背包问题九讲 超全归纳

Genevieve_xiao 关注 IP属地: 浙江

背包问题一般套路：

```
for 物品
    for 体积
        for 决策
```

1. 01背包

题目描述

有 N 件物品和一个容量是 V 的背包。**每件物品只能使用一次。**

第 i 件物品的体积是 v_i，价值是 w_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。

输出最大价值。

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 v_i, w_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

输出一个整数，表示最大价值。

数据范围

0< N, V ≤1000

0< v_i, w_i ≤1000

输入样例

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

输出样例

```
8
```

代码

01背包朴素

状态表示： $dp[i][j]$ **只从前** i **个物品中选**，**总体积不超过** j 的选法的集合。

属性: MAX

状态计算:

- $dp[i][j] = max(dp[i-1][j], dp[i-1][j-c[i]] + w[i])$
- $dp[i-1][j]$：**不选**第 i 个物品，直接由上一层转移
- $dp[i-1][j-c[i]] + w[i]$：**选**第 i 个物品，由上一层空出体积 $c[i]$ 的状态+ $w[i]$ 得来

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

int n,v,dp[8000][8000],c[1005],w[1005];
//n物品个数 v总体积 volume[i]第i件物品体积 worth[i]第i件物品价值
int main(){
    scanf("%d%d",&n,&v);
    for(int i=1;i<=n;i++){
        scanf("%d%d",&c[i],&w[i]);
    }
    for(int i=1;i<=n;i++){ // 枚举物品
        for(int j=0;j<=v;j++){ // 枚举体积
            dp[i][j]=dp[i-1][j]; // 上一层状态转移
            if(j>=c[i]) dp[i][j]=max(dp[i][j],dp[i-1][j-c[i]]+w[i]);//曲线救国
        }
    }
    printf("%d",dp[n][v]);
}
```

01背包优化

由于本层状态不受上一层以前的状态影响，故可直接使用**滚动数组**，但注意内层循环要**倒序遍历**，以便保证每个物品只使用过一次（即每次状态更新时使用到的前一个状态未被更新。

例如 $f[5]$ 由 $f[3]$ 转移得来，倒序遍历在第 i 层更新 $f[5]$ 时，$f[3]$ 仍处在第 $i-1$ 层的状态而未被更新，与二维时的状态转移保持一致。

状态表示： $dp[j]$ **只从前 i 个物品中选**，**总体积不超过** j 的选法的集合。

属性： MAX

状态计算：

- $dp[j] = max(dp[j], dp[j-c[i]] + w[i])$
- $dp[j-c[i]] + w[i]$ ：**选**第 i 个物品，由上一层空出体积 $c[i]$ 的状态+ $w[i]$ 得来

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

int n,v,dp[8000],c[1005],w[1005];
//n物品个数 v总体积 volume[i]第i件物品体积 worth[i]第i件物品价值
int main(){
    scanf("%d%d",&n,&v);
    for(int i=1;i<=n;i++){
        scanf("%d%d",&c[i],&w[i]);
    }
    for(int i=1;i<=n;i++){ // 枚举物品
        for(int j=v;j>=c[i];j--){//倒序遍历 枚举体积
            dp[j]=max(dp[j],dp[j-c[i]]+w[i]);//曲线救国
        }
    }
    printf("%d",dp[v]);
}
```

---

2. 完全背包

题目描述

有 N 种物品和一个容量是 V 的背包，**每种物品都有无限件**可用。

第 i 种物品的体积是 v_i ，价值是 w_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。
输出最大价值。

输入格式

第一行两个整数， N ， V ，用空格隔开，分别表示物品种数和背包容积。
接下来有 N 行，每行两个整数 v_i, w_i ，用空格隔开，分别表示第 i 种物品的体积和价值。

输出格式

输出一个整数，表示最大价值。

数据范围

0< N, V ≤1000
0< v_i, w_i ≤1000

输入样例

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

输出样例

```
10
```

代码

完全背包朴素

状态表示： $dp[i][j]$ **只从前 i 个物品中选，总体积不超过** j 的选法的集合。

属性: MAX

状态计算:

- $dp[i][j] = max(dp[i-1][j], dp[i-1][j-k*c[i]]+k*w[i])$
- $dp[i-1][j]$：**不选**第 i 种物品，直接由上一层转移
- $dp[i-1][j-k*c[i]]+k*w[i]$：**选**第 i 种物品k件，由上一层空出体积 $k*c[i]$ 的状态+ $k*w[i]$ 得来

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

int n,v,c[1007],w[1007],dp[5000][5000];
int main(){
    scanf("%d%d",&n,&v);
    for(int i=1;i<=n;i++)
        scanf("%d%d",&c[i],&w[i]);

    for(int i=1;i<=n;i++) // 枚举物品
        for(int j=0;j<=v;j++) // 枚举体积
            for(int k=0;k*c[i]<=j;k++) // 三重循环 枚举每种取用件数*c[i]不大于当前总体积j
                dp[i][j]=max(dp[i][j],dp[i-1][j-c[i]*k]+w[i]*k);

    printf("%d",dp[n][v]);
}
```

完全背包一级优化

将时间优化为二重。

状态表示： $dp[i][j]$ **只从前 i 个物品中选，总体积不超过** j 的选法的集合。
属性: MAX
状态计算:

- $dp[i][j] = max(dp[i-1][j], dp[i][j-c[i]] + w[i])$
- $dp[i-1][j]$：**不选**第 i 种物品，直接由上一层转移
- $dp[i][j-c[i]] + w[i]$：**选**第 i 种物品+1件，件数由体积的**正序遍历**不断累加

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

int n,v,c[1007],w[1007],dp[5000][5000];
int main(){
    scanf("%d%d",&n,&v);
    for(int i=1;i<=n;i++)
        scanf("%d%d",&c[i],&w[i]);

    for(int i=1;i<=n;i++) // 枚举物品
        for(int j=0;j<=v;j++){//参照01背包朴素 优化为二重循环 正序枚举体积
            dp[i][j]=dp[i-1][j];
            if(j>=c[i]) dp[i][j]=max(dp[i][j],dp[i][j-c[i]]+w[i]);
        }

    printf("%d",dp[n][v]);
}
```

完全背包二级优化

将空间优化为一维。

状态表示： $dp[j]$ **只从前 i 个物品中选**，**总体积不超过** j 的选法的集合。
属性: MAX
状态计算:

- $dp[j] = max(dp[j], dp[j-c[i]] + w[i])$
- $dp[j-c[i]] + w[i]$ ：**选**第 i 个物品+1件，件数由体积的**正序遍历**不断累加

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

int n,v,c[1007],w[1007],dp[5000];

int main(){
    scanf("%d%d",&n,&v);
    for(int i=1;i<=n;i++)
        scanf("%d%d",&c[i],&w[i]);

    for(int i=1;i<=n;i++) // 枚举物品
        for(int j=c[i];j<=v;j++) // 正序枚举体积
            dp[j]=max(dp[j],dp[j-c[i]]+w[i]);

    printf("%d",dp[v]);
}
```

- 注意！！**当空间优化为一维时，只有完全背包是正序遍历**

3. 多重背包

题目描述

有 N 种物品和一个容量是 V 的背包。

第 i 种物品最多有 si **件**，每件体积是 v_i，价值是 w_i。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。

输出最大价值。

输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 v_i, w_i, s_i，用空格隔开，分别表示第 i 种物品的体积、价值和数量。

输出格式

输出一个整数，表示最大价值。

数据范围

$$0 < N, V \leq 100$$
$$0 < v_i, w_i, s_i \leq 100$$

输入样例

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

输出样例

```
10
```

代码

多重背包朴素

状态表示： $dp[i][j]$ **只从前 i 个物品中选**，**总体积不超过** j 的选法的集合。
属性: MAX
状态计算:

- $dp[i][j] = max(dp[i-1][j], dp[i-1][j-k*c[i]] + k*w[i])$
- $dp[i-1][j]$： **不选**第 i 种物品，直接由上一层转移
- $dp[i-1][j-k*c[i]] + k*w[i]$： **选**第 i 种物品k件，由上一层空出体积 $k*c[i]$ 的状态+ $k*w[i]$ 得来

  与完全背包状态表示以及转移相同

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

int n,v,c[1500],w[1500],s[1500],dp[1500][1500];

int main(){
    scanf("%d%d",&n,&v);
    for(int i=1;i<=n;i++)
        scanf("%d%d%d",&c[i],&w[i],&s[i]);
    for(int i=1;i<=n;i++) // 枚举物品
        for(int j=0;j<=v;j++) // 枚举体积
```

```
            for(int k=0;k<=s[i]&&k*c[i]<=j;k++) // 枚举决策
                dp[i][j]=max(dp[i][j],dp[i-1][j-k*c[i]]+k*w[i]);
    printf("%d",dp[n][v]);
}
```

多重背包二进制优化

由于**每组物品的件数均不一样**，所以**不能使用完全背包的优化方法**（具体件数不可控），因此采用另一种思路——**二进制优化。**

将每一种物品由1.2.4.8.16.128...的件数打包，不足一组的零头重新打包，转化为01背包问题

```
//多重背包二进制拆分
#include<iostream>
#include<cstdio>
using namespace std;

int n,v,c[12500],w[12500],dp[25000];
int cnt;//划分成01背包后的总包数

int main(){
    scanf("%d%d",&n,&v);
    while(n--){
        int a,b,s;
        scanf("%d%d%d",&a,&b,&s);
        //倍增思想
        int k=1; //相当于base(每组件数): 1 2 4 8 16 32 64 128 256...据此打包
        while(k<=s){
            cnt++;
            c[cnt]=k*a;
            w[cnt]=k*b;
            s-=k;
            k*=2;
        }
        if(s>0){ //若拆完之后还有零头
            cnt++; //再分一个包
```

```
            c[cnt]=a*s;
            w[cnt]=b*s;
        }
    }
    //相当于将多重背包转化为01背包
    n=cnt;//01物品总个数
    for(int i=1;i<=n;i++)
        for(int j=v;j>=c[i];j--)//注意倒序遍历
            dp[j]=max(dp[j],dp[j-c[i]]+w[i]);

    printf("%d",dp[v]);
}
```

多重背包单调队列优化

其实一般很少会被逼到需要使用单调队列优化的背包了

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

using namespace std;

const int N = 20010;

int n, m;
int f[N], g[N], q[N];

int main()
{
    cin >> n >> m;
    for (int i = 0; i < n; i ++ )
    {
        int v, w, s;
```

```
            cin >> v >> w >> s;
            memcpy(g, f, sizeof f);
            for (int j = 0; j < v; j ++ )
            {
                int hh = 0, tt = -1;
                for (int k = j; k <= m; k += v)
                {
                    if (hh <= tt && q[hh] < k - s * v) hh ++ ;
                    while (hh <= tt && g[q[tt]] - (q[tt] - j) / v * w <= g[k] - (k - j) / v * w) tt -- ;
                    q[ ++ tt] = k;
                    f[k] = g[q[hh]] + (k - q[hh]) / v * w;
                }
            }
        }
        cout << f[m] << endl;

        return 0;
    }
    // 由于懒我先放的闫总的代码
```

---

4. 分组背包

题目描述

有 N 组物品和一个容量是 V 的背包。

每组物品有若干个，**同一组内的物品最多只能选一个。**

每件物品的体积是 v_{ij}，价值是 w_{ij}，其中 i 是组号，j 是组内编号。

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，且总价值最大。

输出最大价值。

输入格式

第一行有两个整数 N， V，用空格隔开，分别表示物品组数和背包容量。

接下来有 N 组数据：

- 每组数据第一行有一个整数 s_i ，表示第 i 个物品组的物品数量；
- 每组数据接下来有 s_i 行，每行有两个整数 v_{ij}, w_{ij} ，用空格隔开，分别表示第 i 个物品组的第 j 个物品的体积和价值；

输出格式

输出一个整数，表示最大价值。

数据范围

$$0 < N, V \leq 100$$
$$0 < Si \leq 100$$
$$0 < v_{ij}, w_{ij} \leq 100$$

输入样例

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

输出样例

```
8
```

代码

和01背包几乎一样，多一层循环一个组的物品，注意 `if(c[i][k]<=j)` 的最内层判断条件就行。

```
#include<iostream>
#include<cstdio>
using namespace std;
int n,v;
int c[200][200],w[200][200],s[200];
int dp[2000];
int main(){
    scanf("%d%d",&n,&v);//n组 v体积
    for(int i=1;i<=n;i++){
        scanf("%d",&s[i]); //s[i]第i组内物品个数
        for(int j=1;j<=s[i];j++)
            scanf("%d%d",&c[i][j],&w[i][j]);
    }
    for(int i=1;i<=n;i++)
        for(int j=v;j>=0;j--) //倒序遍历
            for(int k=1;k<=s[i];k++) //每组s[i]个
                if(c[i][k]<=j) //注意判断条件！！！！！！！！！！
                    dp[j]=max(dp[j],dp[j-c[i][k]]+w[i][k]); //选或不选

    printf("%d",dp[v]);
}
```

---

5. 混合背包

题目描述

有 N 种物品和一个容量是 V 的背包。

物品一共有三类:

- 第一类物品只能用1次（01背包）；
- 第二类物品可以用无限次（完全背包）；
- 第三类物品最多只能用 s_i 次（多重背包）；

  每种体积是 v_i ，价值是 w_i 。

  求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。

  输出最大价值。

输入格式

第一行两个整数， N ， V ，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 v_i, w_i, s_i ，用空格隔开，分别表示第 i 种物品的体积、价值和数量。

- $s_i = -1$ 表示第 i 种物品只能用1次;
- $s_i = 0$ 表示第 i 种物品可以用无限次;
- $s_i > 0$ 表示第 i 种物品可以使用 s_i 次;

输出格式

输出一个整数，表示最大价值。

数据范围

$0 < N, V \leq 1000$
$0 < v_i, w_i \leq 1000$
$-1 \leq s_i \leq 1000$

输入样例

```
4 5
1 2 -1
2 4 1
3 4 0
4 5 2
```

输出样例

```
8
```

代码

其实01**背包就相当于特殊的多重背包**，多重背包每件物品的个数为s，而01背包中每件物品的 $s=1$.
而对于完全背包，我们可以算出其上限：由于完全背包的物品是无限的，我们定其上限为 v 整除 tc （假定背包只装这一种物品 它最多也就只能装 v/tc 件 再多背包就装不下辽！）

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

int n,v,tc,tw,s,cnt=1;
int c[11000],w[11000],dp[50000];

int main(){
    scanf("%d%d",&n,&v);
    for(int i=1;i<=n;i++){
        scanf("%d%d%d",&tc,&tw,&s);
        if(s==0) s=v/tc; // 完全背包
        if(s==-1) s=1; // 01背包
```

```
            // 二进制优化
            int k=1;
            while(k<=s){
                cnt++;
                c[cnt]=k*tc;
                w[cnt]=k*tw;
                s-=k;
                k*=2;
            }
            if(s>0){
                cnt++;
                c[cnt]=s*tc;
                w[cnt]=s*tw;
            }
        }
        // 将01背包 完全背包 多重背包全部打包成cnt件
        n=cnt;// 接下来就是普通的01背包啦
        for(int i=1;i<=n;i++){
            for(int j=v;j>=c[i];j--){
                dp[j]=max(dp[j],dp[j-c[i]]+w[i]);
            }
        }
        printf("%d",dp[v]);
    }
```

---

6. 二维费用的背包

题目描述

有 N 件物品和一个**容量是** V 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 v_i ，重量是 m_i ，价值是 w_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

第一行两个整数，N, V, 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 v_i, m_i, w_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。

数据范围

$0 < N \leq 1000$
$0 < V, M \leq 100$
$0 < v_i, m_i \leq 100$
$0 < w_i \leq 1000$

输入样例

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

输出样例

```
8
```

代码

状态表示： $dp[j][k]$ **所有从前** i **个物品中选，总体积不超过** j**，总质量不超过** k 的选法的集合。

属性: MAX
状态计算:

- $dp[j][k] = max(dp[j][k], dp[j-tv][k-tm]+w)$

~~无非就是再加一维~~

```
#include<iostream>
#include<cstdio>
using namespace std;
int n,v,m,tv,tm,w;
int dp[5000][5000];// 无非就是再加一维
int main(){
    scanf("%d%d%d",&n,&v,&m);
    for(int i=1;i<=n;i++){
        scanf("%d%d%d",&tv,&tm,&w);
        for(int j=v;j>=tv;j--){
            for(int k=m;k>=tm;k--){// 无非就是再加一维
                dp[j][k]=max(dp[j][k],dp[j-tv][k-tm]+w);
            }
        }
    }
    printf("%d",dp[v][m]);
}
```

---

7. 有依赖的背包问题

题目描述

有 N 个物品和一个容量是 V 的背包。

物品之间具有依赖关系，且依赖关系组成一棵树的形状。如果选择一个物品，则必须选择它的父节点。

[图片上传失败...(image-2ed05f-1628507177264)]

如果选择物品5，则必须选择物品1和2。这是因为2是5的父节点，1是2的父节点。

每件物品的编号是i，体积是 v_i ，价值是 w_i ，依赖的父节点编号是 p_i 。物品的下标范围是 $1 \ldots N$ 。

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，且总价值最大。

输出最大价值。

输入格式

第一行有两个整数 N ， V ，用空格隔开，分别表示物品个数和背包容量。

接下来有 N 行数据，每行数据表示一个物品。

第 i 行有三个整数 v_i, w_i, p_i ，用空格隔开，分别表示物品的体积、价值和依赖的物品编号。

如果$p_i = -1$ ，表示根节点。 数据保证所有物品构成一棵树。

输出格式

输出一个整数，表示最大价值。

数据范围

$1 \le N, V \le 100$

$1 \le v_i, w_i \le 100$

父节点编号范围：

- 内部结点：$1 \le p_i \le N$；
- 根节点 $p_i = -1$；

输入样例

```
5 7
2 3 -1
2 2 1
3 5 1
4 7 2
3 6 2
```

输出样例

```
11
```

代码

状态表示： $dp[u][j]$ **所有从以u为根的子树中选，总体积不超过** j 的选法的集合。

属性： MAX

状态计算：

- $dp[k][j] = max(dp[k][j], dp[k][j-l] + dp[son][l])$
- $dp[k][j]$ 不选 son 子树
- $dp[k][j-l] + dp[son][l]$ 选了体积为 l 的 son 子树

其实很简单 就是**把线性的01背包简单变形为一棵树**

链式前向星+dfs

```
#include<iostream>
#include<cstdio>
#define maxn 200
#define maxm 400
```

```
using namespace std;
int n,v,cnt,w[maxn],c[maxn],dp[maxn][maxm];
int root;
int head[maxn],dis[maxn],vis[maxn];
struct node{
    int to,next;
}e[maxm];
// 链式前向星 或者叫 邻接表
//加边操作
void add(int x,int y){
    cnt++;
    e[cnt].to=y;
    e[cnt].next=head[x];
    head[x]=cnt;
}

void dfs(int k){ //当前节点k
    for(int i=head[k];i;i=e[i].next){// 枚举物品
        int son=e[i].to; // 记录子节点
        dfs(son);// 向下递归到最末子树 在回溯的过程中从最末更新dp值 直到回到root
        // 由于当前节点k必选 因此体积j需要将c[k]空出来 01背包倒序枚举体积
        for(int j=v-c[k];j>=0;j--){
            for(int l=0;l<=j;l++){// 枚举决策
                dp[k][j]=max(dp[k][j],dp[k][j-l]+dp[son][l]);
            }//             不选son子树    选son子树
        }
    }
    for(int i=v;i>=c[k];i--) dp[k][i]=dp[k][i-c[k]]+w[k];
    for(int i=0;i<c[k];i++) dp[k][i]=0;
}

int main(){
    scanf("%d%d",&n,&v);
    for(int i=1;i<=n;i++){
        int p;
        scanf("%d%d%d",&c[i],&w[i],&p);
        if(p==-1) root=i; // 根节点
        add(p,i); // 加边加边 由父节点指向子节点
    }
```

```
    dfs(root); // 从根节点开始搜
    printf("%d",dp[root][v]);
}
```

---

8. 背包问题求方案数

求方案数类问题，我们需要调整一下dp数组的含义。以下以01背包为例：

- $dp[i][j]$ 表示的是**从前 i 个物品中选，体积不超过 j 的选法**的集合。而此处为了便于方案数的计算，令 $dp[i][j]$ 表示为**从前 i 个物品中选，体积恰好为 j 的选法**的集合。注意dp数组含义改变后需要**初始化**，只有当体积恰好为零时，总价值才恰好为零，即 $dp[0]=0$，其他情况均出于未更新的状态，因此需要全部初始化为 $-inf$。
- **dp数组的值等于此状态下的最大价值**，另外我们还需要一个数组 $g[i][j]$，表示**此种状态下取最大值**（即取 $dp[i][j]$）的**方案数**。 最后我们只需要遍历一下 $dp[n][j]$ 得到最大价值（最优方案并不一定会把背包装满 因此需要遍历），再将价值=最大价值的所有对应 $g[i][j]$ 加起来，即为最优方案总数。

01背包求最优方案数

题目描述

有 N 件物品和一个容量是 V 的背包。每件物品只能使用一次。
第 i 件物品的体积是 v_i，价值是 w_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**最优选法的方案数**。注意答案可能很大，请输出答案**模** 109+7 的结果。

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 v_i, w_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

输出一个整数，表示 方案数 模 $109+7$ 的结果。

数据范围

$0 < N, V \leq 1000$
$0 < v_i, w_i \leq 1000$

输入样例

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

输出样例

```
2
```

代码

此处dp数组和g数组均可优化成一维的。

01背包在选择时有**两种决策**，自然，dp数组中存的是两种决策中价值最大的那个。此时我们只需要比较两种决策的价值大小，有**三种情况**：

- 若 $dp[j] > dp[j-c[i]] + w[i]$ 此时采用前者 则 $g[j] = g[j]$（由 i - 1 层转移来）
- 若 $dp[j] < dp[j-c[i]] + w[i]$ 此时采用后者 则 $g[j] = g[j-c[i]]$

- 若 $dp[j] < dp[j-c[i]] + w[i]$ 此时两种都选 则 $g[j] = g[j] + g[j-c[i]]$

  注意**初始化**！！ 当背包体积为零，什么都不选，也是一种决策 即 $g[0] = 1$

```
#include<iostream>
#include<cstdio>
#define inf 0x3f3f3f3f
const int mod=1e9+7;
using namespace std;

int n,v,res,ans;
int c[1010],w[1010],dp[1010],g[3000];

int main(){
    scanf("%d%d",&n,&v);
    for(int i=1;i<=n;i++)
        scanf("%d%d",&c[i],&w[i]);
    // 初始化
    for(int i=1;i<=n;i++) dp[i]=-inf;
    g[0]=1; dp[0]=0;

    for(int i=1;i<=n;i++){
        for(int j=v;j>=c[i];j--){ //01背包倒序遍历
            int cnt=0,maxn; // 该状态下的方案数 和 最大价值
            maxn=max(dp[j],dp[j-c[i]]+w[i]);
            if(maxn==dp[j])
                cnt+=g[j];
            if(maxn==dp[j-c[i]]+w[i])
                cnt+=g[j-c[i]];
            g[j]=cnt%mod;
            dp[j]=maxn;
        }
    }
    // 求出最大价值
    for(int i=1;i<=v;i++)
        res=max(res,dp[i]);
    // 最大价值的方案数求和
    for(int i=1;i<=v;i++)
```

```
            if(res==dp[i]) ans+=g[i];

        printf("%d",ans);
    }
```

---

9. 背包问题求具体方案

背包问题求具体方案的思路基本相同，重点在于**判断每件物品到底选了还是没选**，好像是废话，类似于最短路求最短路径。且由于要输出方案，所以我们**不能使用空间优化**后的转移方程。另外，要求输出字典序最小的方案时还须考虑**选择顺序。**

01背包求字典序最小的最优方案

题目描述

有 N 件物品和一个容量是 V 的背包。每件物品只能使用一次。
第 i 件物品的体积是 v_i ，价值是 w_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 v_i, w_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

$0 < N, V \leq 1000$
$0 < v_i, w_i \leq 1000$

输入样例

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

输出样例

```
1 4
```

思路

我们要考虑两件事：

- 字典序最小

  由于要输出字典序最小的方案，我们应使选取的物品尽可能靠前。因此在枚举时，**物品应从大到小枚举**，编号小的物品放在最后更新，相当于若遇到等价物品，会将后枚举到的小编号物品代替等价大编号物品，这样一来，所得的dp状态也是最后更新的值，实现了字典序最小。

  最外层循环：`for (int i = n; i >= 1; i--)`

  自然，我们将选取物品的顺序改变后，相应的转移方程也会随之适当调整：

$$dp[i][j] = max(dp[i+1][j], dp[i+1][j-c[i]] + w[i])$$

- 选了哪些物品

  由状态转移方程可得:

  当 $dp[i][j]=dp[i+1][j]$ 时，说明第 i 件物品没选

  当 $dp[i][j]=dp[i+1][j-c[i]]+w[i])$ 时，说明第 i 件物品选了

  此时我们只需要判断 $dp[i][j]$ 是否等于 $dp[i+1][j-c[i]]+w[i]$ 即可。

代码

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

int n,v,c[1100],w[1100],dp[1100][1100];

int main(){
    scanf("%d%d",&n,&v);
    for(int i=1;i<=n;i++)
        scanf("%d%d",&c[i],&w[i]);

    for(int i=n;i>=1;i--){ //由大到小枚举物品 保证选取方案字典序最小
        for(int j=0;j<=v;j++){
            dp[i][j]=dp[i+1][j];
            if(j>=c[i])   dp[i][j]=max(dp[i][j],dp[i+1][j-c[i]]+w[i]);
            // 这两行代码等价于我们刚刚写的状态转移方程
        }
    }
    int j=v; //由小到大枚举体积
    for(int i=1;i<=n;i++){
        if(j>=c[i]&&dp[i][j]==dp[i+1][j-c[i]]+w[i]){
        //若选了第i件物品则输出
            printf("%d ",i);
            j-=c[i];
        }
    }
```

```
        }
```

最后编辑于： 2021.08.09 19:09:27