

一、本周作业

作业 1：最大字典序（注意输出是 3 4 不是 4 3）

### 8、背包问题求具体方案

有  $N$  件物品和一个容量是  $M$  的背包。每件物品只能使用一次。  
第  $i$  件物品的体积是  $w_i$ ，价值是  $v_i$ 。  
求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。  
输出 字典序最大的方案。这里的字典序是指：所选物品的编号所构成的序列。  
物品的编号范围是  $1 \dots N$ 。

**输入格式**  
第一行两个整数， $N, V$ ，用空格隔开，分别表示物品数量和背包容积。  
接下来有  $N$  行，每行两个整数  $w_i, v_i$ ，用空格隔开，分别表示第  $i$  件物品的体积和价值。

**输出格式**  
输出一行，包含若干个用空格隔开的整数，表示最优解中所选物品的编号序列，且该编号序列的字典序最大。  
物品编号范围是  $1 \dots N$ 。

**数据范围**  
 $0 < N, V \leq 1000$   
 $0 < w_i, v_i \leq 1000$

输入样例

4 7

1 1

2 3

3 4

4 5

输出样例

3 4

作业 2：二维费用背包

一天，小智和皮卡丘来到了小精灵狩猎场，里面有很多珍贵的野生宠物小精灵。小智也想收服其中的一些小精灵。然而，野生小精灵并不容易收服。对于每一个野生小精灵而言，小智可能需要使用很多个精灵球才能收服它，而在收服过程中，野生小精灵也会对皮卡丘造成一定的伤害（从而减少皮卡丘的体力）。当皮卡丘的体力小于等于 0 时，小智就必须结束狩猎（因为他需要给皮卡丘疗伤），而使得皮卡丘体力小于等于 0 的野生小精灵也不会被小智收服。当小智的精灵球用完时，狩猎也宣告结束。

我们假设小智遇到野生小精灵时有两个选择：收服它，或者离开它。如果小智选择了收服，那么一定会扔出能够收服该小精灵的精灵球，而皮卡丘也一定会受到相应的伤害；如果选择离开它，那么小智不会损失精灵球，皮卡丘也不会损失体力。

小智的目标有两个：主要目标是收服尽可能多的野生小精灵；如果可以收服的小精灵数量一样，小智希望皮卡丘受到的伤害越小（剩余体力越大），因为他们还要继续冒险。

现在已知小智的精灵球数量和皮卡丘的初始体力，已知每一个小精灵需要的用于收服的精灵球数目和它在被收服过程中会对皮卡丘造成的伤害数目。请问，小智该如何选择收服哪些小精灵以达到他的目标呢？

**输入**  
输入数据的第一行包含三个整数： $N(0 < N < 1000)$ ， $M(0 < M < 500)$ ， $K(0 < K < 100)$ ，分别代表小智的精灵球数量、皮卡丘初始的体力值、野生小精灵的数量。  
之后的  $K$  行，每一行代表一个野生小精灵，包括两个整数：收服该小精灵需要的精灵球的数量，以及收服过程中对皮卡丘造成的伤害。

**输出**  
输出为一行，包含两个整数： $C, R$ ，分别表示最多收服  $C$  个小精灵，以及收服  $C$  个小精灵时皮卡丘的剩余体力值最多为  $R$ 。

样例输入1:

10 100 5

7 10

2 40

2 50

1 20

4 20

样例输出1:

3 30

二、上周作业

1、P2623

```

#include <bits/stdc++.h>
using namespace std;
const long long N = 2010;
long long n,m,f[N];
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        int pos,v,w,s;
        cin>>pos;
        if(pos==1)//01 背包变形
        {
            cin>>v>>w;
            for(int j=m;j>=0;j--) //j表示可用体积
                for(int k=1;k<=j;k++)//k表示分配给此物品的体积 1 <= k <= j
                    f[j]=max(f[j],f[j-k]+v*k-k*w*k);
        }
        else if(pos==2) //多重背包
        {
            cin>>v>>w>>s;
            for(int j=m;j>=w;j--)
                for(int k=1;k<=s&& k*w<=j;k++)
                    f[j]=max(f[j],f[j-k*w]+k*v);
        }
        else if(pos==3)//完全背包
        {
            cin>>v>>w;
            for(int j=w;j<=m;j++)
                f[j]=max(f[j],f[j-w]+v);
        }
    }
    cout<<f[m];
    return 0;
}

```

## 2、P1757

```

#include<bits/stdc++.h>
using namespace std;
const int maxn = 110;
vector<int> w[maxn];
vector<int> v[maxn];
int dp[1100];
int n, m;

int main()
{
    cin >> m >> n;
    int a, b, c;
    int g = 0;
    for(int i = 0; i < n; i++)
    {
        cin >> a >> b >> c;
        w[c].push_back(a); //不知每组物品个数，利用vector插入操作
        v[c].push_back(b);
        g = max(g, c); //统计组数
    }
}

```

```

for(int i = 1; i <= g; i++)          //组数
{
    for(int j = m; j >= 0; j--)      //容量一维逆序
    {
        for(int k = 0; k < w[i].size(); k++)    //每组件数
        {
            if(j >= w[i][k])
                dp[j] = max(dp[j], dp[j - w[i][k]] + v[i][k]);
        }
    }
}
cout << dp[m] << endl;
return 0;
}

```

### 三、课堂内容

#### 1、背包问题求方案总数

有  $N$  件物品和一个容量是  $M$  的背包，每件物品只能使用一次，第  $i$  件物品的体积是  $w_i$ ，价值是  $v_i$ 。求解将哪些物品装入背包，可以使物品的总体积不超过背包容量，且总价值最大。输出最优选法的方案数，输出答案是先模  $10^9+7$ 。

输入格式：

$n, m$  表示物品数量和背包容量。

接下来  $n$  行，每行两个整数  $w_i$  和  $v_i$ ，分别表示第  $i$  件物品的体积和价值。

输出格式

输出一个整数，表示方案数模  $10^9+7$  的结果。

输入样例：

```

5 5
5 8
2 4
3 4
5 7
4 8

```

输出样例：

```

3

```

解析：

①容量从  $j$  到  $j-w$ ，方案数不变

$num[j]=num[j-w]$

②若  $dp[j]=dp[j-w[i]]+v[i]$

$num[j] += num[j - w[i]]$

$dp[i]$ 表示容量为  $i$  时的最大价值

num[i]表示容量为 i 时的最优方案数

初始化:  $dp[i] = 0$ ,  $num[i] = 1$

```
19 int main () {
20     cin >> _n >> _m;
21     for(int i = 0; i <= _m; i++) {
22         _num[i] = 1; // 没有装物品时, 背包为空也是一种方案
23     }
24     int v = 0, w = 0;
25     for(int i = 1; i <= _n; i++) {
26         cin >> w >> v;
27         for(int j = _m; j >= w; j--) {
28             if(_dp[j - w] + v > _dp[j]) {
29                 _dp[j] = _dp[j - w] + v;
30                 _num[j] = _num[j - w];
31                 // 多装入一件物品, 容量增加, 但方案数量没有增, 赋值即可
32             }
33             else if(_dp[j - w] + v == _dp[j]) {
34                 _num[j] = (_num[j] + _num[j - w] % mod);
35                 // 在容量j下, 装入和不装入新物品的价值是相同的, 因此方案装量求和
36             }
37         }
38     }
39     cout << _num[_m];
40     return 0;
41 }
```

## 2、背包问题求具体方案

有  $N$  件物品和一个容量是  $M$  的背包。每件物品只能使用一次。

第  $i$  件物品的体积是  $w_i$ , 价值是  $v_i$ 。

求解将哪些物品装入背包, 可使这些物品的总体积不超过背包容量, 且总价值最大。

输出 字典序最小的方案。这里的字典序是指: 所选物品的编号所构成的序列。物品的编号范围是  $1 \dots N$ 。

输入格式

第一行两个整数,  $N, V$ , 用空格隔开, 分别表示物品数量和背包容积。

接下来有  $N$  行, 每行两个整数  $w_i, v_i$ , 用空格隔开, 分别表示第  $i$  件物品的体积和价值。

输出格式

输出一行, 包含若干个用空格隔开的整数, 表示最优解中所选物品的编号序列, 且该编号序列的字典序最小。

物品编号范围是  $1 \dots N$ 。

数据范围

$0 < N, V \leq 1000$

$0 < w_i, v_i \leq 1000$

例题: 最优解为 9, 为选第 1, 2, 4 件物品和选第 3, 4 件物品组成。字典序最小的序列为 1, 2, 4。



分析：为保证字典序最小，设存在一个包含第一个物品的最优解，为了确保字典序最小那么必然要选第一个，则问题转化为从  $2 \sim N$  的物品中找最优解。

①从后向前遍历物品，最优解将落在  $\_dp[1][m]$  中

②从  $\_dp[1][m]$  开始搜索字典序最小的路径方案

③  $\_dp[i][j]$  表示从第  $i$  个物品到最后一个物品装入容量为  $j$  的背包最优解。

选第  $i$  个物品：  $\_dp[i][j] = \_dp[i+1][j-w[i]] + v[i]$

不选第  $i$  个物品：  $\_dp[i][j] = \_dp[i+1][j]$

状态转移：  $\_dp[i][j] = \max(\_dp[i+1][j], \_dp[i+1][j-w[i]] + v[i])$

```
int _n, _m;
int _w[110];
int _v[110];
int _dp[110][110];
int main () {
    cin >> _m >> _n;
    for(int i = 1; i <= _m; i++) {
        cin >> _w[i] >> _v[i];
    }

    for(int i = _m; i >= 1; i--) { // 逆序取物
        for(int j = 1; j <= _n; j++) {
            if(j < _w[i]) {
                _dp[i][j] = _dp[i+1][j];
            }
            if(j >= _w[i]) {
                _dp[i][j] = max(_dp[i+1][j], _dp[i+1][j - _w[i]] + _v[i]);
            }
        }
    }

    int j = _n; // 剩余容量
    for(int i = 1; i <= _m; i++) {
        if(j >= _w[i] && _dp[i][j] == _dp[i+1][j - _w[i]] + _v[i]) { // 选了
            cout << i << " ";
            j -= _w[i];
        }
    }

    // cout << _dp[1][_n];
    return 0;
}
```

### 3、二维背包费用

题目

$N$  件物品和一个容量是  $P$  的背包，背包能承受的最大重量是  $M$ 。

每件物品只能用一次。体积是  $p_i$ ，重量是  $w_i$ ，价值是  $v_i$ 。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，总重量不超过背包可承受的最大重量，且价值总和最大。输出最大价值。

输入格式

第一行三个整数， $N$ ， $P$ ， $M$ ，用空格隔开，分别表示物品件数、背包容积和背包可承受的最大重量。

接下来有  $N$  行，每行三个整数  $p_i, w_i, v_i$ ，用空格隔开，分别表示第  $i$  件物品的体积、重量和价值。

输出格式

输出一个整数，表示最大价值。

输入样例

```
4 5 6
1 2 3
2 4 4
3 4 5
4 5 6
```

输出样例：

8

一维费用：

```
1 cin >> n >> m;
2
3 for(int i = 1; i <= n; i++) {
4     cin >> w[i] >> v[i]; //重量和价值
5 }
6
7 for(int i = 1; i <= n; i++) {
8     for(int j = m; j >= w[i]; j--) {
9         dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
10    }
11 }
12
13 cout << dp[m];
```

- $dp[j]$ : 背包承重为  $j$  时，能放入的最大价值
- $dp[m]$ : 背包承重为  $m$  时，能放入的最大价值，为全局最优解

二维费用：

```
17 for(int i = 1; i <= _n; i++) {
18     for(int j = _m; j >= _w[i]; j--) { //重量
19         for(int k = _p; k >= _p[i]; k--) { //体积
20             _dp[j][k] = max(_dp[j][k], _dp[j - _w[i]][k - _p[i]] + _v[i]);
21         }
22     }
23 }
24
25 cout << _dp[_m][_P];
```

- $dp[j][k]$ : 背包承重为  $j$ ，且体积为  $k$  时，能放入的最大价值
- $dp[m][p]$ : 背包承重为  $m$  时，体积为  $p$  时，能放入的最大价值，为全局最优解

## 四、课堂练习

### 1、有依赖的背包问题

P1064

```
#include <iostream>
#define maxn 32005
using namespace std;
int n, m;
int v, p, q;
int main_item_w[maxn]; //主件费用
int main_item_c[maxn]; //主件价值
int annex_item_w[maxn][3]; //附件费用
int annex_item_c[maxn][3]; //附件价值
int f[maxn];
```

```

int main(){
    cin >> n >> m;
    for (int i=1;i<=m;i++){
        cin >> v >> p >> q;
        //主件
        if (!q){
            main_item_w[i] = v;
            main_item_c[i] = v * p; //保存价值: 价格*重要度
        }
        /*附件:
        [q][ ]:q表示依附于哪个主件
        [q][0]:第几个附件 1或2
        W[q][[q][0]]:价格
        C[q][[q][0]]:价值(价格*重要度)
        */
        else{
            annex_item_w[q][0]++; //统计该物品附件的个数
            annex_item_w[q][annex_item_w[q][0]] = v;
            annex_item_c[q][annex_item_w[q][0]] = v * p;
        }
    }

    for (int i=1;i<=m;i++)
        for (int j=n; main_item_w[i]!=0 && j>=main_item_w[i]; j--){ //必须为主件
            f[j] = max(f[j],f[j-main_item_w[i]]+main_item_c[i]);

            if (j >= main_item_w[i] + annex_item_w[i][1])
                f[j] = max(f[j],f[ j - main_item_w[i] - annex_item_w[i][1] ] + main_item_c[i] + annex_item_c[i][1]);

            if (j >= main_item_w[i] + annex_item_w[i][2])
                f[j] = max(f[j],f[ j - main_item_w[i] - annex_item_w[i][2] ] + main_item_c[i] + annex_item_c[i][2]);

            if (j >= main_item_w[i] + annex_item_w[i][1] + annex_item_w[i][2])
                f[j] = max(f[j],f[ j - main_item_w[i] - annex_item_w[i][1] - annex_item_w[i][2] ] +
                    main_item_c[i] + annex_item_c[i][1] + annex_item_c[i][2]);
        }

    cout << f[n] << endl;
    return 0;
}

```

## 2、book.cpp

book.cpp

小明手里有n元钱全部用来买书，书的价格为10元，20元，50元，100元。

问小明有多少种买书方案？（每种书可购买多本）

输入

一个整数 n，代表总共钱数。（0 <= n <= 100000）

输出

一个整数，代表选择方案种数

样例输入

样例输入1:

20

样例输入2:

15

样例输入3:

0

样例输出

样例输出1:

2

样例输出2:

0

样例输出3:

0

代码:

```
int main(){
    int n;
    int dp[n+1];
    int w[5]= {0,10,20,50,100};

    cin >> n;

    dp[0]=1; //初始化

    if(n==0){
        cout<<0;
        return 0;
    }

    for(int i=1;i<=4;i++){//选不同面额的纸币
        for(int j = w[i]; j <= n; j++){//完全背包
            dp[j] += dp[j-w[i]];
        }
    }

    cout<<dp[n];
    return 0;
}
```