栈

描述

实现一个栈，完成以下功能：

1. 入栈
2. 出栈
3. 询问栈中位置Y是谁

一开始栈为空。栈中的位置从1开始（即栈底位置为1）。

输入

第一行一个整数n，表示操作个数。

接下来n行，每行第一个数字表示操作（见描述）：

- 若为数字1，则接下来有一串字符串X，表示将X压入栈中。
- 若为数字2，表示弹出栈顶（保证栈非空），并输出出栈的这个人。
- 若为数字3，则接下来有一个整数Y，表示询问栈中位置Y是谁（保证位置Y合法），并输出名字。

输出

将所有操作2和操作3输出，一行一个。

输入样例

```
11
1 a
1 b
1 c
3 1
3 2
3 3
2
1 d
3 1
3 2
3 3
```

输出样例

```
a
b
c
c
a
b
d
```

限制

对于30%的数据，1 ≤ n ≤ 2000；

对于另30%的数据，没有操作3；

对于100%的数据，1 ≤ n ≤ 100000。

数据中出现的字符串只包含26个小写字母（无空格等分隔符），且长度不超过15。

字符串有可能重复。正如现实中可能有重名一样。

时间：2 sec

空间：256 MB

提示

[入栈和出栈都是操作着栈顶。]

[开一个大小为n的数组，记录栈顶的位置，入栈出栈就是将这栈顶加一减一，栈中某个位置Y在数组相应的下标就是Y。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/5593/5593c251bf5c613e9d073ce5a75c1f48a4ac7cfb.zip)】下载。

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

const int N = 100005;
string Stack[N];//array to query
int n,top;

//Refresh the three operations
void push(string name){
    Stack[++top] = name;
}
string pop(){
    return Stack[top--];
}
string query(int pos){
    return Stack[pos];
}

int main(){
    //Implement the stack using a array: concise the query operation
    scanf("%d",&n);
    char name[20];

    for(;n--;){//think: n-- or --n
        int OP;
        scanf("%d",&OP);
        if( OP == 1){
            scanf("%s",name);
            push(name);
        } else if(OP ==2){
            printf("%s\n",pop().c_str());
        }
        else{
            int pos;
            scanf("%d",&pos);
            printf("%s\n",query(pos).c_str());
        }
    }
}
```

队列

描述

实现一个队列，完成以下功能：

1. 入列
2. 出列
3. 询问队列中位置Y是谁

一开始队列为空。队列中的位置从1开始（即队头位置为1）。

输入

第一行一个整数n，表示操作个数。

接下来n行，每行第一个数字表示操作（见描述）：

- 若为数字1，则接下来有一串字符串X，表示将X加入队列。
- 若为数字2，表示出列（保证队列非空），并输出出列的这个人。
- 若为数字3，则接下来有一个整数Y，表示询问队列中位置Y是谁（保证位置Y合法），并输出名字。

输出

将所有操作2和操作3输出，一行一个。

输入样例

```
11
1 a
1 b
1 c
3 1
3 2
3 3
2
1 d
3 1
3 2
3 3
```

输出样例

```
a
b
c
a
b
c
d
```

限制

对于30%的数据，1 ≤ n ≤ 2000；

对于另30%的数据，没有操作3；

对于100%的数据，1 ≤ n ≤ 100000。

数据中出现的字符串只包含26个小写字母（无空格等分隔符），且长度不超过15。

字符串有可能重复。正如现实中可能有重名一样。

时间：2 sec

空间：256 MB

提示

[队头出列，队尾入列。]

[开一个大小为n的数组，记录队头和队尾的位置，入列出列就是将这两个位置改变一下，队列中某个位置Y在数组相应的下标为队头的位置+Y-1。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/9071/90711f65150a6fab45c28e78504179bccea71e2b.zip)】下载。

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

const int N = 100005;
int n,op,header=1,tailer=1;
string Queue[N];
//Implement a queue using the array
void enqueue(string name){
    Queue[tailer++] = name;
}

string dequeue(){
    return Queue[header++];
}

string query(int pos){
    return Queue[header+pos-1];//Notice:the [pos] is basic on the head
}

int main(){
    scanf("%d",&n);
    char name[20];//local or global ,which is better?
    int pos;//create here? or global? which is better?_ _answer :all is ok,global is more
    efficiency
    for(;n--;){
        scanf("%d",&op);
        if(op==1){
            scanf("%s",&name);
            enqueue(name);
        }else if(op==2){
            printf("%s\n",dequeue().c_str());
        }else{
            scanf("%d",&pos);
            printf("%s\n",query(pos).c_str());
        }
    }
    return 0;
}
```

二叉树

描述

给定一个1到n的排列，按顺序依次插入到一棵二叉排序树中，请你将这棵二叉树前序遍历和后序遍历输出。

前序遍历的定义 (https://baike.baidu.com/item/%E5%89%8D%E5%BA%8F%E9%81%8D%E5%8E%86)

后序遍历的定义 (https://baike.baidu.com/item/%E5%90%8E%E5%BA%8F%E9%81%8D%E5%8E%86)

输入

第一行一个整数n。

接下来一行表示为n个整数，代表1到n的一个排列。

输出

输出所建成的二叉树的前序遍历和后序遍历。

输入样例

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

输出样例

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

限制

对于50%的数据，1 ≤ n ≤ 100；

对于100%的数据，1 ≤ n ≤ 100000。

保证建成的树的高度不超过50。

时间：2 sec

空间：256 MB

提示

[二叉树的操作基本都是递归操作，只要想想如何在一个节点上判断是朝着左孩子走还是朝着右孩子走就行了。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/9b21/9b2103b89188f9b6b5c595f03cf15c0a021dd245.zip)】下载。

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

const int N = 100005;

struct node{
    int val,l,r;
} t[N];

//root:tree root;
//cnt:整个二叉树的大小
int root,cnt;//initial will give 0, as a global variable

//以x为根的树中插入v
//v:要插入的数字
//x: 当前节点的编号
//返回值: x
int insert(int v,int x){
    if(x == 0){
        //若当前节点不存在，则将x变成一个新节点
        x = ++cnt;
        t[x].val = v;
        t[x].l = 0;
        t[x].r = 0;
        return x;
    }
    //递归插入左右子树
    if(t[x].val > v)
        t[x].l = insert(v,t[x].l);
    else
        t[x].r = insert(v,t[x].r);
    return x;
}

//求以x为根的二叉树的前序遍历
//x:当前节点
//ans:存储结果的数组
void dlr(int x,vector<int> &ans){
    if(x){
        //加入x节点的val到ans中，递归求解左右子树
        ans.push_back(t[x].val);
        dlr(t[x].l,ans);
        dlr(t[x].r,ans);
    }
}

//求以x为根的二叉树的后序遍历
//x:当前节点
//ans:存储结果的数组
void lrd(int x,vector<int> &ans){
    if(x){
        //加入x节点的val到ans中，递归求解左右子树
        lrd(t[x].l,ans);
        lrd(t[x].r,ans);
        ans.push_back(t[x].val);//x is the serial number,the value is our need.
    }
}

// 给定一个1到n的排列，依次插入到二叉树中，返回前序遍历和后序遍历
// n: 元素的总个数
// sequence: 给定的排列，大小为n
// 返回值: 将要输出的元素依次加入到返回值中
vector<int> getAnswer(int n, vector<int> sequence) {
    root = cnt= 0;//初始化

    for (int i = 0;i<int(sequence.size());++i)
        root = insert(sequence[i],root);
    vector<int> ans;//返回值
    dlr(root,ans);//先加入前序遍历
    lrd(root,ans);//后加入后序遍历
```

```
    return ans;
}


int main() {
    int n;
    scanf("%d", &n);
    vector<int> sequence;
    for (int i = 0; i < n; ++i) {
        int x;
        scanf("%d", &x);
        sequence.push_back(x);
    }
    vector<int> ans = getAnswer(n, sequence);
    //output after select
    for (int i = 0; i < n; ++i)
        printf("%d%c", ans[i], " \n"[i == n - 1]);
    for (int i = 0; i < n; ++i)
        printf("%d%c", ans[n + i], " \n"[i == n - 1]);
    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≤5*10^5。

对于所有数据，保证 op∈{1,2}。

时间限制：2 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！]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/e666/e666a262851e0e64c5ba58e4c868db99c8733a7e.zip)】下载。

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

const int Mod = 10000003;
vector<ll> table[Mod];

int Hash(ll x){
    return x%Mod;
}
/*solution description:
    hashTable,decrease the length of array,use a list to solve the element conflict
    hash_function: get mod
*/

bool check(int op, ll x){
    int h = Hash(x);

    vector<ll>::iterator ptr = table[h].end();//the type of iterator is the element which he
    will visit
    //iterator:neglect the inner structure,visit the element one by one
    for(vector<ll>::iterator it = table[h].begin();it != table[h].end();++it){
        if( *(it) == x){//if find x,give the pointer to x
            ptr = it;
            break;
        }
    }
    if(op==1){
        if(ptr == table[h].end()){//ptr is not change,which means did not find x in the
        hashTable
            table[h].push_back(x);//put x into the table (element is vector too) at location
            h,
            return 1;
        }
        return 0;
    }else{
        if(ptr != table[h].end()){// change means find
            *ptr = table[h].back();//swap tail and the to delete element ,more efficiency
            table[h].pop_back();
            return 1;
        }
        return 0;
    }
}

int main() {
    int Q, op;
    ll x;
    scanf("%d", &Q);
    while (Q--) {
        scanf("%d%lld", &op, &x);//%i64d or %lld for long long data
        puts(check(op, x) ? "Succeeded" : "Failed");
    }
    return 0;
}
```

重编码

问题描述

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 2 进制串（即只包含 0 或 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≤i,j≤n（i≤j），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[i]，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 S1（不妨假设长度为 n）被称为字符串 S2 的前缀，当且仅当：S2 的长度不小于 n，且 S1 与 S2 前 n 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 n，表示单词种数。

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

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[1]=000，s[2]=001，s[3]=01，s[4]=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[1]=00，s[2]=01，s[3]=10，s[4]=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。

对于第 2 个测试点，保证 n=5。

对于第 3 个测试点，保证 n=16，且所有 w[i] 都相等。

对于第 4 个测试点，保证 n=1,000。

对于第 5 个测试点，保证所有 w[i] 都相等。

对于所有的 7 个测试点，保证 2≤n≤100,000，w[i]≤10^11。

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)。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/775f/775f99d3b33eada7dd323a316c9917370a4a6883.zip)】下载。

Source

改编自：【NOI2015】荷马史诗（http://uoj.ac/problem/130 (http://uoj.ac/problem/130)）

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

//declare a function to sort a stack
stack<int> sorting(stack<int> MyStack);

int main(){
    //create a stack named MyStack,to receive data input
    int n,x;
    scanf("%d",&n);
    stack<int> MyStack;
    for(int i = 0;i<n;++i){
        scanf("%d",&x);
        MyStack.push(x);
    }

    //New empty stack result :to get the ordered MyStack element
    stack<int> result;
    result = sorting(MyStack);

    //New vector to get the order stack element,for output
    vector<int> answer;
    while(!result.empty()){
        int temp = result.top();
        answer.push_back(temp);
        result.pop();//Delete after put into the out_vector
    }

    //Output the element orderly
    for(int i = answer.size()-1;i >=0;--i){
        printf("%d\n",answer[i]);
    }
    return 0;
}

//Stack_sort executive
stack<int> sorting(stack<int> MyStack){
    //new a stack to get the result
    stack<int> result;

    //The input extreme situation
    if(MyStack.empty())
        return result;

    //New a temp to contain MyStack.top
    int temp = MyStack.top();
    MyStack.pop();

    //Sorting : when MyStack is not void,or result.top(not null) bigger than(>) MyStack.top
    ,the cycle should continue
    while(!MyStack.empty() || ( !result.empty() && result.top()>temp)){
        //when result.top(not null,if null ,it can just continue) is smaller(<=) than temp
        ,push the temp into result ,and refresh temp
        if(result.empty() || result.top() <= temp){
            result.push(temp);
            temp = MyStack.top();
            MyStack.pop();
        }else{//Or backward: push result.top into the MyStack ,to find the right position of
        temp
            MyStack.push(result.top());
            result.pop();
        }
    }
    result.push(temp);//Push the last element of MyStack  into result ,on conditions that it
    is bigger than result.top
    return result;
}
```

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：2 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。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/2e74/2e7429da45c00297950eabba70c98282ede6c68b.zip)】下载。

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

int main(){
    int n,tac,dsa;//n range from 0 to 5,000;
    scanf("%d",&n);
    int tacScore[n+1];
    int dsaScore[n+1];
    int num[n+1];
    int score[n+1];

    memset(tacScore,-1,sizeof(tacScore));
    memset(dsaScore,-1,sizeof(dsaScore));
    memset(num,-1,sizeof(num));
    memset(score,-1,sizeof(score));


    //complicity: o(n) + o(n) + o(n^2) + o(n)
    for(int i = 1;i <= n; ++i){
        scanf("%d%d",&tac,&dsa);
        tacScore[i] = tac;
        dsaScore[i] = dsa;
        num[i] = i;//都从1开始，到n
    }

    //o(n)
    for(int j = 1;j<=n;++j){
        score[j] = tacScore[j]+ dsaScore[j];//total score
    }

    //o(n^2)
    int cnt = 0;
    for(int t = 1;t<=n;++t){
        for(int k = n;k >= t+1;--k){//bubble sort by reverse sequence
            if( score[k-1] < score[k] || (score[k-1]==score[k] && tacScore[k-1] < tacScore[k
            ])){
                int temp = score[k-1];
                score[k-1] = score[k];
                score[k] = temp;

                int temptac = tacScore[k-1];
                tacScore[k-1] = tacScore[k];
                tacScore[k] = temptac;

                int tempdsa = dsaScore[k-1];
                dsaScore[k-1] = dsaScore[k];
                dsaScore[k] = tempdsa;

                int tempnum= num[k-1];
                num[k-1] = num[k];
                num[k] = tempnum;
                cnt++;
            }
        }
    }

    //o(n)
    for(int m = 1;m<=n;++m){
        printf("%d %d %d %d\n",num[m],score[m],tacScore[m],dsaScore[m]);
    }
    printf("%d\n",cnt);

    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 (attachment/d07e/d07ec7fd5e8cf1fc3efb30bd14da0db4c3c13b2b.zip)

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,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]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/8cd4/8cd42618df19d952f7c59158c7f50cf37435630a.zip)】下载。

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

const int N = 300005;
int Father[N];//n variable in total;
int Rank[N];

int findRoot(int x){
    return Father[x] == x ? x:Father[x] = findRoot(Father[x]);
}
string getAnswer(int n,int m,vector<int> A,vector<int> B,vector<int> E){
    //initialize n union sets
    for(int i=1;i<=n;++i){
        Father[i] = i;
        Rank[i] = 0;
    }

    //judge the equation firstly:sort by E;
    int cnt = 0;//point to the not equation
    for(int j = 0;j < m;++j){
        if(E[j] == 1){
            swap(A[j],A[cnt]);
            swap(B[j],B[cnt]);
            swap(E[j],E[cnt]);
            ++cnt;
        }
    }

    //merge and judge ,give the result
    for(int k=0;k < m;++k){
        int setA = findRoot(A[k]);
        int setB = findRoot(B[k]);
        if(E[k] == 0){
            if(setA == setB ){//wrong1:same set ,not equal
                return "No";
            }
        }else{
            if(setA != setB ){//merge
                if(Rank[setA] > Rank[setB]){
                    swap(setA,setB);//设A为rank（树高小）小的树
                }
                Father[setA] = setB;//树高的setA的根节点，设为setB;
                Rank[setB]++;//这样也可以通过
                //if(Rank[setA] == Rank[setB]){//合并之后，自然秩相等，
                  //  Rank[setB]++;
                //}
            }
        }
    }
    return "Yes";
}

int main(){
    int T;
    scanf("%d",&T);
    while(T--){
        int n,m;
        scanf("%d%d",&n,&m);
        vector<int> A,B,E;//要在这里建新向量，刷新向量里的元素，否则向量将继承上一次的数据输入
        for(int i = 0;i<m;++i){
            int a,b,e;
            scanf("%d%d%d",&a,&b,&e);
            A.push_back(a);
            B.push_back(b);
            E.push_back(e);
        }
        printf("%s\n",getAnswer(n,m,A,B,E).c_str());
    }
    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：2 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？]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/466f/466f2c610f19e24f46662bafbb5a39949990a3da.zip)】下载。

```cpp
#include <bits/stdc++.h>
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 = 500005;
struct UnionSet{
    int f[N];

    void init(int n){
        for(int i=1;i<=n;++i)
            f[i]=i;
    }
    int find (int x){
        return f[x] == x ? x : f[x] = find(f[x]);
    }
    bool merge(int x,int y){
        int setx = find(x);
        int sety = find(y);
        if(setx != sety){
            f[setx] =sety;
            return true;
        }
        return false;
    }
}us;

// 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 */
    vector<int> ans;
    us.init(n);
    for(int i = m-1;i>=0;--i)
        if (us.merge(U[i],V[i]))
        ans.push_back(i+1);
    //reverse:将ans数组反过来，加入顺序是从大到小加入，而ans要求从小到大排列
    reverse(ans.begin(),ans.end());
    return ans;
}

// ================= 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.push_back(u);
        V.push_back(v);
    }
    vector<int> ans = getAnswer(n, m, U, V);
    printf("%d\n", int(ans.size()));
    for (int i = 0; i < int(ans.size()); ++i)
        printf("%d\n", ans[i]);
    return 0;
}
```

排序

描述

给出n个整数，将它们从小到大排序后输出。

输入

第一行为一个正整数n，第二行为n个整数。

输出

输出一行n个整数，表示排序后的n个整数。

样例1输入

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

样例1输出

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

样例2

请查看下发文件 (attachment/ca72/ca72592293f6fe6287d0c81859597323ab739710.zip)内的sample2_input.txt和sample2_output.txt。

限制

对于前30%的数据，n ≤ 100，给出的n个整数的绝对值不超过10；

对于前60%的数据，n ≤ 5000，给出的n个整数的绝对值不超过10^9；

对于另20%的数据，n ≤ 500000，给出的n个整数的绝对值不超过10^5；

对于100%的数据，n ≤ 500000，给出的n个整数的绝对值不超过10^9。

时间：2 sec

空间：256 MB

提示

若大家使用cin、cout进行输入输出，则需在main函数里的第一行加入ios::sync_with_stdio(false)，否则可能会超时。

推荐大家使用scanf和printf进行输入输出。

大家不妨使用各种排序算法进行测试。

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/313e/313eeb1b4c9367fa56778cce0b59513687baf99d.zip)】下载。

```java
import java.util.Scanner;

public class Main {

    private static Scanner sc;

    public static void sort(int[] array, int low, int high) {//
    传入low=0, high=array.length-1;
        int pivot, p_pos, i, t;// pivot->位索引;p_pos->轴值。
        if (low < high) {
            p_pos = low;
            pivot = array[p_pos];
            for (i = low + 1; i <= high; i++)
                if (array[i] < pivot) {
                    p_pos++;
                    t = array[p_pos];
                    array[p_pos] = array[i];
                    array[i] = t;
                }
            t = array[low];
            array[low] = array[p_pos];
            array[p_pos] = t;
            // 分而治之
            sort(array, low, p_pos - 1);// 排序左半部分
            sort(array, p_pos + 1, high);// 排序右半部分
        }
    }

    public static void main(String[] args) {
        sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] data=new int[n];
        for (int i = 0;i<n;++i) {
            data[i] = sc.nextInt();
        }

        sort(data, 0, data.length- 1);
        for(int i = 0;i<n;++i)
            System.out.print(data[i]+" ");
    }
}
```

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+3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 (attachment/04aa/04aabace4f50257938798afa1192fd723ec546ba.zip)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：2 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。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/6a2e/6a2e17f2999183bad71e4e983811fb3a07d579b7.zip)】下载。

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

bool check(long long d,int n,int m,vector<int> &a){
    long long sum = 0;
    int cnt = 1;
    for (int i =0 ;i<n;++i){
        if(a[i]>d)
            return false;
        sum += a[i];
        if(sum>d){
            sum = a[i];
            cnt = cnt+1;
        }
    }
    return cnt <=m;
}
long long getAnswer(int n,int m, vector<int> a) {
    long long L = 1,R = 0;
    for(int i = 0;i<n;++i)
        R += a[i];
    while(L<=R){
        long long mid = (L+R)>>1;
        if(check(mid, n, m, a)){
            R = mid -1;
        }
        else
            L = mid + 1;
    }
    return R+1;
}


int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    vector<int> a;
    a.resize(n);
    for (int i = 0; i < n; ++i)
        scanf("%d", &a[i]);
    printf("%lld\n", getAnswer(n, m, a));
    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。

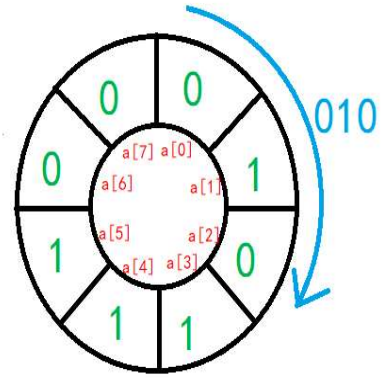
邓老师还有一个长度为 2^n 的数组 a（下标从 0 开始），其中的每个元素都是 0 或 1。于是邓老师就可以选择大转盘上的一个位置，将 a[0] 填入其中，然后按顺时针顺序依次将 a[1],a[2],…,a[2^n-1] 填入。

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？]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/0ea2/0ea28032dda673b14739c01fa33dbd4b6bb45db2.zip)】下载。

```cpp
#include<bits/stdc++.h>
using namespace std;
/*运用的知识点：
        1.欧拉回路

*/
//allOne:全1的二进制数，用于二进制的与运算， allone= 2^(n-1)-1，把最高位去掉
//vis:vis[i][u]表示从u出发，值为i的边
// ans :答案

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

//计算2^x
//x为指数
//return: 2^x;

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

//求欧拉回路的写法
//u: 当前所在节点
void dfs(int u){
    for (int i=0;i<2;++i){
        if(!vis[i][u]){
            int v= ((u<<1) | i)&allOne;//将u左移一位，然后将最低位置设为i,最高位去掉
            vis[i][u] = 1;
            dfs(v);
            ans.push_back('0'+i);//获得字符的0或1；递归之后，才能加
        }
    }
}


// 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) {
    //初始化
    allOne = npow(n-1)-1;
    ans = "";
    for(int i=0;i<2;++i)
        vis[i].resize(npow(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 0 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 (attachment/2976/297656de591a9fe6b8a22f1451315e6ec07fc5ca.zip)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？]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/ba5a/ba5abf438601927d8956ca69057fd985b8fd4058.zip)】下载。

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

const int N = 505*2,M = N*N;
struct E {
    int next,to;
}e[M];

//ihead:邻接表的表头
//cnt: 邻接表大小
//mc: 表示每个点所匹配的另一个点
//vis:Y元素是否被访问过
int cnt,ihead[N],mc[N];
bool vis[N];

//邻接表连边：表示连一条X到Y的有向边
//x: 起点
//y: 终点
void add(int x,int y){
    ++cnt;
    e[cnt].next = ihead[x];
    e[cnt].to = y;
    ihead[x] = cnt;
}

//匈牙利算法中的增广路：一直寻找增广路，直到找不到
//x: x集合上的点，从当前点出发找增广路
//返回值：若找到增广路则返回true，否则返回false
bool dfs(int x){
    //访问邻接表
    for (int i = ihead[x];i!=0;i = e[i].next){
        int y = e[i].to;
        if(!vis[y]){
            vis[y] = true;//起终点没有匹配，是增广路
            if (mc[y] == 0 || dfs(mc[y])){

                //如果y没有匹配点的，说明找到了一条增广路，或者说递归找到y的匹配点，得到了一条增广路
                mc[x] = y;
                mc[y] = x;
                return true;
            }
        }
    }
    return false;
}
// 求解棋盘上最多能放多少个"车"
// n: 棋盘的大小为n×n的
// board: 所给棋盘，对于某个位置上的数：若值为1表示可以放"车"；若值为0表示不能放"车"
// 返回值：能放"车"的最大个数
int getAnswer(int n, vector<vector<int> > board) {
    //将行看做n个点，将列看成另外n个点，标号分别是1到n和n+1到2n

    //初始化
    cnt = 0;
    for(int i = 0;i<=n*2;++i){
        ihead[i]=0;
        mc[i]=0;
    }

    //连边
    for(int i=1;i<=n;++i){
        for(int j = 1;j<=n;++j)
            if(board[i-1][j-1] == 1)
                add(i,j+n);//单向边，数组下标从0开始，i从1开始
    }

    int ans = 0;
    for(int i=1;i<=n;++i){
        if(!mc[i] ){
            //如果x集中的第i个点没有匹配到Y集合上的点，则从这个点出发寻找增广路
```

```
            memset(vis,0,sizeof(bool)*(n*2+1));
            if (dfs(i))//如果找到，答案直接+1;
                ++ans;//找到一条匹配，则之前的局面不用考虑
        }
    }
    return ans;
}


int main() {
    int n;
    scanf("%d", &n);
    vector<vector<int> > e;
    for (int i = 0; i < n; ++i) {
        vector<int> t;
        for (int j = 0; j < n; ++j) {
            int x;
            scanf("%d", &x);
            t.push_back(x);
        }
        e.push_back(t);
    }
    printf("%d\n", getAnswer(n, e));
    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 (attachment/0c86/0c864fe88e9e6594dd688d0ed73cad3be9b4759e.zip)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：4 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）]

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

const int N = 300005;
/*
n:n个互异的数，组成的序列
d:序列的限定长度
max_value:存放solve函数中的前缀最大值
min_value:存放solve函数的前缀最小值
*/

int n,d,max_value[N],min_value[N];
vector<int> a;

/*
分治计算区间[left,right]中有多少个连续子序列满足限定要求
left:区间的左边界
right:区间的右边界
返回值：满足条件的连续子序列的个数
*/
long long solve(int left,int right){
    if(left == right)
        return 0;//extreme condition

    int mid = (left+right)>>1;//序列的中点
    long long ans = solve(left,mid) +solve(mid+1,right);//分治两半

    //计算区间[mid+1,right]的min_value和max_value
    for(int i = mid+1;i<=right;++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]);
    }

    //倒序枚举子序列，左半段区间
    //mn后缀最小，mx后缀最小
    //
    int mn=0,mx=0,pos=right;
    for(int i = mid;i >= left && pos > mid;--i){
        mn = (i == mid ) ? a[i]:min(mn,a[i]);
        mx = (i == mid ) ? a[i]:max(mx,a[i]);
        for(; pos > mid && (max(mx,max_value[pos])-min(mn,min_value[pos]))> d;--pos);//pos
        随着i的递减变小
        ans += pos - mid;
    }
    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) {
    //初始化
    ::n = n;
    ::d = d;
    ::a = a;
    return solve(0,n-1);
}



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

```
    printf("%lld\n", getAnswer(n, d, a));
    return 0;
}
```

最小交换

时间限制： 1 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。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/f7d0/f7d04e3017338180d7d332e9f8bab85d5aa45d63.zip)】下载。

```cpp
#include <bits/stdc++.h>
using namespace std;
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<set>
#include<ctime>
#include<algorithm>
#include<queue>
#include<cmath>
#include<map>
#define oo 100000007
#define ll long long
#define pi acos(-1.0)
#define MAXN 500005

int n,a[MAXN],temp[MAXN];
ll ans;
void merge(int s1,int e1,int s2,int e2)
{
	int x,i,j,num=0;
	i=s1,j=s2;
	for (x=s1;x<=e2;x++)
	{
		if (i>e1) temp[x]=a[j++];
		  else
		if (j>e2) temp[x]=a[i++];
		  else
		if (a[i]>a[j])
		{
			temp[x]=a[j++];
			ans+=e1-s1+1-num; //从后面段插进来的数插在了多少个前面段数的前面...
		}else temp[x]=a[i++],num++;
	}
	for (i=s1;i<=e2;i++) a[i]=temp[i];
	return;
}
void merge_sort(int l,int r)
{
	if (l==r) return;
	int mid=(l+r)/2;
	merge_sort(l,mid);
	merge_sort(mid+1,r);
	merge(l,mid,mid+1,r);
	return;
}

int main()
{
    scanf("%d",&n);
    for (int i=1;i<=n;i++)  scanf("%d",&a[i]);
    ans=0;
    merge_sort(1,n);
    printf("%lld\n",ans);
    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！]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/c9dd/c9dd219c86beff2673d45889df8a9064f47eb8cd.zip)】下载。

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

vector<int> getAnswer(int n, vector<int> a, int Q, vector<int> query) {
    vector<int> ans;
    ans.clear();//初始化一个存放询问结果的向量
    sort(a.begin(),a.end());//对a进行排序

    for(int i = 0;i < Q;++i){
        int key = query[i];//key是本次询问的x
        int l = 0,r = a.size()-1,mid;

        while(l<=r){

            mid = (l+r)>>1;

            if(a[mid]<key)
                l = mid+1;
            else
                r = mid-1;
        }
        int pos = l;//答案的小标
        if(a[pos]<key)
            ans.push_back(-1);
        else
            ans.push_back(a[pos]);
    }
    return ans;

}


int main() {
    int n, Q, tmp;
    vector<int> a, query;
    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： 1 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）（attachment/2144/21447a0a2374eeb572839e7062dc0e0a21faa642.zip）

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。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/053c/053c6bb6ca19f8ab00f5407e079ff685bdb164d7.zip)】下载。

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

const int N = 100005;
typedef pair<int,int> pii;
vector<pii> graph[N];
priority_queue<pii,vector<pii>,greater<pii> >pq;
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、重点
// 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) {
    //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，连图中各边:连接成为 双向or无向图
    for(int i=  0;i<m;++i){
        graph[U[i]].push_back(make_pair(V[i],W[i]));
        graph[V[i]].push_back(make_pair(U[i],W[i]));
    }

    //设置起点的最短路为零，将起点加入优先级队列中
    mind[s] = 0;
    pq.push(make_pair(mind[s],s));

    //执行dijkstra算法
    while(!pq.empty()){
        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
            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(flag[v] || mind[v] < mind[u]+w)
                    continue;
                mind[v] = mind[u]+w;//进行答案更新
                pq.push(make_pair(mind[v],v));//将v伴随其最新的的最短路长度，加入优先级队列
            }
        }
    }
    return mind[t];
}


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;
}
```

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状态转移方程啦！]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/dd5f/dd5fb770e7af4b41e1e674b5a48304ed48c51d68.zip)】下载。

```cpp
#include <bits/stdc++.h>
#include<cstdio>
#include<algorithm>
using namespace std;
int r,a[1002][1002],F[1002][1002];

main()
{
    scanf("%d",&r);
    for(int i=1;i<=r;i++)
        for(int j=1;j<=i;j++)
        {
            scanf("%d",&a[i][j]);
            F[i][j]=a[i][j];
        }
    for(int i=r-1;i>0;i--)
        for(int j=1;j<=i;j++)
            F[i][j]+=max(F[i+1][j],F[i+1][j+1]);
    printf("%d",F[1][1]);
}
```

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 (attachment/8262/82628d7878ea65d780ebb99a6a41a66dd2b8d194.zip)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：2 sec

空间：512 MB

提示

[经典的01背包和完全背包问题。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/4a60/4a6066b59e76b039593b62f76c5084fc69575a6b.zip)】下载。

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


const int N = 5005;
//f:动态规划用得数组，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) {
    for(int i = 0;i<n;++i){
        if(t[i]==0)
            for(int j = V;j>=v[i];--j)//01背包，倒序枚举
                f[j] = max(f[j],f[j-v[i]]+w[i]);
        else//完全背包，顺序枚举
            for(int j = v[i];j<=V;++j)//01背包，倒序枚举
                f[j] = max(f[j],f[j-v[i]]+w[i]);
    }
    return f[V];
}


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 (attachment/d56c/d56c0905d17332cfb453967b00526aa62ae2b7e7.zip)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：2 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。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/b4a8/b4a80e7bdfd3cc943d1d3ff1c626f257640c9917.zip)】下载。

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

//d:前缀背包，d[][]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[][]表示物品i到物品n放进容量j的背包中的最大价值

const int N = 5005;
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) {
    //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];
        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，抽掉x，然后用前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 (attachment/88aa/88aa0797e099c6d0e7c3c8323e5c08d97ad8a297.zip)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：2 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。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/3bdb/3bdb9bda1761deac2a23886222b4cfacac82acea.zip)】下载。

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

const int N = 21,mo = 23333;
//f :记录已经计算过的答案，减少重复计算
int f[N][N][N][N][N][6];

//动态规划（记忆策略搜索），求当车辆数目为a+b+c+d+e时涂油漆的方案数
//a,b,c,d,e: 分别表示够涂1,2,3,4,5,辆车的油漆种类数;
//last: 若last==2，则表示前一辆车涂的油漆是从b中取出来的，。。。
//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，表示只有一种空方案
        return 1;
    if(f[a][b][c][d][e][last] != -1)
        return f[a][b][c][d][e][last];
    long long ret = 0;
    if(a)//若last==2，表示上一辆车从b里取出来的放到了a中，所以a中可以选择的油漆种类数要少一个
        ret += dp(a-1,b,c,d,e,1)*(a-(last==2));
    if(b)//减一表示，相邻车不能同色
        ret += dp(a+1,b-1,c,d,e,2)*(b-(last==3));
    if(c)
        ret += dp(a,b+1,c-1,d,e,3)*(c-(last==4));
    if(d)
        ret += dp(a,b,c+1,d-1,e,4)*(d-(last==5));
    if(e)
        ret += dp(a,b,c,d+1,e-1,5)*e;
    return f[a][b][c][d][e][last] =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: m种油漆
// a: 长度为m的数组，第i种油漆够涂ai辆车
// 返回值：方案数
int getAnswer(int m, vector<int> a) {
    memset(f,-1,sizeof f);
    for(int i =0;i<m;++i)
        b[a[i]]++;
    return dp(b[1],b[2],b[3],b[4],b[5],0);
}


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 (attachment/bcb5/bcb5e302e839ad69f19ca497d879d7d212af710c.zip)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

[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。]

请大家别打表。

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/14d7/14d798b5d9cabd0f7ee5d916814454fe4286c295.zip)】下载。

```cpp
#include <bits/stdc++.h>
using namespace std;
int ans,allOne;
void test(int row, int ld, int rd)
{
    int pos, p;
    if ( row != allOne)
    {
        pos = allOne & (~(row | ld | rd ));
        while ( pos )
        {
            p = pos & (~pos + 1);
            pos = pos - p;
            test(row | p, (ld | p) << 1, (rd | p) >> 1);
        }
    }
    else
        ++ans;
}


//
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) {
    ans = 0;
    allOne = ( 1 << n )-1;//生成对应棋盘n的全1的二进制数
    test(0,0,0);
    return ans;
}


int main() {
    int n;
    cin >> n;
    cout << getAnswer(n) << endl;
    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？]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/e4d3/e4d3861eeef06c0bd129ff68bbec149d1d5e9730.zip)】下载。

```cpp
#include <iostream>
#include <stdio.h>
#include <memory.h>
using namespace std;

#define LEN 100005

int a[LEN], b[LEN];
int loc[LEN], n;

void calLoc()
{
    int i;
    for(i = 1; i <= n; i++)
        loc[b[i]] = i;
}

int LIS()
{
    int i, k, l, r, mid;

    a[1] = b[1], k = 1;
    for(i = 2; i <= n; i++)
    {
        if(a[k] < b[i]) a[++k] = b[i];
        else {
            l = 1; r = k;
            while(l <= r)
            {
                mid = ( l + r ) / 2;
                if(a[mid] < b[i])
                    l = mid + 1;
                else
                    r = mid - 1;

            }
            a[l] = b[i];
        }
    }
    return k;
}

int main()
{
    int i;
    scanf("%d", &n);
        for(i = 1; i <= n; i++)
            scanf("%d", &a[i]);
        for(i = 1; i <= n; i++)
            scanf("%d", &b[i]);
        calLoc();
        for(i = 1; i <= n; i++)
            b[i] = loc[a[i]];
                printf("%d\n", LIS());
    return 0;
}
```

倒水问题

时间限制： 1 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。

提示

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/6d5d/6d5ddf8f2737b5374e71d240ace6e09647797218.zip)】下载。

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

typedef pair<int,int> pii;
#define fi first
#define se second

const int N = 2003;
int mind[N][N];
pii q[N*N];
int qh,qt;

pii to(pii p,int k,int n,int m){
    if(k == 0)//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.fi,m);
    else if(k == 4)//将杯子二向杯子一倒水
        return pii(min(p.fi+p.se,n),max(p.fi+p.se-n,0));
    else if(k == 5)//将杯子一向杯子二倒水
        return pii(max(p.fi+p.se-m,0),min(p.fi+p.se,m));//
    else
        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) {
    memset(mind,-1,sizeof(mind));
    qh = qt = 0;
    q[++qt] = pii(0,0);
    mind[0][0] = 0;

    //BFS
    while(qh < qt){
        pii u = q[++qh];
        if(mind[u.fi][u.se] == t)
            break;
        for(int k = 0;k < 6;++k){
            pii v = to(u,k,n,m);
            if(mind[v.fi][v.se] != -1)//判断目标状态是否未曾到达过
                continue;
            q[++qt] = v;
            mind[v.fi][v.se] = mind[u.fi][u.se] + 1;
        }
    }

    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(i+j-d));
    return ans;
}



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：1 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。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/f530/f530ca8e3ee1d2bec810ffa2ca655e182721e69f.zip)】下载。

```cpp
#include<cstdlib>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<iostream>
#include<climits>
#include<cctype>
#include<string>
#define maxn 3100
#define min(a,b) ((a)>(b)?(b):(a))
using namespace std;
int f[maxn][maxn][2];
int a[maxn],n,x;
int main(){
    scanf("%d%d",&n,&x);
    for(int i=1;i<=n;i++)
        scanf("%d",a+i);
    sort(a+1,a+n+1);
    for(int i=1;i<=n;i++)f[i][i][0]=f[i][i][1]=abs(a[i]-x)*n;
    for(int len=2;len<=n;len++)
        for(int i=1;i<=n-len+1;i++){
            int j=i+len-1;
            f[i][j][0]=f[i][j][1]=INT_MAX;
            f[i][j][0]=min(f[i][j][0],f[i+1][j][0]+(n-(j-i))*(a[i+1]-a[i]));
            f[i][j][0]=min(f[i][j][0],f[i+1][j][1]+(n-(j-i))*(a[j]-a[i]));
            f[i][j][1]=min(f[i][j][1],f[i][j-1][0]+(n-(j-i))*(a[j]-a[i]));
            f[i][j][1]=min(f[i][j][1],f[i][j-1][1]+(n-(j-i))*(a[j]-a[j-1]));
        }
    printf("%d\n",min(f[1][n][0],f[1][n][1]));
}
```

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 (attachment/8785/87857d0f6284f999593b8b05066e4055683f9e8c.zip)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：3 sec

空间：512 MB

提示

[对于长度为L的数列S, S中最大的元素为K，我们设他的$hash$值$H(S) = S_1C^0 + S_1C^1 + ... + S_LC^{L-1}$, 其中$C$为任意大于$K$的常数]

[对于不同的字符串A, B，$H(A) \neq H(B)$]

[我们先来看看一维的情况，给定两个字符串A, B，我们怎么判断A在B中出现？显然我们可以用$hash$来判断。但是$hash$值太大了怎么办？取模呀！找一个比较好的质数p，对于字符串A, B，若$A = B$则显然$H(A) \bmod p = H(B) \bmod p$；若$A \neq B$，$H(A) \bmod p$有一定概率会和$H(B) \bmod p$相同。怎么办呢？我们再找第二个质数$p$，再来验证$H(A) \bmod q$和$H(B) \bmod q$！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d的矩阵元素的$hash$值，得到一个矩阵。]

[然后我们再用新矩阵，再做一次$hash$，就是向上长度为c的矩阵元素的$hash$值，得到新矩阵X。]

[接着我们将第二个字符串的$hash$值求出来，就是先每一行求一个$hash$值，再将这c个$hash$值再$hash$一次变成一个数字，然后我们就去X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$O(n^2)$]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

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

    //用a数组计算横向的hash值，存储h1[0]
    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;
}

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<pii> 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 (attachment/7cb8/7cb8543e04d03c4b78d0da57002f3388ae2bfa34.zip)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。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/12f5/12f571a4629bb4fe2aa7b5247c6c1cfc43a5cf13.zip)】下载。

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

const int N = 500005;

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) {
    int n = str.size();

    //字符串中间加入#，下方将视为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（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;
}



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
```

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？]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/8d9a/8d9a1971c4c4c8ae7502de168cbd6fca64f967e7.zip)】下载。

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

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) {
    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;//所有的合数中，因子都是质数的数是邓数
            if(!isPrime[j/i]){
                isDeng[j]=0;
            }
        }
    }
    return ans;
}


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 (attachment/7e00/7e009f48ee5fdf7730645cdbd25a5ceefc48284e.zip)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$f(i)$为前i中本质不同的的子序列个数,令$pre(i)$表示字符s_i之前出现的位置,则]

[

$$f(i) = \begin{cases} f(i-1) + f(i-1) + 1 & pre(i) = 0 \\ f(i-1) + f(i-1) - f(pre(i)-1) & pre(i) \neq 0 \end{cases}$$

]

[答案等于$f(n)$.]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/c95f/c95fc8e52669959e3f180ab4522e72b609807896.zip)】下载。

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


const int N = 500005, mo = 23333;
//f:表示前i个字符形成的本质不同的子序列个数
//p:表示字符s[i]最后出现的位置
//last: 表示字符i最后出现的位置
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
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(int n) {
    //计算p数组
    for(int i = 1;i <= n;++i){
        int a = s[i] - 'a';
        p[i] = last[a];
        last[a] =i;
    }

    //动态规划
    for(int i = 1;i<=n;++i){
        //看题面的提示，注意取模
        if(p[i] == 0)
            f[i]=(2*f[i-1]+1)%mo;
        else{
            f[i]=(2*f[i-1]-f[p[i]-1] +mo )% mo;//如出现负值，处理
        }
    }
    return f[n];
}



int main() {
    scanf("%s", s + 1);
    int n = strlen(s + 1);
    printf("%d\n", getAnswer(n));
    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 (attachment/ee94/ee94a337846b78a74301a8870808af5dd5aa8522.zip)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：2 sec

空间：512 MB

提示

[trie树基本题。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/7712/77128d90267316ecf04e2ce26264a1bd8c708f3c.zip)】下载。

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

const int M = 505;
struct trieNode {
    trieNode() : prefixLatterWords(0) {
        for (size_t i = 0; i < 26; ++i) {
            children[i] = NULL;
        }
    }

    ~trieNode() {
        for (size_t i = 0; i < 26; ++i) {
            if (children[i]) {
                delete children[i];
                children[i] = NULL;
            }
        }
    }

    int prefixLatterWords; //后续单词个数
    trieNode *children[26];
};

class trie {
public:
    trie() : root(new trieNode) {}

    size_t Index(char c) {
        return static_cast<size_t>(c % 26);
    }

    void insert(const string& word);

    int countPrefix(const string& prefix);

public:
    trieNode *root;
};

void trie::insert(const string& word) {
    trieNode *cur = root;
    for (size_t i = 0; i < word.size(); ++i) {
        size_t idx = Index(word[i]);
        if (!cur->children[idx]) {
            cur->children[idx] = new trieNode;
        }
        cur = cur->children[idx];
        ++cur->prefixLatterWords;
    }
}

int trie::countPrefix(const string& prefix) {
    trieNode *cur = root;
    for (size_t i = 0; i < prefix.size(); ++i) {
        size_t idx = Index(prefix[i]);
        if (!cur->children[idx]) {
            return 0;
        }
        cur = cur->children[idx];
    }
    return cur->prefixLatterWords;
}

char s[M];
int main()
{
    trie t;
    int n, m;
```

```
    scanf("%d%d", &n, &m);
    string word;
    for (size_t i = 0; i < n; ++i) {
        scanf("%s", s);
        t.insert(s);
    }

    string prefix;
    for (size_t i = 0; i < m; ++i) {
        cin >> prefix;
        cout << t.countPrefix(prefix) << endl;
    }
    return 0;
}
```

最大间隙

时间限制： 7 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！）]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

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


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) {
    //初始化
    const int m = 1 << 26;//桶的个数=2^26次方的一个数
    memset(l,-1,sizeof(int)*m);//将l中的所有位置赋值为-1
    memset(r,-1,sizeof(int)*m);

    const int _k = max(k-26,0);//辅助后续用位运算代替除法，保证非负
    for(int i=0;i<n;++i){
        u32 bl = a[i] >> _k;//这个式子等价于a[i]除以2的_k次幂，求出a[i]所在的桶
        //更新对应桶的l,r
        if(l[bl] == -1)
            l[bl] = r[bl] = a[i];//置为当前值
        else if (a[i]<l[bl])
            l[bl] = a[i];
        else if(a[i]>r[bl])
            r[bl] = a[i];
    }
    //统计答案
    u32 last = a[0];//表示在之前出现的最大的数，
    u32 ans = 0;
    for (int i=0;i<m;++i){
        if(l[i] != -1){//等于-1时，桶中没有元素
            if(last > l[i])
                last = l[i];
            if(l[i] - last > ans)
                ans = l[i] - last;//两个空格(4)中需要填的东西是完全一致的
            last = r[i];
        }
    }
    return ans;
}



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： 7.5 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）]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/662c/662c10efdd680a166d106e185d6c83a2b135bd6b.zip)】下载。

Source

[改编自：「WC2017」挑战（[[https://loj.ac/problem/2286]]）]

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


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;

//sum：在基数排序中记录各值出现的次数
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) {
    //针对_a的后16位进行基数排序
    memset(sum,0,sizeof(sum));
    for(int i=0;i<n;++i)
        ++sum[a[i] & b];
    //对sum数组进行操作
    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];

    //再重复一遍上面的基数排序，只不过这次对_a的前16位排序，并存入a中
    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];

}
```

```cpp
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。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/3f22/3f22d1121edd99e32da8a56bf1ea0d3f071f55b9.zip)】下载。

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

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);//初始化

    int j = Next[0] = -1;//j为匹配失败时跳转到的位置
    for (int i = 1; i<m;++i){
        while(j>=0 && B[i]!= B[j+1])//如果下一位无法匹配，则利用next数组跳转
            j = Next[j];
        if(B[i]== B[j+1])
            ++j;//当前位置匹配，++j
        Next[i] = j;//记录当前位置的next数组
    }
    j = -1;//j为当前模式串匹配到的位置

    vector<int> ans;//返回答案
    for(int i =0; i<n;++i){
        while(j>=0 && A[i] != B[j+1])//下一位是否匹配，或退无可退，需要跳转
            j = Next[j];
        if(A[i] == B[j+1])//当前位置匹配，或者
            ++j;
        if(j == m-1)
            ans.push_back(i-j);//若整个模式串匹配，则找到一个答案
    }
    return ans;
}


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)

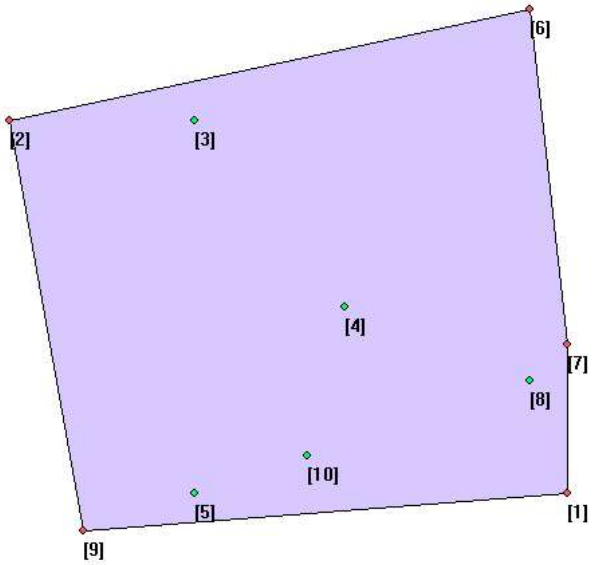
样例1输入

```
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 (attachment/98c2/98c2e705208ae44deb25d670daa13f94981afd5d.zip)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

来源

CG课程PA1 (https://dsa.cs.tsinghua.edu.cn/oj/problem.shtml?id=1645)（需要先加入CG课程，否则没权限看到题目）

提示

为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/9f46/9f46d5155dc79651a19292d52a6de643a6182d6f.zip)】下载。

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

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) {

    sort(a,a+n);//sort的作用，是找起点？
    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;
}



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 (attachment/febb/febbe7df3847f336691c59564754ad7b968e431a.zip)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：2 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。]

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/03c4/03c41ef8fb275abd9888ef5ce16446a60480fadc.zip)】下载。

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


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];

/* 请在这里定义你需要的全局变量 */

// 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() {
    //检查节点的入度，保证只有一个入度为零
    int x = 0;
    for(int i = 1;i <= n; ++i){
        if( in[i] == 0){//某个节点的入度为零
            if(x != 0 )//超过1个入度为零的节点，说明不存在
                return 0;
            x = i;
        }
    }

    //入度为零的点，删去后，仍然只有一个节点的入度为零
    //x表示的是图中唯一的入度为零的点
    for(int _ = 1;_<= n;++_){
        int z = 0;
        for(int i=0;i<(int)e[x].size();++i){//节点x的出度
            int y = e[x][i];//点x的第i条边，指向的节点
            if(in[y] == 1){//某个节点的入度为零
                if(z != 0 ){//超过1个入度为零的节点，说明不存在
                    return 0;}
                z = y;
            }
        }
        x = z;
    }
    return 1;
}


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。

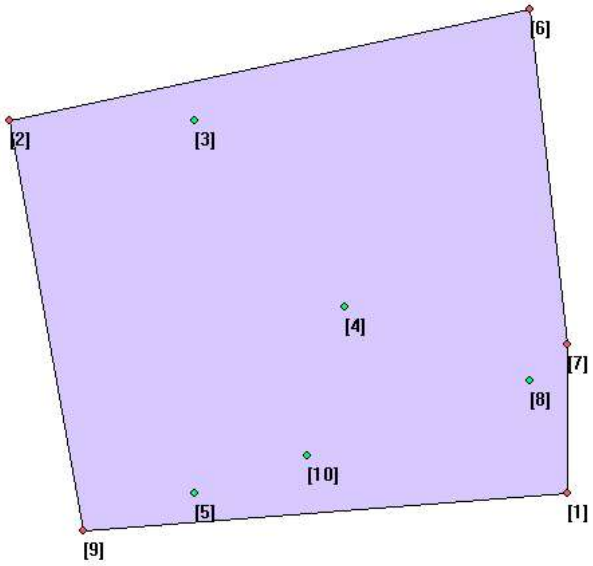
样例1输入

```
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$\sqrt{(7-6)^2+(5-6)^2}=\sqrt{2}\approx 1.41$

样例2输入

点击下载 (https://dsa.cs.tsinghua.edu.cn/oj/attachment/9390/9390aef38836b16b6c0dea64279a090883abe204.zip)

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：2 sec

空间：512 MB

提示

[分治求最近点对。当然也可以用kdtree，虽然应该会超时。]

图片来源CG课程PA1 (https://dsa.cs.tsinghua.edu.cn/oj/problem.shtml?id=1645)（需要先加入CG课程）

另外，为了帮助大家完成题目，我们提供了只包含了输入输出功能的程序模板，也提供了含有算法的大部分实现细节的程序。

你可以根据自己的实际情况，在这些程序的基础上进行作答，或不参考这些程序，这将与你的得分无关。

这些程序可以从【这里 (attachment/85a0/85a0a06fa7e0c9b1cf22cf68647ef0ec9130068c.zip)】下载。

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

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;

void solve(int l,int r){
    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;
    }

    //分治计算两边
    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，并且去掉，与中间点mid的距离，比答案大的点
    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++];
    }

    //b数组的点，按y轴升序，根据结论，一个点周围不会超过8个点
    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;
    //对y轴进行归并排序，不去掉任何点
    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+1,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);//对数组a中的元素进行排序, a+1是第一个元素的地址引用
    solve(1,n);
    return ans;
}



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。

另外，本题不提供程序模板和代码填空。

大家经过了一期算法训练营的训练，现在，来尝试一下自己独立写一道题吧！

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



int main() {
    int n;
    int x,cnt=0;
    scanf("%d", &n);
    int N = 2*n+1;
    int a[N],b[N];
    memset(a,0,sizeof(a));
    memset(b,0,sizeof(b));


    for (int i = 1; i <= n; ++i) {
        scanf("%d", &x);
        a[x] = x;
    }

    for(int j=1;j <= 2*n;j++)
        if(a[j]==0){
            b[j]=j;
        }

    int index1 = 1,index2 =1;
    for(;index1<= 2*n ,index2 <= 2*n;){
        for(;a[index1] == 0 && index1<2*n;++index1);
        for(;b[index2] == 0 && index2<2*n;++index2);
        if(a[index1] < b[index2]){
            ++cnt;
            ++index1;
            ++index2;
        }else{
            ++index2;
        }

    }

    printf("%d",cnt);

    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。

提示

本题时间限制较大，可以考虑一些效率一般的算法哦！

另外，本题不提供程序模板和代码填空。

大家经过了一期算法训练营的训练，现在，来尝试一下自己独立写一道题吧！

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

const int N = 1005;
pair<int,int> a[N];
int n;
int dp[N][N];


int main(){
    scanf("%d",&n);
    //接收数轴上有荷叶的坐标点
    for(int i=1;i<=n;++i){
        int x,s;
        scanf("%d%d",&x,&s);
        a[i] = pair<int,int>(x,s);
    }
    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));
            }
        }
        //将荷叶关于数轴对称调整一下，使排序算法继续适用
        for(int i = 1;i<=n;++i)
            a[i].first = -a[i].first;
    }

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