

Agrégation externe de mathématiques, session 2006

Quelques remarques du jury sur l'option *Informatique*

Luc Bougé

26 octobre 2006

Table des matières

1	Remarques générales	1
2	Remarques sur le classement d'admissibilité	2
3	Remarques sur le classement d'admission	2
4	Remarques détaillées sur l'épreuve de leçon d'informatique	4
5	Remarques générales sur l'épreuve de modélisation/analyse de systèmes informatiques	5
5.1	Spécificités de cette épreuve	5
5.2	Organisation et déroulement de l'épreuve	5
5.3	L'exercice d'informatique	6
A	Annexe : remarques spécifiques sur quelques leçons	8
A.1	901 : Exemples de structures de données et de leurs applications	8
A.2	902 : Diviser pour régner : exemples et applications	9
A.3	904 : Arbres binaires de recherche. Applications	9
A.4	905 : Parcours de graphes : exemples et applications	10
A.5	906 : Programmation dynamique : exemples et applications	11

1 Remarques générales

Cette option a été ouverte en 2006 pour la première fois. Des préparations spécifiques ont été officiellement organisées à l'ENS Cachan (rassemblant les candidats de l'ENS Cachan, de l'ENS Ulm, de l'Université Paris 7 et de l'Université Paris 11), à l'ENS Lyon, à l'Antenne de Bretagne de l'ENS Cachan (Rennes/Ker Lann) et à l'Université Joseph Fourier à Grenoble. Au total, ces préparations regroupaient une quinzaine d'étudiants.

Environ 250 candidats s'étaient inscrits à l'option Informatique pour le concours 2006, mais en fait seulement environ 120 d'entre eux ont passé les épreuves écrites. Ce phénomène est observé à chaque session dans toutes les options : une partie significative des candidats inscrits en janvier renoncent finalement à se présenter aux épreuves en avril.

Sur les 595 candidats classés à l'admissibilité, 34 étaient inscrits dans l'option Informatique, soit environ 5 %. Parmi eux, 29 se sont présentés aux épreuves orales et ont été classés à l'admission parmi les 514 candidats classés. Finalement, 18 d'entre eux ont finalement été reçus parmi les 291 candidats reçus, soit 62 %. C'est légèrement supérieur au ratio global (291 reçus sur 514 classés, soit 56 %).

Les candidats de l'option Informatique se répartissent à peu près uniformément dans le classement final. Le premier candidat reçu est 17^e, le dernier reçu est 284^e et les médians (les 8^e et 9^e) sont respectivement 101^e et 134^e sur 291 candidats finalement reçus globalement.

Il est important de rappeler que seul le classement global est officiel, qu'il n'y a pas de quota prédéfini, et que l'option choisie par les candidats est "oubliée" après le concours : tous les candidats reçus sont agrégés de mathématiques, quelle que soit l'option choisie pour le concours, et tous sont gérés par l'Inspection générale selon les mêmes procédures.

2 Remarques sur le classement d'admissibilité

Les épreuves écrites d'admissibilité sont indépendantes de l'option Informatique, et les candidats de cette option sont donc sélectionnés sans référence à leur choix.

Si l'on étudie la distribution des 34 candidats admissibles de l'option par rapport à l'ensemble des 595 candidats admissibles (figure 1, partie gauche), on remarque une densité plus importante dans la seconde moitié de classement que dans la première moitié : les candidats de l'option Informatique de la première partie du classement sont moins bien classés en moyenne que les candidats des autres options. Par contre, l'effet s'inverse après le 300^e : les candidats de l'option Informatique sont alors mieux classés que la moyenne des candidats.

Comme attendu d'après l'histogramme, la moyenne des candidats admissibles de l'option Informatique est légèrement inférieure à celle de l'ensemble des candidats admissibles : la différence est de 1.17 point sur 40. Par contre, leurs notes sont un plus homogènes : l'écart-type est de 4.14, contre 5.04 pour l'ensemble des 595 candidats classés à l'admissibilité.

3 Remarques sur le classement d'admission

En ce qui concerne les épreuves orales d'admission, spécifiques à l'option Informatique, on note une amélioration significative du rang des candidats reçus entre l'admissibilité et l'admission. L'histogramme de droite de la figure 1 est très net sur ce point : les 29 candidats de l'option Informatique classés pour l'admission sont répartis parmi les 514 candidats classés de manière nettement plus uniforme qu'à l'admissibilité.

On peut remarque que la densité est cette fois légèrement supérieure en première moitié de classement, c'est-à-dire parmi les candidats finalement reçus (les 291 premiers). Ceci confirme donc la validité de la conception de l'option Informatique : elle sélectionne bien des candidats de valeur tout à fait comparable aux autres options.

Si l'on calcule l'écart entre le rang d'admission et le rang d'admissibilité pour les 514 candidats admissibles classés à l'issue de l'admission, on trouve un gain de 25 places en moyenne. Mais pour les 29 candidats de l'option Informatique parmi ces 514 candidats, ce

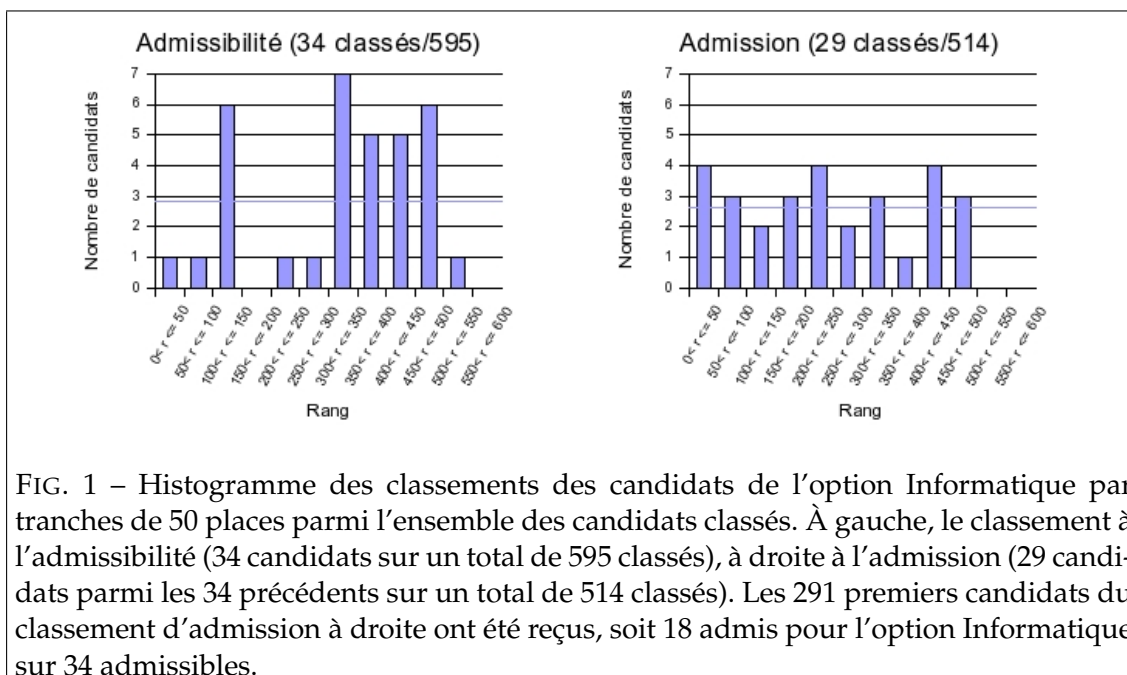


FIG. 1 – Histogramme des classements des candidats de l’option Informatique par tranches de 50 places parmi l’ensemble des candidats classés. À gauche, le classement à l’admissibilité (34 candidats sur un total de 595 classés), à droite à l’admission (29 candidats parmi les 34 précédents sur un total de 514 classés). Les 291 premiers candidats du classement d’admission à droite ont été reçus, soit 18 admis pour l’option Informatique sur 34 admissibles.

gain est de plus de 60 places. Les 18 candidats de l’option Informatique finalement reçus ont remonté en moyenne de 125 places entre l’admissibilité et l’admission, pour certains même plus de 300 places !

Ceci donne donc à penser que cette option a été choisie par des candidats qui y ont vu une opportunité stratégique pour valoriser leur formation complémentaire et ainsi améliorer leur classement final. La conception de cette nouvelle option a atteint son objectif : permettre aux candidats ayant une solide formation complémentaire en informatique d’émerger.

En étudiant la situation des 18 candidats finalement reçus, on peut de plus remarquer que cette formation complémentaire a été acquise pour environ la moitié en dehors des préparations spécifiques. Ceci montre le caractère ouvert de cette option, enrichissant ainsi la diversité des agrégés de mathématiques.

Une étude plus précise montre qu’une grande partie de ces candidats reçus n’ayant pas préparé le concours au sein d’une préparation spécifique sont en fait en cours de thèse en mathématiques ou informatique, voire même déjà titulaire d’un doctorat. Cette année, 4 candidats sur les 18 reçus sont en stage de validation du concours, et tous les 4 sont dans ce dernier cas.

Épreuves de leçon de mathématique et d’informatique. La moyenne des deux épreuves de leçon pour les 514 candidats admissibles classés à l’oral est de 9.62, avec un écart-type de 4.56. Si l’on se restreint aux 29 candidats de l’option Informatique de ce groupe, la moyenne des deux leçons est de 9.71, avec un écart-type de 5.57.

Le niveau général des candidats admissibles de l’option Informatique est donc comparable à celui de l’ensemble des candidats admissibles. Cependant, comme attendu, on note une moyenne meilleure pour la leçon d’informatique que la leçon de mathématiques.

Épreuve de modélisation, ou analyse des systèmes informatiques. En ce qui concerne l'épreuve de modélisation (aussi appelée analyse de systèmes informatiques dans le cas spécifique de l'option Informatique), la moyenne des 514 candidats admissibles classés à l'oral est de 9.68, avec un écart-type de 4.61. Par contre, si l'on se restreint aux 29 candidats admissibles de l'option Informatique de ce groupe, alors cette moyenne est de 11.50, avec un écart-type de 4.59.

Cette différence significative montre que les candidats de l'option Informatique ont donc su mettre en valeur leur formation complémentaire et leur sens de la modélisation acquis dans la pratique de l'informatique pour compenser leur niveau légèrement inférieur en mathématiques générales.

4 Remarques détaillées sur l'épreuve de leçon d'informatique

L'épreuve de leçon d'informatique a été organisée exactement sur le modèle de celles de mathématiques. L'épreuve dure 45 minutes. Un ensemble de 25 titres de leçons ayant été diffusé à l'avance, le candidat tire un couplage de deux titres parmi ces 25 et choisit l'un des deux. Après 3 heures de préparation, il expose son plan de leçon au jury pendant une petite dizaine de minutes, ainsi que deux propositions de développement. Le jury choisit l'une des propositions, le candidat expose ce développement pendant une quinzaine de minutes, puis une discussion libre s'engage avec le jury pendant le reste du temps.

Le jury a noté que la plupart des candidats ont choisi un sujet d'algorithmique quand le couplage le leur proposait. Seuls quelques candidats ont traité des leçons plus avancées comme *Machine de Turing* ou *Classes P et NP, NP-complétude*, ce qui est tout à fait compréhensible pour la première année de cette option.

De manière générale, le jury a plutôt été heureusement surpris par la qualité de certaines leçons présentées, qui confirme le bon travail des préparations spécifiques en amont du concours. Ceci a conduit à une moyenne des notes supérieure de 1 point sur 20 par rapport à l'épreuve d'algèbre des autres options.

Par contre, le niveau est très hétérogène, ce qui a conduit à un écart-type lui aussi plus grand de 1 point. À côté de candidats solides et bien formés en informatique fondamentale, il apparaît que d'autres candidats ne maîtrisent pas la différence entre l'*interface* d'une structure de données (pile, file, etc.) et son *implémentation* (par un tableau, par exemple). Ceci conduit à des raisonnements très confus et à l'opposé de toute pédagogie. De plus, l'évaluation de la complexité des algorithmes perd tout sens : l'accès à un élément d'une liste n'a pas le même coût que l'accès à un élément d'un tableau, même si la liste est implémentée par un tableau !

D'autre part, beaucoup de candidats, ne semblant pas connaître la notion d'invariant, sont incapable d'argumenter de manière convaincante sur la correction d'un algorithme simple (sans même parler de preuve formelle). Le jury a constaté une certaine difficulté des candidats à sortir d'une *ambiance mathématique* pour faire appel au *bon sens informatique*, si l'on peut s'exprimer ainsi. Par exemple, pour le calcul de la complexité du tri fusion, on arrive à une formule du type $f(2^{k+1}) = 2 \times f(2^k) + c$. De manière surprenante, plusieurs candidats ont eu les plus grandes difficultés à résoudre cette récurrence par des arguments

élémentaires. De même, les présentations des arbres binaires de recherche et des tas a donné lieu à des contradictions étonnantes.

Une tentation bien compréhensible est de mathématiser les sujets de leçons en oubliant l'aspect informatique. Ainsi, sur le sujet *Langages typés : objectifs, mise en oeuvre, applications*, il était tentant de faire une leçon complètement centrée sur la théorie abstraite des types (lambda-calcul typé, système F , etc.), en oubliant complètement les aspects concrets et élémentaires de typage statique et dynamique en Caml, C, C++ ou Java.

Le jury tient donc à rappeler qu'il s'agit bien d'une épreuve d'*informatique fondamentale*, et non pas d'outils mathématiques pour l'informatique. Il appartient au candidat de montrer la pertinence des outils mathématiques qu'il développe vis-à-vis des objectifs du thème informatique développé dans la leçon, que ce thème soit d'intérêt scientifique ou technologique.

La présentation d'outils mathématiques pour eux-mêmes, en particulier lorsqu'il s'agit d'outils sophistiqués comme ceux de la théorie des types, s'apparente donc à un "hors-sujet". Les deux questions-clés de cette épreuve sont toujours *A quoi cet outil mathématiques sert-il dans le cadre informatique considéré?* et *La complexité ou le coût de son utilisation est-elle bien compensée par la qualité supplémentaire d'information qu'il permet d'obtenir?*

5 Remarques générales sur l'épreuve de modélisation/analyse de systèmes informatiques

5.1 Spécificités de cette épreuve

Cette épreuve a plusieurs spécificités qui la distinguent des épreuves de leçon.

1. Le jury n'est pas censé connaître en profondeur le problème à modéliser.
2. Un exercice d'informatique doit être présenté.
3. L'organisation est souple, du fait de la variété des textes et de la très grande liberté du candidat dans le choix de ses développements.
4. La juste place du formalisme mathématique est *au service* de la modélisation.

Les candidats semblent avoir choisi le sujet adapté à leur profil. Ils ont eu tendance à préférer les textes de modélisation sur les jeux. Ils ont alors spontanément présenté les solutions non triviales sur des exemples, ce qui a été apprécié du jury.

Ils ont bien géré leur temps de préparation et relativement peu se sont trouvés à court de matière. À l'inverse, certains ont fait un exposé trop long qui ne leur a pas permis de faire une présentation détaillée des points originaux choisis par eux.

5.2 Organisation et déroulement de l'épreuve

Un schéma type. Le candidat dispose de 45 mn pour présenter le texte et son exercice d'informatique. Le candidat doit d'abord montrer qu'il a compris le problème et sa modélisation. Il peut s'appuyer sur un exemple, celui du texte, ou une construction personnelle. Ceci dure

environ 10 mn. L'exercice s'insère naturellement lorsque la propriété à implémenter a été exposée. Durant les 25 autres minutes, le candidat doit développer la modélisation, et aborder les développements qu'il a choisis.

Les variantes. Le temps passé sur l'exercice d'informatique est variable. Certains textes proposent des modélisations complètement indépendantes, d'autres approfondissent progressivement une modélisation ou solution.

Les suggestions. Le jury souligne que les mots-clefs, qui permettent de guider le choix initial du candidat, indiquent aussi les notions du programme qui se rapportent à la modélisation. Par exemple, il est souhaitable que le candidat exhibe un automate, voire un langage régulier, si le mot-clef *automate* est donné. En développant un modèle, le candidat est amené à présenter des solutions non élémentaires. Faire tourner l'algorithme eût été profitable à de nombreux candidats pour bien comprendre la solution... ou plus simplement pour bien l'exposer.

Les écueils à éviter. La présentation n'est pas une leçon. Un exposé trop détaillé peut montrer une compréhension étendue de l'ensemble du texte, mais prive le candidat de temps pour développer des points précis. Enfin, le candidat mène l'interrogation, et peut toujours refuser une voie proposée par le jury si elle dépasse le programme.

Pédagogie. La qualité de l'exposition du problème à un jury, qui n'est pas familier du problème traité, est un facteur important d'évaluation des qualités pédagogiques. La présentation d'un algorithme difficile n'est pas un facteur d'appréciation en soi : c'est la clarté de la présentation, de l'argumentation de sa correction et des exemples représentatifs qui seront pris en compte, et non pas la difficulté algorithmique.

Formalisme. De même, la démonstration de théorèmes généraux n'est pas un but de cette épreuve. Il s'agit plutôt de mettre en évidence leurs liens avec le problème étudié. Les théorèmes classiques sont considérés comme admis, leur preuve relevant de l'épreuve de leçon. Cependant, le traitement d'un cas spécifique et original sera apprécié. Par exemple, la démonstration du théorème de Ford-Fulkerson n'est pas un développement attendu, mais par contre il est intéressant de montrer un exemple de situation non-triviale de l'application de l'algorithme.

5.3 L'exercice d'informatique

Le candidat dispose de 10 minutes pour présenter cet exercice. La programmation est obligatoirement faite dans un langage au programme : C, Caml et Java pour 2006. Le jury évalue la qualité du code et sa lisibilité. La clarté des explications et la justification des choix effectués, par exemple les structures de données ou la complexité du traitement en temps et en espace, sont des éléments d'appréciation. Une modification des données pourra être demandée en cours de présentation.

Quelques candidats se sont présentés avec un programme qui ne s'exécutent pas correctement : c'est inévitable dans ce type d'exercice. Ceci n'est pas dramatique en soi : un candidat capable d'expliquer sa démarche conceptuelle et d'analyser de manière raisonnée la source des difficultés rencontrées a été récompensé. Par contre, un candidat refusant de présenter son programme inachevé a été sévèrement pénalisé.

Il n'est pas utile de faire plus que l'exercice demandé. Cependant, le candidat peut choisir d'illustrer un développement par une présentation informatique plus large indépendamment de l'exercice de programmation. Il peut alors utiliser l'ensemble des outils à sa disposition pour l'épreuve s'il y trouve son intérêt : Matlab, Maple, etc.

A Annexe : remarques spécifiques sur quelques leçons

A.1 901 : Exemples de structures de données et de leurs applications

Cette leçon a été traitée plusieurs fois en 2006, et nous en faisons donc ici un commentaire détaillé à destination des futurs candidats.

Le principal problème constaté par le jury est que les candidats confondent la notion de *structure de données* avec celle d'*implémentation*. Cette confusion est d'ailleurs largement encouragée par des langages comme Matlab, Maple ou Perl...

Il nous semble que la bonne approche de cette leçon est de mettre en avant la notion de *type de données abstrait*. Par exemple, une liste est soit la liste vide, soit un élément suivi d'une liste, et idem pour un arbre. De cette manière, on fait une nette différence entre les fonctions de manipulation des structures de données (dans un langage orienté-objet, on dirait la *déclaration* des méthodes) et leur *implémentation*.

Une liste peut évidemment être implémentée par un tableau, mais aussi par des cellules chaînées et sûrement de beaucoup d'autres manières. Mais une liste *n'est pas* un tableau : l'accès au n^e élément de la liste se fait par la combinaison de n opérations élémentaires, et non pas par une seule comme dans un tableau. De même, considérer la tête (*head*) ou la queue (*tail*) de la liste vide est une erreur, alors qu'il est tout à fait possible de manipuler un tableau non initialisé. La contrepartie de cette garantie minimale est qu'une liste est une structure de données abstraite *dynamique*, sans limitation de taille *a priori*, contrairement aux tableaux qui doivent être introduits avec une *taille fixée*.

Il s'agit ici de raisonner *abstraitement* à partir des garanties algorithmiques minimales spécifiées par un type de données abstrait, et non à partir d'une implémentation donnée, peut-être particulièrement efficace dans le cas considéré. Pour une bonne introduction à cette approche, la référence classique (mais déjà un peu ancienne) est le livre de Christine Froidevaux, Marie-Claude Gaudel et Michèle Soria, *Types de Données et Algorithmes*, McGraw-Hill France, février 1990, 570 pages.

Une fois que le candidat a mis en place les méthodes de manipulation de listes : *cons*, *head*, *tail*, *empty*, *is_empty*, il peut présenter proprement toutes les fonctions usuelles sur les listes : *append*, *reverse*, etc. On attend des candidats qu'ils aient intégré que *append* n'est pas une méthode de base de coût élémentaire, mais une méthode dérivée, de coût potentiellement proportionnel à la taille de la liste de gauche. Encore une fois, il n'est pas exclu que cette méthode puisse être exécutée en un temps plus petit à cause des particularités de telle ou telle implémentation spécifique. Mais les contraintes algorithmiques minimales exigées par le *type de données abstrait* ne permettent pas de le garantir en général.

Le cas des arbres binaires est semblable, et il a donné lieu au même genre de confusion. Le cas des piles et des files semble mieux intégré, peut-être à cause de l'aspect "familier" de ces structures dans la vie courante. Cependant, certains candidats confondent *empiler* et *enfiler*, au motif que les deux peuvent être vues comme une simple manipulation d'index sur un tableau.

Il est frappant de remarquer combien peu de candidats s'aident d'un dessin pour illustrer les manipulations de structures de données ! Le jury encourage chaleureusement les candi-

dates à présenter leurs idées d'abord *intuitivement* par un petit schéma, et seulement dans un deuxième temps de les *formaliser* dans un énoncé algorithmique plus rigoureux, si c'est demandé explicitement par le jury.

Pour tous les exemples d'algorithmes, la référence de base est le livre de Thomas Cormen, Charles Leiserson, Ronald Rivest et Clifford Stein disponible en français sous la titre *Introduction à l'algorithmique, Cours et exercices corrigés* chez Dunod.

A.2 902 : Diviser pour régner : exemples et applications

Cette leçon a été traitée plusieurs fois en 2006, et nous en faisons donc un commentaire détaillé à destination des futurs candidats.

L'exemple le plus souvent traité est le tri (souvent avec une confusion entre liste et tableau, voir la leçon 901!) La résolution de l'équation de récurrence de base, $t(n) = a \times t(n/b) + c(n)$ est souvent traitée de manière approximative, alors qu'il est possible dans le cas simple du tri fusion d'obtenir une valeur exacte pour $t = 2^k$. La plupart des candidats oublient d'ailleurs de remarquer que la fonction $t(n)$ est bien croissante en n , ce qui autorise à interpoler pour les valeurs de n qui ne sont pas des puissances de 2.

Les candidats les plus solides soulignent que la complexité des différentes méthodes de tri dites *optimales* sont asymptotiquement les mêmes dans le pire cas, mais que ces tris ont des comportements très différents "en pratique", par exemple si les éléments du tableau sont en fait déjà partiellement rangés.

Les candidats ne semblent pas connaître le concept d'*hybridation* d'algorithmes : par exemple pour les méthodes de tri de type *diviser pour régner*, il est possible avant chaque division de commencer par parcourir le tableau (en temps linéaire) pour vérifier s'il ne serait pas déjà trié, et rendre directement le résultat s'il l'est. De manière plus générale en algorithmique numérique, par exemple pour la résolution de systèmes linéaires, la technique algorithmique choisie sera étroitement dépendante des propriétés du système à résoudre, et il peut être intéressant de changer de méthode à la volée.

Ici encore, pour tous les exemples d'algorithmes, la référence de base est le livre de Thomas Cormen, Charles Leiserson, Ronald Rivest et Clifford Stein disponible en français sous la titre *Introduction à l'algorithmique, Cours et exercices corrigés* chez Dunod. Une référence plus encyclopédique est bien sûr la somme de Donald Knuth, mais la présentation ne prend pas en compte l'approche moderne des types de données abstraits.

A.3 904 : Arbres binaires de recherche. Applications

Une leçon très populaire, souvent choisie... mais rarement bien traitée!

Le premier point est de présenter la *structure de donnée abstraite* des arbres binaires, et de définir correctement ce qu'est un arbre de recherche (ABR). Il faut ensuite montrer que les principales opérations d'insertion, de recherche et d'extraction d'un élément se font en temps $O(h)$, où h est la hauteur de l'arbre, d'où l'intérêt de garder cette hauteur petite.

Très peu de candidats montrent que cette hauteur ne peut être inférieure à $\log(n)$, et qu'il faut donc viser des arbres approximativement équilibrés, en faisant un compromis entre :

1) le coût et la précision des rééquilibrages ; et 2) le coût des opérations d'insertion et de recherche. Chaque méthode d'équilibrage doit être discutée spécifiquement par rapport à ce compromis.

Les méthodes suivantes nous ont été présentées : arbres AVL, arbres rouge-noir, arbres 2-3-4, etc. Cependant, très peu de candidats ont su comparer les méthodes par rapport au compromis ci-dessus. La difficulté est de bien expliquer par rapport à quels arbres ces méthodes sont comparées. Il y a une grande confusion entre un ABR *aléatoire* et un ABR engendré par l'insertion des éléments successifs d'une *permutation aléatoire*. Il faudrait donc préciser clairement quel est le modèle de distribution de données retenu. Ce point est très délicat : le candidat qui l'aborde doit s'attendre à des questions du jury.

A.4 905 : Parcours de graphes : exemples et applications

Certainement l'une des leçons les plus populaires de la session 2006, et probablement l'une des plus délicates ! Les candidats les plus solides ont traité le parcours en profondeur et en largeur, l'algorithme de Bellman-Ford et celui de Dijkstra.

Beaucoup de candidats sont troublés par le problème de l'orientation des graphes et n'arrivent pas à se décider. Il nous semble plus logique de considérer des graphes *orientés*. Attention cependant à bien souligner que seuls les sommets *atteignables* sont parcourus !

En ce qui concerne les parcours en profondeur (*Depth-First Search*, DFS) et en largeur (*Breadth-First Search*, BFS), les candidats sont en général capables de les présenter sur le cas particulier des arbres, mais le plus souvent oublient le cas des graphes généraux contenant des cycles.

Aucun candidat n'a fait clairement la liaison entre les *stratégies de parcours* DFS/BFS et les *structures de données* pile/file. En fait, il semble qu'ils soient déroutés par le fait que le DFS est souvent programmé en récursif, avec une pile *implicite*, alors que le BFS l'est en itératif, avec une file *explicite*.

Il faudrait que les candidats montrent que les deux algorithmes dérivent en fait du même *algorithme itératif générique* de parcours qui utilise une structure de données plus générale qui inclut les piles et les files comme cas particuliers. Suivant la spécialisation de cette structure en pile ou en file, on obtient respectivement les parcours DFS ou BFS. L'écriture de DFS en récursif n'est alors qu'une optimisation secondaire. Ceci impose cependant de bien définir *abstraitement* les propriétés d'un parcours de graphe : par exemple, les sommets parcourus sont exactement les sommets atteignables depuis le sommet origine du parcours.

Les propriétés caractéristiques de BFS et DFS devraient être énoncées. Pour le parcours BFS, aucun sommet à distance $d + 1$ de l'origine du parcours n'est visité avant que tous les sommets à distance d ne l'aient été. Pour le parcours DFS, aucun sommet n'est quitté (dé-pilé) avant que tous ses descendants accessibles par un chemin ne contenant pas de sommet encore empilé ne soient quittés. Il faut noter cependant que la preuve *rigoureuse* de ces deux propriétés est délicate.

Pour le traitement de Bellman-Ford et Dijkstra, il est important de bien mettre en place la notion de *poids des arêtes*, pour pouvoir distinguer le *poids* d'un chemin de sa *longueur*. Une notion fondamentale est celle de *chemin/circuit élémentaire* et de *cycle absorbant*. Une fois ces

notions posées, les preuves sont beaucoup plus simples.

On remarque une grande difficulté chez la plupart des candidats à distinguer les propriétés de type *sûreté* (*safety*, invariant) et de type *vivacité* (*liveness*, variant sur un ordre bien fondé). Ils ont donc beaucoup de peine à organiser logiquement leurs preuves d'algorithmes, et surtout à pointer les propriétés clés.

Les applications de ces algorithmes sont nombreuses : recherche d'itinéraire sous mappy (Dijkstra), stratégie de jeu au casino pour maximiser le gain (Bellman-Ford), etc. Dans ce dernier cas, par exemple, l'absence de cycle absorbant exprime qu'il n'existe pas de moyen de gagner indéfiniment... ce qui est assez raisonnable !

A noter que certains candidats ont proposé de présenter l'algorithme de Tarjan de recherche des composantes fortement connexes (atteignables). Aucun n'a fait de présentation satisfaisante, même sur un exemple simple, sans parler de preuve, d'ailleurs particulièrement délicate. Les candidats doivent savoir qu'ils devront être rigoureux sur ce point.

A.5 906 : Programmation dynamique : exemples et applications

Cette leçon a été choisie par de nombreux candidats, avec des exposés parfois très bons.

Un point délicat est de bien faire la relation entre le paradigme de la *programmation dynamique* et celui de *diviser pour régner*. Les deux approches sont étroitement liées, mais diffèrent par la manière dont les résultats intermédiaires sont traités.

Quelques (bons) exemples traités par les candidats ont été : la recherche de l'ordonnement optimal sur deux lignes de montage parallèles ; la multiplication optimale d'une chaîne de matrices ; la recherche de la plus longue sous-séquence commune de deux mots, avec plus ou moins d'approximations dans la comparaison ; etc.

Dans tous les cas, après avoir écrit les équations de récurrence, il est éclairant de ramener le problème au parcours d'un polyèdre en respectant les contraintes de dépendance, et de voir ainsi qu'il existe un grand nombre de possibilités pour parcourir cet espace. On peut alors demander au candidat d'en esquisser quelques-unes, ou de discuter de l'efficacité de l'implémentation de ces variantes, etc.

Voici quelques autres exemples possibles : Knuth-Morris-Pratt, table des préfixes-suffixes d'un mot, conjugué minimal d'un mot, recherche d'un génome dans un ruban d'ADN, algorithme de Dijkstra (plus courts chemins dans un graphe orienté), etc.

Pour les candidats les plus solides, on peut aussi discuter de l'implémentation parallèle de ces algorithmes, en montrant que certains parcours sont plus favorables que d'autres. Cela touche au domaine de la *parallélisation automatique*. Un candidat particulièrement solide a discuté la notion d'algorithme glouton dans cette leçon, en mentionnant la théorie des matroïdes, avec comme application le sac à dos fractionnable.

Un point délicat de cette leçon souvent oublié est la *remontée* : une fois que la valeur optimale est trouvée, exhiber un objet correspondant à cette valeur. Ceci suppose de laisser de l'information lors du parcours du polyèdre : ce n'est pas très difficile, mais encore faut-il y avoir réfléchi un peu à l'avance ! La question de la remontée, ou plus précisément de l'exhibition d'une solution explicite au problème posé, est beaucoup plus importante que celle

des raffinements algorithmiques qui n'ont parfois pas de pertinence pratique démontrée. Le jury a été très attentif à ce point.