Programmes des classes préparatoires aux Grandes Ecoles

Filière : **scientifique**

Voie : **Mathématiques, physique et sciences de l'ingénieur (MPSI) – Mathématiques et physique (MP)**

Discipline : **Option Informatique**

Première et seconde années

http://www.enseignementsup-recherche.gouv.fr

Option informatique

Objectifs de formation

L'enseignement de l'informatique en classes préparatoires MPSI, MP ou MP* a pour objectif la formation de futurs chercheurs et ingénieurs. L'informatique est un secteur marqué à la fois par une forte croissance de la recherche et développement mais aussi par une obsolescence rapide des technologies.

C'est pourquoi ce programme met l'accent sur les méthodes générales et l'ingénierie logicielle qui seront utilisées dans une démarche de résolution de problème. Cette formation doit permettre de développer les compétences suivantes :

- analyser et modéliser un problème, une situation, en lien avec les autres disciplines scientifiques ;
- concevoir une solution modulaire, utilisant les méthodes de programmation et les structures de données appropriées ;
- traduire un algorithme dans un langage de programmation ;
- spécifier rigoureusement les modules ou fonctions ;
- développer des processus d'évaluation, de contrôle et de validation ;
- communiquer à l'écrit ou à l'oral, une problématique, une solution.

Le programme se veut ambitieux, cohérent, sans toutefois aborder des concepts trop difficiles, et en restant dans un cadre pratique. Les étudiants doivent mettre en œuvre les outils conceptuels étudiés, en programmant dans un langage de programmation, sous la forme de programmes clairs, courts et précis.

Une note de service précisera la liste des langages recommandés.

La virtuosité dans l'écriture de programmes ou une connaissance exhaustive des bibliothèques de programmation ne sont pas des objectifs de la formation.

Programme de première année

Méthodes de programmation

On présente la méthode d'analyse descendante (par raffinements successifs). Même si on ne prouve pas systématiquement tous les algorithmes, il faut dégager l'idée qu'un algorithme doit se prouver.

On étudie la complexité des algorithmes du programme ainsi que le lien entre complexité et structures de données : on présente des exemples de complexité logarithmique, linéaire, quadratique, polynomiale, exponentielle, en ne s'attachant qu'à l'étude du cas le pire. On s'intéresse également aux questions d'occupation de la mémoire. Les récurrences usuelles : $T(n)=T(n-1)+a\ n$, $T(n)=a\ T(n/2)+b$, ou $T(n)=2\ T(n/2)+f(n)$ seront introduites au fur et à mesure de l'étude de la complexité des différents algorithmes rencontrés.

On s'attache à obtenir des étudiants une documentation aussi complète que possible de leurs algorithmes (condition d'entrée, de sortie, invariants dans les boucles ou les appels récursifs). Toutes ces notions sont dégagées à partir des algorithmes étudiés sans aucune théorie générale sur les prédicats ou les invariants de boucles.

Itération

Boucles conditionnelles et boucles inconditionnelles.

Récursivité

On mettra l'accent sur la gestion au niveau de la machine, en terme d'occupation mémoire, de pile d'exécution, et de temps de calcul, en évoquant les questions de sauvegarde et restauration du contexte.

On évitera de se limiter à des exemples informatiquement peu pertinents (factorielle, suite de Fibonacci…).

Toute théorie générale de la dérécursification est hors programme.

Contenus	**Commentaires**
Lien avec le principe de récurrence, exemples tirés des mathématiques. Récursivité simple, récursivité croisée.	On se limite à une présentation pratique de la récursivité.
Lien avec les relations d'ordre ; exemples de récursions fondées sur des relations d'ordre sur des parties de **N** ou de **N**x**N**.	Il faut insister sur l'importance de la preuve de terminaison de l'algorithme.

Diviser pour régner

L'objectif poursuivi ici est de parvenir à ce que les étudiants puissent par eux-mêmes, dans une situation donnée, mettre en œuvre la stratégie « diviser pour régner ».

Contenus	**Commentaires**
Principe général de la méthode.	Exemples d'application : tri par partition-fusion (merge sort), comptage du nombre d'inversions dans une liste, multiplication des entiers (algorithme de Karatsuba), calcul des

	deux points les plus proches dans un nuage de points du plan.

Programmation dynamique

On attend des étudiants qu'ils sachent reconnaître, dans les cas simples, les situations où la programmation dynamique peut être utilisée, puis qu'ils l'utilisent effectivement, de façon autonome. Les cas plus complexes seront guidés.

On utilise la programmation dynamique dans différents algorithmes du programme des deux années (par exemple, l'algorithme de Floyd-Warshall sur les graphes).

Contenus	**Commentaires**
Principe général de la méthode : choix d'une valeur caractérisant une solution optimale, définition récursive associée, calcul par mémoïzation, reconstruction d'une solution optimale à partir de l'information calculée.	Exemples d'application : ordonnancement de tâches pondérées (weighted interval scheduling), alignement de séquences (distance d'édition).

Structures de données et algorithmes

Il s'agit de montrer l'intérêt et l'influence des structures de données sur les algorithmes et les méthodes de programmation.

On insiste sur la distinction entre structure de données abstraite (un type muni d'opérations, ou encore : une interface) et une structure de données concrètes (une implémentation). On montre l'intérêt d'une structure de données abstraite en termes de modularité : plusieurs réalisations concrètes sont interchangeables.

On distingue les structures de données persistantes (ou immuables) des impératives (ou modifiables). L'accès à des mémoires de taille toujours croissante permet aujourd'hui de reconsidérer l'intérêt des structures de données persistantes : on peut ainsi par exemple assurer une gestion d'historique d'une base de données, à l'instar de ce que permet Wikipedia.

Les algorithmes sont présentés au tableau, en étudiant, dans la mesure du possible, leur complexité. On limitera les calculs de complexité au cas le pire.

Certains de ces algorithmes font l'objet d'une programmation effective : les programmes correspondants doivent rester clairs, courts et précis.

Aucune connaissance sur les bibliothèques de l'environnement de programmation n'est exigible.

Structures de données

Contenus	Commentaires
Définition d'une structure de données abstraite comme un type muni d'opérations. Spécification en termes de modèle. Distinction entre structure de données persistante (immuable) et impérative (modifiable).	On montre l'intérêt d'une structure de données abstraite en termes de modularité (plusieurs réalisations concrètes sont interchangeables). Grâce aux bibliothèques de l'environnement de programmation, on peut utiliser des structures de données avant d'avoir programmé leur réalisation concrète.
Piles, files, dictionnaires, files de priorité. Utilisation d'une structure de données.	Applications : évaluation d'une expression arithmétique postfixée à l'aide d'une pile ; si une file de priorité offre des opérations d'ajout et de retrait de coûts logarithmiques (ce qui sera réalisé plus loin), alors on en déduit un tri en O(N log N).

Tableaux et listes

Contenus	Commentaires
Définition récursive du type liste. Réalisation de la structure de pile à l'aide d'une liste.	On ne parle pas de tableau redimensionnable.
Réalisation de la structure persistante de file à l'aide de deux listes. Réalisation de la structure impérative de file à l'aide d'un tableau.	Pour la structure de file réalisée dans un tableau, on se fixe une taille maximale.
Réalisation de la structure impérative de dictionnaire à l'aide d'un tableau.	On pourra aussi présenter une réalisation à l'aide d'une table de hachage.

Arbres

Contenus	Commentaires
Définition récursive du type arbre binaire. Vocabulaire : nœuds, feuilles, hauteur. Relation entre le nombre de nœuds et le nombre de feuilles.	On se limite aux arbres immuables.