动态规划-背包模型

一、AcWing 6. 多重背包问题 III

【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s_i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】

$0 < N \leq 1000$

$0 < V \leq 20000$

$0 < v_i, w_i, s_i \leq 20000$

【提示】

本题考查多重背包的单调队列优化方法。

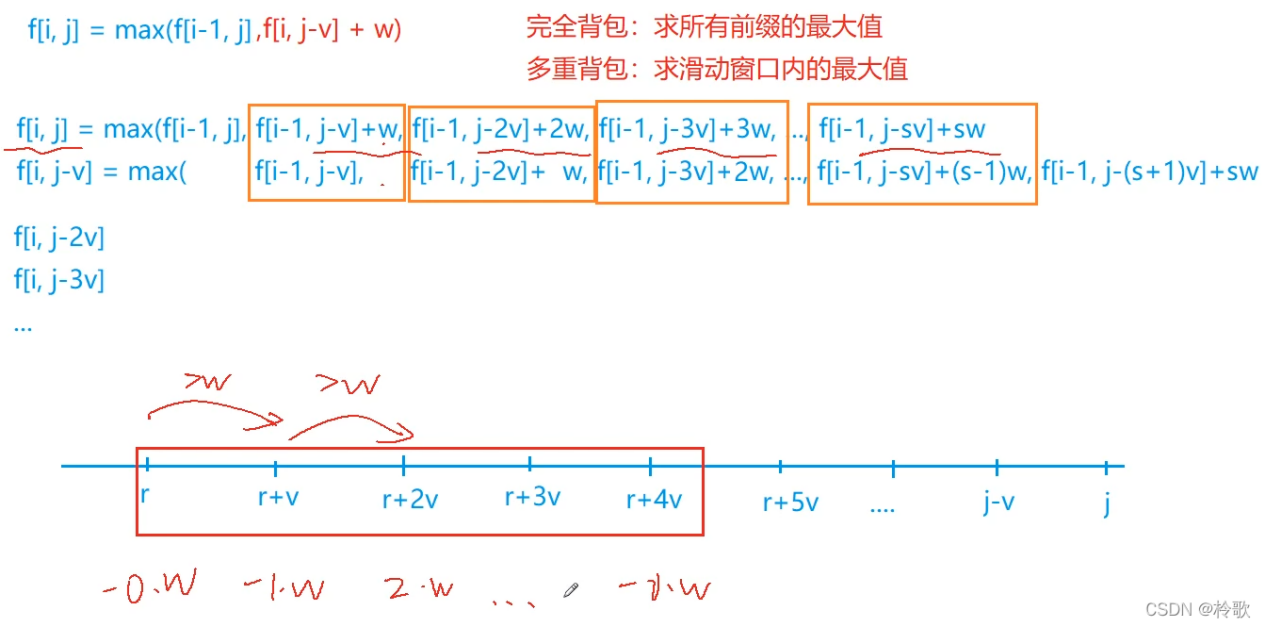
【输入样例】

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

【输出样例】

```
10
```

【分析】



【代码】

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

const int N = 20010;
int f[N], g[N], Q[N];//g表示第i-1层的f的状态
int n, m;

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);//备份一下上一层的状态
        for (int j = 0; j < v; j++)//枚举余数r
        {
            int hh = 0, tt = -1;//初始化单调队列

            for (int k = j; k <= m; k += v)
```

```
            {
                if (hh <= tt && Q[hh] < k - s * v) hh++;//队头已经滑出窗口了
                if (hh <= tt) f[k] = max(f[k], g[Q[hh]] + (k - Q[hh]) / v * w);
                while (hh <= tt && g[Q[tt]] - (Q[tt] - j) / v * w <= g[k] - (k - j) / v * w) tt--;
                Q[++tt] = k;
            }
        }
    }
    cout << f[m] << endl;
    return 0;
}
```

二、AcWing 423. 采药

【题目描述】

辰辰是个天资聪颖的孩子，他的梦想是成为世界上最伟大的医师。

为此，他想拜附近最有威望的医师为师。

医师为了判断他的资质，给他出了一个难题。

医师把他带到一个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

如果你是辰辰，你能完成这个任务吗？

【输入格式】

输入文件的第一行有两个整数T和M，用一个空格隔开，T代表总共能够用来采药的时间，M代表山洞里的草药的数目。

接下来的M行每行包括两个在$1 \sim 100$（包括1和100）的整数，分别表示采摘某株草药的时间和这株草药的价值。

【输出格式】

输出文件包括一行，这一行只包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。

【数据范围】

$1 \leq T \leq 1000$

$1 \leq M \leq 100$

【输入样例】

```
70 3
71 100
69 1
1 2
```

【输出样例】

```
3
```

【分析】

很裸的01背包问题，直接看代码~

【代码】

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

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

int main()
{
    cin >> m >> n;
    for (int i = 0; i < n; i++)
    {
        int v, w;
        cin >> v >> w;
        for (int j = m; j >= v; j--) f[j] = max(f[j], f[j - v] + w);
    }
    cout << f[m] << endl;
    return 0;

}
```

三、AcWing 1024. 装箱问题

【题目描述】

有一个箱子容量为V，同时有n个物品，每个物品有一个体积（正整数）。

要求n个物品中，任取若干个装入箱内，使箱子的剩余空间为最小。

【输入格式】

第一行是一个整数V，表示箱子容量。

第二行是一个整数n，表示物品数。

接下来n行，每行一个正整数（不超过10000），分别表示这n个物品的各自体积。

【输出格式】

一个整数，表示箱子剩余空间。

【数据范围】

$0 < V \leq 20000$

$0 < n \leq 30$

【输入样例】

```
24
6
8
3
12
7
9
7
```

【输出样例】

```
0
```

【分析】

要使箱子的剩余空间最小，即所装的物品体积之和最大。因此物品的体积即为价值，求出可装入的最大价值，然后将总体积减去最大价值即为结果。

【代码】

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

const int N = 20010;
int f[N];
int n, m;

int main()
{
    cin >> m >> n;
    for (int i = 0; i < n; i++)
    {
        int v;
        cin >> v;
        for (int j = m; j >= v; j--) f[j] = max(f[j], f[j - v] + v);
    }
    cout << m - f[m] << endl;
    return 0;
}
```

四、AcWing 8. 二维费用的背包问题

【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】

第一行三个整数，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
```

【分析】

状态表示：$f[i][j][k]$表示所有只从前i个物品中选，并且总体积不超过j，总重量不超过k的选法中的最大价值。

状态计算：

- 不选第i个物品：$f[i][j][k] = f[i-1][j][k]$
- 选第i个物品：$f[i][j][k] = f[i-1][j-v_i][k-m_i] + w_i$

【代码】

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

const int N = 110;
int f[N][N];
```

```cpp
int n, m1, m2;

int main()
{
    cin >> n >> m1 >> m2;
    for (int i = 0; i < n; i++)
    {
        int v, m, w;
        cin >> v >> m >> w;
        for (int j = m1; j >= v; j--)
            for (int k = m2; k >= m; k--)
                f[j][k] = max(f[j][k], f[j - v][k - m] + w);
    }
    cout << f[m1][m2] << endl;
    return 0;
}
```

五、AcWing 1022. 宠物小精灵之收服

【题目描述】

宠物小精灵是一部讲述小智和他的搭档皮卡丘一起冒险的故事。

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，野生的小精灵并不那么容易被收服。

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？

【输入格式】

输入数据的第一行包含三个整数：N, M, K，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。

【输出格式】

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。

【数据范围】

$0 < N \leq 1000$

$0 < M \leq 500$

$0 < K \leq 100$

【输入样例1】

```
10 100 5
7 10
2 40
2 50
1 20
4 20
```

【输出样例1】

```
3 30
```

【输入样例2】

```
10 100 5
8 110
12 10
20 10
5 200
1 110
```

【输出样例2】

```
0 100
```

【分析】

这是一题二维费用的背包问题，首先需要对题目进行阅读理解，其中需要注意的点如下：

- 皮卡丘的体力不能全部用完，即最多只能使用$M-1$的体力值；
- 需要求的是在捕捉精灵数量最多的情况下体力剩余的最大值，也就是体力消耗的最小值。

状态表示：$f[i][j][k]$表示所有只从前i只精灵中选，且花费精灵球数量不超过j，花费体力不超过k的选法的最大价值。

状态计算：$f[i][j][k]=max(f[i-1][j][k],f[i-1][j-v1[i]][k-v2[i]]+1)$。

$f[K][N][M-1]$即为能捕捉的最大精灵数，然后我们从大到小枚举消耗的体力值，找到满足结果等于最大价值的最小消耗体力，则初始体力值减去最小消耗体力即为剩余的最大体力。

【代码】

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

const int N = 1010, M = 510;
int f[N][M];
int n, m1, m2;

int main()
{

    cin >> m1 >> m2 >> n;
```

```
    for (int i = 0; i < n; i++)
    {
        int v1, v2;
        cin >> v1 >> v2;
        for (int j = m1; j >= v1; j--)
            for (int k = m2 - 1; k >= v2; k--)//由于体力不能全用完因此要从最大值减一开始枚举
                f[j][k] = max(f[j][k], f[j - v1][k - v2] + 1);
    }
    int idx = m2 - 1;
    while (idx > 0 && f[m1][idx - 1] == f[m1][m2 - 1]) idx--;//idx为捕捉数量等于最大值的最小体力
    cout << f[m1][m2 - 1] << ' ' << m2 - idx << endl;
    return 0;
}
```

六、AcWing 1020. 潜水员

【题目描述】

潜水员为了潜水要使用特殊的装备。

他有一个带2种气体的气缸：一个为氧气，一个为氮气。

让潜水员下潜的深度需要各种数量的氧和氮。

潜水员有一定数量的气缸。

每个气缸都有重量和气体容量。

潜水员为了完成他的工作需要特定数量的氧和氮。

他完成工作所需气缸的总重的最低限度的是多少？

例如：潜水员有5个气缸。每行三个数字为：氧，氮的量和气缸的重量：

```
3 36 120
10 25 129
5 50 250
1 45 130
4 20 119
```

如果潜水员需要5升的氧和60升的氮则总重最小为249（1,2或者4,5号气缸）。

你的任务就是计算潜水员为了完成他的工作需要的气缸的重量的最低值。

【输入格式】

第一行有2个整数m,n。它们表示氧，氮各自需要的量。

第二行为整数k表示气缸的个数。

此后的k行，每行3个整数a_i,b_i,c_i。分别表示第i个气缸里的氧和氮的容量以及气缸的重量。

【输出格式】

仅一行包含一个整数，为潜水员完成工作所需的气缸的重量总和的最低值。

【数据范围】

$1 \leq m \leq 21$

$1 \leq n \leq 79$

$1 \leq k \leq 1000$

$1 \leq a_i \leq 21$

$1 \leq b_i \leq 79$

$1 \leq c_i \leq 800$

【输入样例】

```
5 60
5
3 36 120
10 25 129
5 50 250
1 45 130
4 20 119
```

【输出样例】

```
249
```

【分析】

状态表示：$f[i][j][k]$表示所有从前i个物品中选，且氧气含量至少是j，氮气含量至少是k的所有选法中的最小重量。

状态计算：

- 不选第i个物品：$f[i][j][k] = f[i-1][j][k]$

- 选第i个物品：$f[i][j][k] = f[i-1][j-a_i][k-b_i] + c_i$

初始化：一个物品都不选，氧气含量至少是0，氮气含量至少是0的重量为0，即$f[0][0][0] = 0$。对于其它的$f[i][j][k]$均初始化为正无穷。

注意：由于状态表示为**至少**，因此j, k如果为负数也应该转移，例如至少为-1这种说法也是合法的，这就区别于**恰好**的状态表示，如果是**恰好**，那么我们循环j, k时需要满足$j \geq a_i$且$k \geq b_i$，而如果是**至少**，那么只需要满足$j \geq 0$且$k \geq 0$即可，如果$j - a_i$为负数，则我们将其看成0即可。

扩展：如果状态表示为**至多**，那么初始化时所有的$f[i][j][k] = 0$，循环时需要满足$j \geq a_i, k \geq b_i$。

【代码】

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

const int N = 25, M = 80;
int f[N][M];
int n, m1, m2;

int main()
{
    cin >> m1 >> m2 >> n;
    memset(f, 0x3f, sizeof f);
    f[0][0] = 0;
    for (int i = 0; i < n; i++)
    {
        int a, b, c;
        cin >> a >> b >> c;
        for (int j = m1; j >= 0; j--)
            for (int k = m2; k >= 0; k--)
                f[j][k] = min(f[j][k], f[max(j - a, 0)][max(k - b, 0)] +
c);
    }
    cout << f[m1][m2] << endl;
    return 0;
}
```

七、AcWing 278. 数字组合

【题目描述】

给定N个正整数$A_1, A_2, \ldots, A_N$，从中选出若干个数，使它们的和为M，求有多少种选择方案。

【输入格式】

第一行包含两个整数N和M。

第二行包含N个整数，表示$A_1, A_2, \ldots, A_N$。

【输出格式】

包含一个整数，表示可选方案数。

【数据范围】

$1 \leq N \leq 100$

$1 \leq M \leq 10000$

$1 \leq A_i \leq 1000$

答案保证在int范围内。

【输入样例】

```
4 4
1 1 2 2
```

【输出样例】

```
3
```

【分析】

M看成是背包容量，把每个数看成是一个物品，A_i为每个物品的体积，那么目标就是求出总体积恰好是M的方案数。

状态表示：$f[i][j]$表示所有只从前i个物品中选，且总体积恰好是j的方案的数量。

状态计算：

- 不选第i个物品：$f[i][j] = f[i-1][j]$
- 选第i个物品：$f[i][j] = f[i-1][j-A_i]$

我们需要求的是方案数量之和，因此$f[i][j] = f[i-1][j] + f[i-1][j-A_i]$。

初始化：$f[0][0]=1$，即一个数都不选，总和恰好为0的方案数有1种。$f[0][i]=0, 1 \le i \le M$，即一个数都不选而总和大于0的方案都是不存在的。

【代码】

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

const int N = 10010;
int f[N];
int n, m;

int main()
{
    cin >> n >> m;
    f[0] = 1;
    for (int i = 0; i < n; i++)
    {
        int a;
        cin >> a;
        for (int j = m; j >= a; j--)
            f[j] += f[j - a];
    }
    cout << f[m] << endl;
    return 0;
}
```

八、AcWing 1019. 庆功会

【题目描述】

为了庆贺班级在校运动会上取得全校第一名成绩，班主任决定开一场庆功会，为此拨款购买奖品犒劳运动员。

期望拨款金额能购买最大价值的奖品，可以补充他们的精力和体力。

【输入格式】

第一行二个数n, m，其中n代表希望购买的奖品的种数，m表示拨款金额。

接下来n行，每行3个数，v, w, s，分别表示第i种奖品的价格、价值（价格与价值是不同的概念）和能购买的最大数量（买0件到s件均可）。

【输出格式】

一行：一个数，表示此次购买能获得的最大的价值（注意！不是价格）。

【数据范围】

$n \leq 500, m \leq 6000$

$v \leq 100, w \leq 1000, s \leq 10$

【输入样例】

```
5 1000
80 20 4
40 50 9
30 50 7
40 30 6
20 20 1
```

【输出样例】

```
1040
```

【分析】

本题数据量不大，使用朴素版的多重背包解法即可~

【代码】

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

const int N = 6010;
int f[N];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 0; i < n; i++)

    {
```

```
        int v, w, s;
        cin >> v >> w >> s;
        for (int j = m; j >= 0; j--)
            for (int k = 0; k <= s && k * v <= j; k++)
                f[j] = max(f[j], f[j - k * v] + k * w);
    }
    cout << f[m] << endl;
    return 0;
}
```

九、AcWing 1023. 买书

【题目描述】

小明手里有n元钱全部用来买书（注意需要将钱全部花完），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）

【输入格式】

一个整数n，代表总共钱数。

【输出格式】

一个整数，代表选择方案种数。

【数据范围】

$0 \leq n \leq 1000$

【输入样例1】

```
20
```

【输出样例1】

```
2
```

【输入样例2】

```
15
```

【输出样例2】

```
0
```

【输入样例3】

```
0
```

【输出样例3】

```
1
```

【分析】

状态表示：$f[i][j]$表示所有只从前i个物品中选，且花费恰好为j的所有方案的数量。

状态计算：

$$f[i][j] = f[i-1][j] + f[i-1][j-v] + f[i-1][j-2v] + \cdots + f[i-1][j-sv]$$

$$f[i][j-v] = f[i-1][j-v] + f[i-1][j-2v] + \cdots + f[i-1][j-sv]$$

因此$f[i][j] = f[i-1][j] + f[i][j-v]$

【代码】

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

const int N = 1010;
int f[N], v[4] = { 10, 20, 50, 100 };
int m;

int main()
{
    cin >> m;
    f[0] = 1;
    for (int i = 0; i < 4; i++)
        for (int j = v[i]; j <= m; j++)
            f[j] += f[j - v[i]];
    cout << f[m] << endl;
    return 0;
}
```

十、AcWing 12. 背包问题求具体方案

【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$1 \sim N$。

【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$1 \sim N$。

【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
```

【分析】

题目要求输出字典序最小的解，假设存在一个包含第1个物品的最优解，为了确保字典序最小那么我们必然要选第一个。那么问题就转化成从$2\sim N$这些物品中找到最优解。之前的$f[i][j]$记录的都是只考虑前i个物品且背包总容量为j的最优解，那么我们现在将$f[i][j]$定义为只考虑第i个元素到最后一个元素且背包总容量为j的最优解。接下来考虑状态转移：

$$f[i][j]=max(f[i+1][j],f[i+1][j-v[i]]+w[i])$$

两种情况：第一种是不选第i个物品，那么最优解等同于只考虑第$i+1$个物品到最后一个物品且背包总容量为j的最优解；第二种是选了第i个物品，那么最优解等于当前物品的价值$w[i]$加上考虑第$i+1$个物品到最后一个物品且背包总容量为$j-v[i]$的最优解。

计算完状态表示后，考虑如何的到最小字典序的解。首先$f[1][m]$肯定是最大价值，那么我们便开始考虑能否选取第1个物品。

- 如果$f[1][m]=f[2][m-v[1]]+w[1]\neq f[2][m]$，说明选取了第$1$个物品才能得到最优解；
- 如果$f[1][m]=f[2][m]\neq f[2][m-v[1]]+w[1]$，说明不选取第$1$个物品才能得到最优解；
- 如果$f[1][m]=f[2][m]=f[2][m-v[1]]+w[1]$，说明选和不选都可以得到最优解，但是考虑字典序最小，我们必须选取该物品。

因此，只要满足$m\geq v[i]$且$f[i][m]=f[i+1][m-v[i]]+w[i]$，那么我们就选第$i$个物品，然后$m-=v[i]$。

【代码】

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

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

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> v[i] >> w[i];
    for (int i = n; i >= 1; i--)
        for (int j = 0; j <= m; j++)

        {
```

```
            f[i][j] = f[i + 1][j];
            if (j >= v[i]) f[i][j] = max(f[i][j], f[i + 1][j - v[i]] +
w[i]);
        }
    for (int i = 1; i <= n; i++)
        if (m >= v[i] && f[i][m] == f[i + 1][m - v[i]] + w[i])
        {
            cout << i << ' ';
            m -= v[i];
        }
    return 0;
}
```

十一、AcWing 1013. 机器分配

【题目描述】

总公司拥有M台相同的高效设备，准备分给下属的N个分公司。

各分公司若获得这些设备，可以为国家提供一定的盈利。盈利与分配的设备数量有关。

问：如何分配这M台设备才能使国家得到的盈利最大？

求出最大盈利值。

分配原则：每个公司有权获得任意数目的设备，但总台数不超过设备数M。

【输入格式】

第一行有两个数，第一个数是分公司数N，第二个数是设备台数M；

接下来是一个$N \times M$的矩阵，矩阵中的第i行第j列的整数表示第i个公司分配j台机器时的盈利。

【输出格式】

第一行输出最大盈利值；

接下N行，每行有2个数，即分公司编号和该分公司获得设备台数。

答案不唯一，输出任意合法方案即可。

【数据范围】

$1 \leq N \leq 10$

$1 \leq M \leq 15$

【输入样例】

```
3 3
30 40 50
20 30 50
20 25 30
```

【输出样例】

```
70
1 1
2 1
3 1
```

【分析】

将每个公司看成是一个分组，$w[i][k]$表示给第i个公司分配k台设备的收益。每个分组分配1台机器看成选择了体积为1的物品，分配2台看成选择了体积为2的物品，以此类推，每个分组只能选择一个物品。因此状态转移方程为$f[i][j]=max(f[i-1][j],f[i-1][j-k]+w[i][k])$。

【代码】

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

const int N = 15, M = 20;
int f[N][M], w[N][M];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> w[i][j];
    for (int i = 1; i <= n; i++)
        for (int j = 0; j <= m; j++)

            for (int k = 0; k <= j; k++)
```

```
                f[i][j] = max(f[i][j], f[i - 1][j - k] + w[i][k]);
    cout << f[n][m] << endl;
    for (int i = n; i; i--)//有Special Judge因此可以倒着输出
        for (int k = 0; k <= m; k++)
            if (f[i][m] == f[i - 1][m - k] + w[i][k])
            {
                cout << i << ' ' << k << endl;
                m -= k;
                break;
            }
    return 0;
}
```

十二、AcWing 426. 开心的金明

【题目描述】

金明今天很开心，家里购置的新房就要领钥匙了，新房里有一间他自己专用的很宽敞的房间。

更让他高兴的是，妈妈昨天对他说：“你的房间需要购买哪些物品，怎么布置，你说了算，只要不超过N元钱就行”。

今天一早金明就开始做预算，但是他想买的东西太多了，肯定会超过妈妈限定的N元。

于是，他把每件物品规定了一个重要度，分为5等：用整数$1 \sim 5$表示，第5等最重要。

他还从因特网上查到了每件物品的价格（都是整数元）。

他希望在不超过N元（可以等于N元）的前提下，使每件物品的价格与重要度的乘积的总和最大。

设第j件物品的价格为$v[j]$，重要度为$w[j]$，共选中了k件物品，编号依次为$j_1, j_2, \ldots, j_k$，则所求的总和为：

$$v[j_1] * w[j_1] + v[j_2] * w[j_2] + \cdots + v[j_k] * w[j_k]$$

请你帮助金明设计一个满足要求的购物单。

【输入格式】

输入文件的第1行，为两个正整数N, M，用一个空格隔开。（其中N表示总钱数，M为希望购买物品的个数）

从第2行到第$M+1$行，第j行给出了编号为$j-1$的物品的基本数据，每行有2个非负整数v和p。（其中v表示该物品的价格，p表示该物品的重要度）

【输出格式】

输出文件只有一个正整数，为不超过总钱数的物品的价格与重要度乘积的总和的最大值（数据保证结果不超过10^8）。

【数据范围】

$1 \leq N < 30000$

$1 \leq M < 25$

$0 \leq v \leq 10000$

$1 \leq p \leq 5$

【输入样例】

```
1000 5
800 2
400 5
300 5
400 3
200 2
```

【输出样例】

```
3900
```

【分析】

01背包裸题，直接看代码~

【代码】

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

const int N = 30010;
int f[N];
int n, m;


int main()
```

```
{
    cin >> m >> n;
    for (int i = 0; i < n; i++)
    {
        int v, w;
        cin >> v >> w;
        for (int j = m; j >= v; j--) f[j] = max(f[j], f[j - v] + v * w);
    }
    cout << f[m] << endl;
    return 0;
}
```

十三、AcWing 487. 金明的预算方案

【题目描述】

金明今天很开心，家里购置的新房就要领钥匙了，新房里有一间金明自己专用的很宽敞的房间。

更让他高兴的是，妈妈昨天对他说：“你的房间需要购买哪些物品，怎么布置，你说了算，只要不超过N元钱就行”。

今天一早，金明就开始做预算了，他把想买的物品分为两类：主件与附件，附件是从属于某个主件的，下表就是一些主件与附件的例子：

主件	附件
电脑	打印机，扫描仪
书柜	图书
书桌	台灯，文具
工作椅	无

如果要买归类为附件的物品，必须先买该附件所属的主件。

每个主件可以有0个、1个或2个附件。

附件不再有从属于自己的附件。

金明想买的东西很多，肯定会超过妈妈限定的N元。

于是，他把每件物品规定了一个重要度，分为5等：用整数$1 \sim 5$表示，第5等最重要。

他还从因特网上查到了每件物品的价格（都是10元的整数倍）。

他希望在不超过N元（可以等于N元）的前提下，使每件物品的价格与重要度的乘积的总和最大。

设第j件物品的价格为$v[j]$，重要度为$w[j]$，共选中了k件物品，编号依次为$j_1, j_2, \ldots, j_k$，则所求的总和为：

$$v[j_1] * w[j_1] + v[j_2] * w[j_2] + \cdots + v[j_k] * w[j_k]$$

请你帮助金明设计一个满足要求的购物单。

【输入格式】

输入文件的第1行，为两个正整数，用一个空格隔开：N, M，其中N表示总钱数，M为希望购买物品的个数。

从第2行到第$M+1$行，第j行给出了编号为$j-1$的物品的基本数据，每行有3个非负整数v, p, q，其中v表示该物品的价格，p表示该物品的重要度（$1 \sim 5$），q表示该物品是主件还是附件。

如果$q=0$，表示该物品为主件，如果$q>0$，表示该物品为附件，q是其所属主件的编号。

【输出格式】

输出文件只有一个正整数，为不超过总钱数的物品的价格与重要度乘积的总和的最大值(<200000)。

【数据范围】

$N<32000, M<60, v<10000$

【输入样例】

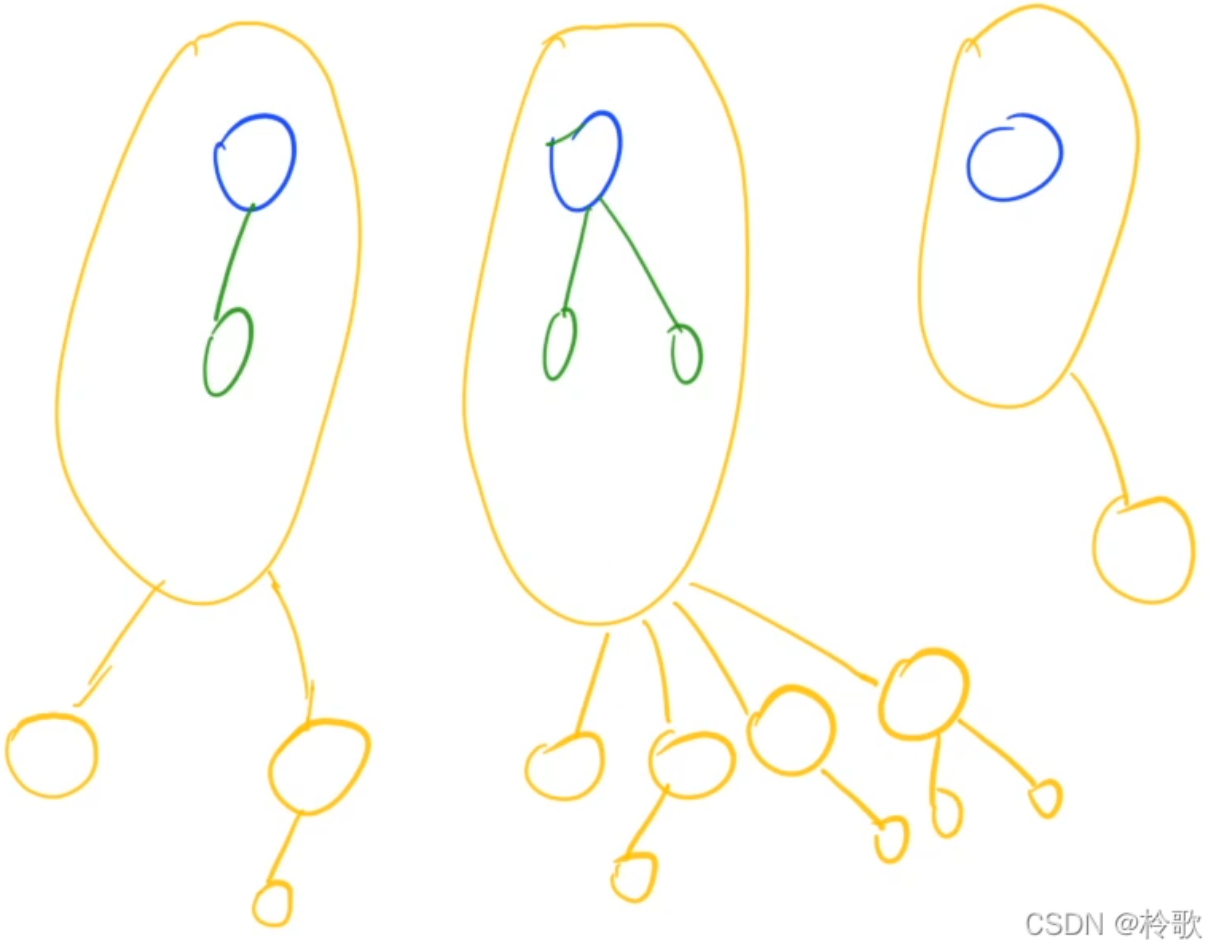
```
1000 5
800 2 0
400 5 1
300 5 1
400 3 0
500 2 0
```

【输出样例】

```
2200
```

【分析】

如下图所示，将每个组件及其所带的附件看成是一个分组，则对于每个分组，首先必须得把主件选上，然后再枚举附件的所有不同的选法，假设有k个附件，那么就有2^k种不同的选法，因此可以使用二进制来枚举所有状态。



【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <vector>
#define X first
#define Y second
using namespace std;

typedef pair<int, int> PII;
const int N = 70, M = 32010;
int f[M];
PII mas[N];//主件
vector<PII> ath[N];//附件
int n, m;

```

```cpp
int main()
{
    cin >> m >> n;
    for (int i = 1; i <= n; i++)
    {
        int v, w, q;
        cin >> v >> w >> q;
        if (!q) mas[i] = { v, v * w };
        else ath[q].push_back({ v, v * w });
    }
    for (int i = 1; i <= n; i++)
        for (int j = m; j >= mas[i].X; j--)
            for (int st = 0; st < 1 << ath[i].size(); st++)//用二进制枚举所有附件的选择方式
            {
                int v = mas[i].X, w = mas[i].Y;//主件必须得选上
                for (int k = 0; k < ath[i].size(); k++)
                    if (st >> k & 1) v += ath[i][k].X, w += ath[i][k].Y;
                if (j >= v) f[j] = max(f[j], f[j - v] + w);
            }
    cout << f[m] << endl;
    return 0;
}
```

十四、AcWing 1021. 货币系统

【题目描述】

给你一个n种面值的货币系统，求组成面值为m的货币有多少种方案。

【输入格式】

第一行，包含两个整数n和m。

接下来n行，每行包含一个整数，表示一种货币的面值。

【输出格式】

共一行，包含一个整数，表示方案数。

【数据范围】

$n \leq 15, m \leq 3000$

【输入样例】

```
3 10
1
2
5
```

【输出样例】

```
10
```

【分析】

完全背包求方案数裸题，直接看代码~

【代码】

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

typedef long long LL;
const int N = 3010;
LL f[N];
int n, m;

int main()
{
    cin >> n >> m;
    f[0] = 1;
    for (int i = 0; i < n; i++)
    {
        int v;
        cin >> v;
        for (int j = v; j <= m; j++) f[j] += f[j - v];
    }
    cout << f[m] << endl;
    return 0;
}
```

十五、AcWing 532. 货币系统

【题目描述】

在网友的国度中共有n种不同面额的货币，第i种货币的面额为$a[i]$，你可以假设每一种货币都有无穷多张。

为了方便，我们把货币种数为n、面额数组为$a[1 \sim n]$的货币系统记作(n, a)。

在一个完善的货币系统中，每一个非负整数的金额x都应该可以被表示出，即对每一个非负整数x，都存在n个非负整数$t[i]$满足$a[i] \times t[i]$的和为x。

然而，在网友的国度中，货币系统可能是不完善的，即可能存在金额x不能被该货币系统表示出。

例如在货币系统$n = 3, a = [2, 5, 9]$中，金额$1, 3$就无法被表示出来。

两个货币系统(n, a)和(m, b)是等价的，当且仅当对于任意非负整数x，它要么均可以被两个货币系统表示出，要么不能被其中任何一个表示出。

现在网友们打算简化一下货币系统。

他们希望找到一个货币系统(m, b)，满足(m, b)与原来的货币系统(n, a)等价，且m尽可能的小。

他们希望你来协助完成这个艰巨的任务：找到最小的m。

【输入格式】

输入文件的第一行包含一个整数T，表示数据的组数。

接下来按照如下格式分别给出T组数据。

每组数据的第一行包含一个正整数n。

接下来一行包含n个由空格隔开的正整数$a[i]$。

【输出格式】

输出文件共有T行，对于每组数据，输出一行一个正整数，表示所有与(n, a)等价的货币系统(m, b)中，最小的m。

【数据范围】

$1 \leq n \leq 100$

$1 \leq a[i] \leq 25000$

$1 \leq T \leq 20$

【输入样例】

```
2
4
3 19 10 6
5
11 29 13 19 17
```

【输出样例】

```
2
5
```

【分析】

对题目进行分析，能得到以下三个性质：

1. $a_1, a_2, \ldots, a_n$一定都能被$b_1, b_2, \ldots, b_m$表示出来；
2. 在最优解中，$b_1, b_2, \ldots, b_m$一定都是从$a_1, a_2, \ldots, a_n$中选择的；
3. $b_1, b_2, \ldots, b_m$一定不能被其它b_i表示出来。

性质一是题目要求，对于性质二，假设$b_i \notin a$，由于a凑不出的数b也不能凑出，因此b_i一定是由若干个a相加凑出的，假设$b_i = a_1 + a_2 + a_3$，根据性质一，这些a_j也都是由若干个b_j凑出来的，因此b_i即为若干个b_j凑出来的，那么这个b_i也就没有存在的意义了。因此b_i一定是a中的某个数。性质三也是显然的。

那么这题的做法就很明显了，我们先将a从小到大排序，对于每一个$a[i]$，我们判断其前面的所有数是否能凑出$a[i]$，如果无法凑出，那么b中一定要选择$a[i]$，否则一定不选。判断能否凑出某个数就是之前的完全背包求方案数的问题。

【代码】

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

const int N = 110, M = 25010;
int a[N], f[M];
int n, m;

int main()
{
    int T;
```

```
    cin >> T;
    while (T--)
    {
        cin >> n;
        for (int i = 0; i < n; i++) cin >> a[i];
        sort(a, a + n);
        m = a[n - 1];
        memset(f, 0, sizeof f);
        f[0] = 1;
        int res = 0;
        for (int i = 0; i < n; i++)
        {
            if (!f[a[i]]) res++;//如果a[i]的面值无法被前面的货币凑出来那么就一定要选
            for (int j = a[i]; j <= m; j++) f[j] += f[j - a[i]];
        }
        cout << res << endl;
    }
    return 0;
}
```

十六、AcWing 7. 混合背包问题

【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
```

【分析】

状态转移方程只与当前枚举的物品有关，也就是根据物品i的类型选择不同的转移方程，而与前$i-1$个物品无关，因此我们分情况判断即可。注意本题的数据量需要用二进制优化版多重背包。

【代码】

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

const int N = 1010;
int f[N];

int n, m;
```

```cpp

int main()
{
    cin >> n >> m;
    for (int i = 0; i < n; i++)
    {
        int v, w, s;
        cin >> v >> w >> s;
        if (!s)//完全背包
            for (int j = v; j <= m; j++) f[j] = max(f[j], f[j - v] + w);
        else
        {
            if (!~s) s = 1;//01背包就是只有1个物品的多重背包
            for (int k = 1; k <= s; k <<= 1)//二进制优化多重背包
            {
                for (int j = m; j >= k * v; j--)
                    f[j] = max(f[j], f[j - k * v] + k * w);
                s -= k;
            }
            if (s)
                for (int j = m; j >= s * v; j--)
                    f[j] = max(f[j], f[j - s * v] + s * w);
        }
    }
    cout << f[m] << endl;
    return 0;
}
```

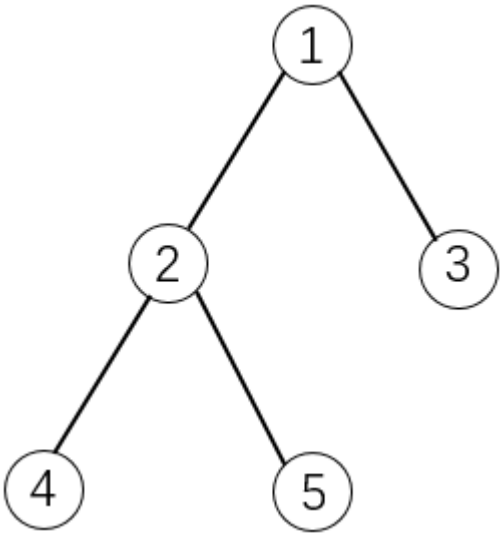
十七、AcWing 10. 有依赖的背包问题

【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。

如下图所示：



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（p_i均是合法的）。物品的下标范围是$1 \sim N$。

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】

$1 \leq N, V \leq 100$

$1 \leq v_i, w_i \leq 100$

【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
```

【分析】

状态表示：$f[u][j]$表示所有从以u为根的子树中选，且总体积不超过j的方案。

思路一：首先我们遍历物品组，也就是遍历当前结点u的子树，然后遍历背包的容积，这个时候当前结点我们看成是分组背包中的一个组，子节点的每一种选择我们都看作是组内一种物品，所以是从大到小遍历。因为我们是要遍历其子节点，所以当前节点我们是默认选择的，因此容积需要给当前结点预留，即j从$m-v[u]$开始循环。然后是枚举决策，我们从小到大枚举k，看看分配给儿子k的体积能带来多少回报。即状态转移方程为：$f[u][j]=max(f[u][j],f[u][j-k]+f[son][k])$。

思路二：DFS在遍历到u结点时，先考虑一定选上根节点u，因此初始化$f[u][v[u]\sim m]=w[u]$。然后遍历子树，同样是看成分组背包，j的范围为$[v[u]\sim m]$，因为小于$v[u]$则没有意义因为连根结点都放不下；k的范围为$[1,j-v[u]]$，因为当k大于$j-v[u]$时分给子树的容量过多，剩余的容量连根节点的物品都放不下了。状态转移方程同样为：$f[u][j]=max(f[u][j],f[u][j-k]+f[son][k])$。

【思路一代码】

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

const int N = 110;
int e[N], ne[N], h[N], idx;
int v[N], w[N], f[N][N];//f[u][j]表示所有从以u为根的子树中选,且总体积不超过j的方案
int n, m;


```

```cpp
void add(int u, int v)
{
    e[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void dfs(int u)
{
    for (int i = h[u]; ~i; i = ne[i])//循环物品组
    {
        int son = e[i];
        dfs(son);
        for (int j = m - v[u]; j >= 0; j--)//循环体积
            for (int k = 1; k <= j; k++)//循环决策,看看分配给儿子k的体积能带来多少回报
                f[u][j] = max(f[u][j], f[u][j - k] + f[son][k]);
    }
    //将物品u加进去
    for (int i = m; i >= v[u]; i--) f[u][i] = f[u][i - v[u]] + w[u];
    for (int i = 0; i < v[u]; i++) f[u][i] = 0;
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    int p, root;
    for (int i = 1; i <= n; i++)
    {
        cin >> v[i] >> w[i] >> p;
        if (!~p) root = i;
        else add(p, i);
    }
    dfs(root);
    cout << f[root][m] << endl;
    return 0;
}
```

【思路二代码】

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

using namespace std;
```

```cpp

const int N = 110;
int e[N], ne[N], h[N], idx;
int v[N], w[N], f[N][N];
int n, m;

void add(int u, int v)
{
    e[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void dfs(int u)
{
    for (int i = v[u]; i <= m; i++) f[u][i] = w[u];//u必选,因此先初始化f[u][v[u]~m]=w[u]
    for (int i = h[u]; ~i; i = ne[i])
    {
        int son = e[i];
        dfs(son);
        for (int j = m; j >= v[u]; j--)//j不能小于v[u],否则就选不了以u为根结点的子树的物品
            for (int k = 1; k <= j - v[u]; k++)//给子树分配的体积不能超过j-v[u],不然就无法选根物品u
                f[u][j] = max(f[u][j], f[u][j - k] + f[son][k]);
    }
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    int p, root;
    for (int i = 1; i <= n; i++)
    {
        cin >> v[i] >> w[i] >> p;
        if (!~p) root = i;
        else add(p, i);
    }
    dfs(root);
    cout << f[root][m] << endl;
    return 0;
}
```

十八、AcWing 11. 背包问题求方案数

【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10^9+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10^9+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
```

【分析】

状态转移方程为：$f[i][j]=max(f[i-1][j],f[i-1][j-v[i]]+w[i])$。设$cnt[i][j]$表示方案数，则：

- 若$f[i][j]=f[i-1][j]\neq f[i-1][j-v[i]]+w[i]$，那么$cnt[i][j]=cnt[i-1][j]$；
- 若$f[i][j]=f[i-1][j-v[i]]+w[i]\neq f[i-1][j]$，那么$cnt[i][j]=cnt[i-1][j-v[i]]$；

- 若 $f[i][j]=f[i-1][j]=f[i-1][j-v[i]]+w[i]$ ， 那 么 $cnt[i][j]=cnt[i-1][j]+cnt[i-1][j-v[i]]$。

本题的状态表示有两种：

- $f[i][j]$表示所有只考虑前i个物品，且总体积恰好为j的方案，则初始化时$cnt[0][0]=1$，最后统计结果时需要遍历$cnt[n][j],0\le j\le m$;
- $f[i][j]$表示所有只考虑前i个物品，且总体积至多为j的方案，则初始化时$cnt[0][j]=1,0\le j\le m$，最后结果即为$cnt[m]$。

【写法一代码】

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

const int N = 1010, MOD = 1e9 + 7;
int f[N], cnt[N];//f[i][j]表示所有只考虑前i个物品,且总体积恰好为j的方案
int n, m;

int main()
{
    cin >> n >> m;
    cnt[0] = 1;//由于是恰好因此初始化只有cnt[0]为1
    for (int i = 0; i < n; i++)
    {
        int v, w;
        cin >> v >> w;
        for (int j = m; j >= v; j--)
        {
            int maxv = max(f[j], f[j - v] + w), t = 0;
            if (maxv == f[j]) t += cnt[j];
            if (maxv == f[j - v] + w) t += cnt[j - v];
            cnt[j] = t % MOD;
            f[j] = maxv;
        }
    }
    int res = 0;
    for (int i = 0; i <= m; i++)
        if (f[i] == f[m]) res = (res + cnt[i]) % MOD;
    cout << res << endl;

    return 0;
```

```
}
```

【写法二代码】

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

const int N = 1010, MOD = 1e9 + 7;
int f[N], cnt[N];//f[i][j]表示所有只考虑前i个物品,且总体积至多为j的方案
int n, m;

int main()
{
    cin >> n >> m;
    fill(cnt, cnt + N, 1);//由于是至多因此初始化所有的cnt都为1
    for (int i = 0; i < n; i++)
    {
        int v, w;
        cin >> v >> w;
        for (int j = m; j >= v; j--)
        {
            int maxv = max(f[j], f[j - v] + w), t = 0;
            if (maxv == f[j]) t += cnt[j];
            if (maxv == f[j - v] + w) t += cnt[j - v];
            cnt[j] = t % MOD;
            f[j] = maxv;
        }
    }
    cout << cnt[m] << endl;
    return 0;
}
```

十九、AcWing 734. 能量石

【题目描述】

岩石怪物杜达生活在魔法森林中，他在午餐时收集了N块能量石准备开吃。

由于他的嘴很小，所以一次只能吃一块能量石。

能量石很硬，吃完需要花不少时间。

吃完第i块能量石需要花费的时间为S_i秒。

杜达靠吃能量石来获取能量。

不同的能量石包含的能量可能不同。

此外，能量石会随着时间流逝逐渐失去能量。

第i块能量石最初包含E_i单位的能量，并且每秒将失去L_i单位的能量。

当杜达开始吃一块能量石时，他就会立即获得该能量石所含的全部能量（无论实际吃完该石头需要多少时间）。

能量石中包含的能量最多降低至0。

请问杜达通过吃能量石可以获得的最大能量是多少？

【输入格式】

第一行包含整数T，表示共有T组测试数据。

每组数据第一行包含整数N，表示能量石的数量。

接下来N行，每行包含三个整数S_i, E_i, L_i。

【输出格式】

每组数据输出一个结果，每个结果占一行。

结果表示为`Case #x: y`，其中x是测试数据组别编号（从1开始），y是可以获得的最大能量值。

【数据范围】

$1 \leq T \leq 10$

$1 \leq N \leq 100$

$1 \leq S_i \leq 100$

$1 \leq E_i \leq 10^5$

$0 \leq L_i \leq 10^5$

【输入样例】

```
3
4
20 10 1
5 30 5
100 30 1
5 80 60
3
10 4 1000
10 3 1000
10 8 1000
2
12 300 50
5 200 0
```

【输出样例】

```
Case #1: 105
Case #2: 8
Case #3: 500
```

【样例解释】

在测试样例一中，有$N = 4$个宝石。杜达可以选择的一个吃石头顺序是：

- 吃第四块石头。这需要5秒，并给他80单位的能量。
- 吃第二块石头。这需要5秒，并给他5单位的能量（第二块石头开始时具有30单位能量，5秒后失去了25单位的能量）。
- 吃第三块石头。这需要100秒，并给他20单位的能量（第三块石头开始时具有30单位能量，10秒后失去了10单位的能量）。
- 吃第一块石头。这需要20秒，并给他0单位的能量（第一块石头以10单位能量开始，110秒后已经失去了所有的能量）。

他一共获得了105单位的能量，这是能获得的最大值，所以答案是105。

在测试样例二中，有$N = 3$个宝石。无论杜达选择吃哪块石头，剩下的两个石头的能量都会耗光。所以他应该吃第三块石头，给他提供8单位的能量。

在测试样例三中，有$N = 2$个宝石。杜达可以：

- 吃第一块石头。这需要12秒，并给他300单位的能量。
- 吃第二块石头。这需要5秒，并给他200单位的能量（第二块石头随着时间的推移不会失去任何能量！）。

所以答案是500。

【分析】

贪心部分：

假设在某一时刻t时第i块能量石还有E'_i的能量，第$i+1$块能量石还有E'_{i+1}的能量，则有以下两种可能的收益情况：

- 先吃第i块再吃第$i+1$块：$E'_i + E'_{i+1} - S_i * L_{i+1}$
- 先吃第$i+1$块再吃第i块：$E'_{i+1} + E'_i - S_{i+1} * L_i$

若先吃第i块的收益更大，那么$S_i * L_{i+1} < S_{i+1} * L_i$，因此我们按照该公式先将所有能量石排序，这样得到的解一定是最优的那部分。

动态规划部分：

状态表示：$f[i][j]$表示所有只从前i块能量石中选，且总体积（即为所需要的时间）恰好为j的方案。

状态计算：

- 不吃第i块能量石：$f[i][j] = f[i-1][j]$
- 吃第i块能量石：吃完之后当前花费的体积（时间）为j，说明是在$j - s_i$的时刻开始吃第i块能量石的，因此已经失去了$(j - s_i) * l_i$的能量，而能量最少为0，因此吃第i块能量石的时候其能量值为$max(0, e_i - (j - s_i) * l_i)$，则状态转移方程为：$f[i][j] = f[i-1][j - s_i] + max(0, e_i - (j - s_i) * l_i)$

由于状态定义是恰好，因此初始化$f[0][0] = 0$，其它的所有$f[0][i]$均为负无穷，最后结果为所有$f[n][j], 0 \le j \le m$的最大值。

【代码】

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

const int N = 110, M = 10010;//总时间最多为10000秒
int f[M];
int n, m;

struct Stone
{
    int s, e, l;
```

```cpp
    bool operator< (const Stone& t) const
    {
        return s * t.l < t.s * l;
    }
}st[N];

int main()
{
    int T;
    cin >> T;
    for (int c = 1; c <= T; c++)
    {
        cin >> n;
        memset(f, 0x8f, sizeof f);
        f[0] = 0, m = 0;
        for (int i = 0; i < n; i++) { cin >> st[i].s >> st[i].e >> st[i].l; m += st[i].s; }
        sort(st, st + n);
        for (int i = 0; i < n; i++)
        {
            int s = st[i].s, e = st[i].e, l = st[i].l;
            for (int j = m; j >= s; j--)
                f[j] = max(f[j], f[j - s] + max(0, e - (j - s) * l));
        }
        int res = 0;
        for (int i = 0; i <= m; i++) res = max(res, f[i]);
        printf("Case #%d: %d\n", c, res);
    }
    return 0;
}
```