Algorithme 4, HAI503I Année 2022-2023
Examen
L3 Info

- Algorithmique 4, examen -

- Durée: 2 heures - Notes de cours autorisées uniquement -
Janvier 2023

Le barême est indicatif. Les exercices peuvent être traités dans un ordre quelconque.
Toutes les réponses doivent être claires et justifiées.

- Exercice 1 - Mise en boîtes, algorithmes d'approximation - 11pts -

Dans le problème du BinPacking, on cherche à ranger n objets de poids $\mathcal{P} = (p_1, p_2, \ldots, p_n)$ avec $0 < p_i \leq 1$ pour tout $i = 1, \ldots, n$ dans un nombre minimum de boîtes qui ne peuvent contenir chacune que des objets dont le poids total est au plus 1.
Plus formellement, on cherche une partition de l'ensemble $\{1, \ldots, n\}$ en ensemble $I_1, \ldots, I_k$ avec k minimum telle que $\sum_{i \in I_\ell} p_i \leq 1$ pour $\ell = 1, \ldots, k$ (les objets dont les indices sont dans I_ℓ étant rangés dans la ℓième boîte).
Par exemple, si on a 8 objets de poids (0.1, 0.5, 0.6, 0.4, 0.2, 0.3, 0.2, 0.3) alors on peut ranger les objets 3 et 4 dans une première boîte, les objets 2, 6 et 7 dans une seconde boîte et les objets 1, 5 et 8 dans une dernière boîte, correspondant à la partition $(I_1 = \{3, 4\}, I_2 = \{2, 6, 7\}, I_3 = \{1, 5, 8\})$ en 3 boîtes.
Le but de l'exercice est de proposer des algorithmes de résolution de BinPacking.

1. Résoudre le problème de BinPacking pour l'instance (0.1, 0.8, 0.9, 0.2, 0.7, 0.8). Justifier bien que le nombre de boîtes proposé est minimum.

2. Pour une instance quelconque de BinPacking, montrer que le nombre de boîtes minimum nécessaire est supérieur ou égal à $\lceil \sum_{i=1}^{n} p_i \rceil$ (où $\lceil x \rceil$ désigne la partie entière supérieure du nombre x).

3. Donner l'exemple d'une instance où le nombre de boîtes nécessaire est strictement supérieur à $\lceil \sum_{i=1}^{n} p_i \rceil$.

On souhaite tout d'abord écrire un algorithme exact exhaustif pour résoudre le problème de BinPacking.

4. Dans un premier temps, on considère une instance de n objets et on souhaite savoir si il est possible de les ranger dans k boîtes avec k fixés. Il va falloir ainsi assigner à chaque objet un numéro de boîte compris entre 1 et k.

   (a) On suppose que l'on a accès à une fonction Suivante qui part de la liste $(1, \ldots, 1)$ de taille n et parcourt successivement toutes les listes de taille n composées d'entiers entre 1 et k. À l'aide de cette fonction, écrire un algorithme exhaustif de résolution pour BinPacking.

   (b) Calculer la complexité de votre algorithme.

5. On cherche maintenant à résoudre le problème entièrement, c'est-à-dire sans que le nombre k de boîtes à remplir soit fixé. Pour cela, pour une instance donnée, on va tester si il est possible de ranger les n objets dans 1 boîte, puis dans 2, puis etc... On arête dès qu'on trouve un nombre de boîtes qui convient.

   (a) Écrire l'algorithme correspondant.

   (b) Évaluer sa complexité.

On veut maintenant produire un algorithme d'approximation BinPacking-Glouton pour le problème de BinPacking. Cet algorithme traite les objets les uns à la suite des autres dans l'ordre donné par la liste de poids, et place l'objet courant dans la dernière boîte utilisée où il tient. Si il ne tient dans aucune boîte alors on le place dans une nouvelle boîte.

6. Écrire le pseudo code de l'algorithme BinPacking-Glouton et donner sa complexité.

7. Dérouler l'algorithme BinPacking-Glouton sur l'instance (0.6, 0.2, 0.3, 0.8, 0.5, 0.4). L'algorithme produit-il une solution optimale?

8. On va montrer que BINPACKING-GLOUTON est une 2-approximation pour le problème de BINPACKING. Pour cela, on note k le nombre de boîtes utilisées par l'algorithme, $B_1, \ldots, B_k$ ces boîtes et $p(B_i)$ le poids total des objets placés dans la boîte B_i.
   (a) Montrer que pour tout $i = 1, \ldots, k-1$ on a $p(B_i) + p(B_{i+1}) > 1$.
   (b) En déduire que $k - 1 < \sum_{i=1}^{k-1} p(B_i) + p(B_{i+1}) \leq 2 \sum_{i=1}^{n} p_i$.
   (c) Conclure.
   (d) Proposer un exemple de $4n$ objet pour lequel l'algorithme BINPACKING-GLOUTON utilise $2n$ boîtes alors que le nombre minimal de boîtes nécessaire est $n + 1$.

- Exercice 2 - Algorithme probabiliste - 5pts -

On considère un graphe G d'ensemble de sommets V et d'ensemble d'arêtes E. Une *3-coloration* c de G attribue à chaque sommet de G une 'couleur' qui est un entier de $\{1, 2, 3\}$. Autrement dit, c est une fonction $V \to \{1, 2, 3\}$. Une arête $e = \{x, y\}$ est dite *satisfaite* si $c(x) \neq c(y)$.
Le problème MIN3COL prend en entrée un graphe G et cherche le nombre maximum d'arêtes qui peuvent être satisfaite par une 3-coloration. On notera $c^\star$ cette valeur optimale.

1. Calculer la valeur $c^\star$ pour le graphe K de sommets $\{1, 2, 3, 4\}$ et d'arêtes $\{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$. Dessiner le graphe correspondant et justifier que votre valeur proposer est bien optimale.
2. On étudie l'algorithme probabiliste MAX3COL-PROBA qui consiste à tirer au hasard uniformément une couleur parmi $\{1, 2, 3\}$ pour chaque sommet et à retourner le nombre d'arêtes satisfaites.
   (a) Écrire en pseudo-code l'algorithme MAX3COL-PROBA. Le graphe d'entrée est codé par son nombre n de sommets, les sommets sont numérotés de 1 à n et E contient la liste des m arêtes de G, chacune étant une liste de 2 sommets. De plus, on fera appel à la primitive *randint(3)* qui renvoie un entier de $\{1, 2, 3\}$ aléatoirement et uniformément.
   (b) Calculer la complexité de votre algorithme en fonction de n et m.
   (c) Pour une arête $e = \{x, y\}$, on note X_e la variable aléatoire qui vaut 1 si e est satisfaite et 0 sinon. Calculer $Pr(X_e = 1)$.
   (d) On note X la variable aléatoire qui compte le nombre d'arêtes satisfaites. Calculer $E[X]$.
   (e) En remarquant que $c^\star \leq m$, conclure que MAX3COL-PROBA est une $\frac{2}{3}$-approximation probabiliste pour le problème MAX3COL.

- Exercice 3 - Table de hachage - 4pts -

1. On considère une table de hachage de taille $m = 10$, et la fonction de hachage $h(k) = ((ak) \bmod p) \bmod m$ avec $a = 2$ et $p = 101$.
   (a) Dans un premier temps, on résout les collisions par chaînage. Donner l'état de la table après insertion, dans cet ordre, des clefs 81, 27, 75, 20, 70, 29, 84 et 22.
   (b) On résout maintenant les collisions par adressage ouvert. Pour cela, on définit pour $i = 1$ à m la fonction de hachage $h_i(k) = h(k) + i - 1 \bmod m$. Pour insérer une clef k, on essaie successivement les cases d'indice $h_1(k), h_2(k), \ldots$ Donner l'état de la table après insertion des mêmes entiers qu'à la question précédente, dans le même ordre.
2. On considère maintenant une table de hachage de taille m, qui contient n clefs, et dans laquelle les collisions sont résolues par chaînage. On suppose que la fonction de hachage utilisée est tirée uniformément dans un ensemble $\mathcal{H}$ universel, c'est-à-dire, vérifiant pour tout $x \neq y$, lorsque l'on choisit h au hasard dans $\mathcal{H}$ on a $Pr[h(x) = h(y)] \leq 1/m$.
   On cherche une clef k dans la table. Calculer l'espérance de la longueur de la liste chaînée qu'il faut parcourir dans le pire cas.
