最大红矩形

时间限制：10 sec

空间限制：256 MB

问题描述

有一个 n*m 的棋盘，棋盘上的每个点都是红的**或**绿的。

你需要找出一个面积最大的矩形区域，使得其中**没有**绿的格子。

输入格式

第一行 2 个正整数 n,m，描述棋盘尺寸。

接下来 n 行描述这个棋盘，每行 m 个字符，每个字符为 . 或 X，其中 . 表示这个位置是红色的，X 表示这个位置是绿色的。

输出格式

一行一个整数，表示最大面积。

样例输入

```
4 5
.....
XXXXX
.X...
.....
```

样例输出

```
6
```

样例解释

以第 3 行第 3 列的方格为左上角，第 4 行第 5 列的方格为右下角的矩形区域是全红的矩形中面积最大的。

数据范围

对于 30% 的数据，n,m<=100；

对于 60% 的数据，n,m<=400；

对于 85% 的数据，n,m<=1,000；

对于 100% 的数据，n,m<=1,500。

提示

[这道题与“直方图最大面积”一题有什么关系呢？]

代码:

```
#include<iostream>
#include<stack>

using namespace std;

/*此题可用直方图最大面积的题目中的方法来计算最大红矩形面积
  即将此题中的数据构造为直方图，例如.X.
                                  ...
                                     ...
  则(0,0)的直方图高度为1,(0,1)为0,(1,0)为2,(1,1)为1(当前行'.'减去上一个'X'所在行数所形成的高度)
*/
int main()
{
    int n, m;
    cin >> n >> m;//读入行数和列数

    stack<int>* N_X = new stack<int>[m];
    /*新建一组堆栈，用于构造当前行中某列所构造的直方图的高度
      堆栈顶部存有当前列最近遇到的'X'字符所在的行数n'，当读入字符为'.'时，使用当前行数n减去n'即可得到用于构造当前行中某列所构造的直方图的高度
```

```
    */
    for (int i = 0; i < m; i++)
        N_X[i].push(-1);//栈的初始化，将-1压入栈(例如第0行高度为1，0-(-1)=1)
    stack<int> TEMP;//初始栈，用于上面所获得的存储直方图的高度

    char** matrix = new char*[n];//建立一个char型n*m的二维矩阵并初始化
    for (int i = 0; i < n; i++)
    {
        matrix[i] = new char[m];
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            cin >> matrix[i][j];//将数据读入二维矩阵中

            //'.'表示红，'X'表示绿
            if (matrix[i][j] == '.')//若遇到红点，则直接将该行该列的直方图的高度压入初始栈TEMP中
            {
                int height = i - N_X[j].top();//获得直方图高度
                TEMP.push(height);
            }
            else if (matrix[i][j] == 'X')//若遇到绿点，则将高度0压入栈中，并向N_X栈压入当前列中X所处的行数
            {
                TEMP.push(0);
                N_X[j].push(i);
            }
        }
        TEMP.push(0);/*当用于在下面中打印输出每一行的直方图高度，若无此句，则有可能会将多行的直方图高度加在一起进行输出
                                例如：.....
                                      XXXXX
                                      .X...
                                      .....
                                      由于栈是先进后出，故下面的代码会认为最大高度为8(即从第20个~13个元素均为直方图面积，高度为1)
                                      而正确的答案是6
                        */
    }

    //以下为原直方图最大面积中的源代码
    stack<int> H;//用于存储高度
```

```
    stack<int> N;//用于存储下标

    int maxRect = 0;
    int tempRect = 0;
    H.push(-1);//压入一个哨位节点，用于卡位
    int temp;//用于读取最初的数据
    int num = n * m + n;//由于每一行多一个0，所需读取的数目为n*m+n
    for (int k = 0; k < num; k = k)
    {
        if (N.empty() || H.top() <= TEMP.top())//若TEMP中栈顶所存在的直方图的高度比H中栈顶的高度小，说明可以压入栈，即点是普通点，非Low或High点（卡位点）
        {
            H.push(TEMP.top());
            N.push(k);
            TEMP.pop();
            k++;
        }
        else//若H.top() > TEMP.top()，说明遇到了卡位点（High点）,此时应将H和N中的元素弹出一个（是否弹出多个等待下一轮循环再进行检测）
        {
            temp = N.top();
            N.pop();

            if (!N.empty())
            {
                tempRect = (k - N.top() - 1)*H.top();//需要先弹出一个N中的栈顶元素然后再乘以H.top()，因为原先N中栈顶元素的左边可能存在应该加上的直方图面积
                                                     //具体原因请见附件Excel演示
            }
            else//如果弹出一个元素后为空，说明N原先栈顶中左边的直方图面积为H.top()*N.top()，即原先N栈顶元素左边直方图面积的长为N.top一直到0
            {
                tempRect = k * H.top();
            }
            H.pop();
            if (tempRect > maxRect)
                maxRect = tempRect;
        }
    }

    while (!N.empty())//当遍历完所有直方图时，需要再次清空N，若N中存在元素，则有可能所获得错误的直方图面积最大值
                      //以下代码类似于上面for循环中的else代码
    {
```

```
            temp = N.top();
            N.pop();

            if (!N.empty())
            {
                tempRect = (num - N.top() - 1)*H.top();//需要先弹出一个N中的栈顶元素然后再乘以H.top()，因为原先N中栈顶元素的左边可能存在应该加上的直方图面积
                                                       //具体原因请见附件Excel演示
            }
            else//如果弹出一个元素后为空，说明N原先栈顶中左边的直方图面积为H.top()*(N.top()-1)，即原先N栈顶元素左边直方图面积的长为N.top一直到0
            {
                tempRect = num * H.top();
            }
            H.pop();
            if (tempRect > maxRect)
                maxRect = tempRect;
    }
    cout << maxRect << endl;
    return 0;
}
```

数字盒子

问题描述

你有一个盒子，你可以往里面放数，也可以从里面取出数。

初始时，盒子是空的，你会依次做 Q 个操作，操作分为两类：

1. 插入操作: 询问盒子中是否存在数 x，如果**不存在**则把数 x 丢到盒子里。
2. 删除操作: 询问盒子中是否存在数 x，如果**存在**则取出 x。

对于每个操作，你需要输出是否成功插入或删除。

输入

第一行一个正整数 Q，表示操作个数。

接下来 Q 行依次描述每个操作。每行 2 个用空格隔开的非负整数 op,x 描述一个操作：op 表示操作类型，op=1 则表示这是一个插入操作，op=2 则表示这是一个删除操作；x 的意义与操作类型有关，具体见题目描述。

输出

按顺序对所有操作输出，对于每个操作输出一行，如果成功则输出“Succeeded”（不含引号），如果失败则输出“Failed”（不含引号）。

样例输入

```
6
1 100
1 100
2 100
1 200
2 100
2 200
```

样例输出

```
Succeeded
Failed
Succeeded
Succeeded
Failed
Succeeded
```

数据范围

对于 60% 的数据，保证 $x<10^5$。

对于 100% 的数据，保证 $x<10^{18}$，$Q\le 5*10^5$。

对于所有数据，保证 $op\in\{1,2\}$。

时间限制：10 sec

空间限制：256 MB

提示

[对于 x 较小的情况，我们只需要用数组记录每个数是否在盒子里即可。]

[对于 x 较大的情况，我们可不可以用什么方法把它们“变小”呢？可以想想哈希表哦！]

代码

```
#include<iostream>
#include<unordered_map>
#include<string>
#include<string.h>
using namespace std;

int main()
{
    unordered_map<string,int> HashMap_0;
    unordered_map<string, int> ::iterator iter;
    int n;
    cin >> n;
    string* str=new string[n];//用于存储输出的Succeeded或Failed
    int choic;//用于选择功能
    string key;
    for (int i = 0; i < n; i++)
    {
        cin >> choic;
        if (choic == 1)//向map中插入key
        {
            getline(cin, key);//获取输入的字符串
            iter = HashMap_0.find(key);//查找map中是否存在相同的key
            if (iter != HashMap_0.end())//若存在相同的key则不进行操作并返回Failed
            {
                str[i] = "Failed";
            }
            else//若不存在相同的key则放入并提示Succeeded
            {
                HashMap_0.emplace(key, 1);
                str[i] = "Succeeded";
            }
        }
        else if(choic == 2)//从map中删除key
        {
```

```
            getline(cin, key);//获取输入的字符串
            iter = HashMap_0.find(key);//查找map中是否存在相同的key
            if (iter != HashMap_0.end())//若存在相同的key则进行删除并返回Succeeded
            {
                HashMap_0.erase(iter);
                str[i] = "Succeeded";
            }
            else//若不存在相同的key则返回Failed
            {
                str[i] = "Failed";
            }
        }
    }

    for (int i = 0; i < n; i++)
    {
        cout << str[i] << endl;
    }
    return 0;
}
```

重编码

问题描述

有一篇文章，文章包含 nn 种单词，单词的编号从 11 至 nn，第 i 种单词的出现次数为 wiwi。
现在，我们要用一个 2 进制串（即只包含 0 或 1 的串） sisi 来替换第 i 种单词，使其满足如下要求：对于任意的 1≤i,j≤n,i≠j1≤i,j≤n,i≠j，都有 sisi 不是 sjsj 的前缀。（这个要求是为了避免二义性）
你的任务是对每个单词选择合适的 sisi，使得替换后的文章总长度（定义为所有单词**出现次数**与**替换它的二进制串的长度**乘积的总和）最小。求这个最小长度。
字符串 S1S1（不妨假设长度为 nn）被称为字符串 S2S2 的前缀，当且仅当：S2S2 的长度不小于 nn，且 S1S1 与 S2S2 前 nn 个字符组组成的字符串完全相同。

输入格式

第一行一个整数 nn，表示单词种数。
第 2 行到第 n+1n+1 行，第 i+1i+1 行包含一个正整数 wiwi，表示第 ii 种单词的出现次数。

输出格式

输出一行一个整数，表示整篇文章重编码后的最短长度。

样例输入

```
4
1
1
2
2
```

样例输出

```
12
```

样例解释

一种最优方案是令 s_1s1=000，s_2s2=001，s_3s3=01，s_4s4=1。这样文章总长即为 1*3+1*3+2*2+1*2=12。
另一种最优方案是令 s_1s1=00，s_2s2=01，s_3s3=10，s_4s4=11。这样文章总长也为 12。

数据范围

对于第 1 个测试点，保证 $n=3$n=3。
对于第 2 个测试点，保证 $n=5$n=5。
对于第 3 个测试点，保证 $n=16$n=16，且所有 $w_i$wi 都相等。
对于第 4 个测试点，保证 $n=1000$n=1000。
对于第 5 个测试点，保证所有 $w_i$wi 都相等。
对于所有的 7 个测试点，保证 $2\le n\le 10^6$2≤n≤106，$w_i\le 10^{11}$wi≤1011。
时间限制：2 sec

空间限制：256 MB

提示

[我们希望越长的串出现次数越少，那么贪心地考虑，让出现次数少的串更长。]

[于是我们先区分出出现次数最少的 2 个串，在它们的开头分别添加 0 和 1。]

[接着，由于它们已经被区分（想一想，为什么？），所以我们可以把它们看作是**一个**单词，且其出现次数为它们的和，然后继续上面的“添数”和“合并”操作。]

[这样，我们不停地“合并单词”，直到只剩 1 个单词，即可结束。]

[可以证明这是最优的。]

[朴素的实现是 $O(n^2)$ 的，可以用二叉堆或__std::priority_queue__将其优化至 $O(nlogn)$。]

代码

```
#include<iostream>
#include<queue>

using namespace std;


int main()
{
    int n;
    cin >> n;
    priority_queue <long long, vector<long long>, greater<long long>> pq;
    /*类似于二叉堆实现的队列，较小值的元素排在前面(可自定义升序或者降序及比较的方式),由于是利用堆实现的比较，故最大比较次数为O(nlogn)
    类似于huffman树，通过将最小的两个子节点之和获得父节点，并将该父节点放到priority_queue合适的位置
    将新的父节点放到合适位置之后，再求和，通过此来获得每个子节点应当放在类Huffman树的二叉树的第几层(每相加一次相当于将该单词下移一层)
    从而获得该单词应当由几个0或1构成(可以想象成父节点左边的子节点前面加一个0，右边加一个1)
    */

    long long* w=new long long[n];//某单词出现的次数w[i]
    for (int i = 0; i < n; i++)
    {
        cin >> w[i];
    }

    bool allSame = true;//比较是否所有的w[i]都一样，若都一样则不使用priority_queue，以免浪费比较时间
    for (int i = 1; i < n; i++)
    {
        if (w[i] != w[i - 1])
```

```
        {
            allSame = false;
            break;
        }
    }

    long long len = 0;//Huffman编码的所有单词的总长

    if (!allSame)//当不是所有单词出现次数w[i]都一样时，插入二叉堆中
    {
        for (int i = 0; i < n; i++)
        {
            pq.emplace(w[i]);
        }
    }
    else//若w[i]都一样，则直接使用普通队列，省去priority_queue的比较过程
    {
        queue<long long> p;
        long long temp = 0;//用于存储合并元素后的值
        for (int i = 0; i < n; i++)
        {
            p.emplace(w[i]);
        }
        while (p.size() != 1)//当队列中还存在有两个以上的单词时，说明该类huffman树内节点距离求和未完成，需要继续进行
        {
            for (int i = 0; i < 2; i++)//由两个较小的子节点的值相加(w[i]+w[i+1])获得该父节点的值，并将该父节点放到队列的末尾(由于w[i]都一样，故相加后的值必为整个队列的最大值)
            {
                temp += p.front();
                p.pop();
            }
            p.emplace(temp);
            len += temp;
            temp = 0;
        }
        cout << len << endl;
        return 0;
    }


    long long temp = 0;//用于存储合并元素后的值
    for(int i = 0; i < n; i++)
```

```
    {
        while (pq.size() != 1)
        {
            for (int i = 0; i < 2; i++)//求父节点之和，并将新的父节点插入到二叉树合适的层数之中
            {
                temp += pq.top();
                pq.pop();
            }
            pq.emplace(temp);
            len += temp;
            temp = 0;
        }
    }
    cout << len << endl;
    return 0;
}
```

成绩排序

问题描述

有 n 名学生，它们的学号分别是 1,2,...,n。这些学生都选修了邓老师的算法训练营、数据结构训练营这两门课程。

学期结束了，所有学生的课程总评都已公布，所有总评分数都是 [0,100] 之间的整数。巧合的是，不存在两位同学，他们这两门课的成绩**都完全相同**。

邓老师希望将这些所有的学生按这两门课程的**总分**进行降序排序，特别地，如果两位同学的总分相同，那邓老师希望把**算法训练营**得分更高的同学排在前面。由于上面提到的巧合，所以，这样排名就可以保证没有并列的同学了。

邓老师将这个排序任务交给了他的助教。可是粗心的助教没有理解邓老师的要求，将所有学生按**学号**进行了排序。

邓老师希望知道，助教的排序结果中，存在多少**逆序对**。

如果对于两个学生 (i,j) 而言，i 被排在了 j 前面，并且 i **本应**被排在 j 的后面，我们就称 (i,j) 是一个**逆序对**。

请你先帮邓老师把所有学生按正确的顺序进行排名，再告诉他助教的错误排名中逆序对的数目。

输入格式

第一行一个整数 n，表示学生的个数。

第 2 行到第 n+1 行，每行 2 个用空格隔开的非负整数，第 i+1 行的两个数依次表示学号为 i 的同学的算法训练营、数据结构训练营的总评成绩。

输出格式

输出包含 n+1 行。

前 n 行表示正确的排序结果，每行 4 个用空格隔开的整数，第 i 行的数依次表示排名为 i 的同学的学号、总分、算法训练营成绩、数据结构训练营成绩。

第 n+1 行一个整数，表示助教的错误排名中逆序对的数目。

样例输入

```
3
95 85
90 90
100 99
```

样例输出

```
3 199 100 99
1 180 95 85
2 180 90 90
2
```

样例解释

学号为 3 的同学总分为 199，是最高的，所以他应该排第一名。

学号为 1 的同学虽然总分与学号为 2 的同学一致，但是他的算法训练营成绩更高，所以在排名规则下更胜一筹。

原错误排名中的逆序对数目为 2 ，这些逆序对分别为 (1,3) 和 (2,3)。

数据范围

对于 25% 的数据，保证 n=3。

对于 50% 的数据，保证 n≤10。

对于另外 25% 的数据，保证所有同学的总分两两不同。

对于 100% 的数据，保证 n≤5,000 ，且保证不存在成绩**完全相同**的学生。

时间限制：10 sec

空间限制：256 MB

提示

[可以使用起泡排序将所有学生进行排名。]

[善良的助教提醒你，在起泡排序的过程中，每次交换都会使逆序对数目减少 1 哦！想一想，为什么？]

[聪明的助教还告诉你，这道题可以设计出时间复杂度为 O(nlogn) 的算法。这会在今后的课程中涉及到。]

代码

```
#include<iostream>
#include<queue>

using namespace std;

int ni = 0;//逆序对

class Student//定义学生类,存储学生的学号\总成绩\算法训练营成绩\数据结构成绩
{
public:
    int stNum, totalSocre, firSocre, secSocre;
    Student(int num, int socre1, int socre2)//构造函数
    {
        stNum = num;
        totalSocre = socre1 + socre2;
        firSocre = socre1;
        secSocre = socre2;
    }
};
```

```
bool operator<(Student a, Student b)//自定义priority_queue的比较
{
    //通过依次比较totalSocre, firSocre, secSocre来获取a与b比较的结果
    //不可通过a与b的比较来获取逆序对，因为priority_queue是类二叉堆，无法一一进行比较，故所获得的逆序对有无
    if (b.totalSocre > a.totalSocre)
    {
        return true;
    }
    else if (b.totalSocre == a.totalSocre)
    {
        if (b.firSocre > a.firSocre)
        {
            return true;
        }
        else if (b.firSocre == a.firSocre)
        {
            if (b.secSocre > a.secSocre)
            {
                return true;
            }
            else
                return false;
        }
        else
            return false;
    }
    else
        return false;
}

//通过对学号进行冒泡排序来获取逆序对(获取逆序对不可通过快速排序，例如{3,2,5,1}通过快速排序获得的是错误的逆序对)
//其中sourceArr是通过priority_queue传出的学号数组，即通过学号的逆序对来获得总的逆序对
//冒泡排序总的时间复杂度为O(n^2)
void BubbleSort(int num[],int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (num[i] > num[j])
                ni++;
        }
```

```
    }
}

int main()
{
    priority_queue<Student> pq;
    int n;
    cin >> n;
    int tempFir, tempSec;
    for (int i = 0; i < n; i++)
    {
        cin >> tempFir >> tempSec;
        Student st(i,tempFir,tempSec);
        pq.emplace(st);//通过比较，将st按由大到小的方式压入priority_queue中(总的时间复杂度为O(nlogn))
    }
    int* disoredrNum=new int[n];//通过priority_queue排序后所输出的st.stNum
    int* tempNum = new int[n];//归并排序的辅助数组
    int i = 0;
    while (!pq.empty())
    {
        Student st = pq.top();
        cout << st.stNum+1 << ' ' << st.totalSocre << ' ' << st.firSocre << ' ' << st.secSocre << endl;
        disoredrNum[i] = st.stNum + 1;
        i++;
        pq.pop();
    }
    BubbleSort(disoredrNum,n);
    cout << ni << endl;
    return 0;
}
```

等式

描述

有 n 个变量和 m 个“相等”或“不相等”的约束条件，请你判定是否存在一种赋值方案满足所有 m 个约束条件。

输入

第一行一个整数 T，表示数据组数。

接下来会有 T 组数据，对于每组数据：

第一行是两个整数 n,m，表示变量个数和约束条件的个数。

接下来 m 行，每行三个整数 a,b,e，表示第 a 个变量和第 b 个变量的关系：

- 若 e=0 则表示第 a 个变量**不等于**第 b 个变量;
- 若 e=1 则表示第 a 个变量**等于**第 b 个变量

输出

输出 T 行，第 i 行表示第 i 组数据的答案。若第 i 组数据存在一种方案则输出"Yes"；否则输出"No"（不包括引号）。

输入样例 1

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

输出样例 1

```
Yes
No
```

样例 1 解释

一共有 2 组数据。

对于第一组数据，有 5 个约束：

- 变量 1=变量 2
- 变量 2=变量 3
- 变量 3=变量 4
- 变量 1=变量 4
- 变量 2≠变量 5

显然我们可以令：

- 变量 1=变量 2=变量 3=变量 4=任意一个数值
- 变量 5=任意一个和变量 2 不同的数值

故第一组数据输出"Yes"。 对于第二组数据，有 3 个约束：

- 变量 1=变量 2
- 变量 2=变量 3
- 变量 1≠变量 3

由前两个约束可推出变量 1=变量 3，但第三个约束表明变量 1≠变量 3，矛盾。

故第二组数据输出"No"。

输入样例 2

点击下载

限制

对于 10%的数据，n,m ≤ 5，T ≤ 5；

对于 50%的数据，n,m ≤ 1000，T ≤ 10；

对于 100%的数据，1 ≤ n ≤ 300000, m ≤ 500000，1 ≤ a,b ≤ n，T ≤ 100。

保证所有数据的 n 总和与 m 总和不超过 500000。

时间：2 sec

空间：256 MB

提示

[用并查集来维护相等的集合。]

[改编自 NOI 2015 day1 T1 程序自动分析]

代码

```
#include<iostream>
#include<vector>
#include<string>
using namespace std;

int *Father;

//P.S. "秩"为一棵树的高度的相反数

int Find(int x)//查找某节点的根节点
{
    int fx;//某节点的根节点(父节点)
    if (Father[x] > 0)//若x的秩大于0，说明x不是根节点
    {
        fx = Find(Father[x]);//若x不是根节点，则递归查找其根节点
        Father[x] = fx;//*****此处至关重要，路径压缩，将某节点的父节点设置为根节点
********
        return fx;
    }
    else
        return x;//若x的秩小于等于0，说明x是根节点，返回x
}

void Union(int x, int y)//将x和y所在的两个集合合并
{
    int fx = Find(x);
    int fy = Find(y);
    fy = fy;
    if (fx == fy && fx != 0)//若fx和fy相等且其两个的秩不为0，则说明他们在同一个集合树中
        return;
    if (Father[fx] < Father[fy])
        Father[fy] = fx;//若x所在的集合树比较高，则将y所在的树合并到x所在的树中，即将y
的根节点设置为fx
    else if (Father[fx] == Father[fy])//若两个树的高度相等，则任意将一个树合并到另一个
```

```
树中(此处是将x的根节点设置为fy)
    {
        Father[fy] -= 1;//合并后树的高度加一，故秩减一
        Father[fx] = fy;
    }
    else//若y所在的集合树比较高，则将x所在的树合并到y所在的树中，即将x的根节点设置为fy
        Father[fx] = fy;
}

int main()
{
    int t;//数据组数
    cin >> t;

    string* ans = new string[t];//存储是否存在方案(Yes或No)

    for (int i = 0; i < t; i++)
    {

        ans[i] = "Yes";//先将答案定为Yes，接着通过比较，若不存在方案则将答案从Yes改为
No

        int n, m;//变量个数和约束条件
        cin >> n >> m;
        Father = new int[n+1];
        for (int j = 0; j < n + 1; j++)
        {
            Father[j] = 0;//每个元素的初始秩为0
        }
        vector<int> InEqu;//用于存储不相等的元素，以便在最后判断是否存在方案满足所有m
个约束条件
        InEqu.resize(2 * m);//设置vector容器大小
        int inEqu_num = 0;//用于标记InEqu中有多少个元素
        int a, b, e;//e表示a变量和b变量的关系
        for (int j = 0; j < m; j++)
        {
            cin >> a >> b >> e;
            if (e == 1 && a != b)//若变量存在对应关系，且两个变量不是同一个数，则合并
集合树
            {
                Union(a, b);
            }
            else if (e == 0&&a!=b)
                //若两个变量不存在对应关系，则压入容器中，在最后查找两个变量的根节
```

```
点，若两个变量的根节点一样（且不为0），说明不存在满足约束条件的解，反之则存在
            {
                InEqu.emplace_back(a);
                InEqu.emplace_back(b);
                inEqu_num += 2;
            }
            else if (e == 0 && a == b)//若两个变量是同一个数且不存在对应关系，则必然无法找出满足约束条件的解
            {
                ans[i] = "No";
                continue;
            }
        }

        //查找InEqu中两个变量的根节点，若两个变量的根节点一样（且不为0），说明不存在满足约束条件的解，反之则存在
        while (inEqu_num != 0)
        {
            int fir = InEqu.back();
            InEqu.pop_back();
            int sec = InEqu.back();
            InEqu.pop_back();
            inEqu_num -= 2;

            int Ffir = Find(fir);
            int Fsec = Find(sec);
            if (Ffir == Fsec && Ffir != 0)
            {
                ans[i] = "No";
                break;
            }
        }

    }

    for (int i = 0; i < t; i++)
    {
        cout << ans[i] << endl;//输出答案
    }

    return 0;

}
```

道路升级

问题描述

Z 国有 n 个城市和 m 条双向道路，每条道路连接了两个不同的城市，保证所有城市之间都可以通过这些道路互达。每条道路都有一个载重量限制，这限制了通过这条道路的货车最大的载重量。道路的编号从 1 至 m 。巧合的是，**所有道路的载重量限制恰好都与其编号相同**。

现在，要挑选出若干条道路，将它们升级成高速公路，并满足如下要求：

- 所有城市之间都可以通过高速公路互达。
- 对于任意两个城市 u,v 和足够聪明的货车司机：只经过高速公路从 u 到达 v 能够装载货物的最大重量，与经过任意道路从 u 到达 v 能够装载货物的最大重量相等。（足够聪明的司机只关注载重量，并不在意绕路）

在上面的前提下，要求选出的道路数目尽可能少。

求需要挑选出哪些道路升级成高速公路（如果有多种方案请任意输出一种）。

输入

第一行 2 个用空格隔开的整数 n,m ，分别表示城市数目、道路数目。

第 2 行到第 m+1 行，每行 2 个用空格隔开的整数 u,v 描述一条从 u 到 v 的双向道路，第 i+1 行的道路的编号为 i 。

注意：数据只保证不存在连接的城市相同的道路（自环），并不保证不存在两条完全相同的边（重边）

输出

第一行一个整数 k ，表示升级成高速公路的道路数。

接下来 k 行每行一个整数，**从小到大**输出所有选出的道路的编号。

输入样例

```
3 3
1 2
```

```
2 3
1 3
```

输出样例

```
2
2
3
```

数据范围

对于 20% 的数据，保证 n≤5，m≤10。

对于 60% 的数据，保证 n≤1,000，m≤5,000。

对于 100% 的数据，保证 n≤200,000，m≤400,000。

时间限制：10 sec

空间限制：256 MB

提示

[提示 1：真的可能有多种方案吗？]

[提示 2：k 是否一定为 n-1 呢？（也就是说，选出的道路是否恰好构成了一棵树？）]

[提示 3：这道题和最小生成树有什么关系呢？]

代码

```cpp
#include <iostream>
#include<vector>
#pragma warning(disable:4996)
using namespace std;

// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */
int *Father;
```

```
int Find(int x)//查找某节点的根节点
{
    int fx;//某节点的根节点(父节点)
    if (Father[x] > 0)//若x的秩大于0，说明x不是根节点
    {
        fx = Find(Father[x]);//若x不是根节点，则递归查找其根节点
        Father[x] = fx;//*****此处至关重要，路径压缩，将某节点的父节点设置为根节点
********
        return fx;
    }
    else
        return x;//若x的秩小于等于0，说明x是根节点，返回x
}


void Union(int x, int y)//将x和y所在的两个集合合并
{
    int fx = Find(x);
    int fy = Find(y);
    fy = fy;//断点调试点
    if (fx == fy && fx != 0)//若fx和fy相等且其两个的秩不为0，则说明他们在同一个集合树中
        return;
    if (Father[fx] < Father[fy])
        Father[fy] = fx;//若x所在的集合树比较高，则将y所在的树合并到x所在的树中，即将y
的根节点设置为fx
    else if (Father[fx] == Father[fy])//若两个树的高度相等，则任意将一个树合并到另一个
树中(此处是将x的根节点设置为fy)
    {
        Father[fy] -= 1;//合并后树的高度加一，故秩减一
        Father[fx] = fy;
    }
    else//若y所在的集合树比较高，则将x所在的树合并到y所在的树中，即将x的根节点设置为fy
        Father[fx] = fy;
}

// 给定一个n个点m条边的无向图，第i条边边权为i，求所有需要升级的边
// n：如题意
// m：如题意
// U：大小为m的数组，表示m条边的其中一个端点
// V：大小为m的数组，表示m条边的另一个端点
// 返回值：所有需要升级的边，从小到大排列；第一小问的答案自然即为返回值的size，所以你不
必考虑如何返回size
vector<int> getAnswer(int n, int m, vector<int> U, vector<int> V) {
    /* 请在这里设计你的算法 */
```

```
    //通过生成最大生成树，我们即可知道哪一些道路应当升级为高速公路

    Father = new int[n + 1];
    for (int j = 0; j < n + 1; j++)
    {
        Father[j] = 0;//每个元素的初始秩为0
    }

    vector<int> Answer;//输出
    Answer.reserve(200000);

    int k = 0;//应升级为高速公路的k条道路，由最大生成树可知，当k=n-1时最大生成树生成完
毕
    for(int j=m;j>0;--j)
    {
        int u = U.back();
        U.pop_back();
        int v = V.back();
        V.pop_back();

        int fu = Find(u);
        int fv = Find(v);
        //下面用城市序号生成最大生成树
        if (fu != fv || fu == 0)//若两个城市的
        {
            Union(u, v);
            Answer.emplace_back(j);//将需要升级的道路序号压入容器中
            k = k;
            k++;
        }
        if (k == n - 1)
            break;
    }
    return Answer;
}

// ================ 代码实现结束 ================

int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    vector<int> U, V;
    for (int i = 0; i < m; ++i) {
```

```
        int u, v;
        scanf("%d%d", &u, &v);
        U.emplace_back(u);
        V.emplace_back(v);
    }
    vector<int> ans = getAnswer(n, m, U, V);
    printf("%d\n", int(ans.size()));
    for (int i = int(ans.size())-1; i >= 0; --i)//由小到大输出道路序号
        printf("%d\n", ans[i]);
    return 0;
}
```

分组

描述

有 n 个正整数排成一排，你要将这些数分成 m 份（同一份中的数字都是连续的，不能隔开），同时数字之和最大的那一份的数字之和尽量小。

输入

输入的第一行包含两个正整数 n，m。

接下来一行包含 n 个正整数。

输出

输出一个数，表示最优方案中，数字之和最大的那一份的数字之和。

样例 1 输入

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

样例 1 输出

```
5
```

样例 1 解释

若分成 2 和 1、2、2、3，则最大的那一份是 1+2+2+3=8；

若分成 2、1 和 2、2、3，则最大的那一份是 2+2+3=7；

若分成 2、1、2 和 2、3，则最大的那一份是 2+1+2 或者是 2+3，都是 5；

若分成 2、1、2、2 和 3，则最大的那一份是 2+1+2+2=7。

所以最优方案是第三种，答案为 5。

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

对于 50%的数据，n ≤ 100，给出的 n 个正整数不超过 10；

对于 100%的数据，m ≤ n ≤ 300000，给出的 n 个正整数不超过 1000000。

时间：4 sec

空间：512 MB

提示

[大家记得看到“最大的最小”这一类语言，一定要想二分能不能做。]

[我们二分最大的那一份的和 d，然后从左向右分组，在一组中，在和不超过 d 的情况下尽量往右分。若最终分出来的组数小于等于 m，则这显然是合法的（我们在分出来的组里随便找个地方切开，总能变到 m 组，且每组的和不超过 d）]

[这个 d 显然是单调的，即，若和不超过 d 能分成 m 组，则和不超过 d+1 也是能分成 m 组的，故二分正确。]

代码

```
#include<iostream>
#include<vector>
#pragma warning(disable:4996)
using namespace std;
typedef long long ll;
```

```
bool check(ll mid, ll n, ll m,vector<int> &a)
{
    ll sum = 0;
    int cut = 1;
    for (int i = 0; i < n; ++i)
    {
        if (a[i] > mid)
            return false;
        sum += a[i];
        if (sum > mid)
        {
            sum = a[i];
            cut += 1;
        }
    }
    return cut <= m;
}

ll GetAnswer(ll n, ll m, vector<int> &a)
{
    ll l = 1, r = 0;
    for (int i = 0; i < n; ++i)
    {
        r += a[i];
    }

    while (l <= r)
    {
        ll mid = (l + r) / 2;
        if (check(mid, n, m, a))
            r = mid - 1;
        else
            l = mid + 1;
    }
    return r + 1;
}

int main()
{
    ll n, m;
    cin >> n >> m;
    vector<int> a;
    for (ll i = 0; i < n; ++i)
```

```
    {
        int temp;
        scanf("%d", &temp);
        a.emplace_back(temp);
    }
    ll result = GetAnswer(n, m, a);
    cout << result << endl;
    return 0;
}
```

大转盘

时间限制：1 sec

空间限制：256 MB

问题描述

邓老师有一个大转盘，被平分成了 2^n 份。

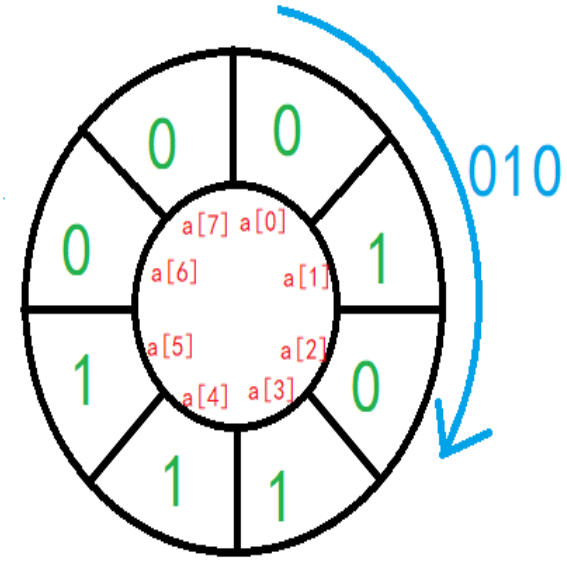
邓老师还有一个长度为 2^n 的数组 a（下标从 0 开始），其中的每个元素都是 0 或 1。于是邓老师就可以选择大转盘上的一个位置，将 a[0] 填入其中，然后按顺时针顺序依次将 a[1],a[2],…,a[2^n-1] 填入。

对于大转盘上的一个指定位置，邓老师可以从它开始，取出顺时针方向的 n 个位置，并将它们按原顺序拼接起来，得到一个长度为 n 的 01 串，也就是一个 n 位二进制数。我们把这个二进制数称作从这个位置开始的**幸运数**。

显然地，大转盘上共有 2^n 个位置可以获得幸运数，而巧合的是 n 位二进制数恰好也有 2^n 个，所以邓老师希望这些所有的幸运数包含了所有的 n 位二进制数。

请输出一个数组 a，使其满足邓老师的要求。（如果有多解，输出任一即可）

下面是一个 n=3 的例子（即样例）。



输入格式

一行一个整数 n。

输出格式

输出一行 2^n 个字符，第 i 个字符（1<=i<=2^n）表示 a[i-1]。

样例输入

```
3
```

样例输出

```
01011100
```

数据范围

本题包含 16 个测试点。对于第 i 个测试点（1<=i<=16），满足 n=i。

提示

[如果把所有 n-1 位二进制数建立节点，将所有的 n 位二进制数视为单向边。对于边 x，设其前 n-1 位为 u，后 n-1 位为 v，则其连接 u,v。]

[在这张图上求出欧拉回路，想一想，答案与欧拉回路有何关联呢？]

代码

```
//#include <bits/stdc++.h>
#include <vector>
#include <iostream>
#include <sstream>
#pragma warning(disable:4996)
using namespace std;

// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */

int allOne;
vector<bool> vis[2];
string ans;

int twoPow(int x)
{
    return 1 << x;
}

void dfs(int u)
{
    for (int i = 0; i < 2; ++i)
    {
        if (!vis[i][u]) {
            int v = ((u << 1) | i) & allOne;
            vis[i][u] = 1;

            dfs(v);
            ans.push_back('0' + i);
        }
    }
}
```

```
// 本函数求解大转盘上的数，你需要把大转盘上的数按顺时针顺序返回
// n：对应转盘大小，意义与题目描述一致，具体见题目描述。
// 返回值：将大转盘上的数按顺时针顺序放到一个string中并返回
string getAnswer(int n) {
    /* 请在这里设计你的算法 */
    allOne = twoPow(n - 1) - 1;
    ans = "";
    for (int i = 0; i < 2; ++i)
        vis[i].resize(twoPow(n - 1), 0);

    dfs(0);
    return ans;
}

// ================ 代码实现结束 ================

int main() {
    int n;
    scanf("%d", &n);

    cout << getAnswer(n) << endl;

    return 0;
}
```

象棋

描述

你有足够多的象棋“车”，在一个 n×n 的棋盘上你能放多少个“车”呢？注意，所给棋盘上有些位置不能放任何东西。同时，某一行（列）最多只能存在一个“车”。

输入

第一行为一个正整数 n。

接下来 n 行，每行包含 n 个整数，若为 0 表示这个位置不能放“车”；若为 1 表示这个位置可以放“车”。

输出

输出一个整数，表示最多能放多少个“车”。

样例 1 输入

```
5
1 0 0 0 0
0 0 0 0 0
0 0 0 1 0
1 1 0 1 0
0 0 0 1 0
```

样例 1 输出

```
3
```

样例 1 解释

我们这样放就只能放 2 个“车”：

```
车 0 0 0 0
0 0 0 0 0
0 0 0 1 0
1 1 0 车 0
0 0 0 1 0
```

若我们这样放就能放下 3 个了：

```
车 0 0 0 0
0 0 0 0 0
0 0 0 1 0
1 车 0 1 0
0 0 0 车 0
```

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

对于 30%的数据，n ≤ 5；

对于 60%的数据，n ≤ 20；

对于 100%的数据，n ≤ 500。

时间：2 sec

空间：256 MB

提示

[将横坐标和纵坐标看做是二分图的 X 集和 Y 集，会了吗？]

代码

```
#include<iostream>
#include<memory.h>
#pragma warning(disable:4996)

using namespace std;

/*该算法思想源于匈牙利算法，其流程可参考以下网址的解析：
https://blog.csdn.net/dark_scope/article/details/8880547
男性对应棋盘中的行，女性对应棋盘中的列
*/

int* used;//用于存储棋盘中某一列是否被使用
int* pointc;//用于存储某一列中被哪一行所使用
int** cb;//用于存储棋盘

bool find(int x,int n)
{
    int i, j;
    for (int j = 0; j < n; ++j)
    {
        if (cb[x][j] == 1 && used[j] == 0)//如果棋盘中该行中某一列为1且此列在此行中未
```

```
被使用，则使用此列
        {
            used[j] = 1;
            if (pointc[j] == -1 || find(pointc[j],n))//若此列未被其他行所使用或者虽然被其他行所使用，但是其他行能找到另外的解(匹配列数)，则其他行使用其他的解，该行使用该列
                                                    //此处的find(pointc[j],n)使用递归，从第pointc[j]行递归到第0行(假设第0行的某一列为1)，直到找到使得上面的所涉及的行数都能找到另外的相匹配的列数
            {
                pointc[j] = x;
                return true;
            }
        }
    }
    return false;
}

int main()
{
    int n;
    scanf("%d", &n);
    cb = new int*[n];//用于存储棋盘
    used = new int[n];
    pointc = new int[n];//用于存储某一列中被哪一行所使用
    for (int i = 0; i < n; ++i)
    {
        cb[i] = new int[n];
        pointc[i] = -1;
    }
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            scanf("%d", &cb[i][j]);
        }
    }
    int all = 0;
    for (int i = 0; i < n; i++)
    {
        memset(used, 0, sizeof(int)*n);    //这个在每一步中清空
        if (find(i, n))
            all++;
    }
    cout << all << endl;
```

```
    return 0;
}
```

序列计数

描述

给定一个 n 个整数的序列以及一个**非负整数** d，请你输出这个序列中有多少个连续子序列（长度大于 1），满足该子序列的最大值最小值之差不大于 d。

连续子序列：序列 1 2 3 中长度大于 1 的连续子序列有：

```
1 2

2 3

1 2 3
```

输入

第一行包含两个整数 n,d。

接下来一行包含 n 个整数。

输出

输出一个整数，表示满足条件的连续子序列个数。

样例 1 输入

```
8 5

5 5 4 8 -10 10 0 1
```

样例 1 输出

```
7
```

样例 1 解释

满足条件的连续子序列有：

```
5 5
5 5 4
5 5 4 8
5 4
5 4 8
4 8
0 1
```

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

对于 60%的数据，n ≤ 5000；

对于 100%的数据，n ≤ 300000。

保证所有整数的绝对值不超过 10^9，d 不超过 2×10^9。

时间：10 sec

空间：512 MB

提示

[考虑分治。]

[令函数 solve(l, r)表示统计[l, r]中合法的连续子序列个数，mid 为(l+r)/2（下取整），那么]

[solve(l, r) = 0，当 l = r]

[solve(l, r) = solve(l, mid) + solve(mid + 1, r) + cal(l, r, mid)，当 l≠r]

[其中 cal(l, r, mid)表示在左端点在区间[l, mid]中、右端点在区间[mid + 1, r]中的符合要求的连续子序列数目]

[那么答案就是 solve(1, n)。]

[至于 cal(l, r, mid)怎么算，大家可以仔细思考思考。（右端点是有单调性的）]

[**注意答案要用 long long**]

[（另外这题也可以用线性的方法做哦~有兴趣去搜一搜单调栈）]

代码

```
#include <iostream>
#include<vector>
#include<stack>
#include<algorithm>
#pragma warning(disable:4996)
typedef long long ll;
using namespace std;

// ================= 代码实现开始 =================

/* 请在这里定义你需要的全局变量 */
const int N = 300005;

//n:题目中的n
//d:题目中的d
//max_value:用于存储solve函数中的最大值
//min_value:用于存储solve函数中的最小值
//a:题目中的a
int n, d, max_value[N], min_value[N];
vector<int> a;
//分治计算区间[l,r]中有多少连续子序列满足最大值最小值之差不大于d
//l:区间左边界
//r:区间右边界
//返回值:满足条件的连续子序列的个数
ll solve(int l, int r)
{
    if (l == r)//边界情况。我们不计算长度等于1的连续子序列,故返回0
        return 0;
    int mid = (l + r) / 2;//获得区间中点
    ll ans = solve(l, mid) + solve(mid + 1, r);//递归，分治地求出左右两半的值，并累加

    //我们计算区间[mid+1,r]的前缀最小值和前缀最大值，也就是说
min_value[i]=min(a[mid+1,mid+2,..,i]),max同理
    for (int i = mid + 1; i <= r; ++i)
    {
        min_value[i] = (i == mid + 1) ? a[i] : min(min_value[i - 1], a[i]);
        max_value[i] = (i == mid + 1) ? a[i] : max(max_value[i - 1], a[i]);
    }
```

```
    //我们倒序枚举子序列的左端点i,i的取值范围在[l,mid]
    //pos表示若连续子序列的左端点是i,那么子序列的右端点最远能拓展到pos位置，当然pos取值范围在[mid+1,r]，一开始初始化为r
    //mn是后缀最小值,mx是后缀最大值，也就是说mn=min(a[i,...,mid]),mx同理
    //那么以i为左端点的连续子序列(右端点在[mid+1,r]内)个数应该有pos-mid个
    int mn = 0, mx = 0, pos = r;
    for (int i = mid; i >= l && pos > mid; --i)
    {
        min_value[i] = (i == mid) ? a[i] : min(min_value[i + 1], a[i]);
        mn = min_value[i];
        max_value[i] = (i == mid) ? a[i] : max(max_value[i + 1], a[i]);
        mx = max_value[i];
        for (; pos > mid&&max(mx, max_value[pos]) - min(mn, min_value[pos]) > d; --pos);//pos随着i的递减也递减(注意此处for循环无执行语句)
        ans += pos - mid;//相当于求左端点在区间[l,mid]、右端点在区间[mid + 1, r]中的符合要求的连续子序列数目
                         //注意此处不是为pos-i的原因是在上面的ll ans = solve(l, mid) + solve(mid + 1, r);中已经获得了mid左边至i的子序列数目，故此时只需要加上mid右边至pos的子序列数目即可得到上一行注释中的连续子序列数目
    }

    return ans;
}
// 求出有多少个a数组中的连续子序列（长度大于1），满足该子序列的最大值最小值之差不大于d
// n：a数组的长度
// d：所给d
// a：数组a，长度为n
// 返回值：满足条件的连续子序列的个数
long long getAnswer(int n, int d, vector<int> a) {
    ::n = n;
    ::d = d;
    ::a = a;
    return solve(0, n - 1);
    /* 请在这里设计你的算法 */
}

// ================= 代码实现结束 =================


int main() {
    scanf("%d%d", &n, &d);
    a.resize(n);
```

```
    for (int i = 0; i < n; ++i)
        scanf("%d", &a[i]);
    printf("%lld\n", getAnswer(n, d, a));
    return 0;
}
```

中位数

描述

小粽最近学习了中位数的相关知识，她知道了这样一个事实：对于任意 $2n-12n-1$ 个数，将它们从小到大排序后，第 nn 个数就是这个 $2n-12n-1$ 个数的中位数。
现在，小粽想解决这样一个问题：对于一个长度为 $2n-12n-1$ 的数列，
前 $2k-1(k=1,2,\ldots,n)2k-1(k=1,2,\ldots,n)$ 个数的中位数各是多少？

输入

第一行一个正整数 nn，表示有一个长度为 $2n-12n-1$ 的数列。
接下来一行 $2n-12n-1$ 个正整数，描述这个数列，用空格隔开。

输出

输出 nn 行，第 ii 行表示这个数列的前 $2i-12i-1$ 个数的中位数。

输入样例

```
3
8 7 6 9 9
```

输出样例

```
8
7
8
```

限制

对于 100%100% 的数据，$n \leq 3 \times 10^5$
所有输入数据的数值都在 int 范围内。

提示

[使用两个堆分别保存前一半和后一半大的数。]

代码

```
#include<iostream>
#include<queue>
#include <vector>
#include <functional>     // std::greater

#pragma warning(disable:4996)

using namespace std;

/* ==================== 代码实现开始 ================== */
typedef priority_queue<int, vector<int>, less<int>>BigHeap;//有序队列，实际上内部是以堆的方式实现,less表示从大到小排序
typedef priority_queue<int, vector<int>, greater<int>>SmallHeap;

BigHeap bigHeap;//大根堆(即任意节点的值均比其左右叶节点的值大),保存较小一半的数
SmallHeap smallHeap;//小根堆(与大根堆相反),保存较大一半的数

//返回值：当前序列中位数的值
int getMedian()
{
    if (smallHeap.size() >bigHeap.size() && !smallHeap.empty())
        return smallHeap.top();
    else
        return bigHeap.top();
}

//向当前数列末尾插入一个数x
void add(int x)
{
    if (smallHeap.empty()||x < smallHeap.top())//若x比较大一半的数中的最小值还小，则说明x属于较小一半的数，用大根堆保存
    {
```

```
            bigHeap.emplace(x);
            if (bigHeap.size() - smallHeap.size() >= 2)//若两个堆相差为2，则说明需要进行调整，否则例如6 5 4 3 2 1的序列插入后5 4 3 2 1均保存在大根堆中，此时通过return bigHeap.top();获得的数不为中位数
            {
                smallHeap.emplace(bigHeap.top());
                bigHeap.pop();
            }
        }
        else
        {
            smallHeap.emplace(x);
            if (smallHeap.size() - bigHeap.size() >= 2)//若两个堆相差为2，则说明需要进行调整，否则例如6 5 4 3 2 1的序列插入后5 4 3 2 1均保存在大根堆中，此时通过return bigHeap.top();获得的数不为中位数
            {
                bigHeap.emplace(smallHeap.top());
                smallHeap.pop();
            }
        }
}

/* ==================== 代码实现结束 ================== */

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= 2 * n - 1; i++) {
        int a;
        scanf("%d", &a);
        // fprintf(stderr, "%d\n", a);
        add(a);
        if (i & 1)
            printf("%d\n", getMedian());
    }
    return 0;
}
```

最小交换

时间限制：4 sec

空间限制：256 MB

问题描述

给定一个 1 到 n 的排列（即一个序列，其中 [1,n] 之间的正整数每个都出现了恰好 1 次）。

你可以花 1 元钱交换两个**相邻**的数。

现在，你希望把它们升序排序。求你完成这个目标最少需要花费多少元钱。

输入格式

第一行一个整数 n，表示排列长度。

接下来一行 n 个用空格隔开的正整数，描述这个排列。

输出格式

输出一行一个非负整数，表示完成目标最少需要花多少元钱。

样例输入

```
3
3 2 1
```

样例输出

```
3
```

样例解释

你可以：

花 1 元交换 1,2，序列变成 3 1 2。

花 1 元交换 1,3，序列变成 1 3 2。

花 1 元交换 2,3，序列变成 1 2 3。

总共需要花 3 元。

可以证明不存在更优的解。

数据范围

对于 20% 的数据，保证 n<=7。

对于 60% 的数据，保证 n<=1,000。

对于 100% 的数据，保证 n<=200,000。

提示

[每次交换相邻的两个数都会使逆序对 +1 或 -1。]

[逆序对数目不为零时，一定存在相邻的逆序对。]

[因此最优策略显然是每次都让逆序对数目 -1。]

[所以答案即为逆序对数目。]

[朴素的算法时间复杂度是 O(n^2) 的。]

[用归并排序求逆序对数可以通过本题。需要提醒的是，答案可能超过 32 位整数的范围哦。]

[逆序对数目同样可以用树状数组优化朴素的 O(n^2) 算法，并获得和归并排序相同的时间复杂度。有兴趣的同学可以自行学习。]

代码

```
#include <iostream>
#include<vector>
#pragma warning(disable:4996)
using namespace std;

// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */

//seq:原序列，为了方便处理，将其设为全局变量
//seqTemp:用以辅助计算的临时数组
//cnt:统计逆序对个数
vector<int>seq, seqTemp;
long long cnt;

//归并排序函数
//l,r:分别为归并排序排序区间的左、右端点
```

```
void mergeSort(int l, int r)
{
    if (l == r)//此时无需排序直接返回
        return;
    int mid = (l + r) >> 1;//等价于mid=(l+r)/2
    mergeSort(l, mid);//递归排序mid以左的部分
    mergeSort(mid + 1, r);//递归排序mid以右的部分
    int p = l, q = mid + 1;//用两个指针来指向归并时需要比较的两个元素
    for (int i = l; i <= r; ++i)
    {
        if (q > r || p <= mid && seq[p] <= seq[q])//注意避免数组越界的情况
            seqTemp[i] = seq[p++];//如果左边的元素更小，则将左边的元素插入末尾
        else
        {
            seqTemp[i] = seq[q++];//如果右边元素更小，则将右边元素插入末尾并增加逆序对
            cnt += mid - p + 1;//统计产生逆序对数目(注意此处不能用cnt++,因为左边已经有
序，此时若seq[q] < seq[p]说明seq[q]比seq[p]~seq[mid]都小
        }
    }
    for (int i = l; i <= r; ++i)
        seq[i] = seqTemp[i];//将排序后的序列复制回原序列的对应位置
}

// 这个函数的功能是计算答案（即最少花费的金钱）
// n：表示序列长度
// a：存储整个序列 a
// 返回值：最少花费的金钱（需要注意，返回值的类型为 64 位有符号整数）
long long getAnswer(int n) {
    /* 请在这里设计你的算法 */
    seqTemp.resize(n);//初始化临时数组的长度，此算法需要额外的O(n)空间
    cnt = 0;//置零计数器
    mergeSort(0, n - 1);//进行归并排序
    return cnt;
}

// ================= 代码实现结束 =================

int main() {
    int n, tmp;
    seq.clear();
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &tmp);
        seq.emplace_back(tmp);
```

```
    }
    long long ans = getAnswer(n);
    cout << ans << '\n';
    return 0;
}
```

楼尔邦德

时间限制：2 sec

空间限制：256 MB

问题描述

给定包含 n 个数的序列 A。

再给出 Q 个询问，每个询问包含一个数 x，询问的是序列 A 中不小于 x 的最小整数是多少（无解输出-1）。

输入格式

第一行一个数 n，表示序列长度。

第二行 n 个用空格隔开的正整数，描述序列中的每一个元素。保证这些元素都不会超过10^9。

第三行一个正整数 Q，表示询问个数。

接下来 Q 行，每行一个正整数 x，描述一个询问。

输出格式

输出 Q 行依次回答 Q 个询问，每行一个正整数，表示对应询问的答案。

样例输入

```
3
3 2 5
6
```

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

样例输出

```
2
2
3
5
5
-1
```

数据范围

对于 50% 的数据，保证 n<=2000。

对于 100% 的数据，保证 n<=300,000。

提示

[如果我们先将原序列排序，那么我们就可以在它上面进行二分查找啦！]

[STL 库中有二分查找的函数__std::lower_bound__，你可以学习一下它的使用。当然啦，作为初学者，还是建议自己实现二分查找哦！]

代码

```cpp
#include <iostream>
#include<vector>
#include <algorithm>// std::lower_bound, std::upper_bound, std::sort
#pragma warning(disable:4996)
using namespace std;
```

```
// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */

// 本函数传入数组 a 及所有询问，你需要求解所有询问并一并返回
// n：序列 a 的长度
// a：存储了序列 a
// Q：询问个数
// query：依次存储了所有询问的参数 x
// 返回值：一个 vector<int>，依次存放各询问的答案
vector<int> getAnswer(int n, vector<int> a, int Q, vector<int> query) {
    vector<int> ans;
    sort(a.begin(), a.end());
    vector<int>::iterator low;
    for (int i = 0; i < Q; i++)
    {
        low = lower_bound(a.begin(), a.end(), query[i]);
        if (low == a.end())
            ans.push_back(-1);
        else
            ans.push_back(a[low-a.begin()]);
    }
    return ans;
    /* 请在这里设计你的算法 */
}

// ================ 代码实现结束 ================

int main() {
    int n, Q, tmp;
    vector<int> a, query;
    a.reserve(300000);
    query.reserve(300000);
    a.clear();
    query.clear();
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%d", &tmp);
        a.push_back(tmp);
    }
    scanf("%d", &Q);
    for (int i = 0; i < Q; ++i) {
        scanf("%d", &tmp);
        query.push_back(tmp);
```

```
    }
    vector<int> ans = getAnswer(n, a, Q, query);
    for (int i = 0; i < Q; ++i)
        printf("%d\n", ans[i]);
    return 0;
}
```

最短路

时间限制： 4 sec

空间限制： 256 MB

问题描述

给定一张 n 个点的无向带权图，节点的编号从 1 至 n，求从 S 到 T 的最短路径长度。

输入格式

第一行 4 个数 n,m,S, T，分别表示点数、边数、起点、终点。

接下来 m 行，每行 3 个正整数 u,v,w，描述一条 u 到 v 的双向边，边权为 w。

保证 1<=u,v<=n。

输出格式

输出一行一个整数，表示 S 到 T 的最短路。

样例输入

```
7 11 5 4
2 4 2
1 4 3
7 2 2
3 4 3
```

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

样例输出

```
7
```

样例文件下载（包含第二个样例）

数据范围

本题共设置 12 个测试点。

对于前 10 个测试点，保证 n<=2500，m<=6200，对于每条边有 w<=1000。这部分数据有梯度。

对于所有的 12 个测试点，保证 n<=100,000，m<=250,000。

提示

[本题是 Dijkstra 算法的模板练习题。]

[使用朴素的 Dijkstra 算法可以通过前 10 个测试点。]

[使用堆或__std::priority_queue__优化的 Dijkstra 算法可以通过所有测试点。]

代码

```cpp
#include <iostream>
#include<vector>
#include<queue>
#include <utility>
#include <cstring>
#pragma warning(disable:4996)
using namespace std;
```

```
// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */
const int N = 100005;
typedef pair<int, int> pii;

//graph:存放图,graph[i]表示的是节点i的出边,其中first存储到达的节点,second存储边权
//pq:辅助Dijkstra算法的优先队列
//flag:记录每个节点是否进行过松弛，1表示进行过，0表示未进行过
//mind:存储起点s到每个节点的最短路径长度
vector<pii> graph[N];
priority_queue<pii, vector<pii>, greater<pii>> pq;//升序队列
bool flag[N];
int mind[N];

// 这个函数用于计算答案（最短路）
// n：节点数目
// m：双向边数目
// U,V,W：分别存放各边的两端点、边权
// s,t：分别表示起点、终点
// 返回值：答案（即从 s 到 t 的最短路径长度）
int shortestPath(int n, int m, vector<int> U, vector<int> V, vector<int> W, int s, int
t)
{
    /* 请在这里设计你的算法 */
    //初始化，清空pq,graph,mind,flag
    while (!pq.empty())
        pq.pop();
    for (int i = 1; i <= n; ++i)
        graph[i].clear();
    memset(mind, 127, sizeof(mind));
    memset(flag, 0, sizeof(flag));

    //建图，连接图中各边
    for (int i = 0; i < m; ++i)
    {
        graph[U[i]].push_back(make_pair(V[i], W[i]));//建立从起点为U[i]出发，到达V[i]
的权值为W[i]的边
        graph[V[i]].push_back(make_pair(U[i], W[i]));
    }

    //设置起点的最短路为0，并将起点加入优先队列中
    mind[s] = 0;
```

```
    pq.push(make_pair(mind[s], s));

    //执行Dijkstra算法
    while (!pq.empty())//循环条件暂定，可能会存在错误
    {
        int u = pq.top().second;//取出堆顶元素
        pq.pop();
        if (!flag[u])//每个节点最多做一次松弛，如果flag[u]为1，说明此节点距离起点更近，已经被访问过(即此节点已被激活)，此时应该跳过此节点(可视为此节点已经被划入与起点相连的集合中)
        {
            flag[u] = 1;
            for (vector<pii>::iterator it = graph[u].begin(); it != graph[u].end(); ++it)//枚举所有u出发的边
            {
                int v = it->first, w = it->second;
                if (mind[v] <= mind[u] + w)//若mind[v]距离原点更近，则说明
                    continue;
                mind[v] = mind[u] + w;
                pq.push(make_pair(mind[v], v));//将v伴随其最新的最短路长度加入优先队列
            }
        }
    }
    return mind[t];
}

// ================ 代码实现结束 ================

int main() {
    int n, m, s, t;
    scanf("%d%d%d%d", &n, &m, &s, &t);
    vector<int> U, V, W;
    U.clear();
    V.clear();
    W.clear();
    for (int i = 0; i < m; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        U.push_back(u);
        V.push_back(v);
        W.push_back(w);
    }
```

```
    printf("%d\n", shortestPath(n, m, U, V, W, s, t));
    return 0;
}
```

重编码-K

背景

小粽学习了哈夫曼树之后，自己设计了贪心算法，用两个队列就过掉了《重编码》这道题。

小粽想：那堆的算法有什么用呢？为了解决小粽的疑惑，邓老师委托小莉命制了这道题目……

描述

有一篇文章，文章包含 n 种单词，单词的编号从 1 至 n，第 i 种单词的出现次数为 w_i。
现在，我们要用一个 k 进制串（即只包含 $0,1,\ldots,k-1$ 的串） s_i 来替换第 i 种单词，使其满足如下要求：对于任意的 $1\le i<j\le n$，都有 s_i 不是 s_j 的前缀（这个要求是为了避免二义性）。
你的任务是对每个单词选择合适的 s_j，使得替换后的文章总长度（定义为所有单词**出现次数**与**替换它的 k 进制串的长度**乘积的总和）最小。求这个最小长度。
字符串 S_1（不妨假设长度为 n）被称为字符串 S_2 的前缀，当且仅当：S_2 的长度不小于 n，且 S_1 与 S_2 前 n 个字符组组成的字符串完全相同。

输入格式

第一行两个整数 n 和 k 。
第 2 行到第 $n+1$ 行，第 $i+1$ 行包含一个正整数 w_i，表示第 i 种单词的出现次数。

输出格式

表示整篇文章重编码后的最短长度

输入样例 1

```
5 3
1
3
5
10
3
```

输出样例 1

```
29
```

样例 2

点此下载。

限制

对于 100% 的数据，满足 $1 \le n \le 3 \times 10^5, 2 \le k \le 10, 1 \le w_i \le 10^9$；
对于 40% 的数据，满足 $n \le 3000$；
对于 35% 的数据，满足 $k = 2$。

代码

```cpp
#include <queue>
#include <cstdio>
#include <vector>
#include<iostream>
#pragma warning(disable:4996)
using namespace std;

typedef long long LL;

// ================== 代码实现开始 ======================
//Q:小根堆
```

```
/*Q作为小根堆，小根堆共有1+log(k)(n-1)(其中k为log的底数)层，为了用堆求Huffman树，故每次取出小根堆中前n个节点，以建立一个父节点
(注意新组成的父节点不能是任何一个单词的节点，但可以是其他单词节点的兄弟节点，这是为了避免语义重复)
由小的节点组成堆底，并组成父节点与其他次小的节点合并再形成父节点，以此循环
*/
//assQ:辅助队列，为节约形成父节点后还需插入到小根堆中所需要的O(logn)时间
priority_queue<LL, vector<LL>, greater<LL> > Q;
queue<LL> assQ;

// n: 初始点数
// k: 哈夫曼树的叉树
// w: w[i] 表示第i个点的点权
//返回值：最终编码的长度
LL solve(const vector<LL>& w, int n, int k)
{
    // 在这里设计你的算法

    for (int i = 0; i < n; i++)
    {
        Q.emplace(w[i]);
    }

    int r = ((k - 1) - ((n - 1) % (k - 1))) % (k - 1);//由于所给的不一定是满k叉树，当所给的条件不满足k叉树时，增加权值为0的节点
    n += r;
    while (r--)
        assQ.push(0);//为形成满k叉树(百度百科中的国外的满k叉树的定义)
    LL sum = 0;
    while (n != 1)//当n=1时，说明总长度合并完成
    {
        LL tmpf=0;//临时的父节点，组成新的父节点后压入assQ中以节约时间
        for (int i = 0; i < k; i++)
        {
            if (Q.empty()||!assQ.empty() && assQ.front() < Q.top())//若assQ中的节点较小，说明应当先从assQ中取出节点来合并
                                                                   //若Q为空，说明节点全在assQ中(例如全部节点的权值都一样的情况)
            {
                tmpf += assQ.front();
                assQ.pop();
                continue;
            }
            tmpf += Q.top();
```

```
            Q.pop();
        }
        sum += tmpf;
        assQ.push(tmpf);//将新产生的父节点压入assQ中
        n -= k - 1;
    }
    return sum;
}

// ================== 代码实现结束 ======================

int main()
{
    int n, k;
    scanf("%d%d", &n, &k);

    vector<LL> w;
    w.reserve(n);
    for (int i = 0; i < n; i++) {
        LL a;
        scanf("%lld", &a);
        w.push_back(a);
    }

    printf("%lld\n", solve(w, n, k));
    return 0;
}
```

数字三角形

时间限制：2 sec

空间限制：256 MB

问题描述

给定一个高度为 n 的“数字三角形”，其中第 i 行（1<=i<=n）有 i 个数。（例子如下图所示）

```
1
2 3
4 5 6
7 8 9 10
```

初始时，你站在“数字三角形”的顶部，即第一行的唯一一个数上。每次移动，你可以选择移动到当前位置正下方或者当前位置右下方的位置上。即如果你在 (i,j)（表示你在第 i 行从左往右数第 j 个数上，下同），你可以选择移动到 (i+1,j) 或 (i+1,j+1)。

你想让你经过的所有位置（包括起点和终点）的数字总和最大。求这个最大值。

输入格式

第一行一个正整数 n，表示数字三角形的大小。

第 2 行到第 n+1 行，第 i+1 行为 i 个用空格隔开的**非负**整数，描述数字三角形的第 i 行。

输出格式

一行一个整数，表示经过路径上数的最大总和。

样例输入

```
4
1
2 3
4 5 6
7 8 9 10
```

样例输出

```
20
```

样例解释

不停地向右下走即可。

数据范围

对于 50% 的数据，保证 n<=5。

对于 100% 的数据，保证 n<=1,000，保证数字三角形内的数不超过 10^6。

提示

[如果我们使用搜索算法，我们会在搜索时记录哪些信息呢？]

[当前到达的点的坐标、当前经过路径上数的总和！]

[总和显然是越大越好！]

[于是可以设计出状态：dp[i][j] 表示走到坐标为 (i,j) 的点时的最大总和。]

[很容易就可以设计出]

代码

```
#include <iostream>
#include <vector>
#include <algorithm>
#pragma warning(disable:4996)
using namespace std;

// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */
vector<vector<int>> dp;

// 本函数计算答案（最大经过位置数字总和）
// n：描述数字三角形大小，意义同题目描述
// a：描述整个数字三角形，第 i 行的第 j 个数存放在 a[i][j]
// 中（你需要特别注意的是，所有下标均从 1 开始，也就是说下标为 0 的位置将存放无效信息）
// 返回值：最大经过位置数字总和
int getAnswer(int n, vector<vector<int>> a) {
    /* 请在这里设计你的算法 */
    dp.resize(n + 1);
    for (int i = 0; i <= n; ++i)
    {
        dp[i].resize(i + 2);
        a[i][i + 1] = 0;
        a[i][0] = 0;
    }
```

```
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= i; ++j)
        {
            dp[i][j] = max(max(a[i - 1][j], a[i - 1][j - 1]), max(dp[i - 1][j], dp[i - 1][j - 1])) + a[i][j];
            dp[i][j] = dp[i][j];
        }
    int ans = 0;
    for (int i = 1; i <= n; ++i)
        ans = max(ans, dp[n][i]);
    return ans;
}

// ================ 代码实现结束 ================

int main() {
    int n;
    vector<vector<int>> a;
    scanf("%d", &n);
    a.resize(n + 1);
    for (int i = 0; i <= n; ++i)
        a[i].resize(i + 2);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= i; ++j)
            scanf("%d", &a[i][j]);
    int ans = getAnswer(n, a);
    printf("%d\n", ans);
    return 0;
}
```

背包问题 1

描述

n 种物品，每种物品有相应的价值和体积，同时物品还分为两类，一类是“单个物品”，即该种物品只有一个；一类是“多个物品”，即该种物品有无限个。

现在你有一个体积为 V 的背包，那么该装些什么物品到背包里使得价值之和最大呢？

输入

第一行包含两个正整数 n,V。

接下来 n 行，每行代表一种物品。每行的第一个数字表示该物品的种类（若为 0 表示“单个物品”，若为 1 表示“多个物品”），第二个数字表示该物品的价值，第三个数字表示该物品的体积。

输出

输出一个整数，表示最大的价值之和。

样例 1 输入

```
5 8
0 6 8
0 7 3
1 1 1
0 8 1
0 5 2
```

样例 1 输出

```
22
```

样例 1 解释

第三种物品有无限个，其余都是单个物品。

若我们放入物品 1，则背包已经装满，此时价值和为 6；

若我们放入物品 2、4、5，背包所剩体积为 8-3-1-2=2，此时价值和为 7+8+5=20；

若我们放入 8 个物品 3，背包装满，此时价值之和为 8×1=8；

若我们放入物品 2、4、5，再放两个物品 3，则背包装满，此时价值和为 7+8+5+2×1=22。

可以验证，最优答案就是 22。

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

对于 30%的数据，n,V ≤ 20；

对于 100%的数据，n,V ≤ 5000。

保证数据中所有的整数均为正整数，且不超过 5000。

时间：6 sec

空间：512 MB

提示

[经典的 01 背包和完全背包问题。]

代码

```
#include <iostream>
#include<vector>
#include<algorithm>
#pragma warning(disable:4996)
using namespace std;

// ================= 代码实现开始 =================

/* 请在这里定义你需要的全局变量 */
const int N = 5005;
//f:动态规划用的数组，f[i]表示容量为i的背包的最大价值
int f[N];

// n：物品个数
// V：背包的体积
// t：长度为n的数组，第i个元素若为0，表示物品i为单个物品；若为1，表示物品i为多个物品。（i下标从0开始，下面同理）
// w：长度为n的数组，第i个元素表示第i个物品的价值
// v：长度为n的数组，第i个元素表示第i个物品的体积
// 返回值：最大价值之和
int getAnswer(int n, int V, vector<int> t, vector<int> w, vector<int> v) {
    /* 请在这里设计你的算法 */
```

```
    for (int i = 0; i < n; ++i)
        if (t[i] == 0)//01背包，倒序枚举
            for (int j = V; j >= v[i]; --j)
                f[j] = max(f[j], f[j - v[i]] + w[i]);
        else
            for (int j = v[i]; j <= V; ++j)
                f[j] = max(f[j], f[j - v[i]] + w[i]);
    return f[V];
}

// ================ 代码实现结束 ================

int main() {
    int n, V;
    scanf("%d%d", &n, &V);
    vector<int> T, W, _V;
    for (int i = 0; i < n; ++i) {
        int t, w, v;
        scanf("%d%d%d", &t, &w, &v);
        T.push_back(t);
        W.push_back(w);
        _V.push_back(v);
    }
    printf("%d\n", getAnswer(n, V, T, W, _V));
    return 0;
}
```

背包问题 2

描述

n 个物品，每个物品有一个体积 v 和价值 w。现在你要回答，把一个物品丢弃后，剩下的物品装进一个大小为 V 的背包里能得到的最大价值是多少。

输入

输入的第一行包含一个正整数 n（n ≤ 5000）。

接下来 n 行，每行包含两个正整数 v 和 w（v,w ≤ 5000），分别表示一个物品的体积和价值。

接下来一行包含一个正整数 q（q ≤ 5000）,表示询问个数。

接下来 q 行，每行包含两个正整数 V 和 x（V ≤ 5000，x ≤ n），表示询问将物品 x 丢弃以后剩下的物品装进一个大小为 V 的背包能得到的最大价值。

输出

输出 q 行，每行包含一个整数，表示询问的答案。

样例 1 输入

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

样例 1 输出

```
4
5
5
```

样例 1 解释

有 3 个物品，第一个物品的体积为 3、价值为 5，第二个物品体积为 2、价值为 2，第三个物品体积为 1、价值为 2。

有 3 个询问：

第一个询问是问去掉 1 物品后剩下的 2、3 物品填进一个大小为 3 的背包能得到的最大价值。显然 2、3 物品都是可以放进背包的，所以最大价值为 2+2=4。

第二个询问是问去掉 2 物品后剩下的 1、3 物品填进一个大小为 3 的背包能得到的最大价值。若我们填 3 物品，我们只能得到价值 2；若我们填 1 物品，则可以得到价值 5。所以最大价值为 5。

第三个询问我们同样也是填 1 物品，最大价值为 5。

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

对于 30%的数据，n,q,v,V,w ≤ 10；

对于 50%的数据，n,q,v,V,w ≤ 200。

时间：10 sec

空间：512 MB

提示

[我们可以预处理“前缀背包”、“后缀背包”，然后询问时做“背包合并”的操作。]

代码

```
#include <iostream>
#include<vector>
#include<algorithm>
#pragma warning(disable:4996)
using namespace std;

// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */
const int N = 5005;
//d:前缀背包,d[i][j]表示物品1到物品i放进容量j的背包中的最大价值
//f:后缀背包,f[i][j]表示物品i到物品n放进容量j的背包中的最大价值
int d[N][N], f[N][N];

// n个物品，每个物品有体积价值，求若扔掉一个物品后装进给定容量的背包的最大价值
// n：如题
// w：长度为n+1的数组，w[i]表示第i个物品的价值（下标从1开始，下标0是一个数字-1，下面同理）
```

```
// v：长度为n+1的数组，v[i]表示第i个物品的体积
// q：如题
// qV：长度为q+1的数组，qV[i]表示第i次询问所给出的背包体积
// qx：长度为q+1的数组，qx[i]表示第i次询问所给出的物品编号
// 返回值：返回一个长度为q的数组，依次代表相应询问的答案
vector<int> getAnswer(int n, vector<int> w, vector<int> v, int q, vector<int> qV,
vector<int> qx) {
    /* 请在这里设计你的算法 */
    //计算前缀背包
    for (int i = 1; i <= n; ++i)
    {
        for (int V = 0; V < v[i]; ++V)
            d[i][V] = d[i - 1][V];//此处copy是为了使得i+2,i+3...行需要使用的时候能够直
接调用数据
        for (int V = v[i]; V <= 5000; ++V)
            d[i][V] = max(d[i - 1][V], d[i - 1][V - v[i]] + w[i]);
    }
    //计算后缀背包
    for (int i = n; i >= 1; --i)
    {
        for (int V = 0; V < v[i]; ++V)
            f[i][V] = f[i + 1][V];
        for (int V = v[i]; V <= 5000; ++V)
            f[i][V] = max(f[i + 1][V], f[i + 1][V - v[i]] + w[i]);
    }

    vector<int> ans;
    for (int k = 1; k <= q; ++k)
    {
        int x = qx[k], V = qV[k];
        int mx = 0;//记录当前询问的最优答案
        //抽去x，然后用前缀背包与后缀背包计算最优背包
        for (int i = 0; i <= V; ++i)
        {
            mx = max(mx, d[x - 1][i] + f[x + 1][V - i]);
        }
        ans.push_back(mx);
    }
    return ans;
}

// ================ 代码实现结束 ================

int main() {
```

```
    int n, q;
    vector<int> v, w, qv, qx;
    v.push_back(-1);
    w.push_back(-1);
    qv.push_back(-1);
    qx.push_back(-1);
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        int a, b;
        scanf("%d%d", &a, &b);
        v.push_back(a);
        w.push_back(b);
    }
    scanf("%d", &q);
    for (int i = 0; i < q; ++i) {
        int a, b;
        scanf("%d%d", &a, &b);
        qv.push_back(a);
        qx.push_back(b);
    }
    vector<int> ans = getAnswer(n, w, v, q, qv, qx);
    for (int i = 0; i < q; ++i)
        printf("%d\n", ans[i]);
    return 0;
}
```

刷油漆

描述

有 n 辆车排成一排，还有 m 种不同颜色的油漆，其中第 i 种油漆够涂 ai 辆车，同时所有油漆恰好能涂完 n 辆车。若任意两辆相邻的车颜色不能相同，有多少种涂油漆的方案？

输入

第一行包含一个正整数 m。

接下来一行包含 m 个正整数，第 i 个正整数表示 ai。

输出

输出一个整数，表示答案除以 23333 的余数。

样例 1 输入

```
3
2 1 3
```

样例 1 输出

```
10
```

样例 1 解释

10 个方案分别是：

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

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

n 为 ai 之和。

对于 50%的数据，n ≤ 10；

对于 100%的数据，m ≤ 20，ai ≤ 5。

时间：10 sec

空间：512 MB

提示

[注意到 ai≤5，所以我们可以将“还能涂 1 辆车的油漆种类数”、“还能涂 2 辆车的油漆种类数”、...、“还能涂 5 辆车的油漆种类数”设计成状态，思考一下便能得到转移。]

代码

```
#include <iostream>
#include <vector>
#include<string.h>
#pragma warning(disable:4996)
using namespace std;

// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */
const int N = 21, mo = 23333;
//f:记忆已经计算过的答案，减少重复的计算
int f[N][N][N][N][N][6];
//动态规划(记忆化搜索)，求当车数目为a+b+c+d+e时涂油漆的方案数
//a:够涂1辆车的油漆种类数
//b:够涂2辆车的油漆种类数
//c:够涂3辆车的油漆种类数
//d:够涂4辆车的油漆种类数
//f:够涂5辆车的油漆种类数
//last:若last==2则表示前一辆车图的油漆是从b中取出来的,last==3则表示是从c中取出来的，以此类推
//返回值:方案数
int dp(int a, int b, int c, int d, int e, int last)
{
    if ((a | b | c | d | e) == 0)//n==0,返回1，即空也表示1种方案
        return 1;
    if (f[a][b][c][d][e][last] != -1)//如果之前计算过答案，直接返回
```

```
        return f[a][b][c][d][e][last];
    long long ret = 0;
    //以下(last==2)等表达式的意思是:若这个表达式成立得到的是1，否则是0
    //假设有a>0，则最开始从属于a的油漆中选择一个，给第一辆车上色，然后减去这一个油漆并递归(最开始时last!=2，故乘a说明了有a中油漆种类数这么多种可能)
    if (a)
        ret += dp(a - 1, b, c, d, e, 1)*(a - (last == 2));//若last==2,则表示上一辆车是从b里取出来的放到了a里(即某一种油漆原先够涂2辆车，后来给前一辆车上色后只够涂1辆车了)，所以a中可以选择的油漆种类数要少一个
    if (b)
        ret += dp(a + 1, b - 1, c, d, e, 2)*(b - (last == 3));//同理,b少一个，则a会加一个
    if (c)
        ret += dp(a, b + 1, c - 1, d, e, 3)*(c - (last == 4));//同理
    if (d)
        ret += dp(a, b, c + 1, d - 1, e, 4)*(d - (last == 5));//同理
    if (e)
        ret += dp(a, b, c, d + 1, e - 1, 5)*e;//同理
    f[a][b][c][d][e][last] = ret % mo;
    return ret % mo;
}
//b:b[i]表示有多少种油漆够涂i辆车
int b[6];

// n辆车，m种油漆，第i种油漆够涂ai辆车，同时所有油漆恰好能涂完n辆车。若任意两辆相邻的车颜色不能相同，有多少种涂油漆的方案
// m：如题
// a：长度为m的数组，意义如题
// 返回值：方案数
int getAnswer(int m, vector<int> a) {
    /* 请在这里设计你的算法 */
    memset(f, -1, sizeof f);//一开始f初始化为-1，表示没有计算过
    for (int i = 0; i < m; ++i)//计算b数组
        b[a[i]]++;
    return dp(b[1], b[2], b[3], b[4], b[5], 0) % mo;
}

// ================ 代码实现结束 ================

int main() {
    int m;
    scanf("%d", &m);
    vector<int> a;
    for (int i = 0; i < m; ++i) {
```

```
        int x;
        scanf("%d", &x);
        a.push_back(x);
    }
    printf("%d\n", getAnswer(m, a));
    return 0;
}
```

n 皇后

描述

n 皇后问题：一个 n×n 的棋盘，在棋盘上摆 n 个皇后，满足任意两个皇后不能在同一行、同一列或同一斜线上的方案有多少种？

输入

第一行包含一个整数 n。

输出

输出一个整数，表示方案数。

样例 1 输入

```
4
```

样例 1 输出

```
2
```

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

一共 10 个测试点， 第 i 个测试点的 n=i+4。

时间：2 sec

空间：512 MB

提示

python 同学注意，标程后两个测试点 10s 都过不去，故自行打表。

[考察剪枝水平，剪枝剪得好（二进制剪枝）的才能过第 10 个测试点。]

代码

```
#include <iostream>
#pragma warning(disable:4996)
using namespace std;

// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */
//ans:总答案
//allOne:用于二进制&的全1数
int ans, allOne;

//搜索(用二进制来优化)
//pos:其二进制上的某个位置的1表示当前所在行的相应的位置(列)已经放了一个皇后
//left:其二进制上的某个位置的1表示当前所在行的相应的位置(是由于右对角线上已经有皇后)不能放置皇后
//right:其二进制上的某个位置的1表示当前所在行的相应的位置(是由于左对角线上已经有皇后)不能放置皇后

void dfs(int pos, int left, int right)
{
    //if判定语句暂定，可能会有错误
    if (pos== allOne)//当且仅当每一列都放了一个皇后那么整个棋盘已经放了n个合法皇后，故要终止
    {
        ++ans;
        return;
    }
    for (int t = allOne & (~(pos | left | right)); t;)//t表示可以放的集合对应的二进制数
```

```
    {
        int p = t & -t;//low bit:二进制最右边的1(负数为二进制原码取反加1，故通过此方法可以获得最低位的1)
        dfs(pos | p, (left | p) << 1, (right | p) >> 1);
        t ^= p;//消掉low bit
    }
}

// 一个n×n的棋盘，在棋盘上摆n个皇后，求满足任意两个皇后不能在同一行、同一列或同一斜线上的方案数
// n：上述n
// 返回值：方案数
int getAnswer(int n) {
    /* 请在这里设计你的算法 */
    ans = 0;
    allOne = (1 << n) - 1;
    dfs(0, 0, 0);
    return ans;
}

// ================ 代码实现结束 ================

int main() {
    int n;
    cin >> n;
    cout << getAnswer(n) << endl;
    return 0;
}
```

Rhizomys

描述

竹鼠养殖场有若干个小房间，有很多条双向道路连接着它们。

值得注意的是，在养殖场中，连接两个房间的道路可能不止一条。由于路上能看到的风景不同，我们认为这两条路是**不同**的。

同时，也可能存在一条道路是从一个房间出发又回到它自身，**但我们规定，从一个房间到它自己的最短距离为 0。**

为了不被吃掉，竹鼠们决定开始运动，运动的方式是从一个小房间经过若干个小房间（**中间经过的房间数可以为 0**）走到另一个小房间。

但竹鼠们也希望在锻炼过程中尽可能地偷懒，这意味着，竹鼠运动的路线总是沿**最短路**的。

现在，竹鼠们希望知道从 1 号房间分别到其他所有房间的运动路线的种数。由于它们害怕会因为写代码而被吃掉，所以它们找到了你帮忙。

输入

第一行是两个整数 N,MN,M，表示房间个数和连接房间的道路的条数。
接下来 mm 行，每行三个整数 u,v,cu,v,c，表示 uu 号房间与 vv 号房间之间存在一条长度为 cc 的双向道路。

输出

输出 NN 行，第 ii 行表示从 11 号房间到 ii 号房间的运动路径的种数，由于答案可能会很大，你只需要输出它模 911814911814 的结果。
当不存在任何一条从 11 号房间到 i 号房间的道路时，请输出 00。

输入样例 1

```
6 12
2 1 2
4 2 2
1 6 2
5 6 1
3 6 1
5 6 2
6 4 2
1 5 2
5 2 1
2 1 2
4 4 2
6 5 2
```

输出样例 1

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

样例 1 解释

以下用 EiEi 表示输入中的第 ii 条边。
到 1 的最短路长度为 0，只有 1 条。

到 2 的最短路长度为 2，有 2 条（ E1E1 ， E10E10 ）。
到 3 的最短路长度为 3，只有 1 条（ E3E3 -> E5E5 ）。
到 4 的最短路长度为 4，有 3 条
（ E3E3 -> E7E7 ， E1E1 -> E2E2 ， E10E10 -> E2E2 ）。
到 5 的最短路长度为 1，只有 1 条（ E9E9 ）。
到 6 的最短路长度为 2，只有 1 条（ E3E3 ）。

输入样例 2

点此下载。

限制

对于 20%的数据，1≤n≤100,1≤m≤20001≤n≤100,1≤m≤2000；
对于 50%的数据，1≤n≤1000,1≤m≤1500001≤n≤1000,1≤m≤150000；
对于 100%的数据，
1≤n≤20000,1≤m≤500000,1≤u,v≤n,1≤c≤101≤n≤20000,1≤m≤500000,1≤u,v≤n,1≤c≤10。

提示

[这道题统计最短路的数目，我们不妨考虑将所有处在最短路上的边提出来]

[提出这些边后形成了原图的一张子图，显然这张子图是不会有环的，否则与最短路的定义矛盾]

[既然没有环，那我们就可以愉快地做动态规划啦~]

[请同学们思考，如果使用 dijkstra 算法，需要先求最短路，再拓扑排序吗？]

代码

```
#include <iostream>
#include<vector>
#include<queue>
#include<vector>
#include<stack>
#include<string.h>
#pragma warning(disable:4996)
using namespace std;

// ================= 代码实现开始 =================

int Dist[20000];
long long State[20000];
bool outQ[20000];
const int mo = 911814;

/* 请在这里定义你需要的全局变量 */
struct QMember
{
    int node;
    int Dist;

    int friend operator > (QMember s1, QMember s2) {
        return s1.Dist > s2.Dist;
    }
};

struct Edge
{
    Edge *next=nullptr;//该点所拥有的下一条边
    int go=0;//该点所指向的节点号
    int cost;//边的权值
};

Edge *Head[20000];
QMember Node[20000];
```

```
priority_queue<QMember,vector<QMember>,greater<QMember>> Q;

// 给定n个点m条边的无向图，求1到其余每个点的最短路的数目
// n：如题意
// m：如题意
// U：大小为m的数组，表示m条无向边中的一个端点
// V：大小为m的数组，表示m条无向边中的另一个端点
// C：大小为m的数组，表示m条无向边的长度
void getAnswer(int n, int m, vector<int> U, vector<int>V, vector<int>C) {
    /* 请在这里设计你的算法 */
    memset(Dist, 999999, sizeof(Dist));

    //初始化点的序号
    for (int i = 0; i < n+1; i++)
    {
        Head[i] = new Edge();
        Node[i].node = i;
    }

    //处理相邻的点及其边
    int i = 0;
    while (i<m)
    {
        int u = U[i];
        int v = V[i];
        int c = C[i];

        if (u == v)
        {
            i++;
            continue;
        }
        //若u或v为1，说明此时应该将相邻点压入Q中以循环
        if (u == 1)
        {
            if (Dist[v] == c)
                State[v]++;
            else if (Dist[v] > c)
            {
                Dist[v] = c;
                State[v] = 1;
                Node[v].Dist = c;
                Q.push(Node[v]);
```

```
            }
        }
        if (v == 1)
        {
            if (Dist[u] == c)
                State[u]++;
            else if (Dist[u] > c)
            {
                Dist[u] = c;
                State[u] = 1;
                Node[u].Dist = c;
                Q.push(Node[u]);
            }
        }
        if(Head[u]->go==0)//若Head[u]->go==0说明v是u的第一条边所指向的节点
        {
            Head[u]->go = v;
            Head[u]->cost = c;
        }
        else
        {
            Edge *ed = new Edge();
            ed->go = v;
            ed->cost = c;
            ed->next = Head[u]->next;
            Head[u]->next = ed;
        }
        if (Head[v]->go == 0)//若Head[u]->go==0说明v是u的第一条边所指向的节点
        {
            Head[v]->go = u;
            Head[v]->cost = c;
        }
        else
        {
            Edge *ed = new Edge();
            ed->go = u;
            ed->cost = c;
            ed->next = Head[v]->next;
            Head[v]->next = ed;
        }
        i++;
    }
```

```
    outQ[1] = true;
    Dist[1] = 0;
    State[1] = 1;
    while (!Q.empty())
    {
        //Qtop:堆顶元素
        QMember QTop = Q.top();
        Q.pop();
        //curNode:当前堆顶的点
        //curDist:1->curNode的最短长度
        //curState:1->curNode的最短数目


        int curNode = QTop.node, curDist = Dist[curNode], curState = State[curNode];

        //如果curNode已经用于更新过，则跳过;否则打上出队标记
        if (outQ[curNode])
            continue;
        else
            outQ[curNode] = true;

        //枚举每一条与curNode相连的边
        //Head[curEdge]为curNode的第一条边
        for (Edge *curEdge = Head[curNode]; curEdge != nullptr; curEdge =
curEdge->next)
        {
            //nextNode:当前与curNode相连的边的另一个端点
            int nextNode = curEdge->go;

            //如果从1沿最短路到curNode，再到nextNode的路与原来的方案长度相同时
            //以下if中的内容暂定，可能会有错误
            if (Dist[nextNode] == curDist + curEdge->cost)
            {
                State[nextNode] += State[curNode];
                if (State[nextNode] > mo)
                    State[nextNode] % mo;
            }

            else if (Dist[nextNode] > curDist + curEdge->cost)
            {
                Dist[nextNode] = curDist + curEdge->cost;
                State[nextNode] = State[curNode];
                Node[nextNode].Dist = Dist[nextNode];
                Q.push(Node[nextNode]);//由于更新了新的最短路径，故原先nextNode所到其
```

```
他点的距离也需要更新
                }

            }
        }

        for (int i = 1; i <= n; i++)
        {
            printf("%d\n", State[i]);
        }
        return;
}

// ================ 代码实现结束 ================

int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    vector<int> U, V, C;
    for (int i = 0; i < m; ++i) {
        int u, v, c;
        scanf("%d%d%d", &u, &v, &c);
        U.push_back(u);
        V.push_back(v);
        C.push_back(c);
    }
    getAnswer(n, m, U, V, C);
    return 0;
}
```

最长公共子序列

时间限制：1 sec

空间限制：256 MB

问题描述

给定两个 1 到 n 的排列 A,B （即长度为 n 的序列，其中 [1,n] 之间的所有数都出现了**恰好**一次）。

求它们的最长公共子序列长度。

输入格式

第一行一个整数 n ，意义见题目描述。

第二行 n 个用空格隔开的正整数 A[1],…,A[n]，描述排列 A。

第三行 n 个用空格隔开的正整数 B[1],…,B[n]，描述排列 B。

输出格式

一行一个整数，表示 A,B 的最长公共子序列的长度。

样例输入

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

样例输出

```
2
```

样例解释

(2,3) 和 (2,4) 都可以是这两个序列的最长公共子序列。

数据范围

对于 80% 的数据，保证 n<=5,000。

对于 100% 的数据，保证 n<=50,000。

提示

[把 A 中的所有数替换成其在 B 中出现的位置，想一想，新序列的最长上升子序列和所求的东西有什么关系呢？]

代码

```
#include <iostream>
#include <vector>
#include <algorithm>
#pragma warning(disable:4996)
using namespace std;

// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */

const int inf = 1e9;

//pos:b中各元素出现位置
//f:f[i]表示长度为i的最长公共子序列的末尾最小可能元素
vector<int> pos, f;

// 计算最长公共子序列的长度
// n：表示两序列长度
// a：描述序列 a（这里需要注意的是，由于 a 的下标从 1 开始，因此 a[0] 的值为-1，你可以忽略它的值，只需知道我们从下标 1 开始存放有效信息即可）
// b：描述序列b（同样地，b[0] 的值为 -1）
// 返回值：最长公共子序列的长度
int LCS(int n, vector<int> a, vector<int> b) {
    /* 请在这里设计你的算法 */
    //初始化，调整pos,f数组的长度，并将f数组置初值
    pos.resize(n + 1);
    f.resize(n + 2, inf);
    for (int i = 1; i <= n; ++i)
        pos[b[i]] = i;//记录b序列中各元素出现位置
    for (int i = 1; i <= n; ++i)
        a[i] = pos[a[i]];
    f[0] = 0;//将f[0]置为0
    for (int i = 1; i <= n; ++i)
        *lower_bound(f.begin(), f.end(), a[i]) = a[i];
    return int(lower_bound(f.begin(), f.end(), n + 1) - f.begin()) - 1;
}

// ================ 代码实现结束 ================

int main() {
    int n, tmp;
```

```
    vector<int> a, b;
    scanf("%d", &n);
    a.clear();
    b.clear();
    a.push_back(-1);
    b.push_back(-1);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &tmp);
        a.push_back(tmp);
    }
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &tmp);
        b.push_back(tmp);
    }
    int ans = LCS(n, a, b);
    printf("%d\n", ans);
    return 0;
}
```

倒水问题

时间限制：10 sec

空间限制：256 MB

问题描述

邓老师有有 2 个容量分别为 n 单位、m 单位的没有刻度的杯子。初始，它们都是空的。

邓老师给了你 t 分钟时间。每一分钟，他都可以做下面 4 件事中的任意一件：

1. 用水龙头装满一个杯子。
2. 倒空一个杯子。
3. 把一个杯子里的水倒到另一个杯子里，直到一个杯子空了或者另一个杯子满了。
4. 什么都不做。

邓老师希望最后能获得 d 个单位的水，假设最后两个杯具中水量的总和为 x，那么邓老师的不满意度就为 |d-x|。

你希望邓老师尽可能地满意，于是请你计算邓老师的不满意度最小是多少。

输入格式

一行 4 个整数 n,m,t,d，分别表示两个杯具的容量、时间限制、以及邓老师的期望值。

输出格式

一行一个整数，表示邓老师最小的不满意度。

样例输入

```
7 25 2 16
```

样例输出

```
9
```

样例解释

你可以在第 1 分钟用水龙头装满任意一个杯子，并在第 2 分钟什么都不做，即可让邓老师的不满意度为 9。

可以证明不存在更优的解。

数据范围

本题共设置 16 个测试点。

对于前 1 个测试点，保证 t=1。

对于前 2 个测试点，保证 t<=2。

对于前 4 个测试点，保证 t<=4。

对于前 10 个测试点，保证 1<=n,m<=100，1<=t<=100，1<=d<=200。

对于所有的 16 个测试点，保证 1<=n,m<=2,000，1<=t<=200，1<=d<=4,000。

代码

```
#include <iostream>
#include<algorithm>
#include<string.h>
```

```cpp
#pragma warning(disable:4996)
using namespace std;

// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */
typedef pair<int,int> pii;
#define fi first
#define se second

const int N = 2003;

//mind:mind[i][j]表示从初始状态到达状态(i,j)需要的最少步数
//q:用数组模拟的队列
//qh:队头下标
//qt:队尾下标
int mind[N][N];
pii q[N*N];
int qh, qt;

//倒水函数，表示状态p经过第k种策略后倒水的状态
//p:初始状态
//k:策略(在下面详细阐释)
//n,m:两杯子容量
//返回值:最终状态
pii to(pii p, int k, int n, int m)
{
    if (k == 0)//策略为倒空杯子一
        return pii(0, p.se);
    else if (k == 1)//倒空杯子二
        return pii(p.fi, 0);
    else if (k == 2)//倒满杯子一
        return pii(n, p.se);
    else if (k == 3)//倒满杯子二
        return pii(p.first, m);

    //以下两个if暂定，有可能有错误
    else if (k == 4)//将杯子二的水向杯子一倒
        return pii(min(p.fi + p.se, n), max(0, p.se - (n - p.first)));
    else if (k == 5)//将杯子一的水向杯子二倒
        return pii(max(0, p.fi - (m - p.se)), min(p.fi + p.se, m));
    else//否则什么也不做
        return p;
}
```

```
// 计算答案的函数
// n, m, t, d：意义均与题目描述一致
// 返回值：即为答案
int getAnswer(int n, int m, int t, int d) {
    /* 请在这里设计你的算法 */
    //初始化，清空队列，将mind所有位置置为-1表示未访问
    memset(mind, -1, sizeof(mind));
    qh = qt = 0;
    q[++qt] = pii(0, 0);
    mind[0][0] = 0;

    //进行BFS
    while (qh < qt)
    {
        pii u = q[++qh];//取出队头元素
        if (mind[u.fi][u.se] == t)
            break;//如果已经进行了t步，那么没必要继续搜索了，退出循环即可
        for (int k = 0; k < 6; ++k)//枚举六种策略
        {
            pii v = to(u, k, n, m);
            //以下if的条件暂定，可能有错
            if (mind[v.fi][v.se] != -1)//判断目标状态是否曾到达过
                continue;
            q[++qt] = v;//加入队列
            mind[v.fi][v.se] = mind[u.first][u.second]+1;//记录mind
        }
    }

    int ans = d;
    for (int i = 0; i <= n; ++i)
        for (int j = 0; j <= m; ++j)
            if (mind[i][j] != -1)
                ans = min(ans,abs(d-i-j));
    return ans;
}

// ================ 代码实现结束 ================

int main() {
    int n, m, t, d;
    scanf("%d%d%d%d", &n, &m, &t, &d);
    int ans = getAnswer(n, m, t, d);
    printf("%d\n", ans);
```

```
    return 0;
}
```

奶牛吃草

时间限制：4 sec

空间限制：256 MB

问题描述

有一只奶牛在一条笔直的道路上（可以看做是一个数轴）。初始，它在道路上坐标为 K 的地方。

这条道路上有 n 棵非常新鲜的青草（编号从 1 开始）。其中第 i 棵青草位于道路上坐标为 x[i] 的地方。贝西每秒钟可以沿着道路的方向向前（坐标加）或向后（坐标减）移动一个坐标单位的距离。

它只要移动到青草所在的地方，就可以一口吞掉青草，它的食速很快，吃草的时间可以不计。

它要吃光所有的青草。不过，青草太新鲜了，在被吞掉之前，暴露在道路上的每棵青草每秒种都会损失一单位的口感。

请你帮它计算，该怎样来回跑动，才能在口感损失之和最小的情况下吃掉所有的青草。

输入格式

第一行两个用空格隔开的整数 n,k，分别表示青草的数目和奶牛的初始坐标。

第 2 行到第 n+1 行，第 i+1 行有一个整数 x[i]，描述第 i 棵青草的坐标。

输出格式

一行一个整数，表示吃掉所有青草的前提下，最小损失的口感之和。保证答案在 32 位有符号整数的范围内。

样例输入

```
4 10
1
```

```
9
11
19
```

样例输出

```
44
```

样例解释

先跑到 9，然后跑到 11，再跑到 19，最后到 1，可以让损失的口感总和为 29+1+3+11=44。可以证明不存在比这更优的解。

数据范围

对于 50% 的数据，保证 1≤n≤4，1≤k,x[i]≤20。 对于 80% 的数据，保证 1≤n≤100。 对于 100% 的数据，保证 1≤n≤1000，1≤k,x[i]≤10^6。

提示

[我们先从另一个角度看答案，即损失的总口感：从初始状态到奶牛吃掉第 1 棵草之间的时间（我们在下面把它叫做第 1 段时间），所有的 n 棵青草都在流失口感；……；从奶牛吃掉第 i 棵草到它吃掉第 i+1 棵草之间的时间（我们在下面把它叫做第 i+1 段时间），还没有被吃掉的 n-i 棵草都在流失口感；……]

[于是我们发现，第 i 段时间对答案的贡献，为这段时间的长度与 n-i+1 的乘积。]

[接着，我们再来关注最优策略。吃完一棵草后（包括初始时），奶牛的最优策略一定是直奔另一棵草。]

[由于奶牛不会飞，所以奶牛走过的所有路一定是一段连续的区间。]

[显然地，被奶牛经过过的地方，按最优策略，一定不会留下青草。]

[所以我们可以**将所有青草的坐标排序**（下面我们都使用排完序后的编号），然后用 dp[l][r][j] 表示吃完 [l,r] 范围内的青草时的最小答案，j 只有 0,1 两种取值，分别表示奶牛吃完最后一棵草停在青草 l 还是 r 上（只有可能是这两种情况，否则与上面的结论矛盾）。]

[于是我们就可以轻易地设计出状态转移方程：]

[dp[l][r][0]=min(dp[l+1][r][0]+(n-r+l)*abs(x[l]-x[l+1]),dp[l+1][r][1]+(n-r+l)*abs(x[l]-x[r]))]

[dp[l][r][1]=min(dp[l][r-1][1]+(n-r+l)*abs(x[r]-x[r-1]),dp[l][r-1][0]+(n-r+l)*abs(x[r]-x[l]))]

[边界为：dp[i][i][j]=abs(x[i]-k)*n（对于所有 1<=i<=n，j=0,1）]

[友情提示：请注意枚举顺序。]

代码

```
#include<iostream>
#include<vector>
#include<algorithm>
#pragma warning(disable:4996)
using namespace std;

const int N = 2000;
//dp:DP数组，dp[l][r][j]表示吃完[l,r]范围内的青草时的最小答案，j只有0，1两种取值，分别表示奶牛吃完最后一颗草停留在青草l或r上
int dp[N + 2][N + 2][2];
// 本函数求解答案（损失的最小口感和）
// n：青草棵数
// k：奶牛的初始坐标
// x：描述序列 x（这里需要注意的是，由于 x 的下标从 1 开始，因此 x[0] 的值为 -1，你可以忽略它的值，只需知道我们从下标 1 开始存放有效信息即可），意义同题目描述
// 返回值：损失的最小口感和
int getAnswer(int n, int k, vector<int> x) {
    sort(x.begin() + 1, x.end());
    for (int i = 1; i <= n; ++i)
        dp[i][i][0] = dp[i][i][1] = abs(x[i] - k) * n;//设置边界条件：只吃一棵草的情况下，答案应该是什么呢？
    for (int len = 1; len < n; ++len)
        for (int l = 1, r; (r = l + len) <= n; ++l) {
            //枚举空间（先枚举区间长度，再枚举左端点，求出右端点）
            //进行转移
            dp[l][r][0] =
                min(dp[l + 1][r][0] + (n - r + 1)* abs(x[l] - x[l + 1]),
                    dp[l + 1][r][1] + (n - r + 1)* abs(x[l] - x[r]));
            dp[l][r][1] =
                min(dp[l][r - 1][1] + (n - r + 1)* abs(x[r] - x[r - 1]),
                    dp[l][r - 1][0] + (n - r + 1)* abs(x[r] - x[l]));
        }
    return min(dp[1][n][0], dp[1][n][1]);
}

int main()
{
    int n, k,tmp;
```

```
    scanf("%d%d", &n,&k);
    vector<int> x;
    x.emplace_back(-1);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &tmp);
        x.emplace_back(tmp);
    }
    int ans = getAnswer(n,k,x);
    printf("%d\n", ans);
    return 0;
}
```

矩形

描述

给定两个矩阵，判断第二个矩阵在第一个矩阵的哪些位置出现过。

输入

输入的第一行包含四个正整数 a,b,c,d，表示第一个矩阵大小为 a×b，第二个矩阵的大小为 c×d。

接下来是一个 a×b 的矩阵。

再接下来是一个 c×d 的矩阵。

保证矩阵中每个数字都为正整数且不超过 100。

输出

若第二个矩阵在第一个矩阵的(i,j)位置出现（即出现位置的左上角），输出 i 和 j。若有多个位置，按字典序从小到大的顺序依次输出。

字典序：对于两个位置(a,b),(c,d)，若 a<c 则(a,b)比(c,d)小，若 a>c 则(a,b)比(c,d)大，若 a=c 则再像前边一样比较 b 和 d。

样例 1 输入

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

样例 1 输出

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

样例 1 解释

矩阵 2 在矩阵 1 的(1,1)、(1,3)、(3,2)这些位置出现了。

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

对于 50%的数据，a,b,c,d ≤ 50；

对于 100%的数据，a,b,c,d ≤ 1000。

时间：4 sec

空间：512 MB

提示

[对于长度为 LL 的数列 SS, SS 中最大的元素为 KK，我们设他的 hashhash 值 H(S)=S1C0+S1C1+...+SLCL-1H(S)=S1C0+S1C1+...+SLCL-1, 其中 CC 为任意大于 KK 的常数]
[对于不同的字符串 A,BA,B，H(A)≠H(B)H(A)≠H(B)]
[我们先来看看一维的情况，给定两个字符串 A,BA,B，我们怎么判断 AA 在 BB 中出现？显然我们可以用 hashhash 来判断。但是 hashhash 值太大了怎么办？取模呀！找一个比较好的质数 pp，对于字符串 A,BA,B，若 A=BA=B 则显然 H(A)modp=H(B)modpH(A)modp=H(B)modp；若 A≠BA≠B，H(A)modpH(A)modp 有一定概率会和 H(B)modpH(B)modp 相同。怎么办呢？我们再找第二个质数 pp，再来验证 H(A)modqH(A)modq 和 H(B)modqH(B)modq！可以证明，这样基本上是不会再出错了的。]
[拓展到二维。]

[对于第一个矩阵：]

[我们可以对每一个元素求向左长度为 dd 的矩阵元素的 hashhash 值，得到一个矩阵。]
[然后我们再用新矩阵，再做一次 hashhash，就是向上长度为 cc 的矩阵元素的 hashhash 值，得到新矩阵 XX。]
[接着我们将第二个字符串的 hashhash 值求出来，就是先每一行求一个 hashhash 值，再将这 cc 个 hashhash 值再 hashhash 一次变成一个数字，然后我们就去 XX 矩阵中找这个数字，找到多少个就说明第二个矩阵在第一个矩阵中出现了多少次。]
[时间复杂度 O(n2)O(n2)]

代码

```
#include <bits/stdc++.h>
using namespace std;
// ================ 代码实现开始 ================
typedef long long ll;
typedef pair<int, int> pii;
const int N = 1005;
const ll mo1 = 1e9 + 7;
const ll mo2 = 1e9 + 9;
const ll pw = 233;
ll h1[2][N][N], h2[2][N][N], bb[2][N][N];
// 为了减少复制开销，我们直接读入信息到全局变量中
// a, b：题目所述数组，a的大小为(n+1)*(m+1)，b的大小为(p+1)*(q+1)，下标均从1开始有意义（下标0无意义，你可以直接无视）
// n, m, p, q：题中所述
int a[N][N], b[N][N], n, m, p, q;

// 求出a中有哪些位置出现了b
```

```
// 返回值：一个pair<int, int>的数组，包含所有出现的位置
vector<pii> getAnswer() {
    ll p1 = 1, p2 = 1;
    for (int i = 1; i <= q; ++i) {
        p1 = p1 * pw % mo1;
        p2 = p2 * pw % mo2;
    }
    p1 = (mo1 - p1) % mo1;
    p2 = (mo2 - p2) % mo2;
    for (int i = 1; i <= n; ++i) {
        ll t1 = 0, t2 = 0;
        for (int j = 1; j <= m; ++j) {
            if (j < q) {
                t1 = (t1 * pw + a[i][j]) % mo1;
                t2 = (t2 * pw + a[i][j]) % mo2;
            }
            else {
                t1 = h1[0][i][j] = (t1 * pw + a[i][j] + p1 * a[i][j - q]) % mo1;
                t2 = h2[0][i][j] = (t2 * pw + a[i][j] + p2 * a[i][j - q]) % mo2;
            }
        }
    }
    p1 = 1, p2 = 1;
    for (int i = 1; i <= p; ++i) {
        p1 = p1 * pw % mo1;
        p2 = p2 * pw % mo2;
    }
    p1 = (mo1 - p1) % mo1;
    p2 = (mo2 - p2) % mo2;
    for (int j = 1; j <= m; ++j) {
        ll t1 = 0, t2 = 0;
        for (int i = 1; i <= n; ++i) {
            if (i < p) {
                t1 = (t1 * pw + h1[0][i][j]) % mo1;
                t2 = (t2 * pw + h2[0][i][j]) % mo2;
            }
            else {
                t1 = h1[1][i][j] = (t1 * pw + h1[0][i][j] + p1 * h1[0][i - p][j]) %
mo1;
                t2 = h2[1][i][j] = (t2 * pw + h2[0][i][j] + p2 * h2[0][i - p][j]) %
mo2;
            }
        }
    }
```

```
    for (int i = 1; i <= p; ++i)
        for (int j = 1; j <= q; ++j) {
            bb[0][i][j] = (bb[0][i][j - 1] * pw + b[i][j]) % mo1;
            bb[1][i][j] = (bb[1][i][j - 1] * pw + b[i][j]) % mo2;
        }
    p1 = p2 = 0;
    for (int i = 1; i <= p; ++i) {
        p1 = (p1 * pw + bb[0][i][q]) % mo1;
        p2 = (p2 * pw + bb[1][i][q]) % mo2;
    }
    vector<pii> ans;
    for (int i = p; i <= n; ++i)
        for (int j = q; j <= m; ++j)
            if (h1[1][i][j] == p1 && h2[1][i][j] == p2)
                ans.push_back(pii(i - p + 1, j - q + 1));
    return ans;
}
// ================ 代码实现结束 ================

// ================ 代码实现结束 ================

int main() {
    scanf("%d%d%d%d", &n, &m, &p, &q);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            scanf("%d", &a[i][j]);
    for (int i = 1; i <= p; ++i)
        for (int j = 1; j <= q; ++j)
            scanf("%d", &b[i][j]);
    vector<pair<int, int>> ans = getAnswer();
    for (int i = 0; i < int(ans.size()); ++i)
        printf("%d %d\n", ans[i].first, ans[i].second);
    return 0;
}
```

回文串

描述

给定一个字符串，求出该字符串有多少子串是回文串。

子串：字符串中连续的一段。比如字符串 abcd 里，bc、abc、a、bcd 都是子串。

回文串：字符串倒序写出来和该字符串相同。比如 aba，倒序写出来也是 aba，故 aba 是回文串。而 abab 不是回文串，因为倒过来写是 baba。

输入

输入一个字符串。

输出

输出子串是回文串的个数。

样例 1 输入

```
abab
```

样例 1 输出

```
6
```

样例 1 解释

abab，abab，abab
abab，abab，abab

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

对于 50%的数据，字符串长度不超过 500；

对于 70%的数据，字符串长度不超过 2000；

对于 100%的数据，字符串长度不超过 500000。

字符串为 26 个小写字母组成。

时间：2 sec

空间：512 MB

提示

[[[https://segmentfault.com/a/1190000003914228]

[这篇文章是求最长的回文串的，那么如何求回文串的数目呢？可以发现 manacher 算法将每个位置为中心能延展出的最长回文串求出来了，那么这个最长回文串的一半（上取整）就是以该点作为中心的回文串数目。]

[注意答案要用 long long。]

代码

```
#include <iostream>
#include<algorithm>
#pragma warning(disable:4996)
using namespace std;

// ================= 代码实现开始 =================

const int N = 500005;

/* 请在这里定义你需要的全局变量 */
//s:变化后的s数组
//len:每个位置能向左拓展的最大长度(回文半径)
char s[N * 2];
int len[N * 2];

// 计算str中有多少个回文子串
// 返回值：子串的数目
long long getAnswer(string str) {
    /* 请在这里设计你的算法 */
    int n = str.size();

    //将字符串变为#a#b#c#d#这样的形式，下方认为#是0
    for (int i = n; i; --i)
        s[i << 1] = str[i - 1], s[i << 1 | 1] = 0;

    //边界(位置0和n+1)设为不同于#的东西(即1和2)
    n = n << 1 | 1;
    s[1] = 0, s[0] = 1, s[n + 1] = 2;

    //manacher算法
```

```
    int cur = 1;
    long long ans = 0;
    for (int i = 2; i <= n; ++i)
    {
        int &now = len[i], pos = (cur << 1) - i;
        now = max(min(len[pos], cur + len[cur] - i), 0);
        while (s[i - now - 1] == s[i + now + 1])
            ++now;
        if (i + now > cur + len[cur])
            cur = i;
        ans += ((now + 1) >> 1);//相当于now/2取上界
    }

    return ans;
}

// ================ 代码实现结束 ================

char _s[N];

int main() {
    scanf("%s", _s + 1);
    printf("%lld\n", getAnswer(_s + 1));
    return 0;
}
```

邓老师数

时间限制：1 sec

空间限制：256 MB

问题描述

众所周知，大于 1 的自然数中，除了 1 与其本身外不再有其他因数的数称作**质数（素数）**。

对于大于 1 的不是质数的自然数，我们又称作**合数**。

参加了邓老师算法训练营的小 Z 突发奇想，定义了新的数：所有合数中，除了 1 与其本身外，其他因数均为质数的数，称作**邓老师数**。

现在，小 Z 给定两个数 n,k，其中 k 的取值为 0 或 1。如果 k=0，小 Z 希望你告诉他所有**不超过** n 的质数；如果 k=1，小 Z 希望你告诉他所有**不超过** n 的邓老师数。

输入格式

一行两个用空格隔开的整数 n,k，意义见题目描述。

输出格式

对于每个找到的质数或邓老师数，输出一行一个整数表示这个你找到的数。

请升序输出所有答案。

样例输入

```
9 1
```

样例输出

```
4
6
9
```

样例解释

4 除去 1 与其本身外的因子有 2，均为质数，因此 4 是邓老师数。

6 除去 1 与其本身外的因子有 2,3，均为质数，因此 6 是邓老师数。

9 除去 1 与其本身外的因子有 3，均为质数，因此 9 是邓老师数。

8 除去 1 与其本身外的因子有 2,4，由于 4 不是质数，因此 8 **不是**邓老师数。

数据范围

本题共设置 8 个测试点，测试点编号从 1 至 8。

对于前 4 个测试点，保证 n<=1,000。

对于编号为偶数的测试点，保证 k=0；对于编号为奇数的测试点，保证 k=1。

对于所有的 8 个测试点，保证 n<=2*10^5。

提示

[对于求质数的问题，可以直接用 Eratosthenes 筛法求解。]

[对于求邓老师数的问题，考虑 Eratosthenes 筛法中“筛去”合数的逻辑，是否可以对其略作修改，使之支持筛出“非邓老师数”呢？]

代码

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

// ================ 代码实现开始 ================

/* 请在这里定义你需要的全局变量 */
//isPrime:若isPrime[i]==true，表示i是质数
//isDent:若isDent[i]==true，表示i是邓老师数
vector<bool> isPrime, isDeng;

// 本函数求解质数或邓老师数（将这两个功能合并在了一起）
// n, k：意义均与题目描述相符
// 返回值：如果 k=0，则将所求的质数按从小到大的顺序放入返回值中；如果 k=1，则将所求的邓老师数按从小到大的顺序放入返回值中。
vector<int> getAnswer(int n, int k) {
    /* 请在这里设计你的算法 */
    isPrime.resize(n + 1, 1);
    isDeng.resize(n + 1, 1);
    vector<int> ans;

    for (int i = 2; i <= n; ++i)
    {
        if (isPrime[i])
            isDeng[i] = 0;
        if (k == 0 && isPrime[i])
            ans.push_back(i);
        if (k == 1 && isDeng[i])
            ans.push_back(i);
        for (int j = i + i; j <= n; j += i)
        {
            isPrime[j] = 0;
            if (!isPrime[i])
                isDeng[j] = false;
        }
```

```
    }
    return ans;
}

// ================ 代码实现结束 ================

int main() {
    int n, k;
    scanf("%d%d", &n, &k);
    vector<int> ans = getAnswer(n, k);
    for (vector<int>::iterator it = ans.begin(); it != ans.end(); ++it)
        printf("%d\n", *it);
    return 0;
}
```

子序列

描述

给定一个字符串，求出该字符串有多少不同的子序列。

子序列：字符串中按顺序抽出一些字符得到的串。比如字符串 abcd 里，ab、ac、ad、abc、acd 都是子序列。

输入

输入一个字符串。

输出

输出不同的子序列的个数除以 23333 得到的余数。

样例 1 输入

```
ababc
```

样例 1 输出

```
23
```

样例 1 解释

有这些子序列：

a，b，c，aa，ab，ac，ba，bb，bc，aba，abb，abc，aab，aac，bab，bac，bbc，abab，abac，abbc，aabc，babc，ababc

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

对于 50%的数据，字符串长度不超过 15；

对于 100%的数据，字符串长度不超过 500000。

字符串为 26 个小写字母组成。

时间：2 sec

空间：512 MB

提示

[令 f(i)f(i)为前 ii 中本质不同的的子序列个数,令 pre(i)pre(i)表示字符 s_i 之前出现的位置,则]
$$[f(i)=\{f(i-1)+f(i-1)+1 pre(i)=0 f(i-1)+f(i-1)-f(pre(i)-1) pre(i)\neq 0]$$
[答案等于 f(n)f(n).]

代码

```cpp
#include <bits/stdc++.h>
#include<string>
using namespace std;

// ================ 代码实现开始 ================

const int N = 500005, mo = 23333;
```

```
int f[N], p[N], last[26];
// 为了减少复制开销，我们直接读入信息到全局变量中
// s：题目所给字符串
int n;
char s[N];

// 求出字符串s有多少不同的子序列
// 返回值：s不同子序列的数量，返回值对mo取模
int getAnswer() {
    for (int i = 1; i <= n; ++i) {
        int a = s[i] - 'a';
        p[i] = last[a];
        last[a] = i;
    }
    f[0] = 0;
    for (int i = 1; i <= n; ++i) {
        f[i] = (p[i] == 0) ?
            (f[i - 1] + f[i - 1] + 1) : (f[i - 1] + f[i - 1] - f[p[i] - 1]);
        f[i] %= mo;
    }
    return (f[n] + mo) % mo;
}

// ================= 代码实现结束 =================

int main() {
    scanf("%s", s + 1);
    n = strlen(s + 1);
    printf("%d\n", getAnswer());
    return 0;
}
```

前缀

描述

给定 n 个字符串，再询问 m 次，每个询问给出一个字符串，求出这个字符串是 n 个字符串里，多少个串的前缀。

前缀：从头开始的一段连续子串。比如字符串 ab 是字符串 abcd 的前缀，也是字符串 ab（自身）的前缀，但不是 bab 的前缀。

输入

第一行包含两个正整数 n,m。

接下来 n 行，每行表示一个字符串，表示给定的 n 个字符串中的一个。

再接下来 m 行，每行一个字符串，表示询问的字符串。

输出

输出 m 行，每行表示询问的答案。

样例 1 输入

```
5 4
ab
abc
ab
ba
bb
a
b
ab
abc
```

样例 1 输出

```
3
2
3
1
```

样例 1 解释

字符串 a 是 ab、abc、ab 的前缀；

字符串 b 是 ba、bb 的前缀；

字符串 ab 是 ab、abc、ab 的前缀；

字符串 abc 是 abc 的前缀。

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

对于 50%的数据，n,m ≤ 500；

对于 100%的数据，n,m ≤ 5000。

字符串为 26 个小写字母组成，且单个长度不超过 500，n 个字符串的长度之和不超过 1000000。

时间：10 sec

空间：512 MB

提示

[trie 树基本题。]

代码

```
#include <bits/stdc++.h>
using namespace std;
// ================ 代码实现开始 ================
const int M = 505, L = 1000005;
// c：trie树上的边，c[x][y]表示从节点x出发（x从1开始），字符为y的边（y范围是0到25）
// sz：sz[x]表示x节点的子树中终止节点的数量（子树包括x自身）
// cnt：trie树上节点的数目
int c[L][26], sz[L], cnt;
// 将字符串s加入到trie树中
// s：所要插入的字符串
void add(char *s) {
    int x = 0;
```

```
    for (; *s; ++s) {
        int y = *s - 'a';
        if (!c[x][y])
            c[x][y] = ++cnt;
        x = c[x][y];
    }
    ++sz[x];
}
// 用于计算sz数组
// x：当前节点
void dfs(int x) {
    for (int y = 0; y < 26; ++y) {
        int z = c[x][y];
        if (z != 0) {
            dfs(z);
            sz[x] += sz[z];
        }
    }
}
// 用字符串s沿着trie树上走，找到相应的节点
// s：所给字符串
// 返回值：走到的节点
int walk(char *s) {
    int x = 0;
    for (; *s; ++s) {
        int y = *s - 'a';
        if (!c[x][y])
            return 0;
        x = c[x][y];
    }
    return x;
}
// ================= 代码实现结束 =================

char s[M];

int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    for (; n--;) {
        scanf("%s", s);
        add(s);
    }
    dfs(0);
```

```
    sz[0] = 0;
    for (; m--;) {
        scanf("%s", s);
        printf("%d\n", sz[walk(s)]);
    }
    return 0;
}
```

最大间隙

时间限制：10 sec

空间限制：1 GB

问题描述

给定长度为 n 的数组 a，其中每个元素都为 [0,2^k) 之间的整数，请求出它们在实数轴上相邻两个数之间的最大值（即 maxGap）。

由于 n 可能很大，为了避免过大的输入、输出规模，我们会在程序内部生成数据，并要求你输出排序后序列的哈希值。具体方法如下（用 c++代码展示）：

```
typedef unsigned int u32;
u32 nextInt(u32 x){
        x^=x<<13;
        x^=x>>17;
        x^=x<<5;
        return x;
}
void initData(u32 *a,int n,int k,u32 seed){
        for (int i=0;i<n;++i){
                seed=nextInt(seed);
                a[i]=seed>>(32-k);
        }
}
```

输入将会给定 n,k,seed。

你可以调用 initData(a,n,k,seed) 来获得需要排序的 a 数组。

输入格式

一行 3 个用空格隔开的整数 n,k,seed，意义见题目描述。

输出格式

一行一个整数，表示最大间隙（即 maxGap）。

样例输入

```
5 4 233333
```

样例输出

```
5
```

样例解释

生成的序列应为 4 10 13 9 4，最大间隙为 9-4=5。

数据范围

本题共设置 4 组数据。

对于第 1 组数据，保证 n=1000，k=16。

对于第 2 组数据，保证 n=5*10^6，k=32。

对于第 3 组数据，保证 n=2^26=67,108,864，k=16。

对于第 4 组数据，保证 n=2^26=67,108,864，k=32。

保证给定的 seed 在 32 位**无符号**整数的范围内。

提示

[对于 k=16 的数据，使用桶排序即可。]

[对于 k=32 的数据，可以用邓老师上课讲的算法哦！]

[进一步地，如何设置桶的大小来避免较慢的除法运算呢？（提示：可以考虑位运算！）]

代码

```
#include <bits/stdc++.h>
using namespace std;

// ================ 代码实现开始 ================
typedef unsigned int u32;
// 以下代码不需要解释，你只需要知道这是用于生成数据的就行了
u32 nextInt(u32 x) {
    x ^= x << 13;
    x ^= x >> 17;
    x ^= x << 5;
    return x;
}
void initData(u32* a, int n, int k, u32 seed) {
    for (int i = 0; i < n; ++i) {
        seed = nextInt(seed);
        a[i] = seed >> (32 - k);
    }
}
// 以上代码不需要解释，你只需要知道这是用于生成数据的就行了
const int N = 67108864;
u32 a[N + 1];
u32 l[N + 1], r[N + 1];
// 这是求解答案的函数，你需要对全局变量中的 a 数组求解 maxGap 问题
// n, k：意义与题目描述相符
// 返回值：即为答案（maxGap）
u32 maxGap(int n, int k) {
    const int m = 1 << 26;
    memset(l, -1, sizeof(int)*m);
    memset(r, -1, sizeof(int)*m);
    const int _k = max(k - 26, 0);
    for (int i = 0; i < n; ++i) {
        u32 bl = a[i] >> _k;
        if (l[bl] == -1)
            l[bl] = r[bl] = a[i];
        else if (a[i] < l[bl])
            l[bl] = a[i];
        else if (a[i] > r[bl])
            r[bl] = a[i];
    }
    u32 last = a[0];
```

```
    u32 ans = 0;
    for (int i = 0; i < m; ++i)
        if (l[i] != -1) {
            if (last > l[i])
                last = l[i];
            if (l[i] - last > ans)
                ans = l[i] - last;
            last = r[i];
        }
    return ans;
}
// ================ 代码实现结束 ================

int main() {
    int n, k;
    u32 seed;

    scanf("%d%d%u", &n, &k, &seed);
    initData(a, n, k, seed);

    u32 ans = maxGap(n, k);

    printf("%u\n", ans);
    return 0;
}
```

基数排序

时间限制：10 sec

空间限制：1 GB

问题描述

给定 n 个 [0,2^k) 之间的整数，请你将它们**升序**排序。

由于 n 可能很大，为了避免过大的输入、输出规模，我们会在程序内部生成数据，并要求你输出排序后序列的哈希值。具体方法如下（用 c++代码展示）：

```
typedef unsigned int u32;
```

```
u32 nextInt(u32 x){
	x^=x<<13;
	x^=x>>17;
	x^=x<<5;
	return x;
}
void initData(u32 *a,int n,int k,u32 seed){
	for (int i=0;i<n;++i){
		seed=nextInt(seed);
		a[i]=seed>>(32-k);
	}
}
u32 hashArr(u32 *a,int n){
	u32 x=998244353,ret=0;
	for (int i=0;i<n;++i){
		ret^=(a[i]+x);
		x=nextInt(x);
	}
	return ret;
}
```

输入将会给定 n,k,seed。

你可以调用 initData(a,n,k,seed) 来获得需要排序的 a 数组。

排序后，你可以调用函数 hashArr(a,n) 来获得我们希望你输出的哈希值。

输入格式

一行 3 个用空格隔开的整数 n,k,seed，意义见题目描述。

输出格式

一行一个整数，表示我们希望你输出的哈希值。

样例输入

```
5 4 233333
```

样例输出

```
740640512
```

样例解释

生成的序列应为 4 10 13 9 4，排序后的结果应为 4 4 9 10 13。

数据范围

本题共设置 4 组数据。

对于第 1 组数据，保证 n=1000，k=16。

对于第 2 组数据，保证 n=5*10^6，k=32。

对于第 3 组数据，保证 n=10^8，k=16。

对于第 4 组数据，保证 n=10^8，k=32。

保证给定的 seed 在 32 位**无符号**整数的范围内。

提示

[对于 k=16 的数据，使用基数排序即可。]

[对于 k=32 的数据，不妨考虑两次基数排序哦！（即先排二进制下后 16 位，再排二进制下前 16 位）]

代码

```cpp
#include <bits/stdc++.h>
using namespace std;
#include <bits/stdc++.h>
using namespace std;
// ================ 代码实现开始 ================
typedef unsigned int u32;
// 以下代码不需要解释，你只需要知道这是用于生成数据的就行了
```

```
u32 nextInt(u32 x) {
    x ^= x << 13;
    x ^= x >> 17;
    x ^= x << 5;
    return x;
}
void initData(u32* a, int n, int k, u32 seed) {
    for (int i = 0; i < n; ++i) {
        seed = nextInt(seed);
        a[i] = seed >> (32 - k);
    }
}
u32 hashArr(u32* a, int n) {
    u32 x = 998244353, ret = 0;
    for (int i = 0; i < n; ++i) {
        ret ^= (a[i] + x);
        x = nextInt(x);
    }
    return ret;
}
// 以上代码不需要解释，你只需要知道这是用于生成数据的就行了
const int N = 100000000;
u32 a[N + 1];
u32 _a[N + 1];
const int m = 16;
const int B = 1 << m;
const int b = B - 1;
int sum[B + 1];
// 这是你的排序函数，你需要将全局变量中的 a 数组进行排序
// n, k：意义与题目描述相符
// 返回值：本函数需不要返回值（你只需要确保 a 被排序即可）
void sorting(int n, int k) {
    memset(sum, 0, sizeof(sum));
    for (int i = 0; i < n; ++i)
        ++sum[a[i] & b];
    for (int i = 1; i < B; ++i)
        sum[i] += sum[i - 1];
    for (int i = n - 1; i >= 0; --i)
        _a[--sum[a[i] & b]] = a[i];
    memset(sum, 0, sizeof(sum));
    for (int i = 0; i < n; ++i)
        ++sum[(_a[i] >> m) & b];
    for (int i = 1; i < B; ++i)
        sum[i] += sum[i - 1];
```

```
    for (int i = n - 1; i >= 0; --i)
        a[--sum[(_a[i] >> m) & b]] = _a[i];
}
// ================ 代码实现结束 ================

int main() {
    int n, k;
    u32 seed;
    scanf("%d%d%u", &n, &k, &seed);
    initData(a, n, k, seed);

    sorting(n, k);

    u32 ans = hashArr(a, n);
    printf("%u\n", ans);
    return 0;
}
```

字符串匹配

时间限制：1 sec

空间限制：256 MB

问题描述

给定一个大串 A 和一个模式串 B，求 B 在 A 的哪些位置出现（输出这些出现位置的起始位置，下标从 0 开始）。

输入格式

第一行一个正整数 n，表示串 A 的长度。

第二行包含一个长度为 n 的串 A。

第三行一个正整数 m，表示串 B 的长度。

第四行包含一个长度为 m 的串 B。

保证串 A,B 只包含小写字母。

输出格式

对于每个 B 在 A 中出现的位置，输出单独一行一个整数表示该次出现的起始位置。

对于所有的这些位置，请升序（从小到大）输出。

样例输入

```
7
abcabca
4
abca
```

样例输出

```
0
3
```

数据范围

对于 60% 的数据，保证 m<=10。

对于另外 20% 的数据，保证 A 的每一位在所有小写字母中等概率随机，且 B 为 A 中截取的一段。

对于 100% 的数据，保证 n<=500,000，m<=100,000。

提示

[此题是单模匹配算法的练习题。]

[可以尝试暴力匹配、KMP 算法、Boyer-Moore 算法、Rabin-Karp 算法，并比较它们的效果。]

代码

```
#include <bits/stdc++.h>
using namespace std;
```

```cpp
// ================ 代码实现开始 ================
vector<int> Next;
// 这是匹配函数，将所有匹配位置求出并返回
// n：串 A 的长度
// A：题目描述中的串 A
// m：串 B 的长度
// B：题目描述中的串 B
// 返回值：一个 vector<int>，从小到大依次存放各匹配位置
vector<int> match(int n, string A, int m, string B) {
    Next.resize(m);
    int j = Next[0] = -1;
    for (int i = 1; i < m; ++i) {
        while (j >= 0 && B[i] != B[j + 1])
            j = Next[j];
        if (B[i] == B[j + 1])
            ++j;
        Next[i] = j;
    }
    j = -1;
    vector<int> ans;
    for (int i = 0; i < n; ++i) {
        while (j >= 0 && A[i] != B[j + 1])
            j = Next[j];
        if (A[i] == B[j + 1])
            ++j;
        if (j == m - 1)
            ans.push_back(i - m + 1);
    }
    return ans;
}
// ================ 代码实现结束 ================

int main() {
    ios::sync_with_stdio(false);
    int n, m;
    string A, B;
    cin >> n >> A;
    cin >> m >> B;
    vector<int> ans = match(n, A, m, B);
    for (vector<int>::iterator it = ans.begin(); it != ans.end(); ++it)
        cout << *it << '\n';
    return 0;
}
```

凸包

描述

给定 n 个二维平面上的点，求他们的凸包。

输入

第一行包含一个正整数 n。

接下来 n 行，每行包含两个整数 x,y，表示一个点的坐标。

输出

令所有在凸包极边上的点依次为 p1,p2,...,pm（序号），其中 m 表示点的个数，请输出以下整数：

(p1 × p2 × ... × pm × m) mod (n + 1)

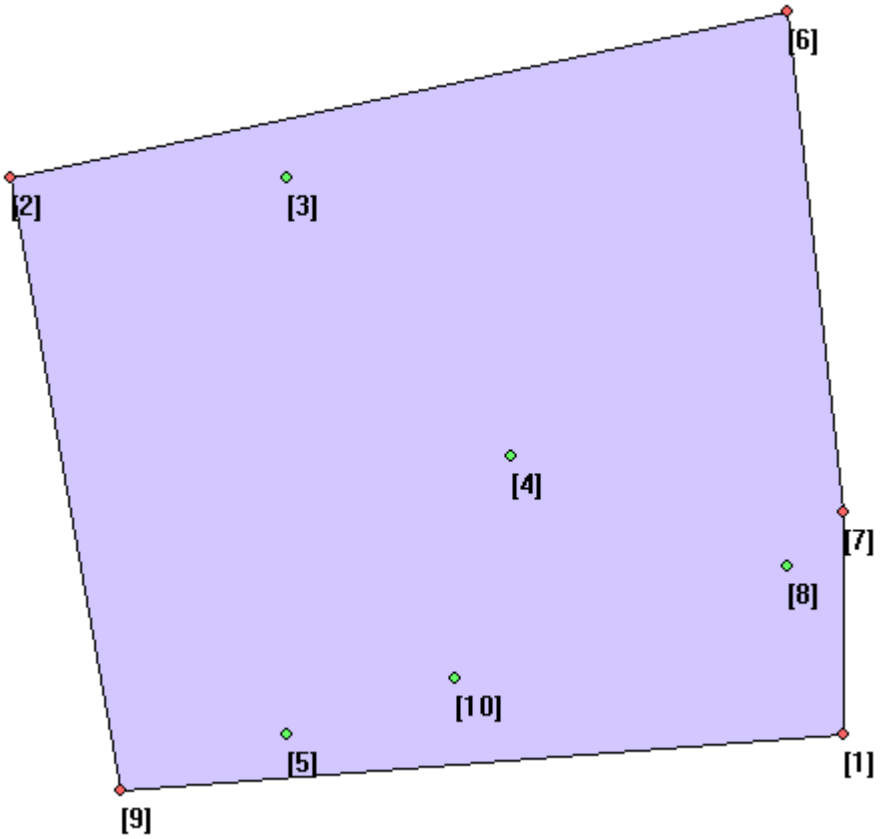
样例 1 输入

```
10
7 9
-8 -1
-3 -1
1 4
-3 9
6 -4
7 5
6 6
-6 10
0 8
```

样例 1 输出

```
7
```

样例 1 解释



所以答案为(9 × 2 × 6 × 7 × 1 × 5) % (10 + 1) = 7

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

3 ≤ n ≤ 10^5

所有点的坐标均为范围(-10^5, 10^5)内的整数，且没有重合点。每个点在(-10^5, 10^5) × (-10^5, 10^5)范围内均匀随机选取

极边上的所有点均被视作极点，故在输出时亦不得遗漏

时间：4 sec

空间：512 MB

代码

```
#include <bits/stdc++.h>
using namespace std;
// ================ 代码实现开始 ================
typedef long long ll;
const int N = 300005;
// 存储二维平面点
struct ip {
    int x, y, i;
    ip(int x = 0, int y = 0) : x(x), y(y), i(0) { }
    void ri(int _i) {
        scanf("%d%d", &x, &y);
        i = _i;
    }
};
// iv表示一个向量类型，其存储方式和ip一样
typedef ip iv;
// 先比较x轴再比较y轴，
bool operator < (const ip &a, const ip &b) {
    return a.x == b.x ? a.y < b.y : a.x < b.x;
}
// 两点相减得到的向量
iv operator - (const ip &a, const ip &b) {
    return iv(a.x - b.x, a.y - b.y);
}
// 计算a和b的叉积（外积）
ll operator ^ (const iv &a, const iv &b) {
    return (ll)a.x * b.y - (ll)a.y * b.x;
}
// 计算二维点数组a的凸包，将凸包放入b数组中，下标均从0开始
// a, b：如上
// n：表示a中元素个数
// 返回凸包元素个数
int convex(ip *a, ip *b, int n) {
    //升序排序
    sort(a, a + n);
    int m = 0;
    //求下凸壳
    for (int i = 0; i < n; ++i) {
```

```
        for (; m > 1 && ((b[m - 1] - b[m - 2]) ^ (a[i] - b[m - 2])) < 0; --m);
        b[m++] = a[i];
    }
    //求上凸壳
    for (int i = n - 2, t = m; i >= 0; --i) {
        for (; m > t && ((b[m - 1] - b[m - 2]) ^ (a[i] - b[m - 2])) < 0; --m);
        b[m++] = a[i];
    }
    return m - 1;
}
// ================ 代码实现结束 ================

ip a[N], b[N];

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        a[i].ri(i + 1);
    int m = convex(a, b, n), ans = m;
    for (int i = 0; i < m; ++i)
        ans = ((ll)ans * b[i].i) % (n + 1);
    printf("%d\n", ans);
    return 0;
}
```

图

描述

一个数列 a 称为合法的当且仅对于所有的位置 i, j（i < j ≤ n），都不存在一条从 aj 点连向 ai 的有向边。现在有很多个有向无环图，请你判断每个图是否只存在**唯一**的合法数列。

输入

输入的第一行包含一个正整数 T ，表示数据组数。

对于每组数据，第一行包含两个正整数 n, m，表示图的节点个数和边数。

接下来 m 行，每行包含两个正整数 x, y（x, y ≤ n），表示这个图有一条从 x 到 y 的有向边。

保证没有自环和重边。

输出

输出 T 行，若所给的图存在唯一的合法数列，输出 1，否则输出 0。

样例 1 输入

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

样例 1 输出

```
1
0
```

样例 1 解释

第一个图只有一个合法数列：1、2、3；

第二个图有两个合法数列：1、2、3 或者 1、3、2。

样例 2

请查看下发文件内的 sample2_input.txt 和 sample2_output.txt。

限制

对于 50%的数据，n, m ≤ 100；

对于 100%的数据，T ≤ 100, n, m ≤ 10000。

时间：4 sec

空间：512 MB

提示

[本题就是判断一个有向无环图是否存在唯一的拓扑序列。]

[回忆一下求拓扑序列是如何做的：每一次都取一个入度为 0 的点，将这个点取出来放进拓扑序列里，然后将这个点连向的所有点的入度减去 1。]

[可以发现，在“每一次都取一个入度为 0”这一步，若入度为 0 的点数多于 1 个，则显然拓扑序不唯一。]

[因此按照这个拓扑序算法做一遍就好。]

代码

```
#include <bits/stdc++.h>
using namespace std;
#include <bits/stdc++.h>
using namespace std;
// ================ 代码实现开始 ================
const int N = 10005;
// 为了减少复制开销，我们直接读入信息到全局变量中，并统计了每个点的入度到数组in中
// n, m：点数和边数
// in：in[i]表示点i的入度
// e：e[i][j]表示点i的第j条边指向的点
int n, m, in[N];
vector<int> e[N];
// 判断所给有向无环图是否存在唯一的合法数列
// 返回值：若存在返回1；否则返回0。
bool getAnswer() {
    //找到一个入度为0的点。有向无环图中至少存在一个入度为0的点
    //若存在多个入度为0的点，说明合法序列不唯一
    int x = 0;
    for (int i = 1; i <= n; ++i)
        if (in[i] == 0) {
            if (x != 0)//表示入度为0的点不唯一
                return 0;
            x = i;
        }
    //x表示的就是图中唯一的入度为0的点，然后去除关联的边
    for (int j = 1; j <= n; ++j) {
```

```
            int z = 0;
            for (int i = 0; i < (int)e[x].size(); ++i) {
                int y = e[x][i];
                --in[y];
                if (in[y] == 0) {
                    if (z != 0)
                        return 0;
                    z = y;
                }
            }
            x = z;
        }
        return 1;
}
// ================ 代码实现结束 ================


int main() {
    int T;
    for (scanf("%d", &T); T--; ) {
        scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; ++i) {
            in[i] = 0;
            e[i].clear();
        }
        for (int i = 0; i < m; ++i) {
            int x, y;
            scanf("%d%d", &x, &y);
            e[x].push_back(y);
            ++in[y];
        }
        printf("%d\n", getAnswer());
    }
    return 0;
}
```

最近点对

描述

给定 n 个二维平面上的点，求距离最近的一对点，输出他们的距离。

输入

第一行包含一个正整数 n。

接下来 n 行，每行包含两个整数 x,y，表示一个点的坐标。

输出

输出距离最近的一对点的距离，保留两位小数。

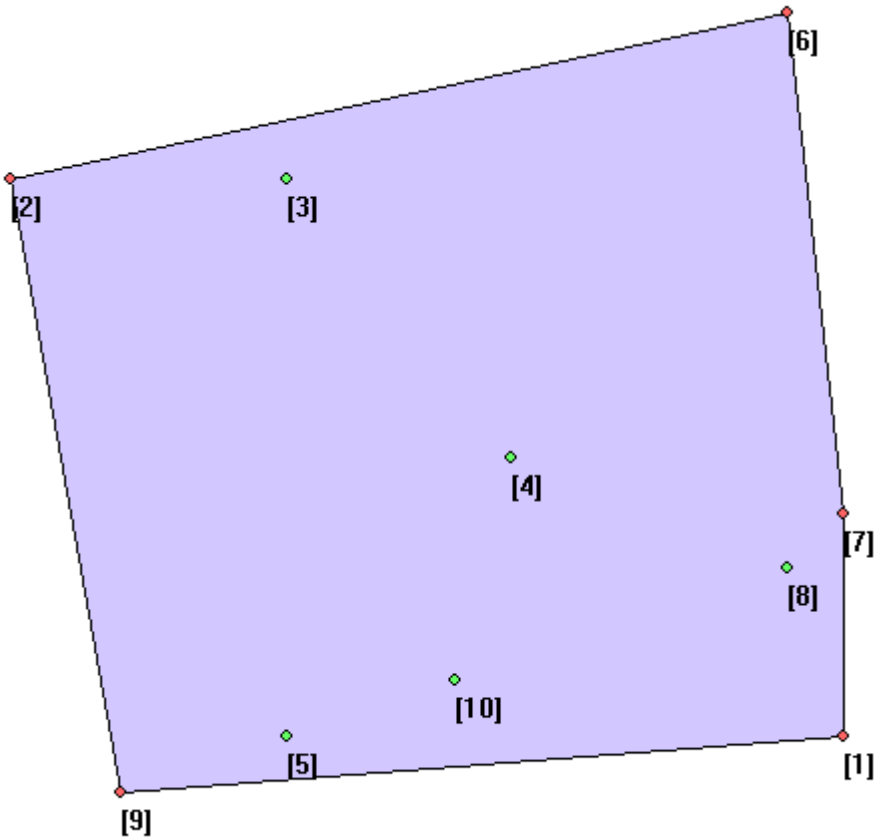
样例 1 输入

```
10
7 9
-8 -1
-3 -1
1 4
-3 9
6 -4
7 5
6 6
-6 10
0 8
```

样例 1 输出

```
1.41
```

样例 1 解释



距离最近的点为 7 和 8，距离为(7-6)2+(5-6)2=2≈1.41$\sqrt{(7-6)^2+(5-6)^2}=\sqrt{2}\approx 1.41$

样例 2 输入

点击下载

限制

对于 70%的数据，2 ≤ n ≤ 2000，每个点坐标的绝对值不超过 10^5；

对于 100%的数据，2 ≤ n ≤ 3×10^5，每个点坐标的绝对值不超过 10^9。

时间：10 sec

空间：512 MB

提示

[ ]

代码

```
#include <bits/stdc++.h>
using namespace std;

#include <bits/stdc++.h>
using namespace std;
// ================ 代码实现开始 ================
typedef double lf;
typedef long long ll;
const int N = 300005;
// 用于存储一个二维平面上的点
struct ip {
    int x, y;
    // 构造函数
    ip(int x = 0, int y = 0) : x(x), y(y) { }
    // 先比较x轴，再比较y轴
    bool operator < (const ip &a) const {
        return x == a.x ? y < a.y : x < a.x;
    }
} a[N], b[N];
// 计算x的平方
ll sqr(const ll &x) {
    return x * x;
}
// 计算点a和点b的距离
lf dis(const ip &a, const ip &b) {
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}
lf ans;
//分治求最近点对
//l,r：表示闭区间[l,r]
void solve(int l, int r) {
    //边界情况
    if (r - l <= 1) {
        if (a[l].y > a[r].y)
            swap(a[l], a[r]);
        if (l != r)
            ans = min(ans, dis(a[l], a[r]));
```

```
        return;
    }
    //分治计算两遍
    int mid = (l + r) >> 1;
    int md = a[mid].x;//中间值
    solve(l, mid);
    solve(mid + 1, r);
    //对y轴进行归并排序
    int cnt = 0;
    for (int i = l, j = mid + 1; i <= mid || j <= r;) {
        for (; i <= mid && md - a[i].x >= ans; ++i);
        for (; j <= r && a[j].x - md >= ans; ++j);
        if (i <= mid && (j > r || a[i].y < a[j].y))
            b[cnt++] = a[i++];
        else
            b[cnt++] = a[j++];
    }
    //现在b数组
    for (int i = 0; i < cnt; ++i)
        for (int j = i + 1; j < cnt && b[j].y - b[i].y < ans; ++j)
            ans = min(ans, dis(b[i], b[j]));
    cnt = 0;
    for (int i = l, j = mid + 1; i <= mid || j <= r;) {
        if (i <= mid && (j > r || a[i].y < a[j].y))
            b[cnt++] = a[i++];
        else
            b[cnt++] = a[j++];
    }
    memcpy(a + l, b, sizeof(ip) * cnt);
}
// 计算最近点对的距离
// n：n个点
// X, Y：分别表示x轴坐标和y轴坐标，下标从0开始
// 返回值：最近的距离
double getAnswer(int n, vector<int> X, vector<int> Y) {
    for (int i = 0; i < n; ++i)
        a[i + 1] = ip(X[i], Y[i]);
    ans = 1e100;
    sort(a + 1, a + 1 + n);
    solve(1, n);
    return ans;
}
// ================ 代码实现结束 ================
int main() {
```

```
    int n;
    scanf("%d", &n);
    vector<int> X, Y;
    for (int i = 1; i <= n; ++i) {
        int x, y;
        scanf("%d%d", &x, &y);
        X.push_back(x);
        Y.push_back(y);
    }
    printf("%.2f\n", getAnswer(n, X, Y));
    return 0;
}
```

纸牌

时间限制：1 sec

空间限制：512 MB

问题描述

小明有 2n 张纸牌，点数依次从 1 到 2n。小明要和你玩一个游戏，这个游戏中，每个人都会分到 n 张卡牌。游戏一共分为 n 轮，每轮你们都要出一张牌，**点数小者**获胜。

游戏开始了，你拿到了你的牌。你现在想知道，你最多（也就是运气最好的情况下）能够获胜几轮？

输入格式

第一行 1 个正整数 n。

第 2 行到第 n+1 行每行一个正整数 a[i]，表示你的第 i 张牌的点数。

输出格式

一行一个整数表示你最多能够获胜的轮数。

样例输入

```
2
1
4
```

样例输出

```
1
```

数据范围

对于 31.25% 的数据，保证 1<=n<=100

对于 100% 的数据，保证 1<=n<=50,000
保证数据的合法性，即你即不会拿到重复的牌，又不会拿到超出点数范围的牌。

代码

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100005;
bool exist[N];
int a[N], t;
int b[N], h;
int n;
int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &b[i]);
        exist[b[i]] = 1;
    }
    h = 1;
    t = 0;
    for (int i = 1; i <= 2 * n; ++i)
        if (!exist[i])
            a[++t] = i;
    sort(a + 1, a + n + 1);
    sort(b + 1, b + n + 1);
```

```
    int ans = 0;
    for (int i = 1; i <= n; ++i)
        if (a[i] > b[h])
            ++ans, ++h;
    printf("%d\n", ans);
    return 0;
}
```

青蛙

题目名称：小青蛙

时间限制：5 sec

空间限制：256 MB

问题描述

一个坐标轴上有 n 个荷叶，编号从 1 到 n。每片荷叶有一个坐标。

有一只可爱的小青蛙，它任选一片荷叶作为起点，并选择一个方向（左或右）然后开始跳。第一次跳跃时，他没有任何限制。从第二次跳跃开始，受到魔法的影响，他每次跳跃的距离都必须不小于前一次跳跃的距离，且跳跃方向必须与上一次跳跃保持一致。

每一片荷叶上都有一个数值。每次小青蛙跳到一片荷叶上时，他就会获得该荷叶对应的数值。特别地，他初始选择的荷叶的数值也是能得到的。

小青蛙可以在任意时刻选择停止跳跃。

可爱的小青蛙希望能获得尽可能大的数值总和。你能帮帮她吗？

输入格式

第一行个整数 n，意义见问题描述。

第 2 行到第 n+1 行，每行 2 个整数 x[i] 和 s[i]，描述一片荷叶，其中 x[i] 表示这片荷叶的坐标，s[i] 表示这片荷叶上的数值。

输出格式

一行一个整数，表示小青蛙能够获得的最大的数值总和。

样例输入

```
6
5 6
1 1
10 5
7 6
4 8
8 10
```

样例输出

```
25
```

数据范围

对于 30% 的测试点，保证 n<=8。

对于 50% 的测试点，保证 n<=120。

对于 70% 的测试点，保证 n<=600。

对于 100% 的测试点，保证 1<=n<=1000，0<=x[i],p[i]<=10^6。

代码

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1003;
pair<int, int> a[N];
int n;

int dp[N][N];

int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        int x, y;
```

```
        scanf("%d%d", &x, &y);
        a[i] = pair<int, int>(x, y);
    }
    int ans = 0;
    for (int round = 0; round < 2; ++round) {
        sort(a + 1, a + n + 1);
        for (int i = 1; i <= n; ++i) {
            dp[i][i] = a[i].second;
            for (int j = 1; j < i; ++j) {
                dp[i][j] = 0;
                for (int k = j; k && 2 * a[j].first <= a[i].first + a[k].first; --k)
                    dp[i][j] = max(dp[i][j], dp[j][k]);
                ans = max(ans, (dp[i][j] += a[i].second));
            }
        }
        for (int i = 1; i <= n; ++i)
            a[i].first = -a[i].first;
    }
    printf("%d\n", ans);
    return 0;
```

}

柿子合并

描述

又到了吃柿饼的季节。

小莉的果园共有 nn 棵柿子树，编号为 11 到 nn 。最开始，这些柿子树之间都没有道路相连。
小莉现在规划出了 mm 对中间可能修建双向道路的柿子树，用 mm 个三元组 (u,v,w)(u,v,w) 表示，表示在编号为 uu 和 编号为 vv 的柿子树之间修建道路需要花费 ww 元。
小莉决定在修完道路后，将能够直接或间接通过道路连接的柿子树划分为一个子集。并且，对于划分出的每一个子集，用这个子集中的所有式子树上长出的柿子做出一个柿饼。

小莉最终一共想要得到 kk 个柿饼，请你帮他计算最小的修路费用是多少。

输入

第 11 行有三个整数 nn， mm ， kk ，含义见题目描述。

接下来 mm 行，每行三个整数 uu ， vv ， ww ，描述每条可能修建的道路，含义如题所述。

输出

输出一行一个整数，表示小莉修路的最小花费。

如果小莉无论如何都不能做出 kk 个柿饼，请输出 −1-1 。

输入样例 1

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

输出样例 1

```
2
```

样例 1 解释

在 2，3 与 2，4 之间修建道路。

这样我们就可以将所有柿子制作成 22 个柿饼。

输入样例 2

点此下载。

限制

对于 30%的数据，$1\le n\le 100,1\le m\le 10001\le n\le 100,1\le m\le 1000$；
对于 100%的数据，
$1\le n\le 10000,1\le m\le 100000,1\le k\le 10,1\le u,v\le n,0\le w\le 100001\le n\le 10000,1\le m\le 100000,1\le k\le 10,1\le u,v\le n,0\le w\le 10000$。

提示

[稍微改一下 kruskal 算法就行辣]

代码

```
#include <iostream>
#include<vector>
#include<algorithm>
#pragma warning(disable:4996)
using namespace std;

// ================= 代码实现开始 =================


/* 请在这里定义你需要的全局变量 */
const int N = 10000 + 7;

//表示一条情报
struct Edge {
    //表示将编号为u的柿子与编号为v的柿子合并为一颗柿子的花费为w
    int u, v, w;
    Edge(){}
    Edge(int u,int v,int w):u(u),v(v),w(w){}
    bool operator < (const Edge& A) const {
        //按花费从小到大排序
        return w < A.w;
    }
};

//Father:每个节点的父亲节点
int Father[N];

//查找节点x所在集合的根
//x:节点x
//返回值:根
int find(int x) {
    int fx;//某节点的根节点(父节点)
    if (Father[x] > 0)//若x的秩大于0，说明x不是根节点
    {
        fx = find(Father[x]);//若x不是根节点，则递归查找其根节点
        Father[x] = fx;//*****此处至关重要，路径压缩，将某节点的父节点设置为根节点
********
```

```
        return fx;
    }
    else
        return x;//若x的秩小于等于0，说明x是根节点，返回x
}


void Union(int x, int y)//将x和y所在的两个集合合并
{
    int fx = find(x);
    int fy = find(y);
    fy = fy;//断点调试点
    if (fx == fy && fx != 0)//若fx和fy相等且其两个的秩不为0，则说明他们在同一个集合树中
        return;
    if (Father[fx] < Father[fy])
        Father[fy] = fx;//若x所在的集合树比较高，则将y所在的树合并到x所在的树中，即将y的根节点设置为fx
    else if (Father[fx] == Father[fy])//若两个树的高度相等，则任意将一个树合并到另一个树中(此处是将x的根节点设置为fy)
    {
        Father[fy] -= 1;//合并后树的高度加一，故秩减一
        Father[fx] = fy;
    }
    else//若y所在的集合树比较高，则将x所在的树合并到y所在的树中，即将x的根节点设置为fy
        Father[fx] = fy;
}

// 给定n个点m条边的有权无向图，求最小的k-生成森林的边权和
// k-生成森林：k-生成森林是原图的一个生成子图，并且使得其中存在k个森林
// n：如题意
// m：如题意
// k：如题意
// E：大小为m的数组，表示m条情报
// 返回值：若能构成k-生成森林，则返回最小k-生成森林的边权和，否则返回-1
int getAnswer(int n, int m, int k, vector<Edge> E)
{
    /* 请在这里设计你的算法 */
    //ans:最小边权和
    //edgeLft:k-生成森林中的边数
    int ans = 0, edgeLft = n - k;

    //初始化
    for (int i = 1; i <= n; ++i) {
        Father[i] = 0;
```

```
    }

    //按花费大小从小到大排序
    sort(E.begin(), E.end());

    //指向E第一个元素的迭代器
    vector<Edge>::iterator it = E.begin();
    //若E中还有边或还未构成k-生成森林则重复以下操作
    while (it != E.end() && edgeLft) {
        int setu = find(it->u); //u所在的集合
        int setv = find(it->v); //v所在的集合
        if (setu != setv) {
            Union(setu, setv); //若u,v不在同一个集合中，将u,v所在的集合合并
            ans += it->w;      //将边权和(总花费)增加
            edgeLft--;         //需要选取的边数减少1
        }
        it++;//迭代器后移
    }

    if (edgeLft)//若edgeLft>0，说明不存在k-生成森林，返回-1
        return -1;
    return ans;//若存在k-生成森林则返回边权和(总花费)
}

// ================= 代码实现结束 =================

int main()
{
    int n, m, k;
    vector<Edge> E;
    scanf("%d%d%d", &n, &m, &k);
    for (int i = 0; i < m; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        E.push_back(Edge(u, v, w));
    }
    cout << getAnswer(n, m, k, E) << endl;
    return 0;
```

问题描述

小粽家里有一块地，地上有 nn 个木桩。小粽家的地可以看作是一个平面，并且小粽知道每个木桩的坐标 (x_i,y_i)(xi,yi)。
小粽很喜欢四边形，现在她想从这些木桩中选出 44 个来围成一个四边形（这个四边形为简单多边形，即每条边不能和自己相交，但不一定要为凸四边形），并使得这个四边形的面积最大。请你帮小粽算出这个最大值是多少。

输入格式

第一行一个正整数 nn 表示木桩的大小。
接下来 nn 行，第 ii 行位两个实数 x_i,y_ixi,yi，描述了第 ii 个木桩的坐标。

输出格式

输出一行一个实数，表示围出的最大的四边形的面积。保留三位小数。

输入样例 1

```
5
0 0
1 0
1 1
0 1
0.5 0.5
```

输出样例 1

```
1.000
```

样例 2

点此下载。

数据范围及约定

20%20% 的数据满足 $n \leq 100$n ≤100；

60%60% 的数据满足 n ≤400n ≤400；
80%80% 的数据满足 n ≤1500n ≤1500；
100%100% 的数据满足 n ≤5000n ≤5000，所有坐标都在 int 范围内。

提示

[显然，答案是在凸包上的]

[可以考虑枚举一条对角线，再设法确定另外两点的位置，例如注意到点的位置具有单峰性。]

[更可考虑求对角线与凸包的切点，利用单调性均摊复杂度]

代码

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<stack>
#include<cmath>
#pragma warning(disable:4996)
#define SF scanf
#define PF printf
#define MAXN 10000
using namespace std;
struct node {
    double x, y;
    node() {}
    node(double xx, double yy) :x(xx), y(yy) {}
    node operator + (const node &a) const {
        return node(x + a.x, y + a.y);
    }
    node operator - (const node &a) const {
        return node(x - a.x, y - a.y);
    }
    node operator * (const double &t) const {
        return node(x*t, y*t);
    }
    double operator *(const node &a) const {
        return x * a.x + y * a.y;
    }
    double operator ^(const node &a) const {
        return x * a.y - y * a.x;
    }
```

```
    bool operator < (const node &a) const {
        return x < a.x || (x == a.x&&y < a.y);
    }
}p[MAXN], l1[MAXN];
stack<node> s1, s;
double ans;
int n, cnt1;
void solve(node a1[], int &cnt) {
    s.push(p[1]);
    s1.push(p[1]);
    s1.push(p[2]);
    for (int i = 3; i <= n; i++) {
        while (!s.empty() && ((p[i] - s.top()) ^ (s1.top() - s.top())) <= 0) {
            s.pop();
            s1.pop();
        }
        s.push(s1.top());
        s1.push(p[i]);
    }
    while (!s1.empty()) {
        a1[++cnt] = s1.top();
        s1.pop();
    }
    while (!s.empty())
        s.pop();
}
double sum(node a, node b, node c) {
    return fabs((b - a) ^ (c - a));
}
bool cmp(node a, node b) {
    return b < a;
}
int main() {
    //freopen("data.in","r",stdin);
    //freopen("data.out","w",stdout);
    SF("%d", &n);
    for (int i = 1; i <= n; i++)
        SF("%lf%lf", &p[i].x, &p[i].y);
    sort(p + 1, p + 1 + n);
    solve(l1, cnt1);
    sort(p + 1, p + 1 + n, cmp);
    cnt1--;
    solve(l1, cnt1);
    cnt1--;
```

```
    /*for(int i=1;i<=cnt1;i++)
        PF("{%.2f %.2f}\n",l1[i].x,l1[i].y);*/
        /*node t1=node(0,0);
        node t2=node(1,1);
        node t3=node(2,0);
        PF("{%.3f}\n",sum(t1,t2,t3));*/
    for (int i = 1; i <= cnt1; i++) {
        int st1 = i % cnt1 + 1;
        int j = (i + 1) % cnt1 + 1;
        int st2 = (j + 1) % cnt1 + 1;
        for (; j%cnt1 + 1 != i; j = j % cnt1 + 1) {
            while (st1%cnt1 + 1 != j && sum(l1[i], l1[st1], l1[j]) < sum(l1[i],
l1[st1%cnt1 + 1], l1[j]))
                st1 = st1 % cnt1 + 1;
            while (st2%cnt1 + 1 != i && sum(l1[i], l1[st2], l1[j]) < sum(l1[i],
l1[st2%cnt1 + 1], l1[j]))
                st2 = st2 % cnt1 + 1;

    //PF("[%d(%.0f,%.0f) %d(%.0f,%.0f)-%d(%.0f,%.0f) %d(%.0f,%.0f) %.2f %.2f]\n",i,l1[i
].x,l1[i].y,st1,l1[st1].x,l1[st1].y,st2,l1[st2].x,l1[st2].y,j,l1[j].x,l1[j].y,sum(l1[i]
,l1[st1],l1[j])/2.0,sum(l1[i],l1[st2],l1[j])/2.0);
            ans = max(ans, (sum(l1[i], l1[st1], l1[j]) + sum(l1[i], l1[st2], l1[j])) /
2.0);
        }
    }
    PF("%.3lf", ans);
    return 0;
```

}循环节

问题描述

小粽今天在玩一个字符串。

最初，小粽手上有很多很多个（你可以认为是无限多个）一模一样的字符串 aa，小粽选出若干个 aa 顺次拼接为一个新的字符串 bb。
由于小粽犯了粗心，她把最初的 aa 搞丢了，并且 bb 的末尾也丢失了一些字符，只剩下了一个 bb 的前缀 cc。
小粽很伤心，为了安慰她，请帮她计算可能的 aa 的最短长度是多少。

输入格式

第一行一个正整数 nn，表示 cc 的长度。
第二行一行一个字符串，描述字符串 cc。

输出格式

输出一行一个整数，表示 aa 的可能的最短长度。

输入样例 1

```
8
cabcabca
```

输出样例 1

```
3
```

样例 1 解释

最短的 aa 为 cab。

样例 2

点此下载。

数据规模及约定

对于 20%20% 的数据有 n≤100n≤100；
对于 50%50% 的数据有 n≤6000n≤6000；
对于 70%70% 的数据有 n≤2×10^5n≤2×10^5；
对于 100%100% 的数据有 1≤n≤10^61≤n≤10^6，并且 cc 中只有小写字母。

提示

[猜不到结论？手动画几个找找规律呗]

代码

```
#include <cstdio>
#include <cstring>
#pragma warning(disable:4996);
#define maxl 1000000

/* ========== 代码实现开始 ========== */

void getNext(char *s, int len, int next[])
{
    int i = 0, j = -1;
    next[0] = -1;
    while (i < len)
    {
        if (-1 == j || s[i] == s[j])
            next[++i] = ++j;
        else
            j = next[j];
    }
}

int next[maxl + 10];

// s, len: 输入字符串（题目中的c）及长度
// 返回值：题目中 a 串的最短长度
int solve(char *s, int len)
{
    getNext(s, len, next);
    if (next[len] == 0)
        return len;
    else
        return len - next[len];
}

/* ========== 代码实现结束 ========== */

char s[maxl + 10];

int main()
{
    int len;
    scanf("%d%s", &len, s);
```

```
    printf("%d\n", solve(s, len));
    return 0;
}
```

数星星

问题描述

小粽今晚在数星星。

小粽把整个天空看作一个平面，她测出了她看见的每个星星的坐标，第 i 颗星星的坐标为 (x_i,y_i)。
光数星星实在是太无聊了，小粽想知道，对于每颗星星，其左下方的星星的数量，即对于每个 i，小粽想要知道满足 $j\neq i$，且 $x_j\leq x_i,y_j\leq y_i$ 的 j 的数量。

输入格式

第一行一个正整数 n，表示星星的数量。
接下来 n 行，每行两个正整数 x_i,y_i，表示第 i 颗星星的坐标。

输出格式

输出共 n 行，每行输出一个正整数，表示第 i 颗星星左下方星星的数量。

输入样例 1

```
5
1 1
5 1
7 1
3 3
5 5
```

输出样例 1

```
0
```

```
1
2
1
3
```

样例 2

点此下载。

数据规模及约定

对于 30%30% 的数据有 $n \le 500$；
对于 60%60% 的数据有 $n \le 8000$；
对于 100%100% 的数据有 $n \le 3\times 10^5$，所有坐标都在 int 范围内，不存在两个不同的点有相同的坐标。

提示

[回想一下归并求逆序对咋做的？你就会做这个题辣]

[有兴趣的同学百度二维偏序~]

代码

```
#include <cstdio>
#include <algorithm>
#pragma warning(disable:4996);
#define maxn 500000

using namespace std;

struct Point {
    int x, y, id;

    void read(int id)
    {
        this->id = id;
        scanf("%d%d", &x, &y);
    }

    // 先比较x轴，再比较y轴
```

```
    bool operator < (const Point &a) const {
        return x == a.x ? y < a.y : x < a.x;
    }
};

// a: 原数组
// b: 归并辅助数组
Point a[maxn + 10], b[maxn + 10];

// ans: ans[i] 表示第 i 个点左下方的点的数目
int ans[maxn + 10];

/* ========== 代码实现开始 ========= */

void solve(int l, int r)
{
    if (l >= r)
        return;

    int m = l + r >> 1;

    solve(l, m);
    solve(m + 1, r);//分治，解决子问题

    //归并过程
    int t1 = l, t2 = m + 1, t = l;
    while (t1 <= m && t2 <= r){
        if (a[t1].y <= a[t2].y) {//归并
            b[t++] = a[t1++];
        }
        else {
            ans[a[t2].id] += t1 - l;//除了正常的归并外，还需统计左侧对右侧的贡献
            b[t++] = a[t2++];
        }
    }
    for (int i = t1; i <= m; i++) {
        b[t++] = a[t1++];
    }
    for (int i = t2; i <= r; i++) {
        ans[a[t2].id] += t1 - l;//除了正常的归并外，还需统计左侧对右侧的贡献
        b[t++] = a[t2++];
    }

    for (int i = l; i <= r; i++)
```

```
        a[i] = b[i];
}


// solve 被调用后，ans 数组应被计算好
void solve(int n)
{
    sort(a, a + n);
    for (int i = 0; i < n; i++)
        ans[i] = 0;

    solve(0, n - 1);
}

/* ========== 代码实现结束 ========= */

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        a[i].read(i);

    solve(n);

    for (int i = 0; i < n; i++)
        printf("%d\n", ans[i]);

    return 0;
```

}匹配

描述

给定两个长度为 5n5n 的序列，其中 [1,n][1,n] 之间的所有数都出现了**恰好 5 次**。
求它们的最长公共子序列长度。

输入

第一行一个整数 nn ，意义如题目描述。
第二行 5n5n 个整数，表示序列 A。
第三行 5n5n 个整数，表示序列 B。

输出

一行一个整数，表示序列 A 与序列 B 的最长公共子序列的长度。

输入样例 1

```
2
2 2 1 1 2 1 2 2 1 1
2 1 2 1 1 2 2 1 2 1
```

输出样例 1

```
8
```

样例 1 解释

一种最长的公共子序列为 22112221 。

样例 2

点此下载。

限制

对于 50%的数据，1≤n≤10001≤n≤1000；
对于 100%的数据，1≤n≤200001≤n≤20000。

提示

[由于每种数都只出现了 5 次，我们考虑把每种数在 A 中出现的位置作为第一维，在 B 中出现的位置作为第二维，就可以得到一个散点图。]

[那么我们现在就可以将问题转化为求平面上满足两维坐标都单调上升的最长点列的长度，然后我们就可以 DP 辣。]

代码

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 2e4 + 5;
int n, pos[MAXN][10], cnt[MAXN], a[MAXN * 5], dp[MAXN * 5];
int lowbit(int x) {
    return x & -x;
}
int query(int p) {
    int ret = 0;
    while (p) {
        ret = max(ret, dp[p]);
        p -= lowbit(p);
    }
    return ret;
}
void update(int p, int x) {
    while (p <= n * 5) {
        dp[p] = max(dp[p], x);
        p += lowbit(p);
    }
}
int main() {
    int x;
    scanf("%d", &n);
    for (int i = 1; i <= n * 5; ++i)
        scanf("%d", &a[i]);
    for (int i = 1; i <= n * 5; ++i) {
        scanf("%d", &x);
        pos[x][++cnt[x]] = i;
    }
    for (int i = 1; i <= n * 5; ++i)
        for (int j = 5; j; --j) // 倒序更新，如果正序的话，会导致前面更新的答案被用来更新后面的 dp 值。
            update(pos[a[i]][j], query(pos[a[i]][j] - 1) + 1); // 用树状数组维护区间最大值，dp[i]表示以i结尾的A数组和B数组的LCS
    printf("%d\n", query(n * 5));
}
```