第5章 动态规划

5.3 定期多阶段决策问题

阶段数固定的多阶段决策问题

- 背包问题
- 最长公共子序列问题
- 旅行售货员问题

背包问题

背包问题（**The Knapsack Problem**）

- 实例：有一个背包，总承重为整数 W。
  有 n 个物品，每个物品重量为 w_i，价值为 v_i，$i = 1..n$。w_i和 v_i 均为整数。
- 目标：装入背包若干物品，使其总重量不超过 W，总价值最大。

例子

- $n = 4, W = 5$。

i	1	2	3	4
v_i	6	10	12	13
w_i	1	2	3	4

- 贪心策略 1：每次装当前价值最大的物品。
  找到的解：14 + 6 = 20。
- 贪心策略 2：每次装当前重量最小的物品（以留出尽可能多的空间给将来的物品使用）。
  找到的解：6 + 10 = 16。
- 贪心策略 3：每次装当前“性价比”最高的物品。
  找到的解：6 + 10 = 16。
- 最优解：10 + 12 = 22 !

解（1）

- 定义 $f(i, j)$表示将物品 $1..i$ 中的若干装入总容量为 j 的背包，所获得的最大价值。则原问题是求 $f(n, W)$。
- 若 $j < w_i$，则第 i 个物品必不能装入背包。
- 若 $j \geq w_i$，则第 i 个物品可以装入背包。到底是否装入背包，取决于装入第 i 个物品，再装入获得价值 $f(i-1, j-w_i)$的那些物品，所获得的总价值，以及将 $1..i-1$ 物品中的若干装入容量为 j 的背包所获得的总价值 $f(i-1, j)$哪个更大。
- 则有：

$$f(i,j)=\begin{cases} f(i-1,j), & i\geq 1, j\geq 1, j<w_i \\ \max\{f(i-1,j-w_i)+v_i, f(i-1,j)\}, & i\geq 1, j\geq w_i \\ 0, & i=0\text{或}j=0 \end{cases}$$

。

动态规划表

f	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	6	6	6	6	6
2	0	6	10	16	16	16
3	0	6	10	16	18	22
4	0	6	10	16	18	22

h	0	1	2	3	4	5
0	×	×	×	×	×	×
1	×	(0,0) +6	(0,1) +6	(0,2) +6	(0,3) +6	(0,4) +6
2	×	(1,1)	(1,0) +10	(1,1) +10	(1,2) +10	(1,3) +10
3	×	(2,1)	(2,2)	(2,3)	(2,1) +12	(2,2) +12
4	×	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)

- f 表记录 $f(i, j)$函数的值。
- h 表记录对应的 $f(i, j)$是如何计算出来的。即，$f(i, j) = f(i-1, j)$，还是 $f(i, j) = f(i-1, j-w_i) + v_i$。后一种情况表明装了物品 i。

时间复杂度

- 以上动态规划法的时间复杂度为O(nW)。
- 这个时间复杂度不是多项式的，原因在于W是问题输入中的一个整数。

解（2）

- 定义 $f(i,j)$表示在物品 $1..i$ 中选择若干装入背包，使其总价值恰好为 j，总重量最小，这样的若干物品的总重量。若这样的集合不存在，则 $f(i,j)=\infty$。
- 则有：

$$f(i,j)=\begin{cases}0, & i=0, j=0\\ \infty & i=0, j\ge 1\\ 0, & i\ge 1, j=0\\ f(i-1,j), & i\ge 1, v_i>j\ge 1\\ \min\{f(i-1,j), f(i-1,j-v_i)+w_i\}, & i\ge 1, j\ge v_i\end{cases}$$ 。

- 则原问题是寻找满足 $f(i,j)\le W$ 的最大的 j。

例子

- 背包问题实例：$n = 4$，$W = 5$。

i	1	2	3	4
v_i	6	10	12	13
w_i	1	2	3	4

- 动态规划表：

i \ j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	∞	∞	1	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	∞	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞	∞	1	∞	∞	∞	2	∞	3	∞	∞	∞
4	0	∞	∞	∞	∞	∞	1	∞	∞	∞	2	∞	3	4	∞	∞

i \ j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	3	∞	4	∞	∞	∞	5	∞	∞	∞	∞	∞	6	∞	∞
4	3	∞	4	4	∞	∞	5	6	∞	7	∞	∞	6	7	∞

表格中 j 的上界 $= \min\{\max\{v_i/w_i\} * W,\ \sum v_i\} = 30$。

最长公共子序列（LCS）问题

最长公共子序列（Longest Common Subsequence）问题

- **实例：序列 $X = x_1x_2\cdots x_m$，$Y = y_1y_2\cdots y_n$。**
- **询问：X 和 Y 的一个最长公共子序列。**

- **子序列：若有 l 个数满足 $1 \le s_1 < s_2 < \cdots < s_l \le m$，则称 $x_{s1}x_{s2}\cdots x_{sl}$ 为 X 的一个子序列。**

递推公式

- 从 X 和 Y 的尾部向前，考虑它们的最长公共子序列。假设现在扫描的字符是 x_i 和 y_j。
- 若 $x_i = y_j$，则 $x_1..x_i$ 与 $y_1..y_j$ 的最长公共子序列就是 $x_1..x_{i-1}$ 与 $y_1..y_{j-1}$ 的最长公共子序列，再加上字符 x_i。
- 否则，$x_1..x_i$ 与 $y_1..y_j$ 的最长公共子序列，就是 $x_1..x_{i-1}$ 与 $y_1..y_j$ 的最长公共子序列，以及 $x_1..x_i$ 与 $y_1..y_{j-1}$ 的最长公共子序列中，较长的那一个。
- 定义 $v(i, j)$为 $x_1..x_i$ 与 $y_1..y_j$ 的最长公共子序列的长度。则有：

$$v(i,j)=\begin{cases} v(i-1,j-1)+1, & 若 i>0, j>0, x_i = y_j \\ \max\{v(i-1,j), v(i,j-1)\}, & 若 i>0, j>0, x_i \neq y_j \\ 0, & 若 i=0 或 j=0 \end{cases}。$$

例子

- 原问题就是计算 $v(m, n)$。
- 例子：X = monkey，Y = human。

v	0	1h	2u	3m	4a	5n
0	0	0	0	0	0	0
1m	0	0	0	1	1	1
2o	0	0	0	1	1	1
3n	0	0	0	1	1	2
4k	0	0	0	1	1	2
5e	0	0	0	1	1	2
6y	0	0	0	1	1	2

f	0	1h	2u	3m	4a	5n
0	×	×	×	×	×	×
1m	×	↑	↑	\	←	←
2o	×	↑	↑	↑	↑	↑
3n	×	↑	↑	↑	↑	\
4k	×	↑	↑	↑	↑	↑
5e	×	↑	↑	↑	↑	↑
6y	×	↑	↑	↑	↑	↑

LCS之动态规划算法

Algorithm LCS(X, Y)

计算$X = x_1x_2...x_m$和$Y = y_1y_2...y_n$的最长公共子序列。

```
for i ← 1 到 m do
    c[i, 0] ← 0。
endfor
for j ← 0 到 n do
    c[0, j] ← 0。
endfor
for i ← 1 到 m do
    for j ← 1 到 n do
        if x[i] = y[j] then
            c[i, j] ← c[i – 1, j – 1] + 1。
```

LCS之动态规划算法

```
      f[i, j] ← "\"。
    else
      if c[i − 1, j] ≥ c[i, j − 1] then
        c[i, j] ← c[i − 1, j]。
        f[i, j] ← "↑"。
      else
        c[i, j] ← c[i, j − 1]。
        f[i, j] ← "←"。
      endif
    endif
  endfor
endfor
return c, f。
```

时间复杂度

- 以上动态规划法（LCS问题）的时间复杂度为O(mn)。
- 这是一个多项式的时间复杂度，表明LCS问题是多项式时间可解的。

旅行售货员（TSP）问题

- 旅行售货员问题是图论中一个著名问题。
- 实例：图$G = (V, E)$，每条边(v_i, v_j)上有长度$d(v_i, v_j)$。
- 询问：找一个最短的圈，走过所有的顶点。

- 等价描述：从v_1点出发，经过其余顶点（$v_2, \ldots, v_n$）各一次，最后返回v_1的最短的圈。

递推方程

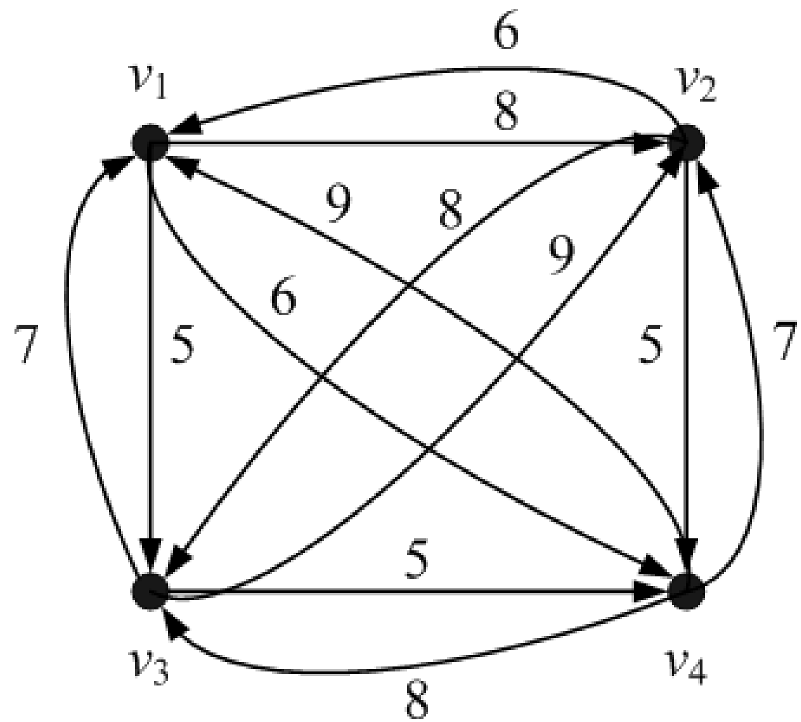
- 如何对问题（按最优化原理）进行分解？
- 由于原问题是找一个圈，从任何一个点开始走这个圈都是可以的。因此不妨假设从 v_1 开始。
- 于是问题就是，从 v_1 开始，经过 $V \setminus \{v_1\}$中的顶点各一次，最后回到 v_1，这样的最短路的长度是多少？
- 用 $f(v_i, U, v_1)$表示从顶点 v_i 出发，经过 U 中的顶点各一次，最后到达 v_1 的最短路的长度，其中 $U \subset V$ 是一个顶点子集，满足 $v_1 \notin U$，$v_i \notin U$。则有

$$f(v_i, U, v_1) = \begin{cases} \min\limits_{v_j \in U}\{d(v_i, v_j) + f(v_j, U \setminus \{v_j\}, v_1)\}, & U \neq \phi \\ d(v_i, v_1), & U = \phi \end{cases}。$$

- 原问题就是计算 $f(v_1, V \setminus \{v_1\}, v_1)$。

例5.3.1

例 5.3.1 解 TSP 问题：



例5.3.1

	v_2	v_3	v_4
$\varnothing$	6	7	9
$\{v_2\}$	×	15	13
$\{v_3\}$	15	×	15
$\{v_4\}$	14	14	×
$\{v_2, v_3\}$	×	×	22
$\{v_2, v_4\}$	×	18	×
$\{v_3, v_4\}$	20	×	×

	v_2	v_3	v_4
$\varnothing$	v_1	v_1	v_1
$\{v_2\}$	×	$v_2, \varnothing$	$v_2, \varnothing$
$\{v_3\}$	$v_3, \varnothing$	×	$v_3, \varnothing$
$\{v_4\}$	$v_4, \varnothing$	$v_4, \varnothing$	×
$\{v_2, v_3\}$	×	×	$v_2, \{v_3\}$
$\{v_2, v_4\}$	×	$v_4, \{v_2\}$	×
$\{v_3, v_4\}$	$v_4, \{v_3\}$	×	×

- 使用表格计算诸 $f_k(v_i, U, v_1)$：$k = 1..2$；$\forall v_i \neq v_1$；$\forall U \subset V$，满足 $v_1 \notin U$，$v_i \notin U$。
- 最后单独计算 $f_3(v_1, U = V \setminus \{v_1\}, v_1)$（因为 $v_1 \in U$，故 $f_3(v_1, U, v_1)$没有列入动态规划表格中）。

计算一个单元格

$f_k(v_i, U, v_1)$

$d \leftarrow \infty$。

for each $v_j \in U$ **do**

  $t \leftarrow d(v_i, v_j) + \mathrm{table}(v_j, U \setminus \{v_j\})$。

  if $t < d$ **then** $d \leftarrow t$。

endfor

return d。

计算过程

- $f_1(v_2, \{v_3\}, v_1) = \min\{d(v_2, v_3) + f_0(v_3, \varnothing, v_1)\} = 8 + 7 = 15$。
- $f_1(v_2, \{v_4\}, v_1) = \min\{d(v_2, v_4) + f_0(v_4, \varnothing, v_1)\} = 5 + 9 = 14$。

- $f_1(v_3, \{v_2\}, v_1) = \min\{d(v_3, v_2) + f_0(v_2, \varnothing, v_1)\} = 9 + 6 = 15$。
- $f_1(v_3, \{v_4\}, v_1) = \min\{d(v_3, v_4) + f_0(v_4, \varnothing, v_1)\} = 5 + 9 = 14$。

- $f_1(v_4, \{v_2\}, v_1) = \min\{d(v_4, v_2) + f_0(v_2, \varnothing, v_1)\} = 7 + 6 = 13$。
- $f_1(v_4, \{v_3\}, v_1) = \min\{d(v_4, v_3) + f_0(v_3, \varnothing, v_1)\} = 8 + 7 = 15$。

- $f_2(v_2, \{v_3, v_4\}, v_1)$
  $= \min\{d(v_2, v_3) + f_1(v_3, \{v_4\}, v_1), d(v_2, v_4) + f_1(v_4, \{v_3\}, v_1)\}$
  $= \min\{8 + 14, 5 + 15\} = 20$。

计算过程

- $f_2(v_3, \{v_2, v_4\}, v_1)$
  $= \min\{d(v_3, v_2) + f_1(v_2, \{v_4\}, v_1), d(v_3, v_4) + f_1(v_4, \{v_2\}, v_1)\}$
  $= \min\{9 + 14, 5 + 13\} = 18$。
- $f_2(v_4, \{v_2, v_3\}, v_1)$
  $= \min\{d(v_4, v_2) + f_1(v_2, \{v_3\}, v_1), d(v_4, v_3) + f_1(v_3, \{v_2\}, v_1)\}$
  $= \min\{7 + 15, 8 + 14\} = 22$。
- 最后一个：$f_3(v_1, \{v_2, v_3, v_4\}, v_1)$
  $= \min\{d(v_1, v_2) + f_2(v_2, \{v_3, v_4\}, v_1),$
  $d(v_1, v_3) + f_2(v_3, \{v_2, v_4\}, v_1),$
  $d(v_1, v_4) + f_2(v_4, \{v_2, v_3\}, v_1)\}$
  $= \min\{8 + 20, 5 + 18, 6 + 22\} = \min\{26, 23, 28\} = 23$。
- 最优 TSP 旅游为：$v_1 \to v_3 \to v_4 \to v_2 \to v_1$。

一般情况

	v_2	v_3	v_4	$\cdots$	$\cdots$	$\cdots$	v_n
$\varnothing$							
$\{v_2\}$							
$\{v_3\}$							
$\vdots$							
$\{v_4\}$							
$\{v_2, v_3\}$							
$\{v_2, v_4\}$							
$\vdots$							
$\{v_3, v_4\}$							
$\vdots$							
$\{v_2, v_3, \cdots, v_{n-1}\}$							
$\{v_2, v_3, \cdots, v_n\}$							
$\vdots$							
$\{v_3, v_4, \cdots, v_n\}$							

时间复杂度

- 计算过程中的基本运算为加法和比较。考虑加法运算的次数。
- 需要计算的 f 函数：

1. $\forall v_i \neq v_1$， $\forall U \subset V$，满足 $|U|=1$ 且 $v_1, v_i \notin U$， $f_1(v_i, U, v_1)$。
2. $\forall v_i \neq v_1$， $\forall U \subset V$，满足 $|U|=2$ 且 $v_1, v_i \notin U$， $f_2(v_i, U, v_1)$。
3. $\cdots$
4. $\forall v_i \neq v_1$，$\forall U \subset V$，满足 $|U|=n-2$ 且 $v_1, v_i \notin U$，$f_{n-2}(v_i, U, v_1)$。

- 因此，需要计算 $(n-1)\binom{n-2}{1}$ 个 $f_1(v_i, U, v_1)$， $(n-1)\binom{n-2}{2}$ 个 $f_2(v_i, U, v_1)$， $\ldots$， $(n-1)\binom{n-2}{n-2}$ 个 $f_{n-2}(v_i, U, v_1)$。

时间复杂度

- 计算每个 $f_k(v_i, U, v_1)$ 需要计算 **k** 次加法，$\boldsymbol{k = 1 .. n-1}$。
- 因此，计算 $f_1(v_i, U, v_1)$，…，$f_{n-2}(v_i, U, v_1)$，以及最后一个 $f_{n-1}(v_1, U, v_1)$，所需要的加法运算的次数为：

$$T = (n-1) + (n-1)\sum_{k=1}^{n-2}\binom{n-2}{k}k$$。

- 当 $\boldsymbol{x = 1}$ 时，$\displaystyle\sum_{k=1}^{n-2}\binom{n-2}{k}k = \sum_{k=1}^{n-2}\binom{n-2}{k}kx^{k-1} = \left(\sum_{k=1}^{n-2}\binom{n-2}{k}x^k\right)'$

$$= \left(\sum_{k=0}^{n-2}\binom{n-2}{k}x^k\right)' = \left((1+x)^{n-2}\right)' = (n-2)(1+x)^{n-3} = (n-2)2^{n-3}$$。

时间复杂度

- 因此，$T=(n-1)+(n-1)(n-2)2^{n-3}$。
- 比较运算的次数与加法运算的次数是同量级的。因此，上述 **TSP** 的动态规划算法的时间复杂度为 **O(*n*²2*ⁿ*⁻³)**。这已经比枚举法的时间复杂度 **O(*n*!)**好了很多。

谢谢大家