Feuille d'exercices n°12 - Schémas algorithmiques 2/2

Notions abordées

- Algorithmes gloutons optimaux, ensemble dominants, preuve par argument d'échange, algorithme gloutons d'approximation
- Algorithmes de programmation dynamique : calcul d'une valeur optimale, calcul d'une solution optimale, réduction de la complexité spatiale si possible
- Algorithmes "diviser pour régner", cas où l'on traite tous les sous-problèmes, cas où l'on en traite qu'un, coût de scission et coût de fusion.

Lorsqu'on demande de modéliser le problème comme un problème d'optimisation il faut identifier :

→ quels objets mathématiques représentent au mieux les données du problème ;

→ quel objet mathématique permet de représenter une éventuelle solution ;

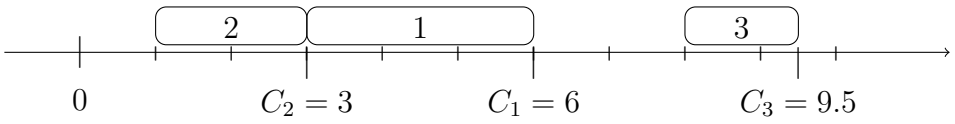
→ à quelles conditions un tel objet représente une solution valide, *i.e.* identifier les contraintes ;

→ quel serait le coût d'une telle solution, autrement dit identifier la fonction objectif.

Exercice 1 Tâches à effectuer

On considère un ensemble de n tâches à effectuer sur la même machine. On numérote ces tâches de 1 à n. Pour chaque tâche $j \in [1..n]$, on connaît son temps d'exécution $p_j \in \mathbb{R}_+^*$ (p comme *processing time* en anglais), c'est-à-dire la durée qu'il faut pour exécuter cette tâche. On suppose qu'on ne peut interrompre les tâches, c'est-à-dire que chaque tâche doit être réalisée d'un trait. De plus la machine ne peut effectuer qu'une seule tâche à la fois, et aucune tâche ne peut démarrer avant le temps 0. Dans ce cadre, on appelle **ordonnancement** la donnée d'une date de fin notée C_j (p comme *completion time* en anglais) pour chaque tâche j.

On peut représenter un ordonnancement par un **diagramme de Gantt** : sur un axe horizontal représentant le temps on place pour chaque tâche un rectangle de largeur égale à sa durée [1]. Par exemple le diagramme ci-dessous représente un ordonnancement à 3 tâches de durées respectives $p_1 = 3$, $p_2 = 2$ et $p_3 = 1.5$ unités de temps. Cet ordonnancement est encodé par le vecteur $C = (6, 3, 9.5)$.



1. la largueur est ici à comprendre au sens dimension horizontale, et elle est en réalité proportionnelle à la durée, selon l'échelle choisie sur le diagramme.

Question 1 ✍

Définir $\mathcal{S}$ l'ensemble des vecteurs $C \in \mathbb{R}^n$ représentant un ordonnancement possible. *On peut commencer par se demander quelle est la date de début de la tâche j si elle termine au temps C_j et à quelles conditions sur leur dates de fins C_i et C_j deux tâches i et j ne se chevauchent pas.*

Question 2 ✍

On cherche d'abord un ordonnancement qui minimise la somme des dates de fin des tâches. Formuler ce problème comme un problème d'optimisation $\mathcal{P}_1$.

Question 3 ✍

On dit qu'un ordonnancement est **calé à gauche** lorsque la première tâche commence au temps 0, et que les autres commencent juste quand la tâche précédente termine, autrement dit, il n'y a aucun temps d'inactivité entre deux tâches. En utilisant un argument d'échange, montrer que l'ensemble des ordonnancements calés à gauche est strictement dominant pour le problème $\mathcal{P}_1$, c'est-à-dire qu'il contient toutes les solutions optimales.

Question 4 ✍

Dans un ordonnancement calé à gauche et pour $k \in [1..n]$ quelle est la date de fin de la k-ième tâche ? *On peut se contenter d'une réponse en français ici, et remettre la formalisation mathématique à la question suivante.*

Question 5 ✍

Quelle relation peut-on établir entre l'ensemble des ordonnancements calés à gauche et S_n l'ensemble des permutations de $[1..n]$? En déduire une reformulation de $\mathcal{P}_1$. Préciser quelle serait la complexité d'un algorithme de recherche exhaustive pour ce problème.

Question 6 ✍

En intervertissant les sommes qui apparaissent dans la fonction objectif de $\mathcal{P}_1'$, identifier l'apport de la tâche placée en k-ième dans un ordonnancement calé à gauche. En déduire un algorithme efficace pour résoudre ce problème. Préciser de quel paradigme relève cet algorithme, ainsi que ses complexités spatiale et temporelle.

Question 7 ✍

À l'aide d'un argument d'échange, justifier que l'algorithme précédent fournit une solution optimale.

Question 8 ✍

Afin de modéliser le fait que les tâches n'ont pas toutes la même importance, on munit chaque tâche j d'un poids $\omega_j \in \mathbb{R}_+^*$. On cherche alors un ordonnancement qui minimise la somme pondérée des dates de fin des tâches. Formuler ce problème comme un problème d'optimisation $\mathcal{P}_2$.

Question 9 ✍

Que peut-on dire de l'ensemble des ordonnancements calés à gauche pour le problème $\mathcal{P}_2$? Est-il est strictement dominant ? Et si les poids ω de certaines tâches étaient nuls, serait-il strictement dominant ?

Question 10 ✍

Proposer un algorithme efficace pour résoudre le problème $\mathcal{P}_2$. Préciser de quel paradigme relève cet algorithme, ainsi que ses complexités spatiale et temporelle.

Exercice 2 Multiplication par l'algorithme de Karatsuba

Dans cet exercice on s'intéresse au problème de la multiplications de grands entiers machines. On suppose les entiers représentés par la séquence indicée des bits de leur décomposition en base 2. Par exemple, l'entier 14 est représenté par la séquence $(1, 1, 1, 0)$. On appellera alors taille d'un entier la longueur d'une telle séquence. On suppose de telles séquences stockées dans des tableaux permettant l'indexation en temps constant. On remarque que les multiplications et divisions entières par des puissances de 2 correspondent à des décalages de bits. En effet si $(u_{n-1}, u_{n-2}, \ldots, u_0)$ est la représentation d'un entier u et $p \in \mathbb{N}$ alors $u/2^p$ est représenté par $(u_{n-p-1}, u_{n-p-2}, \ldots, u_p)$ et $u \times 2^p$ par $(u_{n-1}, u_{n-2}, \ldots, u_0, \underbrace{0, 0, \ldots 0}_{p})$ et .

Dans tout l'exercice on s'intéresse uniquement au problème de la multiplication de deux entiers de même taille et on mesure la complexité au regard du nombre de multiplications de nombres à 1 bit. Ainsi l'opération 1×0 est considérée comme une opération atomique de coût 1 alors que le coût de calcul de 1110110×101110 dépend de l'algorithme choisi pour faire le calcul de $\times$.

Question 1
Expliquer en quoi l'hypothèse d'une taille commune des deux opérandes n'est pas trop simplificatrice.

Question 2
Rappeler l'algorithme de multiplications naïf d'entiers (celui appris en primaire).
Quelle est sa complexité ?

On remarque que si $u = u_a + 2^p u_b$ avec $u_a < 2^p$ et $v = v_a + 2^p v_b$ avec $v_a < 2^p$ alors $u \times v = u_a \times v_a + 2^p(u_b \times v_a + v_b \times u_a) + 2^{2p} u_b \times v_b$.

Question 3
Proposer un algorithme de multiplication d'entiers utilisant le paradigme diviser-pour-régner et utilisant la remarque précédente. Faire une rapide étude de complexité dans le cas où les entrées sont de taille 2^p. Cet algorithme diviser-pour-régner présente-t-il un avantage par rapport à l'algorithme naïf.

Question 4
En s'intéressant à la valeur de $u_a v_a + u_b v_b - (u_a - u_b)(v_a - v_b)$ proposer un algorithme faisant 3 appels récursifs sur des entrées dont la taille est divisée par 2.

Question 5
Faire une étude rapide de complexité dans le cas où les entiers en arguments sont de taille $n = 2^p$.

Question 6
Donner la complexité pire cas de l'algorithme de la question 14 au moyen d'un Θ. Au vu de la question précédente, on pourra intuiter puis démontrer par récurrence un encadrement sur le nombre de multiplications élémentaires effectué par l'algorithme de la question 14 sur deux entrées de taille respectivement n et m.

Question 7
Dans les questions précédentes nous avons ignoré le coût des additions et soustractions sur les grands entiers. On ne suppose plus que c'est le cas : une addition ou soustraction sur deux entiers de tailles n et m coûte dorénavant $\max(n, m)$. Faire une rapide étude de complexité de l'algorithme de la question 14 (on se contentera des entiers de la forme 2^p). Conclure.

Exercice 3 Parenthésage optimal d'un produit matriciel

Sachant que le produit matriciel est une opération associative (non commutative), il est usuel en mathématiques de noter seulement $A \times B \times C$ pour désigner le produit de trois matrices A, B, C, c'est-à-dire $A \times (B \times C)$ ou $(A \times B) \times C$. Si la valeur de ces deux dernières expression est bien la même, la manière de les évaluer est différente : dans le premier cas on multiplie d'abord B par C, puis A par le résultat, tandis que dans le second cas on multiplie d'abord A par B, puis le résultat par C. Dans cet exercice, on cherche à profiter de cette associativité pour réduire le nombre de multiplications entre nombres réels effectuées pour le calcul d'un produit de matrices données.

Question 1
Étant donné deux matrices M de dimensions $n \times m$ et L de dimensions $m \times k$, donner le nombre de multiplications de nombres réels effectuées par l'algorithme classique de multiplication matricielle pour calculer le produit de matrices ML.

Question 2
En déduire, étant donné les matrices M de dimensions $n \times m$, L de dimensions $m \times k$ et K de dimensions $k \times p$, le nombre de multiplications de nombres réels effectuées lors du calcul de $(ML)K$ et lors du calcul de $M(LK)$. Donner un exemple pour lequel $(ML)K$ induit moins (strictement) de multiplications de réels que $M(LK)$, un exemple pour lequel $(ML)K$ induit plus (strictement) de multiplications de réels que $M(LK)$.

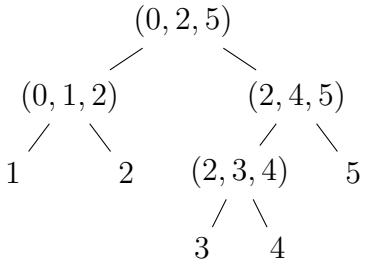
Dans la suite de cet exercice on s'intéresse donc au problème suivant : étant donné une suite de $n+1$ nombres entiers non nuls $(u_i)_{i \in [\![0,n]\!]}$ représentant les dimensions de n matrices : M_1 de dimensions $(u_0 \times u_1)$, M_2 de dimensions $(u_1 \times u_2)$, ..., M_n de dimensions $(u_{n-1} \times u_n)$, on souhaite minimiser le nombre de multiplications de réels induites par le produit matriciel $M_1 M_2 \dots M_n$ en choisissant un parenthésage optimal.

Question 3
Proposer un algorithme glouton, inspiré de l'algorithme de Huffman.
Fournit-il une solution optimale ?

On définit inductivement la famille des *arbres de parenthésage de l'intervalle* $[\![g, d]\!]$ avec $g < d$ par :

- Une feuille étiquetée par d si $g = d - 1$;
- Un nœud étiqueté par le triplet (g, i, d) pour $i \in [\![g+1, d-1]\!]$ ayant un fils gauche qui est un arbre de parenthésage de l'intervalle $[\![g, i]\!]$ et un fils droit qui est un arbre de parenthésage de l'intervalle $[\![i, d]\!]$.

Dans la suite lorsque $g < d$, on note $\mathcal{T}_{g,d}$ l'ensemble des arbres de parenthésage de l'intervalle $[\![g, d]\!]$. Un arbre de parenthésage de l'intervalle entier $[\![0, n]\!]$ représente alors un choix de parenthésage d'une expression matricielle $M_1 M_2 \dots M_n$. Par exemple le parenthésage $((M_1 M_2)((M_3 M_4) M_5))$ est représenté par l'arbre ci-dessous.



Étant donné une suite de dimensions $(u_0, u_1, \ldots, u_n) \in (\mathbb{N}^*)^{n+1}$ et T un arbre e parenthésage de l'intervalle $[\![i, j]\!]$ avec $0 \leq i < j \leq n$, on définit inductivement sur T le coût de T :

— $\mathsf{cout}(T) = 0$ si T est une feuille ;

— $\mathsf{cout}(T) = u_g\, u_i\, u_d + \mathsf{cout}(A) + \mathsf{cout}(B)$ si T un nœud interne d'étiquette (g, i, d), de fils gauche A, de fils droit B.

Question 4
Formaliser le problème du parenthésage optimal.

Question 5
Établir une relation de récurrence permettant la résolution du problème.

Question 6
Les sous-problèmes induits par la relation de récurrence de la question précédente sont-ils indépendants les uns des autres ? Proposer une méthode évitant de résoudre plusieurs fois les mêmes sous-problèmes.

Question 7
Illustrer (au moyen d'une illustration) l'ordre de remplissage et la relation de dépendance de calcul de la matrice de programmation dynamique (pour remplir une case (g, d) de quelles autres cases a-t-on besoin ?).

Question 8
Proposer un algorithme permettant le calcul de la valeur de la solution optimale. Préciser sa complexité.

Question 9
Proposer un algorithme permettant le calcul de l'arbre de parenthésage optimal.

Question 10
Préciser la complexité pire cas de l'algorithme précédent au moyen d'un Θ.

Question 11
Proposer une amélioration de la réponse à la Question 25 qui permettrait d'obtenir une complexité en $\Theta(n)$, prouver votre réponse.

Exercice 4 Éléments majoritaires

On dit qu'un élément x est majoritaire dans un multi-ensemble E de cardinal n lorsque le nombre d'occurrences de x dans E est strictement supérieur à $\frac{n}{2}$. On peut remarquer que s'il existe, un élément majoritaire est nécessairement unique. Dans cet exercice, on cherche à déterminer si un multi-ensemble E représenté par un tableau de ses éléments admet un élément majoritaire, et si oui, quel est l'élément en question. On évalue la complexité d'un algorithme résolvant ce problème au regard du nombre de tests d'égalité effectués entre éléments du multi-ensemble.
On suppose définie une fonction nbOccur prenant en arguments un tableau T, un élément x et deux indices i et j du tableau et retournant le nombre d'occurrences de x dans le tableau T entre les indices i et j (au sens large). On suppose de plus que l'appel $\mathsf{nbOccur}(T, x, i, j)$ fait $j - i + 1$ comparaisons. [2]

2. Les suppositions des deux phrases précédentes sont là pour vous faire gagner du temps, il *faut* que vous sachiez implémenter un tel algorithme nbOccur.

Question 1
Proposer un algorithme naïf majoritaire(T) retournant s'il existe un élément majoritaire du multi-ensemble E représenté par le tableau T. Préciser la complexité pire cas de l'algorithme.

Question 2
Supposant connu l'élément majoritaire du sous-tableau $T[0..n/2-1]$ [3] et l'élément majoritaire du sous-tableau $T[n/2..n-1]$ où T est un tableau de taille n, donner l'élément majoritaire de T en faisant de l'ordre de n comparaisons.

Question 3
Donner alors un algorithme du paradigme diviser pour régner permettant la résolution du problème de la recherche d'un élément majoritaire.

Question 4
Donner la complexité pire cas pour un tableau en entrée de taille une puissance de 2.

Exercice 5 Le problème Bin Packing

Le problème est le suivant : on veut affréter un train de fret pour transporter des paquets qui ne sont pas tous de même volume. Les wagons permettant le stockage de ces paquets sont tous identiques et peuvent donc tous contenir un même volume de paquets (pas forcément le même nombre). On suppose que deux paquets occupent un volume qui est la somme de leurs volumes respectifs (comprendre : on laisse de côté le problème de la forme des paquets). Pour des raisons écologiques évidentes, on souhaiterait minimiser le nombre de wagons à utiliser. On appellera coût d'une solution le nombre de wagons utilisés.

Question 1
Formaliser le problème.

Question 2
Donner une minoration du coût d'une solution, en fonction des volumes des paquets et du volume des wagons.

Dans tout le reste de l'exercice, on suppose que les paquets à stocker arrivent un par un et on souhaiterait éviter d'avoir à ressortir les paquets une fois ceux-ci placés dans un wagon. De plus on décrit les algorithmes du point de vue de l'employé ou employée de la SNCF remplissant les wagons.

Algo *Next-Fit.* On ouvre un premier wagon, on prend les paquets un par un et on les place dans ce premier wagon. Au premier paquet qui ne rentre plus dans le wagon, on ferme *définitivement* le wagon et on passe au suivant, que l'on remplit de même avec les paquets, un par un jusqu'à ce qu'un paquet ne rentre pas, on ferme alors *définitivement* ce second wagon, on en ouvre un troisième, etc. ...

Question 3
Donner l'algorithme correspondant à cette stratégie *Next-fit*.

Question 4
Montrer que l'algorithme *Next-fit* ne fournit pas une solution optimale.

3. comprendre : on sait s'il existe un élément majoritaire et s'il existe, on sait qui il est

Question 5
Montrer que l'algorithme *Next-fit* fournit une 2-approximation, à savoir, pour toute entrée I la solution retournée par l'algorithme utilise un nombre de wagons inférieur à 2 fois le nombre de wagons utilisés par la solution optimale.

Algorithme *First-fit.* (Du point de vue de l'employé toujours :) À chaque paquet qui arrive je parcours tout le quai, du premier wagon ouvert vers le dernier wagon ouvert, à la recherche d'un wagon pouvant contenir le paquet courant, si je n'en trouve pas j'ouvre un nouveau wagon après le dernier wagon ouvert.

Question 6
Donner l'algorithme correspondant à cette stratégie *First-fit*.

Question 7
Montrer que cet algorithme n'est pas lui non plus optimal.

Question 8
Montrer que, tout comme la stratégie *Next-fit*, la stratégie *First-Fit* fournit une 2-approximation.