906. PROGRAMMATION DYNAMIQUE : EXEMPLES ET APPLICATIONS

I/ Programmation dynamique : application à un exemple et principes généraux

1) Les limites de diviser pour régner

ex 1.1 Calcul des coefficients binomiaux : méthode naïve

On utilise un algorithme diviser pour régner basé sur la constatation du fait suivant : $\forall k, n \in \mathbb{N}^*, \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

```
1 bin(k,n) =
2 |  si k = 0 renvoyer 1
3 |  si k > n renvoyer 0
4 |  sinon, renvoyer bin(k-1, n-1) + bin(k, n-1)
```

Mais cet algorithme a une complexité qui peut être exponentielle en n. La raison en est que certains calculs sont effectués plusieurs fois comme explicité dans l'annexe A

ex 1.2 Amélioration de la complexité temporelle de ex 1.1

On remplit le triangle de Pascal jusqu'à la case (n, k)

Cette fois la complexité temporelle est en $\Theta(nk)$

rq 1.3 La complexité spatiale de cet algorithme est aussi en $\Theta(nk)$ puisqu'on remplit un demi tableau de taille $n \times k$. Mais comme le calcul de la ligne $n+1$ dans ce tableau ne dépend que des résultats de la ligne n, on peut travailler sur 2 lignes de longueur k ce qui fait passer la complexité en mémoire d'un $\Theta(nk)$ à un $\Theta(k)$

2) Idées générales de la programmation dynamique

[C] p351

rappel 1-4 Le cadre dans lequel on utilise la méthode "diviser pour régner" est le suivant : pour résoudre un problème P_n, on résout un certain nombre de problèmes $P_{i_1}, \dots P_{i_k}$ tels que leur entrée i_j est plus petite que n puis on combine les solutions obtenues pour obtenir la solution au problème P_n.

def 1-5 Le cadre de la programmation dynamique est le même mais suppose une hypothèse sur les sous problèmes $P_{i_1}, \dots, P_{i_k}$: ils se chevauchent car ont des sous problèmes communs.

Dès lors, si on conserve une approche DPR, les solutions de ces sous problèmes communs sont calculées plusieurs fois. L'idée de la programmation dynamique est de calculer les solutions des sous problèmes "dans le bon ordre" de façon à ne le faire qu'une fois. Ainsi :

```
Fonction Résoudre le problème P =
|  Résoudre les sous problèmes triviaux de P
|  Parcourir les sous problèmes de P dans l'ordre croissant
|  |  Résoudre le sous problème grâce aux solutions des problèmes résolus
|  Renvoyer la solution de P
```

est le schéma type de la programmation dynamique.

rq 1-6 Dans le schéma précédent, on résout les problèmes de façon ascendante. On peut garder une approche descendante grâce à la mémoïsation :

def 1-7 Le schéma de la mémoïsation est le suivant :

```
Fonction Résoudre le problème P =
|  Si la solution du problème P est déjà calculée
|  |  Renvoyer la solution du problème P
|  Sinon
|  |  Calculer la solution Sp de P
|  |  Stocker Sp
|  |  Renvoyer Sp
```

Rq 1.8 • Un avantage de la mémoïsation est qu'on ne calcule que les sous problèmes nécessaires, un inconvénient est les accès mémoire fréquents
• La complexité temporelle pour l'approche ascendante ou descendante est asymptotiquement la même.

II/ Exemples en algorithmique des graphes

[D] p156

ex 2.1 Plus court chemin dans un graphe $G = (S, A)$ orienté et acyclique pondéré par un poids p à partir d'une origine $s \in S$

Il se base sur le fait que $s \rightsquigarrow u \to v$ est de poids minimal $\Rightarrow$ $s \rightsquigarrow u$ est de poids minimal.

D'où
```
PCC (G, s) =
    Initialiser d(s) = 0 et ∀ v ≠ s, d(v) = +∞
    Pour tout v ∈ S \ {s} pris dans l'ordre topologique
        d(v) ← min_{(u,v)∈A} ( d(u) + p(u,v) )
```

Complexité : $\Theta(|S| + |A|)$

[C] p603

ex 2.2 Algorithme de Bellman-Ford (DEV 1)

Entrée = graphe $G = (S, A)$ orienté et pondéré par $p : A \to \mathbb{Z}$, $s \in S$

Sortie = "il y a un cycle de poids < 0 accessible depuis s" si tel est le cas dans G
un tableau contenant les tailles des plus courts chemins de s à v pour tout $v \in S$ sinon

Sous problèmes : $\{ d(t,k),\ t \in S,\ k \in [\![0, |S|-1]\!] \}$ où
$d(t,k)$ = distance de s à t où le chemin passe par moins de k arcs

```
BF (G, s) =
    Initialiser d(s,0) = 0, d(t,0) = +∞ ∀ t ≠ s
    Pour k = 1 à |S| - 1
        Pour tout t ∈ S
            Pour tout (u,t) ∈ A
                d(t,k) ← min ( d(t,k-1), d(u,k-1) + p(u,t) )
    Pour tout (u,v) ∈ A
        Si d(v, |S|-1) > d(u, |S|-1) + p(u,v)
            Renvoyer "il y a un cycle de poids négatif atteint accessible depuis s"
    Renvoyer (d(v, |S|-1))_{v∈S}
```

Complexité : temporelle en $\Theta(|S|^2|A|)$, spatiale en $\Theta(|S|^2)$

Amélioration possible : complexité temporelle en $\Theta(|S||A|)$ et spatiale en $\Theta(|S|)$

[D] p172

ex 2.3 Algorithme de Floyd-Warshall

Entrée : $G = (S, A)$ un graphe pondéré par $p : A \to \mathbb{Z}$ sans cycle de poids < 0

Sortie : la distance minimale entre u et v pour tout $(u,v) \in S^2$

Sous problèmes : On numérote de 1 à n les sommets

$\forall k \in [\![0, n]\!],\ \forall i, j \in S$, $D_k(i,j)$ = distance du plus court chemin entre i et j passant par des sommets uniquement numérotés par 1 élément de $[\![1, k]\!]$

```
FW(G) =
    Initialiser ∀ i ∈ S, D_0(i,i) = 0 et ∀ i ≠ j ∈ S, D_0(i,j) = p(i,j) si (i,j) ∈ A
                                                              = +∞ sinon.
    Pour k = 1 à n
        Pour i = 1 à n
            Pour j = 1 à n
                D_k(i,j) = min ( D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j) )
    Renvoyer (D_n(i,j))_{i,j∈S²}
```

Complexité : temporelle en $\Theta(|S|^3)$, spatiale en $\Theta(|S|^3)$

Rq : pour des graphes denses, il vaut donc mieux utiliser Floyd-Warshall que $|S|$ fois Bellman-Ford

Amélioration possible : complexité spatiale en $\Theta(|S|^2)$

Application : Calcul de la fermeture transitive d'un graphe

[D] p175

ex 2.4 Ensemble indépendants dans un arbre

Entrée : $G = (S, A)$ un arbre de racine r

Sortie : la taille maximale d'un ensemble indépendant $I \subset V$

Sous problèmes : $\forall s \in S$ $I(s)$ = taille maximale d'un ensemble indépendant dans l'arbre enraciné en s

Algo : se déduit des relations $I(\ell) = 1$ pour toute feuille ℓ

$$I(s) = \max \left\{ 1 + \sum_{t \text{ petit fils de } s} I(t),\ \sum_{t \text{ fils de } s} I(t) \right\}$$

Complexité : temporelle en $\Theta(|S|)$, spatiale aussi mais on peut passer à $\Theta(1)$ (axiété max)

Rq : Dans un graphe quelconque, trouver un ensemble indépendant maximal est un problème NP-complet

III / Exemples en algorithmique du texte

[P] p 174

ex 3.1 Distance d'édition

On se donne deux mots x et y, et on veut déterminer une distance entre ces deux mots, c'est à dire un nombre minimum d'ajouts, suppressions et substitutions de caractères pour passer de l'un à l'autre.

Entrée : deux mots $x, y \in \Sigma^*$

Sortie : la distance entre x et y, ou éventuellement un tableau de taille $(|x|+1)\times(|y|+1)$ contenant les distances entre tous les préfixes de x et ceux de y.

Sous-problèmes : $\{E(\overline{x_k}, \overline{y_\ell}),\ k \in [\![0, |x|]\!],\ \ell \in [\![0, |y|]\!]\}$ où
$E(\overline{x_k}, \overline{y_\ell})$ contient la distance entre $\overline{x_k}$ et $\overline{y_\ell}$;
$\overline{x_k}$ le préfixe $x_1 \dots x_k$ de x (vide si $k=0$) ;
$\overline{y_\ell}$ le préfixe $y_1 \dots y_\ell$ de y (vide si $\ell=0$).

Algo : dist(x,y) =

```
1  n ← taille(x)
2  m ← taille(y)
3  E ← tableau(n+1, m+1)
4  Pour i=0 à n
5  |  E(i,0) ← i
6  Pour j=0 à m
7  |  E(j,0) ← j
8  Pour i=1 à n
9  |  Pour j=1 à m
10 |  |  Si x[i] = y[j]
11 |  |  |  E(i,j) ← min(E(i-1,j)+1, E(i,j-1)+1, E(i-1,j-1))
12 |  |  Sinon
13 |  |  |  E(i,j) ← min(E(i-1,j)+1, E(i,j-1)+1, E(i-1,j-1)+1)
14 |  Renvoyer E(n,m)
```

Complexité : temporelle et spatiale en $\Theta(|x||y|)$.

Possible amélioration en espace en ne gardant que une ligne en mémoire (en prenant le plus petit des deux mots pour les colonnes) : Complexité spatiale en $\Theta(\min(|x|,|y|))$

ex 3.2 Plus longue sous-séquence commune (PLSC) DVT 2

[C] p 341 (ed 2)

Entrée : x et y deux mots de Σ^*

Sortie : $w = x_{i_1} \dots x_{i_n} = y_{j_1} \dots y_{j_n}$ une plus longue sous-séquence commune de x et y.

Sous-problèmes : $\{PLSC(\overline{x_k}, \overline{y_\ell}),\ k \in [\![1, |x|]\!],\ \ell \in [\![1, |y|]\!]\}$

avec $\overline{x_k} = x_1 \dots x_k$ préfixe de x
$\overline{y_\ell} = y_1 \dots y_\ell$ préfixe de y

NB : ici, l'algorithme retourne la longueur d'une PLSC et un tableau permettant de la retrouver et de l'écrire en temps $\Theta(|x|+|y|)$

Algo : PLSC(x,y) =

```
1  m ← taille(x)
2  n ← taille(y)
3  C ← tableau(m+1, n+1)   initialisé à 0
4  B ← tableau(m+1, n+1)
5  Pour i=1 à m
6  |  Pour j=1 à n
7  |  |  Si x[i]=y[j]
8  |  |  |  C[i,j] ← C[i-1,j-1] + 1
9  |  |  |  b[i,j] ← « ↖ »
10 |  |  Sinon si C[i-1,j] ≥ C[i,j-1]
11 |  |  |  C[i,j] ← C[i-1,j]
12 |  |  |  b[i,j] ← « ↑ »
13 |  |  Sinon
14 |  |  |  C[i,j] ← C[i,j-1]
15 |  |  |  b[i,j] ← « ← »
16 |  Renvoyer C[m,n] et b
```

Complexité en temps et en espace : $\Theta(|x||y|)$

Amélioration possible en espace : on ne calcule que la longueur d'une PLSC, en ne gardant que une ligne de C et sans utiliser b : complexité spatiale en $\Theta(\min(|x|,|y|))$

Ex 3.3 Cock Younger Kasami

Entrée : une grammaire G sous forme de Chomsky et un mot $w = w_1 w_2 \dots w_n$

Sortie : oui ssi $w \in L(G)$

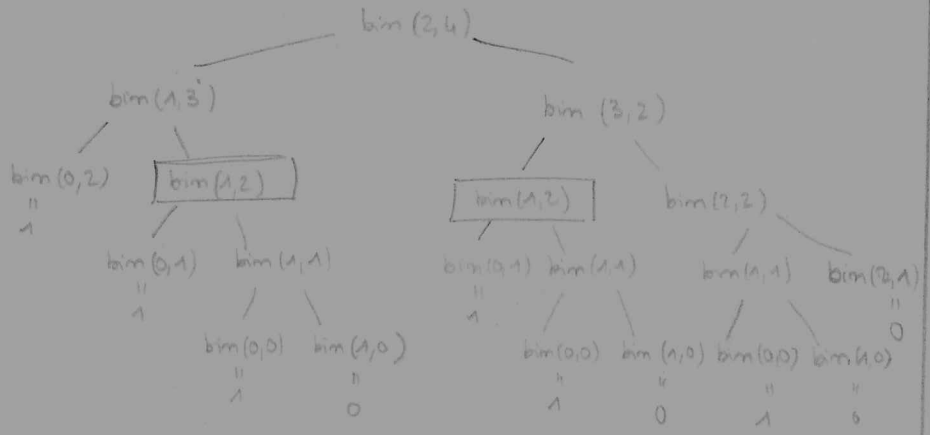
Sous problèmes : $\forall i,j \in [\![1,n]\!]^2,\ E_{i,j} = \{A \text{ non terminaux } | A \to w_i \dots w_j\}$

Complexité temporelle en $\Theta(n^3)$

DEV 3

ANNEXE A

Arbre des appels récursifs dans le calcul de $\binom{4}{2}$



bin(1,2) et tous les appels qui en découlent sont calculés deux fois.

ANNEXE B

		B	A	N	A	N	E
	0	1	2	3	4	5	6
B	1	0	1	2	3	4	5
A	2	1	0	1	2	3	4
R	3	2	1	1	2	3	4
B	4	3	2	2	2	3	4
A	5	4	3	3	2	3	4
R	6	5	4	4	3	3	4
E	7	6	5	5	4	4	3