TD 04 – Programmation dynamique

Exercice 1. *Plus grand carré de 1*

1. Donner un algorithme de programmation dynamique pour résoudre le problème suivant :

   Entrée : une matrice A de taille $n \times m$ où les coefficients valent 0 ou 1.

   Sortie : la largeur maximum K d'un carré de 1 dans A, ainsi que les coordonnées (I, J) du coin en haut à gauche d'un tel carré (autrement dit pour tout i, j, $I \leq i \leq I + K - 1$, $J \leq j \leq J + K - 1$, $A[i, j] = 1$).

2. Quelle est sa complexité ?

Exercice 2. *Bibliothèque*

La bibliothèque planifie son déménagement. Elle comprend une collection de n livres $b_1, b_2, \ldots, b_n$. Le livre b_i est de largeur w_i et de hauteur h_i. Les livres doivent être rangés dans l'ordre donné (par valeur de i croissante) sur des étagères identiques de largeur L.

1. On suppose que tous les livres ont la même hauteur $h = h_i, 1 \leq i \leq n$. Montrer que l'algorithme glouton qui range les livres côte à côte tant que c'est possible minimise le nombre d'étagères utilisées.

2. Maintenant les livres ont des hauteurs différentes, mais la hauteur entre les étagères peut se régler. Le critère à minimiser est alors l'encombrement, défini comme la somme des hauteurs du plus grand livre de chaque étagère utilisée.

   (a) Donner un exemple où l'algorithme glouton précédent n'est pas optimal.

   (b) Proposer un algorithme optimal pour résoudre le problème, et donner son coût.

3. On revient au cas où tous les livres ont la même hauteur $h = h_i, 1 \leq i \leq n$. On veut désormais ranger les n livres sur k étagères de même longueur L à minimiser, où k est un paramètre du problème. Il s'agit donc de partitionner les n livres en k tranches, de telle sorte que la largeur de la plus large des k tranches soit la plus petite possible.

   (a) Proposer un algorithme pour résoudre le problème, et donner son coût en fonction de n et k.

   (b) On suppose maintenant que la la taille d'un livre est en $2^{o(kn)}$. Trouver un algorithme plus rapide que le précédent pour répondre à la même question.

Exercice 3. *Arbres binaires de recherche optimaux*

Un *arbre binaire de recherche* est une structure de données permettant de stocker un ensemble de clés ordonnées $x_1 < x_2 < \ldots < x_n$, pour ensuite effectuer des opérations du type *rechercher*, *insérer* ou *supprimer* une clé. Il est défini par les propriétés suivantes :
(i) C'est un arbre où chaque nœud a 0, 1 ou 2 fils, et où chaque nœud stocke une des clés.
(ii) Etant donné un nœud avec sa clé x, alors les clés de son sous-arbre gauche sont strictement inférieures à x et celles de son sous-arbre droit sont strictement supérieures à x.

La figure 1 représente un arbre binaire de recherche pour les clés $x_1 < x_2 < x_3 < x_4 < x_5 < x_6 < x_7 < x_8 < x_9$.
Les requêtes auxquelles on s'intéresse ici sont les *recherches* de clés. Le coût de la recherche d'une clé x correspond au nombre de tests effectués pour la retrouver dans l'arbre en partant de la racine, soit exactement la profondeur de x dans l'arbre, plus 1 (la racine est de profondeur 0).
Pour une séquence fixée de recherches, on peut se demander quel est l'arbre binaire de recherche qui minimise la somme des coûts de ces recherches. Un tel arbre est appelé *arbre binaire de recherche optimal* pour cette séquence.

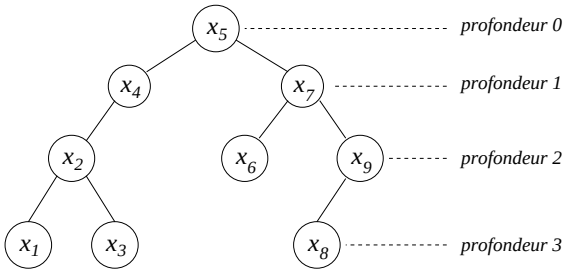

FIGURE 1 – Exemple d'arbre binaire de recherche.

1. Pour un arbre binaire de recherche fixé, le coût de la séquence ne dépend clairement que du nombre de recherches pour chaque clé, et pas de leur ordre. Pour $n = 4$ et $x_1 < x_2 < x_3 < x_4$, supposons que l'on veuille accèder une fois à x_1, 9 fois à x_2, 5 fois à x_3 et 6 fois à x_4. Trouver un arbre binaire de recherche optimal pour cet ensemble de requètes.

2. Donner un algorithme en temps $\mathcal{O}(n^3)$ pour construire un arbre binaire de recherche optimal pour une séquence dont les nombres d'accès aux clés sont $c_1, c_2, \ldots, c_n$ (c_i est le nombre de fois que x_i est recherché). Justifier sa correction et sa complexité.

Indication : pour $i \leq j$, considérer $t[i, j]$ le coût d'un arbre de recherche optimal pour les clés $x_i < ... < x_j$ accédées respectivement $c_i, \ldots, c_j$ fois.