Correction de l'examen d'Algorithmique et Programmation

Math312

1 Programmation en C [3 pts]

Cet exercice est juste un rappel de ce qui s'est passé avant novembre. Une méthode découlait logiquement de la relation binomiale proposée, qui a été ignorée par bon nombre d'étudiant. Il ne fallait évidemment pas non plus se lancer dans une réccurence brutale (comme Fibbonacci).

Voici un programme en C qui demande à l'utilisateur un entier n et affiche les n premières lignes du triangle de Pascal. On place les coefficients dans un tableau à l'aide de la propriété $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$.

```
#include <stdio.h>
#define MAX 100

int main(void){
  int i,j,n; int T[MAX][MAX];

  printf("Entrez le nombre de lignes désiré (<100)");
  scanf("%d",&n);

  for(i=0;i<n;i++){
    T[i][0]=1;
    T[i][i]=1; }

  for(i=1;i<n;i++)
    for(j=1;j<i;j++)
      T[i][j]= T[i-1][j-1]+T[i-1][j];

  for(i=0;i<n;i++){
    for(j=0;j<i+1;j++)
      printf("%3d ",T[i][j]);
    printf("\n");}
return 0; }
```

On aurait pu économiser de la place mémoire en ne retenant à tout moment que 2 lignes du tableau, mais le programme perd alors en clarté.

2 Tris [1,5 pts]

Cet exercice sur les tris est presque une question de cours. Il suffit de connaître la complexité des tris standard pour résoudre l'exercice très vite. Bien sur, il n'est pas demandé d'écrire chacuns des algorithmes en détail.

Étant donné un tableau de n nombres, on veut obtenir les i plus grands d'entre eux triés. Voici trois méthodes :

(a) Trier les n nombres et renvoyer les i plus grands :

On trie le tableau à l'aide de l'un des tris efficaces (en $\mathcal{O}(n \log n)$) puis on retourne les valeurs des cases d'indices $n - i$ à n, ce qui prend un temps en $\mathcal{O}(i)$. La complexité est donc de $\mathcal{O}(n \log n + i) = \mathcal{O}(n \log n)$.

(b) Trouver le i-ème plus grand élément (à l'aide d'un algorithme linéaire), partitionner et trier les i plus grands :

Il est précisé que l'on utilise un algo linéaire pour trouver le i-ème élément, qui prend donc un temps $\mathcal{O}(n)$. On partitionne les éléments pour séléctionner ceux qui sont supérieurs à l'élément isolé, il nous suffit de comparer chaque élément 1 fois : $\mathcal{O}(n)$. Enfin, on trie ces i éléments en $\mathcal{O}(i \log i)$. La complexité finale est $\mathcal{O}(n + i \log i)$, que l'on ne peut pas simplifier sans avoir de condition plus précise sur i.

(c) Construire un tas avec les n nombres et extraire la racine i fois :

La construction du tas (qui contient tout le tableau) coûte $\mathcal{O}(n \log n)$. Chaque extraction de racine coûte $\mathcal{O}(\log n)$. La complexité finale est donc $\mathcal{O}((n + i) \log n) = \mathcal{O}(n \log n)$.

Clairement, la meilleure méthode est la (b).

3 Algorithmes gloutons [4,5 pts]

La bibliothèque prévoit de refaire ses étagères. Elle comprend une collection de n livres $b_1, b_2, \ldots, b_n$. Chaque livre b_i a une largeur w_i et une hauteur h_i. Les livres doivent être rangés dans l'ordre fixé (par valeur de i croissante[1]) sur des étagères de largeur L.

[1] Je ne sais pas pourquoi, beaucoup d'étudiants n'ont pas tenu compte de cette restriction. Sans elle, le problème est bien plus difficile

1. On ne se soucie d'abord pas de la hauteur des livres. Pour minimiser le nombre d'étagères, l'algorithme glouton suivant semble très naturel :

   S `Algo range livres :`
   $i \leftarrow$ `1 ;`
   $etagere \leftarrow$ `1 ;`
   $largeur\_libre \leftarrow L$ `;`
   `tant que` $i \leq n$ `faire`
     `si` $w_i \leq largeur\_libre$
       $largeur\_libre \leftarrow largeur\_libre - w_i$
     `sinon`
       $etagere \leftarrow etagere + 1$
       $largeur\_libre \leftarrow L - w_i$
     `fin si`
     `placer le livre` b_i `sur l'étagere` $etagere$
   `fin tant que`

2. Montrons que cet algorithme est optimal : supposons qu'il existe une solution *Opt* qui utilise moins d'étagères que la solution S proposée par cet algorithme. Alors, il existe un premier livre b_i placé par *Opt* sur une étagère k strictement inférieure à l'étagère k' choisie par la solution S (on choisit i minimum). Soit b_j le premier livre placé sur l'étagère k dans la solution S ($j < i$). Par minimalité de i, *Opt* place b_j sur la même étagère k. *Opt* place donc tous les livres de b_j à b_i (et peut-être d'autres encore) sur une seule étagère. Donc $\sum_{l=j}^{i} w_l < L$. Par conséquent, l'algorithme glouton pouvait aussi placer le livre b_i sur l'étagère k, une contradiction. Donc S est optimale. □

3. On peut maintenant régler la hauteur des étagères. Nous allons donc chercher à minimiser l'encombrement, défini comme la somme des hauteurs du plus grand livre de chaque étagère utilisée. On voit dans l'exemple suivant que l'algorithme glouton précédent ne trouve pas la solution optimale.

   On pose $L = 3, n = 3, w_1 = 2, w_2 = w_3 = 1, h_1 = 1, h_2 = h_3 = 7$. L'algorithme glouton place b_1 et b_2 sur la première étagère, de hauteur 7, et b_3 sur une deuxième étagère, de hauteur 7 aussi, fournissant un encombrement total de 14. La solution optimale fournit un encombrement total de 8 en plaçant b_1 seul sur la première étagère et b_2 et b_3 sur la deuxième étagère.

4 Diviser pour régner [4 pts]

Soient $A[1..n]$ et $B[1..n]$ deux tableaux triés par ordre croissant. On cherche à trouver l'élément médian de ces deux tableaux (élément qui a autant d'éléments

supérieurs stricts que d'éléments inférieurs ou égaux[2]).

Voici un algorithme diviser pour régner qui trouve le médian en $\mathcal{O}(\log n)$:

```
Algo médian(A[1..n], B[1..n]) :
  si n = 1
    si A[1] ≤ B[1] renvoyer A[1]
    sinon renvoyer B[1]
    fin si
  sinon
    x ← ⌈n/2⌉
    si A[x] ≤ B[x]
      renvoyer médian(A[(x+1)..n], B[1..⌊n/2⌋])
    sinon
      renvoyer médian(A[1..⌊n/2⌋], B[(x+1)..n])
    fin si
  fin si
```

On remarque que cet algo s'appelle lui même sur des tableaux de taille inférieure de moitié. On a donc $C(n) = \mathcal{O}(1) + C(\lfloor \frac{n}{2} \rfloor)$. On en déduit facilement que $C(n) = \mathcal{O}(logn)$.

[2]En fait, ceci est un raccourci. Le médian est un élément m tel qu'il existe une partition de l'ensemble des éléments en deux ensembles E_p et E_g telle que $|E_p| = |E_g|$, $\forall x \in E_p, x \leq m$ et $\forall y \in E_g, y \geq m$

5 Programmation dynamique [7 pts]

Cette année, la mère Noël s'est exceptionnellement proposée pour aider le père Noël. Ils vont donc devoir à eux deux passer par toutes les cheminées pour déposer leurs cadeaux. Ils aimeraient pouvoir réduire la longueur de leurs trajets. Le problème est le suivant : étant donnés un point de départ x_0 et n cheminées $x_1, \dots x_n$, trouver deux chemins $p_0, p_1, \dots, p_{n_p}$ et $m_0, m_1, \dots, m_{n_m}$ avec $p_0 = m_0 = x_0$ et $\{x_1, \dots, x_n\} = \{p_1, \dots, p_{n_p}\} \cup \{m_1, \dots, m_{n_m}\}$ qui minimisent

$$\sum_{i=1}^{n_p} d(p_{i-1}, p_i) + \sum_{j=1}^{n_m} d(m_{j-1}, m_j)$$

où $d(x_i, x_j)$ désigne la distance entre x_i et x_j.

En fait, pour profiter du décalage horaire, le père Noël a l'habitude de voyager plutôt d'est en ouest, en partant de la ligne de changement de date. Pour simplifier le problème, nous allons donc supposer que ni le père Noël, ni la mère Noël ne pourront revenir vers l'est. De plus, nous supposons notre connaissance des positions des cheminées si précise que 2 cheminées n'ont jamais la même longitude [3].

1. Dans la suite, on considère que les x_i sont triés par longitude décroissante. Comme les parents Noël voyagent d'est en ouest, ils visitent les x_i dans l'ordre des i croissants grâce à ce tri. Cela simplifiera les algorithmes.
2. Supposons que le père Noël a visité en dernier la cheminée x_i et la mère Noël la cheminée x_j. La prochaine cheminée à visiter est la cheminée x_k avec $k = \max(i, j)+1$. Nous nous trouvons dans un sous problème du problème principal. On désigne par $\mathcal{L}(i, j, [k..n])$ la longueur de la solution optimale de ce sous-problème. En particulier, $\mathcal{L}(0, 0, [1..n])$ est la longueur de la solution optimale du problème initial. On remarque l'inégalité suivante :

   $$\mathcal{L}(i, j, [k..n]) = \min \left\{ \begin{array}{l} \mathcal{L}(k, j, [k+1..n]) + d(x_i, x_k), \\ \mathcal{L}(i, k, [k+1..n]) + d(x_j, x_k) \end{array} \right\}$$

   On peut aussi remarquer que si i ou j vaut n, il ne reste plus de cheminée à visiter et $\mathcal{L}(i, j, [])$ vaut 0.
3. On définit une matrice $(n+1) \times n+1$ $\mathcal{L}$ qui contient les valeurs $\mathcal{L}(i, j, [k..n])$ (k se déduit de i et j) que l'on remplit avec l'algorithme de programmation dynamique suivant :

[3] Rappelons que la longitude est l'angle par rapport au méridien de Greenwich. Il s'agit d'une valeur entre $-180°$ et $+180°$ qui détermine la position est-ouest d'un point à la surface de la Terre. (La longitude décroit d'est en ouest) La ligne de changement de date correspond à une démarcation dans la zone limite entre $-180°$ et $+180°$ qui correspond à la limite entre les fuseaux horaires +12h et −12h.

```
Algo remplit matrice :
  pour i de 0 à n
    L[i][n] ← 0
    L[n][i] ← 0
  fin pour

  pour i de n − 1 à 0
    pour j de n − 1 à 0
      k ← max(i, j) + 1
      L[i][j] ← min (L[k][j] + d(x_i, x_k), L[i][k] + d(x_j, x_k))
    fin pour
  fin pour
```

Pour récupérer la distance minimale du parcours des parents Noël, il suffit de lire la valeur de $\mathcal{L}[0][0]$. Pour remplir cette matrice, nous avons calculé deux distance pour chaque case (sauf les dernières ligne et colonne), nous avons donc fait $2n^2 = \mathcal{O}(n^2)$ appels à la fonction distance.

4. Pour trouver les deux chemins $p_0, p_1, \ldots, p_{n_p}$ et $m_0, m_1, \ldots, m_{n_m}$, il nous faut trouver le chemin dans la matrice qui conduit d'un bord (i ou $j = n$) à la case $0, 0$. Pour cela, on peut partir de $0, 0$ et pour chaque case i, j, vérifier si $\mathcal{L}[i][j] = \mathcal{L}[k][j] + d(x_i, x_k)$ auquel cas le père Noël visite la cheminée k, et on se déplace vers la case k, j. Sinon, c'est que c'est la mère Noël qui a visité la cheminée k et on se déplace vers la case i, k. La complexité de cette étape est $\mathcal{O}(n)$.