Licence de Sciences et Technologies, option Informatique
Algorithmique Avancée
TD 5: octobre/novembre 2006
Programmation Dynamique

1. On considère le calcul du nombre de *partitions* d'un entier positif n. Une partition est une division de n en parts où l'ordre des parts n'est pas pris en compte. Donc, par exemple, 4 a les 5 partitions, $\{4\}$, $\{3,1\}$, $\{2,2\}$, $\{2,1,1\}$ et $\{1,1,1,1\}$.

   (a) Dans le cour on vous a donné un algorithme de ce calcul en utilisant une récurrence sur le nombre de partitions avec plus grande part i ($Part(n, max) = \sum_{i=1}^{max} Part(n - max, i)$),

   (b) En donner un autre avec une récurrence sur le nombre de partitions en j parts,
   Soit $Part'(n,m)$ le nombre de partitions de n en m parts ($>= 1$), si j'enlève le premier élément de chaque part (donc m en total), j'ai $n - m$ restant à partitionner dans les j parts de taille supérieure à 1; donc la récurrence est $Part'(n,m) = \sum_{j=1}^{m} Part'(n-m, j)$ avec les conditions sur les bords $Part'(n,n) = 1$ et $Part'(n,m) = 0$ pour $m > n$.

   (c) Quelle est la différence entre les deux récurrences?
   Aucune sauf le changement des noms i à j et max à m.
   Pourquoi?
   Si on considère la partition représentée géométriquement par une figure où chaque part équivaut à une barre verticale de hauteur égale à sa valeur et avec les barres en ordre décroissant de gauche à droite, la condition que la figure soit valide est préservée par une réflexion dans la diagonale; et cette réflexion transforme une partition avec valeur maximum max en une partition avec max parts.
   Par exemple, le diagramme:

   ```
   *
   **
   **
   *****
   *****
   *********
   ```

   peut être interprété comme la partition $(1, 2, 2, 5, 5, 9)$ (en regardant les lignes) ou comme $(1, 1, 1, 1, 3, 3, 3, 5, 6)$ (en regardant les colonnes.

   (d) Donner un autre algorithme plus performant qui a comme paramètre un majorant de la plus grande part (ou du nombre de parts) au lieu du nombre exact.
   La partie importante est la récurrence valable dans les cas où le majorant (maj) est supérieur ou ègal à n: $Part(n, maj) = Part(n, maj -$

$1) + Part(n - maj, maj)$
(le $Part(n, maj - 1)$ est pour les cas où il n'y a pas de part maj et, donc, $maj - 1$ est un majorant; le $Part(n - maj, maj)$ est pour le cas où il y a un part maj et le reste $(n - maj)$ sera ensuite partitionné, avec le majorant max toujours valable. En Exalgo ça peut s'écrire:

```
procedure Remplir (ref T:carre);
var n:entier;
var maj:entier;
debut
        T[0][0]:=1;
        pour n allant de 1 a 100 par pas de 1 faire
            T[n][0]:=0
        finpour;
        pour maj allant de 1 a 100 par pas de 1 faire
                pour n allant de 0 a maj-1 par pas de 1 faire
                        T[n][maj]:=T[n][n]
                finpour;
                pour n allant de maj a 100 par pas de 1 faire
                        T[n][maj]:=T[n][maj-1]+T[n-maj][maj]
                finpour
        finpour
fin
finprocedure;
```

2. Je veux découper une pièce de bois en sections de superficie maximum 1000 cm^2. Le bord supérieur et le bord inférieur de la pièce sont des lignes brisées $A_1, \cdots, A_n$ d'une part et $B_1, \cdots, B_m$ de l'autre, et les extrémités latérales sont les segments A_1, B_1 et A_n, B_m. Je dois couper suivant des traits rectilignes joignant certains des points A_i à certains des points B_j sans faire deux coupes qui se croisent, sauf à un des points A_i ou B_j.
Il y a deux tableaux $Cout$ et $Droite$ à deux dimensions avec les données: $Cout[i][j]$ le coût de faire une coupe de A_i à B_j et $Droite[i][j]$ la superficie en cm^2 de la partie à droite de cette coupe.
Donner un algorithme qui calcule le coût minimum de la découpe (en utilisant la programmation dynamique).

Je calcule le coût total $Tot[i][j]$ de la découpe de la partie à droite de la ligne (A_i, B_j); ce coût est le minimum sur tous les $i' >= i$ et $j' >= j$, avec une superficie de au plus 1000 cm^2 entre les deux coupes (i, j) et (i', j'), de $Cout[i'][j'] + Tot[i'][j']$
Une petite complication (inattendue) vient de la possibilité qu'il soit impossible de découper la partie à droite d'une ligne (A_i, B_j) en parties de la taille demandée, même si la pièce initiale était capable d'être découpée. Ce problème est traité par le tableau `Poss[][]`; `Poss[i][j]` est *vrai* si au moins un système de découpage a été trouvé pour la partie à droite de (A_i, B_j).

```
procedure Remplir (ref T:tab);
var i:entier;
var j:entier;
var jj:entier;
debut
   T[n][m]:=0;
   Poss[n][m]:=vrai;
   i:=n;
   tant que i>0 faire
      si i=n alors j:=m-1 sinon j:=m finsi;
      tant que j>0 faire
         Poss[i][j]:=0;
         pour ii allant de i a n par pas de 1 faire
            si i=ii alors jj:=j+1 sinon jj:=j finsi;
            tant que jj<=m et droite[ii][jj]>=droite[i][j]-1000 faire
               si Poss[ii][jj] alors
                  si Poss[i][j] alors
                     T[i][j]:=min(T[i][j],cout[i][j]+T[ii][jj]);
                  sinon
                     T[i][j]:=cout[i][j]+T[ii][jj];
                     Poss[i][j]:=vrai
                  finsi
               finsi;
            jj:=jj+1
            fintantque
         finpour;
         j:=j-1
      fintantque;
      i:=i-1
   fintantque
fin
finprocedure;
```