusqu'à former une unique poule avec toutes les équipes inscrites.

Question 22 Plusieurs copies ont oublié de proposer une méthode de classement pour la classe `Poule` définie.

Question 23 Une majorité de réponses n'introduisent pas de structures de données distinctes pour représenter *les deux* vues du planning. De plus, au lieu d'effectuer le calcul une seule fois, dans beaucoup de cas les méthodes `par_terrain` et `par_creneau` recalculent les vues.

Question 24 Pour les quatre cas de test logiciel pertinents, le jury attendait une explication du cas de test et un code correspondant bien formé. Dans beaucoup de cas, uniquement du code ou, inversement, du texte, était présent.

Question 25 Cette question a été traitée par très peu de copies. Dans une majorité de cas, la solution proposée se contente de trouver *un seul* planning possible. Or, la réponse attendue devait énumérer les différents plannings et les évaluer vis-à-vis des différentes contraintes pour proposer une solution valide qui minimisait les pénalités dues aux enchaînements de matchs.

Question 26–29 Cette question n'a pas posé de difficultés particulières aux candidats.

Question 30 Certains candidats ne réalisent pas que, pour sauvegarder une liste `L`, il ne suffit pas d'écrire `temp = L` en Python. La copie doit être une copie profonde.

Question 31 La réponse à cette question devait tout simplement faire usage des méthodes et des structures introduites dans les questions précédentes.

Question 32 La majorité des réponses indiquent bien que les deux notions sont semblables. Toutefois, peu de copies détaillent les factorisations qui peuvent être réalisées.

Question 33 La longueur de l'épreuve a dû se faire sentir au commencement de cette partie. De nombreuses réponses ébauchent la définition de la classe avec ses attributs mais ne spécifient pas les méthodes de calcul d'indicateurs sur les postes, le ratio des genres, etc.

Question 34 Très peu de copies ont traité cette question qui nécessite des explications pédagogiques et du code pour quatre cas de test distincts.

Question 35 Cette question n'a pas posé de difficultés particulières aux candidats.

Question 36 Cette question est sans difficulté mais dont certaines réponses montraient que la notion de *précondition* n'est pas connue de toutes les copies.

Question 37 La réponse attendue doit introduire la notion de matrice de distances, respecter les contraintes de places et minimiser la distance parcourue. Certaines propositions ne faisaient qu'annoncer une solution, sans justification.

Question 38 Peu de candidats ont traité cette question et parmi celles qui l'ont fait l'existence de plusieurs chemins équivalents était souvent mal expliquée.

Question 39 Sans difficulté particulière, la place de cette question dans le sujet a fait qu'elle n'a pas été traitée par une majorité de candidats.

Question 40 Très peu de candidats ont fait des propositions de modélisation sur cette question. Les réponses proposées ont été très incomplètes.

Question 41 Les candidats ayant traité cette question ont surtout énoncé les définitions des propriétés SOLID mais malheureusement sans les appliquer au cas d'étude.

Question 42 Cette dernière question a été facile pour les candidats courageux qui ont lu l'ensemble du sujet avant commencer à répondre aux questions, comme recommandé dans les remarques préliminaires.

2.3.2 Fondements de l'informatique

— 55 présents
— Meilleure note : 19,01/20
— Moyenne : 8,16 ; écart-type : 5,97.

≥ 2.19	75,0%
≥ 5	63,6%
≥ 8.37	50%
≥ 10	41,8%
≥ 13.6	25%
≥ 15	16,3%

Présentation du sujet Le sujet de cette épreuve aborde la question de la mécanisation de la démonstration de formules logiques via l'exploitation d'un système formel de déduction. La première partie justifie cette approche, pour la logique propositionnelle, en démontrant le théorème de complétude de la déduction naturelle. Cette démonstration est menée dans un sous-système ne contenant que les connecteurs logiques de la négation et la conjonction, en admettant en partie son équivalence avec la déduction naturelle dans toute sa généralité. La deuxième partie étudie l'implémentation d'un algorithme de recherche de preuve en calcul des séquents, dont l'équivalence avec la déduction naturelle est admise. L'introduction de ce système de déduction est justifiée par l'inversibilité des règles de déduction, qui simplifie grandement la recherche de preuve. Enfin, la troisième partie propose une ouverture à la logique du premier ordre via l'étude de la méthode des tableaux. Cette méthode repose sur la résolution de problèmes d'unification, qui sont résolus dans ce sujet grâce à l'algorithme de Martelli et Montanari.

Commentaires généraux L'épreuve a été compliquée pour un certain nombre de candidats qui avaient manifestement fait l'impasse sur la logique formelle — laquelle figure au programme de CPGE MPI et aussi au programme complémentaire spécifique de l'option 3B —, et qui de surcroît n'ont pas été en mesure de comprendre adéquatement les rappels de l'énoncé. Le jury rappelle l'importance de se confronter, durant sa préparation, à l'intégralité du programme du concours.

Concernant la rédaction d'arbres de dérivation, de trop nombreux candidats se servent d'équivalences sémantiques pour simplifier la recherche de dérivations en modifiant les séquents manipulés.

Cela va à l'encontre du principe même de l'usage de systèmes formels de déduction, qui sont par essence purement syntaxiques, et cela constitue donc un contre-sens quant à l'objectif affiché de ce sujet. Dans le même ordre d'idées, une dérivation de l'équivalence de deux formules ne peut servir de prétexte à la substitution de certaines formules d'un séquent sans utiliser les règles de déduction idoines. Par ailleurs, le fait qu'une formule est une tautologie, ou plus généralement une conséquence sémantique d'un ensemble de formules, ne peut être un argument justifiant qu'un séquent est dérivable tant que l'on n'a pas démontré le théorème de complétude. Enfin, lorsqu'il est demandé de donner une dérivation d'un séquent sans hypothèses $\vdash A$, il est inutile d'ajouter un ensemble quelconque d'hypothèses pour démontrer le séquent $\Gamma \vdash A$.

Certains candidats utilisent la locution « par induction » lorsqu'ils invoquent l'hypothèse d'induction. C'est une erreur : l'induction n'ayant pas encore été établie à ce stade, elle ne saurait être utilisée. En revanche, on dispose bien d'une *hypothèse d'induction* portant, par exemple, sur les sous-formules.

Les candidats semblent avoir bien compris que la connaissance de tous les langages de programmation au programme du concours est importante. Le langage OCaml semble mieux maîtrisé par les candidats ayant opté pour le sujet de fondements de l'informatique que les années passées, et moins de candidats évitent systématiquement les questions de programmation. On rappelle toutefois qu'en OCaml, `a ; b` provoque un avertissement si `a` n'est pas de type `unit` ; le cas échéant, il convient d'utiliser toute construction permettant d'ignorer la valeur de `a`. Par exemple, `ignore a; b` est la construction idiomatique, mais `let _ = a in b` conviendrait également parfaitement.

Commentaires par question

Question 3 De nombreuses copies contenaient une formule raisonnable permettant de caractériser l'implication. La difficulté résidait dans la rédaction d'une dérivation prouvant la correction de cette caractérisation.

Question 4 Sans grande difficulté, cette question a nettement révélé les candidats ayant fait l'impasse sur la logique formelle.

Question 5 Une preuve par induction, sans nécessairement détailler tous les cas, était attendue, et ce d'autant plus qu'il s'agit de la première question du sujet qui fait appel à ce principe de démonstration. Les candidats doivent comprendre qu'elle a pour objet de vérifier qu'ils savent rédiger convenablement ce genre de preuve, une qualité *indispensable* pour un agrégé d'informatique. Se borner à indiquer que les règles d'inférence restent applicables en ajoutant des formules aux hypothèses des séquents est donc ici insuffisant.

Question 6 Certaines copies rédigent la preuve qu'il existe une formule de $\mathcal{F}_1$ équivalente à une formule de $\mathcal{F}_0$ donnée, alors que l'énoncé précise que ce point est admis.

Question 7 Il faut se garder de mélanger des éléments de sémantique (les valeurs de vérité V et F) et des symboles appartenant à la syntaxe des formules. Ainsi, on ne peut écrire $(F \to (F \to F)) \to F \equiv F$ pour justifier que la formule donnée n'est pas une tautologie.

Certains candidats oublient de donner une valeur de vérité à la variable p_1 (pour obtenir une valuation initiale) ou de mentionner la minimalité du rang de la valuation.

Question 9 Plusieurs copies utilisent implicitement l'associativité de $\wedge$ dans leur manipulation des formules. Ce n'est pas convenable : l'énoncé définit précisément la construction $\bigwedge_{k=0}^{n} A_k$ comme étant égale à $\left(\bigwedge_{k=0}^{n-1} A_k\right) \wedge A_n$; par conséquent, en définissant $B_k = A_k$ pour $k < n$ et $B_n = A_n \wedge A_{n+1}$, on n'a pas $\bigwedge_{k=0}^{n+1} A_k = \bigwedge_{k=0}^{n} B_k$, car le parenthésage n'est pas le même. Il aurait fallu définir $B_0 = A_0 \wedge A_1$ puis $B_k = A_{k+1}$ pour $k > 0$.

Pour la même raison, si on choisit de procéder par applications des règles $\wedge_e$, on doit appliquer *une* fois $\wedge_{e,d}$ pour passer de A_i à $\bigwedge_{k=0}^{i} A_k$ dans la conclusion, puis $n-i$ fois $\wedge_{e,g}$ pour passer à $\bigwedge_{k=0}^{n} A_k$.

Question 10 De nombreuses copies font appel à la question 2 pour utiliser une loi de De Morgan qui n'est pas celle qui a été démontrée en question 2. Cela a toutefois été toléré.

Le jury accepte que l'on se contente de dire d'un cas de preuve qu'il est similaire à celui qu'on vient de détailler, mais seulement lorsque c'est vrai ! En l'espèce, lorsqu'on traite le cas inductif $A = \neg B$ par exemple, le cas $[\![A]\!]_v = F$ n'est pas similaire au cas $[\![A]\!]_v = V$.

Question 14 Ici encore, on retrouve parfois un mélange de syntaxe et de sémantique, les valeurs de vérité V et F étant utilisées comme des formules par certains candidats.

Question 16 Certains candidats, visiblement gênés par la possibilité de n'avoir aucune formule dans la conclusion d'un séquent, ont choisi d'y ajouter une formule $\bot$ lorsque c'était le cas, puis de la retirer lorsqu'une autre formule était ajoutée à la conclusion du séquent. C'est une erreur d'application des règles de l'énoncé, même si les règles implicitement utilisées par ces candidats sont tout à fait valides.

Question 17 De nombreux candidats se servent de la notation $\vDash$ à mauvais escient. En effet, on ne peut écrire un raisonnement de la forme « si A est vraie alors $\Gamma \vDash A, \Delta$, puis si A est fausse, alors il existe une formule de Δ qui est vraie, donc $\Gamma \vDash A, \Delta$ » car $\Gamma \vDash A, \Delta$ signifie que *pour toute valuation* qui rend vraies les formules de Γ, il existe une formule de $\Delta \cup \{A\}$ qui est rendue vraie par cette valuation.

Question 19 La construction `try … except` n'est pas valide en langage OCaml.

Question 20 La plupart des candidats ont écrit une fonction permettant de localiser ou d'extraire une formule non atomique d'une liste de formules. Malheureusement, aucun candidat n'a pensé à mettre de côté les formules atomiques rencontrées afin d'éviter de les parcourir à nouveau dans les appels récursifs.

Question 23 Certaines copies proposent une borne linéaire en la taille du séquent initial en oubliant que la règle de la conjonction à droite duplique des formules.

Question 26 Certains candidats proposent pour la première équation la substitution σ telle que $\sigma(x) = f(y, z)$ et $\sigma(y) = a$: cela ne convient pas, car une substitution est appliquée en une seule fois à un terme et non par couches successives, de sorte que $t\sigma = f(f(y, z), a)$ tandis que $u\sigma = f(f(a, z), a)$.

Pour la deuxième équation, il ne fallait pas se borner à dire que l'unification est impossible, mais aussi dire pourquoi (par exemple, donner un argument de hauteur établissant qu'on ne peut avoir $\sigma(z) = f(\sigma(y), \sigma(z))$).

Question 28 Utiliser `List.fold_left` ici est un peu pédant dans la mesure où `List.exists` convient très bien.

Question 29 La composition des substitutions dans le dernier cas de l'algorithme d'unification a souvent été oubliée ou mal implémentée : la liste $(x, t) :: \sigma$ ne représente pas la composition $(x := t) \circ \sigma$.

Chapitre 3

Épreuves orales

Pour alléger le texte, ce chapitre est uniquement écrit au féminin [1], mais cela doit être pris dans le sens du neutre : l'ensemble de ce texte, sans exception, concerne uniformément candidates et candidats.

Les épreuves orales se sont déroulées du 17 au 22 juin 2024 au lycée Guy Mollet à Arras. Les candidates admissibles ont été convoquées pour les trois épreuves orales sur trois jours consécutifs. Le jury est bien conscient du stress et de la fatigue des candidates avec des temps de préparation longs et des horaires parfois décalés. Son objectif n'est pas de piéger les candidates mais de les mettre dans un contexte leur permettant de montrer le meilleur d'elles-mêmes.

Le jury a jugé l'oral de l'agrégation d'un très solide niveau global, à la fois en matière de connaissances disciplinaires et de compétences pédagogiques.

Le jury rappelle que les livres sont disponibles pour les préparations de toutes les épreuves. La liste des livres fournis par le jury est disponible sur le site `https://agreg-info.org` et sera mise à jour pour la session 2025. Les préparations à l'agrégation ont elles aussi mis à disposition des livres, également accessibles à toutes les candidates. L'environnement informatique à disposition des candidates (pour les trois épreuves) est celui téléchargeable sur le site du jury et les fichiers des candidates sont sauvegardés régulièrement sur un serveur. Il est à noter que les responsables informatiques ne sont pas là pour aider les candidates à débugger leur code, mais peuvent intervenir en cas de souci avec l'environnement à disposition. Consulter la documentation disponible dans cet environnement doit être le premier réflexe.

De nombreux conseils et remarques s'appliquent à l'ensemble des épreuves orales et sont regroupés ici.

Les trois épreuves sont distinctes et permettent au jury d'évaluer des points différents. Il est donc requis des candidates qu'elles connaissent la description et les attendus des trois épreuves.

Des temps maximaux sont donnés pour les trois épreuves et rappelés au début de chaque oral. Il est important d'essayer de se rapprocher le plus possible de ces temps sans toutefois les dépasser : même s'ils sont indiqués comme des temps à ne pas dépasser (le jury interrompra la candidate le cas échéant), ils sont en fait des temps-cibles. Et le jury n'interrompt pas la candidate pendant cette présentation, même si celle-ci n'est pas claire.

Le jury d'une épreuve d'oral n'a pas connaissance des résultats de l'écrit de la candidate et s'impose une discipline stricte de ne pas discuter des prestations d'une candidate devant les autres membres du jury avant que celle-ci ait passé l'ensemble de ses épreuves. Lorsqu'une candidate se présente à une épreuve, elle est ainsi assurée d'être examinée sans aucun préjugé.

Il est attendu que la candidate adopte une posture d'enseignante lors des interrogations orales, sans toutefois considérer que le jury doit nécessairement adopter une posture d'apprenant. Des présentations didactiques et illustrées sont notamment attendues. Le jury apprécie notamment les remarques pédagogiques.

1. Le chapitre 2 est écrit au masculin.

Lorsque la candidate présente du code (notamment en TP et modélisation), le jury s'attend à une vraie exécution du code pendant la présentation.

Les épreuves orales ne sont pas un jeu de rôle dans lequel la candidate joue à la professeure et le jury joue à la classe indisciplinée. Au cours de son exposé ou de ses réponses aux questions du jury, la candidate ne doit en aucun cas faire référence à un élément de compréhension qu'elle aurait expliqué dans une séance antérieure comme elle pourrait le faire avec des élèves qu'elle suivrait sur une année complète. Lorsque le jury pose une question, la réponse « vous n'avez pas été attentive lors du cours de la semaine dernière » est dilatoire et à éviter.

Les candidates ne doivent pas poser de questions au jury (autre que demander le temps) ou tester les connaissances du jury. Cette forme de pédagogie inversée n'est pas le but d'une épreuve d'agrégation. Le jury est là pour évaluer les candidates, et non pas pour enseigner aux candidates ce qu'elles ne savent pas.

On s'attend à de l'honnêteté de la part des candidates : il est préférable de dire qu'on ne sait pas que de dire une bêtise avec aplomb.

Le jury peut poser des questions simples à la candidate, qui ne doit pas chercher systématiquement des réponses compliquées. Le jury peut aussi interroger sur un spectre thématique plus large mais en lien avec le thème de l'épreuve en cours, par exemple en posant des questions liées à l'impact de la hiérarchie mémoire sur le comportement d'un programme, même si le thème de l'épreuve est plus algorithmique. Le jury vise à recruter des informaticiennes *complètes*. Les questions peuvent permettre d'évaluer la capacité des candidates à faire des liens avec d'autres sous-domaines que celui étudié spécifiquement par le sujet de l'oral.

Une membre de jury est susceptible d'interrompre une candidate pendant sa réponse à une question, sans que cela ne doive être perçu négativement, pour différentes raisons : elle a obtenu la réponse attendue, elle souhaite enchaîner sur une autre question sur le même thème, elle estime que le temps pour répondre risque d'être trop long…

Les candidates ne doivent pas chercher un sens caché ou figuratif aux questions du jury. En cas de doute, il peut être constructif pour une candidate de reformuler la question avant d'y répondre. Le jury ne manquera pas de reformuler la question à son tour si les deux diffèrent.

Le jury rappelle que la gestion du tableau est également évaluée. Un contenu propre, clair et organisé, éventuellement utilisant des couleurs, sera valorisé. Les candidates peuvent effacer le tableau ou une partie de tableau après avoir demandé l'autorisation du jury. En particulier, le jury peut demander à ne pas effacer des parties du tableau. Les candidates peuvent toutefois s'autoriser à effacer ce qui n'est pas du contenu (une faute d'orthographe, un trait mal tracé…). Elles devraient pour cela utiliser la brosse plutôt que d'effacer avec leurs doigts. Le jury s'attend à ce que la candidate économise l'espace du tableau et structure son occupation, pour éviter tant que possible de réclamer à effacer.

Dans les épreuves au cours desquelles la candidate est amenée à projeter du code au tableau, le jury abaisse et relève l'écran blanc à la demande de la candidate. L'écran de projection pouvant cacher une partie du tableau, la candidate est invitée à anticiper l'usage des différentes zones du tableau et sa chronologie, dès son entrée dans la salle d'examen. Il est, par exemple, admissible que la candidate commence son exposé sur la partie du tableau jamais cachée par l'écran et exploite initialement l'autre zone pour projeter son code sur l'écran déroulé, puis, en fin d'exposé, fasse relever l'écran définitivement et écrive sur la partie de tableau restante. Enfin, les candidates ne doivent pas faire de transparents (comme une présentation beamer), sauf éventuellement pour présenter un schéma.

Il est à noter qu'en 2024, l'ensemble des interrogations étaient ouvertes aux visiteuses.

3.1 Leçon

— 43 présents
— Meilleure note : 18.25/20
— Moyenne : 9.64 ; écart-type : 4.55.

≥ 5	81,3%
≥ 7.13	75%
≥ 8.13	50%
≥ 10	46,5%
≥ 14.0	25%
≥ 15	18,6%

L'épreuve comprend un temps de préparation de quatre heures et un oral d'une heure avec le jury. La candidate se voit proposer (aléatoirement) deux sujets au choix ; elle doit choisir de traiter l'un des deux, mais n'a pas à justifier son choix sur ce point. Elle acte son choix au début de l'interrogation proprement dite et peut donc changer d'avis en cours de préparation (il va sans dire que ce n'est toutefois guère conseillé). Le jury n'interroge pas, bien entendu, sur le sujet qui n'a pas été choisi.

La candidate a le droit d'utiliser les notes qu'elle a prises pendant sa préparation tout au long de l'oral. Elle dispose à sa guise du tableau, avec pour seule contrainte de demander au jury avant d'effacer.

L'oral se décompose en trois parties successives.

Premièrement, pendant 10 minutes, la candidate présente le plan de leçon qu'elle a préparé et dont une copie aura été remise au jury. Au cours ou à la fin de cette présentation, la candidate indique au jury deux propositions de développement (ils doivent également être signalés comme tels dans le plan).

Deuxièmement, pendant 20 minutes, la candidate présente celui des deux développements qui aura été choisi par le jury.

Tout au long de ces deux premières phases, le jury n'intervient à aucun moment. La candidate ne doit donc pas lui poser de question ni attendre de sa part aucune marque d'approbation ou d'improbation. Les temps indiqués ci-dessus sont des cibles : le jury interrompra la candidate en cas de dépassement, mais il faut également éviter de faire des présentations trop courtes (cela n'est pas sanctionné en soi, mais un contenu trop peu abondant le sera). Attention : le jury ne peut garantir que les salles d'interrogation seront équipées d'une horloge. Il est donc prudent que les candidates se munissent de leur propre dispositif, qui ne peut bien sûr pas être un téléphone ni une montre connectée.

La troisième partie occupe le reste de l'heure et est consacrée aux questions du jury et aux réponses de la candidate à ces questions.

Si la leçon d'agrégation permet d'évaluer les compétences didactiques des candidates, elle est cependant un oral scientifique avant tout. Cela a des conséquences sur les trois moments de l'oral :

— le plan doit comporter du contenu scientifique, et non seulement l'annoncer ;
— le développement doit bien sûr montrer des qualités d'exposition, mais appliquées à un contenu scientifique clair et précis ;
— les questions posées par le jury peuvent porter sur la façon dont on expliquerait telle chose à tel type d'élève, mais elles portent également fréquemment sur le fond des notions abordées, au niveau de recul M2 attendu des candidates.

Sur chaque point, la maîtrise et le recul didactique sur le contenu scientifique sont appréciés et valorisés, mais il s'agit de la cerise sur le gâteau : il faut d'abord qu'il y ait un gâteau.

Les candidates ne doivent pas transformer l'épreuve de leçon en une épreuve de TP ou de modélisation. Ainsi, bien qu'un vidéo-projecteur soit mis à leur disposition, son usage devrait être très exceptionnel et bref dans cette épreuve (par exemple pour afficher un graphe avec beaucoup d'arcs qu'il serait inutilement long de dessiner pendant le développement). Évaluer l'aptitude des candidates à utiliser l'ordinateur pour illustrer des phénomènes ou des concepts (comme par exemple une animation montrant le déroulement d'un algorithme) n'est pas un objectif de cette épreuve. De même, la production de programmes informatiques est évaluée lors de l'épreuve de TP.

Par ailleurs, l'agrégation est un concours de recrutement d'enseignantes : une expertise sur des domaines hors programme ne se substitue pas à un recul pédagogique.

Présentation du plan

Le jury souhaite voir proposé par la candidate un plan de la leçon d'au plus trois feuillets A4 manuscrits. Il est demandé de rédiger le plan dans un format spécifique permettant la photocopie pour le jury dans de bonnes conditions. Le fichier pdf correspondant se trouve sur le site du jury. Des feuilles à ce format sont bien entendu mises à disposition lors de la préparation.

Un bon plan n'occupe pas nécessairement la totalité de ces trois pages, mais il est regrettable de voir des candidates n'en utiliser qu'une seule, surtout très aérée.

La dénomination traditionnelle de *plan* peut être trompeuse : **le plan d'une leçon d'agrégation n'est pas une table des matières ni une simple annonce de la structure de la leçon. Il doit comporter, outre cette structure, le contenu scientifique correspondant.** Ainsi, dans une leçon sur l'algorithmique du texte, une candidate ne peut pas simplement, par exemple, écrire dans son plan « motif », elle doit choisir une définition formelle de cette notion et l'écrire ; de même, si elle choisit de mentionner l'algorithme de Rabin–Karp, elle ne peut se contenter d'en donner le nom, elle doit en écrire une description formalisée. Il en va de même des propriétés, théorèmes, résultats de complexité, etc. En effet, le jury souhaite pouvoir, dans la phase de questions, interroger, par exemple, la cohérence ou l'intérêt du choix de formalisation de la notion de motif vis-à-vis de la formulation retenue pour l'algorithme de Rabin–Karp. L'inobservance de ces prescriptions emporte une triple peine pour la candidate, car

— le format de l'épreuve n'est pas respecté,
— le plan est souvent superficiel et sa présentation tourne court et
— au lieu d'interroger sur les liens entre les éléments du plan, le jury en est réduit à demander d'expliciter les éléments qui auraient dû l'être dans le plan, ce qui est peu intéressant et surtout bien plus périlleux.

Il est opportun de numéroter les éléments du plan afin de permettre au jury d'y faire référence au cours de la discussion. Une numérotation séquentielle globale est la plus pratique (c'est-à-dire : définition 1, exemple 2, définition 3, et non : définition 1, exemple 1, définition 2). Si on choisit une numérotation par partie, on rappellera le numéro de la partie (c'est-à-dire, dans la partie 2 : définition 2.1, exemple 2.2, définition 2.3, et dans la partie 3 : exemple 3.1).

Lors de la présentation orale du plan, le jury ne souhaite pas que la candidate recopie les grandes lignes du plan en restant dos au jury. La présentation du plan ne doit pas non plus consister en sa lecture ou sa paraphrase. Le jury apprécie que les choix pédagogiques et l'enchaînement logique du plan soient soulignés. Un plan « catalogue » (reprenant par exemple toutes les structures de données imaginables pour la leçon *Implémentations et applications des piles, files et files de priorité*) n'apparaît pas comme un choix judicieux. Il est tout à fait possible de signaler pourquoi on a choisi de ne pas inclure certains éléments dans la leçon : enseigner, c'est aussi savoir poser les limites de ce que l'on transmet à un public donné.

Le jury apprécie des suggestions de travaux pratiques, démonstrations ou codes qui seraient une partie du cours correspondant.

Développement

Les deux développements proposés au jury doivent être introduits dans la présentation orale du plan et visibles sur le plan écrit. L'intitulé des développements doit permettre au jury de se faire une idée assez précise de leur contenu. Proposer un seul développement est nettement sanctionné. À cet effet, deux développements trop semblables comptent comme un seul et un développement manifestement hors sujet ou manifestement trivial ne compte pas. Cependant, préparer un seul développement et en proposer un deuxième pour la forme est encore plus risqué : le jury ne choisira pas systématiquement le développement le plus alléchant. Proposer trois développements ou davantage n'apporte aucun bénéfice ; il est demandé aux candidates de ne pas le faire.

Le développement peut traiter de sujets très variés : démonstration d'un théorème, présentation d'un protocole et formalisation de ses propriétés, exposition rigoureuse d'un algorithme, etc.

À titre d'exemple, dans ce dernier cas, le déroulé de l'algorithme sur un exemple peut apporter une plus-value pédagogique au développement, mais il ne saurait en aucun cas en constituer le cœur : le jury attend des résultats précis sur la correction, la complexité, les performances, etc.

De même, un développement centré sur un théorème, comme le fait que tout algorithme de tri par comparaison a une complexité au pire $\Omega(n \log(n))$, bénéficie d'une présentation et d'une explication du raisonnement, mais celui-ci doit aussi être formalisé : certes, on ne démontrera probablement pas *chacun* des lemmes (relation entre nombre de feuilles et taille dans un arbre binaire strict, etc.), mais, le résultat n'étant pas d'un niveau très élevé par rapport à ces lemmes, il est malvenu de ne pas prouver rigoureusement *une partie* d'entre eux.

En résumé, si la qualité et la clarté de la présentation sont des aspects importants, il ne faut en aucun cas négliger d'apporter un contenu scientifique profond ; on ne peut se contenter d'un exposé type « leçon de choses ».

Le développement doit être soigné et maîtrisé, ce qui ne signifie pas être appris par cœur et récité ou recopié depuis les notes — le jury le vérifiera lors de la phase de questions. Il est par exemple préférable de présenter un algorithme simple et bien maîtrisé qu'un algorithme complexe et mal compris. Il est toutefois rappelé que la candidate a le droit, sans pénalité ni permission préalable, de consulter ses notes pendant son développement.

Le développement doit avoir un réel rapport avec la leçon et être convenablement inséré dans le plan. Ainsi, un développement sur un algorithme classique de calcul d'enveloppe convexe d'un nuage planaire n'a pas vraiment sa place dans la leçon *Implémentations et applications des piles, files et files de priorité* dès lors que, s'il utilise inévitablement des structures de données, celles-ci n'y jouent pas un rôle central.

Questions

En général, le jury commence par poser des questions sur le développement qui vient d'être exposé, puis élargit la discussion au plan et au thème de la leçon. Il peut interroger, d'une part, sur toute partie du programme de l'épreuve et, d'autre part, sur tout élément que la candidate introduit dans son plan ou dans son exposé.

De nombreuses questions simples visent simplement à vérifier que la candidate comprend et maîtrise ce qu'elle écrit ; elles sont posées assez systématiquement et la candidate ne doit pas en être déstabilisée.

Certaines questions peuvent aussi porter sur l'enchaînement des éléments du plan ou sur sa cohérence interne. Le jury s'attend à des notations homogènes de la part de la candidate, même si elle a utilisé plusieurs sources adoptant des conventions de notation ou des définitions légèrement différentes. Si le travail d'homogénéisation n'est pas fait dans le plan, la candidate sera interrogée et invitée à le faire pendant l'échange avec le jury. La candidate doit également faire attention au développement logique de son cours : il peut être maladroit d'utiliser des notions essentiellement introduites postérieurement dans le plan. Le jury ne manquera pas de questionner ces choix.

Les candidates doivent bien écouter les questions qui leur sont posées, et notamment bien différencier, par exemple :

- *Comment expliqueriez-vous l'algorithme de Rabin–Karp à un bon élève de terminale NSI ?* : de nombreuses réponses sont possibles ; on peut notamment envisager de dérouler l'algorithme à la main sur un exemple bien choisi ;
- *Pouvez-vous expliquer l'algorithme de Rabin–Karp ?* : on attend une réponse scientifique de même nature que pour un oral de master.

Si elles présentent une « méthode générale » dans leur plan, les candidates peuvent s'attendre à ce que le jury leur fournisse un exemple ou un cas d'usage à dérouler. Par exemple, dans une leçon sur la *programmation dynamique* où la candidate expose une manière générale d'obtenir une équation de récurrence entre sous-problèmes, le jury peut proposer sous forme d'un petit exercice un exemple de problème où il faut d'abord trouver cette équation et les bons sous-problèmes. Plus généralement, les candidates doivent s'attendre à des questions techniques (résoudre un exercice, détailler un calcul de complexité, répondre à une question de programmation...).

Il est bien sûr attendu des candidates qu'elles répondent avec honnêteté aux questions : il n'est pas scandaleux de se tromper, mais il est scandaleux de tenter de tromper le jury en affirmant avec aplomb des choses fausses ou pour le moins douteuses — heureusement, ce phénomène est très rare. Lorsqu'on ne sait pas, la bonne attitude est de réfléchir avec le jury, ou, en dernier recours, de dire « je ne sais pas ». Du reste, certaines questions posées par le jury (notamment au plan pédagogique) n'admettent pas *une* bonne réponse, et c'est davantage la capacité de la candidate à réfléchir et envisager plusieurs possibilités qui est évaluée.

3.2 Travaux pratiques

— 43 présents
— Meilleure note : 20/20
— Moyenne : 11.09 ; écart-type : 4.55.

≥ 5	93,0%
≥ 8.33	75%
≥ 10	53,4%
≥ 10.67	50%
≥ 15	25,5%
≥ 15.0	25%

Cette épreuve orale a pour objectif l'évaluation des compétences pratiques des candidates dans un cadre pédagogique. D'une durée de 60 minutes, elle se déroule en deux phases : une présentation de 30 minutes par la candidate suivi de 30 minutes de questions et d'échange avec le jury.

La phase de présentation comporte deux parties : la partie "Travaux pratiques" proprement dite où la candidate doit, à partir d'un sujet, produire et commenter du code en insistant sur sa correction. La seconde partie, dite de "Revue de code", consiste pour la candidate à commenter un code de mauvaise qualité ou défectueux, comme l'on pourrait en trouver durant une véritable séance de travaux pratiques avec des élèves ou des étudiantes, et présenter des pistes de correction et d'amélioration.

Pour chaque partie, l'accent est mis sur la pédagogie en plus de la compétence disciplinaire et non sur la virtuosité. Dans chaque partie, le code à écrire ou à analyser ne doit pas poser de difficultés insurmontables.

Comme pour les deux autres épreuves orales, on attend de la candidate une conduite claire et organisée de l'exposé, incluant une courte mise en contexte et une présentation structurée des résultats. L'usage du tableau doit être au service de la compréhension du discours de la candidate, et celui d'une vidéo-projection doit se concentrer autour de la projection de code, de l'exécution de tests et éventuellement de la présentation (éventuellement sous forme graphique) de résultats d'exécution. Une présentation basée sur des diapositives n'est pas souhaitée.

Enfin, la candidate doit veiller au respect de l'équilibre entre les deux parties de l'épreuve et, en particulier, ne pas passer trop de temps sur la revue de code. On s'attend à ce qu'elle dure entre 5 et 10 minutes.

Partie Travaux pratiques

Dans cette partie, lors de la production du code, l'accent doit être mis sur la clarté et la facilité de compréhension, ce n'est pas une épreuve de virtuosité ou de vitesse. Durant la préparation de cette épreuve, la candidate devrait avoir en tête des questions comme "Mon code peut-il être présenté à des élèves ?", voire "Est-il digne de figurer dans un manuel ?" Pour cela, des éléments auxquels il faut porter une attention particulière sont le choix des noms de variables (conventions de casse par exemple), l'usage de commentaires, la factorisation des expressions ou encore l'organisation spatiale du code (indentations, sauts de lignes).

La candidate doit particulièrement travailler la réflexion concernant les choix de structures de données lorsqu'on lui demande de les proposer ; en effet, un mauvais choix peut avoir de lourdes conséquences.

Durant la présentation, les candidates doivent éviter une présentation linéaire du code produit. À la place, elles sont encouragées à décrire la structure de leur code et les principaux points clés, le jury pouvant ensuite, durant les questions, demander à la candidate de revenir sur des portions spécifiques du code ou de développer plus en détail un point technique. Il est possible, voire recommandé, de structurer le code produit en plusieurs fichiers. Une utilisation d'un langage en respectant ses usages et spécificités est appréciée. Par contre, une utilisation excessive de bibliothèques peut nuire à la compréhension du code. De plus, il est déconseillé de corriger le code en direct pendant sa présentation.

Il est mal vu pour une candidate d'essayer d'impressionner le jury en présentant des optimisations excessives de code si elles ne sont pas utiles. Par exemple, réécrire une fonction récursive afin d'avoir de la récursivité terminale n'est pas pertinent si le nombre d'appels récursifs de cette fonction reste peu élevé. L'accent doit être mis sur la production d'un code simple, lisible, maîtrisé, dont le fonctionnement est parfaitement compris et expliqué.

Une fois le code présenté, il importe de transmettre l'idée que celui-ci est correct. Cela repose tout d'abord sur une exécution du code produit qui doit être conforme aux spécifications : le jury s'attend à de telles exécutions, sur des exemples pertinents. Il peut aussi, durant la phase de questions, demander de nouvelles exécutions, avec par exemple du code compilé en direct, ou sur des entrées de son choix.

Mais la justification de la correction d'un code ne s'arrête pas à des exécutions correctes sur des exemples spécifiques, et d'autres techniques doivent être mises en œuvre, par exemple une preuve formelle ou l'utilisation de tests. En particulier, trop peu de candidates présentent spontanément des tests aux cas limites.

Concernant les preuves formelles, sans forcément utiliser une formalisation excessive, l'utilisation et la justification de variants et d'invariants bien choisis peuvent contribuer efficacement à convaincre du bon comportement d'un algorithme.

Une autre méthode pouvant simplement être mise en œuvre lors de l'oral repose sur l'usage de jeux de tests. Une présentation exhaustive de ceux-ci n'est a priori pas pertinente, et la candidate doit au contraire transmettre l'idée que le choix des tests utilisés se place dans une démarche méthodologique rigoureuse et construite, et éviter d'en faire une simple énumération. Enfin, l'utilisation d'outils d'analyse simples est appréciée.

Les candidates peuvent être questionnées sur tous les aspects de leur code, ainsi que sur toutes les phases de développement. Concernant le code, elles peuvent être interrogées sur les structures de données choisies, les algorithmes mis en place, le typage, la complexité, les performances, etc. Sur le processus de développement, le jury s'attend à ce que les candidates connaissent les processus de compilation, d'exécution, de test et de déboguage.

Partie Revue de code

Durant cette partie, la candidate est confrontée à des éléments de code défectueux ou maladroits, dans un ou plusieurs langages parmi ceux du programme, et doit les étudier afin d'en proposer des améliorations (correction d'erreurs, restructuration, etc.), comme le ferait une enseignante avec une élève durant une séance de travaux pratiques.

Comme pour les autres épreuves orales, le jury attend un exposé structuré de la revue de code, avec une mise en contexte et une présentation organisée et motivée des corrections à apporter plutôt qu'une énumération séquentielle. Le jury apprécie que la candidate en profite pour rappeler et appliquer de bonnes pratiques de programmation.

Bien sûr, il est attendu que le code corrigé soit correct, celui-ci doit donc avoir été exécuté et testé et le jury peut, durant les questions, demander explicitement une exécution du code dans les conditions de son choix.

Rappelons enfin qu'il est demandé à la candidate de corriger le code proposé, en essayant de rester au plus près du code initial (et donc de l'intention de l'élève) et non d'en proposer une

version sans rapport avec le code initial, même si celle-ci est correct. L'objectif n'est pas, pour la candidate, de montrer qu'elle est capable de produire le code demandé mais, dans un objectif pédagogique, de voir comment elle peut aider une élève pour que celle-ci soit capable d'écrire correctement un tel code.

Remarques spécifiques pour la session 2024

Remarques sur le sujet *Ordonnancement*

Ce sujet étudiait des problèmes d'ordonnancement autour d'un flux de tâches : un ensemble de commandes doit être traité séquentiellement, chacune d'elles devant passer successivement par un ensemble d'ateliers dont l'ordre est fixé. L'objectif est de minimiser le temps total mis pour traiter toutes les commandes, en fonction de l'entrée indiquant le temps de traitement de chaque commande dans chaque atelier.

Le sujet était décomposé en deux parties, suivant le langage de programmation à utiliser et les techniques algorithmiques à mettre en œuvre.

Dans un premier temps, on effectuait en langage C une résolution exacte pour deux ateliers à l'aide de la méthode de Johnson puis, en autorisant une phase de maintenance (le problème devenant alors NP-difficile), le calcul efficace d'une approximation de la solution, et la résolution exacte par programmation dynamique.

Dans une seconde partie, on demandait de programmer en langage OCaml pour le cas général du nombre d'ateliers (là encore, le problème est NP-difficile) une résolution non-optimale par méthode gloutonne puis une méthode optimale par recherche exhaustive avec séparation et évaluation (ou *branch and bound*).

La revue de code proposait deux algorithmes de tri en Python et en C.

Le jury a globalement été satisfait du traitement de cette épreuve, qui offrait un survol assez exhaustif des techniques algorithmiques au programme. Certaines candidates ont traité le sujet quasi-intégralement, allant jusqu'à présenter plusieurs métriques pour la recherche exhaustive, ce que le jury a apprécié. À l'inverse, le jury a regretté que certaines candidates ne maîtrisent pas les méthodes algorithmiques de base.

Remarques sur le sujet *Gestion de réservations*

Ce sujet portait sur la mise en place d'un serveur de gestions de réservations. Il se décomposait en deux parties.

Dans un premier temps, on demandait d'implémenter en langage OCaml une structure de données, inspirée de l'algorithme de nœuds-chapeaux et des *segment trees*, permettant d'effectuer efficacement les divers opérations nécessaires à une gestion de réservation : déterminer si une réservation est possible, effectuer ou annuler une réservation.

Dans un second temps, on demandait de mettre en œuvre une petite architecture client-serveur (puis pair-à-pair) pour simuler un service décentralisé de réservations, en langage C. On fournissait aux candidates une petite bibliothèque fournissant une implémentation en C de la structure de données que l'on demandait d'écrire dans la première partie, permettant d'avoir deux parties globalement indépendantes. Les candidates avaient de plus à leur disposition une aide-mémoire pour programmation de *sockets*.

La revue de code proposait une fonction en langage OCaml et quelques requêtes en SQL.

La première partie reposait sur la manipulation d'arbres binaires en langage OCaml. Dans la seconde partie du sujet, la mise en place d'une petite architecture client-serveur consistait principalement à programmer l'ouverture d'un *socket*, ce que de nombreuses candidates ont eu des difficultés à faire, en dépit de l'aide-mémoire fourni.

3.3 Modélisation

— 43 présents
— Meilleure note : 20/20
— Moyenne : 9.83 ; écart-type : 4.93.

≥ 5	79,0%
≥ 5.25	75%
≥ 10	53,4%
≥ 11.25	50%
≥ 13.88	25%
≥ 15	20,9%

Forme et structure de l'oral

L'heure d'oral se décompose en deux parties.

Premièrement, un exposé dont la durée attendue est de 35 minutes. Celui-ci prend la forme d'un cours d'ouverture à destination d'élèves de niveau L1/L2 ou CPGE. Le cadre d'énonciation repose sur une triple fiction :

— le jury n'a pas connaissance du texte proposé à la candidate ;
— la candidate a trouvé ce texte toute seule lors de recherches pour préparer son exposé ;
— la candidate a formulé elle-même les pistes de réflexion qui sont proposées à la fin du texte (ou bien : ces pistes sont présentes dans les brouillons d'une collègue, et la candidate les reprend à son compte pour les développer).

On ne demande donc pas un exposé sur le texte, mais sur le sujet traité par le texte. Il n'y a donc pas lieu de citer le texte et encore moins de s'y référer. L'exposé doit être structuré et il est fortement conseillé de présenter son plan en début d'oral après une courte introduction. L'exposé doit également inclure la présentation argumentée d'une ou plusieurs illustrations sur ordinateur, ainsi que la discussion d'une dimension éthique, sociétale, environnementale, économique ou juridique telle que mentionnée par l'arrêté.

La deuxième partie de l'oral consiste en un échange avec le jury.

Contenu de l'exposé

Il est attendu des candidates d'une part une présentation et une discussion du problème et de sa formalisation, d'autre part une capacité à développer, compléter, voire améliorer (ou critiquer) les ébauches de solutions esquissées. Par exemple, lorsque l'on fait des études de complexité asymptotique d'algorithmes présentés pour être appliqués à des cas concrets, le jury attend que la candidate soit capable de raisonner en ordres de grandeur. Est-ce que la complexité de cet algorithme est un problème étant donné la taille attendue de l'entrée ? Jusqu'à quelle taille d'entrée cet algorithme peut-il raisonnablement être appliqué ?

Outre la capacité à mobiliser ses connaissances dans un contexte concret, le jury cherche également à évaluer la communication scientifique et pédagogique, c'est-à-dire la capacité de tenir un discours structuré, cohérent et pédagogique sur un sujet non trivial. Pour cela, la candidate doit construire un exposé fondé sur le texte fourni. Il ne s'agit pas de faire une lecture commentée du texte, mais bien de construire un exposé autonome. En particulier, les candidates ne doivent supposer du jury aucune connaissance préalable du texte. Le jury n'attend pas forcément un traitement exhaustif de tout ce qui est abordé dans le texte et dans les pistes, mais l'esprit du document doit être respecté.

L'exposé de la candidate doit prendre la forme d'un cours d'ouverture à destination d'élèves niveau L1/L2 ou CPGE. L'évaluation de l'utilisation du tableau est partie intégrante de l'évaluation des compétences didactiques et pédagogiques des candidates. Le jury attend que la candidate montre sa capacité à utiliser le tableau pour transmettre des notions et expliquer des exemples bien choisis. Le vidéo-projecteur n'est utilisé que ponctuellement durant la présentation. Il sert à

projeter l'illustration informatique et éventuellement une figure ou un algorithme. En revanche, il ne doit pas être utilisé comme support principal de l'exposé, pour projeter un plan, un diaporama, des définitions ou des énoncés de résultats.

Les illustrations informatiques doivent s'intégrer à l'exposé de manière cohérente et la plus fluide possible. L'exercice porte en particulier sur la communication, et on attend donc une présentation efficace d'une ou plusieurs illustrations qui éclairent réellement un aspect du texte. Il ne s'agit pas d'une seconde épreuve de TP ou d'un petit exercice de programmation vaguement dans le thème du sujet. Le jury sera plus sensible à la pertinence de l'illustration qu'à la complexité de l'implémentation, même s'il restera toujours attentif à la qualité du code présenté. Cette illustration doit être le moyen d'évaluer des solutions proposées dans le sujet voire d'en comparer plusieurs. Les jeux de tests et l'interprétation des résultats sont centraux dans cet exercice. La candidate peut choisir d'insister plus ou moins sur la programmation, voire de ne pas détailler le code pendant la présentation et de ne montrer que les résultats obtenus, par exemple sous forme de courbes de performance ou d'animation. À cet effet, la candidate peut librement utiliser des bibliothèques fournies dans le langage choisi. À l'inverse, un programme dont les tests ne contiennent qu'une suite de vérifications avec `assert` ne répond pas du tout aux attentes du jury. En cas de programmation, le jury attend que la candidate exécute ses programmes durant la présentation afin de vérifier leur fonctionnalité. Le jury a regretté que peu de candidates présentent des tests de performance.

Au delà de l'illustration informatique, le reste de l'exposé doit être équilibré et contenir des apports personnels autres que de la programmation, qu'il s'agisse d'une preuve, d'un algorithme, de l'architecture d'une solution matérielle ou logicielle, etc.

De même qu'à l'épreuve de leçon, le jury est attaché à la capacité des candidates à prendre du recul et envisager l'informatique comme une discipline et non comme un patchwork de domaines ; la capacité à s'appuyer sur le texte pour faire apparaître des liens dans d'autres sous-domaines que celui étudié par le texte constitue une vraie preuve de maturité scientifique. Insistons toutefois sur le fait que le discours doit rester dans le cadre du texte, et que ce dernier ne doit pas servir de simple prétexte à une longue digression dans un domaine où la candidate se sent plus à l'aise. Les candidats doivent également intégrer une discussion courte et synthétique d'une dimension éthique, sociétale, environnementale, économique ou juridique. Cette ouverture ne suit pas nécessairement une piste proposée par le jury, toute initiative personnelle pertinente ayant d'ailleurs vocation à être valorisée.

L'épreuve de modélisation nécessite une réflexion préalable des candidates sur les différents aspects de l'épreuve : construction d'un véritable exposé, utilisation du tableau, place de l'illustration informatique, conclusion du propos... On pourra, à cette fin, s'appuyer sur les exemples de sujets donnés aux sessions précédentes publiés sur le site `agreg-info.org`.

Questions

L'interrogation se poursuit ensuite par les questions du jury, qui peuvent porter sur l'ensemble de la présentation de la candidate, sur l'ensemble des sujets du programme se rattachant à celle-ci, ou encore sur les dimensions éthiques, sociétales, environnementales, économiques ou juridiques en lien avec le texte. Ce temps est l'occasion pour le jury d'affiner sa perception de la compréhension du texte par les candidates, de faire préciser ou corriger des points, de fournir des indications pour permettre à la candidate de développer certains aspects qui lui ont résisté, ou encore de proposer des pistes de réflexions, des prolongements, ou des variantes du problème étudié par le texte. Comme pour les séances de questions des autres épreuves, il est primordial que la candidate adopte une posture facilitant les échanges. Elle doit écouter les questions jusqu'au bout et répondre de façon concise et la plus précise possible. Bien que cela soit parfois nécessaire, il est difficile et désagréable pour le jury de devoir couper la parole d'une candidate qui s'est lancé dans une réponse très longue.

Remarques générales sur la session 2024

L'épreuve de modélisation de cette session reposait sur deux sujets portant sur des thèmes différents : le prétraitement algorithmique avec des applications variées et le routage efficace en énergie dans un réseau de capteurs sans fil. Les candidates ont plutôt bien compris le format de l'épreuve. Il reste tout de même quelques exceptions qui poussent le jury à rappeler des conseils d'ordre général.

— Il est important de présenter un plan de son exposé dans son introduction.
— L'épreuve de modélisation n'est pas une seconde épreuve de TP, les illustrations informatiques ne doivent pas représenter l'essentiel de la présentation. En revanche, elle doivent servir le propos, mettre en lumière des phénomènes et ne pas être considérées comme de simples exercices de programmation.
— La gestion du tableau est un enjeu important de l'épreuve. Il ne faut pas utiliser le tableau comme une feuille de brouillon, mais bien comme un support pédagogique.
— Les résultats énoncés dans le texte tels que les complexités de certains algorithmes ne doivent pas être énoncés dans l'exposé sans justification : soit ils sont prouvés par la candidate, soit ils sont explicitement admis.

Le jury a été satisfait du niveau global des candidates. Malgré tout, certaines maladresses et lacunes sont apparues à plusieurs reprises. Le jury attend une posture d'enseignante de la part des candidates. Elles doivent en particulier parler et écrire dans un français correct, en utilisant un vocabulaire précis et rigoureux. Par exemple, les termes "aléatoire" et "optimal" ont été utilisés de façon très maladroites. En particulier, l'optimalité n'est pas une échelle de valeur. On n'est pas plus ou moins optimal.

Remarques sur le sujet *Prétraitement algorithmique et applications*

Le premier sujet portait sur l'intérêt d'une étape de précalcul pour accéder efficacement au minimum des valeurs dans les tranches d'un tableau, puis sur l'utilité de ce problème en algorithmique du texte ou pour calculer des plus proches ancêtres communs dans un arbre. Certaines candidates ont fait des choix intéressants en centrant leurs exposés sur une des applications. Le jury a été sensible à ces prises d'initiative, mais il faut veiller à la cohérence du plan. Le passage d'une partie à une autre doit être soigné et montrer le recul pris par la candidate vis à vis du texte. Pour ce qui est de l'illustration informatique, le jury attendait naturellement qu'elle porte sur l'intérêt du précalcul d'une manière ou d'une autre : soit en comparant les différentes approches pour le calcul des minimums dans des tranches d'un tableau, soit en comparant des approches avec et sans précalculs pour les applications. L'implémentation de certains algorithmes sans qu'il n'y ait de discussion sur les performances observées n'avait que très peu d'intérêt dans cette épreuve.

Remarques sur le sujet *Routage efficace en énergie dans un réseau de capteurs sans fil*

Le second sujet portait sur le routage dans un réseau de capteurs sans fil en se concentrant sur l'économie des batteries des capteurs. Ce sujet proposait plusieurs solutions plus ou moins sophistiquées et plus ou moins détaillées. Il esquissait la comparaison de ces approches au travers de courbes tracées indiquant l'énergie totale restante ou le nombre de capteurs encore en marche. Le jury attendait des candidates qu'elles comparent plusieurs approches et qu'elles discutent les résultats, ainsi que la pertinence des différentes métriques utilisées (date où la première batterie se vide, date où la dernière batterie se vide, quantité d'énergie au fil du temps, quantité de capteurs en marche au fil du temps, couverture géographique au fil du temps). Comme pour le premier sujet, l'implémentation de certaines approches, sans étude des performances n'avait que très peu d'intérêt ici.

MINISTÈRE
DE L'ÉDUCATION
NATIONALE
ET DE LA JEUNESSE
Liberté
Égalité
Fraternité

Concours externe de l'agrégation du second degré

Section informatique

Liste des leçons pour la session 2025

La liste des sujets de leçons pour la session 2025 est la suivante :

1. Exemples de méthodes et outils pour la correction des programmes.
2. Paradigmes de programmation : impératif, fonctionnel, objet. Exemples et applications.
3. Tests de programme et inspection de code.
4. Principe d'induction.
5. Implémentations et applications des piles, files et files de priorité.
6. Implémentations et applications des ensembles et des dictionnaires.
7. Accessiblité et chemins dans un graphe. Applications.
8. Algorithmes de tri. Exemples, complexité et applications.
9. Algorithmique du texte. Exemples et applications.
10. Arbres : représentations et applications.
11. Algorithmes d'approximation et algorithmes probabilistes. Exemples et applications.
12. Algorithmes glouton et de retour sur trace. Exemples et applications.
13. Algorithmes utilisant la méthode « diviser pour régner ». Exemples et applications.
14. Programmation dynamique. Exemples et applications.
15. Algorithmes d'apprentissage supervisé et non supervisé. Exemples et applications.
16. Algorithmes pour l'étude des jeux. Exemples et applications.
17. Algorithmes d'ordonnancement de tâches et de gestion de ressources.
18. Gestion et coordination de multiples fils d'exécution.
19. Mémoire : du bit à l'abstraction vue par les processus.
20. Problèmes et stratégies de cohérence et de synchronisation.
21. Stockage et manipulation de données, des fichiers aux bases de données.
22. Fonctions et circuits booléens en architecture des ordinateurs.
23. Principes de fonctionnement des ordinateurs : architecture, notions d'assembleur.
24. Échanges de données et routage. Exemples.
25. Client-serveur : des sockets TCP aux requêtes HTTP.
26. Architecture d'Internet.
27. Modèle relationnel et conception de bases de données.
28. Requêtes en langage SQL.
29. Langages rationnels et automates finis. Exemples et applications.
30. Grammaires hors-contexte. Applications à l'analyse syntaxique.
31. Classes P et NP. Problèmes NP-complets. Exemples.
32. Décidabilité et indécidabilité. Exemples.
33. Formules du calcul propositionnel : représentation, formes normales, satisfiabilité. Applications.