南京工業大學
NANJING TECH UNIVERSITY

# ICPC Template Manual

**作者: 贺梦杰**

September 18, 2019

# Contents

# Chapter 1

# 基础

## 1.1 测试

# Chapter 2

# 搜索

# Chapter 3

# 动态规划

# 3.1 背包 DP

## 3.1.1 0-1 背包洛谷 P1048 采药

### 3.1.1.1 题目描述

有 $n(1 \le n \le 100)$ 株药草，每一株有一个采摘时长 $a_i$ 和价值 $b_i$ $(1 \le a_i, b_i \le 100)$，给你 $m(1 \le m \le 1000)$ 时间，问最大价值是多少?

### 3.1.1.2 代码

```cpp
#include <bits/stdc++.h>

using namespace std;
const int N = 1e5 + 10;

int n, m;            // 物品数量，最大容量
int f[N];            // f[i]表示用i容量能得到的最大价值
int c[110], v[110];  // 每一个物品的消耗和价值

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int i, j;
    cin >> m >> n;
    for (i = 1; i <= n; i++)
        cin >> c[i] >> v[i];

    // 对于每个物品i，恰好使用j容量时，不取或取a[i]
    // 注意j要倒着更新
    for (i = 1; i <= n; i++)
        for (j = m; j >= c[i]; j--)
            f[j] = max(f[j], f[j - c[i]] + v[i]);

    int ans = 0;
    for (i = 1; i <= m; i++)
        ans = max(ans, f[i]);

    cout << ans << endl;

    return 0;
}
```

## 3.1.2 完全背包洛谷 P1616 疯狂的采药

### 3.1.2.1 题目描述

与上一题描述差不多，但此题和原题的不同点：
1. 每种草药可以无限制地疯狂采摘。
2. 药的种类眼花缭乱，采药时间好长好长啊！
另外，物品数 $n(1 \le n \le 10000)$，最大容量 $m(1 \le m \le 100000)$，物品消耗和价值 $(1 \le a_i, b_i \le 10000)$，并且 $n * m \le 10^7$

### 3.1.2.2 代码 1，直接枚举每种取的数量

大概率 TLE，但还是要放在这，因为有时候可能数据范围不大，但却有别的限制

```cpp
#include <bits/stdc++.h>

using namespace std;
const int N = 1e4 + 10, M = 1e5 + 10;

int n, m;        // 物品数量，最大容量
int f[M];        // f[i]表示用i容量能得到的最大价值
int c[N], v[N];  // 每一个物品的消耗和价值
```

```

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int i, j, k;
    cin >> m >> n;
    for (i = 1; i <= n; i++)
        cin >> c[i] >> v[i];

    // 对于每个物品i，恰好使用j容量时，取k个a[i]
    // 注意j要倒着更新
    for (i = 1; i <= n; i++)
        for (j = m; j >= c[i]; j--)
            for (k = 0; k <= j / c[i]; k++)
                f[j] = max(f[j], f[j - k * c[i]] + k * v[i]);

    int ans = 0;
    for (i = 1; i <= m; i++)
        ans = max(ans, f[i]);

    cout << ans << endl;

    return 0;
}
```

#### 3.1.2.3 代码 2，巧妙地用无限取这个条件

由于可以无限取，所以其实不用关心先后更新

```
// 对于每个物品i，恰好使用j容量时，取k个a[i]
// 注意j的更新顺序，非常妙
for (i = 1; i <= n; i++)
    for (j = c[i]; j <= m; j++)
        f[j] = max(f[j], f[j - c[i]] + v[i]);
}
```

### 3.1.3 分组背包

#### 3.1.3.1 问题描述

有 N 件物品和一个容量为 V 的背包。第 i 件物品的费用是 c[i]，价值是 w[i]。这些物品被划分为若干组，每组中的物品互相冲突，最多选一件。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

#### 3.1.3.2 解决方案

状态空间：f[k][v] 表示前 k 组物品花费费用 v 能取得的最大权值
状态转移方程：
f[k][v]=maxf[k-1][v],f[k-1][v-c[i]]+w[i]| 物品 i 属于第 k 组
使用一维数组的伪代码如下：

```
for 所有的组k
    for v=V..0
        for 所有的i属于组k
            f[v]=max{f[v],f[v-c[i]]+w[i]}
```

### 3.1.4 多重背包

#### 3.1.4.1 问题描述

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 vi，价值是 wi。
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。

#### 3.1.4.2 解法 1：直接枚举

与完全背包的暴力解法相似，只要加上 k<=s[i] 这个条件

#### 3.1.4.3 解法 2：二进制优化

```cpp
#include <bits/stdc++.h>

using namespace std;
const int N = 16 * 1e3 + 10, M = 2e3 + 10;

int n, m;               // 物品数量，最大容量
int f[M];               // f[i]表示用i容量能得到的最大价值
int s[N], v[N], w[N];   // 每一个物品的个数、消耗和价值

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int i, j, cnt;
    cin >> n >> m;
    for (i = 1; i <= n; i++)
        cin >> v[i] >> w[i] >> s[i];

    // 二进制优化
    cnt = n;
    for (i = 1; i <= n; i++) {
        int t = s[i], d = 1;
        while (t >= d)
            v[++cnt] = v[i] * d, w[cnt] = w[i] * d, t -= d, d <<= 1;
        if (t)
            v[++cnt] = v[i] * t, w[cnt] = w[i] * t;
    }

    // 0-1背包DP
    // 注意i要从n+1开始，因为前n个是未拆分的，如果算上那么每个物品的数量就被错误的乘2了
    for (i = n + 1; i <= cnt; i++)
        for (j = m; j >= v[i]; j--)
            f[j] = max(f[j], f[j - v[i]] + w[i]);

    int ans = 0;
    for (i = 1; i <= m; i++)
        ans = max(ans, f[i]);

    cout << ans << endl;

    return 0;
}
```

#### 3.1.4.4 解法 3：单调队列优化

```cpp
#include <bits/stdc++.h>

using namespace std;
const int N = 1e3 + 10, M = 2e4 + 10;

int n, m;               // 物品数量，最大容量
int f[M];               // f[i]表示用i容量能得到的最大价值
int s[N], v[N], w[N];   // 每一个物品的个数、消耗和价值

struct CycQue {
    const static int MAXSZ = M;
    typedef pair<int, int> Ele;
    Ele q[MAXSZ];
    int head, tail, sz;
```

```
    CycQue()
        : head(1), tail(0), sz(0) {}
    void clear() {
        head = 1, tail = 0, sz = 0;
    }
    bool empty() {
        return !sz;
    }
    void push_back(Ele x) {
        // 队列满了
        if (sz >= MAXSZ - 1)
            sz = sz / (sz - sz);  // 除0，抛出异常
        tail = (tail + 1) % MAXSZ, q[tail] = x, ++sz;
    }
    void push_front(Ele x) {
        if (sz >= MAXSZ - 1)
            sz = sz / (sz - sz);
        head = (head - 1 + MAXSZ) % MAXSZ, q[head] = x, ++sz;
    }
    void pop_front() {
        if (!sz)
            sz = sz / (sz - sz);
        head = (head + 1) % MAXSZ, --sz;
    }
    void pop_back() {
        if (!sz)
            sz = sz / (sz - sz);
        tail = (tail - 1 + MAXSZ) % MAXSZ, --sz;
    }
    Ele front() {
        if (!sz)
            sz = sz / (sz - sz);
        return q[head];
    }
    Ele back() {
        if (!sz)
            sz = sz / (sz - sz);
        return q[tail];
    }
} q;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int i, j, b;
    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; i++)
        scanf("%d%d%d", &v[i], &w[i], &s[i]);

    // 枚举物品
    for (i = 1; i <= n; i++) {
        // 枚举余数
        for (b = 0; b < v[i]; b++) {
            q.clear();
            // 枚举当前取的个数
            for (j = 0; j <= (m - b) / v[i]; j++) {
                int t = f[b + j * v[i]] - j * w[i];
                // 入队
                while (!q.empty() && q.back().second <= t)
                    q.pop_back();
                q.push_back({j, t});
                while (j - q.front().first > s[i])
                    q.pop_front();
                f[b + j * v[i]] = q.front().second + j * w[i];
            }
```

```
81          }
82      }
83
84      int ans = 0;
85      for (i = 1; i <= m; i++)
86          ans = max(ans, f[i]);
87
88      printf("%d\n", ans);
89
90      return 0;
91  }
```

### 3.1.5 杭电多校 2019 第一场 C HDU6580 Milk

#### 3.1.5.1 题目描述

有一个 n*m 的矩阵，左右可以随便走，但只能在每一行的中点往下走，每走一格花费时间 1. 现在这个矩阵里放了 k 瓶牛奶，第 i 个牛奶喝下去需要 ti 时间起点是（1，1）对于每个 i [1,k], 问喝掉 k 瓶牛奶花费的最小时间

#### 3.1.5.2 解决方案

首先对瓶子的横坐标离散化处理，一行一行地更新答案。每到新的一行，先预处理出向左（右）走喝了 $i$ 瓶水后，回到原点以及不回到原点所需要的最短时间，分别记为 $l, r, l\_back, r\_back$。然后将左右两边合并，求出喝了 $i$ 瓶水后，回到或不回到原点所需时间，记为 $g, g\_back$。设 $f[i]$ 为从 $(1,1)$ 出发，喝了 $i$ 瓶水后回到当前行中点所需的最短时间，则这时就可以根据 $g$ 来和之前的 $f[i]$ 来更新答案，并用 $g\_back$ 来更新 $f$ 的值。时间复杂度为 $O(k^2)$

### 3.1.6 ICPC 上海 2019 网络预选赛 J-Stone game

#### 3.1.6.1 题目描述

有 n 个石头，每个石头都有一个重量为 $a_i$。现要取出一部分，我们设它们为集合 A，则剩余部分为集合 B，要求重量 $A \geq B$，并且当移去 A 中的任意一个石头 $t$ 时，要求 $(A-t) \leq B$。问取法有多少种，最后输出模 $1e9+7$ 其中 $n \leq 300, a_i l \leq 500$，样例数 $T \leq 100$

#### 3.1.6.2 解决方案

容易发现，当划分出 A 之后，只要移除 A 中**最轻的石头 t** 是满足题意的这种方案就是可行的。
于是我们可以先将**数组排序，然后枚举 t**，对每一个 t 分别考虑。最暴力的想法是考虑 t 后面的每一个 $a_i$ 取或不取，但是这样复杂度上天。
再仔细想想，就会发现所有 $a_i$ 加起来也就 $1.5e5$，也就是可以用**背包 dp**，$f(i,j)$ 表示 $t=i$ 时总重量为 j 的方案数。复杂度是？不考虑 T，枚举 t 为 $O(N)$，对于每个 t，考虑其后的每个数取或不取为 $O(N*1.5e5)$，总复杂度为 $O(N^2*1.5*10^5)$，又上天了。
但是当我们**倒过来想**，**将数组从大到小排序**，然后从前向后枚举 t，就会发现复杂度降到了 $O(N*1.5*10^5)$。每次 $++t$，都相当于整体加上 $a_t$（$a_t$ 必取），然后再考虑不取 $a_{t-1}$ 的情况（$a_{t-1}$ 在上一个 t 中是必选的）。

## 3.2 位置 DP

这类 dp 一般都会以数字出现位置作为一个维度。

### 3.2.1 杭电多校 2019 第一场 A HDU6578 Blank

#### 3.2.1.1 题目描述

有 n ( 100) 个格子，向其中填入 0、1、2、3 这 4 个数，但是有 m ( 100) 个限制
限制 l r x： 表示 l r 的格子内不同的数的个数为 x
问一共有多少种填入方案?

#### 3.2.1.2 解决方案

**初步分析：**构建 dp[i][j][k][t][cur]，表示到 cur 位置为止，0,1,2,3 四个数最后出现在位置 i,j,k,t 的情况数。
dp[cur][j][k][t][cur]=dp[cur][j][k][t][cur]+dp[i][j][k][t][cur−1]
dp[i][cur][k][t][cur]=dp[i][cur][k][t][cur]+dp[i][j][k][t][cur−1]
dp[i][j][cur][t][cur]=dp[i][j][cur][t][cur]+dp[i][j][k][t][cur−1]
dp[i][j][k][cur][cur]=dp[i][j][k][cur][cur]+dp[i][j][k][t][cur−1]
**(注意以上求法是对每个待求 dp 分别求，等号后面第二项需要迭代)**
这样当 cur==r，只要依据四个数出现位置是否大于等于 l 就可以判断有几个不同的数，若不满足限制则令 dp 等于 0。

**优化 1:** 我们发现上面状态中一定会有一个 cur，即有一维度是不会变的，但是这一维度的位置一直在变。
那么就可以构建 dp[i][j][k][cur] ，其中 i < j < k < cur （若为 0 可等于）：只知道 i, j, k, cur 是不同的数最后出现的位置，且 i < j < k < cur （并不需要知道 i,j,k,cur 对应的填入数是哪个，因为这并不影响对限制的判断）
dp[j][k][cur−1][cur]=dp[j][k][cur−1][cur]+dp[i][j][k][cur−1]
dp[i][k][cur−1][cur]=dp[i][k][cur−1][cur]+dp[i][j][k][cur−1]
dp[i][j][cur−1][cur]=dp[i][j][cur−1][cur]+dp[i][j][k][cur−1]
dp[i][j][k][cur]=dp[i][j][k][cur]+dp[i][j][k][cur−1]
**(注意以上求法是对每一个已知 dp 去累加至待求 dp，因此不需要迭代**
分别表示将最后一次出现位置为 i, j, k ,cur-1 的数填入第 cur 个格子中

**优化 2:** 我们发现上面状态中最后一个维度要么为 cur 要么为 cur-1，即当前状态仅与上一个状态有关，所以可以用滚动数组优化空间。
构建 dp[i][j][k][2]，得到状态转移方程：
dp[j][k][cur−1][now]=dp[j][k][cur−1][now]+dp[i][j][k][pre]
dp[i][k][cur−1][now]=dp[i][k][cur−1][now]+dp[i][j][k][pre]
dp[i][j][cur−1][now]=dp[i][j][cur−1][now]+dp[i][j][k][pre]
dp[i][j][k][now]=dp[i][j][k][now]+dp[i][j][k][pre]
注意：因为这里是累加，所以用滚动数组时，dp[i][j][k][pre] 用完以后要清空!

### 3.2.2 杭电多校 2019 第一场 J HDU6587 Kingdom

#### 3.2.2.1 题目描述

给出 n(<=100) 个点的树的前序遍历和中序遍历，这两个序列中某些编号未知 (用 0 来表示)。问有多少种合法的树。

#### 3.2.2.2 解决方案

位置 DP，二叉树前序和中序遍历的关系

最关键的不是 dp 的状态和方程，而是限制，**要求左子树和右子树的节点必须在对应的区间内（即不交叉，不出界）**
**记录 a 中节点在 b 中出现的位置，b 中节点在 a 中出现的位置。**
**在每进入一个新的区间时，都检查一遍，是否 a 的区间中所有节点都在 b 的对应区间中，同理反过来再做一次**

## 3.3 树形动态规划

### 3.3.1 加分二叉树

#### 3.3.1.1 问题描述

设一个 n 个节点的二叉树 tree 的中序遍历为（l,2,3,…,n），其中数字 1,2,3,…,n 为节点编号。每个节点都有一个分数（均为正整数），记第 i 个节点的分数为 di，tree 及它的每个子树都有一个加分，任一棵子树 subtree（也包含 tree 本身）的加分计算方法如下：
subtree 的左子树的加分 × subtree 的右子树的加分+ subtree 的根的分数
若某个子树为空，规定其加分为 1，叶子的加分就是叶节点本身的分数。不考虑它的空子树。
试求一棵符合中序遍历为（1,2,3,…,n）且加分最高的二叉树 tree。要求输出；
（1）tree 的最高加分
（2）tree 的前序遍历

#### 3.3.1.2 输入格式

第 1 行：一个整数 n（n < 30），为节点个数。
第 2 行：n 个用空格隔开的整数，为每个节点的分数（分数< 100）。

#### 3.3.1.3 输出格式

第 1 行：一个整数，为最高加分（结果不会超过 4,000,000,000）。
第 2 行：n 个用空格隔开的整数，为该树的前序遍历。

#### 3.3.1.4 思路

这道题看上去是树形 DP，但仔细思考，我们发现很难依据中序遍历建出树。但中序遍历有其独特之处，即一旦根被确定，则左右子树也被确定。
所以我们应该用区间 DP 来解决这道题。
$f(i,j)$: 选 i 到 j 的节点作为一颗子树最大的得分

$$f(i,j) = \max_{i \le k \le j} \{f(i,k-1) * f(k+1,j) + f(k,k)\}$$

由题意，当 $i > j$，$f(i,j) = 1$ 为空节点。
前序遍历就是重新用 dfs 走一遍：每次找到使 $f(i,j)$ 最大的 $k$，然后以 $k$ 为分界向左右两边递归。

### 3.3.2 洛谷 P2015 二叉苹果树

#### 3.3.2.1 问题描述

有一棵苹果树，如果树枝有分叉，一定是分 2 叉（就是说没有只有 1 个儿子的结点）这棵树共有 N 个结点（叶子点或者树枝分叉点），编号为 1-N, 树根编号一定是 1。我们用一根树枝两端连接的结点的编号来描述一根树枝的位置。现在这颗树枝条太多了，需要剪枝。但是一些树枝上长有苹果。
给定需要保留的树枝数量，求出最多能留住多少苹果。

#### 3.3.2.2 输入格式

第 1 行 2 个数，N 和 Q(1<=Q<= N,1<N<=100)。
N 表示树的结点数，Q 表示要保留的树枝数量。接下来 N-1 行描述树枝的信息。
每行 3 个整数，前两个是它连接的结点的编号。第 3 个数是这根树枝上苹果的数量。
每根树枝上的苹果不超过 30000 个。

#### 3.3.2.3 输出格式

剩余苹果的最大数量。

#### 3.3.2.4 思路

$f(i,j)$ 表示以 i 为节点的根保留 k 条边的最大值接下来对每一个节点分类讨论，共三种情况：
1. 全选左子树
2. 全选右子树
3. 左右子树都有一部分

为了方便，我们设左儿子为 ls，右儿子为 rs，连接左儿子的边为 le，连接右儿子的边为 re

$$f(i,j)=\max\{f(ls,j-1)+le,f(rs,j-1)+re,\max_{0\le k\le j-2}\{f(ls,k)+f(rs,j-2-k)\}+le+re\}$$

### 3.3.3 最大利润

#### 3.3.3.1 问题描述

政府邀请了你在火车站开饭店，但不允许同时在两个相连接的火车站开。任意两个火车站有且只有一条路径，每个火车站最多有 50 个和它相连接的火车站。
告诉你每个火车站的利润，问你可以获得的最大利润为多少。
最佳投资方案是在 1，2，5，6 这 4 个火车站开饭店可以获得利润为 90

#### 3.3.3.2 输入格式

第一行输入整数 N(<=100000)，表示有 N 个火车站，分别用 1，2。。。，N 来编号。接下来 N 行，每行一个整数表示每个站点的利润，接下来 N-1 行描述火车站网络，每行两个整数，表示相连接的两个站点。

#### 3.3.3.3 输出格式

输出一个整数表示可以获得的最大利润。

#### 3.3.3.4 思路

这道题虽然是多叉树，但状态仍然是比较简单的。
对于某个结点，如果选择该节点，则该节点的所有儿子都不能选，如果不选该节点，则它的儿子可选可不选。
所以，我们令 $f(i)$ 表示以 i 节点为根的子树中选 i 的最大利润，$h(i)$ 表示以 i 节点为根的子树中不选 i 的最大利润。a[i] 是 i 本身的利润，j 是 i 的儿子

$$f(i)=a[i]+\sum_j h(j)$$

$$h(i)=\sum_j \max\{f(j),h(j)\}$$

### 3.3.4 洛谷 P2014 选课

#### 3.3.4.1 问题描述

学校实行学分制。每门的必修课都有固定的学分，同时还必须获得相应的选修课程学分。学校开设了 N（N<300）门的选修课程，每个学生可选课程的数量 M 是给定的。学生选修了这 M 门课并考核通过就能获得相应的学分。
在选修课程中，有些课程可以直接选修，有些课程需要一定的基础知识，必须在选了其它的一些课程的基础上才能选修。例如《Frontpage》必须在选修了《Windows 操作基础》之后才能选修。我们称《Windows 操作基础》是《Frontpage》的先修课。每门课的直接先修课最多只有一门。两门课也可能存在相同的先修课。每门课都有一个课号，依次为 1，2，3，…。
你的任务是为自己确定一个选课方案，使得你能得到的学分最多，并且必须满足先修课优先的原则。假定课程之间不存在时间上的冲突。

#### 3.3.4.2 输入格式

输入文件的第一行包括两个整数 N、M（中间用一个空格隔开），其中 $1\le N\le 300, 1\le M\le N$。以下 N 行每行代表一门课。课号依次为 1，2，…，N。每行有两个数（用一个空格隔开），第一个数为这门课先修课的课号（若不存在先修课则该项为 0），第二个数为这门课的学分。学分是不超过 10 的正整数。

#### 3.3.4.3 输出格式

只有一个数：实际所选课程的学分总数。

#### 3.3.4.4 思路一

分析一下，似乎也不是很难，对于某个节点，只有选择它，才能选择它的儿子们。
令 $f(x,i)$ 表示在以 x 结点为根的子树中选择 i 个点的最大学分。$f(x,1)=s[x]$，s[x] 是课程 x 本身的学分。
假设结点 x 有 k 个儿子，标号为 $y_1,y_2,..,y_k$，让他们分别选 $i_1,i_2,..,i_k$ 门课程，那么状态转移方程似乎是..

$$f(x,i)=s[x]+\max_{i_1+i_2+..+i_k=i-1}\sum_{1\le j\le k}f(y_j,i_j)$$

好像.. 写不出这样的循环啊，当然，如果用 dfs 强行写也不是不可以，但是时间复杂度必炸。

这时候我们要用一种类似**前缀和**的思维
即，我们每 dfs 完一棵子树，都进行一次完整的 dp 更新。假设当前根结点为 x，我们刚 dfs 完它的一棵子树 y，那么我们进行如下更新（对所有 i）：

$$f(x,i)=\max_{1\le j\le i}\{f(x,j)+f(y,i-j)\}$$

**此时 $f(x,i)$ 表示在以 x 结点为根的子树中已经被 dfs 过的部分中选 i 个点的最大学分，而每次更新就像是把一棵新子树添加到答案中。**
除此以外，还有一点要注意，就是我们要让 **i 递减更新**，因为大的 i 要用到之前小的 i，而小的 i 不要用到大的 i

#### 3.3.4.5 思路二

先将**森林转二叉树（左孩子右兄弟）**。如何转呢？设 D[x]、c[x] 和 b[x] 分别表示 x 结点的父亲、左孩子和右兄弟，对于每个节点，D[x] 是已知的，所以 **b[x]=c[D[x]],c[D[x]]=x**。
再分析，对于某个节点，如果要选其左孩子，则必须选它，而选右孩子则没有限制。
令 $f(x,i)$ 表示在以 x 结点为根的子树中选择 i 个点的最大学分。$f(x,1)=s[x]$，s[x] 是课程 x 本身的学分。
还是沿用上面**前缀和思想**，先由左孩子 (ls) 更新，再由右孩子 (rs) **倒着**更新，两次的方程如下：

$$f(x,i)=f(x,1)+f(ls,i-1)$$

$$f(x,i)=\max_{0\le j\le i}f(x,j)+f(rs,i-j)$$

### 3.3.5 HYSBZ 2427 软件安装

#### 3.3.5.1 题目描述

现在我们的手头有 N 个软件，对于一个软件 i，它要占用 Wi 的磁盘空间，它的价值为 Vi。我们希望从中选择一些软件安装到一台磁盘容量为 M 的计算机上，使得这些软件的价值尽可能大（即 Vi 的和最大）。
但是现在有个问题：软件之间存在依赖关系，即软件 i 只有在安装了软件 j（包括软件 j 的直接或间接依赖）的情况下才能正确工作（软件 i 依赖软件 j）。幸运的是，一个软件最多依赖另外一个软件。如果一个软件不能正常工作，那么他能够发挥的作用为 0。
我们现在知道了软件之间的依赖关系：软件 i 依赖 Di。现在请你设计出一种方案，安装价值尽量大的软件。一个软件只能被安装一次，如果一个软件没有依赖则 Di=0，这是只要这个软件安装了，它就能正常工作。
subsubsection 输入格式 第 1 行：N,M（0<=N<=100,0<=M<=500）第 2 行：W1,W2, …Wi, …,Wn 第 3 行：V1,V2, …Vi, …,Vn 第 4 行：D1,D2, …Di, …,Dn

#### 3.3.5.2 输出格式

一个整数，代表最大价值。

#### 3.3.5.3 思路

**注意，此图可能有环**
仔细读题，题中说一个软件只依赖最多一个软件，所以在一个联通分之内最多只有一个环，并且该环可以指向环以外的节点，而环以外的节点不会指向环。
因此，我们可以**把环当做一个新节点**来处理，这个新节点所占空间和价值都为环内元素加和。
怎么确定环及环内元素呢？我推荐**着色法（有向图为黑白灰，无向图为黑白）**。
有向图中白色为未探索的点，灰色为正在探索的点，黑色为已经探索过的并且确定不成环的点。若在探索过程中遇到灰点，则说明成环。
无向图中白色为未探索的点，黑色为已经探索过和正在探索的点。若在探索过程中遇到黑点，则说明成环。
处理完成后，DP 思路同”洛谷 *P2014* 选课”

### 3.3.6 CF486D Valid Sets

#### 3.3.6.1 题目描述

给定一棵树，每个点都有一个权，现在要你选择一个连通块，问有多少种选择方法，使得连通块中的最大点权和最小点权的差值小于等于 d。答案对 1e9+7 取模。

#### 3.3.6.2 输入格式

The first line contains two space-separated integers d (0 d 2000) and n (1 n 2000).
The second line contains n space-separated positive integers a1, a2, ..., an(1 ai 2000).
Then the next n - 1 line each contain pair of integers u and v (1 u, v n) denoting that there is an edge between u and v. It is guaranteed that these edges form a tree.

#### 3.3.6.3 输出格式

Print the number of valid sets modulo 1000000007.

#### 3.3.6.4 思路

我起初的错误想法是，设 $f(x,i)$ 和 $g(x,i)$ 分别为以 x 为根的子树中最大值和最小值为 i 的方案数，企图通过对 g 和 f 的运算得出答案，但这是错误的，主要原因是 f 和 g 无法把范围限定住。
受到前面以子树为对象思考的影响，形成了**思维惯性**，这里我们并不是以子树为对象，在一遍 dfs 里算出所有答案。
而是**枚举**，枚举每一点 x，将 x 作为连通块的最大值，以 x 为根进行 dfs，求方案数。
另外要注意：因为点权可能相同，所以为了避免重复计算，我们需要定序，若权值相同，让编号大的点访问编号小的，而编号小的不能访问大的。

### 3.3.7 CF294E Shaass the Great

#### 3.3.7.1 题目描述

给一颗带边权的树，让你选一条边删除，然后在得到的两个子树中各选一个点，用原来被删除的边连起来，重新拼成一棵树。使得这棵树的所有点对的距离总和最小。

#### 3.3.7.2 输入格式

The first line of the input contains an integer n denoting the number of cities in the empire, ($2 \leq n \leq 5000$). The next n - 1 lines each contains three integers ai, bi and wi showing that two cities ai and bi are connected using a road of length wi, ($1 \leq a_i, b_i \leq n, a_i \neq b_i, 1 \leq w_i \leq 106$).

#### 3.3.7.3 输出格式

所有点对距离总和最优解。

#### 3.3.7.4 思路

设我们要删除的边为 e，以 e 为分界树被分割成了左右两个部分 (我们称为左树和右树)，最后我们连接的两个点为 vl 和 vr
则答案为：
左树内点对距离总和 + 右树内点对距离总和
+ 左树中所有点到 vl 的距离总和 * 右树中点的个数 + 右树中所有点到 vr 的距离总和 * 左树中点的个数
+ e 的权重 * 左树中点的个数 * 右树中点的个数

再观察一下，其实在删除 e 的情况下左右树内点对距离总和、左右树中的点的个数、e 的权重都是不变的，变化的只有左右树中所有点到 vl 和 vr 的距离总和
由此我们知道了，在删除 e 的情况下，答案最优的条件即为**找到 vl 和 vr，使得左树中所有点到 vl 的距离总和最小，右树中所有点到 vr 的距离总和最小**
于是我们知道了，要得到答案，就是要求一棵树的三个值：树内点的个数、树上所有点到某一点的距离总和的最小值、树内所有点对距离总和
树内点的个数最简单，不多说了，而树内所有点对的距离和也可以由树上所有点到某一点的距离和算得 (即树上所有点到**每**一点的和除以 2)

下面主要考虑如何在 $O(n)$ 的时间内求出树上所有点到每一点的距离：

**考虑 dfs，结果发现一遍 dfs 无论如何都不可能得到，但是可以知道以某一点 x 为根的子树上所有点到根 x 的距离总和**
**在第一遍 dfs 的基础上，我们再进行一次 dfs，这次 dfs 我们不再是自底向上更新，而是自顶向下更新，更新父节点及其以上所有的点到 x 的距离总和**
**可以理解为，第一次 dfs 我们得到了 x 以下的所有点到 x 的距离总和，第二次 dfs 我们得到了 x 以上的所有点到 x 的距离总和**

对于第二次 dfs，例如我们现在在 a 节点上，即将去往 x 结点，我们需要下传给 x 节点的距离有哪些呢?
1.a 结点上面传下来的距离
2.a 结点上除了 x 分支所有旁路上的距离 (由第一次 dfs 可以算出)
3.a 与 x 的边权

所以最后的算法是：**枚举每一条边**，运用以上算法，求出最小值