算法第三次作业

朱一帆-222018321212058-智科

2020年4月22日 17:22

算法分析题：5个物体，重量分别为3，5，7，8，9，价值分别为4，6，7，9，10，背包容量为22，物体不能分割，求装入背包物体最大价值，写出求解过程。

解：质量 $W_i = 3, 5, 7, 8, 9$.

价值 $V_i = 4, 6, 7, 9, 10$.

用矩阵 m[i][j] 来存储相应值.

m

j/i	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	4	0	0	0	0
4	0	4	0	0	0	0
5	0	6	6	0	0	0
6	0	6	6	0	0	0
7	0	7	7	7	0	0
8	0	10	9	9	9	0
9	0	10	10	10	10	10
10	0	11	10	10	100	101
11	0	13	10	10	10	10
12	0	14	13	10	10	10
13	0	15	15	10	10	10
14	0	16	16	10	10	10
15	0	17	16	16	10	10
16	0	19	17	17	10	10
17	0	20	19	19	19	10
18	0	20	19	19	19	10
19	0	21	19	19	19	10
20	0	22	22	19	19	10
21	0	24	23	19	19	10
22	0	25	25	19	19	10

最后输出结果：0 1 1 0 1

最优值为 V = 25

最优组为 W_1, W_3, W_5.

算法实现题：最少硬币问题

问题描述：设有n种不同面值的硬币，各硬币的面值存于数组$T[1:n]$中。现要用这些面值的硬币来找钱。可以使用的各种面值的硬币个数存于数组$coins[1:n]$中。
对任意钱数$0 \le m \le 20001$，设计一个用最少硬币找钱m的方法。

算法设计：对于给定的$1 \le n \le 10$，硬币面值数组T和可用使用的各种面值的硬币个数数组$coins$，以及钱数m，$0 \le m \le 20001$，计算找钱m的最少硬币数。

解：设计该算法的主要思路是，先尽可能使用较大币值的钱，如果较大金额的钱不足或者超出了所需的最大钱数则考虑较小币值的。因此设计一个二维数组$m(i,j)$来存放结果。每一行表示钱的个数，每一列表示钱的面值。
先从最大的面值开始取，直到超出金额或者用完为止，将这一结果保存后继续抽取较小面额的钱币，直到抽取的钱币数达到要求或者是使用完所有的钱币。最后做一个辅助数组统计每种币值的钱使用量。程序如下。

```
void coins(int* T, int* coins, int **m,int n,int sum)
{
    int max=0;
    m[0][n] = 0;
    for (int i = 1; i <= 10; i++)
    {
        if (m[0][n] +  i*T[n] <= sum && i<=coins[n])
        {
            m[i][n] = i * T[n];
            max = m[i][n];
        }
        else
            m[i][n] = 0;
    }
    for (int j = n - 1; j >= 1;j--)
    {
        m[0][j] = max;
        for (int i = 1; i <= 10; i++)
        {
            if (m[0][j] + i*T[j] <= sum && i <= coins[j])
            {
                m[i][j] = m[i - 1][j] + T[j];
                max = m[i][j];
            }
            else
                m[i][j] = 0;
        }
    }
    if (max < sum)  printf("补钱数少于总数");
}
```

O(1)
O(n-1)
O(m)
m

```
int main()
{
    int T[7] = { 0,1,2,5,10,20,50 };
    int coin[7] = { 0,4,5,8,5,2,1 };

    int **m = new int*[11];
    for (int i = 0; i <= 11; i++)
        m[i] = new int[7];
    for (int i = 0; i <= 11; i++)
        m[i][0] = i;

    int sum = 109;
    coins(T, coin, m, 6, sum);
    int s[7] = { 0 };
    traceback(m, s, 6);

    for (int i = 1; i <= 6; i++)
        cout << T[i] << "\t" << s[i] << endl
```

```
C:\WINDO
1       0
2       2
5       1
10      1
20      2
50      1
```

设共n个币值，每个币值最多m枚硬币，则时间复杂度为$O(mn)$。

```
void traceback(int **m,int *s,int n)
{

    s[0] = 0;
    for (int j = 1; j <= n; j++)
    {
        for (int i = 1; i <= 10; i++)
            if (!m[i][j])
            {
                s[j] = i - 1;
                break;
            }
    }
}
```

O(n)
O(m)
O(mn)

课本P81 3-4 给定n种物品和一背包。物品i的重量是w_i，体积是b_i，其价值为v_i，背包的容量为c，容积为d。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大？装入的物品遵循0-1背包问题。试设计一个解此问题的动态规划算法，并分析算法的计算复杂性。

解：该问题与0-1背包问题类似，只是多增加了一个参数体积，扩大成一个二维的问题。因此原来的二维数组也要扩充为三维数组。

设所给0-1背包问题的子问题

$$\max\sum_{k=i}^{n}v_k x_k \qquad \begin{cases}\sum_{k=i}^{n}w_k x_k \le j\\ x_k\in(0,1) \quad \mathrm{i}\le k\le n\end{cases}$$

的最优值为$m(\mathrm{i},j,k)$，即$m(\mathrm{i},j,k)$是背包容量为j，容积为k，可选择物品为$\mathrm{i},\mathrm{i}+1,\cdots,n$时0-1背包问题的最优值。由0-1背包问题的最优子结构性质，可以建立如下递归式。

$$m(\mathrm{i},j,k)=\begin{cases}\max(m(i+1,j,k),m(i+1,\mathrm{j}-w_i,\mathrm{k}-b_{b_i i})+v_i) & j\ge w_i且k\ge b_i\\ m(x_i+1,j,k) & 0\le j<w_i或0\le k<b_i\end{cases}$$

$$m(\mathrm{i},j,k)=\begin{cases}v_n & j\ge w_i且k\ge b_i\\ 0 & 0\le j<w_i或0\le k<b_i\end{cases}$$

代码与0-1背包问题类似，只是将数组变为三维数组，并增加一重循环就能实现。

```
void Knapsack(int* v, int* w, int* b, int c, int d, int n, int*** m)
{
    int jMax = min(w[n] - 1, c);
    int kMax = min(b[n] - 1, d);
    for (int j = 0; j <= jMax; j++)
        for (int k = 0; k <= kMax; k++)
            m[n][j][k] = 0;
    for (int j = w[n]; j <= c; j++)
        for (int k = b[n]; k <= d; k++)
            m[n][j][k] = v[n];

    for (int i = n - 1; i >= 1; i--)
    {
        jMax = min(w[i] - 1, c);
        kMax = min(b[i] - 1, c);
        for (int j = 0; j <= jMax; j++)
            for (int k = 0; k <= kMax; k++)
                m[i][j][k] = m[i + 1][j][k];
        for (int j = w[i]; j <= c; j++)
            for (int k = b[i]; k <= d; k++)
                m[i][j][k] = max(m[i + 1][j][k], m[i + 1][j - w[i]][k - b[i]] + v[i]);
    }
}
```

```
int main()
{
    int w[6] = { 0, 2, 2, 6, 5, 4 };
    int b[6] = { 0, 2, 5, 6, 4, 2 };
    int v[6] = { 0, 6, 3, 5, 4, 6 };
    int j = 14;
    int k = 13;
    int*** m = new int**[6];
    for (int i = 0; i < 6; i++)
        m[i] = new int*[15];
    for (int i = 0; i < 6; i++)
        for (int j = 0; j < 15; j++)
            m[i][j] = new int[14];

    Knapsack(v, w, b, j, k, 5, m);

    int x[6] = { 0 };
    Traceback(m, w, b, j, k, 5, x);

    for (int i = 1; i <= 5; i++)
        cout << x[i] << " ";
    cout << endl;
}
```

```
void Traceback(int*** m, int* w, int* b, int c, int d, int n, int *x)
{
    for (int i = 1; i < n; i++)
    {
        if (m[i][c][d] == m[i + 1][c][d])
            x[i] = 0;
        else
        {
            x[i] = 1;
            c -= w[i];
            d -= b[i];
        }
    }
    x[n] = (m[n][c][d]) ? 1 : 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
1 1 0 1 1
请按任意键继续. . .
```