# Projet de programme pour l'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

### I 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* $T(n) = 2T(n/2)+f(n)$ *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.*

### I.1 Itération

Boucles conditionnelles et boucles inconditionnelles.

### I.2 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 récusions fondés sur des relations d'ordre sur des parties de $\mathbb{N}$ ou de $\mathbb{N} \times \mathbb{N}$. | Il faut insister sur l'importance de la preuve de terminaison de l'algorithme. |

### I.3 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. |

### I.4 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). |

## II 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.*

### II.1 Structures de données

| **Contenus** | **Commentaires** |
|---|---|
| Définition d'une structure de données abstraite comme un type muni d'opérations. Spécification en termes de modèle.<br>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.<br>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 $O(N \log N)$. |

### II.2 Tableaux et listes

| **Contenus** | **Commentaires** |
|---|---|
| Définition récursive du type liste.<br>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.<br>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. |

### II.3 Arbres

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

# Programme de deuxième année

## II Structures de données et algorithmes

### II.3 Arbres

*Les arbres permettent la réalisation de structures de données : structure persistante de dictionnaire, structure persistante de file de priorité. Ils permettent aussi de représenter des expressions arithmétiques ou des formules logiques.*

| Contenus | Commentaires |
| --- | --- |
| Arbre binaire de recherche.<br>Réalisation de la structure persistante de dictionnaire à l'aide d'un arbre binaire de recherche.<br>Structure de tas.<br>Réalisation de la structure persistante de file de priorité à l'aide d'un arbre binaire ayant la structure de tas ; réalisation de la structure impérative de file de priorité à l'aide d'un tas stocké dans un tableau.<br>Représentation d'une formule de la logique propositionnelle par un arbre.<br>Application : satisfiabilité d'une formule logique. | On ne cherchera pas à équilibrer les arbres. Les arbres AVL, 2–3, 2–3–4, bicolores, ... sont hors programme. |

## III Notions de logique

*Le but de cette partie est de familiariser progressivement les étudiants avec la différence entre syntaxe et sémantique, à travers l'étude des expressions logiques et arithmétiques. L'étude du calcul des prédicats et les théorèmes généraux de la logique du premier ordre sont hors programme.*

### III.1 Calcul propositionnel

| Contenus | Commentaires |
| --- | --- |
| Variables propositionnelles. Connecteurs et formules logiques.<br>Tables de vérité, tautologies, satisfiabilité. | Il s'agit d'insister sur l'interprétation d'une formule logique et sur les manipulations logiques élémentaires. On mettra en évidence la difficulté du problème de la satisfiabilité d'une formule. |

### III.2 Exemples de manipulation formelle de termes et de formules sans quantificateur

| Contenus | Commentaires |
|---|---|
| Différence entre syntaxe abstraite (ou arborescente) et valeur d'une expression de la logique propositionnelle, d'une expression arithmétique. Évaluation et interprétation. | On illustre la différence entre syntaxe et sémantique, expression formelle et interprétation. On utilise les arbres pour représenter les formules. |

## IV Graphes

*Il s'agit de définir le modèle des graphes, leurs représentations, leurs manipulations, et les algorithmes de parcours les plus fondamentaux.*
*On s'efforce de mettre en avant des applications importantes et si possible modernes : carte routière, métro, graphe du web, bio-informatique. On précise autant que possible la taille typique de tels graphes.*
*Une attention toute particulière est portée sur le choix judicieux du mode de représentation en fonction de l'application et du problème considéré. On étudie en conséquence l'impact de la représentation sur la conception d'un algorithme et sur sa complexité (en temps et en espace).*

### IV.1 Vocabulaire des graphes

| Contenus | Commentaires |
|---|---|
| Sommet (ou nœud), arête (ou arc), orienté/non-orienté, graphe pondéré, degré, et pour le cas orienté degré entrant/sortant.<br>Pour les graphes orientés et non-orientés, notions de chemins, de composantes connexes, et fortement connexes dans le cas orienté. | On n'évoque pas les multi-arêtes, ni les arêtes qui bouclent sur le même sommet. |

### IV.2 Représentation des graphes

| Contenus | Commentaires |
|---|---|
| On présente, compare et implémente les deux représentations naturelles : par matrice d'adjacence, ou par listes d'adjacence.<br>Opérations élémentaires de manipulation de graphes : construction d'un graphe, ajout/suppression d'une arête, ajout/suppression d'un sommet. | On commente l'impact du choix de la représentation (matrice ou listes d'adjacence) du graphe sur l'espace mémoire. La manipulation de graphes non-orientés est délicate car il faut préserver la non-orientation. On explique comment rendre non-orienté un graphe orienté. |

### IV.3 Algorithmes sur les graphes

| Contenus | Commentaires |
|---|---|
| Notion de parcours (sans contrainte). Parcours en largeur (BFS), parcours en profondeur (DFS). | On implémente ces parcours à l'aide d'une représentation du graphe en listes d'adjacence. On fait le lien avec la recherche d'un plus court chemin pour les graphes non pondérés. On justifie le choix d'un parcours en largeur (plus court chemin) ou en profondeur (existence d'un chemin, connexité) en fonction des applications. On compare les implémentations de ces deux parcours et leurs complexités en temps et en espace. |
| Recherche des composantes connexes d'un graphe non orienté. | |
| Pour les graphes pondérés, algorithmes de Floyd-Warshall et Dijkstra. | On justifie le choix entre l'utilisation de l'algorithme de Floyd-Warshall et celui de Dijkstra en fonction des applications. On implémente l'algorithme de Floyd-Warshall avec la représentation des graphes par matrice d'adjacence. On implémente l'algorithme de Dijkstra avec la représentation des graphes par listes d'adjacence et une file de priorité. |

# V Motifs, automates et expressions

*La recherche de motifs est une pratique récurrente dans de nombreux secteurs du numérique : on peut par exemple citer la biotechnologie (et tout particulièrement la recherche de certains facteurs génétiques ou bien de repliements de code) ou encore l'analyse du langage naturel et les bases de données textuelles.*
*L'objectif est, à partir du problème simple de la recherche de motifs, d'introduire les expressions rationnelles comme formalisme dénotationnel pour spécifier les motifs, et les automates finis comme formalisme opérationnel efficace pour la recherche de motifs.*
*On souligne l'intérêt de l'approche modulaire qui consiste à spécifier le motif dans un formalisme haut niveau, traduire efficacement ces expressions rationnelles dans le formalisme bas niveau, pour finalement exécuter l'automate déterministe sur le texte afin de trouver toutes les occurrences du motif cherché.*
*On étudie la complexité dans le cas le pire, en temps et en espace, des algorithmes et on privilégie les algorithmes efficaces pouvant être utilisés en pratique pour la recherche de motifs dans les compilateurs, les traitements de textes...*

## V.1 Recherche de motifs

*On introduit les motifs au moyen d'exemples en dimension 2 (une croix, un chiffre, un flashcode) et en dimension 1 (un code barre, une adresse mail, un identificateur dans un programme). Un motif définit un ensemble d'objets. Ainsi, les croix dans une image peuvent avoir différentes largeurs, hauteurs, épaisseurs, couleurs...*

*Dans la suite, on se concentre sur les motifs en dimension 1 qui définissent donc des ensembles de mots.*
*On insiste sur la nécessité d'avoir un formalisme pour définir les motifs.*
*Les algorithmes naïfs de recherche d'un motif permettent de faire pressentir la notion d'état.*

| Contenus | Commentaires |
|---|---|
| Recherche de motifs dans un texte. | L'objectif est l'étude d'algorithmes génériques permettant la recherche d'un motif définissant un ensemble de mots et pas l'étude d'algorithmes spécialisés dans la recherche d'un unique mot (Knuth-Morris-Pratt, Boyer-Moore...). |
| Algorithmes naïfs sur des exemples. | Par exemple : entiers divisibles par 3 et codés en binaire, mots dont les voyelles apparaissent la première fois dans l'ordre alphabétique (*bateau* mais pas *binaire*)... |

### V.2 Expressions rationnelles

| Contenus | Commentaires |
|---|---|
| Expressions rationnelles (ou régulières) standards (union, concaténation, itération). Langage dénoté par une expression. | Introduction des expressions rationnelles comme formalisme dénotationnel pour les motifs. Motivation de l'extension par des exemples : phrases contenant à la fois les 3 mots fils, fille, mère mais pas le mot père... |
| Expressions rationnelles étendues (intersection, différence). | |
| Langages locaux. | Un langage local est défini par la donnée des préfixes de longueur 1, des suffixes de longueur 1 et des facteurs de longueur 2 interdits. Les propriétés de clôture des langages locaux ne sont pas exigibles. |
| Expressions rationnelles linéaires (chaque lettre apparaît au plus une fois dans l'expression). Le langage d'une expression rationnelle linéaire est local. | La partie sur les expressions rationnelles linéaires prépare à la transformation efficace d'une expression en automate. On donne les algorithmes permettant de calculer les ensembles $P$ (préfixes), $S$ (suffixes) et $F$ (facteurs) définissant le langage local associé à une expression linéaire. On précise la complexité de ces algorithmes. On remarque qu'il y a des langages locaux qui ne sont pas linéaires. |
| Passage d'une expression rationnelle standard à une expression linéaire par marquage. | |

### V.3 Automates

| Contenus | Commentaires |
|---|---|
| Automates finis déterministes. | |
| Automate local. Construction de l'automate local associé à un langage local. Complexité. | Un automate local est un automate déterministe (en général non complet) tel que pour chaque lettre $a$, toutes les transitions étiquetées par $a$ arrivent dans un même état. Il est *standard* si aucune transition n'arrive sur l'état initial. |
| Automates finis non-déterministes. | Les automates avec $\varepsilon$-transitions ne sont pas au programme. |

| **Contenus (suite)** | **Commentaires** |
| --- | --- |
| Construction de l'automate de Glushkov associé à une expression rationnelle standard. Complexité. | L'automate de Glushkov s'obtient par linéarisation de l'expression, calcul des ensembles définissant le langage local, construction de l'automate local, suppression des marques utilisées pour la linéarisation. (Cette procédure est aussi connue sous le nom d'algorithme de Berry-Sethi.) |
| Déterminisation.<br>Clôture par intersection et complémentaire. | On applique la déterminisation à l'automate de Glushkov d'une expression. La réciproque du théorème de Kleene (passage d'un automate à une expression) n'est pas exigible. |