Empaquetage

Épreuve pratique d'algorithmique et de programmation

Concours commun des Écoles Normales Supérieures

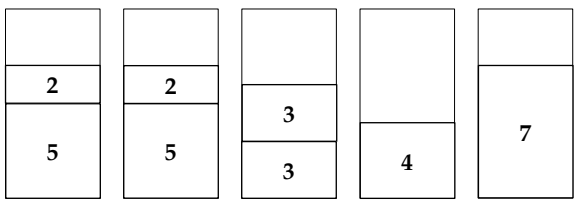
Durée de l'épreuve : 4 heures – Cœfficient: 4

Juillet 2003

N'oubliez en aucun cas de recopier votre u_0 clairement encadré au début de votre copie.

Notes. Lorsque la description d'un algorithme est demandée, celle-ci doit être courte et précise. Vous ne devez en aucun cas recopier le code de vos procédures sur votre copie !
Quand on demande le temps de calcul d'un algorithme en fonction d'un paramètre n, on demande son ordre de grandeur en fonction du paramètre, par exemple : $O(n^2)$, $O(n \log n)$,...
Il est recommandé de commencer par lancer vos programmes sur de petites valeurs des paramètres.

Le problème de l'empaquetage. L'empaquetage est l'un des premiers problèmes algorithmiques qui fut étudié en informatique. Il s'agit d'emballer des objets unidimensionnels donnés par leur taille dans un nombre minimum de boîtes de taille identique. Voici un exemple d'empaquetage non-optimal dans des boîtes de taille 10, de la suite d'objets donnés par leurs tailles : 5, 2, 3, 5, 7, 2, 4, 3.



Un empaquetage des objets dans 5 boîtes

On distingue deux types d'algorithmes : les algorithmes « en-ligne » qui placent les objets l'un après l'autre, dans leur ordre d'arrivée, sans regarder

les suivants (section 2, 3 et 4) et les algorithmes « hors-ligne », qui regardent tous les objets au départ, avant de les placer (section 5 et 6).
Le problème de l'empaquetage est NP-difficile, il est donc désespéré de rechercher une solution optimale dès que l'instance est trop complexe (à l'exception de la question 18). Nous nous proposons ici d'étudier différentes heuristiques pour ce problème.
Les sections sont très largement indépendantes. La dernière section traite d'une variante bidimensionnelle du problème de l'empaquetage.

1 Préliminaires

On définit la suite $(u_n)_{n\geqslant 0}$ suivante qui sera utilisée pour construire les entrées des algorithmes demandés. Le plus grand soin doit être porté à son implémentation.

- u_0 est donné sur votre table. **(À reporter en haut de votre copie !)**
- $u_n = (16383 \times u_{n-1})$ modulo 59047, pour $n \geqslant 1$.

Question 1 *Que valent u_{903} et u_{9763} ? Quel est le nombre d'indices $0 \leqslant i < 10000$ tels que $2000 \leqslant u_i \leqslant 5000$?*

2 Algorithmes élémentaires

On considère $n = 10000$ objets à empaqueter dans des boîtes de taille $T = 100$. Les tailles des objets à empaqueter sont $t_0, \ldots, t_{9999}$, donnés dans cet ordre, où $t_i = 1 + (u_i \text{ modulo } 60)$.

Question 2 *Combien d'objets ont pour taille 37 ? Donnez le plus petit et le plus grand indice d'un tel objet.*

Nous dirons qu'une boîte est *fermée* si l'algorithme décide de ne plus y placer d'objets à l'avenir. À l'inverse, tant qu'une boîte est *ouverte*, l'algorithme peut essayer d'y ajouter un objet. Une boîte est dite *pleine* si la somme des tailles des objets qu'elle contient, est égale à son volume (i.e., 100 dans cette section).
Les boîtes sont numérotées $B_0, B_1, B_2, B_3, ...$ par ordre de première ouverture.

Algorithme NextFit. Cet algorithme, noté NF, maintient une unique boîte ouverte à chaque pas de l'algorithme : les objets sont placés l'un après l'autre, dans l'ordre $t_0, ..., t_{9999}$; NextFit essaie de placer chaque objet dans la boîte ouverte courante ; s'il n'y tient pas, il la ferme et l'objet est placé dans une nouvelle boîte, qui devient la boîte ouverte courante.

Question 3 *Donnez l'empaquetage exact construit par NF pour la séquence d'objets suivants donnés par leurs tailles :* 50, 20, 40, 70, 20, 40, 30.

Question 4 *Proposez, pour implémenter cet algorithme, une structure de données, qui permette de connaître le contenu de chaque boîte utilisée et de savoir dans quelle boîte a été placé chaque objet.*

Question 5 *Quel est le nombre boîtes utilisées $B(NF)$ par l'algorithme NextFit pour empaqueter nos 10000 objets ? Évaluez le gâchis $G(NF)$ de cet algorithme, c'est-à-dire la place perdue totale dans les boîtes utilisées (expliquez rapidement comment vous avez procédé).*

Question 6 *Donnez le temps de calcul de votre implémentation de NextFit, en fonction de n (le nombre d'objets).*

Algorithme FirstFit. Cet algorithme, noté FF, garde ouvertes toutes les boîtes utilisées (et non-pleines). Les objets sont pris dans l'ordre $t_0, \ldots, t_{9999}$. Chaque objet est placé dans la première boîte (dans l'ordre de première ouverture), dans laquelle il tient ; si aucune des boîtes ouvertes ne convient, FirstFit ouvre une nouvelle boîte et y place l'objet.

Question 7 *Donnez l'empaquetage exact construit par FirstFit sur la séquence de tailles :* 50, 70, 55, 10, 45, 20, 30, 20 *; puis sur la séquence :* 50, 70, 55, 45, 20, 30, 20 *? On dit que FirstFit est un algorithme non-monotone, expliquez ce que cela signifie (en une ligne).*

Question 8 *Quel est le nombre boîtes utilisées $B(FF)$ par l'algorithme FirstFit pour empaqueter nos 10000 objets ? Dans quelle boîte est placé l'objet t_{557} ? Quels sont les objets placés dans la boîte B_{1327} ? Quel est le gâchis $G(FF)$ de cet algorithme ?*

Question 9 *Précisez l'action de votre implémentation sur la structure de données que vous avez choisie pour implémenter FirstFit. Évaluez le temps de calcul de votre implémentation en fonction de n (le nombre d'objets). Pensez-vous qu'il est possible de l'améliorer ? Comment ?*

Algorithme BestFit. Cet algorithme, noté BF, garde également ouvertes toutes les boîtes utilisées (et non-pleines). Les objets sont pris dans l'ordre $t_0, ..., t_{9999}$. BestFit place chaque objet dans la boîte ouverte la plus pleine, qui peut le contenir (en cas d'ex aequo, il choisit la première par ordre de première ouverture) ; si aucune ne convient, BestFit ouvre une nouvelle boîte et y place l'objet.

Question 10 *Quel est l'empaquetage exact construit par BestFit sur la séquence de tailles :* 70, 65, 45, 37, 18, 16, 15, 14, 13 *? et sur la séquence :* 70, 65, 45, 35, 18, 16, 15, 14, 13 *? BestFit est-il un algorithme monotone ? (réponse en une ligne)*

Question 11 *Quel est le nombre boîtes utilisées $B(BF)$ par l'algorithme BestFit pour empaqueter nos 10000 objets ? Dans quelle boîte est placé l'objet t_{557} ? Quels sont les objets placés dans la boîte B_{5327} ? Donnez les indices des 10 premières boîtes qui contiennent un élément de taille* 37*. Donnez le gâchis $G(BF)$ de BestFit.*

Question 12 *Précisez l'action de votre implémentation sur la structure de données que vous avez choisie pour implémenter BestFit. Évaluez le temps de calcul de votre implémentation en fonction de n (le nombre d'objets). Pensez-vous qu'il est possible de l'améliorer ? Comment ?*

3 Algorithme "Somme des carrés"

Dans cette section, les boîtes ont toujours taille $T = 100$. Comme précédemment, on considère $n = 10000$ objets à empaqueter dont les tailles sont $t_0, \ldots, t_{9999}$, données dans cet ordre, où $t_i = 1 + (u_i \text{ modulo } 60)$. Les boîtes sont numérotées $B_0, B_1, B_2, \ldots$, par ordre de première ouverture.

Définition Pour tout empaquetage $\mathcal{E}$ et $h \in \{1, \ldots, T-1\}$, on note $N_{\mathcal{E}}(h)$ le nombre de boîtes remplies jusqu'au niveau h exactement dans $\mathcal{E}$. On pose $ss(\mathcal{E}) = \sum_{h=1}^{T-1} N_{\mathcal{E}}(h)^2$.

Algorithme SumOfSquares. Cet algorithme, noté SOS, garde toutes les boîtes utilisées (et non-pleines) ouvertes. Les objets sont placés les uns après les autres dans l'ordre d'arrivée $t_0, \ldots, t_{9999}$. Notons $\mathcal{E}$ l'empaquetage courant. L'objet suivant est placé dans la boîte (qui peut le contenir) telle que si l'on note $\mathcal{E}'$ le nouvel empaquetage obtenu (qui compte éventuellement une boîte de plus que $\mathcal{E}$), alors la quantité suivante est minimum :

$$ss(\mathcal{E}') = \sum_{h=1}^{T-1} N_{\mathcal{E}'}(h)^2$$

En cas d'ex aequo, SumOfSquares place l'objet dans la boîte la plus remplie, et en cas de nouvel ex aequo, l'objet est placé dans la boîte la plus anciennement ouverte parmi les plus remplies. Remarquez que l'algorithme peut décider d'ouvrir une nouvelle boîte, même si l'objet tient dans une des boîtes ouvertes. L'algorithme est ainsi complètement déterminé.

Question 13 *Quel est le nombre de boîtes utilisées $B(SOS)$ par l'algorithme SumOfSquares pour empaqueter nos 10000 objets ? Donnez la valeur finale du vecteur $N_{\mathcal{E}}(h)$ pour l'empaquetage $\mathcal{E}$ calculé. Dans quelle boîte est placé l'objet t_{5357} ? Quels sont les objets placés dans la boîte B_{2327} ? Évaluez le gâchis $G(SOS)$ de cet algorithme. Comparez $G(SOS)$ et $G(FF)$, défini à la question 8.*

Question 14 *Décrivez votre implémentation et la structure de données utilisée. Donnez son temps de calcul en fonction du nombre d'objets n et de la taille des boîtes T.*

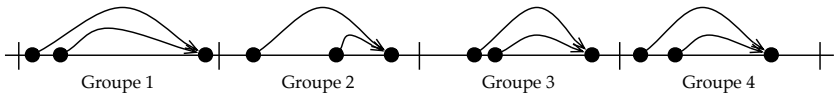
4 Algorithme arbitrairement proche de l'optimal

Dans cette section, les boîtes ont taille $T = 100$. On se propose d'empaqueter $n = 1000$ objets dont les tailles sont $t_0, \ldots, t_{999}$, où les $t_i = 1 + (u_i \text{ modulo } 60)$.

Les algorithmes des sections précédentes, même s'ils donnent de bonnes performances sur des instances aléatoires, admettent des pires cas sur lesquels ils produisent des empaquetages utilisant jusqu'à 1.54 fois plus de boîtes que l'optimum (1.7 fois pour FirstFit). L'algorithme décrit ici, noté $PTAS$, permet dans toute sa généralité d'approcher, avec une précision arbitraire ε, le nombre de boîtes optimum quelque soient les tailles des objets. Nous nous proposons de l'implémenter pour $\varepsilon = 1/2$.
Contrairement aux précédents, cet algorithme est "hors-ligne" et traite les objets dans un ordre qu'il détermine en fonction de leurs tailles. Il procède en trois étapes. Les deux premières étapes calculent un très bon empaquetage des objets de taille $\geqslant 34$. Et la troisième étape insère les objets de tailles $\leqslant 33$ avec l'algorithme FirstFit dans l'empaquetage calculé par les deux premières étapes.

Nous commençons donc par ne considérer que les objets de taille $\geqslant 34$.

Première étape : arrondir la taille des objets $\geqslant 34$. Notons q le nombre des objets de tailles $\geqslant 34$ parmi les $n = 1000$ objets à empaqueter. Ces q objets sont triés par taille croissante (en cas d'es ex aequo, les objets sont triés par indice croissant). Puis on partitionne ces objets en 4 groupes $(G_j)_{1 \leqslant j \leqslant 4}$ de tailles identiques à une unité près, $\operatorname{card} G_j = \lfloor \frac{q}{4} \rfloor + r_j$, pour $j = 1, .., 4$, avec $1 \geqslant r_1 \geqslant ... \geqslant r_4 = 0$. Remarquez que des objets différents de même taille peuvent appartenir à des groupes différents. L'algorithme associe alors aux objets de chaque groupe une *taille arrondie* t'_i, qui est la taille du plus grand objet du groupe. Ainsi, si l'objet $i \in G_j$, alors $t'_i = \max\{t_k : k \in G_j\}$. Cette étape est illustrée ci-dessous :



Question 15 *Pour nos $n = 1000$ objets : Que vaut q ? Quelles sont les 4 tailles arrondies ? Combien il y a-t-il d'objets pour chacune des tailles arrondies ? Quelle est la taille arrondie du premier objet de taille $\geqslant 34$ et d'indice $\geqslant 557$ (précisez sa taille réelle et son indice) ?*

Question 16 *Détaillez l'algorithme et les structures de données que vous avez utilisés. Quel est en fonction de n et q, le temps de calcul de votre algorithme ?*

Deuxième étape : calcul d'un empaquetage optimum des objets pris avec leur taille arrondie. Dans cette étape, nous considérons les q objets (de taille $\geqslant 34$) avec leur taille arrondie. Puisqu'il n'y a plus que 4 tailles possibles, le nombre de possibilités pour remplir une boîte avec ces objets arrondis, est très petit. Nous pouvons donc espérer trouver rapidement, un empaquetage optimum des objets arrondis.

Question 17 *Donnez toutes les façons possibles de remplir une boîte avec les objets arrondis.*

Question 18 *Énumérez tous les empaquetages optimaux des q objets arrondis (les objets de même taille arrondie ne sont pas différenciés). Expliquez votre méthode et l'algorithme utilisé. Évaluez son temps de calcul en fonction de q.*

Question 19 *Choisissez un des empaquetages optimaux (précisez lequel) et substituez les objets arrondis par les objets réels correspondants (comme les tailles ont été arrondies vers le haut, l'empaquetage obtenu est toujours valide). Les objets de même taille arrondie sont placés par taille réelle croissante et, en cas d'ex aequo, par indice croissant. Quel est le nombre de boîtes utilisées ? Quel est le contenu de la boîte B_0 ? de la boîte B_{50} ?*

Troisième et dernière étape : insertion des objets de taille $\leqslant 33$. L'algorithme insère enfin dans l'empaquetage calculé à la question 19, les objets de taille $\leqslant 33$, pris par indice croissant, avec l'algorithme FirstFit (décrit à la question 8).

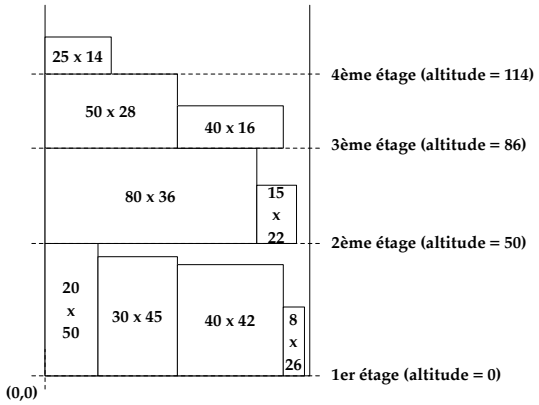
Question 20 *Combien de boîtes utilise l'algorithme $PTAS$ pour empaqueter nos $n = 1000$ objets ? Quel est son gâchis $G(PTAS)$? Quel est le contenu final des boîtes B_0, B_{10} et B_{50} ?*

5 Empaquetage bidimensionnel

Dans cette section (uniquement), nous étudions l'empaquetage d'objets bidimensionnels dans une bande verticale de largeur $L = 100$. Le but est de minimiser la hauteur de bande utilisée, c'est-à-dire l'ordonnée du coin supérieur gauche de l'objet le plus haut dans la bande. Les objets sont des rectangles donnés par leur largeur et leur hauteur. Les objets ne peuvent pas être tournés ou retournés. Nous nous proposons d'étudier une heuristique, puis une façon d'améliorer ses performances.
Nous souhaitons empaqueter $n = 1000$ objets donnés par leurs largeur $(\ell_i)_{i=0..999}$ et hauteur $(h_i)_{i=0..999}$, définies, pour $0 \leqslant i \leqslant 999$, par :

$$\ell_i = 1 + (u_{999-i} \text{ modulo } 60) \quad \text{et} \quad h_i = 1 + (u_{i+1000} \text{ modulo } 60)$$

Algorithme FirstFitDecreasingHeight. L'idée est de remplir la bande de largeur $L = 100$ par étages (v. illustration ci-dessous). L'algorithme commence par trier les objets par hauteur décroissante (en cas d'égalité des hauteurs, les objets sont triés par indice croissant). Le fond de la bande est le premier étage. Lors de la i-ème insertion, le i-ème objet est placé le plus à gauche sur le premier étage en partant du bas, qui peut l'accueillir ; si aucun des étages ouverts ne peut l'accueillir, alors un nouvel étage est ouvert juste au-dessus de l'objet le plus haut. Les étages sont numérotés $E_0, E_1, E_2, \ldots$ par ordre d'ouverture ; l'*altitude* d'un étage est l'ordonnée de son « plancher ». Voici une illustration de cet algorithme ; l'empaquetage obtenu a pour hauteur 128 :

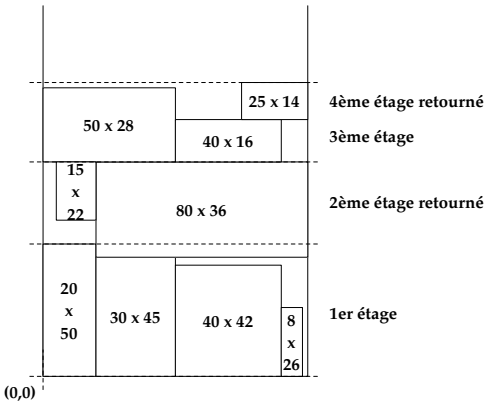


Question 21 *Donnez l'empaquetage obtenu par l'algorithme $FFDH$ sur les 10 premiers objets, $(\ell_0 \times h_0)$ à $(\ell_9 \times h_9)$.*

Question 22 *Quelle est la hauteur de l'empaquetage des $n = 1000$ objets, calculé par l'algorithme $FFDH$? Combien d'étages ont été créés ? Quelle*

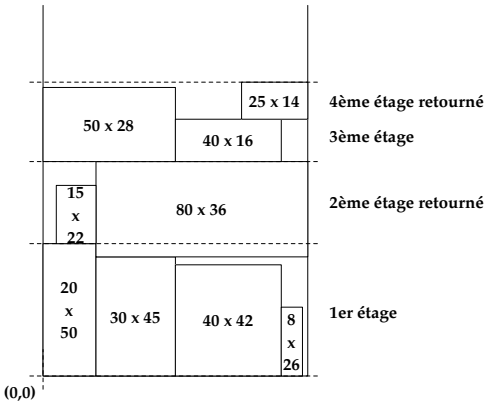
est l'altitude de l'étage E_{102} ? Quel est son contenu ? Quelles sont les coordonnées dans la bande des objets $(\ell_{103} \times h_{103})$, $(\ell_{237} \times h_{237})$ et $(\ell_{847} \times h_{847})$? Expliquez le fonctionnement de votre algorithme et précisez son action sur les structures de données. Évaluez son temps de calcul en fonction de n.

Optimisation de FirstFitDecreasingHeight. Une étape d'optimisation permet de réduire l'espace perdu par $FFDH$. Il s'agit de retourner de 180° les étages impairs $E_1, E_3, \ldots$, et de les serrer le plus possibles sur l'étage pair immédiatement inférieur (remarquez que cette opération déplace les rectangles sans les pivoter). Voici le résultat de cette optimisation sur l'exemple illustré ci-dessous :



Question 23 *Quelle est la hauteur de l'empaquetage optimisé pour nos $n = 1000$ objets ? Quelle est l'altitude de l'étage E_{102} ? Quelles sont les coordonnées dans la bande des objets $(\ell_{103} \times h_{103})$, $(\ell_{237} \times h_{237})$ et $(\ell_{847} \times h_{847})$?*

On propose maintenant d'optimiser encore l'empaquetage obtenu ci-dessus en « laissant agir la gravité », c'est-à-dire, en tassant tous les objets le plus bas possible, sans changer leur position horizontale. Voici une illustration de l'empaquetage ainsi obtenu à l'illustration précédente. T.P.S.V.P. ⇒



Question 24 *Quelle est la hauteur de l'empaquetage ré-optimisé pour nos* $n = 1000$ *objets ? Quelles sont les nouvelles coordonnées des objets, initialement placés sur l'étage* E_{102} *? Quelles sont les coordonnées dans la bande des objets* $(\ell_{103} \times h_{103})$*,* $(\ell_{237} \times h_{237})$ *et* $(\ell_{847} \times h_{847})$ *?*

Question 25 *On appelle pile critique, une pile d'objets adjacents verticalement (qui reposent les uns sur les autres) dont la hauteur est la hauteur de l'empaquetage. On appelle hauteur d'une pile critique, le nombre d'objets qui la compose. Quelle est la hauteur minimale d'une pile critique pour cet empaquetage ?*

— FIN —