Conception d'algorithmes

Programmation dynamique

V. Chevrier

Programmation dynamique

- Approche bottom-up
- On calcule les solutions de sous problèmes, on les stocke et on les réutilise pour calculer la solution d'un problème plus grand
- « programmation » : méthode de tabulation, stockage des solutions intermédiaires
  - La « table » peut être une table, une matrice, un arbre, …

Un exemple concret : Fibonnacci

- $F_0 = 0, F_1 = 1, \ \forall\, n \geq 2, \quad F_n = F_{n-1} + F_{n-2}$
- 0, 1, 1, 2, 3, 5, 8, etc.
- Calcul du prochain = somme des deux derniers
- D'où un algo itératif = ?

Un exemple concret : Fibonnacci

- $F_0 = 0, F_1 = 1, \forall n \geq 2, \quad F_n = F_{n-1} + F_{n-2}$
- 0, 1, 1, 2, 3, 5, 8, etc.
- Calcul du prochain = somme des deux derniers
- D'où un algo itératif:

```
fonction fib(n:entier):entier
    prev ←-1, res ←1
    Pour i de 0 à n
            sum ← res+prev
            prev ← res
            res ← sum
    fp
    retourner res
```

Un exemple concret : Fibonnacci

- Complexité ?

Un autre exemple

- Coefficient binomial $\binom{n}{k} = C_n^k = \frac{n!}{k!(n-k)!}$.
- Algorithme « naif »:

```
private static int bin(int n, int k) {
        return fact(n)/(fact(k)*fact(n-k));
}
```

- Problème ?

Un autre exemple

- Coefficient binomial $\binom{n}{k} = C_n^k = \frac{n!}{k!(n-k)!}.$
- Algorithme « naif »:

```
private static int bin(int n, int k) {
        return fact(n)/(fact(k)*fact(n-k));
}
```

- Problème ?

  14! 1278945280
  15! 2004310016
  16! 2004189184
  17! -288522240
  18! -898433024

- Représentation des entiers → évitons d'utiliser n!

Reformulons

- Propriétés (wikipedia)

$$\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$$

- Nouvel algorithme
  - C(n,m) =1 si m = 0
  - C(n,m) = 1 si n = m
  - C(n,m) = C(n-1,m)+C(n-1,m-1)

Complexité ?

Reformulons

- Propriétés (wikipedia)

$$\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$$

- Nouvel algorithme
  - C(n,m) =1 si m = 0
  - C(n,m) = 1 si n = m
  - C(n,m) = C(n-1,m)+C(n-1,m-1)

Complexité ?

pas mieux que tout à l'heure

Reformulons encore !

- Idée : ne pas refaire deux fois les mêmes calculs
- Reprenons depuis le début :
- C(0,0)=1
- C(1,0)=1, C(1,1)=1
- C(2,0)=1, C(2,1)=2, C(2,2) = 1
- Etc

n\m	0	1	2	3	4	5	6	7	8	9
0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
5	1	5	10	10	5	1				
6	1	6	15	20	15	6	1			
7	1	7	21	35	35	21	7	1		
8	1	8	28	56	70	56	28	8	1	
9	1	9	36	84	126	126	84	36	9	1

- Triangle de Pascal
- Valeurs correspondent à C(n,m)
- Donc calculons depuis le début !
  - On utilise un tableau (taille n) pour stocker les valeurs intermédiaires (géré comme une liste)
  - On remplit successivement le tableau

n\m	0	1	2	3	4	5	6	7	8	9
0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
5	1	5	10	10	5	1				
6	1	6	15	20	15	6	1			
7	1	7	21	35	35	21	7	1		
8	1	8	28	56	70	56	28	8	1	
9	1	9	36	84	126	126	84	36	9	1

Complexité ?

```
private static int bino(int n, int k) {
    int bino [] = new int[n+1];
    bino[0]=1;
    for (int i = 1; i<=n; i++){
        for (int j = i-1; j>0; j--){
            bino[j] = bino[j]+ bino[j-1];
        }
    }
    return bino[k];
}
```

Programmation dynamique

- Le problème se décompose en sous-problèmes du même type
- Avec 2 propriétés
  - Sous-structure optimale: *la solution optimale du problème est composée de solutions optimales aux sous-problèmes*
  - Recouvrement des sous-problèmes: *des sous problèmes distincts partagent certains de leurs sous-problèmes respectifs*

Principe

- Définir des sous problèmes
- Définir la relation de récurrence qui relie les problèmes
- Identifier et résoudre les cas de base

Plus longue sous séquence commune

- Soit deux chaines X =<x_1,x_2,x_3, …,x_m> et Y = <y_1,y_2,y_3, …,y_k>

Trouver la longueur de la plus longue sous-séquence commune PLSC

- Exemple
  - X = "qisuddsfhes", Y="ujjjdd »
  - Plus longue sous-séquence: udd (3)

Plus longue sous séquence commune

a. Définition du problème:

Soit $D_{i,j}$ la PLSC de $x_{1..i}$ et $y_{1..j}$

b. Récurrence

Cas trivial

Plus longue sous séquence commune

a. Définition du problème:

Soit $D_{i,j}$ la PLSC de $x_{1..i}$ et $y_{1..j}$

b. Récurrence

Si $x_i=y_j$ alors ils contribuent à la PLSC ➔$D_{i,j} = D_{i-1,j-1} + 1$

Sinon l'un ou l'autre peut être ignoré ➔ $D_{i,j} = \max (D_{i-1,j}, D_{i,j-1})$

Cas trivial $D_{i,0} = D_{0,j} =0$

Un autre exemple (détaillé)

- Le pb du sac à dos

Comparaison

- Algorithme glouton
  - Problèmes d'optimisation
  - Optimum global formé d'une suite de choix localement optimal
  - Solution par itération
- Divide-and-conquer
  - Problèmes composés de sous-problèmes similaires
  - Solution optimale composée de solutions optimales à des problèmes sans recouvrements
  - Solution par récursion
- Programmation dynamique
  - Problèmes d'optimisation composés de sous-problèmes similaires
  - Solution optimale composée de solutions optimales à des problèmes avec recouvrements
  - Solution par itération sur une table

Memoization (sans typo)

- Résoudre le problème de manière top-down en mémorisant les calculs
- Principe:
  - On reprend l'algorithme naturel récursif inefficace
  - On reprend le principe de la programmation dynamique (ie de stocker les solutions dans une table)
    - Une entrée dans la table par sous-problème
    - Initialement on associe la valeur « pas résolu »
  - On regarde à chaque sous-problème/appel si c'est déjà résolu → solution mémorisée sinon on applique le calcul récursif
-

Memoization (sans typo)

- Exemple
  - methode fibonacci (n)

```
Si connu[n] alors retourner valeur [n]
        Sinon
                Si n <=1 alors retourner n
                Sinon
                        valeur[n] ← fibonnacci(n-1)+fibonacci(n-2)
                        connu[n] ← vrai
                        retourner valeur [n]
                Fsi
Fsi
```

Algorithme général

- Methode monCalculM(n)
  - Si connu(n) alors
    - Retourner solution(n)
  - Sinon
    - solution(n) ←monCalculR(n)
    - Retourner solution(n)
  - Fsi
- Methode monCalculR(n)
  - Si « cas trivial » alors retourner soltriviale
    - Sinon
      - Retourner appel récursif AVEC monCalculM
  - Fsi
- Mise en œuvre java ..?

FIn

Later …

- Faire la monnaie en code ar pas mal

Problèmes

- La monnaie

http://www.montefiore.ulg.ac.be/~piater/courses/INFO0902/notes/basic-algos/foil28.xhtml

Returning Change

A formulation and solution similar to the 0-1 knapsack problem:

$$\min_{T \subseteq S} |T| \text{ avec } \sum_{i \in T} v_i = V$$

For the sum to be always correct, we treat the one-cent coins specially:

- There is an unlimited supply.
- The algorithm uses them only at the end to fill up any remaining difference.

Subproblem: Calculate $C[k, v]$, the maximum size of a subset $T_k \subseteq S_k$ of total value equal to v:

$$C[k,v] = \begin{cases} C[k-1,v] & \text{if } v_k > v \\ \min\{C[k-1,v], C[k-1,v-v_k]+1\} & \text{otherwise} \end{cases}$$

Example 4.

$S = \{4, 3, 3\}$ cents, $V = 6$ cents

- Voir copie site Piater
  - 01 Knapsack
  - Plis longue sous séquence
- Site Bpreiss
  - Justifiaction de paragraphe
- Cormen
  - Chaien de multiplication de matrice
  - Plus longue soussequence ..
  - Triangulation optiale de polygone