Informatique

2

Dictionnaires et programmation dynamique

Résumé

Dernière mise à jour	Informatique	Denis DEFAUCHY – Site web
07/02/2023	2 – Dictionnaires et programmation dynamique	Résumé

Principe	
Problème	Un problème d'optimisation est posé au rang n
Force brute	Les algorithmes par force brute qui étudient toutes les possibilités sont souvent de complexité en temps de l'ordre de 2^n ou 3^n
Sous problème simple	Trouver un sous problème plus simple qui permet de passer du rang n-1 au rang n par récurrence
Initialisation	Initialiser le traitement au rang 0
Récurrence (Equation de Bellman)	Itérer par récurrence jusqu'au rang n
Résultat	Obtenir le résultat attendu
Etapes intermédiaires (Rétro programmation)	Remonter la solution pour obtenir les étapes intermédiaires

<table>
<tr><th colspan="2">Exemple : Partition équilibrée d'entiers positifs PEEP</th></tr>
<tr><td>Problème</td><td>On souhaite partitionner un ensemble d'entiers strictement positifs $E = \{e_0, e_1, \dots, e_{n-1}\}$ de somme totale S_E en deux familles F et G avec $E = F \cup G$ et $F \cap G = \emptyset$ telles que la somme des éléments de F soit aussi proche que possible de la somme des éléments de G</td></tr>
<tr><td>Sous problème simple</td><td>Pour $i \in [\![0, n]\!]$ et $j \in [\![0, S]\!]$ avec $S = \left\lfloor \frac{S_E}{2} \right\rfloor$, est-il possible de trouver un sous-ensemble avec au plus i éléments parmi les premiers éléments $E_i = \{e_0, e_1, \dots, e_{i-1}\}$ de E dont la somme est égale à j ?</td></tr>
<tr><td>Initialisation</td><td>$$D(i,j) = \begin{cases} True \ si \ j = 0 \\ False \ si \ i = 0 \ et \ j \neq 0 \end{cases}$$</td></tr>
<tr><td>Récurrence (Equation de Bellman)</td><td>$\forall i \in [1, n], \forall j \in [1, S]$
$D(i,j) = D(i-1,j) \ ou \ \left(j \geq e_{i-1} \ et \ D(i-1, j-e_{i-1})\right)$
Remplissage d'une table visible ci-dessous</td></tr>
<tr><td>Résultat</td><td>La plus grande valeur de j telle que $D(n,j) = True$ indique la plus grande demi-somme atteignable avec tous les éléments de E</td></tr>
<tr><td>Etapes intermédiaires (Rétro programmation)</td><td>Pour $E = \{1,1,2,1\}$, objectif $S = 2$, on peut atteindre 2 par deux chemins. Avec ces deux conditions :
- C1 : $D(i-1,j) = True$: On remonte d'une ligne $(i-1,j)$ et on n'ajoute pas l'élément e_i à F, on l'ajoute donc à G
- C2 : $j \geq e_{i-1} \ et \ D(i-1, j-e_{i-1}) = True$: On va à la case $(i-1, j-e_{i-1})$ une ligne au-dessus et e_{i-1} cases à gauche, et on ajoute e_{i-1} à F
Alors, tant que $i > 0$ ou $j > 0$: je privilégie C1
- Si C1 ($\forall$C2) : Action associée à C1
- Sinon (C2) : action associée à C2
<table>
<tr><th>Chemin en privilégiant C1</th><th>Chemin en privilégiant C2</th></tr>
<tr><td>
<table>
<tr><th>e_i</th><th>i\j</th><th>0</th><th>1</th><th>2</th></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td>2</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>3</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td></td><td>4</td><td>1</td><td>1</td><td>1</td></tr>
</table>
</td><td>
<table>
<tr><th>e_i</th><th>i\j</th><th>0</th><th>1</th><th>2</th></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td>2</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>3</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td></td><td>4</td><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>$F = \{1,1\}$; $G = \{1,2\}$</td><td>$F = \{1,1\}$; $G = \{2,1\}$</td></tr>
</table>
</td></tr>
</table>

Méthodes de programmation

On remplit généralement une table à deux dimensions qui représente l'intégralité des sous problèmes traités lors de la résolution et un utilise généralement un dictionnaire pour mémoriser ces cas. Les clés sont des couples (i,j) et les valeurs sont celles que renvoie l'algorithme

Méthode itérative « de bas en haut »

La programmation itérative est très simple à mettre en œuvre

```python
def PEEP_it(E):
    D = {}
    S = sum(E)
    ni = len(E)
    nj = int(S/2)
    for i in range(ni+1):
        D[(i,0)] = True
    for j in range(1,nj+1):
        D[(0,j)] = False
    for i in range(1,ni+1):
        for j in range(1,nj+1):
            c1 = D[(i-1,j)]
            e = E[i-1]
            c2 = (j >= e) and D[(i-1,j-e)]
            D[(i,j)] = c1 or c2
    return D
```

Méthode récursive « de haut en bas »

La méthode récursive doit impérativement être mémoïsée afin que la complexité soit la même qu'avec la méthode itérative.
On passe souvent par une première fonction simple qui renvoie une valeur du tableau, mais de complexité importante, pour ensuite l'intégrer à une version mémoïsée

```python
def PEEP_rec_ij(i,j,E):
    if j==0:
        return True
    elif i==0:
        return False
    else:
        c1 = PEEP_rec_ij(i-1,j,E)
        e = E[i-1]
        c2 = (j >= e) and PEEP_rec_ij(i-1,j-e,E)
        return (c1 or c2)
```

```python
def PEEP_rec_mem(E):
    def rec(i,j,E):
        if (i,j) in dico:
            return dico[(i,j)]
        else:
            if j==0:
                res = True
            elif i==0:
                res = False
            else:
                c1 = rec(i-1,j,E)
                e = E[i-1]
                c2 = (j >= e) and rec(i-1,j-e,E)
                res = (c1 or c2)
            dico[(i,j)] = res
            return res
    dico = {}
    S,ni,nj = sum(E),len(E),len(E)//2
    for i in range(ni+1):
        for j in range(nj+1):
            rec(i,j,E)
    return dico
```

Attention à bien créer la variable intermédiaire `res` puis à l'utiliser (ne pas faire plusieurs appels de `rec`)
De même, c'est un détail qui coûte le prix du hachage, renvoyer `res` plutôt que `dico[(i,j)]`
Parfois, la version descendante ne calcule pas toutes les cases et peut être légèrement plus intéressante

Dans les deux cas, avec la liste $\{1,1,2,1\}$, on obtient une table de dimensions $(n+1)(S+1)$:

e_i	i\j	0	1	2
1	0	1	0	0
1	1	1	1	0
2	2	1	1	1
1	3	1	1	1
	4	1	1	1

Remarque : A chaque étape k, il est possible de stocker l'étape k-1 y ayant conduit dans le dictionnaire, pour réaliser ensuite relativement simplement la remontée de la solution
ex : TD2-3 – Le compte est bon

<table>
<tr><th colspan="2">Exemples des TD au programme</th></tr>
<tr><td>2-4
Partition équilibrée d'un tableau d'entiers positifs
PEEP
$O(nS)$</td><td>L'ensemble E est de dimension n. On souhaite partitionner un ensemble d'entiers strictement positifs $E = \{e_0, e_1, \dots, e_{n-1}\}$ de somme totale S_E en deux familles F et G avec $E = F \cup G$ et $F \cap G = \emptyset$ telles que la somme des éléments de F soit aussi proche que possible de la somme des éléments de G
Pour $i \in [\![0, n]\!]$ et $j \in [\![0, S]\!]$ avec $S = \left\lfloor \frac{S_E}{2} \right\rfloor$, est-il possible de trouver un sous-ensemble avec au plus i éléments parmi les premiers éléments $E_i = \{e_0, e_1, \dots, e_{i-1}\}$ de E dont la somme est égale à j ?
$$\forall i \in [0,n], \forall j \in [0,S], D(i,j) = \begin{cases} True\ si\ j = 0 \\ False\ si\ i = 0\ et\ j \neq 0 \\ D(i-1,j)\ ou\ \left(j \geq e_{i-1}\ et\ D(i-1, j-e_{i-1})\right) \end{cases}$$
La plus grande valeur de j telle que $D(n,j) = True$ indique la plus grande demi-somme atteignable avec tous les éléments de E</td></tr>
<tr><td>2-5
Ordonnancement de tâches pondérées
OTP
$O(nP_{max})$</td><td>Soit $E = \{e_0, e_1, \dots, e_{n-1}\}$ un ensemble fini de n objets tels que chaque objet possède un poids $p_i > 0$ (contrainte) et une valeur $v_i > 0$ (objectif). On souhaite déterminer le sous ensemble F de E maximisant la somme total des valeurs de ses éléments, pour un poids limité à une valeur maximale P_{max} (dans un sac à dos).
Quelle est la valeur maximale atteinte avec des éléments parmi les i premiers éléments de E pour un poids du sac valant au plus j
$$\forall i \in [0,n], \forall j \in [0,P_{max}], D(i,j) = \begin{cases} 0\ si\ i = 0 \\ max\begin{pmatrix} D(i-1,j) \\ D(i-1,j-p_{i-1}) + v_{i-1} \end{pmatrix}\ si\ j \geq p_{i-1} \\ D(i-1,j)\ sinon \end{cases}$$
La dernière case $D(n, P_{max})$ donne la valeur maximale atteinte en piochant parmi tous les éléments de E pour un poids au plus égal à P_{max}</td></tr>
<tr><td>2-6
Plus longue sous-suite commune
PLSC
$O(nm)$</td><td>Soient $X_n = [x_1, \dots, x_n]$ et $Y_m = [y_1, \dots, y_m]$ deux séquences. Appelons X_i et Y_i les séquences $X\ et\ Y$ contenant leurs i premiers termes. On souhaite trouver la plus longue sous-séquence commune à X_n et Y_m (ex : agbcjkf & abdef → abf).
Quelle est la longueur de la PLSC entre X_i et Y_i, séquences $X\ et\ Y$ contenant leurs i premiers termes
$$\forall i \in [0,n], \forall j \in [0,m], D(i,j) = \begin{cases} 0\ si\ i = 0\ ou\ j = 0 \\ D(i-1,j-1) + 1\ si\ x_i = y_j \\ max\begin{pmatrix} D(i-1,j) \\ D(i,j-1) \end{pmatrix}\ si\ x_i \neq y_j \end{cases}$$
La dernière case $D(n,m)$ donne la longueur de la PLSC de X et Y</td></tr>
<tr><td>2-7
Distance d'édition Levenshtein
$O(N_1 N_2)$</td><td>Soient deux chaines de caractères M_1 et M_2 de tailles respectives N_1 et N_2. On cherche le nombre de modifications (insertion, suppression, remplacement) minimal à réaliser pour passer de l'une à l'autre.
Quelle est la distance de Levenshtein entre $M1_i$ et $M2_j$, contenant respectivement les i premières lettres de M_1 et les j premières lettres de M_2
$$\forall i \in [0,N_1], \forall j \in [0,N_2], D(i,j) = \begin{cases} i\ si\ j = 0 \\ j\ si\ i = 0 \\ sinon \begin{cases} D(i-1,j-1)\ si\ M_1[i-1] = M_2[j-1] \\ 1 + min\begin{cases} D(i-1,j-1) \\ D(i-1,j) \quad sinon \\ D(i,j-1) \end{cases} \end{cases} \end{cases}$$
$D(N_1, N_2)$ contient le nombre minimal de modifications à réaliser de M_1 à M_2</td></tr>
<tr><td>2-8
Distances dans un graphe - Floyd-Warshall
$O(n^3)$</td><td>Soit l'ensemble $E = \{0,1, \dots, n-1\} = \{e_1, e_2, \dots, e_n\}$ des n sommets du graphe représenté par la matrice d'adjacence G et W^k une matrice telle que $W^0 = G$.
Pour $k > 0$, quels sont les poids minimaux W_{ij}^k des chemins, s'ils existent (∞ sinon), de tous les sommets i à tous les sommets j du graphe n'empruntant que des sommets intermédiaires dans $\{e_1, e_2, \dots, e_k\}$
$$\forall (i,j) \in [0,n-1]^2, \forall k \in [0,n], D(i,j,k) = \begin{cases} G_{ij}\ si\ k = 0 \\ min\begin{pmatrix} D(i,j,k-1) \\ D(i,k-1,k-1) + D(k-1,j,k-1) \end{pmatrix} \end{cases}$$
$W_{ij}^n = D(i,j,n)$ est le poids minimal du chemin de i vers j
Cet algorithme se programme avec le même coût en temps sur des matrices ou avec un dictionnaire</td></tr>
<tr><th colspan="2">Spécificités de la programmation dynamique</th></tr>
<tr><td>Propriété de sous-structure optimale</td><td>Si un chemin $E = \{x_0, \dots, x_n\}$ est solution, tout sous-chemin $\{x_k, \dots, x_l\}$ de E est aussi un chemin optimal parmi tous les chemins allant de x_k à x_l</td></tr>
<tr><td>Chevauchement de sous-problèmes</td><td>L'algorithme récursif rencontre souvent les mêmes sous-problèmes et stocke leurs solutions dans un tableau afin de les réutiliser à chaque fois par mémoïsation</td></tr>
</table>