Révisions pour les vacances
Récapitulatif MP2I

Guillaume Rousseau
MP2I Lycée Pierre de Fermat

2023-2024

Ce document est un guide de révision pour les vacances, afin d'arriver bien préparés pour la MPI.

Conseils d'utilisation

— Lisez ce document en entier au début des vacances, et déterminez les notions que vous pensez ne pas maîtriser sur le bout des doigts, histoire d'évaluer la quantité de travail que vous prendront vos révisions.

— Au moins deux semaines avant la rentrée, commencez les révisions. Revenez chapitre par chapitre sur le cours, et révisez comme vous en avez l'habitude (fiches, recopie du cours, exos, etc...), en repassant en priorité sur les notions qui vous posent le plus de problèmes.

— Ces deux semaines peuvent être un bon moment pour revenir sur les TP que vous n'avez pas pu finir, ou bien ceux où vous n'avez pas fait les parties bonus.

Dans la suite, le sigle **YFMD** signifie "les yeux fermés et les mains dans le dos", il indique une fonction/un algorithme à pouvoir écrire sans réfléchir, et dont il faut comprendre le fonctionnement de A à Z. C'est d'autant plus vrai si vous passez en MPI* : vous ne pouvez **pas** arriver en spé sans savoir ce qu'est l'exponentiation rapide, un parcours de graphe, l'algorithme de Rabin-Karp, etc...

1 Premiers programmes en C

— Connaître la syntaxe de base du C

— Savoir compiler un programme avec GCC

— Preuves de correction **propres** avec invariant de boucle.

2 Encodage des nombres

— Principe d'écriture en base B : savoir montrer l'existence, et l'unicité à longueur fixée.

— Algorithmes de lecture et d'écriture en base B en $\mathcal{O}(n)$ (avec n le nombre de chiffres).

— Bien maîtriser le binaire : savoir compter, passer du binaire au décimal et inversement, savoir faire des additions, etc...

3 Pointeurs, tableaux, structures

— Syntaxe C pour les pointeurs, les tableaux, les types struct (avec et sans pointeurs)
— Utilisation de malloc et free
— Compilation avec plusieurs fichiers
— Modèle mémoire : la pile / le tas, savoir représenter la mémoire d'un programme à un moment donné de l'exécution
— Lecture et écriture dans des fichiers
— Algorithmes de base sur les tableaux (chercher le maximum/minimum, l'indice du maximum/minimum, etc...) **YFMD**

4 Complexité

— Définition de la complexité d'un algorithme / programme
— Notations de Landau
— Calcul de la complexité : en utilisant un variant de boucle, en comptant le nombre de passages de boucle, etc...
— Complexités classiques : exponentiation rapide, dichotomie, recherche linéaire dans un tableau, algorithmes de tri, etc...
— Algorithmes de tri : insertion, sélection, fusion, rapide **YFMD**

5 Structures de données

— Structure de données abstraite / concrète (et la différence entre les deux)
— Structures abstraites : piles, files, ensemble, dictionnaire
— Implémentations en C. A savoir faire **YFMD** : insertion / suppression dans une liste chaînée ou doublement chaînée.
— Dictionnaires : pas besoin de savoir réimplémenter par vous même une table de hachage mais il faut pouvoir en expliquer le principe clairement.

6 Premiers programmes en OCaml

— Syntaxe de base d'OCaml, typage
— Récursivité terminale : savoir à quoi ça sert, pouvoir identifier si une fonction est récursive terminale ou pas, et savoir écrire des fonctions récursives terminales en utilisant des fonctions auxiliaires avec accumulateurs

7 Ordre et induction

Il n'est pas attendu que vous maîtrisiez les aspects théoriques de ce chapitre. Le plus important est que vous sachiez faire des **preuves par induction structurelle** sur des listes, des arbres, des formules, etc... Il est quand même nécessaire d'être bien au point sur les notions suivantes qui serviront l'année prochaine, aussi bien en informatique qu'en mathématiques :

— Relation binaire, réflexivité, symétrie, antisymétrie, transitivité
— Clôture réflexive transitive d'une relation

8 Structures de données arborescentes

— Arbres généraux, arbres k-aires, arbres stricts, arbres complets

— Nœuds, feuilles. Bien comprendre la différence entre un nœud et un sous-arbre, et entre un nœud et son étiquette.

— Arbres binaires de recherche : savoir faire une recherche, une insertion, une suppression **YFMD**. Connaître les complexités

— Arbres rouge-noir : connaître la définition par cœur, savoir que ça permet de garantir une hauteur logarithmique en la taille. Sur des exemples, pouvoir corriger les défauts après une insertion ou une suppression.

Quelques algorithmes à savoir écrire **YFMD** :

— Calculer la hauteur et la taille d'un arbre

— Suivre un chemin dans un arbre de la racine vers un nœud

— Parcours préfixe/infixe/postfixe, récursivement ou avec une pile

— Parcours en largeur avec une file

9 Logique propositionnelle

— Variable propositionnelle, formule du calcul propositionnel

— Table de vérité

— Règles de calcul (lois de De Morgan, distributivité,...)

— Problème SAT

— Modélisation de problèmes comme des formules

— Mise sous forme normale conjonctive/disjonctive d'une formule

— Algorithme de Quine

10 Schémas algorithmiques

— Problème d'optimisation, vocabulaire associé

— Stratégies : Glouton, Diviser pour régner, programmation dynamique, backtracking

— Savoir dans quelle situation utiliser quelle stratégie

— Programmation dynamique : Il faut savoir passer de la formule récursive à un algorithme de bas-en-haut ou de haut-en-bas, pouvoir évaluer la complexité, et pouvoir modifier les algorithmes pour permettre la reconstruction d'une solution

Certains problèmes et algorithmes sont ultra-classiques :

— Problème des intervalles disjoints : étant donné des intervalles, en trouver le plus grand nombre qui soient disjoints 2 à 2. Algorithme glouton en triant par date de fin croissante

— Rendu de monnaie : algorithme glouton (sur certains systèmes), prog. dyn.

— Problème du sac à dos : aucun glouton ne fonctionne, on peut trouver des cas où le glouton est arbitrairement mauvais. Prog. dyn.

11 Algorithmique du texte

— Algorithme de Boyer-Moore : règle du mauvais caractère à connaître et à savoir implémenter (algorithme de Boyer-Moore-Horspool).
— Rabin-Karp
— Codage de Huffman et décodage
— Codage de LZW et décodage en reconstruisant le dictionnaire au fil de la lecture

12 Bases de données

— Tables, attributs, enregistrements
— Clé primaire, clé étrangère
— Modèle entité-association, savoir passer d'un diagramme EA à un schéma de base de données

Syntaxe du SQL :

— SELECT, FROM, WHERE, conditions booléennes
— ORDER BY, LIMIT, ASC/DESC, OFFSET
— JOIN, ON, LEFT JOIN
— Agrégats : GROUP BY, HAVING, fonctions d'agrégation (min, max, count, sum, ...)
— Renommage de tables et d'attributs. Ex : SELECT count(*) AS c, name FROM table t1 GROUP BY name

13 Graphes

Le chapitre des graphes est particulièrement riche en vocabulaire. Voici une liste de termes que vous devez connaître. Pour chacun, il faut pouvoir l'expliquer en langage naturel, mais aussi en donner une définition mathématique précise, et également être capable de le représenter par un dessin / un schéma.

— Graphe orienté, non-orienté, arête, arc, voisin, successeur/prédécesseur.
— Voisinage, degré, voisinage entrant/sortant, degré entrant/sortant
— Chemin, chaîne, composante connexe, composante fortement connexe
— Graphe complet / clique
— Graphe biparti
— Coloration de graphe
— Tri topologique
— Graphes pondérés
— Représentations : par matrice d'adjacence, par listes d'adjacence
— Plus court chemin

Quelques algorithmes classiques à connaître **YFMD** :

— Parcours de graphe (profondeur et largeur), recherche de plus court chemin dans un graphe non-pondéré, recherche des composantes connexes dans un graphe non-orienté, détection de cycles impairs
— Algorithme pour trouver un tri topologique dans un graphe orienté acyclique
— Algorithme de Floyd-Warshall, algorithme de Dijkstra