动态规划

动态规划的思想方法

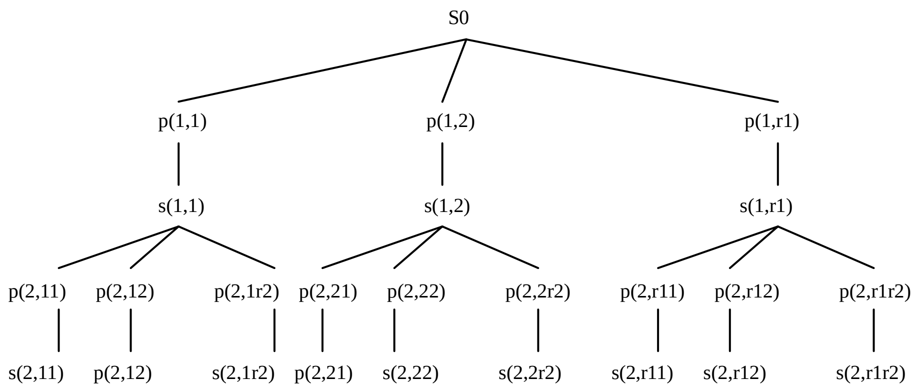
动态规划的最优决策原理

活动过程划分为若干个阶段，每一阶段的决策，依赖于前一阶段的状态，由决策所采取的动作，使状态发生转移，成为下一阶段的决策依据。

$$S_0 \xrightarrow{P_1} S_1 \xrightarrow{P_2} S_2 \;\text{------}\; S_{n-1} \xrightarrow{P_n} S_n$$

动态规划的决策过程

最优性原理：无论过程的初始状态和初始决策是什么，其余决策都必须相对于初始决策所产生的状态，构成一个最优决策序列。



令最优状态为 $s(2,22)$，由此倒推：

$s(2,22) \rightarrow p(2,22) \rightarrow s(1,2) \rightarrow p(1,2) \rightarrow s0$

最优决策序列，$p(1,2) \rightarrow p(2,22)$

状态转移序列：$s0 \rightarrow s(1,2) \rightarrow s(2,22)$

赖以决策的策略或目标，称为动态规划函数。

整个决策过程，可以递归地进行，或用循环迭代的方法进行。

动态规划函数可以递归地定义，也可以用递推公式来表达。

最优决策是在最后阶段形成的，然后向前倒推，直到初始阶段；

而决策的具体结果及所产生的状态转移，却是由初始阶段开始进行计算的，然后向后递

归或迭代，直到最终结果。

动态规划实例、货郎担问题

例 货郎担问题。

在有向赋权图$G=<V,E>$中，寻找路径最短的哈密尔顿回路问题。

一、解货郎担问题的过程

令$d(i,\overline{V})$：从顶点i出发，经$\overline{V}$中各顶点一次，最终回到顶点i的最短路径的长度，开始时，$\overline{V}=V-\{i\}$。

动态规划函数：

$$d(i,V-\{i\})=d(i,\overline{V})=\min_{k\in\overline{V}}\{c_{ik}+d(k,\overline{V}-\{k\})\} \quad (6.1.1)$$

$$d(k,\varphi)=c_{ki} \qquad k\neq i \quad (6.1.2)$$

4 个城市的费用矩阵是：

$$C=(c_{ij})=\begin{pmatrix} \infty & 3 & 6 & 7 \\ 5 & \infty & 2 & 3 \\ 6 & 4 & \infty & 2 \\ 3 & 7 & 5 & \infty \end{pmatrix}$$

根据（6.1.1）式，由城市 1 出发，经城市 2, 3, 4 然后返回 1 的最短路径长度为：

$$d(1,\{2,3,4\})=\min\{c_{12}+d(2,\{3,4\}),c_{13}+d(3,\{2,4\}),c_{14}+d(4,\{2,3\})\}$$

它必须依据$d(2,\{3,4\}),d(3,\{2,4\}),d(4,\{2,3\})$的计算结果：

$$d(2,\{3,4\})=\min\{c_{23}+d(3,\{4\})\ ,c_{24}+d(4,\{3\})\}$$

$$d(3,\{2,4\})=\min\{c_{32}+d(2,\{4\})\ ,c_{34}+d(4,\{2\})\}$$

$$d(4,\{2,3\})=\min\{c_{42}+d(2,\{3\})\ ,c_{43}+d(3,\{2\})\}$$

这一阶段的决策，又必须依据下面的计算结果：

$$d(3,\{4\}),d(4,\{3\}),d(2,\{4\})\ \ ,d(4,\{2\}),d(2,\{3\}),d(3,\{2\})$$

再向前倒推，有：

$$d(3,\{4\})=c_{34}+d(4,\varphi\ )=c_{34}+c_{41}=2+3=5$$

$$d(4,\{3\})=c_{43}+d(3,\varphi\ )=c_{43}+c_{31}=5+6=11$$

$$d(2,\{4\})=c_{24}+d(4,\varphi)=c_{24}+c_{41}=3+3=6$$

$$d(4,\{2\})=c_{42}+d(2,\varphi)=c_{42}+c_{21}=7+5=12$$

$$d(2,\{3\})=c_{23}+d(3,\varphi)=c_{23}+c_{31}=2+6=8$$

$$d(3,\{2\})=c_{32}+d(2,\varphi)=c_{32}+c_{21}=4+5=9$$

有了这些结果，再向后计算，有：

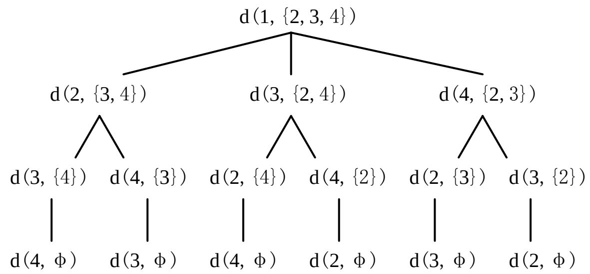
$$d(2,\{3,4\})=\min\{2+5\ ,3+11\}=7$$ 路径顺序是：2, 3, 4, 1

$d(3,\{2,4\}) = \min\{4+6, 2+12\} = 10$　　　　路径顺序是：3, 2, 4, 1

$d(4,\{2,3\}) = \min\{7+8, 5+9\} = 14$　　　　路径顺序是：4, 3, 2, 1

最后：

$d(1,\{2,3,4\}) = \min\{3+7, 6+10, 7+14\} = 10$　　路径顺序是：1, 2, 3, 4, 1



货郎担问题求解过程示意图

二、复杂性分析

令 N_i 是计算从顶点 i 出发，返回顶点 i 所需计算的形式为 $d(k, \overline{V}-\{k\})$ 的个数。

开始计算 $d(i, V-\{i\})$ 时，集合 $V-\{i\}$ 中有 $n-1$ 个城市。

以后，在计算 $d(k, \overline{V}-\{k\})$ 时，集合 $\overline{V}-\{k\}$ 的城市数目，在不同的决策阶段，分别为 $n-2$, ---, 0。

在整个计算中，需要计算大小为 j 的不同城市集合的个数为 C_{n-1}^{j}， $j = 0, 1, \cdots n-1$。因此，总个数为：

$$N_i = \sum_{j=0}^{n-1} C_{n-1}^{j}$$

当 $\overline{V}-\{k\}$ 集合中的城市个数为 j 时，为计算 $d(k, \overline{V}-\{k\})$，需 j 次加法， $j-1$ 次比较。从 i 城市出发，经其它城市再回到 i，总的运算时间 T_i 为：

$$T_i = \sum_{j=0}^{n-1} j \cdot C_{n-1}^{j} < \sum_{j=0}^{n-1} n \cdot C_{n-1}^{j} = n \sum_{j=0}^{n-1} C_{n-1}^{j}$$

由二项式定理：

$$(x+y)^n = \sum_{j=0}^{n} C_n^j x^j y^{n-j}$$

令 $x = y = 1$；可得：

$$T_i < n \cdot 2^{n-1} = O(n 2^n)$$

则用动态规划方法求解货郎担问题，总的花费 T 为：

$$T = \sum_{i=1}^{n} T_i < n^2 \cdot 2^{n-1} = O(n^2 2^n)$$

0/1 背包问题

0/1 背包问题的求解过程

一、动态规划函数

x_i：物体 i 被装入背包的情况，$x_i = 0, 1$。约束方程和目标函数：

$$\sum_{i=1}^{n} w_i x_i \le M \qquad optp = \max \sum_{i=1}^{n} p_i x_i$$

解向量 $X = (x_1, x_2, \cdots, x_n)$。

背包的载重量： $0 \sim m$

$optp_i(j)$：前 i 个物体中，能装入载重量为 j 的背包中的物体的最大价值，$j = 1, 2, \cdots, m$。

动态规划函数：

$$optp_i(0) = optp_0(j) = 0 \qquad (6.6.1)$$

$$optp_i(j) = \begin{cases} optp_{i-1}(j) & j < w_i \\ \max\{optp_{i-1}(j), optp_{i-1}(j - w_i) + p_i\} & j > w_i \end{cases} \qquad (6.6.2)$$

二、求解过程

1、决策阶段

第一阶段，只装入一个物体，确定在各种不同载重量的背包下，能够得到的最大价值；

第二阶段，装入前两个物体，确定在各种不同载重量的背包下，能够得到的最大价值；

依此类推，直到第 n 个阶段。

最后，$optp_n(m)$ 便是在载重量为 m 的背包下，装入 n 个物体时，能够取得的最大价值。

2、解向量的确定

从 $optp_n(m)$ 的值向前倒推。

递推关系式：

若 $optp_i(j) \le optp_{i-1}(j)$ 则 $x_i = 0$ （6.6.3）

若 $optp_i(j) > optp_{i-1}(j)$ 则 $x_i = 1$，$j = j - w_i$ （6.6.4）

例 6.6 有 5 个物体，其重量分别为 2, 2, 6, 5, 4，价值分别为 6, 3, 5, 4, 6，背包的载重量为 10，求装入背包的物体及其总价值

计算结果，如图所示。

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	***6***	6	6	6	6	6	6
2	0	0	6	6	9	9	***9***	9	9	9	9
3	0	0	6	6	9	9	9	9	11	11	14
4	0	0	6	6	9	9	9	10	11	13	14
5	0	0	6	6	9	9	12	12	15	15	***15***

5 个物体的 0/1 背包问题的例子

装入背包的物体为 $x=\{1,1,0,0,1\}$ 。

6.6.2　0/1 背包问题的实现

数据结构。下面的数据用于算法的输入和输出：

```
int   w[n];                /* n 个物体的重量 */
Type  p[n];                /* n 个物体的价值 */
int   m;                   /* 背包的载重量 */
BOOL  x[n];                /* 装入背包的物体,元素为 true 时,对应物体被装入 */
Type  v;                   /* 装入背包中物体的最大价值 */
```

下面的数据用于算法的工作单元：

```
Type  optp[n+1][m+1];      /* i 个物体装入载重量为 j 的背包中的最大价值 */
```

算法描述如下：

算法 6.5　0/1 背包物体的动态规划算法

输入：物体的重量 w[]和价值 p[],物体的个数 n,背包的载重量 m

输出：装入背包的物体 x[],背包中物体的最大价值 v

```
template <class Type>
Type knapsack_dynamic(int w[],Type p[],int n,int m,BOOL x[])
{
    int i,j,k;
    Type v,(*optp)[m+1] = new Type[n+1][m+1]; /* 分配工作单元 */
    for (i=0;i<=n;i++){                        /* 初始化第 0 列 */
        optp[i][0] = 0;   x[i] = FALSE;     /* 解向量初始化为 FALSE */
    }
    for (i=0;i<=m;i++)                      /* 初始化第 0 行 */
```

```
10.         optp[0][i] = 0;
11.     for (i=1;i<=n;i++) {                      /* 计算 optp[i][j] */
12.         for (j=1;j<=m;j++) {
13.             optp[i][j] = optp[i-1][j];
14.             if ((j>=w[i])&&(optp[i-1,j-w[i]]+p[i]>optp[i-1][j])
15.                 optp[i][j] = optp[i-1,j-w[i]]+p[i];
16.         }
17.     }
18.     j = m;                                    /* 递推装入背包的物体 */
19.     for (i=n;i>0;i--) {
20.         if (optp[i][j]>optp[i-1][j]) {
21.             x[i] = TRUE;   j = j – w[i];
22.         }
23.     }
24.     v = optp[n][m];
25.     delete optp;                          /* 释放工作单元 */
26.     return v;                             /* 返回最大价值 */
27. }
```

时间复杂性是$\Theta(nm)$。

第 6~8 行、第 9~10 行都花费$\Theta(m)$时间;

第 11~17 行花费$\Theta(nm)$时间;

第 18~23 行花费$\Theta(n)$时间;

工作空间是$O(nm)$