Décomposition en sous-problèmes

2023/2024

Plan

Algorithmes gloutons

Diviser-pour-régner

Programmation dynamique

Recherche exhaustive

DS à la rentrée

Le programme du DS de la rentrée porte sur les graphes.

Implémentations en C

Il semblerait qu'il y ait quelques difficultés à implémenter des programmes en C.
Je vous proposerai quelques implémentations durant les vacances pour vous aider à travailler un peu les implémentations en C.

Avant-propos

Dans les diapos suivantes font figurer des informations sur les épreuves d'informatiques dans les différents concours.
Il est possible que les informations qui figurent sur les diapos suivantes changent d'ici à votre passage.
Vous aurez le temps de rentrer un peu plus dans les détails sur les différentes épreuves en deuxième année.

E3A

Écrit mutualisé avec CCINP. Une seule épreuve à l'écrit de 4 heures.
Coefficients variable selon l'école concernée, mais souvent équivalent à l'épreuve de maths, et supérieur à l'épreuve de physique.
À l'oral, chaque école a ses propres oraux.

CCINP

Écrit mutualisé avec CCINP. Une seule épreuve de 4h.
Le coefficient dépend de l'école, mais généralement un peu plus que l'épreuve de maths et de physique chimie (mais il y a 2 épreuves de maths).
À l'oral, 30 minutes de préparation et 30 minutes de passage au plus. Une partie de la note correspond à un exercice papier, et l'autre partie correspond à un exercice sur l'ordinateur qu'il faudra présenter à l'aide du code source.

Mines Télécom

Mêmes écrits qu'à Mines-Ponts : un écrit de coeff 3 de 3 heures, et un écrit de coeff 4 de 4 heures sur un total de 30.
Un oral coeff 8/30 avec 15 minutes d'appropriation du sujet puis un passage de 30 minutes devant le jury. (Il n'y a pas d'oral de physique).

Mines-ponts

À l'écrit, deux épreuves, une coefficient 3 de 3 heures, et une coefficient 4 de 4 heures sur un total de coefficient de 30.
Un seul oral de 3h30 (avec la préparation sur machine). Le coefficient est moins élevé que l'épreuve de physique, et beaucoup moins élevé que l'épreuve de mathématique.

Centrale-Supélec

Un seul écrit de 4h.
Coefficients qui dépendent des écoles, mais pour la plupart des écoles, l'écrit d'info vaut moins qu'un écrit de maths, plus qu'un écrit de physique, mais moins que la physique au total.
À l'oral, un TP d'informatique (coeff 16/100), et un oral de maths-info (coeff 18/100).
L'oral de maths info correspond à 30 minutes de préparation sur ordinateur, et 30 minutes de passage.
L'oral de TP d'info a une durée de 3h.

X

Un écrit d'info (Info C) de 4h qui compte pour un coefficient de 9/39.
Un oral d'info de 50 minutes coeff 22/96.

ENS

À l'écrit, pour toutes les ENS, il y a informatique C aussi de 4h.
À l'oral, pour toutes les ENS : oral d'informatique fondamentale, et de TP. L'oral d'informatique fondamentale commence par 30 minutes de préparation puis 30 minutes de passage. Le TP a une durée de préparation de 3h30 pour 23 minutes de présentation à l'oral.

Algorithmes gloutons

Définition 1 : Problème algorithmique

Un **problème algorithmique** est une question à laquelle on veut répondre par un algorithme sur un ensemble d'entrée possible.

Exemple 1 : Quelques exemples de problèmes algorithmiques

- Étant donné un tableau en entrée, déterminer le maximum de ce tableau ;
- Étant donné un graphe et deux sommets, trouver un plus court chemin entre ces sommets ;
- Étant donné un graphe et deux sommets, déterminer s'il existe un chemin de longueur inférieur ou égale à l ;
- Proposer un algorithme qui permet de déterminer si un programme termine ou non.

Définition 2 : Problème d'optimisation

Un **problème d'optimisation** est un problème dans lequel on cherche une solution qui est minimale (ou maximale selon les problèmes) pour une certaine caractéristique sur une entrée donnée.
Une solution qui atteint ce minimum (ou ce maximum) est appelé **solution optimale**.

Définition 3 : Algorithme Glouton

Un **algorithme glouton** est un algorithme qui réalise à chaque étape le choix localement optimal dans la construction d'une solution pour un problème d'optimisation pour donner une solution globale.

Exemple 2 : Problème du rendu de monnaie

Le problème du rendu de monnaie est le suivant : on dispose de n de pièces de valeurs croissantes v_1, v_2, ..., v_n, et on dispose d'une valeur S à rendre.
L'objectif est de sélectionner un k-uplet i_1, ..., i_k de sorte à ce que $S = \sum_{j=1}^{k} v_{i_j}$, et de sorte à ce que k soit minimal.

Algorithme 1 : Algorithme glouton pour le rendu de monnaie

Pour les valeurs des pièces $v_1 = 1$, $v_2 = 2$, $v_3 = 5$, $v_4 = 10$, l'algorithme suivant donne la solution optimale :

Entrée : S la somme à obtenir

Sortie : $i_1, \cdots, i_k$ les indices des pièces à utiliser

$k \leftarrow 1$

Pour Chaque j de 4 à 1 **Faire**

    Tant Que $S \geqslant v_j$ **Faire**

        $S \leftarrow S - v_j$

        $i_k \leftarrow j$

        $k \leftarrow k + 1$

Renvoyer $i_1, \cdots, i_k$

Proposition 1 : La méthode gloutonne n'est pas toujours optimale

Pour certaines valeurs des (v_j), la méthode gloutonne n'est pas optimale.

Proposition 1 : La méthode gloutonne n'est pas toujours optimale

Pour certaines valeurs des (v_j), la méthode gloutonne n'est pas optimale.

Exercice 1. Trouver un système monétaire dans lequel il y a un contre exemple pour la méthode gloutonne.

Définition 4 : Problème du sac à dos

Le **problème du sac à dos** est un problème d'optimisation qui travaille sur l'entrée suivante : on dispose d'un sac de taille T, et d'un ensemble de n objets de poids $(p_i)_{1 \leqslant i \leqslant n}$ et de valeurs $(v_i)_{1 \leqslant i \leqslant n}$.
L'objectif est de trouver un sous-ensemble $I \subset [\![1, n]\!]$ d'objets tel que l'on respecte la contrainte que le poids des objets choisis ne dépassent pas la taille du sac, et en maximisant la valeur totale des objets.

$$\text{Maximiser } \sum_{i \in I} v_i$$

$$\text{Avec } \sum_{i \in I} p_i \leqslant T$$

Proposition 2 : Inefficacité de la méthode gloutonne

La méthode gloutonne de prendre les objets que l'on peut porter par ordre décroissant de $\frac{v_i}{p_i}$ ne donne pas nécessairement une solution optimale dans le problème du sac à dos.

Définition 5 : Problème du sac à dos fractionnaire

Le **problème du sac à dos** est un problème d'optimisation qui travaille sur l'entrée suivante : on dispose d'un sac de taille T, et d'un ensemble de n objets de poids $(p_i)_{1 \leqslant i \leqslant n}$ et de valeurs $(v_i)_{1 \leqslant i \leqslant n}$.
L'objectif est de trouver un vecteur $(x_1, \cdots, x_n)$ de fraction des objets qui maximise la valeur, en respectant la contrainte que le poids total ne dépasse pas T.

$$\text{Maximiser} \sum_{1 \leqslant i \leqslant n} x_i v_i$$

$$\text{Avec} \sum_{1 \leqslant i \leqslant n} x_i p_i \leqslant T$$

$$\forall 1 \leqslant i \leqslant n, 0 \leqslant x_i \leqslant 1$$

Algorithme 2 : Algorithme glouton pour le sac à dos fractionnaire

Quitte à prendre les objets de poids nul, on peut supposer que pour tout i, $p_i > 0$. On suppose par ailleurs que l'on dispose des objets dans l'ordre décroissant de $\frac{v_i}{p_i}$, et on a donc :

$$\frac{v_1}{p_1} \geqslant \frac{v_2}{p_2} \geqslant \cdots \geqslant \frac{v_n}{p_n}$$

.

L'algorithme suivant donne une solution optimale :

Entrée : $p_1, \cdots, p_n$ le poids des objets, $v_1, \cdots, v_n$ la valeur des objets, T la capacité du sac

Sortie : $(x_1, \cdots, x_n)$ une solution optimale

$(x_1, x_2, \cdots, x_n) \leftarrow (0, 0, \cdots, 0)$

Pour Chaque i de 1 à n **Faire**

$x_i \leftarrow \min(1, \frac{T}{p_i})$

$T \leftarrow T - x_i p_i$

Renvoyer $(x_1, \cdots, x_n)$

Diviser-pour-régner

Définition 6 : Algorithme diviser pour régner

Un algorithme **diviser pour régner** est composé de trois étapes :

- La **séparation** (ou **division**) qui consiste à séparer un problème en sous-problèmes ;
- Le **règne** qui consiste à traiter les sous-problèmes indépendamment les uns des autres ;
- La **réunion** qui consiste à combiner les solutions des sous-problèmes pour construire la solution du problème.

Algorithme 3 : Multiplication naïve en diviser-pour-régner

Entrée : $x = \sum_{0 \leqslant i \leqslant n-1} a_i 2^i$ un entier, $y = \sum_{0 \leqslant i \leqslant n-1} b_i 2^i$
Sortie : xy

Si $n = 1$ **Alors**

  Renvoyer $a_0 b_0$

Sinon

  On écrit $x = x_1 2^{\lfloor \frac{n}{2} \rfloor} + x_0$ et $y = y_1 2^{\lfloor \frac{n}{2} \rfloor} + y_0$

  $z_3 \leftarrow x_1 y_1$ calculé récursivement.

  $z_2 \leftarrow x_0 y_1$ calculé récursivement.

  $z_1 \leftarrow x_1 y_0$ calculé récursivement.

  $z_0 \leftarrow x_0 y_0$ calculé récursivement.

  Renvoyer $z_3 2^{2\lfloor \frac{n}{2} \rfloor} + (z_1 + z_2) 2^{\lfloor \frac{n}{2} \rfloor} + z_0$

Proposition 3 : Complexité de la méthode naïve

Dans le pire des cas, la méthode naïve a une complexité temporelle en $O(n^2)$, et une complexité spatiale en $O(n^2)$.

En effet, on peut remarquer que :

$$\begin{aligned} xy &= (x_1 2^{\lfloor \frac{n}{2} \rfloor} + x_0)(y_1 2^{\lfloor \frac{n}{2} \rfloor} + y_0) \\ &= x_1 y_1 2^{2\lfloor \frac{n}{2} \rfloor} + (x_1 y_1 + x_0 y_0 - (x_1 - x_0)(y_1 - y_0)) 2^{\lfloor \frac{n}{2} \rfloor} + x_0 y_0 \end{aligned}$$

Algorithme 4 : Algorithme de Karatsuba

Entrée : $x = \sum_{0 \leqslant i \leqslant n-1} a_i 2^i$ un entier, $y = \sum_{0 \leqslant i \leqslant n-1} b_i 2^i$
Sortie : xy

Si $n = 1$ **Alors**
- **Renvoyer** $a_0 b_0$

Sinon
- On écrit $x = x_1 2^{\lfloor \frac{n}{2} \rfloor} + x_0$ et $y = y_1 2^{\lfloor \frac{n}{2} \rfloor} + y_0$
- $z_4 \leftarrow (x_1 - x_0)(y_1 - y_0)$ calculé récursivement.
- $z_3 \leftarrow x_1 y_1$ calculé récursivement.
- $z_0 \leftarrow x_0 y_0$ calculé récursivement.
- **Renvoyer** $z_3 2^{2\lfloor \frac{n}{2} \rfloor} + (z_3 + z_0 - z_4) 2^{\lfloor \frac{n}{2} \rfloor} + z_0$

Proposition 4 : Complexité temporelle de l'algorithme de Karatsuba

La complexité temporelle de l'algorithme de Karatsuba est en $O(n^{\log_2 3})$.

Programmation dynamique

Recherche exhaustive