简单背包问题

背包问题的定义

给定若干种物品，每种物品都有自己的重量和价格，在限定的总重量内，使得选取的物品的总价格最高。

0-1背包

0 - 1背包问题的定义就是：有N件物品和一个容量为V的背包。第i件物品的重量是$c[i]$，价值是$w[i]$。求解将哪些物品装入背包可使这些物品的重量总和不超过背包容量，且价值总和最大

因为每件物品只能有选 / 不选两种策略，因此被称为0 - 1背包

[# 2663] 采药

题目描述

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，在一段时间内如何让采到的草药价值最大。

输入格式

第一行有两个用空格隔开的整数T和$M(1 \leq T, M \leq 1000)$，T代表总共采药时间，M代表草药数目。接下来的M行每行包括两个在1到1000之间（包括1和1000）的整数c_i, v_i，分别表示采摘某种草药的时间和这株草药的价值。

输出格式

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。

样例输入

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

样例输出

```
3
```

Solution

根据线性动态规划的知识，我们不难想出该题的一个状态设计：$\mathrm{dp}[i][j]$表示使用了前i株草药，总时间为j的情况下，所能达到的最大的价值

同时，转移方程也不难想到：

$$\mathrm{dp}[i][j] = \max(\mathrm{dp}[i-1][j], \mathrm{dp}[i-1][j-c[i]] + v[i]) \text{ where } j \geq c[i]$$

其中，$c[i]$是采摘第i株药草所需的时间，$w[i]$是第i株药草的价值，顺序枚举i, j并且

此时，我们的时空复杂度均为$O(nm)$

考虑如何优化空间复杂度，注意到对于$\mathrm{dp}[i][j]$而言，我们使用到的**转移**只涉及到$\mathrm{dp}[i-1]$这一行，因此，我们可以使用**滚动数组**对空间进行优化

滚动数组一般将数组的第i行存储在$\mathrm{dp}[i\&1]$的位置，这样，随着i的不断增大，我们**实际**存储的行号会在$0, 1$之间不断切换，代码大致实现如下：

```
for (int i = 0; i < n; i++)
    for (int j = c[i]; j < m; j++)
        dp[i & 1][j] = dp[(i - 1) & 1][j];
    for (int j = c[i]; j < m; j++) {
        dp[i & 1][j] = max(dp[i][j], dp[(i - 1) & 1][j - c[i]] + v[i])
    }
```

此时，我们的空间复杂度降为了$O(m)$

如何进一步优化空间复杂度呢?

我们注意到，我们在每一行更新时，首先将前一行全部**拷贝**到该行，在进行转移的时候也使用了前一行的**更低列**的结果，那么，如果我们从**高到低**枚举列号——即我们采药花费的时间，那么就可以做到仅使用一个一维数组解决问题

Code

```
void solve(){
    int n, m; cin >> m >> n;
    vector dp(m + 1, 0);
    for (int i = 0; i < n; i++) {
        int c, v; cin >> c >> v;
        for (int j = m; j >= c; j--) cmax(dp[j], dp[j - c] + v);
    }

    cout << dp[m] << endl;
}
```

[# 6114] 数字组合

题目描述

给定N个正整数$A_1, A_2, \cdots A_N$，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$N(1 \le N \le 100)$和$M(1 \le M \le 10000)$

第二行包含N个整数，表示$A_1, A_2, \cdots A_N(1 \le A_i \le 1000)$。

输出格式

包含一个整数，表示方案总数，保证答案小于INT_MAX。

样例输入

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

样例输出

```
3
```

Solution

本题是一个比较裸的背包题，我们使用$\mathrm{dp}[i][j]$表示前i个数，得到数字M的方案数

转移方程也很显然：

$$\mathrm{dp}[i][j] = \mathrm{dp}[i-1][j] + \mathrm{dp}[i-1][j-a[i]]$$

其中，$a[i]$是第i个数，该方程的现实意义是，前i个数得到j的方案数，等于前$i-1$个数得到j的方案数，加上前$i-1$个数得到$j-a[i]$的方案数（加上$a[i]$即可）

同样的，我们也可以将dp数组的维度压缩至一维

Code

```cpp
void solve(){
    int n, m; cin >> n >> m;
    vector<int> dp(m + 1);
    dp[0] = 1;
    for (int i = 0; i < n; i++) {
        int u; cin >> u;
        for (int j = m; j >= u; j--) dp[j] += dp[j - u];
    }

    cout << dp[m] << '\n';
}
```

[# 4712] 子集的和

题目描述

对于从1到N的**连续整数**集合，能划分成两个子集合，且保证每个集合的数字和是相等的。举个例子，如果$N=3$，对于1，2，3能划分成两个子集合，每个子集合的所有数字和是相等的：3和$1,2$
这是唯一一种分法（交换集合位置被认为是同一种划分方案，因此不会增加划分方案总数）如果$N=7$，有四种方法能划分集合1，2，3，4，5，6，7，每一种分法的子集合各数字和是相等的:

$1, 6, 7$和$2, 3, 4, 5$
$2, 5, 7$和$1, 3, 4, 6$
$3, 4, 7$和$1, 2, 5, 6$
$1, 2, 4, 7$和$3, 5, 6$

给出N，你的程序应该输出划分方案总数，如果不存在这样的划分方案，则输出0。

输入格式

一行一个整数$N(1 \leq N \leq 71)$

输出格式

一行一个整数，划分方案总数

样例输入

```
7
```

样例输出

```
4
```

Solution

注意到，要是$sum = \sum_{i=1}^{n}$不能被2整除时，集合无论如何都不可能被划分成相等的两个部分

而如果能整除，就等价于于我们刚才讲过的，子集的和

我们使用$\mathrm{dp}[i][j]$表示前i个数，得到数字M的方案数

转移方程也很显然：

$$\mathrm{dp}[i][j] = \mathrm{dp}[i-1][j] + \mathrm{dp}[i-1][j-a[i]]$$

其中，$a[i]$是第i个数，该方程的现实意义是，前i个数得到j的方案数，等于前$i-1$个数得到j的方案数，加上前$i-1$个数得到$j-a[i]$的方案数（加上$a[i]$即可）

同样的，我们也可以将dp数组的维度压缩至一维

最后输出$\mathrm{dp}[\frac{sum}{2}]$即可

Code

```
void solve(){
    int n; cin >> n;
    int m = n * (n + 1) / 2;
    if (m & 1) {cout << "0\n"; return;}
    m >>= 1;
    vector<ll> dp(m + 1);
    dp[0] = 1;

    for (int i = 1; i <= n; i++) for (int j = m; j >= i; j--) dp[j] += dp[j -
i];

    cout << dp[m] / 2 << endl;
}
```

[# 49648] 目标和

题目描述

给你一个整数数组a和一个整数m。

向数组中的每个整数前添加**加号**或**减号**，然后串联起所有整数，可以构造一个表达式：

例如$a = \{2, 1\}$可以在2之前添加加号，在1之前添加减号，然后串联起来得到表达式为$(+2 - 1)$。

返回可以通过上述方法构造的、运算结果等于m的不同**表达式**的数目。

输入格式

第一行输入一个整数$n(1 \leq n \leq 60)$；第二行输入n个整数$a_1, a_2, \cdots a_n(-10^3 \leq a_i \leq 10^3)$；第三行输入目标数$m(-10^3 \leq m \leq 10^3)$。

输出格式

满足计算和为目标数的所有符号添加形式的总数

样例输入

```
5
1 1 1 1 1
3
```

样例输出

```
5
```

Solution

本题的状态设计也比较简单，$\mathrm{dp}[i][j]$表示前i个数构成的表达式，其值为j的数量

转移方程也比较好想：

$$\mathrm{dp}[i][j] = dp[i-1][j-a[i]] + dp[i-1][j+a[i]]$$

代表前i个数构成的表达式想要凑出j来，要么往$a[i]$前加正号（即前$i-1$个数得到$j-a[i]$），要么往$a[i]$前加负号（即前$i-1$个数得到$j+a[i]$）

Code

```
void solve(){
    int n; cin >> n;
    vector<int> a(n);
    for (auto& v: a) cin >> v;
    const int offset = 1e3 * n + 100;
    const int maxm = offset << 1;
    int m; cin >> m;
    vector dp(2, vector(maxm, 0ll));
    int now = 1;
    dp[0][offset] = 1;
    for (auto v: a) {
        for (int i = 0; i < maxm; i++) dp[now][i] = 0;
        for (int i = 0; i < maxm; i++) {
            if (i - v >= 0 && i - v < maxm) dp[now][i - v] += dp[now ^ 1][i];
            if (i + v >= 0 && i + v < maxm) dp[now][i + v] += dp[now ^ 1][i];
        }
        now ^= 1;
    }

    cout << dp[now ^ 1][m + offset] << endl;
}
```

[# 2665] 预算

题目描述

张琪曼等人要为太空战指挥中心购置设备，魔法学院的院长昨天说：“指挥中心需要购买哪些设备，你们研究了算，只要不超过N元钱就行”。所以今天一早，张琪曼就开始做预算了，她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。指挥中心想配备的东西很多，肯定会超过院长限定的N元。于是，她把每件物品规定了一个重要度，分为5等：用整数$1,2,3,4,5$表示，第5等最重要。她还从互联网上查到了每件物品的价格（都是10元的整数倍）。她希望在不超过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，$\cdots$，j_k，则所求的总和为：

$v[j_1] \times w[j_1] + v[j_2] \times w[j2] + \cdots + v[j_k] \times w[j_k]$。

请你帮助张琪曼设计一个满足要求的购物单。

输入格式

第1行为两个正整数，用一个空格隔开：$N(1 \leq N \leq 32000), m(1 \leq m \leq 60)$，分别表示总钱数和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_j, p_j, q_j（其中$v(1 \leq v \leq 10000)$表示该物品的价格，$p(1 \leq p \leq 5)$表示该物品的重要度，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是所属主件的编号）

输出格式

输出只有一个正整数，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。

样例输入

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

样例输出

```
2200
```

Solution

如果说只有**主件**，那么本题就是一个简单的0 - 1背包问题，但由于选择附件的时候，必须选择主件，因此，我们需要考虑如何处理选择附件时选择主件这一限制条件

一个很简单的想法是，我们将主件和附件**组合**起来，比如某主件的价格是v_1，重要度是p_1，某附件的价格是v_2，重要度是p_2，我们考虑创造一件新的物品，价格为v_1+v_2，重要度为$v_1p_1+v_2p_2$，这件物品代表该主件和附件**捆绑**的结果

但此时会有一个问题：如果说我们只从原物品中删去**附件**，那么我们可能会购买**两次**主件，如果说主件和附件**一起删去**，那我们就失去了**只选择主件**的可能性

因此，我们对于每个**主件及其附件**一起讨论，开一个临时数组，在**强制选择**主件的情况下，对附件进行0 - 1背包的做法，然后将这个临时数组和我们的主数组元素取$\max$即可

Code

```
void solve(){
    int n, m; cin >> m >> n;
    struct item {
        int v, p, q;
    };

    struct mi {
        int v, p, q;
```

```
        vector<item> att;
    };

    vector<mi> a(n);

    for (auto& [v, p, q, att]: a) {
        cin >> v >> p >> q; q--;
    }

    for (auto& [v, p, q, att]: a) if (q != -1) {
        a[q].att.push_back({v, p, q});
    }

    vector<int> dp(m + 1);

    for (auto& [v, p, q, att]: a) {
        if (q != -1) continue;
        auto tmp = dp;
        for (int i = m; i >= v; i--) tmp[i] = tmp[i - v] + p * v;
        for (auto [vv, pp, qq]: att) for (int i = m; i >= v + vv; i--)
            cmax(tmp[i], tmp[i - vv] + pp * vv);
        for (int i = 0; i <= m; i++) cmax(dp[i], tmp[i]);
    }

    cout << dp[m] << endl;
}
```

[# 29204] 0/1背包问题第k优解

题目描述

0 - 1背包的第k个最大价值。注意，两种不同方法得到相同价值是算作同一个价值。这意味着，可以得到的价值序列将是一个**严格递减**的序列，从第1个最大值，第2个最大值，$\cdots$，第k个最大值。

如果不同值的总数小于k，就输出0

输入格式

第一行包含一个整数$t(1 \leq t \leq 50)$,表示有几组测试数据。

每一组测试数据分为三行：

第一行包含三个整数$n, v, k(1 \leq n \leq 100, 1 \leq v \leq 1000, 1 \leq k \leq 30)$表示物品的数量，背包的体积以及我们题目描述中的的$k$。

第二行，包含n个整数表示每个物品的价值。

第三行，包含n个整数表示每个物品的体积。

输出格式

每行一个整数，第k大值

样例输入

```
3
5 10 2
1 2 3 4 5
5 4 3 2 1
5 10 12
1 2 3 4 5
5 4 3 2 1
5 10 16
1 2 3 4 5
5 4 3 2 1
```

样例输出

```
12
2
0
```

Solution

在普通的0 - 1背包中，我们求的是**最大值**，相当于题中$k = 1$的情况

先从简单的入手，考虑$k = 2$的情况如何求解

在普通的0 - 1背包中，我们用$\mathrm{dp}[i][j]$表示前i个物品，总重量为j时的最大价值。显然，我们可以使用$\mathrm{dp}[i][j][1/2]$分别代表最大值和次大值，转移方程也很好维护，当一个新的价值转移到$\mathrm{dp}[i][j]$时，我们只需要把取$\max$改为和$\mathrm{dp}[i][j][1/2]$比较即可

那如果$k = 30$，此时的时间复杂度为$O(nmk^2)$，因为有$O(nmk)$种状态，每种状态转移是$O(k)$的，此时不能通过此题

考虑如何优化，如果我们用`std::set`来维护k大值的话，状态转移就从$O(k)$降为了$O(\log k)$，但由于`std::set`底层是**红黑树**，会涉及到较多的非连续内存操作，因此还是不能通过此题

注意到，我们在将$\mathrm{dp}[i-1][j-w_i][1\sim k]+v_i$转移到$\mathrm{dp}[i][j][1\sim k]$时，可以**先算出所有的前项**，再将其和后项进行合并，这就是两个**有序数组**合并的问题，我们可以用归并排序的思想轻松解决，此时，我们使用$O(k)$的时间复杂度处理了$O(k)$的状态，因此对于每个状态，我们处理的时间是$O(1)$的，至此，我们将算法的整体时间复杂度降为了$O(nmk)$，足以通过此题

Code

```cpp
void solve() {
    int n, m, k;
    cin >> n >> m >> k;
    vector<vector<int>> dp(m + 1);
    vector<pair<int, int>> a(n);

    auto insert = [&k](vector<int>& vec, int v) {
        if (!vec.size()) vec.push_back(v);
        else if (vec.back() != v) vec.push_back(v);
    };
```

```
    auto merge = [&k, &insert](vector<int> &a, vector<int>& b, vector<int>& mid)
{
        int i = 0, j = 0;
        while (i < a.size() && j < b.size() && mid.size() < k) {
            if (a[i] > b[j]) insert(mid, a[i++]);
            else insert(mid, b[j++]);
        }

        while (i < a.size() && mid.size() < k) insert(mid, a[i++]);
        while (j < b.size() && mid.size() < k) insert(mid, b[j++]);
        a = mid;
    };

    for (auto& v: dp) v.push_back(0);

    for (auto& [w, v]: a) cin >> v;
    for (auto& [w, v]: a) {
        cin >> w;
        for (int i = m; i >= w; i--) {
            static vector<int> tmp, c;
            tmp.clear(); c.clear();
            for (auto val: dp[i - w]) tmp.push_back(val + v);
            merge(dp[i], tmp, c);
        }
    }

    if (dp[m].size() < k) cout << "0\n";
    else cout << dp[m].back() << endl;
}
```

完全背包

完全背包问题，就是在0 - 1背包问题的基础上，去除了**每件物品只能选一次**的限制，即有N**种**物品和一个容量为V的背包。第i件物品的重量是$c[i]$，价值是$w[i]$。求解将哪些物品装入背包可使这些物品的重量总和不超过背包容量，且价值总和最大

[# 2716] 完全背包问题

题目描述

话说张琪曼和李旭琳又发现了一处魔法石矿（运气怎么这么好？各种嫉妒羡慕恨啊），她们有一个最多能装m公斤的背包，现在有n种魔法石，每种的重量分别是W_1，W_2，$\cdots$，W_n，每种的价值分别为C_1，C_2，$\cdots$，C_n。若每种魔法石的个数足够多，求她们能获得的最大总价值。

输入格式

第一行为两个整数，背包容量m和魔法石的数量$n(1 \le n \le 100, 1 \le m \le 1000)$。

以后每行为两个整数$c_1, w_i(1 \le c_i, w_i \le 100)$，表示每块魔法石的重量和价值。

输出格式

一行一个整数，代表获得的最大总价值。

样例输入

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

样例输出

```
5
```

Solution

对于一个简单的想法是，由于重量为c_i的物品，至多能够选择$max_i = \lfloor \frac{m}{c_i} \rfloor$件，因此我们将**每件物品**复制$max_i$件，然后，问题就转换成了一个简单的0 - 1背包问题

由于每件物品至多会被选择max_i次，而max_i是$O(m)$的，因此我们至多会有$O(nm)$件物品，此时的时复杂度是$O(nm^2)$

考虑我们压缩数组时，为什么需按照**从大到小**的顺序枚举？当时我们讲，是为了避免**重复选取**某个物品，这不恰好就是我们完全背包问题所**需要**的吗？

因此，我们将枚举顺序改回**从小到大**即可

Code

```
void solve() {
    int n, m; cin >> m >> n;
    vector dp(m + 1, 0);

    for (int i = 0; i < n; i++) {
        int w, v; cin >> w >> v;
        for (int i = w; i <= m; i++) cmax(dp[i], dp[i - w] + v);
    }

    cout << dp[m] << endl;
}
```

[# 6115] 自然数拆分

题目描述

给定一个自然数N，要求把N拆分成若干个正整数相加的形式，参与加法运算的数可以重复。

注意:

1. 拆分方案不考虑顺序;

2. 至少拆分成2个数的和。

求拆分的方案数对2147483648的结果。

输入格式

一个自然数$N(1 \leq N \leq 4000)$。

输出格式

输入一个整数，表示结果。

样例输入

```
7
```

样例输出

```
14
```

Solution

本题可以转换为：有N个物品，每个物品的重量是$1, 2, \cdots, N$，背包容量为N，求恰好将背包装满的方案数

转换后就是道很简单的完全背包题目了，但是题目给了我们一个限制，**至少拆分成两个数的和**

对于容量为N而言，我们有且只有一种方案，即选择**一件**重量为N的物品，才不满足条件

因此，我们输出总方案数减一即可

Code

```
void solve() {
    int n; cin >> n;
    vector dp(n + 1, (unsigned int)0);
    dp[0] = 1;
    for (int i = 1; i <= n; i++) {
        for (int j = i; j <= n; j++) dp[j] += dp[j - i];
    }

    cout << ((dp[n] - 1) & (UINT_MAX >> 1)) << endl;
}
```

[# 29179] 完全背包求具体方案

题目描述

有N**种**重量和价值分别为w_i, v_i的物品，每种物品可以挑选任意多件。从这些物品中挑选总重量不超过V的物品，求出挑选物品价值总和最大的挑选方案，输出任意一组方案即可。

输入格式

第一行两个整数$N, V(1 \leq N, V \leq 100)$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$v_i, w_i(1 \leq v_i, w_i) \leq 100$，用空格隔开，分别表示第$i$件物品的价值和体积。

输出格式

输出第一行，包含一个整数，表示最大价值和。

接下来若干行，每行输出两个整数和$i, k(k > 0)$，表示物品i挑选k件。若有多种解，输出任意一组即可。

样例输入

```
3 7
4 3
5 4
3 2
```

样例输出

```
10
1 1
3 2
```

Solution

回忆我们完全背包的求解过程，我们有如下两种转移：

$$\begin{aligned}\mathrm{dp}[i][j] &= \max(\mathrm{dp}[i][j], \mathrm{dp}[i][j - w[i]] + v[i]) \\ \mathrm{dp}[i][j] &= \max(\mathrm{dp}[i][j], \mathrm{dp}[i - 1][j])\end{aligned}$$

那么，我们从$\mathrm{dp}[n][m]$一路回溯，如果发现$\mathrm{dp}[i][j] = \mathrm{dp}[i - 1][j]$，则说明我们没有选用第$i$项物品，那就$i = i - 1$，否则说明我们选用了**一次**第$i$项物品，并且将$j = j - a[i]$

Code

```cpp
void solve() {
    int n, m; cin >> n >> m;
    vector<pair<int, int>> a(n);
    for (auto& [f, s]: a) cin >> s >> f;

    vector dp(n + 1, vector(m + 1, 0));
    for (int i = 1; i <= n; i++) {
        auto [w, v] = a[i - 1];
        for (int j = 0; j <= m; j++) {
            dp[i][j] = dp[i - 1][j];
            if (j >= w) cmax(dp[i][j], dp[i][j - w] + v);
        }
    }

    cout << dp[n][m] << '\n';
    vector ans(n, 0);
```

```
    while (n && m) {
        if (m < a[n - 1].first || dp[n - 1][m] == dp[n][m]) n--;
        else {m -= a[n - 1].first; ans[n - 1]++;}
    }

    for (int i = 0; i < ans.size(); i++) if (ans[i]) cout << i + 1 << ' ' <<
ans[i] << '\n';
}
```

[# 2715] 收益

题目描述

“建太空梯进入太空要1兆亿？”魔法学院的院长瞪大了眼睛。

“这只是基础设施的费用，后期还要……”墨老师掰着手指算。

“哎呀，现在地主也很穷啊，学院的钱批下来就这么多，你想办法用这笔钱在债券市场上获得最大收益吧。”院长皱着眉头。

简单来说，就是你有一笔钱，你要将这笔钱去投资债券，现在有d种债券，每种债券都有一个价值和年收益，债券的价值是1000的倍数，问你如何投资在n年后的获得最大收益。

输入格式

第一个为一个整数$t(1 \leq t \leq 3)$，表示有t组数据。

每组数据第一行有两个整数，表示初始资金m和年数$n(1 \leq m \leq 5 \times 10^4, 1 \leq n \leq 100)$。

每组数据第二行为一个整数$1 \leq d \leq 10$，表示债券种类数。

随后d行每行有两个整数，表示该债券的价值和年收益。年收益不会超过债券价值的10%。

所有数据不超过整型取值范围。

输出格式

每组数据，输出n年后获得的最大收益，保证最大收益不超过400%。

样例输入

```
1
10000 4
2
4000 400
3000 250
```

样例输出

```
14050
```

Solution

如果年数为1，那么本题就是一个简单的完全背包

一个显然的贪心结论是，我们每一年的**投资收益**都最大的情况下，n年的收益一定是最大的

而收益，等于我们**收入减去本金**的最大值，因此我们枚举我们的所有**本金**，找到最大的收益并且加到本金上，即最大的$\mathrm{dp}[i]-i$，作为下一年的投资本金（因为**本金使用最多**不一定收益最大）

在进行n此上述操作后，我们输出本金即可

Code

```
void solve() {
    int n, m, t; cin >> m >> t >> n;
    vector<pair<int, int>> a(n);
    for (auto& [w, v]: a) { cin >> w >> v; v += w; }
    vector dp(m << 2, 0);

    while (t--) {
        fill(all(dp), 0);
        for (auto& [w, v]: a) for (int i = w; i <= m; i++)
            cmax(dp[i], dp[i - w] + v);
        int tmp = m;
        for (int i = m; i >= 0; i--) cmax(m, tmp - i + dp[i]);
    }

    cout << m << '\n';
}
```

[# 3264] 猪猪储蓄罐

题目描述

小林决定存钱准备买房。但是他平时花钱如流水，所以也存不出什么钱。因此他决定从最小最小的零钱开始存。而小林为了不让自己乱用钱，决定用那种不砸破拿不出钱的猪猪储蓄罐。但是在砸碎储蓄罐之前，小林还是不能知道自己到底有多少钱。请帮他计算他储蓄罐里**最少**有多少钱。只要知道最少就好了，好让他可以知道离渺茫的房子还差多少钱。

输入格式

第一行两个正整数E和$F(1\le E\le F\le 10000)$，表示空的猪猪储蓄罐的重量和存满了钱的重量，中间用一个空格隔开。

接下来一行一个正整数$N(1\le N\le 500)$，表示有N种硬币。

再接下来N行，每行有两个正整数W_i,V_i，分别表示每种硬币的价值和每种硬币的重量，中间用一个空格隔开。

输出格式

输出重量F的**最小价格**，如果能，则输出一行字符串`The minimum amount of money in the piggy-bank is X.`”，X是最小的价值；否则，输出`This is impossible.`注意字符串严格匹配。

样例输入

```
10 110
2
1 1
30 50
```

样例输出

```
The minimum amount of money in the piggy-bank is 60.
```

Solution

首先做一个简单的转换$m = F - E$，此时背包的容量就为m了

如果说对于某个重量m，我们求它的**最大**价格，那就是一个简单的完全背包板子题

回忆完全背包的两个转移方程：

$$\mathrm{dp}[i][j] = \max(\mathrm{dp}[i][j], \mathrm{dp}[i-1][j])$$
$$\mathrm{dp}[i][j] = \max(\mathrm{dp}[i][j], \mathrm{dp}[i][j-w_i] + v_i)$$

很明显，我们在求最大价格时，选用的是两种转移的较大值，那么，如果我们选用较小值，即将$\max$改为$\min$，就能求到**最小价格**

Code

```
void solve() {
    int e, f; cin >> e >> f;
    int m = f - e;
    int n; cin >> n;
    vector<pair<int, int>> a(n);
    for (auto& [w, v]: a) cin >> v >> w;

    vector dp(m + 1, inf);
    dp[0] = 0;
    for (int i = 1; i <= n; i++) {
        auto [w, v] = a[i - 1];
        for (int j = w; j <= m; j++) cmin(dp[j], dp[j - w] + v);
    }

    if (dp[m] == inf) {cout << "This is impossible." << endl; return;}

    cout << "The minimum amount of money in the piggy-bank is " << dp[m] << "."
<< endl;
}
```

多重背包

完全背包问题，就是在0 - 1背包问题的基础上，将**每件物品只能选一次**的限制，转变为了每件物品至多选$c[i]$件，即有N**种**物品和一个容量为V的背包。第i件物品的重量是$w[i]$，价值是$v[i]$，一共有$c[i]$件。求解将哪些物品装入背包可使这些物品的重量总和不超过背包容量，且价值总和最大

[# 28687] 多重背包问题 I

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

样例输入

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

样例输出

```
10
```

Solution

和完全背包的思路类似，我们可以把第i种物品**拆成**c_i个重量为w_i，价值为v_i的物品，这样就成了0 - 1背包的问题

对于本题而言，时间复杂度为$O(NCV)$，足以通过本题

Code

```
void solve(){
    int n, m; cin >> n >> m;
    vector<pair<int, int>> a;
    for (int i = 0; i < n; i++) {
        int w, v, c; cin >> w >> v >> c;
        while (c--) a.emplace_back(w, v);
```

```
    }

    vector<int> dp(m + 1);
    for (auto [w, v]: a) for (int i = m; i >= w; i--)
        cmax(dp[i], dp[i - w] + v);

    cout << dp[m] << '\n';
}
```

[# 28688] 多重背包问题 II

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，$N, V(0 < N \leq 1000, 0 < V \leq 2000)$，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$v_i, w_i, s_i(0 < v_i, w_i, s_i \leq 2000)$，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。

样例输入

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

样例输出

```
10
```

Solution

此时题目数据得到了一定的加强，采用我们刚刚使用的，将多重背包拆解成多个0 - 1背包的做法已经没有办法通过此题

假设我们的物品数量为c，我们将c拆为如下形式：

$$c = 2^0 + 2^1 + \cdots + 2^k + res$$

其中，$2^0, 2^1, \cdots, 2^k$是一段连续的2的幂次，而res是保证$res \leq 2^k$下的**余数**

显然，右式的项数，是$O(\log c)$级别的，同时，其中某些项组合起来的话，能表示$[0, c]$之间的任意一个整数

因此，我们可以将物品分别拆成由$2^0, 2^1, \cdots, 2^k, res$个原始物品组成的**新物品**，这样既可以保证增加的物品数量是$O(\log c)$级别，又可以选取**任意数量**的原始物品，后续的状态和转移方程，和0 - 1背包是一样的

总体的时间复杂度$O(NM\log c)$，足以通过本题

Code

```
void solve() {
    int n, m; cin >> n >> m;
    vector<pair<int, int>> a;
    for (int i = 0; i < n; i++) {
        int w, v, c; cin >> w >> v >> c;
        int res = c, t = 1;
        while(c) {
            if (res <= t) {
                a.emplace_back(w * res, v * res);
                break;
            }
            a.emplace_back(w * t, v * t);
            res -= t; t <<= 1;
        }
    }

    vector<ll> dp(m + 1);
    for (auto [w, v]: a)
        for (int j = m; j >= w; j--) cmax(dp[j], dp[j - w] + v);

    cout << dp[m] << endl;
}
```

[# 2690] 硬币问题

题目描述

给你n种硬币，知道每种的面值A_i和每种的数量C_i。问能凑出多少种不大于m的面值。

输入格式

有多组数据，每一组第一行有两个整数$n(1 \leq n \leq 100)$和$m(m \leq 10^5)$，第二行有$2n$个整数，即面值$A_1, A_2, \cdots A_n(1 \leq A_i \leq 10^5)$和数量$C_1, C_2, \cdots C_n(1 \leq C_i \leq 1000)$。所有数据结束以2个0表示。

输出格式

每组数据输出一行答案。

样例输入

```
3 10
1 2 4 2 1 1
2 5
1 4 2 1
0 0
```

样例输出

```
8
4
```

Solution

本题是多重背包的一个简单应用，我们将多重背包问题转换成相应的0 - 1背包问题之后，将状态$\mathrm{dp}[i][j]$设为使用前i个硬币，**能否凑出面值**j，转移方程如下：

$$\mathrm{dp}[i][j] = \mathrm{dp}[i-1][j] \mid \mathrm{dp}[i-1][j-w_i]$$

其中，$\mid$表示**或**操作，w_i表示第i个硬币的面值

使用滚动数组优化后的代码如下：

```
for (auto [w, v]: b) for(int j = m; j >= w; j--) dp[j] |= dp[j - w];
```

此时的时间复杂度为$O(NM\log c)$，通过此题可能比较困难

注意到，在第i个硬币加入我们的集合时，对于面值$m, m-1, m-2, \cdots, w$，我们分别和$m-v, m-v-1, \cdots, 0$进行了**或操作**，可以看到$\{m, m-v\}, \{m-1, m-v-1\}\cdots$是**等差**的，我们写成二进制形式能够更加简单地理解

$$\begin{aligned} w_i &= 3 \\ \mathrm{dp}[i-1] &= 0001000011 \\ \mathrm{dp}[i] &= 1001011011 \end{aligned}$$

在二进制形势下不难看出，$\mathrm{dp}[i] = \mathrm{dp}[i-1] \mid \mathrm{dp}[i-1] \ << w_i$

此时，我们可以使用`std::bitset`进行优化，代码大致如下：

```
for (auto [w, v]: b) bs |= bs << w;
```

此时的时间复杂度为$O\left(\frac{NM\log c}{\omega}\right)$，其中，$\omega$代表机器字长，通常为$32$或$64$

对于本题而言，还有一种动态规划结合**贪心**的做法：使用一个used数组，$\mathrm{used}[i][j]$表示使用前i种硬币，和为j时，所用的硬币i的最小个数，转移方程如下：

$$\begin{aligned} &\mathrm{used}[i][j] = 0 \text{ where } \mathrm{dp}[i-1][j] = 1 \\ &\mathrm{used}[i][j] = \mathrm{dp}[i][j-w] + 1 \text{ where } \mathrm{dp}[i][j-w] = 1 \text{ and } \mathrm{dp}[i-1][j] = 0 \end{aligned}$$

其中，转移方程一代表着我们面值j能够被前$i-1$种硬币表达出来，我们此时就**不需要**第i种硬币，即其使用的最小个数就是0；而转移方程二则是在**转移方程一失效**时，使用一个i类型硬币，从$\mathrm{dp}[i][j-w]$转移而来

Code

```
// greedy + dp
void solve(){
    int n, m; cin >> n >> m;

    if (!n) exit(0);
    vector<pair<int, int>> a(n);
    for (auto& [w, c]: a) cin >> w;
    for (auto& [w, c]: a) cin >> c;
    if (m <= 0) {cout << 0 << endl; return;}

    vector<int> used(m + 1), dp(m + 1);
    dp[0] = 1;
    for (auto [w, c]: a) {
        fill(all(used), 0);
        for (int i = w; i <= m; i++) {
            if (!dp[i] && dp[i - w] && used[i - w] < c) {
                dp[i] = 1; used[i] = used[i - w] + 1;
            }
        }

    }
    cout << accumulate(all(dp), -1) << endl;
}
```

```
// bitset
bitset<maxm> bs;

void solve(){
    int n, m; cin >> n >> m;
    bs <<= m + 1;

    if (!n) exit(0);
    vector<pair<int, int>> a(n), b;
    for (auto& [w, c]: a) cin >> w;
    for (auto& [w, c]: a) cin >> c;
    for (auto& [w, c]: a) {
        int res = c, t = 1;
        while(c) {
            if (res <= t) {
                b.emplace_back(w * res, w * res);
                break;
            }
            b.emplace_back(w * t, w * t);
            res -= t; t <<= 1;
        }
    }

    bs[0] = 1;
    for (auto [w, v]: b) bs |= bs << w;

    int ans = 0;
```

```
    for (int i = 1; i <= m; i++) ans += bs[i];

    cout << ans << '\n';
}
```

[# 2671] 太空梯

题目描述

有一群牛要上太空。他们计划建一个太空梯-----用一些石头垒。他们有$n(1 \leq n \leq 400)$种不同类型的石头，每一种石头的高度为$h_i(1 \leq h_i \leq 100)$，数量为$c_i(1 \leq c_i \leq 10)$,并且由于会受到太空辐射，每一种石头不能超过这种石头的最大建造高度$a_i(1 \leq a_i \leq 4 \times 10^4)$。帮助这群牛建造一个最高的太空梯。

输入格式

第一行为一个整数即n。

第二行到第$n+1$行每一行有3个数，代表每种类型石头的特征，即高度h_i，限制高度a_i和数量c_i。

输出格式

一个整数，即修建太空梯的最大高度。

样例输入

```
3
7 40 3
5 23 8
2 52 6
```

样例输出

```
48
```

Solution

本题在普通的**多重背包**上加了一点点限制，就是有些物品在总高度**超过其限制**时不能再加入到背包里

这个限制其实很容易解决，我们仍设$\mathrm{dp}[i][j]$为使用前i种石头，能否达到j高度，我们在每个物品进行拿或者不拿的判断时，将背包第二维（即高度）枚举的值域由$[h_i, m]$降为$[h_i, a_i]$即可

太空梯的最大高度m也很好得到，就是所有石头的最大高度

转移方程如下：

$$\mathrm{dp}[i][j] = \mathrm{dp}[i-1][j] \mid \mathrm{dp}[i-1][j - k \times w_i] \text{ where } k \leq c_i$$

其中，w_i是石头的长度，c_i是石头的块数

但我们直接按照如上状态和转移方程，并不能通过样例，这是因为，我们在选择石头的时候，并没有**优先选择最大高度较低的石头**，导致我们可能没有先选择最大高度**较低**的石头作为**垫脚石**

假设我们最优解石头序列是$x_1, x_2, \cdots, x_k$，一定存在一种方案，使得他们的最大高度**单调不减**，假设x_i的最大高度高于x_j且$i < j$，那么，我们简单地交换x_i, x_j，原限制一定满足

因此，我们按照**最大高度**为关键字，对所有的石头进行**排序**，排完序后进行普通的多重背包，即可通过此题

Code

```
void solve(){
    int n, m = 0; cin >> n;
    vector<tuple<int, int, int>> a(n);
    for (auto& [h, c, lim]: a) {
        cin >> h >> lim >> c;
        cmax(m, lim);
    }

    sort(all(a), [](auto& x, auto& y){
        return get<2>(x) < get<2>(y);
    });

    vector<int> dp(m + 1), used(m + 1);
    dp[0] = 1;
    for (auto& [h, c, lim]: a) {
        fill(all(used), 0);
        for (int i = h; i <= lim; i++)
            if (!dp[i] && dp[i - h] && used[i - h] < c) {
                used[i] = used[i - h] + 1;
                dp[i] = 1;
            }
    }

    for (int i = m; i >= 0; i--) if (dp[i]) {
        cout << i << endl; return;
    }
}
```

[# 2670] 均分魔法石

题目描述

输入6个数字，分别代表价值为$1, 2, 3, 4, 5, 6$的魔法石的数量，问能不能将所有魔法石分为价值相同的两份。并且不能将魔法石割开。

输入格式

有多组数据，每组数据每行包括6个非负数的整数，$n_1, n_2, \cdots, n_6$代表价值为i的魔法石有$n_i (0 \le n_i \le 20000)$个。全部数据结束以**全0或** `EOF` **表示。**

输出格式

每组数据输出`Collection #k:`，k是测试数据组数，然后输出`Can be divided.`或`Can't be divided.`，分别代表**能**被均分和**不能**被均分。

样例输入

```
1 0 1 2 0 0
1 0 0 0 1 1
0 0 0 0 0 0
```

样例输出

```
Collection #1:
Can't be divided.
Collection #2:
Can be divided.
```

Solution

首先计算魔法石总价值$tot = \sum_{i=1}^{6} i \times n_i \leq 420000$，而我们判断的是**能否达到总价值的一半**，因此背包容量$m \leq 210000$，将物品进行**二进制拆分**后的数量$n \leq 6 \times \log(20000) = 90$，因此，我们选用**多重背包**模型进行求解，即可通过此题

特别需要注意的是，如果魔法石总价值tot是**奇数**，那么我们怎么也无法均分魔法石

Code

```cpp
void solve() {
    int n = 6, tot = 0;
    vector<int> a;
    for (int i = 0; i < n; i++) {
        int w = i + 1, c, t = 1;
        if (!(cin >> c)) exit(0);
        tot += w * c;
        while(c) {
            if (c <= t) {
                a.emplace_back(w * c);
                break;
            }
            a.emplace_back(w * t);
            c -= t; t <<= 1;
        }
    }

    if (!a.size()) exit(0);

    static int cnt = 0;
    cout << "Collection #" << ++cnt << ":" << endl;

    if (tot & 1) {cout << "Can't be divided.\n"; return;}

    int m = tot / 2;
    vector<int> dp(m + 1); dp[0] = 1;
    for (auto w: a) for (int j = m; j >= w; j--) dp[j] |= dp[j - w];

    cout << (dp[m] ? "Can be divided." : "Can't be divided.") << endl;
}
```

分组背包

分组背包问题，就是在0 - 1背包问题的基础上，将物品分组，每组内部选的物品件数**加以限制**，即有N**种**物品和一个容量为V的背包。第i件物品的重量是$w[i]$，价值是$v[i]$，一共有$c[i]$件。求解将哪些物品装入背包可使这些物品的重量总和不超过背包容量，且价值总和最大，并且满足某一分组要求

[# 19643] 分组背包

题目描述

给定N组物品，其中第i组物品有n_i个物品。第i组的第j个物品的重量为$w_{i,j}$，价值为$v_{i,j}$。有一容积为M的背包，要求选择若干个物品放入背包，使得每组至多选择一个物品并且物品总体积不超过的前提下，物品的价值总和最大。

输入格式

第一行输入$N, M(1 \leq N, M \leq 100)$，分别代表物品组数$N$和背包容量$M$。

后面N组物品数据，每组数据第一行一个$n_i(1 \leq n_i \leq 100)$，表示每组物品的数量，接着n_i行，每行两个数$w_{i,j}, v_{i,j}(1 \leq w_{i,j}, v_{i,j} \leq 100)$，分别代表物品的重量和价值。

输出格式

一行一个整数，每组至多选择一个物品并且物品总体积不超过V的前提下，物品的最大价值总和

样例输入

```
3 10
3
3 2
2 3
2 2
2
5 4
6 4
3
1 2
2 1
4 3
```

样例输出

```
9
```

Solution

回忆我们0 - 1背包的状态设计和转移方程，$\mathrm{dp}[i][j]$表示前i**件**物品，重量总和为j时的最大价值

$$\mathrm{dp}[i][j] = \max(\mathrm{dp}[i-1][j], \mathrm{dp}[i-1][j-w] + v)$$

我们在转移到第i个物品的时候，通过使用$i-1$这一维度，保证了我们不会**重复选取**第i件物品

那么稍微拓展一下，我们将$\mathrm{dp}[i][j]$改为前i**组**物品，重量总和为j时的最大价值

那么我们仍然能得到类似的转移方程：

$$\mathrm{dp}[i][j]=\max(\mathrm{dp}[i-1][j],\mathrm{dp}[i-1][j-w_j]+v_j)$$

其中，前式代表不使用第i组物品，而后式代表使用第i组物品的第j个，后式是由$\mathrm{dp}[i-1]$这一维转移过来，保证了**没有选取任何第i组的物品**

Code

```
void solve() {
    int n, m; cin >> n >> m;
    vector<int> dp(m + 1);

    while(n--) {
        int nn; cin >> nn;
        vector<pair<int, int>> a(nn);
        for (auto& [w, v]: a) cin >> w >> v;
        auto tmp = dp;
        for (auto [w, v]: a) for (int i = m; i >= w; i--)
            cmax(tmp[i], dp[i - w] + v);
        dp = tmp;
    }

    cout << dp[m] << endl;
}
```

混合背包问题

混合背包问题是背包内物品涵盖**多种**0 - 1 / 完全 / 多重 / 分组背包的统称

[# 19635] 混合背包问题

题目描述

有N类物品和一个容量为M的背包，每类物品的体积为w_i，价格为v_i，数量为c_i（$c_i=0$代表该物品只能用次，$c_i=-1$代表该物品可以使用无数次），问如何装取物品使得装入背包中的物品总价格最高?

输入格式

第一行两个数$N,M(1\le N,M\le 1000)$和，分别表示物品种类数，背包容量

接下来行，每行三个用空格隔开的整数w_i,v_i,c_i，分别表示该类物品的体积，价格，数量

输出格式

最优装取背包容量M后的最大价格

样例输

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

样例输出

```
8
```

Solution

对于数量为1的物品，我们可以将其视为**特殊的**多重背包进行处理

而对于数量为无穷的物品，我们注意到至多只能选择$c = \frac{m}{w}$项，因此我们也可以视为容量为c的多重背包进行处理

Code

```
void solve() {
    int n, m; cin >> n >> m;
    vector<pair<int, int>> a;
    for (int i = 0; i < n; i++) {
        int w, v, c, t = 1; cin >> w >> v >> c;
        if (c == 0) c = m / w;
        else if (c == -1) c = 1;

        while(c) {
            if (c <= t) {
                a.emplace_back(w * c, v * c);
                break;
            }
            a.emplace_back(w * t, v * t);
            c -= t; t <<= 1;
        }
    }

    vector<ll> dp(m + 1);
    for (auto [w, v]: a) for (int j = m; j >= w; j--)
        cmax(dp[j], dp[j - w] + v);

    cout << dp[m] << endl;
}
```

二维费用背包

在普通的背包问题种，我们物品通常只有两个属性，一个**限制属性**和一个**价值属性**，而在二维费用背包中，物品通常有**两个限制属性**，更明确一点的描述如下：

给定若干种物品，每种物品都有自己的重量w_i、体积v_i和价格p_i，在限定的总重量和总体积内，使得选取的物品的总价格最高。

[# 19693] 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：$N(1 \leq N \leq 1000)$，$M(1 \leq M \leq 500), K(1 \leq K \leq 100)$，分别代表小智的精灵球数量、皮卡丘初始的体力值、野生宝可梦的数量。

之后的K行，每一行代表一个野生宝可梦，包括两个整数：收服该宝可梦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个宝可梦，以及收服C个宝可梦时皮卡丘的剩余体力值最多为R。

样例输入

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

样例输出

```
3 30
```

Solution

首先，由于每个小精灵只会被收服**一次**，因此我们考虑使用0 - 1背包模型

回忆0 - 1背包的状态设计和转移方程，我们不难想出该题的状态设计，$\mathrm{dp}[i][j][k]$表示收服前i个小精灵的过程中，使用了j个精灵球，对皮卡丘造成了k的伤害时，最多收服的小精灵数量

而转移方程也比较好想，分为两种情况：收服了第i个小精灵和没收服第i个小精灵，具体如下所示：

$$\mathrm{dp}[i][j][k] = \max(\mathrm{dp}[i-1][j][k], \mathrm{dp}[i-1][j-a_i][k-b_i])$$

其中，a_i, b_i分别是收服第i个小精灵需要的精灵球数量和对皮卡丘造成的伤害

最后在输出答案时，我们**顺序**枚举对皮卡丘的伤害，找到首次收服小精灵数量最多的$\mathrm{dp}[n][m][j]$输出即可

特别的，对于**二维费用**背包问题，我们一般会使用滚动数组，或者将数组压缩至**二维**进行空间上的优化

Code

```
void solve(){
    int n, m, o; cin >> m >> o >> n;
    vector a(n, make_pair(0, 0));
    for (auto& [v1, v2]: a) cin >> v1 >> v2;

    vector dp(m + 1, vector(o + 1, 0));
    for (auto [vm, vo]: a) for (int i = m; i >= vm; i--)
    for (int j = o; j >= vo; j--) {
        cmax(dp[i][j], dp[i - vm][j - vo] + 1);
    }

    int maxx = *max_element(all(dp[m]));
    for (int i = 0; i <= o; i++) if (dp[m][i] == maxx) {
        cout << maxx << ' ' << o - i << endl;
        return;
    }
}
```

[# 7085] 潜水员

题目描述

潜水员有一定数量的气缸，每个气缸都有重量和气体容量。气缸同时带有2种气体：一种为氧气，一种为氮气。潜水员为了完成潜水工作需要一定数量的氧气和氮气，允许他带多个气缸完成工作。 他所需气缸的总重量最少是多少？如果不能恰好带这么多气体，允许多带一些（先考虑**多带的氧气最小**，再考虑多带的氮气最小）。

输入格式

输入第一行有3整数$u, v(1 \le u, v \le 1000), n(1 \le n \le 100)$，$u, v$表示氧，氮各自需要的量，$n$表示气缸的个数。

此后的n行，每行包括$a_i, b_i(1 \le a_i \le 100), c_i(1 \le c_i \le 800)$，分别表示第$i$个气缸里的氧和氮的容量及汽缸重量。

输出格式

输出需要带的气缸的最小重量。如果无法凑齐，输出`-1`

样例输入

```
5 5 9
1 1 12
3 1 52
1 3 71
2 1 33
3 2 86
2 3 91
2 2 43
3 3 113
1 2 28
```

样例输出

```
104
```

Solution

首先，由于每个气缸只会被选取**一次**，因此我们考虑使用0 - 1背包模型

对于本题而言，是个0 - 1背包**最小化**费用的问题，因此我们不难设计出状态$\mathrm{dp}[i][j][k]$，代表使用前i个气缸，一共有j的氧气和k的氮气，所选取气缸总重量的最小值，转移方程如下：

$$\mathrm{dp}[i][j][k] = \min(\mathrm{dp}[i-1][j][k], dp[i-1][j-a_i][k-b_i] + w_i)$$

其中，前式代表我们**不使用**第i个气缸所达到的最小重量，而后式代表我们**使用**第i个气缸所达到的最小重量

由于本题在最小化重量的基础上，额外要求**多的氧气最少的情况下，多的氮气最少**，因此，我们在确定j, k的枚举范围时，就不是单纯的$[a_i, u]$和$[b_i, v]$了，我们枚举的上限要分别加上气缸中氧气的**最大值**和氮气的**最大值**，即$[a_i, u + \max(a)]$和$[b_i, v + max(b)]$

最后，我们根据题目要求，枚举氧气的氮气的容量，并输出相应的$\mathrm{dp}[i][j]$即可

Code

```cpp
void solve() {
    int u, v, n; cin >> u >> v >> n;
    vector<tuple<int, int, int>> a(n);
    for (auto& [x, y, v]: a) cin >> x >> y >> v;

    int lx = u + 110, ly = v + 110;
```

```
    vector dp(lx + 1, vector(ly + 1, inf));
    dp[0][0] = 0;
    for (auto& [x, y, w]: a)
        for (int i = lx; i >= x; i--) for (int j = ly; j >= y; j--) {
            cmin(dp[i][j], dp[i - x][j - y] + w);
        }

    for (int i = u; i <= lx; i++) for (int j = v; j <= ly; j++)
        if (dp[i][j] != inf) {cout << dp[i][j] << endl; return;}

    cout << -1 << endl;
}
```