TD12: Stratégies algorithmiques

MP2I Lycée Pierre de Fermat

Exercice 1. *Collection de problèmes*

Pour chacun des problèmes décrits, proposer :

— Une formalisation comme un problème d'optimisation, en spécifiant l'espace des solutions et la fonction objectif ;

— Un exemple d'instance, avec représentation graphique lorsque c'est pertinent ;

N'hésitez pas à introduire des notations intermédiaires (ensembles ou fonctions utiles, etc...)

Q1. Problème de la somme des sous-ensembles (nom classique : `Subset-Sum`) :
Pour un ensemble E fini d'entiers naturels, et un entier cible $t \in \mathbb{N}$, trouver le sous-ensemble de E dont la somme est la plus proche de t, sans dépasser t.

Q2. Problème du groupe en festival :
Un groupe de p personnes vont à un festival proposant n évènements $E_1, \dots E_n$, où chaque E_i commence à l'horaire $d_i \in \mathbb{N}$, et termine à l'horaire $f_i \in \mathbb{N}$. Elles veulent maximiser le nombre d'évènements distincts **total** auxquels elles assistent.

Q3. Problème des journalistes en festival :
En reprenant le festival de la question précédente, un magazine veut envoyer des journalistes assister à **tous** les évènements proposés. Quel est le nombre minimal de journalistes à envoyer ?

Q4. Problème du voyageur de commerce (nom classique : `TSP` pour Travelling Salesperson Problem) :
On considère n villes, de coordonnées $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$. Un voyageur de commerce veut faire un circuit passant exactement une fois par chaque ville, pour y vendre ses marchandises. Pour être le plus efficace possible, il souhaite minimiser son temps de trajet total, et revenir à sa ville de départ. On suppose qu'il se déplace en ligne droite.

Q5. Problème des éoliennes :
Une région veut installer un parc d'éoliennes dans une zone qu'on modélise par le carré unité $C = [0; 1]^2$. Ce parc est d'une taille négligeable, mais génère du bruit dans un rayon r. Les habitants de la région se situent à des points $(x_0, y_0), \dots, (x_{n-1}, y_{n-1}) \in C$. On suppose que chaque point correspond à un unique habitant. Comment placer le parc de façon à déranger le moins d'habitants ?

Q6. Problème des éoliennes désagréables :
Un politicien corrompu veut installer un parc d'éoliennes dans une zone qu'on modélise par le carré unité C, comme précédemment, où habitent des personnes à des points $(x_0, y_0), \dots, (x_{n-1}, y_{n-1}) \in C$. Afin d'être le plus dérangeant possible, le parc doit être tel que la distance maximale de son centre à une habitation est la plus petite possible.

Q7. Problème des entrepôts :
n villes $V_1, \dots V_n$ dont situées dans le plan. Une entreprise veut ouvrir des entrepôts pour y stocker ses produits, et a repéré $m > k$ locations potentielles $L_1, \dots L_m$. Le coût d'ouvrir un entrepôt est noté $c \in \mathbb{R}^+$, et le coût de relier la ville V_i à la location L_j est noté $c_{ij} \in \mathbb{R}^+$. Ces coûts ont été évalués à l'avance par l'entreprise, qui veut relier **toutes** les villes à exactement un entrepôt, de la manière la moins coûteuse possible.

Exercice 2.

Banquet

Q1. Reprendre les problèmes **Q1.**, **Q3.**, **Q4.** et **Q7.** de l'exercice précédent. Pour chacun, donner un algorithme glouton, et sa complexité asymptotique en fonction des paramètres du problème.

Q2. Pour les problèmes **Q1.**, **Q4.** et **Q7.**, donner un contre-exemple à l'algorithme glouton proposé.

Exercice 3.

Bin-packing

On considère n fichiers de taille $t_1, \ldots, t_n$, que l'on veut stocker sur des disques durs externes, chacun de capacité maximale C, en utilisant le moins de disques durs possible.
On note $S = t_1 + \cdots + t_n$ la taille totale des fichiers.

Q1. Formaliser le problème d'optimisation sous-jacent.

Q2. Donner une minoration simple de la valeur optimale du problème en fonction de S et C

De plus, on considère que les fichiers arrivent dans un ordre $t_1, \ldots t_n$ fixé, et que l'on doit les traiter dans cet ordre. On ne peut donc pas trier les fichiers au préalable. On dit que l'on considère une version **en ligne** du problème car on doit prendre les décisions en direct, à chaque nouvelle donnée reçue.

Algorithme Next-Fit On prépare un disque dur vide initial. On stocke tous les fichiers qui arrivent sur ce disque dur, jusqu'à ce qu'un fichier ne tienne pas sur le disque. Alors, on range le disque dur, et on en prépare un nouveau, vide, dans lequel on met le fichier, et on reprend. On continue jusqu'à avoir traité tous les fichiers. En particulier, une fois que l'on a rangé un disque dur, on ne stocke plus aucun fichier dessus.

Q3. Appliquer cet algorithme avec $C = 10$, et la suite de tailles de fichiers suivante :

$$3, 2, 7, 4, 5, 1, 6, 5, 4, 5$$

obtient-on une solution optimale ?

On considère une instance $(t_1, \ldots, t_n, C)$. Notons K^* le nombre minimal de disques à utiliser, et K_g le nombre de disques utilisé par l'algorithme Next-Fit. Nous allons montrer que $K_g \leq 2K^*$. Autrement dit, l'algorithme glouton utilise au plus deux fois plus de disques qu'une solution optimale. On dit alors que l'algorithme Next-Fit est une **2-approximation**.

Q4. Montrer que dans la solution générée par Next-Fit, deux disques successifs contiennent nécessairement des fichiers dont la taille fait strictement plus que C.

Q5. Montrer que $K_g \leq 2K^*$.

Il existe de nombreuses variantes de l'algorithme glouton Next-fit : on pourrait garder tous les disques ouverts, et mettre chaque nouveau fichier dans un disque quelconque, ou bien dans le disque ayant le moins d'espace libre, etc... Cependant, aucun algorithme polynomial n'est connu à ce jour. Le problème est même NP-complet, autrement dit aussi dur que SAT !

Exercice 4.

Multiplication de matrice : algorithme de Strassen

Soit $n \in \mathbb{N}^*$. On considère deux matrices $A, B \in \mathcal{M}_n(\mathbb{R})$. On s'intéresse au calcul de AB.

Q1. Préliminaire : justifier que la **somme** $A + B$ peut se calculer en temps $\mathcal{O}(n^2)$.

Q2. Proposer un algorithme simple pour calculer AB, et donner sa complexité.

On suppose dans la suite que n est une puissance de 2, quitte à ajouter des coefficients nuls. Notons $C = AB$. On remarque que si l'on décompose A, B et C en 4 matrices de tailles moitié :

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

alors on a :

- $C_{11} = A_{11}B_{11} + A_{12}B_{21}$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22}$
- $C_{21} = A_{21}B_{11} + A_{22}B_{21}$
- $C_{22} = A_{21}B_{12} + A_{22}B_{22}$

On peut donc calculer C en effectuant 8 produits de matrices de taille moitié.

Q3. En utilisant cette observation, proposer un algorithme récursif de multiplication de matrice. En notant $C(n)$ le coût de multiplication de deux matrices de taille $n \times n$ par cet algorithme, donner la relation de récurrence vérifiée par $C(n)$ et en déduire la complexité de l'algorithme.

L'algorithme de **Strassen**, plus efficace, consiste à effectuer seulement 7 multiplications de matrices de taille moitié. On introduit les matrices suivantes :

- $M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
- $M_2 = (A_{21} + A_{22})B_{11}$
- $M_3 = A_{11}(B_{12} - B_{22})$
- $M_4 = A_{22}(B_{21} - B_{11})$
- $M_5 = (A_{11} + A_{12})B_{22}$
- $M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$
- $M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

Q4. Vérifier que $C_{11} = M_1 + M_4 - M_5 + M_7$

Q5. Exprimer C_{12}, C_{21}, C_{22} en fonction de $M_1, \ldots, M_6$. Vous n'avez pas besoin d'utiliser M_7, qui n'est utilisée que pour le calcul de C_{11}.

Q6. En déduire l'algorithme de Strassen, et calculer sa complexité asymptotique.

Exercice 5.

Voyageur de commerce

On reprend le problème **TSP** de l'exercice 1. Nous avons vu que l'algorithme glouton consistant à systématiquement se rendre vers la ville non-visitée la plus proche n'est pas optimal.
Plutôt que de prendre en entrée les coordonnées des villes, on suppose qu'une instance est directement donnée par la matrice des distances entre chaque ville : on dispose donc d'une matrice $M \in \mathcal{M}_n(\mathbb{R})$ telle que M_{ij} donne la distance entre les villes i et j.

Q1. Donner le pseudo-code d'un l'algorithme exhaustif naïf pour résoudre ce problème, et donner sa complexité.

Nous allons mettre en place un algorithme de programmation dynamique pour résoudre TSP. Sa complexité ne sera pas polynomiale, mais sera sensiblement meilleure que celle de l'algorithme naïf.

Q2. Pour $I \subseteq [\![1, n]\!]$ et $k \in [\![1, n]\!]$, on pose $C(I, k)$ la longueur du chemin le plus court qui part de la ville 1, visite toutes les villes de I, puis arrive en k. Donner une relation de récurrence pour C.

On encode les sous-ensembles de $[\![1, n]\!]$ en binaire : l'ensemble $I \subseteq [\![1, n]\!]$ sera encodé par un entier dont les bits d'indices dans I valent 1, et dont les autres valent 0. On comptera les indices à partir de 1, en partant de la droite. Par exemple, l'ensemble $\{1, 3, 4, 7\}$ sera encodé par 1001101.
Cet encodage permet de stocker les $C(I, k)$ dans un tableau T de 2^n lignes et n colonnes.

Q3. En déduire un algorithme de programmation dynamique. Quelle est sa complexité ?

Exercice 6.

La matmult, elle assure

On s'intéresse à un autre aspect de la multiplication de matrices. On suppose que l'on utilise un algorithme naïf de multiplication, de telle sorte que la multiplication d'une matrice A de taille $m \times n$ et d'une matrice B de taille $n \times p$ coûte exactement nmp multiplications de scalaires.

Considérons trois matrices A, B, C, avec A de dimensions $m \times n$, B de dimensions $n \times p$ et C de dimensions $p \times q$. La matrice produit ABC est de dimensions $m \times q$. Pour la calculer, deux possibilités :

- Calculer AB puis $(AB)C$
- Calculer BC puis $A(BC)$

Q1. Donner le coût de chaque méthode, en fonction de m, n, p, q.

Q2. Pour tout $N \in \mathbb{N}$, donner m_N, n_N, p_N, q_N, de telle sorte que le coût de la première méthode est négligeable devant le coût de la deuxième méthode lorsque $N \to +\infty$.

De manière générale, si l'on considère des matrices $A_1, \ldots, A_n$, avec chaque A_i de dimensions $p_{i-1} \times p_i$, l'ordre dans lequel on calcule le produit $A_1 A_2 \ldots A_n$ peut avoir un impact très fort sur le temps de calcul. Notons qu'un ordre de calcul revient en fait à parenthéser le produit $A_1 \ldots A_n$ de manière non-ambigüe, ou bien à construire un arbre syntaxique.

On introduit le problème d'optimisation **MATMULT** : étant donné $p_0, p_1, \ldots p_n$ des dimensions de matrices $A_1, \ldots A_n$, quel est le coût minimal pour calculer le produit $A_1 \ldots A_n$?

Q3. Proposer un algorithme glouton, et donner un contre exemple montrant qu'il n'est pas optimal.

Nous allons résoudre ce problème par une approche de programmation dynamique. Pour $1 \leq i \leq j \leq n$, on note $C(i, j)$ le coût minimal nécessaire pour calculer le produit $A_i \ldots A_j$.

Q4. Donner une relation de récurrence sur $C(i, j)$, et donner ses cas de base.

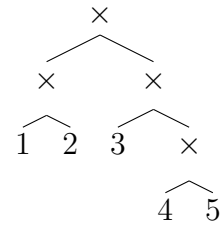
Q5. Vaut-il mieux procéder par une approche de bas en haut ou de haut en bas ?

Q6. En déduire un algorithme de programmation dynamique pour le problème **MATMULT**, et donner sa complexité.

On s'intéresse maintenant à la construction d'une solution optimale à partir du tableau calculé dans l'algorithme de prog. dyn. Explicitement, on cherche à construire un arbre de syntaxe, dont :

- Les feuilles sont des entiers $k \in [\![1, n]\!]$
- Les noeuds internes marquent les multiplications

de telle sorte que les feuilles sont ordonnées dans l'ordre croissant et contiennent $1, 2, \ldots, n$. Par exemple, l'arbre ci-contre correspond à l'ordre de multiplication $(A_1 A_2)(A_3(A_4 A_5))$.



Q7. Décrire un algorithme permettant de construire un arbre syntaxique à partir du tableau de prog. dyn. :

Algorithme 1 : Reconstruction d'une solution

Entrée(s) : $p_0, \ldots, p_n$ dimensions des matrices à multiplier, T tableau de taille $n \times n$ des coûts de multiplication calculé par programmation dynamique

Sortie(s) : A arbre syntaxique de coût optimal pour le calcul de $A_1, \ldots, A_n$

Indication : on pourra écrire un algorithme récursif qui prend également en entrée $0 \leq i \leq j \leq n$ et calcule l'arbre syntaxique optimal pour le calcul du produit partiel $A_i \ldots A_j$.

Q8. Quelle est la complexité de l'algorithme précédent ? Quelles informations pourrait-on stocker lors du calcul du tableau afin d'accélérer cette phase de construction ?