Programmation dynamique

Francois Schwarzentruber

ENS Rennes, France

Exemple de problème

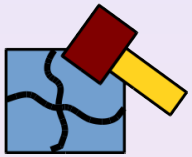
Calculer une plus longue sous-suite croissante

5 **2** 8 6 **3** **6** **9** 7

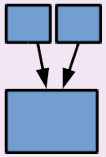
Revoir l'objectif à la baisse...

Calculer la longueur d'une plus longue sous-suite croissante.

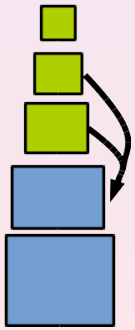
Principe de la programmation dynamique



Définir une collection de sous-problèmes



Trouver une relation de récurrence



Écrire un algorithme qui résout les sous-problèmes

des plus petits au plus gros

À ne pas suivre

Oublier de stocker les valeurs

Ecrire un algorithme récursif

function calculerLongueur(n)
  return $1 + \max_{j \in \{1,\ldots,n-1\} | x_j \leq x_n}$ *calculerLongueur*(j)
endFunction

Complexité exponentielle en n

Bonne conception d'algorithme

function résoudreProblème(un grand problème)
    Soit T un tableau pour stocker les résultats7
    On parcourt les sous-problèmes P par ordre croissant
        On résout P avec la relation de récurrence
        On utilise les valeurs stockés dans T
        On stocke le résultat pour P dans T
    return la solution (en utilisant les valeurs stockées dans T)
endFunction

Outline

Construire une solution

la longueur d'une plus longue sous-suite croissante

ce que l'on a

ce que l'on veut

$$L[i] = 1 + \max_{j \in \{1,\dots,n-1\} | x_j \leq x_n} L[j]$$

$\uparrow$

stocker dans $\pi[i]$ un bon j s'il existe

5	**2**	8	6	**3**	**6**	**9**	7
	$\pi^3(i_0)$			$\pi^2(i_0)$	$\pi(i_0)$	i_0	

et $\pi^4(i_0) = \bullet$

Outline

Outline

Texte

- Si l'entrée est $x_1 x_2 \dots x_n$, les sous-problèmes peuvent être :

$$x_1 x_2 \dots x_i \; x_{i+1} \dots x_n$$

- Si l'entrée est $x_1 x_2 \dots x_n$, les sous-problèmes peuvent être :

$$x_1 \dots x_i \; x_{i+1} x_2 \dots x_j \; x_{j+1} \dots x_n$$

- Si l'entrée est $x_1 x_2 \dots x_n$ et $y_1 y_2 \dots \quad \dots y_m$, les sous-problèmes peuvent être :

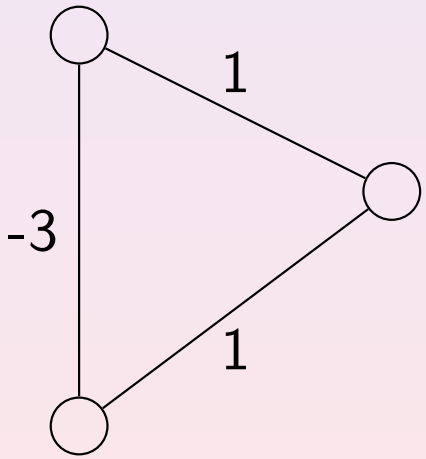
$$\begin{array}{l} x_1 x_2 \dots \quad\quad x_i \; x_{i+1} \dots x_n \\ y_1 y_2 \dots y_j \; y_{i+1} \dots \quad\quad \dots y_m \end{array}$$

$\vdots$

Outline

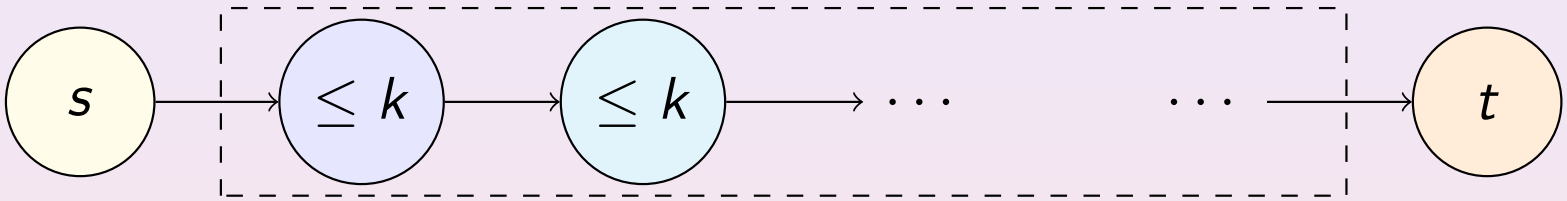
Plus court chemin dans un graphe

Cycle négatif !

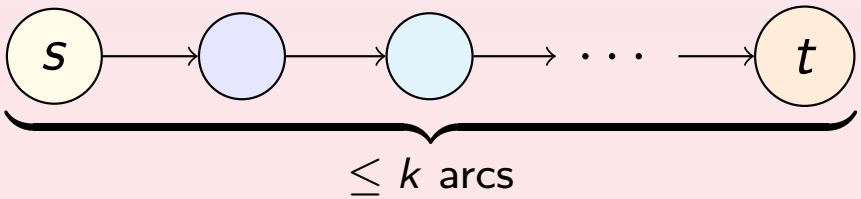


Deux types de décomposition en sous-problèmes

- Calculer la longueur d'un plus court chemin de s à t où les numéros des sommets intermédiaires sont $\leq k$

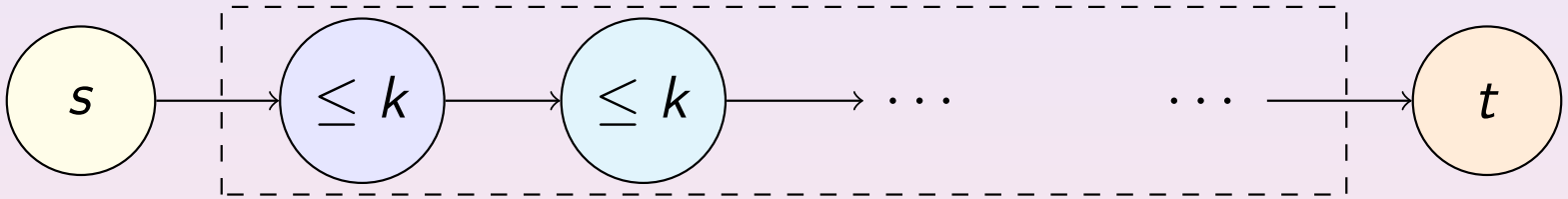


- Calculer la longueur d'un plus court chemin de s à t avec au plus k arcs



Vers l'algorithme de Floyd-Warshall

Calculer la longueur d'un plus court chemin de s à t
où les numéros des sommets intermédiaires
sont $\leq k$



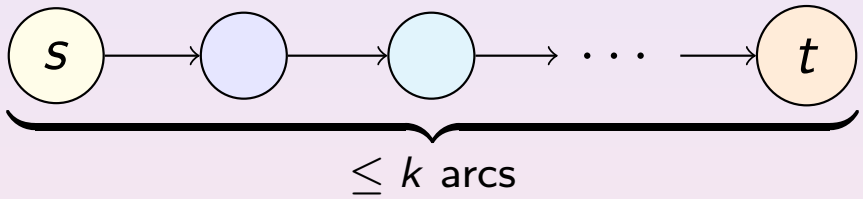
Problème

- Entrée : G graphe orienté pondéré sans cycle négatif
- Sortie : les longueurs de tous les plus court chemins
  (+ détection de cycles négatifs)

$\rightarrow$ s'adapte pour calculer expressions rationnelles correspondant à un automate

Vers l'algorithme de Bellman-Ford

Calculer la longueur d'un plus court chemin de s à t
avec au plus k arcs



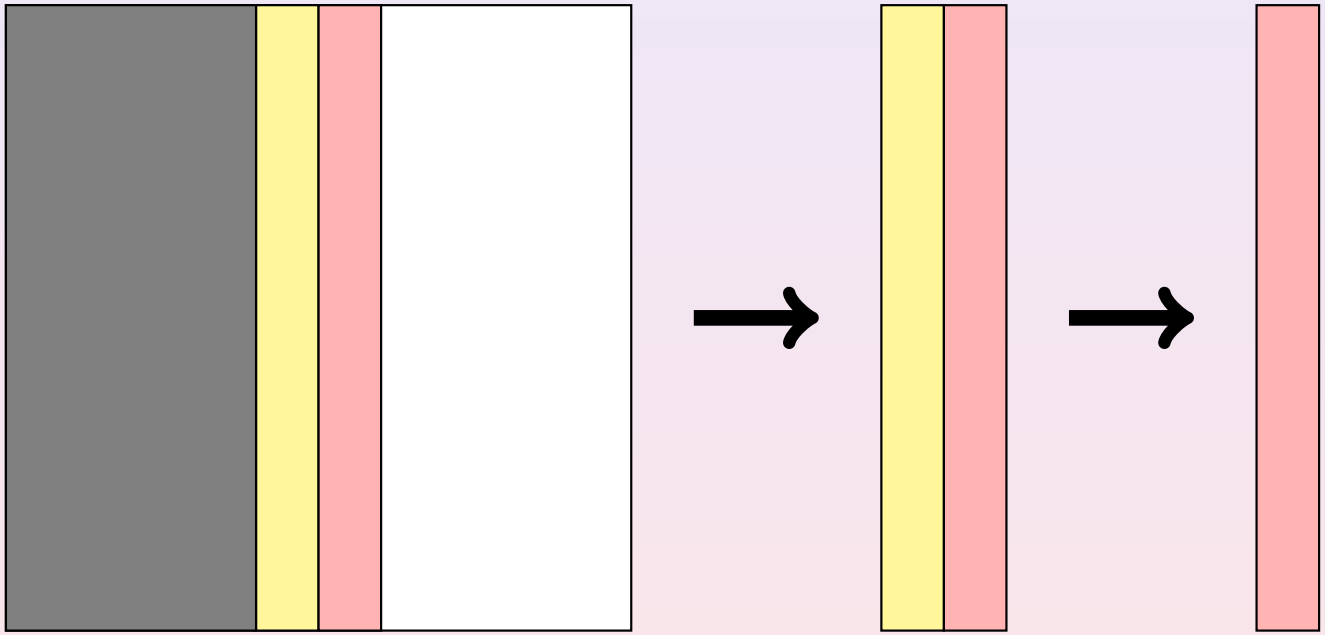
Problème

- Entrée : G graphe orienté pondéré sans cycle négatif, un sommet source s
- Sortie : les longueurs de tous les plus court chemins depuis s
(+ détection de cycles négatifs)

Outline

Outline

Exemple : Bellman-Ford



Outline

Plus longue sous-suite commune [Papadimitriou et al., Algorithms]

- Entrée : deux chaînes de caractères $x[1 \dots m]$ et $y[1, \dots, n]$
- Sortie : la plus longue sous-suite communes à x et y

x : A **B** **C** B D **A** **B**
y : **B** D **C** **A** **B** C

Outline

Outline

Repérer les points intéressants dans la table (*)

	“	A	L	G	O	R	I	T	H	M	S	”
“	0	0	0	0	0	0	0	0	0	0	0	0
A	0	**1**	1	1	1	1	1	1	1	1	1	1
L	0	1	**2**	2	2	2	2	2	2	2	2	2
T	0	1	2	2	2	2	2	**3**	3	3	3	3
R	0	1	2	2	2	**3**	3	3	3	3	3	3
U	0	1	2	2	2	3	3	3	3	3	3	3
I	0	1	2	2	2	3	**4**	4	4	4	4	4
S	0	1	2	2	2	3	4	4	4	4	**5**	5
T	0	1	2	2	2	3	4	**5**	5	5	5	5
I	0	1	2	2	2	3	**4**	5	5	5	5	5
C	0	1	2	2	2	3	4	5	5	5	5	5
”	0	1	2	2	2	3	4	5	5	5	5	**6**

Outline

Exemple : sac à dos [Papadimitriou et al., Algorithms]

Version avec répétition

Mémoïzation

Soit T un tableau / table de hachage pour stocker les résultats

function résoudreProblème(P)
    if P est déjà résolu **then**
        return la valeur correspondante stockée dans T
    else
        On résout P avec la relation de récurrence
        On appelle récursivement résoudreProblème
        On stocke le résultat pour P dans T
        return le résultat pour P
endFunction

Exemple : sac à dos [Papadimitriou et al., Algorithms]

Version sans répétition

Sac à dos sans répétition : approximation [Papa., p. 284]

1.99€

38.4€

...

238.12€

Sac à dos sans répétition : approximation [Papa., p. 284]

Outline

Bilan

- Bien spécifier en français les sous-problèmes sans cela... pas de relation de récurrence possible
- Pas d'appel récursif (sauf si mémoïzation)
- Stocker les résultats intermédiaires dans un tableau ou... une table de hachage
- Bien comprendre l'ordre partiel donné par la relation de récurrence
- Choisir une bonne extension linéaire qui donne l'ordre effectif des sous-problèmes

Questions

- On peut adapter l'algorithme de Floyd-Warshall pour calculer une expression rationnelle d'un automate. Peut-on adapter Bellman-Ford ?

Thank you for your attention!