# Chapitre 5

## Programmation dynamique

*«...toute politique optimale est composée de sous-politiques optimales»*

Bellman





## Eléments de définition

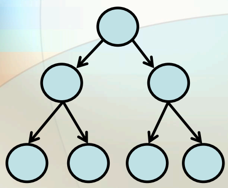
*Principe :* La *programmation dynamique* résout les problèmes en combinant les solutions de sous-problèmes. Elle est applicable lorsque les sous-problèmes ne sont pas indépendants.

Un algorithme de programmation dynamique résout chaque sous problème une seule fois et mémorise sa solution dans un tableau. Cela évite ainsi le re calcule de la solution chaque fois que le sous-problème est rencontré.

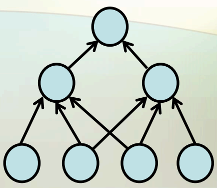




## Deux approches



Diviser pour régner



Programmation dynamique

## Illustration : nombres de Fibonacci

**Nombres de Fibonacci :**

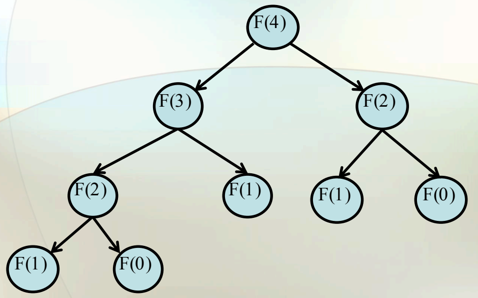
F(0) = 1; F(1) = 1;  F(n) = F(n-1) + F(n-2)

*Fibonacci*(n)
si n=1 alors retourner 1;
si n=2 alors retourner 1;
sinon retourner *Fibonacci*(n-1) + *Fibonacci*(n-2);





## Exécution approche récursive






## Algorithmes

*Fib*(n)
Si Fib(n) est dans la table retourner table[n];
si n <=2 retourner 1;
sinon
    table[n] ← Fib(n-1) + Fib(n-2);
    retourner table[n];

*Fib*(n)
F[0] = 1; F[1] = 1;
Pour (i=2 à n) faire
    F[i] = F[i-1] + F[i-2];
Retourner F[n];

## Illustration : Coefficients binomiaux

**Coefficient binomial :**

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \qquad 0 < k < n$$

$$\binom{n}{k} = 1, \qquad sinon$$

*Binomial* (n,k)
si ( k=0 ou k=n ) alors retourner 1;
sinon retourner Binomial(n-1,k-1) + Binomial(n-1,k)





## Triangle de Pascal

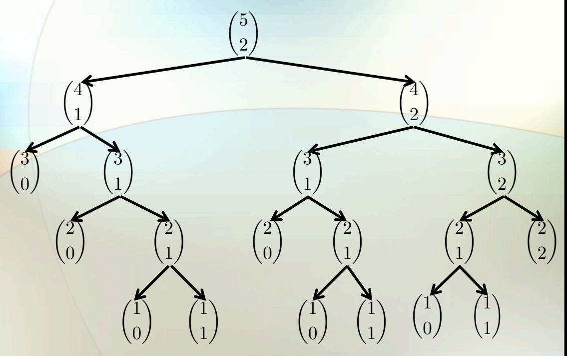
k

| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 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 |





## Arbre des appels récursifs



## Algorithme de prog. dynamique

Binomial (n,k)
Données : n,k : entier;
Initialisation : B[1,0] ← 1; B[1,1] ← 1

```
pour (i=2 à n) faire
        pour (j=0 à min(i,k)) faire
                si (j=0  ou j=i) alors B[i,j] ← 1;
                sinon B[i,j] ← B[i-1,j-1] + B[i-1,j];
retourner B[n,k];
```

**Complexité :** l'algorithme rempli la moitié du tableau de coté n et k, donc la complexité temporelle est en $\mathcal{O}(k.n)$ .





## Principe d'optimalité

Pour un problème d'optimisation, le **principe d'optimalité de Bellman** s'applique lorsque la solution optimale peut être obtenue à partir des solutions optimales des sous-problèmes.





## Processus de résolution

Le développement d'un algorithme de programmation dynamique peut être planifié dans une séquence de 4 étapes :

1)Caractériser la structure d'une solution optimale;

2)Définir récursivement la valeur d'une solution optimale;

3)Calculer la valeur d'une solution optimale;

4)Construire une solution optimale

## Problème du plus court chemin

**Problème :**

Soit un graphe G=(S,A), S l'ensemble de sommets, et A l'ensemble d'arcs.
Le poids de l'arc a est un entier naturel noté l(a).

La longueur d'un chemin est égale à la somme des longueurs des arcs qui le composent.

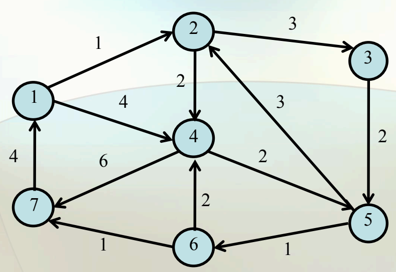
**Question :**

Déterminer pour chaque couple de sommets (si,sj), le plus court chemin, s'il existe, qui joint si à sj .





## Plus court chemin




Graphe G = (S,A)



## Algorithme de Floyd / Warshall

**Etape 1 (Caractérisation de la structure d'une solution optimale) :**

Si f est un chemin de longueur minimale joignant x à y et qui passe par z, alors il se décompose en deux chemins de longueur minimale. Le premier joint x à z et le second joint z à y.

On suppose les sommets numérotés : $s_1, s_2, \ldots s_n$ et pour tout k > 0, on considère la propriété $P_k$ suivante:

$P_k$(f): Tous les sommets de f, autres que son origine et son extrémité, ont un indice strictement inférieur à k.

## Expression récursive

**Lemme** **(Etape 2: Définition récursive d'une solution optimale )**
Les relations suivantes sont satisfaites par les $\delta_k$:

$$\forall(s_i, s_j), \quad \delta_1(s_i, s_j) = l(s_i, s_j) \quad ou \quad \infty.$$

$$\delta_{k+1}(s_i, s_j) = min[\delta_k(s_i, s_j), \delta_k(s_i, s_k) + \delta_k(s_k, s_j)]$$





## Algorithmes Floyd / Warshall

Etape 3 : Calcul d'une solution optimale

Plus court Chemin

Données : G un graphe orienté valué;

Initialisation pour k=1;
pour (k=2 à n) faire
    pour (i=1 à n) faire
        pour (j=1 à n) faire
            delta[i,j]=Min(delta[i,j],delta[i,k]+delta[k,j]





## Algorithmes Floyd / Warshall

Phase d'initialisation pour k=1

j

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | | 4 | | | |
| | | 0 | 3 | 2 | | | |
| | | | 0 | | 2 | | |
| i | | | | 0 | 2 | | 6 |
| | | 3 | | | 0 | 1 | |
| | | | | 2 | | 0 | 1 |
| | 4 | | | | | | 0 |

$$\forall(s_i, s_j), \quad \delta_1(s_i, s_j) = l(s_i, s_j) \quad ou \quad \infty.$$

## Algorithmes Floyd / Warshall

Matrice finale correspondant à $\delta(s_i, s_j)$ :

j

i

| | | | | | | |
|---|---|---|---|---|---|---|
| **0** | **1** | **4** | **3** | **5** | **6** | **7** |
| 10 | 0 | 3 | 2 | 4 | 5 | 6 |
| 8 | 5 | 0 | 5 | 2 | 3 | 4 |
| 8 | 5 | 8 | 0 | 2 | 3 | 4 |
| 6 | 3 | 6 | 3 | 0 | 1 | 2 |
| 5 | 6 | 9 | 2 | 4 | 0 | 1 |
| 4 | 5 | 8 | 7 | 9 | 10 | 0 |

$$\delta_{k+1}(s_i, s_j) = min[\delta_k(s_i, s_j), \delta_k(s_i, s_k) + \delta_k(s_k, s_j)]$$





## Algorithmes Floyd / Warshall

Etape 4 : Calcul effectif des chemins optimaux

Plus court Chemin

Données : G un graphe orienté valué;

Initialisation pour k=1;
pour (k=2 à n) faire
    pour (i=1 à n) faire
        pour (j=1 à n) faire
            si ( delta[i,j] > delta[i,k] + delta[k,j] ) alors
                delta[i,j] ← delta[i,k]+delta[k,j]
                suivant[i,j] ← suivant[i,k]





## Algorithmes Floyd / Warshall

Matrice 'suivant[i,j]' :

j

i

| | | | | | | |
|---|---|---|---|---|---|---|
| **1** | **2** | **2** | **2** | **2** | **2** | **2** |
| 4 | 2 | 3 | 4 | 4 | 4 | 4 |
| 5 | 5 | 3 | 5 | 5 | 5 | 5 |
| 5 | 5 | 5 | 4 | 5 | 5 | 5 |
| 6 | 2 | 2 | 6 | 5 | 6 | 6 |
| 7 | 7 | 7 | 4 | 4 | 6 | 7 |
| 1 | 1 | 1 | 1 | 1 | 1 | 7 |

## Algorithmes Floyd / Warshall

Etape 4 : Calcul effectif des chemins optimaux

Plus court Chemin ()

Données : suivant[n,n] matrice d'entiers; i,j :entier;

```
k ← i;
tant que (k != j) faire
        écrire (k, '');
        k ← suivant[k,j]
écrire j;
```





## Pour résumer

Nous avons introduit le principe d'optimalité de Bellman:

"toute solution optimale s'appuie elle-même sur des sous-problèmes résolus localement de façon optimale"

- Nous avons décrit un processus en 4étapes pour la conception d'algorithmes de programmation dynamique:
  - Structure d'une solution optimale;
  - Définition récursive d'une solution optimale;
  - Calcul de la valeur de la solution optimale;
  - Calcul de la solution optimale;
- Nous avons mis en oeuvre ces principes pour les problèmes du calcul des nombres de Fibonacci, du calcul des coefficients binomiaux et pour le calcul des chemin minimaux dans un graphe orienté valué.