École normale supérieure **2010-2011**
Département d'informatique

Algorithmique et Programmation
TD n° 1 : Introduction
Avec Solutions

Exercices théoriques

Exercice 1. Le Grand saut. Le problème est de déterminer à partir de quel étage d'un immeuble sauter par la fenêtre est fatal. Vous êtes dans un immeuble à n étages (numérotés de 1 à n) et vous disposez de k étudiants. Il n'y a qu'une seule opération possible pour tester si la hauteur d'un étage est fatale : faire sauter un étudiant par la fenêtre. S'il survit, vous pouvez le réutiliser ensuite, sinon vous ne pouvez plus.

Vous devez proposer un algorithme pour trouver la hauteur à partir de laquelle un saut est fatal (renvoyer $n+1$ si on survit encore au n-ième étage) en faisant le minimum de sauts.

1. Si $k \geq \lceil \log_2(n) \rceil$, proposer un algorithme en $O(\log_2(n))$ sauts.
2. Si $k < \lceil \log_2(n) \rceil$, proposer un algorithme en $O(k + \frac{n}{2^{k-1}})$ sauts.
3. Si $k = 2$, proposer un algorithme en $2\sqrt{n}$ sauts.
4. Dans ce dernier cas, proposer aussi un algorithme en $\sqrt{2n}$ sauts.

Solution :

Exercice 2. 3SAT. Soit ϕ une formule en forme normale conjonctive, c'est-à-dire une conjonction de clauses qui sont des disjonctions, comportant au plus 3 littéraux par clauses. Nous souhaitons décider si une formule donnée ϕ ayant n variables peut être satisfaite, c'est-à-dire s'il existe une assignation des variables rendant vrai la formule. Nous supposons que ϕ n'est pas la formule vide et que :

$$\Phi = (x \vee y \vee z) \wedge \Phi',$$

pour des littéraux x, y, z et Φ' une formule 3CNF.

1. Donner un algorithme par recherche exhaustive résolvant ce problème est donner sa complexité.
2. Montrer que $\Phi = (x \wedge \Phi'|x) \vee (y \wedge \Phi'|y) \vee (z \wedge \Phi'|z)$ avec $\Phi'|x$ la formule Φ' obtenue en remplaçant x par vrai. En déduire un algorithme récursif et analyser sa complexité.
3. Améliorer cet algorithme en remarquant que dans le premier appel récursif si la formule $\Phi'|x$ ne peut pas être satisfaite avec x vrai, alors dans le second appel récursif, nous pouvons vérifier $\Phi'|\bar{x}y$ au lieu de . Quelle est la nouvelle complexité de votre algorithme ?
4. Un littéral est *pur* si il apparaît dans la formule Φ mais pas sa négation $\bar{x}$. Montrer que si Φ peut être satisfaite, il existe alors une assignation qui satisfait Φ avec tous les littéraux purs à vrai. Si $\Phi = (x \vee y \vee z) \wedge \Phi'$ ne contient pas de littéral pur, écrivons alors

   $$\Phi = (x \vee y \vee z) \wedge (\bar{x} \vee u \vee v) \wedge () \wedge \Phi'.$$

   Simplifiez la formule Φ pour x à vrai et donner un nouvel algorithme récursif. Analyser sa complexité.

Exercice 3. Étude des sous-suites monotones de taille maximale d'un tableau. On se donne un tableau A de n éléments entiers $A[1], \ldots, A[n]$. On appelle *sous-suite de longueur m* une suite d'indices $i_1, \ldots, i_m$ telle que $\forall k$, $1 \leq i_k \leq n$ et $i_k < i_{k+1}$. Une sous-suite est de plus *croissante* si $\forall k$, $A[i_k] \leq A[i_{k+1}]$, *décroissante* si $\forall k$, $A[i_k] \geq A[i_{k+1}]$, et *monotone* si elle est soit croissante, soit décroissante.

1. *(Théorème de Erdös–Szekeres)* On note $\psi : [1 \dots n] \to [1 \dots n] \times [1 \dots n]$ la fonction qui à k associe deux entiers : la longueur maximale des sous-suites croissantes (respectivement décroissantes) de A dont le dernier élément est d'indice k. Montrez que ψ est injective. Déduisez-en que si $n \geq (N-1)^2 + 1$, alors il existe forcément une sous-suite monotone de taille N.
2. Exprimez $\psi(k)$ en fonction de tous les $\psi(k')$ pour $k' < k$. Déduisez-en un algorithme polynomial qui détermine, par *programmation dynamique*, la longueur maximale des sous-suites monotones de A. Vous donnerez son pseudo-code, ainsi que son coût asymptotique en temps et en mémoire, en fonction de n.
3. Modifiez l'algorithme de la question précédente pour qu'il affiche, en plus de la longueur maximale des sous-suites monotones de A, une sous-suite monotone de longueur maximale.

Exercice 4. Problème d'Optimisation : Bin Packing. Soit n objets de volume des rationnels $a_1, \dots, a_n$ où $0 < a_i \leq 1$. Peut-on les partitionner en k boîtes $B_1, \dots, B_k$ de capacité au plus 1, tel que $\sum_{j \in B_k} \leq 1$.

On cherche à minimiser la valeur k. Une λ-approximation est un algorithme polynomial tel que, pour toute instance I du problème, $Sol_{algo}(I) \leq \lambda \cdot Opt(I)$.

1. L'algorithme Next Fit est le suivant :
   – prendre les objets dans un ordre quelconque
   – placer l'objet courant dans la dernière boîte utilisée s'il tient, sinon en créer une nouvelle
   Montrer que Next Fit est une 2-approximation. Montrer que la borne est fine.
2. L'algorithme Dec First Fit est le suivant :
   – Trier les a_i par ordre décroissant
   – Placer l'objet courant dans la première boîte utilisée où il tient, sinon en créer une nouvelle
   Montrer que Dec First Fit est une 3/2-approximation.

Solution :

$$K_{DFF} \leq \frac{3}{2} K_{Opt} + 1$$

$$A = \{a_i > \frac{2}{3}\}, B = \{\frac{2}{3} \geq a_i > \frac{1}{2}\}, C = \{\frac{1}{2} \geq a_i > \frac{1}{3}\} \text{ et } D = \{\frac{1}{3} \geq a_i\}$$

Dans la solution DFF, s'il existe au moins une boîte ne contenant que des éléments de D, alors au plus une boîte (la dernière) a un taux d'occupation inférieur ou égal à $\frac{2}{3}$. En effet, si les éléments de D de la dernière boîte n'ont pu être mis dans les boîtes précédentes, c'est que celles-ci sont remplies au moins $\frac{2}{3}$ (d'où la borne).

Sinon, la solution de DFF est la même que celle de l'instance où on enlève les éléments de D. La solution de DFF pour les éléments A, B et C est optimale. Dans toute solution,
– les éléments de A sont tout seuls,
– les éléments de B et C sont au plus 2 par boîtes et on a au plus un élément de B par boîte.
Or DFF fait la meilleure répartition des C en les appariant au plus grand B ou C compatible.

Remarque : on peut montrer $K_{DFF} \leq \frac{11}{9} K_{Opt} + 4$?

Exercice 5. Mariage. Un graphe biparti a ses sommets constitués de deux ensembles disjoints H et F et n'a d'arête qu'entre un sommet de H et un sommet de F. Un mariage est un ensemble d'arêtes n'ayant deux à deux aucun sommet commun. Si M est un mariage, on note $M(h)$, $h \in H$, l'unique élément f de F s'il existe tel que l'arête $\{h, f\}$ soit dans M. On pose $M(h) = \perp$ si f n'existe pas. On définit $M(f)$, $f \in F$, de manière analogue.

On se donne un graphe biparti complet à $2n$ sommets, ce qui signifie que chaque sommet h de H est lié à chaque sommet f de F. On se donne de plus pour chaque sommet h de H une fonction injective $r_h(f)$ qui affecte un entier $\leq n$ aux éléments de F. De même, on se donne pour chaque sommet f de F une fonction injective $r_f(h)$ qui affecte un entier aux éléments de H. Un mariage est *stable* s'il n'existe pas de couple (h, f), $h \in H$, $f \in F$ tel que :
– $\{h, f\} \notin M$
– $M(f)$ et $M(h)$ sont définis
– $r_h(f) > r_h(M(h))$
– $r_f(h) > r_f(M(f))$
Un mariage est *H-saturé* si pour tout $h \in H$, tel que $M(h)$ existe et pour tout $f \in F$ tel que $r_h(f) > r_h(M(h))$, on a $M(f) \neq \perp$.

1. Soit M un mariage stable H-saturé. Soit $h \in H$ tel que $M(h)$ soit non défini. On considère la réunion des ensembles

$$\{f \in F \mid M(f) = \perp\} \text{ et } \{f \in F \mid M(f) \neq \perp,\ r_f(h) > r_f(M(f))\}$$

   Soit f l'élément de la réunion de ces ensembles rendant $r_h(f)$ maximal. On définit M'
   – en ajoutant l'arête (h, f) à M si $M(f) = \perp$
   – en retirant de M l'arête $(f, M(f))$ et en ajoutant (h, f) sinon
   Montrer que M' est stable et H-saturé.
2. Proposer un invariant entier $n(M)$ qui augmente strictement quand on passe de M à M'.
3. Proposer un algorithme qui calcule un mariage stable ayant n arêtes. Montrer sa correction et évaluer sa complexité.