Université de Monastir

Cours: Conception et Analyse d'Algorithmes

Chapitre 3: La programmation dynamique

Réalisé par:
Dr. Sakka Rouis

1

I. Classifications de paradigmes algorithmiques

- **Algorithme glouton** : construit une solution de manière incrémentale, en optimisant un critère de manière locale.

- **Diviser pour régner** : divise un problème en sous-problèmes indépendants (qui ne se chevauchent pas), résout chaque sous-problème, et combine les solutions des sous-problèmes pour former une solution du problème initial.

- **Programmation dynamique** : divise un problème en sous-problèmes qui sont non indépendants (qui se chevauchent), et cherche (et stocke) des solutions de sous-problèmes de plus en plus grands

2

II. Historique de la programmation dynamique

- Paradigme est développé par Richard Bellman en 1953.
- Technique de conception d'algorithme très générale et performante.
- Permet de résoudre de nombreux problèmes d'optimisation:
  - Problème de sac à dos
  - Problème de voyageurs de commerce

.

III. Principe de la programmation dynamique

Il est basé sur la décomposition du problème initial en problèmes plus simples et la résolution de ces sous-problèmes à partir des plus simples.

Le but du programmation dynamique est de trouver un **objet optimal** à partir d'un ensemble donné d'objets.

La programmation dynamique est un paradigme algorithmique qui résout un problème complexe en le divisant en sous-problèmes et stocke les résultats des sous-problèmes **pour éviter de calculer à nouveau les mêmes résultats**.

III. Principe de la programmation dynamique

Les deux propriétés principales d'un problème suggérant qu'il peut être résolu à l'aide de la programmation dynamique sont:

- Chevauchement de sous-problèmes

- Sous-structure optimale

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

Similairement au paradigme basé sur le principe diviser pour régner, la programmation dynamique combine des solutions à des sous-problèmes.

La programmation dynamique est principalement utilisée lorsque **des solutions des mêmes sous-problèmes** sont nécessaires à plusieurs reprises.

Dans la programmation dynamique, les solutions calculées aux sous-problèmes sont stockées dans une table de sorte qu'elles ne doivent pas être recalculées.

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

➔La programmation dynamique n'est pas utile lorsqu'il n'y a pas de sous-problèmes communs (qui se chevauchent) parce qu'il n'y a pas de raison de stocker les solutions si elles ne sont pas nécessaires à nouveau.

Exemples:

La recherche dichotomique n'a pas de sous-problèmes communs.

Le tri par fusion n'a pas de sous-problèmes communs.

Le tri rapide n'a pas de sous-problèmes communs.

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

Voici une implémentation **naïvement** (c-a-d la **transcription intuitive de sa définition**) de la suite de Fibonacci

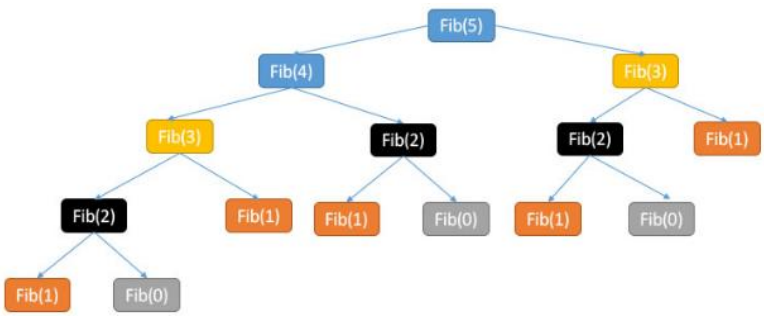
```
function fibo (n: entier) : entier
debut
        si (n =0 ou n=1 ) alors
                return n
        Sinon
                return fibo(n-1) + fibo(n-2)

        finsi
finFn
```

Nous remarquons que la **complexité algorithmique** de cette fonction est **exponentielle** ➔ **Solution inefficacie**

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

➔Pour calculer Fibo (5) on peut conclure que
Fib(3) est appelé (calculé) **2** fois.
Fib(2) est appelé (calculé) **3** fois.
Fib(1) est appelé (calculé) **5** fois.
Fib(0) est appelé (calculé) **3** fois.



9

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

➔**Idé**: Si nous avions stocké les valeurs de **Fib(0)**, **Fib(1)**, **Fib(2), Fib(3),… et Fib(n-1)**, alors au lieu de les calculer à nouveau, nous aurions pu réutiliser les anciennes valeurs stockées.

Il y a deux façons différentes de stocker les valeurs pour que ces valeurs puissent être réutilisées :

- **Mémoïsation** (Méthode descendante: du haut vers le bas)
- **Tabulation** (Méthode ascendante: de bas en haut)

10

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

Mémoïsation

Le programme mémoisé pour un problème est similaire à la version récursive avec une petite modification qui consiste à regarder dans un dictionnaire avant de calculer la solution.

Nous créons un dictionnaire vide. Chaque fois que nous avons besoin de la solution à un sous-problème, nous examinons d'abord le dictionnaire. Si la valeur pré-calculée est là, nous renvoyons cette valeur. Sinon, nous calculons la valeur et plaçons le résultat dans le dictionnaire afin qu'il puisse être réutilisé ultérieurement.

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

Mémoïsation

Voici la version mémoisée de la suite de Fibonacci

```
fonction fiboMem (n: entier) : entier
debut
        si (memo[n] existe) alors
                return memo [n]
        finSi
        si (n =0 ou n=1 ) alors
                return n
        Sinon
                memo[n] = fiboMem (n-1) + fiboMem (n-2)
                return memo[n];
        finsi
finFn
```

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

Mémoïsation

FiboMem (k) induit des appels récursifs seulement la première fois qu'elle est appelée.

Nombre d'appels non mémoïsés : n.
Temps d'un appel (sans compter les appels récursifs non mémoïsés) : O(n) du fait de l'addition entière FiboMem(k-1) + FiboMem(k-2).

➔ Complexité temporelle : $O(n^2)$.

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

Mémoïsation

Idée générale: mémoïser et réutiliser les solutions de sous-problèmes qui aident à résoudre le problème.

Complexité temporelle = nombre de sous-problèmes
x
complexité par sous-problème

NB. Dans le calcule de la complexité par sous-problème, on ne compte pas les appels récursifs.

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

B. Tabulation

Le programme tabulé pour un problème donné crée une table de manière **ascendante** et renvoie la dernière entrée de la table.

Par exemple, pour le même nombre de Fibonacci, nous calculons d'abord Fib(0) puis Fib(1) puis Fib(2) puis Fib(3) et ainsi de suite.

Nous construisons donc littéralement les solutions de sous-problèmes ascendantes.

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

B. Tabulation

Voici la version tabulée de la suite de Fibonacci

```
fonction fiboTab (n: entier) : entier
debut
        memo[0] = 0
        memo[1] = 1
        Pour i de 2 a n faire
                memo[i] = memo[i-1] + memo[i-2]
        fin pour
        return memo[n];
finFn
```

Nous remarquons que la **complexité algorithmique** de cette fonction est **linéaire ➔ O(n)**

III. Principe de la programmation dynamique

1/ Chevauchement de sous-problème

C. Tabulation Vis Mémoïsation

Si le problème initial nécessite la résolution de tous les sous-problèmes, la **tabulation** est généralement plus performante que la **mémoïsation** par un facteur constant. En effet, la tabulation ne nécessite pas de temps supplémentaire pour la récursivité et peut utiliser un tableau préalloué plutôt qu'un dictionnaire.

Si seulement certains des sous-problèmes doivent être résolus pour que le problème initial soit résolu, la **mémoïsation** est préférable car les sous-problèmes sont résolus paresseusement, c'est-à-dire que les calculs nécessaires sont effectués.

17

III. Principe de la programmation dynamique

2/ Sous-structure optimale

Un problème donné a une propriété de sous-structure optimale si une solution optimale du problème donné peut être obtenue par des solutions optimales de ses sous-problèmes.

Par exemple, le problème du plus court chemin a la propriété de sous-structure optimale suivante :

Si un nœud x se trouve dans le plus court chemin d'un nœud source u à un nœud de destination v,

➔Le plus court chemin de u à v est la combinaison du plus court chemin de u à x et du plus court chemin de x à v.

NB. Le problème du chemin le plus long n'a pas la propriété de sous-structure optimale.

18

III. Principe de la programmation dynamique

2/ Sous-structure optimale

Exemples typiques de la programmation dynamique.

-L'algorithme **Floyd-Warshall** est un algorithme pour déterminer les distances des plus courts chemins entre toutes les paires de sommets dans un graphe orienté.

-L'algorithme **Bellman-Ford** est un algorithme qui calcule des plus courts chemins depuis un sommet source donné dans un graphe orienté pondéré.

III. Principe de la programmation dynamique

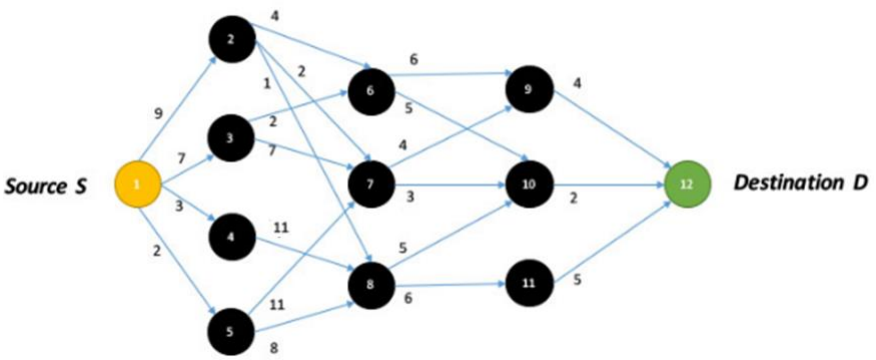
3/ Comment résoudre un problème de programmation dynamique

Le développement d'un algorithme de programmation dynamique se devise en quatre étapes :

1/ Identifiez s'il s'agit d'un problème de programmation dynamique
2/ Identifiez les variables du problème
3/ Exprimer clairement la relation de récurrence
4/ Faire la tabulation (ou ajouter une mémoisation)

III. Principe de la programmation dynamique

3/ Comment résoudre un problème de programmation dynamique **Exemple : Graph de décision en plusieurs étapes**



III. Principe de la programmation dynamique

3/ Comment résoudre un problème de programmation dynamique **Exemple : Graph de décision en plusieurs étapes**

1- Identifiez s'il s'agit d'un problème de programmation dynamique

En règle générale, la programmation dynamique permet de résoudre tous les problèmes nécessitant de maximiser ou de minimiser certaines quantités ou les problèmes de dénombrement nécessitant de compter les arrangements dans certaines conditions ou avec certains problèmes de probabilité.

Tous les problèmes de programmation dynamique satisfont la propriété de sous-problèmes qui **se chevauchent** et la plupart des problèmes dynamiques classiques satisfont également à la propriété de sous-structure optimale. Une fois que nous observons ces propriétés dans un problème donné, assurez-vous qu'il peut être résolu en utilisant la programmation dynamique.

III. Principe de la programmation dynamique

Dans notre exemple, le problème consiste à calculer le plus cours chemin entre la source **S** et la destination **D**. Ce problème a les deux propriétés : Chevauchement de sous-problèmes et Sous-structure optimale donc c'est un problème de programmation dynamique.

Chevauchement de sous-problèmes : deux appels au plus court chemin à partir du noeud 9, une à partir du noeud 6 et l'autre du noeud 7

Sous-structure optimale : le nœud 7 se trouve dans le plus court chemin du nœud 1 à un nœud de destination 12,
le plus court chemin de 1 à 12 est la combinaison du plus court chemin de 1 à 7 et du plus court chemin de 7 à 12

III. Principe de la programmation dynamique

3/ Comment résoudre un problème de programmation dynamique **Exemple : Graph de décision en plusieurs étapes**

2- Identifiez les variables du problème

Nous avons maintenant établi qu'il existe une structure récursive entre les sous-problèmes. Ensuite, nous devons exprimer le problème en terme de paramètres de fonction et voir lesquels de ces paramètres changent.
Une façon de déterminer le nombre de paramètres changeants est d'énumérer des exemples de plusieurs sous-problèmes et de comparer les paramètres. Compter le nombre de paramètres changeants est précieux pour déterminer le nombre de sous-problèmes que nous devons résoudre. Il est également important pour nous aider à renforcer la compréhension de la relation de récurrence de l'étape 1.

III. Principe de la programmation dynamique

3/ Comment résoudre un problème de programmation dynamique Exemple : Graph de décision en plusieurs étapes

2- Identifiez les variables du problème

Dans notre exemple, les deux paramètres susceptibles de changer pour chaque sous-problème sont :

- On veux calculer le cout minimum du nœud x au nœud D

→ Cout minimum entre deux noeuds?

- On veux déterminer le prochaine nœud dans le chemin le plus court entre le nœud x et le nœud D

→ Prochaine noeud dans le chemin le plus court?

25

III. Principe de la programmation dynamique

3/ Comment résoudre un problème de programmation dynamique Exemple : Graph de décision en plusieurs étapes

3- Exprimer clairement la relation de récurrence

C'est une étape important. Exprimer la relation de récurrence aussi clairement que possible renforcera la compréhension de votre problème et facilitera considérablement le reste.

Une fois que vous avez déterminé que la relation de récurrence existe et que vous avez spécifié les problèmes en terme de paramètres, cela devrait être une étape naturelle. Comment les problèmes se rapportent-ils ? En d'autres termes, supposons que vous avez calculé les sous-problèmes. Comment calculeriez-vous le problème principal ?

26

III. Principe de la programmation dynamique

3/ Comment résoudre un problème de programmation dynamique Exemple : Graph de décision en plusieurs étapes

3- Exprimer clairement la relation de récurrence

Voici comment nous y pensons dans notre exemple :

Pour calculer le plus court chemin entre le nœud **6** et le nœud **12 (D)** il faut récupérer le minimum du cout entre les nœuds voisins de **6** dans le chemin vers le nœud **12 (deux choix passant par le nœud 9 ou par le nœud 10)**

→ On peut représenter cette relation comme suit :

Cout(6)=Min{poids(6-9) + Cout(9), poids(6-10) + Cout(10)}

en générale :

Cout(i)=Min $_{\text{v=nœuds voisins}}$ **{ poids(i, v) + cout(v) }**.

27

III. Principe de la programmation dynamique

3/ Comment résoudre un problème de programmation dynamique Exemple : Graph de décision en plusieurs étapes

4- Faire la tabulation (de bas en haut)

C'est l'étape la plus simple, consiste à commencer le calcule à partir du destination **(D)** et remonté dans le graph jusqu'à le nœud source **(S)**.

28

III. Principe de la programmation dynamique

Etape 1 : trivial le plus court chemin entre 12 et 12 c'est zéro

Nœud	1	2	3	4	5	6	7	8	9	10	11	12
Cout	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	0
Nœud suivant	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	12

Etape 2 : calculer le plus court chemin entre la destination et les noeud menant directement au nœud 12

Nœud	1	2	3	4	5	6	7	8	9	10	11	12
Cout	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	4	2	5	0
Nœud suivant	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	12	12	12	12

29

III. Principe de la programmation dynamique

Etape 3 :

Nœud	1	2	3	4	5	6	7	8	9	10	11	12
Cout	nulle	nulle	nulle	nulle	nulle	7	5	7	4	2	5	0
Nœud suivant	nulle	nulle	nulle	nulle	nulle	10	10	10	12	12	12	12

Cout(8)=min{ poids(8-11)+cout(11), poids(8-10)+cout(10)}
=min{6+5, 5+2}=7 (nœud suivant est 10)

Cout(7)=min{ poids(7-9)+cout(9), poids(7-10)+cout(10)}
=min{4+4, 3+2}=5 (nœud suivant est 10)

Cout(6)=min{ poids(6-9)+cout(9), poids(6-10)+cout(10)}
=min{6+4, 5+2}=7 (nœud suivant est 10)

30

III. Principe de la programmation dynamique

Etape 4 :

Nœud	1	2	3	4	5	6	7	8	9	10	11	12
Cout	nulle	7	9	18	15	7	5	7	4	2	5	0
Nœud suivant	nulle	7	6	8	8	10	10	10	12	12	12	12

Cout(5)=min{ poids(5-7)+cout(7), poids(5-8)+cout(8)}
=min{11+5, 8+7}=15 (nœud suivant est 8)

Cout(4)=min{ poids(4-8)+cout(8)}
=min{11+7}=18 (nœud suivant est 8)

Cout(3)=min{ poids(3-6)+cout(6), poids(3-7)+cout(7)}
=min{2+7, 7+5}=9 (nœud suivant est 6)

Cout(2)=min{ poids(2-6)+cout(6), poids(2-7)+cout(7) , poids(2-8)+cout(8)}
=min{4+7, 2+5, 1+8}=7 (nœud suivant est 7)

31

III. Principe de la programmation dynamique

Etape 5:

Nœud	1	2	3	4	5	6	7	8	9	10	11	12
Cout	16	7	9	18	15	7	5	7	4	2	5	0
Nœud suivant	{2,3}	7	6	8	8	10	10	10	12	12	12	12

Cout(1)=min{ poids(1-2)+cout(2), poids(1-3)+cout(3) , poids(1-4)+cout(4) , poids(1-5)+cout(5)}
=min{9+7, 7+9, 3+18, 2+15}=16(nœud suivant est 2 ou 3)

Donc le cout du plus court chemin allant du nœud **1 (S)** au noeud **12(D)** est **16** en partant soit de **2 ou de 3:**
1->2->7->10->12 ou bien 1->3->6->10->12
La complexité de cet algorithme est **O(n^2)** dans un graph complet ou **n** est le nombre de nœuds
Comme vous voyez au cours de résolution du problème on a 5 étapes et dans chaque étape il faut prendre une décision pour le chemin optimal à suivre d'ou le nom de **graphe en plusieurs étapes** et **c'est l'idée générale de la programmation dynamique,**
Implémentation ==> https://sourceforge.net/projects/algocompmr1/files/ **(GraphDesision.py)**

32