背包问题 + 贪心问题 + Dijkstra

01背包问题描述

> 有n个物品和一个容量为m的背包，每个物品只能选或不选，第i个物品的体积为v[i]，价值为w[i]，求解将哪些物品装入背包可使这些物品的体积总和不超过背包容量，且价值总和最大。

```
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 1010;
int n, m;
int v[N], w[N];
int f[N][N];

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i ++ ) cin >> v[i] >> w[i];

    for (int i = 1; i <= n; i ++ )
        for (int j = 0; j <= m; j ++ )
        {
            f[i][j] = f[i - 1][j]; // 初始化
            if (j >= v[i]) f[i][j] = max(f[i][j], f[i - 1][j - v[i]] + w[i]); //
转移方程
        }

    cout << f[n][m] << endl; // 输出结果

    return 0;
}
```

```
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 1010;
int n, m;
int v[N], w[N];
int f[N];

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i ++ ) cin >> v[i] >> w[i];
```

```
    for (int i = 1; i <= n; i ++ )
        for (int j = m; j >= v[i]; j -- )
            f[j] = max(f[j], f[j - v[i]] + w[i]);

    cout << f[m] << endl; // 输出结果

    return 0;
}
```

```
// 测试数据1
// 输入:
// 4 5
// 1 2
// 2 4
// 3 4
// 4 5
// 输出:
// 8

// 测试数据2
// 输入:
// 5 10
// 2 6
// 2 3
// 6 5
// 5 4
// 4 6
// 输出:
// 15

// 测试数据3
// 输入:
// 3 5
// 2 3
// 2 4
// 1 2
// 输出:
// 7

// 测试数据4
// 输入:
// 2 3
// 1 2
// 2 1
// 输出:
// 2

// 测试数据5
// 输入:
// 1 10
// 5 10
```

```
// 输出:
// 0
```

贪心算法

> 活动安排问题: 有n个活动，每个活动有一个开始时间和结束时间，求最多能参加多少个活动，假设一个人同时只能参加一个活动

解题思路： 按照结束时间从小到大排序，每次选择结束时间最早且和前面的活动不冲突的活动

```
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 1e5 + 10;
struct Activity {
    int start, end;
    //比较函数 重载<号，方便后面排序
    bool operator< (const Activity& t) const {
        return end < t.end;
    }
} act[N];

int main() {
    int n;
    cin >> n; // 输入活动数量
    for (int i = 0; i < n; i++) {
        cin >> act[i].start >> act[i].end; // 输入每个活动的开始时间和结束时间
    }
    sort(act, act + n); // 按照结束时间从小到大排序
    int cnt = 1, end = act[0].end; // 初始化计数器和结束时间
    for (int i = 1; i < n; i++) {
        if (act[i].start >= end) { // 如果当前活动的开始时间晚于等于上一个活动的结束时
间
            cnt++; // 计数器加1
            end = act[i].end; // 更新结束时间
        }
    }
    cout << cnt << endl; // 输出最多能参加的活动数量
    return 0;
}
```

```
// 测试数据1
// 输入:
// 3
// 1 3
```

```
// 2 5
// 4 7
// 输出:
// 2

// 测试数据2
// 输入:
// 4
// 1 3
// 2 5
// 4 7
// 6 9
// 输出:
// 2

// 测试数据3
// 输入:
// 5
// 1 3
// 2 5
// 4 7
// 6 9
// 8 10
// 输出:
// 3

// 测试数据4
// 输入:
// 2
// 1 3
// 3 5
// 输出:
// 1

// 测试数据5
// 输入:
// 1
// 1 3
// 输出:
// 1
```

Dijkstra

> 题目描述：给定一个有向图，求起点到终点的最短距离
> 输入格式：第一行包含四个整数n,m,s,t，表示图的大小和起点终点编号。
> 接下来m行每行包含三个整数a,b,c，表示存在一条从a到b的有向边，边长为c。
> 输出格式：输出一个整数，表示起点到终点的最短距离，如果从起点无法到达终点，则输出-1。

```
#include <iostream>
#include <cstring>
```

```cpp
#include <algorithm>
using namespace std;



const int N = 510, INF = 0x3f3f3f3f;
int n, m, s, t;
int g[N][N], dist[N];
bool st[N];

int dijkstra()
{
    memset(dist, 0x3f, sizeof dist); // 初始化距离数组
    dist[s] = 0; // 起点到起点的距离为0

    for (int i = 0; i < n; i ++ )
    {
        int t = -1; // t用来记录当前未确定最短路的点中距离最小的点
        for (int j = 1; j <= n; j ++ )
            if (!st[j] && (t == -1 || dist[t] > dist[j])) // 如果j未确定最短路且j的
距离比t小
                t = j; // 更新t为j

        st[t] = true; // 标记t为已确定最短路

        for (int j = 1; j <= n; j ++ )
            dist[j] = min(dist[j], dist[t] + g[t][j]); // 用t更新其他点的距离
    }

    if (dist[t] == INF) return -1; // 如果终点不可达，返回-1
    return dist[t]; // 返回起点到终点的最短距离
}

int main()
{
    cin >> n >> m >> s >> t;

    memset(g, 0x3f, sizeof g); // 初始化邻接矩阵
    while (m -- )
    {
        int a, b, c;
        cin >> a >> b >> c;
        g[a][b] = min(g[a][b], c); // 有重边，取最小值
    }

    cout << dijkstra() << endl; // 输出结果

    return 0;
}

// 测试数据1
// 输入：
// 3 3 1 3
// 1 2 2
```

```
// 2 3 1
// 1 3 4
// 输出:
// 3

// 测试数据2
// 输入:
// 3 3 1 3
// 1 2 2
// 2 3 1
// 1 3 1
// 输出:
// 1

// 测试数据3
// 输入:
// 3 3 1 3
// 1 2 2
// 2 3 1
// 1 3 2
// 输出:
// 2

// 测试数据4
// 输入:
// 3 3 1 3
// 1 2 2
// 2 3 1
// 1 3 3
// 输出:
// 3

// 测试数据5
// 输入:
// 3 3 1 3
// 1 2 2
// 2 3 1
// 1 3 5
// 输出:
// -1
```