

OBJECTIFS DE FORMATION ET PROGRAMME

DE L'OPTION INFORMATIQUE DANS LA FILIÈRE MP

I. OBJECTIFS DE FORMATION

L'informatique est une science opérant sur des représentations rigoureuses de concepts bien définis. Le programme doit donc permettre de présenter les principes de la programmation ainsi que les bases de l'algorithmique, de la théorie des automates et de la logique. Le programme qui suit se veut à la fois ambitieux et cohérent, tout en évitant d'aborder les concepts trop difficiles ou trop techniques, qui relèvent des études ultérieures.

Par ailleurs, un enseignement d'informatique doit être confronté à un « principe de réalité » : les étudiants doivent donc mettre en œuvre les outils conceptuels étudiés, en programmant dans un langage de programmation, sous la forme de programmes clairs, courts et précis. La liste des langages autorisés fait l'objet d'une publication actualisée. La virtuosité à manipuler un langage n'est pas un objectif du programme.

II. PROGRAMME DE LA CLASSE MPSI

II.1. MÉTHODES DE PROGRAMMATION

On présente la méthode d'analyse descendante (c'est-à-dire par raffinements successifs). Même si l'on ne prouve pas systématiquement tous les algorithmes, il faut dégager l'idée qu'un algorithme doit se prouver, à l'instar d'un théorème de mathématiques. On étudie systématiquement la complexité des algorithmes du programme et, sur certains exemples, le lien entre cette complexité et les structures de données. On s'attache à obtenir des étudiants une documentation aussi complète que possible de leurs algorithmes (conditions d'entrée dans un module, conditions de sortie, invariants dans les boucles ou les appels récursifs). Toutes ces notions sont dégagées à partir des algorithmes du programme, sans aucune théorie sur les prédicats ou les invariants de boucles.

II.1.1 Itération

Boucles conditionnelles et boucles inconditionnelles.

II.1.2. Récursivité

Lien avec le principe de récurrence, exemples tirés des mathématiques (factorielle, puissances, dérivées d'ordre n). Récursivité simple (la fonction s'appelle elle-même), récursivité croisée (deux fonctions s'appellent l'une l'autre).

Lien avec les relations d'ordre ; exemples de récursions fondés sur des relations d'ordre sur des parties de $\mathbf{N} \times \mathbf{N}$.

Comparaisons, sur quelques exemples, d'algorithmes récursifs et itératifs (factorielle, puissances entières, suites récurrentes, PGCD, insertion et suppression dans un arbre binaire).

Exemples de structures récursives.

On se limite à une présentation pratique de la récursivité.

Il faut insister sur l'importance de la relation d'ordre sur l'ensemble permettant de garantir la terminaison de l'algorithme.

On mentionne certains problèmes posés par la gestion de la récursion au niveau de la machine (occupation mémoire et temps d'exécution) : sauvegarde et restauration du contexte. Toute théorie générale de la dérécursification est hors programme.

On cherche à dégager sur quelques exemples le lien entre les structures de données et les méthodes de programmation.

II.1.3. Diviser pour régner

Principe général de la méthode. Exemples d'application : multiplication des entiers (par dichotomie), des polynômes, des matrices ; tri rapide, tri par fusion.

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

II.1.4. Éléments de complexité des algorithmes

Notion de taille de données, évaluation du nombre d'opérations (calculs, comparaisons, ...) nécessaires à l'exécution et de l'encombrement mémoire.

Notion et exemples de complexité logarithmique, polynomiale, exponentielle (comparaison de temps d'exécution).

On se limite à dénombrer les opérations nécessaires pour évaluer le temps d'exécution. Tout autre considération est exclue du programme.

On peut consacrer une séance de travaux pratiques à une évaluation expérimentale de la complexité de divers algorithmes pour résoudre un même problème.

II.2. STRUCTURES DE DONNÉES ET ALGORITHMES

Il s'agit de montrer l'influence des structures de données sur les algorithmes et les méthodes de programmation. On met notamment en parallèle les structures récursives des types de données et des programmes qui les manipulent. Les algorithmes sont présentés au tableau, en étudiant, dans la mesure du possible, leur complexité (dans le cas le pire et/ou en moyenne). Certains de ces algorithmes font l'objet d'une programmation effective : les programmes correspondants doivent rester clairs, courts et précis.

II.2.1. Listes et Piles

Définition récursive du type liste.

Fonctions Tête et Queue, parcours correspondant. Calculs récursifs de la longueur, du maximum, du n -ième élément, de la concaténation de deux listes et de l'image miroir. Insertion et suppression d'un élément. Insertion dans une liste triée.

Piles. Évaluation d'une expression arithmétique postfixée à l'aide d'une pile.

On peut utiliser la notation :

$$\text{Liste} = \text{nil} + \text{Élément} \times \text{Liste}.$$

Mêmes programmes avec une programmation itérative. Comparaison des deux méthodes, en montrant la simplicité de l'écriture récursive. Exemples de représentation d'objets mathématiques par des listes.

Seule la définition des piles est au programme. La dualité entre récursivité et itération avec piles ne l'est pas.

II.2.2. Arbres

Ce paragraphe n'est pas abordé en première année.

II.3. AUTOMATES FINIS

Ce chapitre n'est pas abordé en première année.

II.4. NOTIONS DE LOGIQUE

Le but de cette partie est de familiariser progressivement les étudiants avec la différence entre syntaxe et sémantique, ceci 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 ne sont pas au programme.

II.4.1. Éléments de base du calcul propositionnel

Variables propositionnelles, connecteurs logiques (NOT, AND, OR, XOR, NAND \implies , \iff), formules logiques, représentation d'une formule logique par un arbre, évaluation des formules logiques, tables de vérité, tautologie et satisfiabilité, difficulté de tester une tautologie : solutions de caractère exponentiel.

Il s'agit d'insister sur l'interprétation d'une formule logique et sur les manipulations logiques élémentaires.

II.4.2. Fonctions booléennes

Fonction booléenne associée à une formule, formules équivalentes : lois de Morgan et du tiers exclu, contraposition, expression de l'implication en fonction de la négation et de la disjonction, formes normales disjonctives et conjonctives.

II.4.3. Circuits élémentaires

Circuits réalisés par des portes logiques (AND, OR, NOT, NAND, NOR, XOR), formules logiques et circuits, additionneur 1-bit et n -bit.

Il s'agit de montrer comment la construction de circuits digitaux suit la formulation logique en calcul des propositions : on donne quelques exemples pour montrer le côté algorithmique. Il ne s'agit pas de faire un cours sur les circuits électroniques ni sur l'architecture des ordinateurs.

II.4.4. Exemples de manipulation formelle de termes et de formules sans quantificateur

Ce paragraphe n'est pas abordé en première année.

III. PROGRAMME DES CLASSES MP ET MP*

III.1. MÉTHODES DE PROGRAMMATION

On présente la méthode d'analyse descendante (c'est-à-dire par raffinements successifs). Même si l'on ne prouve pas systématiquement tous les algorithmes, il faut dégager l'idée qu'un algorithme doit se prouver, à l'instar d'un théorème de mathématiques. On étudie systématiquement la complexité des algorithmes du programme et, sur certains exemples, le lien entre cette complexité et les structures de données. On s'attache à obtenir des étudiants une documentation aussi complète que possible de leurs algorithmes (conditions d'entrée dans un module, conditions de sortie, invariants dans les boucles ou les appels récursifs). Toutes ces notions sont dégagées à partir des algorithmes au programme, sans aucune théorie sur les prédicats ou les invariants de boucles.

III.1.1. Itération

Ce paragraphe ne fait pas l'objet d'un programme spécifique en classe de seconde année.

III.1.2. Récursivité

On donne de nouveaux exemples de structures récursives en liaison avec les notions introduites aux paragraphes III.2.2 et III.2.4.

On insistera sur l'équivalence des différentes formes d'une expression algébrique (arbre, notation préfixée, expression parenthésée).

III.1.3. Diviser pour régner

Ce paragraphe ne fait pas l'objet d'un programme spécifique en classe de seconde année.

III.1.4. Éléments de complexité des algorithmes

Étude des récurrences usuelles :

$$T(n) = T(n-1) + a.n$$

$$T(n) = a.T(n/2) + b$$

$$T(n) = 2.T(n/2) + f(n)$$

Utilisation de la notation O .

Notion de taille de données, évaluation du nombre d'opérations (calculs, comparaisons...) nécessaires à l'exécution et de l'encombrement mémoire.

Notion et exemples de complexité logarithmique, polynomiale, exponentielle (comparaison de temps d'exécution).

Exemples de calculs de complexité dans le cas le plus défavorable, dans le cas moyen (tris, recherches dans un tableau ou dans un arbre, calcul de PGCD).

Comparaisons de complexités de divers algorithmes pour résoudre un même problème : calcul des puissances, évaluation de polynômes, suites récurrentes (méthode récursive simple, méthode récursive avec stockage des résultats intermédiaires, méthode itérative), recherche dans un tableau ordonné (recherche linéaire, recherche dichotomique).

par exemple pour le tri par insertion.

par exemple pour la recherche dichotomique.

notamment pour la méthode « diviser pour régner ».

On se limite à dénombrer les opérations nécessaires pour évaluer le temps d'exécution. Tout autre considération est exclue du programme.

Pour le cas moyen, on traite un ou deux exemples où l'on peut mener simplement un calcul explicite grâce à l'analyse combinatoire et dans certains cas on se contente de notions intuitives sur la probabilité de répartitions des données.

En particulier, on mettra en évidence sur des exemples le compromis souvent nécessaire entre temps de calcul et occupation de la mémoire. On montrera comment le stockage de résultats intermédiaires peut diminuer la complexité d'un algorithme.

III.2. STRUCTURES DE DONNÉES ET ALGORITHMES

Il s'agit de montrer l'influence des structures de données sur les algorithmes et les méthodes de programmation. On met notamment en parallèle les structures récursives des types de données et des programmes qui les manipulent. Les algorithmes sont présentés au tableau, en étudiant, dans la mesure du possible, leur complexité (dans le cas le pire et/ou en moyenne). Certains de ces algorithmes font l'objet d'une programmation effective : les programmes correspondants doivent rester clairs, courts et précis.

III.2.1. Listes et Piles

Ce paragraphe ne fait pas l'objet d'un programme spécifique en classe de seconde année.

III.2.2. Arbres

Certaines définitions n'étant pas unifiées, il convient, pour l'évaluation, de préciser systématiquement la définition choisie et d'attirer l'attention des étudiants sur ce choix.

Définition récursive du type arbre binaire ; nœuds, feuilles, hauteur. Relation entre le nombre de nœuds et le nombre de feuilles.

On peut utiliser la notation :

$$\text{Arbre} = \text{feuille} + \text{Arbre} \times \text{nœud} \times \text{Arbre}.$$

Arbres équilibrés ; hauteur en $\log n$ dans le cas d'un tel arbre de n feuilles.

Calculs récursifs du nombre de nœuds, de la hauteur, du nombre de feuilles.

Insertion et suppression dans un arbre. Arbres de recherche.

Il faut insister sur la simplicité de l'écriture récursive.

Arbres n -aires.

On ne parlera pas des arbres AVL, 2-3, 2-3-4 ou bicolores.

Représentation des expressions arithmétiques par des arbres. Parcours préfixes, infixes et postfixes d'arbres. Application à l'impression préfixe, infixes ou postfixes d'expressions arithmétiques.

Cette partie interagit avec le paragraphe 4 du chapitre logique.

III.3. AUTOMATES FINIS

Il s'agit ici de présenter un modèle élémentaire de calculateur. On insiste sur cet aspect en présentant les automates comme des systèmes simples réagissant à des événements extérieurs.

Automates finis déterministes et non-déterministes. Algorithme de déterminisation.

On donne une représentation graphique des automates.

Langage reconnu par un automate. Exemples de langages non reconnus par un automate.

On présente à ce propos les définitions de base : alphabet, mot, langage. On n'impose pas un énoncé du lemme de l'étoile.

Expression rationnelle. Langage associé.

On précise bien la différence entre expressions rationnelles et langages.

Automates avec transitions instantannées (ou ε -transitions). Équivalence avec les définitions précédentes.

Propriétés de fermeture des langages reconnus par automate : complémentation, intersection, union, concaténation et itération.

On insiste évidemment sur l'aspect constructif de ces propriétés de fermeture.

Théorème de Kleene.

Démonstration non exigible.

III.4. NOTIONS DE LOGIQUE

Le but de cette partie est de familiariser progressivement les étudiants à la différence entre syntaxe et sémantique, ceci 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 ne sont pas au programme.

III.4.1. Éléments de base du calcul propositionnel

Ce paragraphe ne fait pas l'objet d'un programme spécifique en classe de seconde année.

III.4.2. Fonctions booléennes

Ce paragraphe ne fait pas l'objet d'un programme spécifique en classe de seconde année.

III.4.3. Circuits élémentaires

Ce paragraphe ne fait pas l'objet d'un programme spécifique en classe de seconde année.

III.4.4. Exemples de manipulation formelle de termes et de formules sans quantificateur

Différence entre syntaxe abstraite (ou arborescente) et valeur d'une expression arithmétique. Évaluation d'une expression arithmétique (avec les opérateurs $+$, $-$, $*$, $/$). Exemples de formules sans quantificateur écrites avec des symboles de prédicat unaires ou binaires ; interprétation.

On illustre la différence entre syntaxe et sémantique, expression formelle et interprétation. C'est l'occasion d'une familiarisation avec du calcul formel tout à fait élémentaire (on peut donner en exemple la dérivation formelle).