IFT2125 - Introduction à l'algorithmique

Introduction

Pierre McKenzie

DIRO, Université de Montréal

Automne 2017

Documentation

Livre obligatoire :

- Brassard et Bratley, *Fundamentals of algorithmics*, Prentice Hall 1996.

Autres :

- Cormen, Leiserson, Rivest, *Introduction à l'algorithmique*, 1994 ou +. Édition anglaise de 2009 comporte 4ième auteur, Stein
- Kleinberg, Tardos, *Algorithm Design*, 2006
- Transparents, notes, références ponctuelles placées sur Studium
- Web
- Bibliothèque (réserve, nombreux bouquins)

Évaluation

Devoirs :

- 4 ou 5 devoirs d'au plus 4 questions
- chaque question évaluée sur 10 indépendamment de sa difficulté
- équipes de 2 recommandées

Examens intra et final :

- Livre fermé
- Final cumulatif

Barême :

- Intra 30%, final 40%, devoirs 30%.
  Seuil à 40%.
- Doctorants en examen prédoctoral : intra 40%, final 60%.
  Pour les doctorants : les devoirs ne comptent pas.

Auxiliaire d'enseigmenent

Teaching Assistant

Maëlle Zimmermann
maelle.zimmermann@umontreal.ca

Tâches :

- anime les séances de travaux pratiques
- corrige les devoirs
- répond aux questions
- disponible sur rendez-vous

Tasks:
facilitates practice sessions corrects homework
answers questions
available by appointment

BB = Livre de Brassard et Bratley

Utile d'avoir vu

Préalable IFT1065 - Mathématiques discrètes :

- Induction mathématique [BB 1.6]
- Logique, propositionnelle, des prédicats [BB 1.4.5]
- Permutations, combinaisons [BB 1.7.3]
- Arithmétique modulaire, polynômes
- Définitions de $O(f(n))$, $\Omega(f(n))$, $\Theta(f(n))$ [BB 1.7.2, 3.2, 3.3]
- Récurrences simples et linéaires homogènes [BB 4.7.2]

Cours de prog. et concomitant IFT2015 - Structures de données :

- Recherche dichotomique
- Quelques tris
- Python (?)

Concomitant IFT1978 - Probabilités et statistique :

- Probabilités de base (BB 1.7.4)

Prerequisite IFT1065 - Discrete Mathematics: Mathematical Induction [BB 1.6]
Logic, propositional, predicates [BB 1.4.5] Permutations, combinations [BB 1.7.3] Modular arithmetic, polynomials (F (n)), Θ (f (n)) [BB 1.7.2, 3.2, 3.3]
Simple and linear homogeneous recurrences [BB 4.7.2]
Prog. and concomitant IFT2015 - Data Structures: Dichotomous research Some sorting
Python (?)
Concomitant IFT1978 - Probability and statistics: Basic probabilities (BB 1.7.4)

Plan approximatif

Heures de cours	Matière
4	Introduction et exemples (en partie hors livre)
4	Compléments sur les ordres et les récurrences (chapitres 3 et 4)
5	Algorithmes voraces (chapitre 6)
5	Diviser pour régner (chapitre 7)
5	Programmation dynamique (chapitre 8)
5	Exploration de graphes (chapitre 9)
5	Algorithmes probabilistes (chapitre 10)
2	Algorithmes parallèles (chapitre 11)
2	Sujets choisis
TOTAL : 37	

Prendre connaissance du Code d'honneur de l'étudiant du DIRO. En particulier,

- citez toute source d'information utilisée dans vos travaux
- remettre un devoir en équipe engage la responsabilité de l'équipe.

Pour plus d'information sur les règlements de l'université, consultez Intégrité à l'Université de Montréal.

Questions ?

L'algorithmique, c'est quoi ?

- concevoir des méthodes efficaces de résolution de problèmes de calcul
- choisir la méthode appropriée pour un problème donné

Beaucoup d'intelligence au fil des ans consacrée à l'algorithmique !

Exemples

Tri d'un tableau

1. Sélection
2. Insertion
3. Merge sort - par fusion
4. Quick sort - rapide
5. Heap sort - par tas
6. Radix sort - par base
7. Bucket sort - par paquets
8. Alouette sort - par alouettes :-)
9. Tri en parallèle

Exemples

Déterminant d'une matrice $m \times m$

$$\begin{vmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{vmatrix} = x_{11}x_{22}x_{33} - x_{11}x_{32}x_{23} - x_{12}x_{21}x_{33} + \cdots$$

$$= \sum_{\sigma \in S_m} (-1)^{\text{signe de } \sigma} x_{1\sigma(1)}x_{2\sigma(2)} \cdots x_{m\sigma(m)}$$

1. Bête
   Faire la somme des $m!$ termes.
2. Gauss-Jordan
   Amener à la forme triangulaire.
   Multiplier les éléments de la diagonale.
3. Berkowitz (Samuelson)
   Réduire au calcul de puissances de matrices.

Recherche en cours : et le permanent d'une matrice ?

Exemples

Déterminer la primalité

On veut déterminer si $\underbrace{xxxxxxxxxxxx}_{12 \text{ chiffres}}$ est un nombre premier.

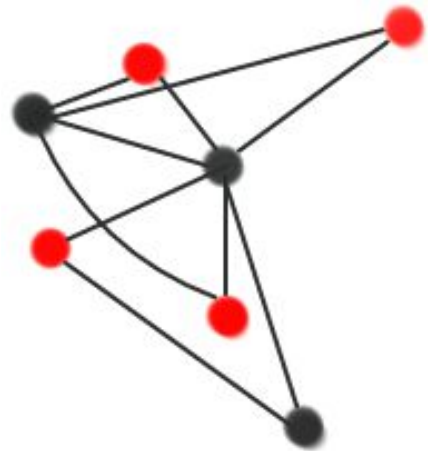
Problème addictif pour mathématiciens.
Problème fondamental pour les cryptographes.

1. Bête
   Essayer $2, 3, 4, 5, 6, 7, 8, \ldots$ éventuellement jusqu'à 10^6
2. Crible d'Erathostènes
   Éliminer tour à tour diviseurs de la liste $2, 3, 4, 5, 6, 7, 8 \ldots, 10^6$
3. Miller-Rabin
   Rapide en acceptant une probabilité d'erreur inférieure à $2^{-1000000}$.
4. Agrawal-Keyal-Saxena (2002)
   Temps polynomial avec certitude mais degré élevé.

Exemples

Stable maximum

Donnée : graphe (S, A)



Déterminer : un stable de taille maximum.

1. Bête
   Essayer tous les $E \subseteq S$ en ordre décroissant de taille.

Recherche en cours : trouver une méthode qui n'est pas bête.

- concevoir des méthodes efficaces de résolution de problèmes de calcul
- choisir la méthode appropriée pour un problème donné

Au fait, qu'est-ce qu'un problème ?

Problèmes et exemplaires (“problems and instances”)

Un problème demande de

- calculer une valeur
  - ex : tri, déterminant
- ou de répondre à une question oui/non
  - ex : le nombre est premier ?, il existe un stable de taille k ?

à partir de données fournies en entrée.

Un problème possède une infinité d'exemplaires

- ex : tri
  - tableau 1, tableau 2, etc.
- ex : nombre premier
  - $0, 1, 2, \ldots$, etc.
- ex : stable
  - $\ldots$  $\ldots$

Le temps

Un algo A résout un problème P.

- A peut prendre un temps différent sur chaque exemplaire
- Qu'est-ce alors que le temps d'exécution de A ?

Simplification fréquente :

- paramétriser en fonction de la taille n des exemplaires

Et qu'est-ce que taille n d'un exemplaire ?

Ultimement, n = nombre de bits utilisés pour coder l'exemplaire.

En pratique, dépend de P et du but de l'analyse :

- ex : tri
  - souvent n = nombre d'éléments du tableau
- ex : évaluer une expression comme $((28783 + 410)/192) \times 159$
  - souvent n = nombre d'opérandes, ici $n = 4$
  - parfois n = nombre total de chiffres, ici $n = 14$.
- ex : stable
  - parfois n = nombre $|S|$ de sommets du graphe (S, A)
  - parfois $n = |S|^2$ = nombre de bits requis pour représenter la matrice d'adjacence du graphe

Diverses mesures de temps

- Pire cas (mesure la plus utilisée) :
  $t(n) = \max_{e \text{ exemplaire de taille n}} \{\text{temps que prend } A \text{ sur } e\}$

- En moyenne
  $$t(n) = \frac{\sum_{e \text{ exemplaire de taille } n} \{\text{temps que prend } A \text{ sur } e\}}{\text{nombre d'exemplaires de taille } n}$$

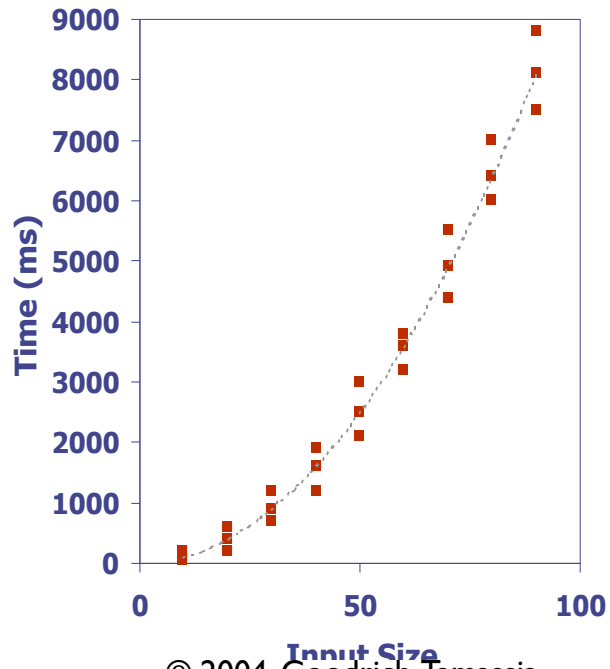
- Amorti (cas d'un A qui agit sur des données externes)
  Moyenne sur plusieurs appels successifs à A

- Espéré (cas d'un A qui utilise l'aléat)
  Espérance mathématique du temps avant l'arrêt.

4 slides empruntées de Sylvie Hamel

Méthode 1: Études expérimentales

- Implémenter l'algorithme en Java (ou autre)
- Faire fonctionner le programme avec des entrées de taille et de composition différentes
- Utiliser une méthode pour obtenir une mesure réelle du temps d'exécution
- Dessiner le graphique des résultats



Limitation de cette méthode

- On doit implémenter l'algorithme
  - On veut connaître la complexité en temps d'un algorithme avant de l'implémenter, question de sauver du temps et de l' $$$$
- Les résultats trouvés ne sont pas représentatifs de toutes les entrées
- Pour comparer 2 algorithmes différents pour le même problème, on doit utiliser le même environnement (hardware, software)

Méthode 2 : analytique

En comptant les opérations élémentaires :

- Opérations de base effectuées par l'algorithme
  - Évaluer une expression
  - Affecter une valeur à une variable
  - Appeler une méthode
  - Incrémenter un compteur
  - etc.
- Indépendantes du langage de programmation choisi
- On suppose que chacune prend un temps d'exécution constant

Count basic operations

Compter les opérations élémentaires

En inspectant le pseudocode d'un algorithme, on peut déterminer le nombre maximum d'opérations élémentaires exécuté par un algorithme, comme une fonction de la taille de l'entrée

Algorithm *arrayMax*(A, n)	# operations
currentMax ← $A[0]$	2
for $i \leftarrow 1$ **to** $n - 1$ **do**	2 (n-1)
if $A[i] >$ *currentMax* **then**	2 (n-1)
currentMax ← $A[i]$	2 (n-1)
return *currentMax*	1
	total = 6n - 3

Du nombre d'opérations élémentaires au temps

From the number of elementary operations to time

- On cherche $t(n)$ = temps en pire cas, lorsque n = taille du tableau
- L'algo *arrayMax* exécute $6n - 3$ opérations élémentaires en pire cas
  - a = temps d'exécution de la plus rapide opération élémentaire
  - b = temps d'exécution de la plus lente opération élémentaire
- Alors le temps de calcul $t(n)$ de *arrayMax* vérifie :

$$\boxed{\forall n,\ a \times (6n-3) \leq t(n) \leq b \times (6n-3)}$$

- Souvent le comportement asymptotique suffit. Ici :
  - de $\leq$ on tire $t(n) \in O(n)$
  - de $\geq$ on tire $t(n) \in \Omega(n)$
  - $t(n) \in O(n) \cap \Omega(n)$ d'où $t(n) \in \Theta(n)$

- arrayMax est particulier en ce que le pire cas est facile à identifier

Autres exemples de l'intelligence consacrée à l'algorithmique

Other examples of intelligence devoted to algorithms

Multiplication de grands entiers

Multiplication of large integers

Plusieurs méthodes pour calculer $xxxxxxxxxxxx \times yyyyyyyyyyyy$

Several methods for calculating

1. Classique classic
2. À la “façon russe” In the "Russian way"
3. À la “façon arabe” In the "Arabic way"

Multiplication de grands entiers

Multiplication of large integers

Plusieurs méthodes pour calculer $xxxxxxxxxxxx \times yyyyyyyyyyyy$

Several methods for calculating

1. Classique classic
2. À la “façon russe” In the "Russian way"
3. À la “façon arabe” In the "Arabic way"
4. Récursive recursive

Express Exprimer $xxxxxxxxxxxx = A \times 10^6 + B$

Express Exprimer $yyyyyyyyyyyy = C \times 10^6 + D$

Caclulate Calculer $AC \times 10^{12} + (AD + BC) \times 10^6 + BD$.

Multiplication de grands entiers

Multiplication of large integers

Plusieurs méthodes pour calculer $xxxxxxxxxxxx \times yyyyyyyyyyyy$

Several methods for calculating

1. Classique classic
2. À la "façon russe" In the "Russian way"
3. À la "façon arabe" In the "Arabic way"
4. Récursive recursive

Express Exprimer $xxxxxxxxxxxx = A \times 10^6 + B$

Express Exprimer $yyyyyyyyyyyy = C \times 10^6 + D$

Caclulate Calculer $AC \times 10^{12} + (AD + BC) \times 10^6 + BD$.

5. Interpolation Interpolation

   Calculer $xxxxxxxxxxxx \times yyyyyyyyyyyy$ modulo 2

   Calculer $xxxxxxxxxxxx \times yyyyyyyyyyyy$ modulo 3

   Calculer $xxxxxxxxxxxx \times yyyyyyyyyyyy$ modulo 5

   $\vdots \qquad \vdots \qquad \vdots$

   verifying these congruences

   Calculer $m \leq 2 \times 3 \times 5 \times \cdots$ vérifiant ces congruences.

   Théorème ("des restes chinois") : Ce m (positif) est unique.

   Theorem ("Chinese remains"): This m (positive) is unique.

Plus grand commun diviseur

Greatest common divisor

We search

On cherche $pgcd(xxxxxxxxxxxx, yyyyyyyyyyyy)$.

Ex : $pgcd(140, 98) = 2 \times pgcd(70, 49) = 2 \times 7 \times pgcd(10, 7) = 2 \times 7 = 14$.

Stupid

1. Bête

   Essayer $xxxxxxxxxxxx, xxxxxxxxxxxx - 1, \ldots$ éventuellement jusqu'à 2

   Try possibly up to

2. Euclide

```
def pgcd(a,b):
  while b != 0:
    a,b = b, a % b
  return(a)
```

Recherche en cours : méthode efficace en parallèle ?

Research in progress: effective method in parallel?

Plus grand commun diviseur

Greatest common divisor

How to ensure that

Comment s'assurer que

```
def pgcd(a,b):
  while b != 0:
    a,b = b, a % b
  return(a)
```

is a correct algorithm?

est un algorithme correct ?

rarely simple

- rarement simple
- demande ingéniosité

ingenuity demand

Tirer davantage de l'algorithme d'Euclide

Draw more from Euclid's algorithm

Algorithme d'Euclide étendu

Extended Euclidean algorithm

```
def pgcd_etendu(a,b):
    """ Calcule s, t et pgcd(a,b)=sa+tb """
    s, t, u, v = 1, 0, 0, 1
    while b != 0:                    # invariant : sa+tb=a
        q = a//b
        a,s,t,b,u,v = b,u,v, a-q*b, s-q*u, t-q*v
    return(s,t,a)
```

Transformée de Fourier

Fourier Transform

Étant donnés une matrice M de forme particulière et un vecteur x, on veut calculer les vecteurs Mx et $M^{-1}x$.

Given a matrix M of particular form and a vector x, we want to calculate the vectors Mx and inv(M)x.

1. Bête (stupid)
   Multiplier sans se soucier de la forme particulière. (Multiply without worrying about the particular shape.)
2. Diviser pour régner (Divide and conquer)
   A révolutionné les télécommunications et le traitement des signaux.
   Sous-tend le format JPEG.
   (A revolutionized telecommunications and signal processing. Underpins the JPEG format.)

Appartenance à un groupe de permutations

Belonging to a group of permutations

Rappel : permutations d'un ensemble $\{1, 2, 3, 4, 5, 6\}$ de "points"

Reminder: permutations of a set {1, 2, 3, 4, 5, 6} of "points"

- $\varepsilon = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix} \in S_6$ est la permutation identité
  is the identity permutation

- $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 2 & 6 & 5 & 1 & 3 \end{pmatrix} \in S_6$ est aussi représentée $(145)(36)$
  is also represented

The product of two permutations:

- Le produit de deux permutations :
  $(145)(26) * (134)(256) = (1)(2)(3465) = (3465)$

The inverse of a permutation:

- L'inverse d'une permutation :
  $[(145)(26)]^{-1} = (541)(62) = (154)(26)$

Le problème de l'appartenance

The problem of the membership

data

Donnée : $p, p_1, \ldots, p_k \in S_m$

Déterminer : si $p \in \underbrace{\langle p_1, \ldots, p_k \rangle}$.

Determine if

toutes les permutations engendrées par composition des p_i

all the permutations generated by composition of the p_i's

1. Bête Stupid
2. Intelligent Clever

   Prochains transparents. Next transparencies
3. Super-intelligent Super-smart

   Rapide en parallèle, repose sur 5000 pages de mathématiques.

   Fast in parallel, relies on 5000 pages of mathematics.

Appartenance : l'algo bête

Membership : the stupid algo

$S \leftarrow \emptyset$
$S' \leftarrow \{p_1, p_2, \ldots, p_k\}$

`while` $S \neq S'$
$\quad S \leftarrow S'$
$\quad S' \leftarrow S' \cup \{s * t : s, t \in S\}$

`if` $p \in S$ `then` TRUE `else` FALSE

Appartenance : l'algo intelligent

Coeur de l'algo : tamiser une permutation dans un tableau en construction

Heart of the algo : sifting a permutation in a table under construction

T : tableau $m \times m$ de permutations des points $\{1, 2, \ldots, m\}$

(table; permutations of points)

r : permutation à traiter

(permutation to be processed)

```
tamiser(r)                                   (sift)
    while r ≠ ε
        i ← min{i : i^r ≠ i}        {i ← plus petit point déplacé par r}
                                    (smaller point moved by r)
        j ← i^r                     {j ← le point où r envoie i}
                                    (the point where r sends i (image of i^r))
        if T[i,j] == ε then
            T[i,j] ← r              {insérer r dans le tableau}
                                    (insert r into the table)
        else
            r ← r * (T[i,j])^-1
```

Appartenance : l'algorithme intelligent complet

Membership : the complete intelligent algorithm

Donnée (data) : $p, p_1, \ldots, p_k \in S_m$

Déterminer (Determine) : si $p \in \langle p_1, \ldots, p_k \rangle$.

fill $m \times m$ table T everywhere with ε

for $i = 1, \ldots, k$

    tamiser(p_i) (sift)

`while` there exist q, r in table T such that $q * r$ was never sifted

    tamiser($q * r$) (sift)

`if` tamiser(p) modifies T then

    "p n'appartient pas au groupe $\langle p_1, \ldots, p_k \rangle$" (p does not belong to the group ⟨p1,. . . , pk⟩)

`else`

    "p appartient au groupe $\langle p_1, \ldots, p_k \rangle$" (p belongs to the group ⟨p1,. . . , pk⟩)

Appartenance : l'algorithme intelligent

Membership : the intelligent algorithm

Cet algorithme est-il correct ? ? ?

Is this algorithm correct? ? ?

Appartenance Membership

Principale propriété du tamisage de r, $r \neq \varepsilon$

Main property of sifting r, r != ε

Suppose

Supposons :

- r vient d'être tamisé
  r has just been sifted
- s était le plus petit point non fixé par r
  s was the smallest point not fixed by r
- $T[t, j] =$ fut la dernière entrée examinée lors du tamisage de r.
  was the last entry examined during the sifting of r.

Alors :

Then

- $s \leq t$
- r s'exprime maintenant sous la forme
  r is now expressed in the form

$$T[t, j] * T[t-1, j_{t-1}] * \cdots * T[s+1, j_{s+1}] * T[s, j_s].$$

Preuve : induction sur le nombre de tours du `while` lors du tamisage.

Proof : induction on the number of revolutions of the while while sifting.

Problème de l'ordre d'un groupe de permutations

On l'a gratuitement du tableau T

It is free from Table T

Data

Donnée : permutations $p_1, p_2, \ldots, p_k$

Déterminer : nombre de permutations du groupe $\langle p_1, p_2, \ldots, p_k \rangle$

Detemine number of permutations of group A

form by sifting then by "closing"

former T en tamisant $p_1, p_2, \ldots, p_k$ puis en "fermant" T

$\mathtt{N} \leftarrow 1$

`for` $i = 1, \ldots, m$

$\quad \ell \leftarrow |\{j : T[i,j] \neq \varepsilon\}|$

$\quad \mathtt{N} \leftarrow \mathtt{N} \times (\ell + 1)$

`return` N