MP

CONCOURS COMMUN INP

RAPPORT DE L'ÉPREUVE ÉCRITE D'INFORMATIQUE

1/ PRÉSENTATION DU SUJET

Ce sujet cherche à évaluer l'ensemble des compétences visées par les programmes d'option informatique MP et d'informatique de tronc commun MP, en restant le plus proche possible des objectifs des programmes. Le sujet se veut strictement conforme aux programmes sus-mentionnés et comporte de nombreux rappels et précisions de vocabulaire, en particulier en ce qui concerne les « éléments techniques devant être reconnus et utilisables après rappel ».

Le sujet cherche à évaluer aussi bien les aspects théoriques et pratiques, avec :

- des questions appelant une réponse précise et rigoureuse ;
- des questions demandant de décrire informellement une idée ;
- des questions proches du cours ;
- des preuves formelles sur des objets informatiques :
- des questions de programmation dans les trois langages OCaml (aspects fonctionnels et impératifs), Python et SQL ;
- des questions d'analyse de programmes, que ceux-ci soient proposés par le sujet ou par les candidats.

Ce sujet s'articule autour d'un unique problème, motivé par une situation fictive mais concrète : celle du partage du réseau d'un pays imaginaire entre deux opérateurs. Le réseau est modélisé par une base de données en SQL (partie I), par un graphe en OCaml (parties II, III et IV) et par un graphe en Python (partie V). La procédure de partage est vue comme un jeu à deux joueurs (partie V) qui fait intervenir la notion de bande passante maximale (partie IV) liée au problème de la recherche d'un arbre couvrant de poids maximal (partie II) que l'on peut trouver à l'aide de l'algorithme de Borůvka (partie III).

La première partie (informatique de tronc commun, langage SQL) porte sur l'étude d'une base de données : sa modélisation, l'élaboration de requêtes et la compréhension d'une requête proposée par le sujet.

La deuxième partie (option informatique) étudie des propriétés élémentaires sur les graphes et les arbres couvrants pondérés, qui seront utiles dans tout le problème. Les questions sont plutôt théoriques et nécessitent un soin et une rédaction rigoureuse, avec des preuves plus ou moins délicates. Les concepts sont réintroduits de manière progressive. Cette partie est proche du cours mais nécessite déjà de faire preuve d'un certain recul. L'ensemble des propriétés peuvent être admises de manière à ne pas bloquer un candidat.

La troisième partie (option informatique, langage OCaml) propose de comprendre et de réaliser une

implémentation de l'algorithme de Borůvka en demandant de programmer de manière progressive de nombreuses fonctions en OCaml, en utilisant à la fois ses aspects fonctionnels et impératifs. Des questions de compréhension de programmes donnés, d'analyse de terminaison, de correction et de complexité sont proposées.

La quatrième partie (option informatique) se propose de montrer que la bande passante limite d'un réseau correspond au poids de l'arête de poids minimal de l'arbre couvrant de poids maximal du graphe. Les questions sont plus délicates mais peuvent être admises.

La cinquième partie (informatique de tronc commun, langage Python) propose de modéliser la procédure de répartition du réseau comme un jeu à deux joueurs. Les premières questions portent sur la théorie des jeux et la compréhension de la modélisation. Des questions élémentaires de programmation en Python demandent d'implémenter des primitives permettant ensuite de calculer les attracteurs, par une fonction score donnée qu'il est demandé de comprendre. Trois extensions indépendantes sont enfin proposées : mémoïser la fonction score en utilisant un dictionnaire ; modifier la fonction score pour implémenter l'algorithme min-max et pour finir une question théorique sur un vol de stratégie.

Les différentes parties ne sont pas complètement indépendantes mais le graphe de dépendance est clairement indiqué au début du sujet et il est explicitement indiqué qu'il est possible d'admettre des résultats intermédiaires pour pouvoir progresser dans le sujet. Le sujet est gradué en difficulté, globalement et au sein de chaque partie et le nombre de questions proches du cours est suffisamment important pour qu'un candidat moyen MP puisse traiter le sujet de manière conséquente.

2/ REMARQUES GÉNÉRALES

Le sujet a été bien compris dans sa globalité par la majorité des candidats. Il a permis de bien classer les candidates et candidats avec une moyenne de **10,12** et un écart type de **3,99**. Sa longueur et le niveau de difficulté semblent adaptés aux connaissances et compétences des étudiants et étudiantes qui ont suivi l'option informatique. Quelques copies ont abordé toutes les questions du sujet. On peut mentionner la qualité de certaines copies et, au contraire, un petit nombre de copies très lacunaires.

Aucune question n'a été systématiquement évitée, même si la dernière question — que l'on peut voir comme une question bonus — n'a été correctement traitée que par de très rares candidats. Soulignons quelques copies qui ne traitent absolument *aucune* question en OCaml, ce qui est très pénalisant.

L'analyse statistique des résultats montre que les premières questions (notamment en Partie I et II) ont été bien traitées, traduisant une bonne préparation des candidats sur les fondamentaux. Les parties III à V ont révélé des écarts plus marqués, mettant en lumière des faiblesses en analyse complexe, gestion du temps et formalisation. Certaines questions techniques, parfois laissées de côté par une majorité des candidats, ont cependant permis à quelques-uns de se distinguer par la qualité de leur raisonnement et la rigueur de leur rédaction.

Les correcteurs rappellent qu'une présentation du code, respectant les bonnes pratiques de programmation (indentation, retours à la ligne, noms de variables significatifs) est absolument impérative. De trop nombreux candidats écrivent trop souvent en mauvais français : orthographe et ponctuation défaillantes. Ils confondent longueur et qualité d'une preuve. On rappelle qu'une preuve est d'abord un texte en français. À ce titre, elle doit respecter les règles de grammaire et de ponctuation du français, qui sont la condition nécessaire de leur intelligibilité. De trop nombreuses copies présentent des lacunes graves dans ce domaine : mauvais accords entre adjectifs et noms, entre verbe et sujet (les défauts d'accord en nombre rendent les textes ambigus) ; ponctuation parfois totalement absente (une

phrase doit *toujours* commencer par une majuscule et être terminée par un point ; les virgules participent également au sens d'une phrase). Savoir exprimer ses idées de façon correcte et intelligible est une compétence essentielle pour un ingénieur. Les candidats qui ont des difficultés langagières doivent s'entraîner pendant leur préparation pour les éliminer. Rappelons qu'une partie du barème prend en compte la présentation, le soin et l'orthographe de la copie.

Le découpage de certains raisonnements en sous-questions a parfois été mal exploité par certains candidats, qui tentent par exemple de traiter en Q8 un problème qui fait l'objet de toute la partie III. De trop nombreux candidats n'analysent pas assez la structure du sujet. Les copies montrent trop souvent un manque de rigueur, avec des idées évoquées sans être démontrées. Enfin, peu de candidats utilisent les invariants et connaissent le terme de variant et encore moins les attracteurs.

En ce qui concerne le langage OCaml, les niveaux sont très variables. La présentation laisse souvent à désirer. On observe toujours des difficultés avec la gestion et la compréhension des booléens. De façon surprenante, certaines copies faibles font un usage fréquent du mot-clé **when,** souvent de façon maladroite ou incorrecte. Rappelons que ce mot-clé n'est pas au programme de CPGE et que toutes les fonctions demandées peuvent être écrites sans.

3/ REMARQUES SPÉCIFIQUES

PARTIE I - BASE DE DONNEES DU RESEAU

Cette partie est centrée sur les bases de données et le langage SQL. Elle a été globalement bien abordée, bien que certaines questions aient posé des difficultés notables. Les questions de type requête SQL ont été correctement traitées, mais la compréhension des jointures et des agrégats reste fragile chez une fraction significative des candidates et candidats.

Q1. Les notions de clés sont généralement mal connues, avec des confusions entre clés primaires et étrangères. La notion de clé étrangère n'étant d'ailleurs généralement pas comprise, voire inconnue. Les candidats ont majoritairement identifié correctement les clés primaires, mais les justifications sont souvent omises ou incorrectes. Certaines copies exploitent ici l'hypothèse d'unicité des bandes passantes, qui est faite seulement plus loin dans l'énoncé et n'était donc pas applicable à cette question. Attention, si **(id1, id2)** est une clé primaire pour la table liaisons, ni **id1** ni **id2** ne l'est.

Q2. Les requêtes **(a)** et **(b)** sont généralement bien traitées, mais les fonctions d'agrégation ne sont pas toujours connues. Notons quelques confusions **WHEN**/**WHERE**. La requête **(c)** a donné lieu à des solutions en deux colonnes (avec une jointure) ou en deux lignes (avec une union) : les deux ont été acceptées, lorsqu'elles étaient convenables. Pour la requête avec deux colonnes, on pouvait par exemple utiliser **ORDER BY bp DESC LIMIT 1** ou bien **WHERE bp = (SELECT MAX(bp) FROM liaisons)**.

Q3. Cette question, plus difficile, a posé de réelles difficultés. On ne demandait pas ici de décrire le résultat en français mais bien d'exécuter la requête sur l'exemple. Beaucoup de candidats ont mal interprété la requête : confusion entre **COUNT(*)** et **GROUP BY**, mauvaise compréhension de ce que réalisait **UNION**, résultat avec une liste des villes ou un ordre incorrect et beaucoup de copies sans réponses.

PARTIE II - ARBRE COUVRANT DE POIDS MAXIMAL

Cette partie théorique a été plutôt bien abordée, avec des taux de zéros très faibles sur toutes les questions. Les réponses sont en majorité présentes, ce qui montre que les notions de graphes (connexité, acyclicité, arbres couvrants) sont bien assimilées. Cependant, les questions de type "montrer que" ont été mal abordées, avec un taux élevé de non-réponses, surtout sur les questions plus abstraites. Les erreurs les plus fréquentes concernent un manque de rigueur dans l'utilisation des hypothèses (injectivité, connexité) ou dans la distinction entre graphe et sous-graphe.

Q4. La majorité des candidats a bien compris que supprimer une arête d'un cycle ne déconnecte pas le graphe et utilise le fait que le cycle fournit un chemin alternatif. Cette question semble cependant avoir été difficile à rédiger pour les candidats. L'idée est en général bien comprise, mais la rédaction est souvent confuse. Trop souvent, on a confondu longueur et qualité d'une preuve. Il n'était pas nécessaire ici d'expliciter les sommets du cycle ni des chemins : il est clair qu'on peut concaténer des chemins. En revanche, il fallait prendre garde à deux écueils : on doit vérifier la connexité pour n'importe quelle paire de sommets, pas seulement pour **x** et **y** ; et lorsqu'on a un chemin de **s** à **t** dans **G**, on doit, pour obtenir un chemin dans **G - a**, remplacer *toutes* les occurrences de l'arête **a** par le cycle, sauf à avoir explicitement considéré un chemin simple de **s** à **t**. Enfin, il faut dans ce genre de question bien prendre soin de préciser si les chemins sont dans **G** ou dans **G - a**.

Q5. On peut répondre à cette question en deux phrases, à condition d'appliquer convenablement la question **4** puis la proposition **1**. De trop nombreuses copies montrent ici une précipitation qui conduit soit à des développements longs mais peu convaincants, soit à une fausse application de la proposition **1** trop souvent confondue avec sa réciproque.

Q6. Réponses correctes mais souvent incomplètes. On n'attendait pas ici de détails concernant le sens réciproque. En revanche, le sens direct est rarement démontré. Beaucoup de candidats se contentent d'un « on peut enlever les cycles » sans justifier l'existence d'un sous-graphe minimal connexe, ou enlèvent des arêtes au graphe sans préciser pourquoi le processus s'arrête, ou encore en enlèvent une seule.

Q7. Beaucoup de copies répondent convenablement à cette question, mais parfois de façon inutilement compliquée, faute d'exploiter la proposition **1** et/ou les questions précédentes. La sous-question **(a)** est généralement très bien réussie. En revanche pour la sous-question **(b)** beaucoup de candidats oublient de justifier que **T + a - a'** est connexe (via **Q4**) et/ou qu'il comporte **|S| - 1** arêtes.

Q8. Cette question a été diversement traitée avec beaucoup de réponses vagues. L'argument de finitude de l'ensemble des arbres couvrants est souvent absent. Il ne suffit pas de dire « il existe un arbre de poids maximal » sans justification : il faut bien préciser que l'ensemble est fini et non vide. L'argument qu'une partie non vide et finie de **R** (et non dans **N** ici) admet un maximum est pourtant classique en informatique. Certaines copies prétendent donner un algorithme pour construire un arbre couvrant de poids maximal : une lecture sommaire de l'ensemble de l'énoncé doit conduire à comprendre que c'est hors de propos ici.

Q9. Question relativement bien traitée, notamment par les copies qui avaient déjà su exploiter les arguments de cardinal en question **7**. L'argument de poids strictement supérieur **p(a2) > p(a1)** est bien exploité.

PARTIE III - RECHERCHE D'UN ARBRE COUVRANT DE POIDS MAXIMAL

Cette partie, en OCaml, est la plus longue et la plus technique. Les résultats sont hétérogènes, avec des pics de réussite sur les questions simples et des difficultés croissantes sur les algorithmes complexes. Cette partie s'est révélée sélective, en permettant de valoriser les candidats capables de structurer un raisonnement complexe. Elle a aussi révélé des lacunes en analyse de cas et en compréhension de la complexité algorithmique. Une meilleure gestion du temps est aussi à envisager. La majorité des candidats maîtrisent les bases d'OCaml, mais peinent sur les algorithmes complexes. Notons également des copies qui ne traitent absolument aucune question en OCaml.

Q10. Une première question de programmation généralement réussie. La logique est pratiquement toujours correcte, même si on observe souvent des erreurs de syntaxe ou une mauvaise compréhension du langage. Si les correcteurs se montrent volontiers tolérants sur les petites erreurs de syntaxe, ils ne peuvent admettre qu'on confonde les listes et les tableaux, ni que l'on prétende calculer un maximum global en comparant un terme avec celui qui le précède. Concernant la complexité, il ne suffisait pas d'observer qu'il y a une boucle, il faut aussi indiquer que le corps de cette boucle est de coût constant.

Q11. Question très bien traitée. La représentation du graphe est comprise. Quelques erreurs mineures de syntaxe.

Q12. Les correcteurs ont été tolérants sur l'emploi du signe **=** ou **:=** au lieu de la flèche **<-**. En revanche, ajouter la nouvelle arête à droite, avec quelque syntaxe que ce soit, ne respecte pas la complexité demandée. Par ailleurs, il fallait préserver la symétrie de la représentation.

Q13. Question plus difficile et moins bien traitée. Il était possible de procéder par concaténations avec **@**, à condition de le faire dans un ordre qui respecte la complexité demandée. Cette approche est cependant source de complications techniques qui n'ont pas toujours été bien gérées. Le plus simple était sans doute d'utiliser, au sein d'une boucle, la fonction **List.iter**, rappelée dans l'énoncé, pour modifier une référence sur liste. Quelques candidats n'ont pas compris la question et ont concaténé les listes **g.(i)**. Le calcul de complexité est rarement précis. Enfin, dans de nombreuses copies, la condition **if i < j** était absente, ce qui a conduit à des réponses incomplètes ou erronées.

Q14. Cette question très classique n'a pas été aussi bien traitée qu'on aurait pu le souhaiter. La récursion est souvent mal maîtrisée. La faculté de filtrer les listes de taille 1 ne semble pas présente à l'esprit de certains candidats. Notons que les fonctions **fst** et **snd** (par ailleurs hors programme de CPGE) n'étaient pas utilisables ici, en plus d'être inutiles. Les fonctions **max** et **min** pourtant rappelées dans l'énoncé sont utilisées de manière non curryfiée (et donc incorrectement) : par exemple **min(a, b)**. La complexité linéaire est rarement justifiée. Notons enfin que ∞ n'est pas un symbole valide en OCaml.

Q15. Les explications ne sont pas souvent suffisamment précises. De nombreux candidats disent « ça explore les voisins » sans mentionner la propagation de l'indice de composante ou la condition de visite. Pour beaucoup de candidats, seuls les sommets adjacents sont mis à jour par cette fonction.

Q16. On note beaucoup d'erreurs dans la structure des programmes. De nombreuses copies négligent de produire une numérotation consécutive des composantes connexes, exigence qui se déduit de la deuxième phrase de l'introduction de la partie **III.3**.

Q17. Question généralement bien traitée. On observe toujours des fonctions qui terminent par **true** et s'évaluent donc systématiquement à **true** qui montrent des problèmes de compréhension du

fonctionnement du langage. Quelques rares confusions entre la longueur du tableau des composantes connexes et le nombre de composantes connexes. Beaucoup de solutions en une ligne qui utilisent judicieusement **max_tab** et **composantes_connexes**.

Q18. Question très bien traitée. L'arête sûre pour **{3}** est bien identifiée, mais celle pour **{7,8}**, de poids 18 est parfois confondue avec celle de poids 19.

Q19. Question peu traitée et rarement correctement. Certaines réponses se bornent à reformuler la question, ce qui n'apporte pas de point. Plusieurs candidats optent pour une démonstration par récurrence, mais la partie hérédité n'est pas souvent correctement traitée, affaiblissant ainsi la rigueur de la preuve.

Q20. On demandait de bien faire apparaître les différentes étapes et peu de copies décrivent clairement les trois étapes. Certaines copies perdent des points ici en ne permettant pas au correcteur de vérifier rapidement que l'algorithme a été bien exécuté. On observe également beaucoup d'erreurs dans le déroulé de l'algorithme, avec oubli de certaines arêtes sûres, ou une mauvaise identification des composantes après fusion.

Q21. Cette question simple a été diversement réussie : certaines copies déduisant la terminaison du fait que l'objet qu'on cherche à calculer existe. Parfois la notion de terminaison semble confondue avec celle de correction. Par ailleurs, fort peu de candidats ont le réflexe d'exhiber un variant/invariant, puis de justifier. Beaucoup de réponses indiquent simplement « l'algorithme termine car il y a un nombre fini d'arêtes » sans lien avec la fusion.

Q22. Question rarement traitée mais lorsque c'est le cas c'est souvent correctement fait. Dans le cas contraire l'ensemble est souvent très confus. L'invariant « **H** est un sous-graphe acyclique de **T*** » est rarement formulé.

Q23. Les explications fournies étaient souvent trop superficielles, manquant de rigueur et de détails. La complexité est rarement bien justifiée : on attendait une réponse courte, mais suffisamment précise. Le type de la fonction est souvent mal écrit. Par « type », on n'entend pas récursif ou impératif.

Q24. Question bien réussie. La boucle **while not (est_connexe h)** est généralement correcte. On observe de nombreux problèmes de parenthésage dans **not est_connexe g**, non pénalisés, ou de **while est_connexe h = false** qui dénote une pratique limitée des booléens.

Q25. On observe trop peu d'explications claires justifiant le raisonnement. En **(a)**, il ne suffit pas de dire qu'une arête sûre réunit deux composantes connexes : il faut dénombrer ces arêtes sûres. Certaines solutions parlent de « division par 2 » sans justification. En **(b)** la complexité **O(|A| log |S|)** avec justification est obtenue dans très peu de cas. Peu combinent les complexités des appels. En **(c)** les réponses sont acceptables pour ceux engagés, mais avec beaucoup d'abandons. Le cas du meilleur cas est cependant bien compris par ceux qui y répondent.

PARTIE IV - CHEMIN DE BANDE PASSANTE MAXIMALE

Cette partie courte mais exigeante, liant arbres couvrants et bandes passantes, a permis de faire une bonne distinction entre les candidats rigoureux et ceux hésitants sur la formalisation algorithmique. Les candidats ont du mal à faire le lien entre poids d'arêtes, bandes passantes et arbres couvrants. Les démonstrations par l'absurde sont mal maîtrisées. Le taux de non-réponses est très élevé, suggérant un abandon.

Q26. Question généralement bien traitée. Certaines copies proposent des chemins non optimaux comme le chemin **0-2-3-1** de bande passante 9 ou simplement l'arête **0-1** (non existante). Ceci est peut-être lié à une mauvaise compréhension du problème initial.

Q27. Question très délicate peu traitée. Très peu de copies ont traités les deux cas. L'avancée de la réflexion a été valorisée.

Q28. Le résultat étant donné dans l'énoncé, on attend ici des arguments précis et détaillés. Les réponses sont souvent partielles et sans réelle rigueur. L'idée est souvent comprise mais oubli fréquent de justifier que le minimum est atteint dans A*. Le cas non connexe (**blim = -∞**) est bien traité, mais le lien avec **min p(a)** dans **T*** est rarement démontré rigoureusement.

PARTIE V - JEU SUR UN GRAPHE

La partie V, en Python, mêle théorie des jeux et programmation. Cette partie très discriminante a souvent été laissée de côté, probablement en partie par manque de temps. Les candidats ont des difficultés avec la modélisation du jeu et les algorithmes de type min-max. Seules les questions de programmation basique sont bien réussies.

Q29. L'argument du variant (nombre d'arêtes restantes) est souvent présent, mais parfois mal formulé. Souvent il est juste écrit « nombre d'arêtes fini », ce qui ne justifie rien.

Q30. Les candidats doivent donner au correcteur le moyen de s'assurer rapidement de la validité de leur exemple. Il est bon de dessiner le graphe choisi, de signaler les arêtes prises par chacun des joueurs, de donner l'ordre dans lequel les arêtes sont prises et d'expliquer pourquoi MaxDébit n'a pas gagné. De nombreuses copies indiquent faussement que, dans leur exemple, le sous-graphe de MinLatence est connexe, alors qu'en réalité il a certes une seule composante étendue, mais aussi des sommets isolés.

Q31. Encore une fois, il n'a pas toujours été compris que la connexité du graphe obtenu par un joueur est celle du graphe avec tous les sommets présents initialement et pas seulement les sommets extrémités des arêtes jouées.

Q32. Question bien traitée. La condition **0 not in etat** est souvent trouvée.

Q33. Question bien traitée. La création de la liste résultat est bien comprise. Plusieurs candidats oublient d'utiliser **etat.copy()** et/ou modifient l'état original.

Q34. On attendait que soient mobilisés le vocabulaire et les algorithmes sur les attracteurs, qui sont au programme et trop peu connus. Un score 0 ne signifie pas qu'aucun joueur ne peut gagner. Les réponses sont souvent floues et cette question de cours est très discriminante.

Q35. Très peu de réponses satisfaisantes. Peu de candidats expliquent que MaxDébit doit jouer vers une configuration de score 1.

Q36. Question diversement traitée. Certaines copies présentent globalement la bonne structure de programme, mais font des appels à **score** au lieu de **score_memo** pour calculer les fils, sans que l'on puisse dire s'il s'agit vraiment d'une erreur de compréhension ou si c'est simplement une erreur d'inattention due au manque de temps à la fin du sujet. Certaines solutions ne mémoïsent que les feuilles.

Q37. Question très bien traitée par les quelques candidats qui l'abordent.

Q38. Question très délicate et peu abordée pour laquelle on a valorisé la réflexion. Quelques réponses excellentes qui ont bien compris l'argument du « vol de stratégie ».