《算法设计与分析》习题 6 作业

学　号：1004191211　　姓　名：郎文鹏

日　期：2021/10/21　　得　分：

问题一：动态规划为什么都需要填表？如何设计表格的结构？

【第一问】动态规划法将带求解问题分解为若干个相互重叠的子问题，通过将每个子问题求解结果填表可以避免重复计算，有效降低算法的复杂度，其自下而上计算的特点导致需要一种数据结构用来存储每个子决策。

【第二问】设计表格按照以自底向上计算各个子问题解的顺序并填表从而实现动态规划过程，表格结构设计一般需要满足快速查改，节省空间等目标。

问题二：对于所示多段图，用动态规划法求从顶点 0 到顶点 2 的最短路径，写出求解过程

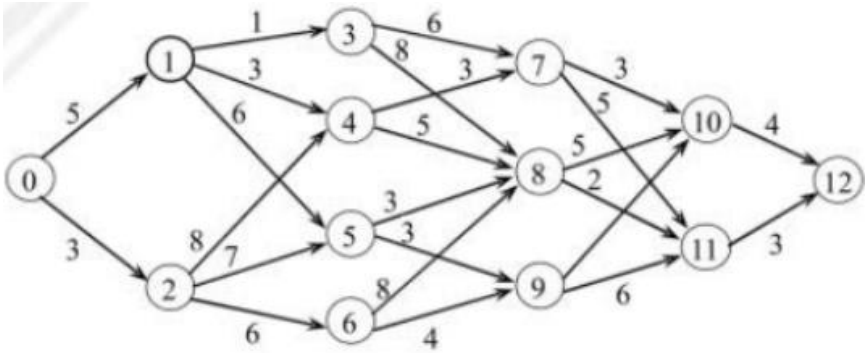

图 6.26　第 2 题图

图 1 问题二插图

【原理】将图划分为k个不相交的子集，不同子集按照从起点到终点顺序进行求解，相同子集内部的点顺序随意进行求解。比较特殊的是这题恰好是按照点号升序进行求解即可。可以证明多段图的最短路径问题满足最优性原理，当前处理的点最优解由前面所连点的最优解中转移过来，即：

$$\begin{cases} dis[v] = graph[s][v] & < s,u > \in E \\ dis[v] = min(dis[u] + graph[u][v] & < u,v > \in E \end{cases}$$

【变式】很明显最短路径可能存在很多条，按照上述公式进行编程只能求出一条，在二重循环进行转移时根据枚举转移点的顺序不同求出来的结果也可能不同，为了求出所有的结果我们可以给每个点开一个数组分别记录所有最短路径结果。

【调试】为了更加详细的了解动态规划法中间过程的变化，我们分别在求每个点的最优解时记录以下信息：

➢找到另外一个可以走的转移点时会扩展出更多的路径，所以我们记录该点并记录拓展出的所有新路径。

➢找到一个更优的转移点时会覆盖之前求出的所有路径，所以我们记录该点并清空之前记录的所有路径，记录下从该点扩展出的所有新路径。

【源码】我们将图中的所有点和边存入用来生成图，按照原理进行编程实现并中间设置输出信息的代码，输入和输出分别存入本地记事本。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 25;  //顶点数

int n, m;
int dis[maxn], pre[maxn];
int graph[maxn][maxn];
vector<string> path[maxn];

void dp_path() {
    memset(dis, 0x3f, sizeof dis);
    dis[0] = 0;
    path[0].push_back("0");
    for (int i = 1; i < n; ++i) {
        cout << "------- " << i << " ------" << endl;
        for (int j = 0; j <= i - 1; ++j) {
            if (dis[i] == dis[j] + graph[j][i]) {
                for (auto &e : path[j]) path[i].push_back(e + "->" +
to_string(i));
                cout << "find another transit point: " << j << endl;
            }
            if (dis[i] > dis[j] + graph[j][i]) {
                dis[i] = dis[j] + graph[j][i], path[i].clear();
                for (auto &e : path[j]) path[i].push_back(e + "->" +
to_string(i));
                cout << "find a better transit point: " << j << endl;
            }
        }

        for (int j = 0; j <= i; ++j) {
            cout << "dis[" << j << "]: " << dis[j] << '\t';
```

```
            for (auto &e : path[j])
                cout << e << '\t';
            cout << endl;
        }
        cout << endl;
    }
}

void solve() {
    cin >> n >> m;
    memset(graph, 0x3f, sizeof graph);
    for (int i = 1; i <= m; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u][v] = w;
    }
    dp_path();
    cout << "------- ans ------" << endl;
    cout << dis[n - 1] << endl;
    for (auto &e : path[n - 1]) cout << e << endl;
}

signed main() {
    ios::sync_with_stdio(false), cin.tie(0);
    freopen("IO\\in.txt", "r", stdin);
    freopen("IO\\out.txt", "w", stdout);
    solve();
    return 0;
}
```

【过程】最终我们考虑的应该输出的最后找到最优转折点及后续信息输出。

多段图动态规划法中间过程输出信息（n=0…12）

编号	中转点	最短距离	最优路径
1	0	5	0->1
2	0	3	0->2
3	0、1	6	0->1->3
4	1	8	0->1->4
5	2	10	0->2->5
6	2	9	0->2->6
7	4	11	0->1->4->7
8	4、5	13	0->1->4->8 0->2->5->8
9	5、6	13	0->2->5->9 0->2->6->9

10	7	14	0->1->4->7->10
11	8	15	0->1->4->8->11 0->2->5->8->11
12	10、11	18	0->1->4->7->10->12 0->1->4->8->11->12 0->2->5->8->11->12

表 1 代码输出信息汇总表

【答案】最终从顶点 0 到顶点 12 的路径有三条，他们的距离都是 18。

问题三：用动态规划法求如下 0/1 背包的最优解：有 5 个物品，其重量分别为（3,2,1,4,5），价值分别为（25,20,15,40,50），背包容量为 6。写出求解过程。

【原理】每个物品在$[0,V]$容量合法情况下都有装或者不装两种选择，所以我们可以随意顺序枚举每个物品，考虑当前物品在上一个物品最优解的不同容量下装或者不装的情况，然后选择最优解记录即可转到当前物品，即：

$$dp[i][j]=\begin{cases} dp[i-1][j] & j<v[i] \\ max(dp[i-1,j],dp[i-1][j-v[i]]+w[i] & j\geq v[i] \end{cases}$$

【变式】该题很明显只有一种最优解路径，考虑如果有多个路径我们选择字典序最小的路径，那么为了回溯路径方便我们将上述按照物品升序顺序递推转变为逆序递推，这样在回溯时即可正向比较。

【调试】为了更详细了解动态规划过程中物品选择情况，加入输出每个不同物品在不同容量下最优解转移情况并记录：

➢如果当前物品放不下或者放得下但是从上个物品最优解转移过来小于不选情况最优解则直接由上个物品最优解转移过来。

➢反之则选择该物品并更新最优解。

【源码】我们将每个物品的容量与价值分别输入，按照原理进行代码编程并输出上述调试信息，输入和输出分别存入本地记事本。

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e3 + 5;

int v[maxn], w[maxn], dp[maxn][maxn];  //维护的是从第 i 个物品到第 n 个物品在剩余容量为 j 的情况下能得到的最大价值

void solve() {
    int n, V;
    cin >> n >> V;
```

```
    for (int i = 1; i <= n; ++i) cin >> v[i] >> w[i];
    for (int i = n; i >= 1; --i) {
        cout << "----- " << i << " ----" << endl;
        for (int j = 0; j <= V; ++j) {
            dp[i][j] = dp[i + 1][j];
            if (j >= v[i] && dp[i + 1][j - v[i]] + w[i] > dp[i][j]) {
                dp[i][j] = dp[i + 1][j - v[i]] + w[i];
                cout << dp[i][j] << '\t' << "when volumn is " << j << "
choose " << i << endl;
                continue;
            }
            cout << dp[i][j] << '\t' << "when volumn is " << j << " do
not choose " << i << endl;
        }
        cout << endl;
    }
    cout << "---- ans ----" << endl;
    cout << dp[1][V] << endl;
    int p = V;
    for (int i = 1; i <= n; ++i) {
        if (p == 0)  //背包满了直接退出
            break;
        else if (i == n && p >= v[i]) {  //特判最后一个物品需不需要装进背包
            cout << i << ' ';
            break;
        } else if (p >= v[i] && dp[i][p] == dp[i + 1][p - v[i]] + w[i])
{  //说明可以装下这个物品且装下后满足最优
            cout << i << ' ';
            p -= v[i];
        }
    }
    cout << endl;
}

signed main() {
    ios::sync_with_stdio(false), cin.tie(0);
    freopen("IO\\in.txt", "r", stdin);
    freopen("IO\\out.txt", "w", stdout);
    solve();
    return 0;
}
```

【过程】最后我们寻找的应该是最优解路径并输出所有状态的最优解。

	0	5	4	3	2	1
0	0	0	0	0	0	0
1	0	0	0	15	15	15
2	0	0	0	15	20	20
3	0	0	0	15	35	35
4	0	0	40	40	40	40
5	0	50	50	55	55	55
6	0	50	50	65	65	65

图 2 状态统计图

【答案】最终选择 3、5 号物品，占用空间 6，权值为 65。