MINISTÈRES
ÉDUCATION
JEUNESSE
SPORTS
ENSEIGNEMENT
SUPÉRIEUR
RECHERCHE

Liberté
Égalité
Fraternité

Rapport du jury

Concours : agrégation externe

Section : Informatique

Session 2025

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 quatrième édition de l'agrégation externe d'informatique, qui s'est déroulée en 2025. 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 2025, les **épreuves écrites** se sont déroulées du 10 au 12 mars, organisées par les académies :

- Composition d'informatique : 10 mars 2025 de 9 heures à 14 heures,
- Étude d'un problème informatique : 11 mars 2025 de 9 heures à 15 heures,
- Épreuve spécifique (selon l'option choisie : étude de cas informatique ou fondements de l'informatique) : 12 mars 2025 de 9 heures à 15 heures.

Les **épreuves orales** se sont déroulées du 17 au 22 juin 2025 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`

Les modalités du concours évoluent à partir de la session **2026**, conformément à l'arrêté du 19 septembre 2025 [2] :

— la troisième épreuve est supprimée ;
— les deux épreuves d'admissibilité et les trois épreuves d'admission ont le même coefficient 1 ;
— un unique programme, publié chaque année sur le site `https://www.devenirenseignant.gouv.fr/` s'applique à l'ensemble des épreuves.

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 2025 et comme les deux années précédentes, 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)
429	168	345	121

Admissibles	Admis liste principale	Liste complémentaire
45	22	0

Le chiffre des inscrits et des présents est en légère baisse par rapport à 2024. Malgré cette baisse, l'agrégation d'informatique reste une section très sélective du concours de l'agrégation externe, avec 5,5 candidats présents pour un poste.

Pour ce qui concerne les épreuves écrites, la moyenne des candidats admissibles est de 12,08/20, avec une **barre d'admissibilité à 7,79/20**.

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 10,00/20**. 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

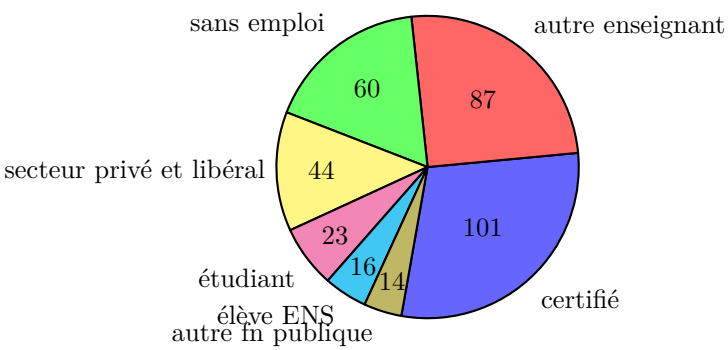
Il est important de rappeler que ce tableau se base sur les statuts indiqués, de manière purement déclarative, par les candidates et candidats lors de l'inscription. Aucune vérification n'est opérée sur ce point, qui fait uniquement l'objet de la présente exploitation statistique.

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.

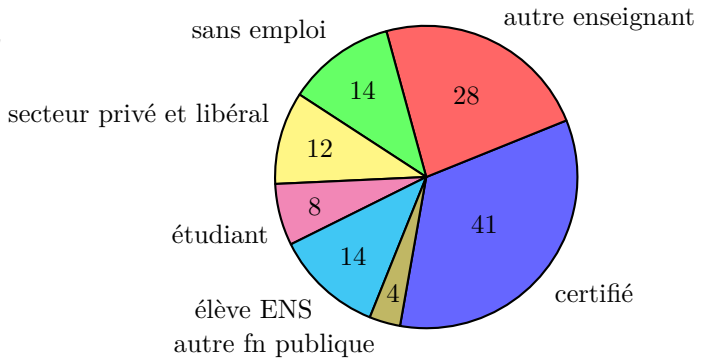
2. `https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000052259965`

inscrits	présents	admissibles	admis	profession
101	41	12	2	Certifié
60	14	3	2	Sans emploi
31	11	2	0	Cadres secteur privé convention collective
16	14	13	12	Élève d'une ENS
14	1	0	0	Étud. hors inspe (sans prépa)
13	3	2	1	Contractuel enseignant supérieur
13	1	1	1	Contractuel 2nd degré
10	4	3	0	Agrégé
8	1	0	0	Professeur associé 2nd degré
7	6	4	2	Étud. hors inspe (prépa mo.univ)
7	5	1	0	Ens. stagiaire 2e deg. col/lyc
6	2	0	0	Personnel de la fonction publique
6	1	0	0	Professeur des écoles
5	4	1	0	Enseignant du supérieur
5	1	0	0	Salariés secteur tertiaire
5	0	0	0	PLP
4	3	1	1	Maître délégué
4	2	0	0	Maître contr. et agréé rem tit
4	0	0	0	Salariés secteur industriel
4	0	0	0	Professions libérales
3	2	0	0	Personnel enseignant titulaire fonction publique
3	1	1	0	Agent non titulaire fonction publique
2	1	0	0	Étudiant en inspe en 2eme année
2	1	0	0	Vacataire du 2nd degré
2	0	0	0	Formateurs dans secteur privé
2	0	0	0	Vacataire enseignant du sup.
1	1	1	1	Personnel enseignant non titulaire fonction publique
1	1	0	0	Agent non titulaire de la fonction territoriale
1	0	0	0	Accompagnant des élèves en situation de handicap (AESH)
1	0	0	0	Assistant d'éducation
1	0	0	0	Contractuel MEN Administratif ou technique
1	0	0	0	Emploi avenir prof. 2nd d. publique
1	0	0	0	Militaire
1	0	0	0	PEPS

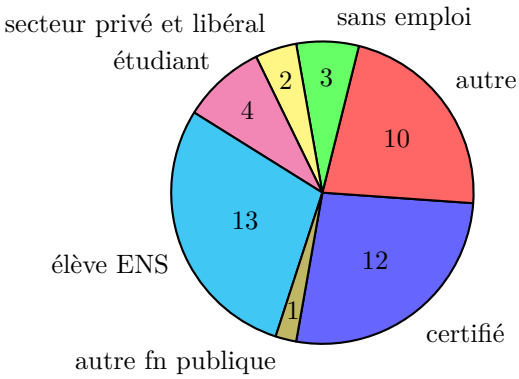
TABLE 1.1 – Répartition des candidats par profession.



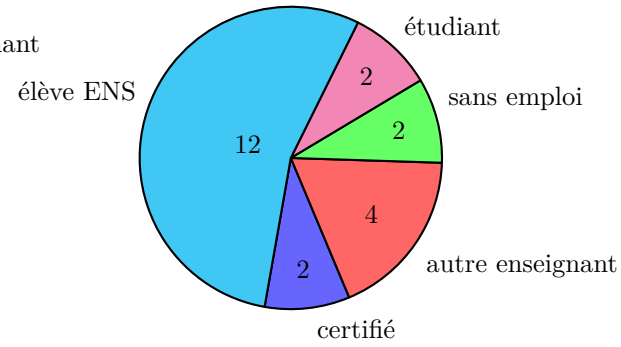
Inscrits par statut.



Présents par statut.



Admissibles par statut.

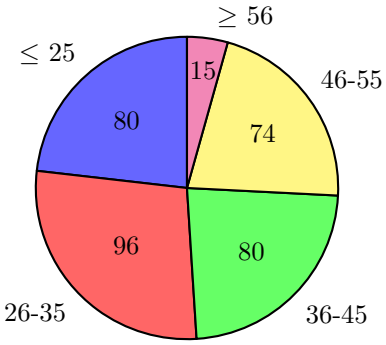


Admis par statut.

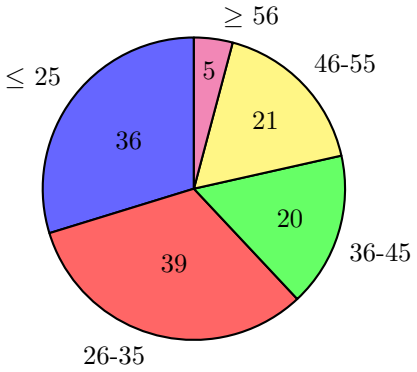
1.2.2 Statistiques par âge

Âge	inscrits	présents aux écrits	admissibles	admis
≤ 25	80	36	21	14
26-35	96	39	18	8
36-45	80	20	3	0
46-55	74	21	3	0
≥ 56	15	5	0	0

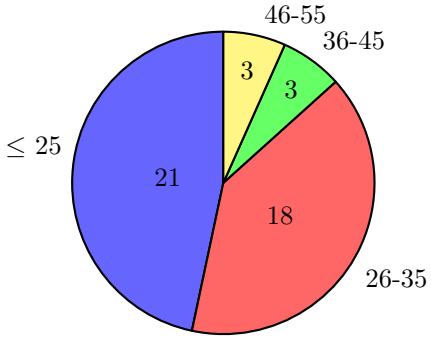
TABLE 1.2 – Répartition des candidats par âge.



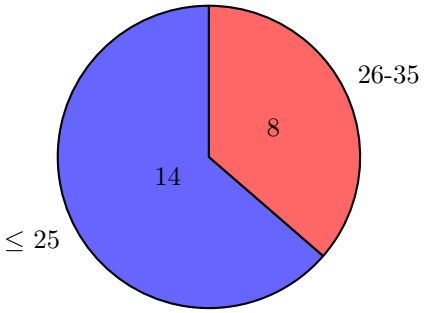
Inscrits par âge.



Présents par âge.



Admissibles par âge.



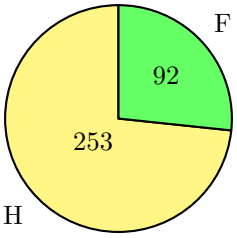
Admis par âge.

1.2.3 Statistiques par genre

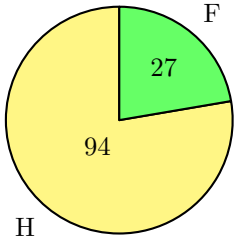
Rappelons que le genre, également déclaratif, ne permet à l'inscription que les deux choix M. ou Mme.

inscrits	présents aux écrits	admissibles	admis	genre
253	94	41	20	M
92	27	4	2	F

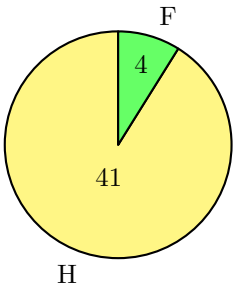
TABLE 1.3 – Répartition des candidats par genre.



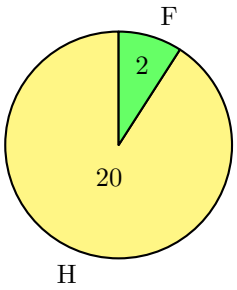
Inscrits par genre.



Présents par genre.



Admissibles par genre.

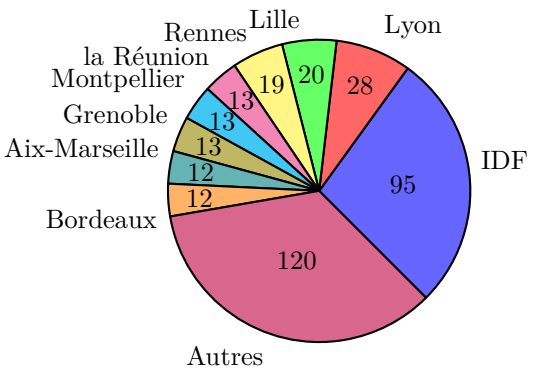


Admis par genre.

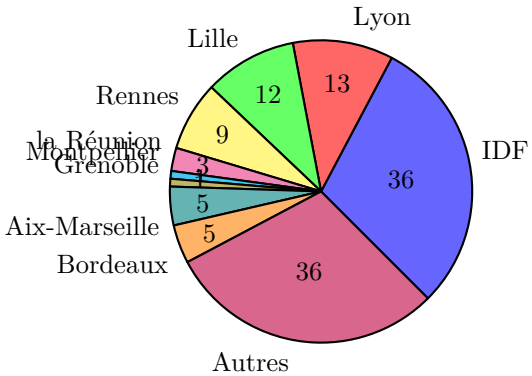
La représentation des femmes en sciences et en informatique en particulier reste très faible. Le taux de femmes inscrites est d'environ 26,7%), les femmes se présentent en moindre proportion (22,3%), et elles ont plutôt moins bien réussi les épreuves écrites (sachant que les copies sont anonymes). Pour 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 encore et toujours une préoccupation du jury.

1.2.4 Statistiques par académie

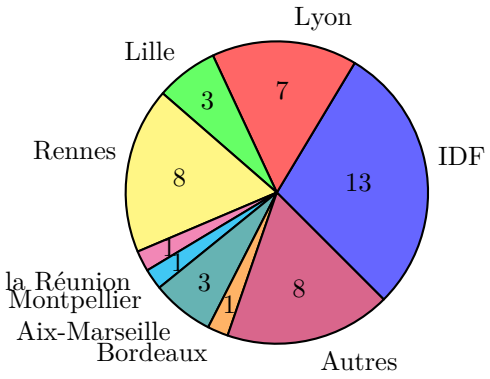
Rappelons que les seules académies (à ce jour) à accueillir une préparation à l'agrégation externe d'informatique sont les académies de Créteil-Paris-Versailles, Lyon et Rennes, même si la première préparation peut se suivre partiellement à distance.



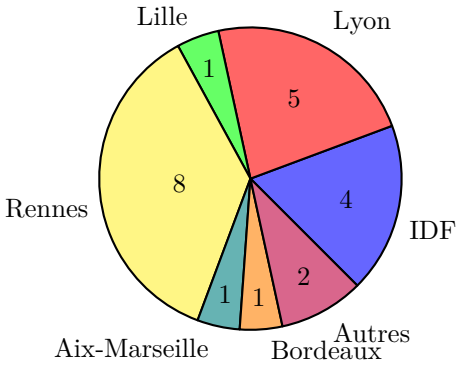
Inscrits par académie.



Présents par académie.



Admissibles par académie.



Admis par académie.

inscrits	présents aux écrits	admissibles	admis	académie
95	36	13	4	Académies de Créteil Paris Versailles
28	13	7	5	Académie de Lyon
28	1	0	0	Hors académie
20	12	3	1	Académie de Lille
19	9	8	8	Académie de Rennes
13	3	1	0	Académie de la Réunion
13	1	1	0	Académie de Montpellier
13	1	0	0	Académie de Grenoble
12	5	3	1	Académie d'Aix Marseille
12	5	1	1	Académie de Bordeaux
11	2	0	0	Académie de Nice
10	5	1	1	Académie de Toulouse
9	5	1	0	Académie de Normandie
8	4	0	0	Académie de Poitiers
8	3	2	0	Académie de Nancy-Metz
8	2	0	0	Académie d'Amiens
8	2	0	0	Académie d'Orléans-Tours
7	5	3	1	Académie de Nantes
5	4	1	0	Académie de Strasbourg
3	0	0	0	Académie de la Nouvelle Calédonie
2	1	0	0	Académie de Corse
2	1	0	0	Académie de la Guadeloupe
2	0	0	0	Académie de la Guyane
2	0	0	0	Académie de Mayotte
2	0	0	0	Académie de Reims
1	1	0	0	Académie de la Martinique
1	0	0	0	Académie de Besancon
1	0	0	0	Académie de Clermont-ferrand
1	0	0	0	Académie de la Polynésie Francaise
1	0	0	0	Académie de Limoges

TABLE 1.4 – Répartition des candidats par académie.

1.3 Remerciements

Le jury du concours 2025 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 2025 se sont tenues pour la deuxième 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 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.

En cette dernière année de présidence, la présidente tient à remercier l'ensemble des jurys des quatre dernières années pour leur énorme travail à créer et animer cette agrégation. Un merci spécial aux merveilleux responsables informatiques et aux différents directoires qui ont géré les imprévus avec efficacité et bonne humeur.

Chapitre 2

Épreuves écrites

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.

L'architecture générale de l'écrit de l'agrégation externe d'informatique 2025 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 *informaticiennes complètes*, 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'une candidate aux fonctions de professeure qu'elle sache rédiger une copie. En conséquence, un passage rayé ou hachuré ne sera jamais lu. De plus, quand plusieurs versions sont proposées par la candidate, la correctrice ne prend pas la meilleure. C'est à la candidate de choisir ce qu'elle 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 jusqu'à la correctrice.

Une préoccupation du jury est de valoriser les candidates précises 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 candidates solides et précises sur les bases de la discipline, afin de pouvoir transmettre ces dernières.

Il est attendu, pour l'écrit comme pour l'oral, que les candidates 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 une future agrégée 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 masculin.
2. `https://www.legifrance.gouv.fr/loda/id/LEGIARTI000046287651/2022-09-01/#LEGIARTI000046287651`

2.1 Épreuve 1

Les trois parties de ce sujet couvrent, sous des angles disjoints, des problématiques au cœur de la réalisation d'un éditeur de texte.

Cette épreuve était composée de trois parties, balayant largement plusieurs domaines de l'informatique : système, structure de donnée et algorithmique du texte et plusieurs langages : C, OCaml et Python.

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

La lisibilité des copies reste un problème. On rappelle aux candidates que leurs copies sont faites pour être lues par d'autres personnes. Si la réponse à une question est incompréhensible car la correctrice ne peut deviner quels sont les mots écrits, aucun point ne sera accordé.

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

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 candidates ayant une note supérieure à 5, 10 et 15.

— 131 présents
— Meilleure note : 17.98/20
— Moyenne : 6.19 ; écart-type : 4.3.

≥ 2.57	75,0%
≥ 5	55,7%
≥ 5.49	50,0%
≥ 8.93	25,0%
≥ 10	19,8%
≥ 15	3,8%

2.1.1 Partie I : gestion de fichiers dans un éditeur de texte

L'exercice 1 visait à évaluer les candidates sur leurs compétences de programmation en langage C. Bien que ce langage soit au programme du concours, la correction des copies a permis de constater qu'il insuffisamment maîtrisé par la majorité des candidates. Les copies ont notamment mis en évidence de nombreuses confusions, non seulement entre les paradigmes de programmation (procédurale, fonctionnelle, objet), mais aussi entre les syntaxes et constructions propres à chaque langage (par exemple, structure de contrôle, allocation de ressources). De plus, il était attendu que les candidates maîtrisent les types propres au langage, ce qui inclut les chaînes de caractères, ainsi que la notion de pointeur et toutes les primitives associées (allocation/libération des ressources, adressage/déréférencement du contenu, passage par valeur/référence, types de pointeurs quelconques). De nombreuses erreurs et confusions ont également pu être constatées entre les opérateurs bit-à-bit, les opérateurs booléens et même les opérateurs arithmétiques.

Afin de maîtriser les exécutions des différentes procédures et fonctions décrites dans le sujet, il est important de s'assurer que pré-conditions et post-conditions d'appels sont respectées, notamment quand celles-ci étaient formulées dans le sujet. Par ailleurs, comme il était indiqué en préambule du sujet, toute proposition de code doit nécessairement être accompagnée d'un commentaire de code pertinent.

Enfin, il est attendu que les réponses données traitent une question à la fois et non une réponse pour un ensemble de questions. De même, le fait de répondre de différentes manières à une même question ne saurait avantager la candidate.

Question 1.1 Cette question abordait les notions élémentaires du langage C et comprenait à la fois des erreurs de syntaxe qui compromettaient la bonne compilation du code (e.g., problèmes de signature des fonctions), des erreurs pouvant conduire à des bugs (e.g., absence de vérification des pré-conditions) et des erreurs d'incohérence par rapport au sujet (utilisation d'un paramètre erroné). Peu de candidates ont su identifier la diversité de ces erreurs.

Question 1.2 Cette question permettait aux candidates de proposer des corrections qui devaient altérer uniquement les portions de code problématiques. Une majorité de candidates ont pu proposer des corrections pour plusieurs erreurs relevées mais peu d'entre elles ont su couvrir toutes les erreurs. Certaines candidates ont proposé des corrections à des erreurs (pertinentes) qui n'avaient pas été identifiées dans la question 1.1.

Question 1.3 Cette question visait à cerner l'utilité de la primitive `mmap(…)` dans le cadre du sujet. De nombreuses candidates ont essentiellement recopié des éléments de la documentation (en lien avec des problématiques d'accès concurrent qui n'étaient pas mentionnées dans le sujet) sans nécessairement considérer les apports de la primitive `mmap(…)` pour le besoin explicité dans le sujet d'une manipulation de très gros fichiers.

Question 1.4 Cette question avait pour objectif de s'approprier la signature de la primitive `mmap(…)` pour en proposer une utilisation dans le cadre du sujet. Il était notamment important que les candidates s'assurent qu'une erreur de projection soit prise en compte et ne génère pas d'effet de bord pour l'application.

Question 1.5 Cette question se concentrait sur la manipulation des chaînes de caractères en langage C et devait permettre aux candidates de démontrer leur capacité à allouer des chaînes de caractères de taille variable, comparer et chercher des chaînes de caractères, et remplacer proprement du texte. Beaucoup de candidates ne semblent pas maîtriser les primitives essentielles à ces opérations.

Question 1.6 Cette question devait faire réfléchir les candidates quant aux risques liés à la simple réutilisation d'une fonction existante pour mettre en oeuvre une nouvelle fonction. Dans le cas présent, la fonction existante peut s'avérer coûteuse en terme de performances, de par les écritures possibles sur le disque. Par ailleurs, les candidates doivent être vigilantes à répondre exactement à la question posée et ne pas reformuler la question pour leur permettre d'apporter une autre réponse que celle attendue.

Question 1.7 Cette question se concentrait la mise en oeuvre d'un décalage à droite des caractères d'un texte pour y insérer de nouveaux caractères. Il était attendu que la stratégie d'allocation des ressources soit gérée par la méthode ou *a minima* documentée (via une pré-condition, par exemple).

Question 1.8 Cette question concernait la définition d'une structure de données pour capturer une modification opérée sur un texte quelconque. *A minima*, la réponse à la question devait démontrer la capacité des candidates à proposer un nouveau type avec la construction `struct`, tandis que l'utilisation d'une énumération pouvait être utile pour distinguer la nature des modifications.

Question 1.9 Dans cette question, il était attendu que les candidates proposent une implémentation de la structure historique (ici une pile) avec des opérations d'ajout et suppression des modifications. Comme indiqué dans l'énoncé, la simple déclaration d'une structure de donnée n'était donc pas suffisante pour répondre à la question.

Question 1.10 Cette question revenait sur le code de la question 5 pour en proposer une évolution capable de tracer les modifications opérées en empilant les changement réalisés dans l'historique.

Question 1.11 Cette dernière question couvrait la problématique de l'annulation des modifications et consistait, après s'être assuré de la validité des paramètres, à dépiler les `n` dernières modifications d'un historique pour annuler une insertion ou un remplacement selon le type de la modification dépilée. Les réponses utilisant la récursivité ou l'itération étaient acceptées.

2.1.2 Partie II : structure de corde

Dans cette partie, on étudie la structure de données de corde, qui est une structure arborescente constituant une alternative efficace aux chaînes de caractères pour l'édition de grands textes. Dans cette structure, l'accent est porté sur les insertions et suppressions de sous-chaînes avec en contrepartie un surcoût pour l'accès aux caractères individuels. On s'intéresse également à la persistance des données et à la gestion d'un historique des modifications.

Cette partie contient principalement des questions de programmation en OCaml qui ne reposent que sur des constructions simples du langage. On peut noter que si les candidates sont encore trop nombreuses à éviter la programmation en OCaml, de nombreuses copies témoignent d'une maîtrise satisfaisante de la programmation fonctionnelle en OCaml. Les questions de fin de partie, reposant sur les constructions impératives du langage, ont en revanche été moins bien traitées.

Question 2.1 Indiquer qu'une `string` est non mutable et que cela nécessite de créer de nouvelles valeurs n'est pas suffisant pour justifier les coûts car c'est aussi le cas d'une `rope`. Indiquer que les opérations d'insertion nécessitent des décalages en fin de chaîne revient à ignorer les considérations mémoires aboutissant à des recopies complètes.

Question 2.2 Il ne s'agissait pas de justifier la pertinence de la structure de corde, mais d'une implémentation persistante de cette structure. La notion de persistance pour une structure de données n'est pas maîtrisée. Il s'agit pourtant d'un concept clé en `OCaml`.

Le jury attend des candidates des réponses précises sur ces questions. Beaucoup se contentent de paraphraser la question sans donner d'arguments précis.

Question 2.3 Cette première question a pour but de comprendre la structure et ses particularités, ici le fait que longueur et hauteur soient présents directement dans les constructeurs du type `rope`. Il était donc crucial de proposer une implémentation en temps constant.

Question 2.4 Cette question demandait d'effectuer un parcours récursif de la structure et ne présentait pas de difficulté. Cependant, de nombreuses candidates ignorent le décalage d'indices dans le cas d'un parcours dans le sous-arbre droit.

Question 2.5 Lever une exception dans les cas où la concaténation de deux feuilles excéderait la longueur `max_string_length` est une erreur ; la concaténation est toujours possible via la création d'un nœud `Concat`.

Question 2.6 et 2.7 Ces deux questions sont de simples utilisation de la fonction `split`. Cette fonction renvoyant un couple, la déconstruction de celui-ci a été un obstacle majeur pour un nombre significatif de candidates.

Question 2.8 L'écriture de la fonction `split` était une question délicate qui a été cependant convenablement traité par les candidates ayant abordé la question. Des erreurs de décalage d'indice de même nature que celles de la question 2.4 ont cependant été observées alors que la logique de la fonction était correcte par ailleurs.

Question 2.9, 2.10 et 2.11 La pratique des types structurés natifs, `'a ref` et `'a list` ici, posent des problèmes à de nombreuses candidates. La contrainte d'avoir une pile `redo` non vide à tout instant a souvent été oubliée. Ces questions ont été peu abordées, mais elles ne présentaient pas de difficultés pour des candidates sachant manipuler ces types.

2.1.3 Partie III : recherche de motif

Cette partie traite de la recherche de motif dans un texte. La première moitié s'intéresse à une variante simplifiée de l'algorithme de Boyer-Moore-Horspool, sans table de saut, et contient principalement des questions algorithmiques. La seconde moitié se concentre sur la construction classique d'un automate qui reconnaît les mots se terminant par le motif, et contenait plusieurs preuves pour en établir la correction.

Globalement, les candidates n'ont pas très bien réussi cet exercice. Il y a eu de nombreuses erreurs sur la première partie, même sur les questions élémentaires, et la seconde a été très peu traitée.

Les correctrices ne souhaitent plus voir énoncer des complexités avec des nombres : pour $n = 10$ et $m = 4$ la complexité est en $O(40)$.

La syntaxe Python était plutôt maîtrisée. En revanche, les correctrices ont vu de nombreuses imprécisions algorithmiques. Dans un sujet où on cherche à être plus efficace dans des cas favorables, appeler trois fois la même fonction plutôt que d'en stocker le résultat est particulièrement maladroit, même si cela ne change pas la complexité en terme de $O()$.

Question 3.1 De nombreuses candidates ne traitent pas le cas de débordement de fin de texte quand i est trop grand.

Question 3.2 Il faut justifier un minimum les résultats de complexité annoncés. Les correctrices s'attendent également à ce que le résultat soit simplifié, et pas sous la forme $O((n-m+1)m)$. Enfin, l'hypothèse que $n \gg m$ ne revient pas à dire que m est constant : on peut simplifier $O(n-m)$ en $O(n)$ sous cette hypothèse, mais pas $O(nm)$ en $O(n)$. Pour montrer que la borne est atteinte, on ne pouvait pas se contenter d'un exemple à n et m fixés, comme l'on fait de nombreuses candidates.

Question 3.3 De nombreuses candidates n'ont pas implanté la consigne de renvoyer -1 quand le motif n'est pas présent. Il était plus efficace de parcourir la plage du texte considérée dans le bon sens, et de s'arrêter dès que possible.

Question 3.4 La question a été mal traitée : sauts sont incorrects, boucles infinies, multiples appels à difference_position() et occurrence_position() pour la même position i, *etc.* L'utilisation d'un *set* pour les lettres du motif permettait d'optimiser l'heuristique de saut.

Question 3.5 Cette question a été mal traitée. Il fallait proposer des exemples de meilleurs cas pour tout n et tout m, comme indiqué dans l'énoncé.

Question 3.6 Les candidates qui ont compris la construction, ou appliqué la définition pas à pas, ont correctement traité cette question.

Question 3.7 Très peu de candidates ont un programme avec la bonne complexité. La plupart des solutions proposées étaient en $O(|u|^2)$, souvent en raison de l'utilisation répétée de *slices* Python.

Question 3.8 et Question 3.9 Très peu traitées. Les candidates qui ont essayé ont globalement bien pensé à le faire par récurrence en s'appuyant sur les définitions.

Question 3.10 Comme indiqué dans l'énoncé, il fallait faire attention à l'alphabet utilisé : on pouvait soit faire l'union des alphabets du texte et du motif, soit ne prendre que l'alphabet du motif et traiter séparément les autres lettres lors du parcours dans l'automate. Il fallait faire attention aux indices des occurrences trouvées : la recherche par automates donne les indices de fin d'occurrence et non de début.

2.2 Épreuve 2 : Mastermind

— 125 présents
— Meilleure note : 20.0/20
— Moyenne : 6.99 ; écart-type : 5.02.

≥ 2.83	75,0%
≥ 5	60,0%
≥ 6.29	50,0%
≥ 10	25,0%
≥ 15	8,8%

Le sujet portait sur la recherche de stratégies pour le jeu du Mastermind. La première partie introduisait le jeu et était axée sur la programmation en Python de fonctions simples ainsi que sur l'analyse de leur complexité. La deuxième partie introduisait formellement les notions d'arbre de stratégie et de stratégies optimales. Elle contenait des questions théoriques appelant des preuves rigoureuses ainsi que des questions de programmation en OCaml. La troisième partie explorait trois moyens d'accélérer la recherche de « bonnes » stratégies : un pré-calcul des similarités entre combinaisons (qu'il fallait programmer en C), une stratégie gloutonne non optimale mais moins coûteuse à calculer (qu'il fallait étudier théoriquement et programmer en OCaml), et l'élagage alpha-beta, pour lequel on demandait d'analyser un pseudo-code fourni. La quatrième partie, peu abordée, portait sur l'exploitation des symétries pour limiter l'espace de recherche. Les questions de programmation qu'elle contenait étaient à traiter en OCaml.

Le sujet contenait malheureusement deux erreurs : une dans la légende de la figure 2, qui a été signalée par de nombreuses candidates, et un u_i transformé en v_i dans la question 29.

Remarques générales

Certaines copies pèchent par leur présentation : orthographe, ratures, écriture illisible, etc., et parfois tout cela en même temps. On rappelle qu'il s'agit là d'un concours pour recruter des enseignantes, ayant vocation à être mis devant des élèves. Pour cette raison, on s'attend à un minimum de clarté dans les copies, tant dans la forme que dans le fond.

De la même manière, les candidates futures enseignantes doivent appliquer les bonnes pratiques de programmation qu'elles auront à enseigner et fournir un code clair, où l'articulation du raisonnement sous-jacent est facilement compréhensible. Cela passe entre autres par le choix du nom des variables et fonctions auxiliaires, et la bonne décomposition du code. On évitera par exemple les imbrications excessives qui rendent le raisonnement difficile à suivre.

Pour la lisibilité de la copie, il faut éviter les réponses écrites dans le pied de page tout en bas de la copie, coupées par le scan et également faire attention aux erreurs dans les numéros de question.

Les correctrices ne souhaitent plus voir écrit ce genre de phrases : "La complexité est linéaire par intervalle." ou "On peut représenter ces nombres sur 24 et 48 bits. Je propose donc une représentation de ces nombres par des tableaux [d'entiers] de 24 et 48 cases."

Les correctrices déplorent les nombreuses erreurs algorithmiques trouvées dans les copies. De plus, elles regrettent l'usage de tests alors qu'une fonction `min` existe. Elles rappellent également que la complexité temporelle de `l.delete()`, où `l` est une liste, n'est pas constante.

Voici quelques remarques de programmation, selon le langage.

En Python :

— Le langage Python permet de sortir prématurément d'une boucle, avec `break` ou `return`. Il y avait dans ce sujet de bonnes opportunités d'en faire usage.
— Plusieurs copies utilisent `=` comme opérateur d'égalité.
— De nombreuses copies contiennent des parcours de listes rang par rang dont les itérations suppriment des éléments de la liste, causant des erreurs d'accès hors bornes ultérieures.

En OCaml :

— Trop de copies estiment que `match e with v -> ...` est équivalent à `if e = v then ...` pour une variable `v` existante.
— Il est inutile d'écrire `if ... then true else false`.
— En l'absence de contraintes de complexité en espace, on peut éviter les fonctions auxiliaires avec passage d'accumulateur, dans la mesure où elles rendent généralement le code moins naturel à lire. Beaucoup de copies y ont systématiquement recours.
— L'utilisation de fonctions d'ordre supérieur imbriquées (en particulier `List.fold_left`) rend le code difficile à lire. Mieux vaut dans ce cas commencer par définir la fonction « interne » en lui donnant un nom approprié.

En C :

— L'opérateur binaire `^` existe bien dans le langage C, mais il ne calcule pas l'exponentiation entière comme le pensent certaines copies. Il y a bien une fonction `pow` en C, mais elle est définie sur le type `double`, pas sur le type `int`. Il n'y a pas d'opérateur `**` en C.
— Il n'y a pas de fonction `length` ou `len` en C qui donnerait la taille d'un tableau.
— Une fonction ne doit pas renvoyer un pointeur vers un objet qu'elle a alloué sur la pile.

Remarques par question

Ne sont détaillées ci-dessous que les questions qui méritent un commentaire.

Question 3 Si la réponse à question 2 contient la réponse à question 3, telle quelle, il est inutile de la réécrire. Beaucoup de copies ne se servent pas de la question 2. Des copies oublient de parler de la complexité. De manière générale, les réponses aux questions 2 et 3 étaient souvent bien trop compliquées (et donc souvent fausses).

Question 4 Il est dommage de voir des copies avec un `return True` dans la boucle et un `return False` à la sortie, c'est-à-dire une fonction qui renvoie `True` à la première vérification passée avec succès.

Question 5 Nombre de copies commencent par tester si la combinaison ne contient que des $P-1$, rendant la solution forcément inefficace.

Question 6 Quelques copies identifient bien le cas d'un élément passant de $P-2$ à $P-1$ (qui justifie une majoration par 2). Beaucoup d'autres proposent des majorations formellement justes mais sans intérêt.

Question 7 Beaucoup de copies identifient correctement une notion de complexité amortie, mais peu parviennent à en donner une démonstration rigoureuse. Un nombre non négligeable d'autres copies contiennent des raisonnements fantaisistes qui donnent l'impression de chercher à produire des mot-clefs (“la complexité est exponentielle”).

Question 8 Peu de copies pensent à faire une copie de la combinaison au moment de l'ajouter dans l'historique. Un certain nombre ne vérifient pas la compatibilité, ce qui suggère une mécompréhension globale de cette partie du sujet.

Question 9 On s'attendait ici à une preuve soignée, par récurrence. L'argument que les sous-arbres constituent une partition est rarement invoqué pour justifier proprement que le nombre de feuilles est égal à $|D|$.

Question 11 Un bon nombre de copies trouvent un arbre optimal, mais très peu arrivent à justifier son caractère optimal de manière satisfaisante. Le plus simple était de raisonner sur l'arité maximale des nœuds, qui correspondait au nombre de réponses possibles.

Question 13 Cette question était délicate mais quelques copies l'ont traitée parfaitement, en trouvant un ensemble D pour lequel l'égalité était fausse.

Question 15 Certaines copies n'utilisent pas la structure de D-arbre pour orienter la recherche (via le choix du sous-arbre étiqueté avec la bonne similarité), mais procèdent par exploration exhaustive. D'autres utilisent -1 comme valeur d'erreur, ce qui est plutôt inélégant (et tend à faire structurer le code de façon inefficace).

Question 16 Beaucoup de copies semblent ignorer l'existence des fonctions `min` et `max`. Très peu de copies calculent correctement le poids (qui était la somme des profondeurs des feuilles).

Question 17 Beaucoup de copies sautent la terminaison ; beaucoup ne rédigent pas proprement les récurrences (par exemple n'annoncent pas procéder par récurrence). Peu de candidates ont compris que la terminaison n'était assurée que parce qu'on limitait le minimum aux combinaisons de $split(X)$.

Question 18 Peu de copies ajoutent correctement $|X|$; quasiment aucune copie ne pense à exclure la réponse $(N, 0)$ de la somme.

Question 20 Question traitée par très peu de candidates.

Question 21 Il fallait penser à exclure la réponse $(N-1, 1)$. Les copies qui tentent une preuve par récurrence pour cette question échouent systématiquement (même si cela restait possible). Un certain nombre de copies partent d'une somme fausse et arrivent miraculeusement au résultat demandé.

Question 22 Certaines réponses étaient vraiment incorrectes. Par exemple, 88 n'est pas égal à 232. De plus, `short short` n'est pas un type de C, pas plus que `int_4` ou `int_1`.

Question 23 De trop rares copies semblent connaître le schéma de Horner. Par ailleurs, on rappelle qu'il n'y a pas d'exponentiation entière en C (que ce soit un opérateur du C ou une fonction de bibliothèque) ; mais il n'y en avait pas besoin pour cette question. L'évaluation en base P n'était d'ailleurs pas la seule option ; on pouvait également coder chaque couleur sur trois bits (mais il fallait alors être très soigneux pour la question suivante). Par ailleurs, beaucoup de copies fixent P à 10 sans raison apparente.

Question 28 L'écrasante majorité des copies ne cherche pas à justifier son affirmation par une démonstration.

Question 33 à 39 : Les candidates ayant traité ces questions semblaient découvrir l'élagage alpha-beta, pourtant au programme. Certaines ont fait le lien (pertinent) avec *branch-and-bound*.

2.3 Épreuve 3

2.3.1 Étude de cas informatique

— 86 présents
— Meilleure note : 17.0/20
— Moyenne : 6.78 ; écart-type : 4.02.

≥ 3.99	75,0%
≥ 5	62,7%
≥ 6.25	50,0%
≥ 9.49	25,0%
≥ 10	23,2%
≥ 15	4,6%

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 holistique de ce qu'est un système logiciel.

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.

Le jury rappelle qu'il ne faut jamais proposer plusieurs solutions : les correctrices n'ont pas à choisir quelle est la bonne. De plus, c'est en général une mauvaise idée de contester chaque question du sujet et de se proposer de répondre à une autre question.

Les correctrices ne souhaitent plus lire l'expression *on assumera* dans le sens de *on supposera*, ce qui est un anglicisme. De même, en français, on parle de *lever une exception* et non pas *raise une exception*.

Les correctrices ont lu dans une copie : "Si la fonction de hachage est bien construite, c'est-à-dire si elle est bijective…".

Remarques par question Pour toutes les questions de la partie I, il n'était pas demandé et donc pas attendu de fournir des tests pour le différentes implantations.

Question 1 De nombreux arguments pouvaient être mis en avant. Le jury s'attendait à lire le terme *automatisation*.

Question 2 À tort, certaines copies n'ont pas dessiné de diagramme. La question invitait explicitement à considérer des classes supplémentaires : il était donc attendu qu'une factorisation soit faite ou que son absence soit motivée. Il était attendu que soient représentées les relations d'agrégation, y compris celle entre la classe *Work* et elle-même.

Des réponses qui indiquent des attributs de classes (des listes) ne représentent pas complètement le type de relations entre entités.

Question 3 Une réponse complète s'appuie sur les notions de polymorphisme, héritage et surcharge. Certaines copies évoquaient à tort la notion d'encapsulation.

Question 4 Dans cette question, seule une première implémentation initiale avec la méthode *run* était attendue. Certaines copies ont souhaité résoudre, dès cette question, l'ensemble des problématiques traitées dans la partie I. Il est recommandé de survoler rapidement l'ensemble des questions d'une même partie pour éviter ce type d'écueil. Les copies qui ont fourni des méthodes non requises (typiquement des *getters*/*setters* d'attributs) n'ont été ni pénalisées, ni favorisées.

Des copies ont défini l'attribut contenant la liste d'actions comme un attribut de classe ce qui est incorrect : chaque instance de `Work` a sa propre liste d'actions à exécuter.

Des copies ont choisi de détruire la liste des actions au fur et à mesure de leur exécution, ce qui est en contradiction avec l'idée d'une chaîne CI/CD qui est exécutée plusieurs fois. Des copies ont voulu utiliser la structure de `set` pour stocker les actions d'un travail. Cette structure est fautive en ce qu'elle ne permet pas de conserver l'ordre entre actions.

Question 5 Il fallait créer un fil d'exécution par travail. De nombreuses copies mélangent `.start()` et `.run()`. Dans la méthode `run`, il était requis d'avoir les trois phases de création, lancement et **attente de terminaison** des threads.

Question 6 Cette question était sans difficulté et permettait d'avancer dans le sujet. Certaines copies l'avaient anticipée dès la question 4.

Question 7 Dans cette question les candidates devaient introduire un dictionnaire faisant correspondre un sémaphore à un travail. Certaines copies ont préféré utiliser une liste avec les identifiants de travaux ce qui a résulté en une implantation complexifiée de la méthode `get_waiting_for`.

Il était attendu des candidates d'indiquer comment les sémaphores sont initialisés. La majorité ont utilisé l'initialisation par défaut qui initialise un sémaphore à 1. Or, ils devaient être initialisés à 0.

Question 8 Cette question est préparatoire pour la question de synchronisation (Q10). Il était demandé d'introduire un *compteur* par travail pour refléter le *nombre* de travaux dépendant de lui et donc bloqués par ce travail. Plusieurs candidates n'ont pas respecté la spécification et ont utilisé la relation inverse concernant les dépendances.

Certaines copies ont voulu recalculer la valeur à chaque appel, il valait mieux stocker cette valeur.

Question 9 De manière générale, dans cette partie, pour construire les objets composant une chaîne de traitement, il était nécessaire de choisir un ordre de création. Un ordre naturel était de créer d'abord les actions, de les utiliser lors de la création des travaux et, enfin, d'utiliser les travaux pour définir la chaîne. Il était également possible de considérer l'ordre inverse. Dans ce cas, les classes correspondantes devaient inclure explicitement une méthode de mise à jour des attributs. Par exemple, si un objet `CICDPipeline` était créé avec une liste de travaux vide, il était nécessaire d'avoir une méthode `addWork` pour mettre à jour cette liste.

Cette question créait une dépendance cyclique dans l'ordre de création et les candidates devaient fournir une solution correcte et cohérente avec le code fourni précédemment.

Question 10 Certaines candidates n'ont pas proposé de solution de synchronisation avec des sémaphores mais ont utilisé la méthode `join` de la classe `Thread`. Or, ceci est faux puisqu'au moment où cette méthode est appelée, le travail correspondant n'est peut être pas lancé. En Python, ceci se traduira par une erreur (exception), alors que la solution exige que le travail courant se bloque.

Question 11 Peu de copies parviennent à relier la question à la notion de circuit dans un graphe orienté. De nombreuses copies se contentent, fautivement, de considérer le cas où seuls deux travaux existent et dépendent l'un de l'autre. Des copies ont évoqué une liste très générale de conditions provoquant des interblocages, ce qui n'était pas nécessaire pour répondre à la question.

Certaines réponses ont interprété la question de manière littérale et mettent en avant le fait que l'interblocage peut se produire entre travaux (instances de `Work`) et donc concerne la méthode `run` de la classe `Work` et non la méthode `run` de la classe `CICDPipeline`. Or, la question concerne l'apparition d'interblocages lors de l'exécution d'une chaîne CI/CD.

Question 13 Des copies ont voulu obtenir des points en citant des problèmes célèbres qui n'avaient aucun lien avec l'épreuve. Il fallait faire le lien avec la détection de circuit dans un graphe orienté. Comme algorithme, on pouvait proposer de repérer un arc arrière dans un parcours de graphe en profondeur ou une variante de l'algorithme de Kahn.

Question 14 Le jury s'attendait à lire des solutions basées sur la levée d'une exception, la modification de la signature de la fonction afin que sa valeur de retour soit booléenne ou encore l'écriture d'un message sur le flux d'erreur standard `stderr`. Plusieurs candidates ont tenté de répondre à la question en faisant appel à une fonction une peu magique de gestion des erreurs et non en levant proprement une exception.

Question 15 Il était attendu ici une modélisation qui reflète toutes les informations données sur le système. De nombreuses candidates ont omis le nombre d'opérations (consultations ou modifications) par utilisateur. D'autres ont introduit des dépendances circulaires dont l'impact a été ignoré lors de la requête de création demandée à la question suivante. L'introduction d'une table 'Operations' donnant le détail (estampille, utilisateur, projet, type) sur une opération n'était pas demandée mais a été considérée correcte.

Quand une réponse appelle le dessin d'un schéma, il est indispensable de réfléchir au brouillon où positionner les entités pour que les flèches ne se coupent pas. Trop de copies ont servi un plat de spaghetti.

Question 16 La requête attendue devait indiquer la clé primaire de la table et contraindre le nom du projet à ne pas être vide. Certaines copies souffraient de leur modélisation trop créative à la question 15.

Question 17 Imbriquer trois sous-requêtes pour répondre à cette question faisait un mauvais effet sur les correctrices. Le concours vise à sélectionner de futurs enseignantes capables de garder les choses simples.

Question 18 Des candidates se sont retrouvées piégées par la complexité de leur proposition.

Question 19 Il était attendu ici une proposition commentée sur la manière d'utiliser les informations sur le projet et l'utilisateur dans la construction de l'URL résultante. Une solution basée sur des routes est à privilégier comparé à une solution utilisant des paramètres de requête.

Question 20 Sans difficulté particulière, le code attendu ici devait explicitement avoir une partie HTML et une partie CSS, et respecter la structure de la page illustrée.

Question 21 Étaient attendus ici des identifiants aux sections qui correspondent aux informations concernées (nom de projet, nombre de consultations, nombre de modifications). Les scripts permettant la mise à jour effective de ces informations étaient à fournir dans les questions suivantes.

Certaines candidates ont rajouté des boutons permettant à l'utilisateur de changer le contenu de la page. Or, ce contenu ne doit être changé que si l'utilisateur charge les informations d'un autre projet ou alors recharge la page courante (mise à jour de nombre de consultations et du nombre de modifications).

Question 22 Il était attendu ici de rajouter un bloc de Javascript (`<script>…</script>`) dans lequel est envoyée régulièrement (activation et ensuite *timeout*) une requête GET au site pour récupérer les informations et mettre à jour le contenu de la page.

La majorité des réponses se contentent de définir une fonction simple de mise à jour de champs sur la page, sans définir la manière d'activer la fonction et d'effectuer la requête HTTP au serveur.

Question 23 La réponse attendue spécifie que toutes les couches sont sollicitées puisque le protocole HTTP(S) est au niveau *Application*. Avant l'échange HTTP, une première requête DNS est nécessaire. Au niveau *Transport*, HTTP s'appuie sur du TCP, DNS sur du UDP. Au niveau *Réseau* on utilise le protocole IP. Au niveau *Données* on utilise typiquement Ethernet. Enfin, la *couche physique* est sollicitée.

Question 24 Les correctrices attendaient une manipulation correcte des pointeurs en C avec allocation et initialisation faisant usage de `memcpy`.

Le jury a lu des copies montrant un manque de maîtrise de C flagrant : par exemple, déclarer une variable de type pointeur, l'initialiser avec une longueur, puis copier avec `memcpy` un contenu à l'adresse pointée.

Question 25 Il était attendu des candidates de définir deux structures : une pour un élément de l'arbre, et une pour l'objet arbre lui-même. Dans les deux cas il fallait définir un en-tête. Dans le cas de l'arbre ce dernier donne le type TREE_TYPE, alors que pour un élément existant il contient le type de ce dernier. Les éléments d'un arbre sont à représenter par un tableau de taille pré-définie donnée par une constante dans le code.

La fonction de création d'arbre vide devait allouer les structures nécessaires et initialiser correctement tous les champs. L'idée ici est que la fonction de hachage doit pouvoir retourner un identifiant, calculé sur la base du contenu de l'arbre, pour un arbre nouvellement créé.

Question 26 Dans ces fonctions, il était attendu que les candidates vérifient si un ajout ou une suppression est possible, en parcourant la liste des éléments. Dans le cas d'un rajout, la liste doit avoir un élément non occupé. Dans le cas d'une suppression, il faut trouver l'élément correspondant en comparant les hachés des éléments. Dans les deux cas, si une modification du contenu de l'arbre est effectué, il faut recalculer le haché global. Ce dernier point a été omis par une majorité de candidates.

De nombreuses candidates ont utilisé l'attribut de taille de l'en-tête d'un objet pour indiquer si l'objet a un contenu pertinent ou non. Or, comme spécifié par l'énoncé, cet attribut représente la taille en octets *du contenu* de l'objet. Par exemple, une structure `tree` a sa taille égale à la taille du tableau de ses éléments. Il est incorrect de considérer que le tableau est alloué mais que sa taille est de 0 puisqu'il ne contient pas d'éléments.

Question 27 Les deux objets sont à comparer en utilisant leurs hachés. Il était attendu que les candidates récupèrent ces hachés en accédant aux en-têtes des objets qui sont toujours stockés en début d'objet.

Les correctrices rappellent à nouveau qu'il est inutile d'écrire `if (a==b) {return true;} else {return false;}`. On peut se contenter de `return (a==b);`.

Utiliser `sizeof(o1)` où `o1` est de type `void *` pour retrouver la taille de l'objet désigné par `o1` est incorrect. La valeur de `sizeof(void*)` est la taille d'un pointeur.

Question 28 Il était attendu ici d'effectuer une "descente" dans la structure arborescente en suivant le chemin de l'élément recherché. À chaque étape l'algorithme exige une récupération des informations de l'élément courant ainsi qu'une vérification de type. En effet, si l'élément courant ne correspond pas à l'élément recherché, il est erroné de poursuivre la descente si son type est `BLOB` (fichier) et non `TREE` (répertoire).

Question 29 Tous les objets dans le système de gestion de versions sont organisés de la même manière : en-tête et contenu. Une majorité de candidates ont omis l'attribut d'en-tête lors de la définition de l'objet `commit`.

Question 30 Ce qui était attendu ici est la durée d'un travail en unités de temps et non une durée en secondes. Cette durée est disponible sans qu'il soit nécessaire d'exécuter la méthode `run` d'un travail. Il suffit de s'appuyer sur la fonction `len`.

Dans une question pareille, recopier l'ensemble de la classe `Work` n'a pas de sens et peut perdre la correctrice.

Question 31 La durée séquentielle est tout simplement la somme des durées des travaux composant une chaîne CI/CD.

Quelques copies ont confondu méthode et valeur attribut en écrivant `w.duration` au lieu de `w.duration()` (où `w` est un travail).

Des copies ont proposé avec élégance d'utiliser la fonction `sum` sur un itérateur mais une solution à base de boucle `for` convenait aussi.

Question 32 Cette question demande de trouver la durée maximale parmi les durées des séquences d'exécution de travaux dépendants d'une chaîne.

Il s'agit d'un problème tout à fait canonique de programmation dynamique, certes mis en contexte. Le jury regrette d'avoir lu de nombreuses réponses maladroites qui n'ont pas su le reconnaître et qui ré-inventent des concepts pourtant classiques en algorithmique. Il fallait veiller à ne pas recalculer les mêmes valeurs plusieurs fois.

Question 33 Cette question, ainsi que les suivantes, donne lieu à beaucoup de bavardages. Il faut analyser les termes de la question pour comprendre ce qui est attendu par le jury. Pour la première partie "en quoi consiste …", il fallait donner la définition du terme "intégration". Il est important qu'une future professeure soit capable de nommer et décrire les choses à bon escient et de donner des définitions des termes qu'elle emploie. Peu de copies évoquent la notion de version de référence.

Question 34 Les termes de la question appelaient une réflexion sur le *transfert* vers l'*environnement* de production. La plupart des copies concentrent malheureusement leur réponse sur le logiciel lui-même et répètent des choses déjà dites.

Question 35 Dans cette question, il est demandé de réfléchir au sens du terme "continu". Il est inutile de répéter sous une autre forme les éléments déjà apportés en réponse aux questions précédentes.

2.3.2 Fondements de l'informatique

— 35 présents
— Meilleure note : 20.0/20
— Moyenne : 10.3 ; écart-type : 5.65.

≥ 5	77,1%
≥ 5.99	75,0%
≥ 10	60,0%
≥ 11.34	50,0%
≥ 14.27	25,0%
≥ 15	20,0%

Présentation du sujet Cette année l'épreuve portait sur une variante d'automates appelée transducteurs. La première partie présentait les transducteurs séquentiels et contenait des questions pour s'approprier le concept ainsi que quelques questions de programmation. La deuxième partie introduisait les transducteurs sous-séquentiels et montrait différents résultats d'équivalence. Enfin, la dernière partie, qui a été très peu abordée par les candidates, proposait de démontrer deux théorèmes importants sur les transducteurs.

Si quelques copies ont impressionné le jury par leur clarté et la quantité de questions traitées, la majorité des copies a été particulièrement décevante, tant par les raisonnements incohérents et/ou incomplets que par le manque de maîtrise sur les langages et automates.

Commentaires généraux Le langage OCaml est au programme. Tenter d'improviser l'écriture de code en inventant sur le tas une syntaxe fantaisiste ressemblant à du Python est nécessairement voué à l'échec et ne rapporte qu'une infime partie des points dans le meilleur des cas. Voici quelques remarques particulières :

— il n'est pas possible de renvoyer une valeur au milieu d'une boucle en OCaml, comme on le ferait en Python. Par exemple, le code suivant contient une erreur de typage (car un `if` sans `else` ne peut être que de type `unit`) :
`for i = 0 to n - 1 do if test_booleen i then false done; true`
— une liste et un tableau sont deux objets très différents qui ne sont pas interchangeables. Inutile donc d'accéder au i-ème élément d'une liste `lst` par `lst.(i)`
— par défaut, les objets en OCaml sont non modifiables. Un code de la forme :
`let lst = [] in  for i = 0 to n - 1 do lst = lst @ [i] done`
passe l'interpréteur ou le compilateur sans erreur (mais avec un avertissement) mais ne fait absolument rien : l'égalité à l'intérieur de la boucle est un test booléen et ne modifie pas la liste ; il faudrait ici utiliser une référence
— toujours en lien avec le point précédent, `::` est un constructeur, pas un opérateur de modification de liste. `x :: lst` ne modifie pas `lst` pour lui rajouter `x`, mais crée une nouvelle liste. Il en est de même pour `@` (qui est un opérateur).
— le filtrage se fait par motif. Le résultat d'une opération n'est pas un motif. Le code suivant déclenchera systématiquement une erreur : `match x with | p - 1 ->`
— `List.make` n'existe pas.

Lorsque la candidate écrit une fonction auxiliaire non demandée par l'énoncé, une description très rapide de cette fonction est demandée.

Certaines copies traitent les questions de code en ajoutant de la coloration syntaxique selon les mots-clés du langage. Cela n'est pris en compte ni négativement, ni positivement par le jury, qui recommande de ne pas perdre de temps à le faire.

Le jury est étonné de constater que de nombreuses copies ne font pas la différence entre un alphabet (un ensemble de lettres) et une lettre (un élément d'un alphabet). On trouve parfois des phrases comme « le transducteur transforme les alphabets a et b en… ». L'agrégation est un concours d'enseignement et savoir utiliser le bon vocabulaire est essentiel pour faciliter les apprentissages de futures élèves.

Recopier les définitions et écrire des suites de mots qui ressemblent à un raisonnement mais qui n'en est pas un dessert la candidate car cela ne fait que rompre la confiance de la correctrice en les capacités de raisonnement. Les correctrices en sont venues à se demander si les candidates se sont entraînées avec une intelligence artificielle.

Une copie peu remplie mais dont les raisonnements sont rigoureux est à privilégier par rapport à une copie très remplie dont les raisonnements sont vides ou faux. L'encre ne rapporte pas de point.

Les notations ne sont pas toujours bien utilisées. Par exemple, on trouve dans plusieurs copies des formulations lourdes du genre « l'état dans lequel on arrive après avoir lu $\lambda(q, a)$ depuis l'état p », au lieu de simplement écrire « l'état $\delta(p, \lambda(q, a))$ ».

Commentaires par question

Question 1 Si cela n'a pas été sanctionné, le jury a remarqué que plusieurs copies parlent de « **l**'antécédent » comme s'il était unique (ce qui n'est pas le cas). Il est inutile de perdre du temps à donner tous les détails de calcul : le seul objectif d'une telle question est de permettre à la candidate de s'approprier les notions introduites.

Question 2 De nombreuses réponses essaient de décrire le fonctionnement d'un transducteur quelconque, alors que l'énoncé attend une description informelle de la fonction séquentielle du transducteur donné sur la figure. Se contenter de lister en français l'ensemble des transitions n'apportait aucun point.

Question 3 Pour cette question et les suivantes qui demandent de donner un exemple, il est important d'indiquer sans ambiguïté quel est l'état initial, par une flèche entrante. L'appeler q_0 ne suffit pas.

Question 4 Des solutions compliquées ont été proposées là où un seul état suffisait. Pour cette question et la suivante, il fallait bien garder en tête que les représentations par base étaient avec chiffre de poids faible à gauche, pas à droite.

Question 6 Il faut faire attention à renvoyer le mot dans le bon sens. Dans certaines solutions, un seul appel à `List.rev` ne suffit pas. En effet, si `v1`, `v2` et `v3` sont des listes, le code `List.rev (v1 @ v2 @ v3)` n'est pas en général égal à `v3 @ v2 @ v1`.

Question 7 Le cas de base doit être celui avec $m = 0$ (et pas $m = 1$), correspondant à la liste `[[]]`, c'est-à-dire ne contenant que le mot vide, et non pas `[]`.

Question 8 On peut tester l'égalité de listes en utilisant directement `=`, pas besoin de recoder une fonction spécifique. En outre, des fonctions compliquées ont été codées pour retrouver la taille de l'alphabet en parcourant toutes les transitions, mais cela donnait la taille de l'alphabet de sortie, pas celle de l'alphabet d'entrée (qui était nécessaire ici), qui pouvait se trouver simplement par un appel à `Array.length t.(0)`.

Question 9 Il faut terminer le raisonnement, et ne pas s'arrêter à $\Phi(\varepsilon)\Phi(\varepsilon) = \Phi(\varepsilon)$. Il ne faut pas confondre le mot vide et l'ensemble vide.

Question 10 De nombreuses copies utilisent les … lorsqu'il faudrait mentionner le raisonnement par récurrence, laissant planer le doute sur la maîtrise de ce raisonnement. Le cas de base $|u| = 1$ n'est pas nécessaire, le cas $|u| = 0$ suffit. Utiliser une récurrence ou une induction est attendu pour cette question, mais les conclusions y sont parfois étonnantes. Ainsi, cela n'a pas de sens d'écrire une phrase comme « Pour le cas de base $\Phi(\varepsilon) = \Psi(\varepsilon)$ donc $\Phi = \Psi$ » tant que la récurrence n'est

pas terminée. Il faut bien faire la différence entre les affirmations « $\Phi = \Psi$ » et « $\Phi(u) = \Psi(u)$ » pour un certain mot u.

Question 12 Plusieurs copies utilisent un raisonnement par l'absurde à la place d'un raisonnement par contraposée, ce qui alourdit la rédaction ; pour le sens réciproque, l'hypothèse d'accessibilité était inutile. En effet, pour un mot u donné, dire « il existe q tel que $q = \delta(q_0, u)$ n'apporte aucune information, car $\delta(q_0, u)$ est de toute façon un état bien défini, sans hypothèse d'accessibilité.

Question 14 Plusieurs copies donnent une machine de Mealy correcte, mais qui n'est pas une machine de Moore, contrairement à ce qui était attendu par l'énoncé. Il était nécessaire ici de rajouter un troisième état au transducteur.

Question 15 Tester si la liste est de taille 1 en utilisant `List.length` augmente inutilement la complexité temporelle et était sanctionné. On attend des candidates du recul sur la complexité des opérations utilisées.

Question 16 Pour ce genre de question, on attend une description formelle du transducteur, ainsi qu'une preuve suffisamment rigoureuse (pas nécessairement longue). Plusieurs copies font la preuve en supposant qu'il n'y a pas de transition entrante sur l'état initial, ce qui est faux dans le cas général.

Questions 21, 23, 24 Ces questions nécessitent de faire des constructions de transducteurs ou d'automates qui sont similaires. Si on attend que la première de ces questions soit traitée rigoureusement, il est acceptable de sauter certaines étapes pour traiter les suivantes, en particulier montrer l'égalité des langages/des fonctions calculées.

Question 24 Certaines solutions proposent une fonction de sortie de la forme $\lambda(q, a) = g(\lambda_f(q, a))$, ou $\lambda(q, a) = \lambda_g(q, f(a))$. Ces solutions sont incorrectes, car elles ne prennent pas en compte les lettres lues pour atteindre l'état q.

Question 26 Pour montrer que la fonction conserve les préfixes, il ne fallait pas oublier de montrer que $f(\varepsilon) = \varepsilon$; plusieurs copies essaient d'utiliser la question 12 pour résoudre cette question, mais toutes les hypothèses nécessaires ne sont pas vérifiées.

Question 28 À cette question, on attendait une preuve par induction en utilisant la question 27.

Question 32 On attend la justification d'une équivalence. Ne traiter qu'un seul sens du théorème de Ginsburg-Ziv n'apportait aucun point pour cette question facile qui ne nécessitait que de faire référence aux bonnes questions précédentes.

Question 42 Il est largement préférable de raisonner par propriétés de clôtures sur les langages rationnels que d'essayer de construire explicitement un ou des automates.

Chapitre 3

Épreuves orales

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.

Les épreuves orales se sont déroulées du 17 au 22 juin 2025 au lycée Guy Mollet à Arras. Les candidats admissibles ont été convoqués pour les trois épreuves orales sur trois jours consécutifs. Le jury est bien conscient du stress et de la fatigue des candidats avec des temps de préparation longs et des horaires parfois décalés. Son objectif n'est pas de piéger les candidats mais de les mettre dans un contexte leur permettant de montrer le meilleur d'eux-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 des livres sont disponibles pour les préparations de toutes les épreuves, à la fois ceux fournis par le jury et par les préparations à l'agrégation. Tous les livres sont accessibles à toutes les candidats. En 2025, les listes de livres étaient disponible dans l'environnement informatique des candidats. L'environnement informatique à disposition des candidats (pour les trois épreuves) est celui téléchargeable sur le site du jury et les fichiers des candidats sont sauvegardés régulièrement sur un serveur. Il est à noter que les responsables informatiques ne sont pas là pour aider les candidats 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 candidats qu'ils connaissent la description et les attendus des trois épreuves.

Il est conseillé aux candidats de passer les trois épreuves. Malheureusement, certains candidats de cette année ont abandonné avant la dernière épreuve. Une absence ou un abandon est éliminatoire alors que certains auraient pu avoir des chances d'être admis. Il est difficile d'évaluer sa prestation à un oral et le jury conseille de persévérer, même si le candidat a eu l'impression d'échouer à une épreuve.

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 le candidat le cas échéant), ils sont en fait des temps-cibles. Et le jury n'interrompt pas le candidat pendant cette présentation, même si celui-ci n'est pas clair.

Pour les trois épreuves, le jury apprécie lorsque le sujet est introduit proprement (intérêt, applications, mise en perspective, etc).

Le jury d'une épreuve d'oral n'a pas connaissance des résultats de l'écrit du candidat et s'impose une discipline stricte de ne pas discuter des prestations d'un candidat devant les autres membres

1. Le chapitre 2 est écrit au féminin.

du jury avant que celui-ci ait passé l'ensemble de ses épreuves. Lorsqu'un candidat se présente à une épreuve, il est ainsi assuré d'être examiné sans aucun préjugé.

Il est attendu que le candidat adopte une posture d'enseignant 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.

Les candidats ne doivent pas demander au jury comment faire leur présentation ("je dois écrire le plan ?", "vous voulez que je détaille ?", "est-ce que je dois lancer le code ?", "est-ce que je vous parle à vous ou à un élève (revue de code) ?").

Le jury conseille aux candidats de regarder son public au moins de temps en temps, de ne pas cacher ce que est écrit au tableau. Le candidat devrait également faire attention à sa posture (blasé, fatigué, indifférent...). Le jury déconseille aussi l'auto-dévaluation manifeste ("ce n'est pas très intéressant") ou une trop grande assurance ("c'est trivial" pour compenser un manque de temps).

Les épreuves orales ne sont pas un jeu de rôle dans lequel le candidat joue au professeur et le jury joue à la classe indisciplinée. Au cours de son exposé ou de ses réponses aux questions du jury, le candidat ne doit en aucun cas faire référence à un élément de compréhension qu'il aurait expliqué dans une séance antérieure comme il pourrait le faire avec des élèves qu'il suivrait sur une année complète.

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

Lorsque le candidat présente du code (notamment en TP et modélisation), le jury s'attend à une vraie exécution du code pendant la présentation. Dans le cas où le code ne fonctionne pas, le candidat peut tout de même le montrer et le commenter.

Le jury regrette que certains candidats recopient longuement au tableau leur code alors qu'il est dans l'environnement informatique et qu'il peut être projeté rapidement et sans erreurs.

Le jury déplore que de trop nombreuses/-eux candidats appellent « test » un simple affichage d'une exécution sur un exemple. Il rappelle qu'un test logiciel est formé d'une instance d'entrée associée à une valeur de retour ou un effet attendu clairement défini (aussi appelé « oracle »). Dans le cas de nombreux tests, il est préférable d'automatiser leur passage sous la forme de suites de tests. À noter que ces tests peuvent, par ailleurs, aider les candidats à produire un code plus juste en adoptant une pratique de développement piloté par les tests (TDD pour *Test-Driven Development*) lors de leur préparation.

Le jury regrette également que de trop nombreux candidats échouent à lire et/ou écrire un fichier texte ou binaire, structuré ou non, alors qu'il s'agit souvent d'une étape préliminaire et/ou indispensable à la réalisation de nombreuses épreuves requérant de la programmation. La mise en œuvre de telles routines dans n'importe quel langage ne doivent pas monopoliser le temps des candidats au détriment des autres questions abordées par l'épreuve.

Le jury peut poser des questions simples au candidat, 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 informaticiens *complets*. Les questions peuvent permettre d'évaluer la capacité des candidats à faire des liens avec d'autres sous-domaines que celui étudié spécifiquement par le sujet de l'oral.

Lors des réponses aux questions, il est attendu une posture d'enseignant. Si on ne connaît pas une réponse, il faut donc éviter les deux écueils suivants : se dénigrer, répéter que ce qu'on dit est faux, dire « je ne sais pas, mais si vous voulez, je peux inventer une réponse », … ou bien répondre avec beaucoup d'aplomb quelque chose de complètement faux (en en ayant connaissance ou non).

Le jury valorise particulièrement des réponses succinctes et précises. Donner une réponse trop longue ou détaillée ne rapporte pas nécessairement plus de points, surtout quand le candidat réalise qu'il a oublié la question initiale à la fin de la réponse. Le jury n'est d'ailleurs pas dupe quand un candidat répond volontairement de façon très longue pour essayer de limiter le nombre de questions.

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

Les candidats ne doivent pas chercher un sens caché ou figuratif aux questions du jury. En cas de doute, il peut être constructif pour un candidat 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 peut poser des questions qui n'appellent pas de réponse unique. L'important est d'expliquer la logique qui conduit à la réponse. Par exemple, à la question du jury : « faut-il demander à ses élèves de commenter son code ? » on peut autant répondre 'oui' que 'non' en indiquant le choix avisé du nom d'une fonction ou des variables peut suffire dans certains cas.

Il est très important pour le candidat de bien gérer son tableau. Ce point est évalué et peut refléter un manque au niveau pédagogique. Un contenu propre, clair et organisé, éventuellement utilisant des couleurs, sera valorisé. Les candidats 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 candidats peuvent toutefois s'autoriser à effacer ce qui n'est pas du contenu (une faute d'orthographe, un trait mal tracé…). Ils devraient pour cela utiliser la brosse plutôt que d'effacer avec leurs doigts. Le jury s'attend à ce que le candidat économise l'espace du tableau et structure son occupation, pour éviter tant que possible de réclamer à effacer.

Dans les épreuves au cours desquelles le candidat est amené à projeter du code au tableau, le jury abaisse et relève l'écran blanc à la demande du candidat. L'écran de projection pouvant cacher une partie du tableau, le candidat est invité 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 le candidat 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 candidats ne doivent pas faire de transparents (comme une présentation beamer), sauf éventuellement pour présenter un schéma.

3.1 Leçon

— 40 présents
— Meilleure note : 19.0/20
— Moyenne : 9.75 ; écart-type : 4.68.

≥ 5	80,0%
≥ 6.0	75,0%
≥ 9.75	50,0%
≥ 10	47,5%
≥ 12.63	25,0%
≥ 15	12,5%

L'épreuve comprend un temps de préparation de quatre heures et un oral d'une heure avec le jury. Le candidat se voit proposer (aléatoirement) deux sujets au choix ; il doit choisir de traiter l'un des deux, mais n'a pas à justifier son choix sur ce point. Il 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.

Le candidat a le droit d'utiliser les notes qu'il a prises pendant sa préparation tout au long de l'oral. Il 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, le candidat présente le plan de leçon qu'il a préparé et dont une copie aura été remise au jury. Au cours ou à la fin de cette présentation, le candidat indique

au jury deux propositions de développement (ils doivent également être signalés comme tels dans le plan).

Deuxièmement, pendant 20 minutes, le candidat 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. Le candidat 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 le candidat 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 candidats 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 du candidat à ces questions.

Si la leçon d'agrégation permet d'évaluer les compétences didactiques des candidats, 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 candidats.

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 candidats 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 candidats 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'enseignants : une expertise sur des domaines hors programme ne se substitue pas à un recul pédagogique.

Présentation du plan

Le jury souhaite voir proposé par le candidat 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 candidats 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, un candidat ne peut pas simplement, par exemple, écrire dans son plan « motif », il doit choisir une définition formelle de cette notion et l'écrire ; de même, s'il choisit de mentionner l'algorithme de Rabin–Karp, il ne peut se contenter d'en donner le nom, il 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 le candidat, 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 le candidat 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 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 candidats 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 le candidat 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 le candidat introduit dans son plan ou dans son exposé.

De nombreuses questions simples visent simplement à vérifier que le candidat comprend et maîtrise ce qu'il écrit ; elles sont posées assez systématiquement et le candidat ne doit pas en être déstabilisé.

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 du candidat, même s'il 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, le candidat sera interrogé et invité à le faire pendant l'échange avec le jury. Le candidat 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 candidats 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.

S'ils présentent une « méthode générale » dans leur plan, les candidats 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 le candidat 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 candidats 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 candidats qu'ils 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 du candidat à réfléchir et envisager plusieurs possibilités qui est évaluée.

3.2 Travaux pratiques

— 38 présents
— Meilleure note : 18.33/20
— Moyenne : 11.29 ; écart-type : 4.31.

≥ 5	94,7%
≥ 8.17	75,0%
≥ 10	63,1%
≥ 11.0	50,0%
≥ 14.83	25,0%
≥ 15	23,6%

Cette épreuve orale a pour objectif l'évaluation des compétences pratiques des candidats dans un cadre pédagogique. D'une durée de 60 minutes, elle se déroule en deux phases : une présentation de 30 minutes par le candidat 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 le candidat 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 le candidat 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 étudiants, 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 du candidat 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 du candidat, 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, le candidat 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 ; cette épreuve n'est pas une épreuve de virtuosité ou de vitesse. Durant la préparation de cette épreuve, le candidat 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).

Le candidat doit particulièrement travailler la réflexion concernant les choix de structures de données lorsqu'on lui demande de les proposer, un mauvais choix pouvant avoir de lourdes conséquences.

Durant la présentation, les candidats doivent éviter une présentation linéaire du code produit. Ils sont au contraire encouragés à décrire la structure globale de leur code et les principaux points clés, le jury pouvant ensuite, lors des questions, demander au candidat 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 respectant ses usages et spécificités est appréciée. Par contre, un recours excessive à des 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 un candidat 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 demeure 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 candidats 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 le candidat 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 candidats peuvent être questionnés sur tous les aspects de leur code, ainsi que sur toutes les phases de développement. Plus particulièrement, concernant le code, ils peuvent être interrogés 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 candidats connaissent les processus de compilation, d'exécution, de test et de déboguage.

Partie Revue de code

Durant cette partie, le candidat est confronté 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 un enseignant avec un élève ou étudiant 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 le candidat 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 au candidat de corriger le code proposé, en essayant de rester au plus près du code initial et 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 le candidat, de montrer qu'il est capable de produire le code demandé mais, dans un objectif pédagogique, de voir comment il peut aider un élève pour que celui-ci soit capable d'écrire correctement un tel code.

Remarques spécifiques pour la session 2025

Remarques sur le sujet *Satisfiabilité*

Ce sujet étudie deux méthodes pour déterminer la satisfiabilité d'une formule propositionnelle mise sous forme normale conjonctive.

Tout d'abord, on demande d'implémenter en langage C l'algorithme de Davis-Putman-Logemann-Loveland (ou DPLL), algorithme récursif qui procède par rebroussement (ou *backtracking*) : tant qu'il reste des littéraux non affectés, on en choisit un et on essaye récursivement de lui affecter la valeur `true` puis `false` si nécessaire, jusqu'à ce que toutes les clauses soient satisfaites.

Ensuite, on s'intéresse à une méthode de résolution basée sur l'utilisation de la structure d'arbre de décision binaire (ou *binary decision diagram*, BDD), une structure persistante où un partage maximal des sous-structures permet un traitement efficace. Cette partie est à implémenter en utiliser le langage OCaml.

Dans les deux cas, la description des formules propositionnelles dont on teste la satisfiabilité s'effectue à l'aide de fichiers au format DIMACS. On fournit aux candidats du code dans les deux langages pour lire ces fichiers.

La revue de code propose deux fonctions, en langages Python et C, pour supprimer les doublons dans des tableaux.

La présence, sous forme de fichiers, de nombreux exemples permettait de tester facilement les fonctions à écrire de manière approfondie.

Remarques sur le sujet *Gestion de réservations*

Ce sujet propose de traiter, dans deux parties indépendantes, des problématiques liées à la gestion d'un parc d'ordinateurs en réseau.

Dans la première partie, à traiter en utilisant le langage C, le candidat doit tout d'abord implémenter un schéma de consommation de ressources de type producteur-consommateur puis, dans un second temps, l'adapter afin de mettre en place un canal de communication bidirectionnel entre deux démons.

La seconde partie, à implémenter en langage OCaml, porte sur l'échange de messages informatiques dans un réseau représenté sous la forme d'un graphe non orienté. À l'aide de cette structure, on s'intéresse à deux problèmes. Tout d'abord, comment faire en sorte qu'un poste puisse envoyer un message à tous les autres postes du réseau. Ensuite, on souhaite simuler des échanges entre démons (un pour chaque poste) afin de mettre en place un processus de routage basé sur un arbre couvrant.

La revue de code présentait deux tentatives d'écriture d'algorithmes de tri.

3.3 Modélisation

— 39 présents
— Meilleure note : 19.5/20
— Moyenne : 9.54 ; écart-type : 4.05.

≥ 5	87,1%
≥ 6.63	75,0%
≥ 9.13	50,0%
≥ 10	48,7%
≥ 12.5	25,0%
≥ 15	7,6%

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 au candidat ;
— le candidat a trouvé ce texte tout seul lors de recherches pour préparer son exposé ;
— le candidat a formulé lui-même les pistes de réflexion qui sont proposées à la fin du texte (ou bien ces pistes sont présentes dans les brouillons d'un collègue, et le candidat 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 candidats 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 le candidat 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, le candidat 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 candidats 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. En outre, le candidat soit s'assurer de maîtriser le vocabulaire scientifique et technique employé, ce qu'il démontre en l'utilisant de façon appropriée dans son exposé et en étant en mesure de définir précisément les termes et concepts employés.

L'exposé du candidat 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 candidats. Le jury attend que le candidat 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. Le candidat 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, le candidat 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 le candidat exécute ses programmes durant la présentation afin de vérifier leur fonctionnalité. Le jury a regretté que peu de candidats 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, de l'interprétation pertinente des résultats et de la démarche etc. De même qu'à l'épreuve de leçon, le jury est attaché à la capacité des candidats 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 du sujet, et que ce dernier ne doit pas servir de simple prétexte à une longue digression dans un domaine où le candidat 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 qui présente un lien avec le texte du sujet. Cette ouverture ne suit pas nécessairement une piste proposée par le jury : toute initiative personnelle pertinente est bienvenue et a d'ailleurs vocation à être valorisée.

L'épreuve de modélisation nécessite une réflexion préalable des candidats 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 du jury.

Questions

L'interrogation se poursuit ensuite par les questions du jury, qui peuvent porter sur l'ensemble de la présentation du candidat, 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 candidats, de faire préciser ou corriger des points, de fournir des indications pour permettre au candidat 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 le candidat adopte une posture facilitant les échanges. Il 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'un candidat qui s'est lancé dans une réponse très longue.

Remarques générales sur la session 2025

L'épreuve de modélisation de cette session reposait sur deux sujets portant sur des thèmes différents : le test de performances par génération aléatoire et le délai de bout-en-bout dans les réseaux commutés. Les candidats 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, elles 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 le candidat, soit ils sont explicitement admis. Le jury a été satisfait du niveau global des candidats. Malgré

tout, certaines maladresses et lacunes sont apparues à plusieurs reprises. Le jury attend une posture d'enseignant de la part des candidats. Ils 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 maladroite. En particulier, l'optimalité n'est pas une échelle de valeur. On n'est pas plus ou moins optimal.

Remarques sur le sujet *Test de performances par génération aléatoire*

Le premier sujet portait sur la génération aléatoire d'entrées pour estimer la complexité en moyenne d'algorithmes. Le texte traitait plusieurs cas, un générateur de permutations pour évaluer le quicksort et plusieurs modèles pour la génération de textes afin d'évaluer un algorithme de compression. La discussion principale du sujet, au-delà des aspects techniques, portait sur le choix d'un modèle aléatoire et sur sa pertinence pour le problème étudié. Certains candidats n'ont pas traité la fin du sujet, et ce n'était pas un problème. Il vaut mieux faire des choix éditoriaux assumés et cohérents que de traiter tout le sujet de façon superficielle ou de manquer de temps pour terminer. Encore une fois, le jury attendait des expérimentations en rapport avec la problématique et pas de simples exercices de programmation. On retrouve trop souvent le cas de candidats qui ont écrit des programmes intéressants, mais ne s'attardent pas assez sur l'expérimentation ce qui rend leurs illustrations inefficaces.

Remarques sur le sujet *Délai de bout-en-bout dans les réseaux commutés*

Le second sujet portait sur l'estimation du délai de bout-en-bout dans les réseaux commutés. Ce sujet proposait plusieurs modèles de calcul de la distributions de ces délais pour des architectures de réseaux commutés plus ou moins sophistiquées. Le jury n'attendait pas un traitement exhaustif du sujet, mais bien une démarche d'expérimentation construite. Certains candidats se contentent de reformuler le sujet sous forme d'une longue liste de définitions de modèles. Cela rend souvent l'exposé un peu superficiel alors que le jury attend que le candidat prenne le temps de faire des observations expérimentales d'un ou plusieurs modèles en profondeur. L'impact des mécanismes de commutation, de l'architecture, des routes et des dates d'envoi de trames interférentes sur le délai de bout-en-bout d'une seule trame a été généralement bien traité. L'extension de l'exercice à la notion de flux de périodique a été peu traitée. C'est cette dernière qui nécessitait un effort individuel de modélisation plus poussé. Certains candidats ont traité l'illustration informatique en proposant d'implanter une file d'attente globale des messages à traiter par l'algorithme de calcul des délais de bout-en-bout. D'autres sont restés plus proches des pistes du sujet en proposant une implantation récursives. Quelle que soit la piste étudiée, il est de bon aloi de justifier les limites de ses choix en terme d'extension possible à d'autres études (routes, architectures de réseau, etc.).

MINISTÈRES
ÉDUCATION
JEUNESSE
SPORTS
ENSEIGNEMENT
SUPÉRIEUR
RECHERCHE
Liberté
Égalité
Fraternité

Concours externe de l'agrégation du second degré

Section informatique

Liste des leçons pour la session 2026

La liste des sujets de leçons pour la session 2026 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.