**MASTER INFORMATIQUE PARCOURS ALGO - DIAPO 1**

Présentation du parcours : que fait-on en algo ?

Informatique fondamentales : bases solides pour débutants:

* 1933 que peut-on calculer (machine de turing = ancêtre de l'ordinateur)
* graphes - structures - algorithmique
* 1936 la logique pour faire de l’informatique (Jon von Neumann)
* cryptographie - algorithmique - protocoles - calcul formel (Adi shamir)
* 1971 complexité - NP-complétude (pas réussi depuis des générations)- comment faire en pratique - RO - IA (Leonid Levin)

Vie professionnelle : Pour quoi faire ? débouchés ?

De la théorie à la réalisation - programmation de haut niveau

Algorithmique spécialisée - recherche opérationnelle - informatique théorique

Maîtriser l’évolution de la discipline pas des connaissances fondamentales

Prérequis : goût pour manipuler l’abstraction, les raisonnements, les algorithmes

Travail : intense mais concentré (moins de travaux de groupe mais plus intellectuel), quelques projets et réalisations, stages,TER …



Débouchés :

Métiers de la recherche : Chercheur et enseignants chercheur

Métiers de la recherche et développement : Ingénieur de recherche

Métiers du management et du développement : responsables de projets

Comment réussir

Notre capital est ce que nous avons. (ce n’est pas les notes, plutôt la compréhension)

Importance relatives des notes : **trouvez ce que vous aimez**

Pas d’impasse

Comment travailler ? Privilégié l’efficacité, aller en cours et TD, relire chaque semaine - ne pas garder du travail pour les révisions

Intéractions et problèmes éventuels :

1. le(s) prof(s) / responsable de l’UE (suivi)
2. responsable de parcours (suivi personnalisé)

(1’) problèmes administratif : secrétariat du master

(2’) problème administratif : responsable du master

Méthode : discussions / emails.

**Nous sommes impliqués.**

**DIAPO 2**

Romain Lebreton

Mini cours (L2, M1,M2 & bonus)

aspects théoriques et pratiques

présentation des sujets de recherche de l’équipe ECO

calcul exact (son domaine):

les opérations exates réduit à la multiplication : pgcd factorisation etc

outils : GMP, Sage, Matlab

objectif : aller plus vite

autre domaines qui utilisent le calcul exact:

cryptographie : discrete logarithm computation; truc sécurisés etc etc

s’assurer chiffrement est sécurise : étude théorique (pour casser il faut résoudre pb NP-complet) mais en pratique parfois peut casser; donc aussi cas pratiques

combinatoire

Calcul numérique (quand les chiffres devient TRES grand) :

les trillions premieres décimales de pi

en robotique : l’quilibre d’un robot parallèle conduit par cable

Mulitiplcation d’entier et polynomes <>

SCREEN DIAPO 4/32

* 5/32

mtn on fait de la multiplication de polynomes

on va parler de ⅔ grands algorithme polynomiales

exemple de multiplication polynomiale : on fait 16 multiplication et 4 additions

-> même si c’est naïf, sur de spolynomes de petite taille on peut pas forcément faire mieux

en général multiplication de polynomes de degré n en O(n^2) avec arithmétique(+,-,x)

page 7

Estimation de temps

contexte : 1000 milliards décimal de pi

equivalent calculatoir de multiplier des entiers avec 10^12 chiffres

la vitesse d’un ordi ~= 1GHz donc environ 10^9 op/s

avec l’algo naif : O(n^2)

(10^12)^2op donc s = 10^15 s (TROP D’ANNEES)

première approche : kartsuba

avec un polynome de degré 1, on peut réduire de 4 multiplication et 1 addition à l’approche naïve

on passe à 3 multiplications et 4 addition/soustraction avec karatsuba

l = low

h = high

-> assigné à des bouts du polynomes pour utiliser “l’astuce” de karatsuba

vérifier estimation du diapo

Karatsuba - Divide & Conquer

O(n^log2(3)) on doit fait 3x plus de calcul (?)

il faut pas que utiliser karatsuba, jusqu’à une plage de petite taille il faut faire naïf et après karatsuba

dommage tout le temps est passé à faire l’allocation mémoire : en pratique il faut regarder ça (en théorie c’est passé au second plan)

exercice de faire la multiplication sans augmenter espace dans la mémoire

DIAPO 13 : chaque carré à chaque itération fait 3 appels récursifs etc (création de fractales si fait infiniment et on voit que plus efficace que classique, rmq : ressemble à une fractale mais pas vraiment car ça s’arrête)

fractales ont des dimensions fractionnelles ( ici c’est log2(3) )

Transformée de fourier (discrète ici j’imagine) :

concentre sur le cout d’évaluation/interpolation (cf diapo4)

il faut que les puissances de E choisi bouclent -> racine de l’unité (diapo 17)

la transformation peut se faire en temps quasiment linéaire

on va chercher les racines de l’unité dans les Z/pZ (p premier) diapo18

faire un demi-tour (E^n/2) , c’est comme être à -1

2 choses de plus : 2 vidéos sur yt sur la transformée de yt (il va nous filer les liens sinon c’est veritaserum et un autre type j’ai oublié le nom)

(petite aparté log \* de n c’est log de log de log de n)

toute cette théorie est très présente dans notre ordinateur (diapo30)

les aspects téchniques pas discussés

QUESTION :

* le grand O : à la constatne multiplcative près donc parfois TRES grand donc dur à implémenter

**DIAPO 3 - PAVAGES APÉRIODIQUES**

Victor Poupet - chercheur au LIRMM et enseignant à l’IUT, cours de M2 (certaines fois)

**Présentation du problème :**

Origine du problème :

Début du XXe sciecle

* Axiomatisation des mathématiques
* Formalisation de la logique
* Recherche de procédures automatiques de décision

Théorème (1935-1970) : Si

Années 60 :

* Travaux de Hao Wang
* Classe AEA (?)
* Équivalence entre problèmes de logique et problème

Tuiles de Wang

* Une tuile est un carré unité dont les bords ont une couleur
* Un ensemble fini de tuiles

Définition : Un pavage du plan est la placement de copies de tuiles

**Réflexion préliminaire**

* Un ensemble de tuiles ne pave pas le plan → semi-décidable (Proposition : Si on peut paver des carrés arbitrairement grands, on peut paver le plan)
* Un ensemble de tuile pave le plan périodiquement → semi-décidable
* Si tous les ensembles de tuiles qui pavent le plan aussi périodiquement, alors ont peut décider le problème du pavage de plan

Wang pensait que c’était le cas

* Berger (1964)

**Sous shifts**

Tuiles de Wang faciles à définir mais pas pratiques à utiliser → sous shifts de type fini

* plan discret
* ensemble fini de décorations
* Configuration : associe à chaque cellule une décoration
* Ensemble fini de motifs interdits

Définition : Un pavage est une configuration ne contenant aucun motif interdit.

Informellement :

* Un jeu de tuiles est un ensemble de règles qui peuvent être vérifiées localement
* Il existe une taille N telle que si toute la fenêtre de taille N x N est correcte alors la configuration est un pavage

Proposition ; Le domino problem avec des tuiles de Wang est équivalent à

**Définition**

Une configuration est périodique si elle est invariante par translation d’un vecteur non nul.

Un configuration est bi-périodique si elle est périodique selon deux vecteurs indépendants .

Un jeu de tuiles est apériodique s’il admet un pavage

**Quelle périodicité ?**

Proposition : Si un ensemble de tuiles admet un pavage périodique, alors il en admet un bi-périodique.

Remarque : un pavage bi périodique a une période verticale et une horizontale.

→ Un ensemble de tuiles est apériodique s’il pave le plan mais n’admet aucun pavage horizontalement et verticalement périodique

**III) Un jeu de tuiles apériodique**

**Exemples pour s’inspirer**

Robinson 1970. Idées :

* Une configuration qui contient des carrées de taille arbitraire disjoints ne peut pas être bi-périodique.
* Chaque carrée est lié à un carrée plus grand

**Idée générale**

Idées :

* On dessine des carrés
* On les connecte
* On les organise en groupe 3x3 pour former un plus grand carré
* On continue récursivement

**Lignes bleues**

* Orientées (intérieur / extérieur)
* Ne peuvent

**Rectangles bleues**

* Rectangle fini
* Ligne infini
* X Demi-tour X
* X Coin infini X

X On suppose qu’il n'apparaissent pas

**Carrés**

* On ajoute des diagonales
* Elles contournent les carrés internes

→ Les rectangles deviennent carrés.

**Les bras**

On étend des bras des coins de chaque carré

**Alignement**

Proposition : des carrés connectés par des bras ont la même taille

Preuve par l’absurde en considérant le plus petit mal

**La grille de départ**

* Il faut

**Induction**

On veut des carrés arbitrairement grands :

**Les groupes de 9**

Numérotation modulo 3

Les bras du centres doivent croiser exctemet une ligne bleue

Cette ligne bleue doit être

**Au centre**

Chaque carré

**Résumé de la construction :**

* Carrés bleue
* Petits carrés pour initialiser
* Bras pour former

**DIAPO 4**

recherche opérationnelle

4 grande branches:

eux sont plutôt ça : spécialistes d’optimisation : engin d’optimisation complexe, etc…

developpeur fullstack (qui peuvent tout faire) :

DevOps utiliser tous les outils pour mettre les applications à disposition des clients (CI/CD)

Analyste d’affaires (consiste à aller voir les clients, comprendre les besoins et les aider etc. Interface entre équipe de dev et clients)

ils vont nous montrer ce qu’ils font concrétement avec des Use Cases:

-> premier use case (A) : concevoir l’edt de techniciens qui doivent réparer des trucs dans des batiments sur une zone géographqiue qui peut être grande

-> deuxième use case (B): équivalent des vel’ib à londres

**A - workforce routineg and scheduling problem**

leurs besoins : une application qui permet de faciliter aux planificateurs la construction des edt de leurs employés

leur solution c’est FSO - Dahsboard (page 7) c’est de l’aide à la décision!! principalement. Ils calculent les indicateurs et doivent bien les choisir.

ils permettent aussi aux utilisateurs d’avoir des solutions qui sont aussi calculées automatiquement -> algo d’optmisation c’est ce qu’on nous demande principalement.

il faut un algo qui répode suffisamment rapidement mais qu’il dise pas que des conneries

page 14 -> formalisation du problème d’optimisation. On entend les clients blablater et nous notre objectif c’est de le modéliser pour pouvoir le résoudre

idée c’est qu’on parcours pas toutes les solution pour trouver la meilleur, il faut trouver formuler un objectif bien précis pour minimiser le nombre de solutions et essayer de trouver (se rapprocher) de l’optimal.

il faut trouver un moyen de donner une solution qui soit bonne sur plusieurs des critères (être au moins moyen partout)

problème NP-difficile : on peut pas appliquer directement les solutions de la rechercher (car recherche est à une échelle trop petite) il faut utiliser et adapter.

**B - A pick-up and Delivery Problem**

page 21 : les planificateurs de serco ont besoin d’assurer la qualité

page 22 : Detecter qulles zones sont en train de devenir critiques dans un futur proche.

Tfl a besoin de minimiser les stations ou c’est critique

23 : décision poussée par les données

24: répartir les ordres aux techniciens

des edt optimisés sont présentés sur le Gantt chart

ça générer un plan optimal pour:

* la redistrinution de vélos
* la récolte pour la maintenace/réparation
* des réparations sur site

25 : sous la capuche (en dessous de l’interface graphique) : le workflow

(ne pas oublié c’est du machine learning)

26 bcp de spcificités à prendre en compte :

-> les requis des utilisateurs

-> identifier les problèmes

-> solution : particulier ici pcq c’est pas une solution générale, il faut constamment s’adapter aux uitlisations faites par les utilisateurs.

Heuristique constructive : simple et efficace.

Donne une solution suffisamment grande (correspond assez aux demandes)

eux c’est

détails de l’algorithme :

coupe l’horizon temporel en temps discret

pour les résultats : le client n’a pas donné de chiffres car confidentiel MAIS disent que ça marche et améliorent les capacités jusqu’à 30%

gagné les pros de la ro 2018

et 3e pour IBM think 2018

grosse différence entre théorie et pratique : on peut toujours faire mieux mais il faut s’arrêter au moment ou on est suffisamment bon en qualité et en temps raisonnable.