Planche 5 type CCINP

juin 2024

1 Partie A

Après un changement de l'équipe dirigeante, l'entreprise Mondelez International, qui fabrique les barres Toblerone, décide de rationaliser sa production pour maximiser ses revenus. En effet, leur chaîne de production fabrique des barres de n "carreaux" qui sont ensuite coupées en barres plus petites avant d'être vendues. Mais une récente étude de marché a déterminé le prix auquel on pouvait vendre des barres de longueur k (pour $1 \leq k \leq n$), et ce prix s'avère ne pas avoir de relation simple avec k. Le problème est donc de décider comment découper la barre initiale de n carreaux en des barres plus petites pour maximiser le prix de vente total. Pour simplifier, on néglige le coût de la découpe et de l'emballage.

Dans tout le problème, on considérera que l'on dispose d'un tableau `t`, indicé de 0 à n, tel que `t[k]` soit le prix de vente d'une barre de longueur `k`, et que le prix d'un morceau de taille 0 est 0 (autrement dit que `t[0] = 0`). Remarquez que si `t[n]` est suffisamment grand, la solution optimale peut très bien être de ne pas découper la barre.

On donne un exemple de tableau `t` pour $n = 10$.

```
int t[10] ={ 0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 26};
```

1. Identifier le découpage en trois blocs qui permet d'obtenir un prix de vente optimal sur l'exemple.
2. On fixe $n \in \mathbb{N}$. Une découpe d'une barre de taille n en k morceaux est la donnée d'un k-uplet $(c_1, c_2, \ldots, c_k)$ tel qu'on aura une barre de taille c_i pour chaque $1 \leq i \leq k$. Avec cette notation, que vaut $\sum_{i=1}^{k} c_i$?
3. Si on voulait calculer le prix de chaque découpe afin d'en déterminer la valeur maximale, combien de cas devrait-on considérer ?
4. Pour arriver à une solution efficace, la première étape est de remarquer qu'une solution du problème (i.e. une découpe optimale d'une barre de taille n) "contient" des solutions à des versions plus petites du même problème. Quel paradigme de programmation semble ici adapté ?
5. On peut supposer (sans perte de généralité), qu'à chaque étape d'une solution optimale, on découpe ce qui reste de la barre en un bloc de taille k ($1 \leq k \leq n$) qui fera partie de la découpe finale et une nouvelle barre qui pourra à nouveau être découpée. Justifier que cette nouvelle barre (de longueur $n-k$) devra elle aussi être découpée de manière optimale.
6. On note $v(n)$ la prix de vente optimal pour une barre de n " carreaux" pour $n \in \mathbb{N}$. Que vaut $v(0)$? Donner une relation de récurrence pour $v(n)$.
7. Proposer un algorithme qui calcule $v(n)$ sur l'entrée n.
8. Quelle est la complexité de votre algorithme ?

2 Partie B : en C

Dans cet exercice, on considère des arbres binaires non étiquetés. On dit qu'un arbre est binaire strict si tout nœud interne a exactement deux enfants, ou autrement dit si tout nœud (interne ou non) a zéro ou deux enfants. On dit qu'un arbre est parfait s'il est binaire strict et que toutes ses feuilles sont à même profondeur. On dit qu'il est presque-parfait si tous les niveaux de profondeur sont remplis par des nœuds, sauf éventuellement le dernier, rempli par la gauche.

1. Combien existe-t-il d'arbres presque-parfaits de taille 6 ? Les représenter graphiquement.
2. Citer une structure de données qui peut s'implémenter avec des arbres presque-parfaits.
3. Proposer une définition par induction d'un arbre parfait de hauteur n, pour $n \in \mathbb{N}$. Justifier rigoureusement que cette définition coïncide avec celle de l'énoncé.

On implémente un arbre binaire en C par le type arbre défini de la manière suivante :

```
struct Noeud {
  struct Noeud* gauche;
  struct Noeud* droite;
};
typedef struct Noeud arbre;
```

Ainsi, un arbre contient un pointeur vers son fils gauche et son fils droit.

4. Écrire une fonction int hauteur (arbre* a) qui prend en argument un pointeur vers un arbre et calcule et renvoie sa hauteur.
5. En déduire une fonction bool est_parfait(arbre* a) qui prend en argument un pointeur vers un arbre et renvoie un booléen qui vaut true si et seulement si l'arbre pointé par a est parfait.
6. Écrire une fonction arbre* plus_grand_presque_parfait(arbre* a) qui prend en argument un pointeur vers un arbre a et renvoie un pointeur vers le plus grand sous-arbre de a qui est presque-parfait. La complexité de cette fonction doit être linéaire en la taille de l'arbre a.

Indication : que peut-on dire des enfants gauche et droit d'un arbre presque-parfait de hauteur h ?