26/04~~1~~2/12/2018
ALGO6

1 TD

Enumeration dans les graphes

Exercice 1 -

1 1 - CAE, DB, CE, AE, CA, CB, A, B, C, D, E, DE

2 - CA, CB, CE, AE, CAE, A, B, C, D, E

3 - CB, CE, AB, AE, DB, DE, A, B, C, D, E

4 - A, B, C, D, E, CB, CE,

2 1 - 1: 5, 2: 6, 3: 1, 4: 0, 5: 0

2 - 1: 5, 2: 4, 3: ~~0~~ 1, 4: 0, 5: 0

3 - 1: 5, 2: 6, 3: 0, 4: 0, 5: 0

4 - 1: 5, 2: 2, 3: 0, 4: 0, 5: 0

Exercice 2 -

1 procedure énumerer_codage (Y)

Données : $Y = \{Y_0, \dots, Y_{n-1}\}$ ensemble d'element.

Résultat : Une et une seule visite de chaque partie de Y.

~~appels~~

$O(n)$ ou $O(n^2)$ en fonction des paramètres

for $i = 0$ to $2^n - 1$
- ecrire i en binaire
- convertir le vecteur de bits en une partie p_i de Y
- visiter P

Version recursive

2^n appels

procedure enumerer_visiter_parties (X, Y)
- if $X = \emptyset$
  - visiter (Y)
- else
  - x = choisir (X)
  - enumerer_visiter_parties($X \setminus \{x\}$, Y)
  - enumerer_visiter_parties($X \setminus \{x\}$, $Y \cup \{x\}$)

2- On peut modifier la fonction visiter(); et faire en sorte qu'elle verifie le cardinal de l'ensemble courant.

a) l'algo n'est pas optimal dans le sens ou il créé quand meme tout l'ensemble.
une amelioration serait de ne créer que les parties de cardinal k

algo :

~~if $X = \emptyset$ ou $|X| == k$~~
~~visiter (Y)~~
~~else if $((|X| + |Y|) == k$~~
~~$f(X \setminus \{x\}, Y \cup \{x\})$~~
~~else if $|X| + |Y| > k$~~
~~x = choisir (X)~~
~~$f(X \setminus \{$~~

$EP\text{-}k(X, Y)$

si $X = \emptyset$ alors
L traiter (Y)
sinon
if $(|X| - 1 + |Y| \geq k)$
L $EP(X \setminus \{x\}, Y)$
if $(|Y| + 1 \leq k)$
L $EP(X \setminus \{x\}, Y \cup \{x\})$

⋮

Exercice 3 -

on peut representer l'echequier par un ensemble de sommets. une reine placée à un endroit represente une serie d'arretes reliant les sommets de la ligne, la colonne et les diagonales

Énumération dans les graphes

Concept : Énumération d'un ensemble
Méthode : schéma récursif, élagage

Auteur: Nicolas Gast

Pour résoudre certains problèmes d'optimisation, on est parfois amené à énumérer toutes les solutions possibles afin de trouver la meilleure (on pense en particulier aux problèmes NP-complets). L'algorithme obtenu est alors souvent de complexité exponentielle mais une énumération intelligente *des parties permet souvent d'accélérer grandement le temps d'exécution.*

Ce TD est à rendre par groupe de quatre à la fin de la séance de TD.

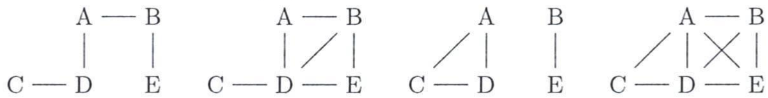
Le but de ce TD est de concevoir un algorithme permettant d'énumérer les stables d'un graphe. Considérons un graphe non orienté $\mathcal{G} = (V, E)$, V l'ensemble des sommets et E l'ensemble des arêtes (edges). Un ensemble $S \subset V$ est un *stable* de $\mathcal{G}$ (*a.k.a. independent set* en anglais) si aucune paire de sommets de S n'est reliée par une arrête. La taille d'un stable est égale au nombre de sommets qu'il contient.

Applications :

- Dans un réseau sans fil des stations peuvent interférer si elles sont trop proches. On modélise l'infrastructure par un graphe de stations (une arête modélisant une interférence entre 2 stations). Une configuration admissible de l'infrastructure est un ensemble de stations actives qui ne sont pas en interférence.
- Montrer que le problème des philosophes vu en système d'exploitation revient à construire des parties stables sur un graphe de processus.

Exercice 1: Échauffement

1. Énumérer les stables des graphes suivants :



(a) (b) (c) (d)

2. Pour chacun des quatre graphes, combien y a-t-il de stables de taille 1, 2, 3, 4 ou 5 ?

Exercice 2: Algorithmes d'énumération

1. Rappeler l'algorithme $\mathcal{EP}$ permettant d'énumérer toutes les les parties d'un ensemble. Quelle est la complexité de votre algorithme ?
2. Modifier cet algorithme afin qu'il énumère toutes les parties de cardinal k, algorithme $\mathcal{EP}_k$
   a) Pensez-vous que cet algorithme est optimal (en quel sens) ? Voyez vous des améliorations à apporter à votre algorithme ?
   b) Modifier la preuve de l'algorithme $\mathcal{EP}$ (vue en cours) afin de prouver $\mathcal{EP}_k$.
3. Modifier l'algorithme $\mathcal{EP}_k$ afin qu'il énumère toutes les parties stables de cardinal k d'un graphe, algorithme $\mathcal{EPS}_k$.
   a) Pensez-vous que cet algorithme est optimal ? Voyez vous des améliorations à apporter à votre algorithme ?
   b) Modifier la preuve de l'algorithme $\mathcal{EP}_k$ afin de prouver $\mathcal{EPS}_k$.
4. Modifier l'algorithme $\mathcal{EPS}_k$ pour calculer le cardinal maximal d'une partie stable d'un graphe graphe, .

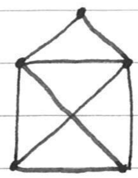
Exercice 3: Les n reines

Dans le jeu d'échec, une reine contrôle toutes les cases situées sur les mêmes lignes, colonnes et diagonales que la sienne. Le problème des n-reines est de trouver toutes les configurations de n reines sur un échiquier (n lignes, n colonnes) telles qu'aucune reine ne soit en prise. Relier ce problème avec ce qui précède.

19/02/2018
ALGO6
(4)

CM **Problemes et complexité -**

Exemple :
circuit eulerien : (chemin)



$G = (X, E)$ graphe non-orienté.
-> Existe-t-il un chemin passant par toutes les aretes une fois.
↳ algorithme en temp polynomial qui decide si ça existe ou non.

Circuit hamiltonien -
Existe-t-il un circuit passant par tout les sommets du graphe une seule fois?
exemple :

: complexité ?

Plus court chemin dans un graphe orienté valué : Dijkstra (n)

Plus long chemin elementaire :
Definition : Un probleme P est dans la classe P s'il existe un algorithme de coût polynomial en la taille des entrées resolvant P.

Definition : Un probleme est ^{n}P si il existe un verificateur du probleme de coût polynomial
(si on trouve une solution, on peut verifier en temps polynomial qu'elle est correcte)

Probleme de base -
SAT : formule, variables booleene, operateur $\vee \wedge \neg$
$\varphi(x_1, \dots, x_n)$ satisfiable.
SAT est dans ^{n}P

Theoreme de Cook -
Tout probleme ^{n}P peut se reduire à un probleme SAT.
-> voir slides (énumeration)

12/02/2018

ALGO6

1

TD

TD 5 - Multiplication rapide : Karatsuba et strassen.

Exercice 1 - Multiplication de polynomes.

1- a) $P + Q = [1, 3, 3, 5]$;

b) $P \times Q = [0, 1, 2, \cancel{\ }, \cancel{\ }, 3, 4]$;

4 6

2- a)

```
int i;
tab t3;
for (i = 0 ; i < t1.lenght && t2.lenght ; i++){
   L t3[i] = t1[i] + t2[j];
}
```

b)

```
int i, j;
tab t3;
for (i = 0 ; i < t1.lenght ; i++){
  | for (j = 0; i < t2.lenght; j++){
  |    L t3[i+j] += t1[i] + t2[j]
  L }
}
```

c) - max.lenght (t_1, t_2)

- lenght $(t_1) \times$ lenght (t_2)

3- a) ~~avec~~

$$(aX + b)(cX + d) = acX^2 + \underbrace{adX + bcX}_{(\text{identite})(x)} + bd$$

Donc $= acX^2 + [ac + bd + (b - a)(c - d)]X + db.$

b) cf question d'avant.

$2^n - 1$

TD 5 : Multiplication rapide: Karatsuba et Strassen

UNIVERSITÉ Grenoble Alpes

Auteur: Nicolas Gast

On est souvent amené à manipuler (et donc multiplier) des entiers de grande taille, par exemple en cryptographie. L'algorithme naïf de multiplication est quadratique en le nombre de bits. L'algorithme de Karatsuba, publié en 1962 par Karatsuba et Ofman utilise une méthode de diviser pour régner afin d'effectuer une multiplication bien plus rapide. L'algorithme de Strassen (1969) utilise une technique similaire pour effectuer la multiplication de matrices. Dans ce TD, on se propose d'analyser ces deux algorithmes et montrer qu'il battent la complexité des algorithmes naïfs.

Depuis, d'autres approches ont encore réduit la complexité, la multiplication de grand entiers repose repose maintenant sur des algorithmes utilisant la transformée de Fourier rapide, et arrivent à une complexité inférieure à $O(n \log n \log \log n)$ (l'algorithme de Schönhage-Strassen ou l'algorithme de Fürer). La multiplication de matrice peut être faite en $O(n^{2.373})$ (voir l'article Powers of Tensors and Fast Matrix Multiplication *de François Le Gall, 2014).*

À cause des constantes cachées dans les algorithmes, ces algorithmes ne sont utilisés que pour des entiers ou matrices de grande taille. Pour la multiplication, Karatsuba est utilisé à partir de 200 bits, Schönhage-Strassen à partir de plusieurs dizaines de milliers de bits et Fürer n'est pas utilisé en pratique. Pour les matrices, Strassen devient rentable pour des matrices de taille supérieure à 100.

Afin d'éviter les problèmes liés à la propagation des retenues, on commencera par étudier l'algorithme de Karatsuba sur des polynômes (Exercice 1), avant de passer aux entiers (Exercice 2).

Exercice 1: Multiplication de polynôme : l'astuce de Karatsuba

Un polynôme $\sum_{i=0}^{n} a_i X^i$ est naturellement représenté par son tableau de coefficients $[a_0 \dots a_n]$, où le $i$ème élément a_i représente le coefficient de X^i. Par exemple, les polynômes $P(X) = X^2 + X + 1$ et $Q(X) = 2X^2 + 1$ peuvent être représentés par les tableaux $[1,1,1]$ et $[1,0,2]$. Il peuvent aussi être représentés par $[1,1,1,0,0,0]$ ou $[1,0,2,0]$ par exemple. Le degré d'un polynôme est égal au plus grand indice d'une case non nulle ($[1,0]$ est de degré 0, $[0,1]$ est de degré 1). La multiplication et l'addition de polynômes sont définies de façon naturelle :

$$(P+Q)(X) = 3X^2 + X + 2 \qquad \text{tableau } [2,1,3]$$
$$PQ(X) = 2X^4 + 2X^3 + 3X^2 + X + 1 \qquad \text{tableau } [1,1,3,2,2]$$

Les degrés de P, Q et $P+Q$ sont 2, celui de PQ est 4.

1. Soit $P = [1,2,3,4]$ et $Q = [0,1,0,1]$.
   a) Que vaut l'addition de $P+Q$?
   b) Que vaut la multiplication PQ ?
2. Algorithmes naïfs.
   a) Écrire un algorithme qui additionne deux polynômes.
   b) Écrire un algorithme qui multiplie deux polynômes.
   c) Quelle est la complexité de vos deux algorithmes ?
3. L'astuce de Karatsuba consiste à remarquer que $bc + ad = ac + bd + (b-a)(c-d)$.
   a) En déduire que

$$(aX+b)(cX+d) = acX^2 + [ac + bd + (b-a)(c-d)]X + db.$$

   b) En déduire que que l'on peut multiplier deux polynômes de degré 1 en effectuant seulement 3 multiplications et 4 additions.

4. Un polynôme P_{2n-1} **de degré** $2n-1$ peut s'écrire

$$P_{2n-1}(X) = A_{n-1}(X)\,X^n + B_{n-1}(X),$$

où A_{n-1} et B_{n-1} sont deux polynômes de degré $n-1$.
Comment calculer A_{n-1} et B_{n-1} depuis la représentation de P_{2n-1} en tant que tableau ?

5. Pour multiplier deux polynômes de degré 2^k-1, on peut utiliser l'astuce de Karatsuba pour se ramener à 3 multiplications de polynômes de degré $2^{k-1}-1$ et 4 additions de polynômes de degré $2^{k-1}-1$ puis opérer récursivement.
   a) En convaincre vos camarades.
   b) Écrire une formule de récurrence pour la complexité de cet algorithme et la résoudre.
   c) Écrire un algorithme utilisant l'astuce de Karatsuba qui prend en entrée deux polynômes de degré 2^k-1 représentés sous la forme d'un tableau de nombres et rend leur produit.
   d) Comment adapter cet algorithme lorsque les degrés de P et Q sont quelconques ?

Exercice 2: Arithmétique sur les entiers : addition et multiplication

Un entier se représente en base b sous la forme $\sum_{i=0}^{n} a_i b^i$, où $a_i \in \{0, \dots, b-1\}$.

1. Quelle est la représentation binaire (en base 2) de 16 ? De 100 ?
2. Écrire un algorithme d'addition de deux entiers représentés sous la forme d'un tableau de chiffres en base b.
   a) Quelle est la complexité de votre algorithme ?
   b) Quelles différences y a-t-il entre cet algorithme et l'algorithme d'addition de deux polynômes ?
3. Écrire un algorithme naïf pour multiplier deux entiers représentés sous la forme d'un tableau de chiffres en base b.
4. Appliquer l'idée de Karatsuba pour concevoir un algorithme de multiplication rapide d'entiers (en base b).
5. Quelle est sa complexité ?

Exercice 3: Multiplication de matrices : algorithme de Strassen

Si $A = (a_{ij})$ et $B = (b_{ij})$ sont deux matrices carrées, leur multiplication est une matrice carrée $C = (c_{ij})$ telle que

$$c_{ij} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

1. Quelle est la complexité de l'algorithme naïf de multiplication (en fonction de n) ?
2. Quelle est la complexité de l'algorithme naïf d'addition (en fonction de n) ?

L'idée de Strassen est de remarquer que :

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{pmatrix},$$

où

$$\begin{aligned}
M_1 &:= (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}),\\
M_2 &:= (A_{2,1} + A_{2,2})B_{1,1},\\
M_3 &:= A_{1,1}(B_{1,2} - B_{2,2}),\\
M_4 &:= A_{2,2}(B_{2,1} - B_{1,1}),\\
M_5 &:= (A_{1,1} + A_{1,2})B_{2,2},\\
M_6 &:= (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2}),\\
M_7 &:= (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2}).
\end{aligned}$$

3. En déduire un algorithme récursif de multiplication de matrice et étudier sa complexité.