# ACM模板

--Dean Wang

目录

[ACM模板 1](#_Toc476993895)

[1设置 4](#_Toc476993896)

[1头文件 4](#_Toc476993897)

[2 STL 6](#_Toc476993898)

[1 map 6](#_Toc476993899)

[2 set 13](#_Toc476993900)

[3 string 14](#_Toc476993901)

[4 list 20](#_Toc476993902)

[5 unique 和 unique\_copy 21](#_Toc476993903)

[6 vector 23](#_Toc476993904)

[7 next\_permutation and prev\_permutation 26](#_Toc476993905)

[8 stack 27](#_Toc476993906)

[9 queue 28](#_Toc476993907)

[3 基本算法 28](#_Toc476993908)

[1 二分 28](#_Toc476993909)

[2 三分 29](#_Toc476993910)

[4 数据结构 29](#_Toc476993911)

[1 树桩数组 29](#_Toc476993912)

[2 RMQ 36](#_Toc476993913)

[3 线段树 39](#_Toc476993914)

[4 树链刨分（没验证过） 42](#_Toc476993915)

[5 离散化 45](#_Toc476993916)

[6 并查集 46](#_Toc476993917)

[5 字符串 47](#_Toc476993918)

[1 KMP 47](#_Toc476993919)

[2 字典树 49](#_Toc476993920)

[3 AC自动机 49](#_Toc476993921)

[4 字符串的最小表示法（没验证过） 54](#_Toc476993922)

[5 二叉树 54](#_Toc476993923)

[6 动态规划 56](#_Toc476993924)

[1 最长上升子序列(LIS) 56](#_Toc476993925)

[2 01背包 58](#_Toc476993926)

[3 数位dp（没验证过） 63](#_Toc476993927)

[7 数论 64](#_Toc476993928)

[1 判断素数 64](#_Toc476993929)

[2 素数打表 65](#_Toc476993930)

[3 分解质因数 65](#_Toc476993931)

[4 快速幂和快速乘 66](#_Toc476993932)

[5 逆元 68](#_Toc476993933)

[6 数模计算公式 69](#_Toc476993934)

[7 米勒罗宾素数测试 69](#_Toc476993935)

[8 Lucas定理 70](#_Toc476993936)

[9 中国剩余定理 72](#_Toc476993937)

[10 指数循环结 73](#_Toc476993938)

[11欧拉函数 77](#_Toc476993939)

[12 欧拉常数 79](#_Toc476993940)

[13 fft 80](#_Toc476993941)

[14 雷德算法和倒位序 84](#_Toc476993942)

[15 原根及快速求原根 85](#_Toc476993943)

[16 本福特定律 90](#_Toc476993944)

[8 图论 90](#_Toc476993945)

[1 拓扑排序 90](#_Toc476993946)

[2 最小生成树 92](#_Toc476993947)

[3 dijk 94](#_Toc476993948)

[4 SPFA 96](#_Toc476993949)

[5网络流 98](#_Toc476993950)

[9 数学 104](#_Toc476993951)

[1 高斯消元 104](#_Toc476993952)

[2 组合数学 113](#_Toc476993953)

[3 卡特兰数 114](#_Toc476993954)

[4 归并排序 115](#_Toc476993955)

[5 高精度 117](#_Toc476993956)

[6 矩阵 122](#_Toc476993957)

[10 博弈 127](#_Toc476993958)

[1 巴士博弈 127](#_Toc476993959)

[2 尼姆博弈 127](#_Toc476993960)

[3 威佐夫博弈 127](#_Toc476993961)

[4 sg图 127](#_Toc476993962)

[11 计算几何 128](#_Toc476993963)

[1 三角形 128](#_Toc476993964)

[2 四边形 128](#_Toc476993965)

[3 最近点对 129](#_Toc476993966)

[4 凸包 131](#_Toc476993967)

[5 公式 133](#_Toc476993968)

[6 计算几何模板 136](#_Toc476993969)

## 1设置

### 1头文件

#include<cstdio>

#include<iostream>

#include<vector>

#include<queue>

#include<cstring>

#include<cmath>

#include<map>

#include <string>

#include<algorithm>

#include<set>

#include<stack>

#define debug puts("xxxxxxx");

#define pi (acos(-1.0))

#define eps (1e-5)

#define inf 0x3f3f3f3f

#define mem(x) memset(x,0,sizeof(x))

#define sfl(x) scanf("%I64d",&x)

#define sfl2(x,y) scanf("%I64d%I64d",&x,&y)

#define sfl3(x,y,z) scanf("%I64d%I64d%I64d",&x,&y,&z)

#define sf(x) scanf("%d",&x)

#define sf2(x,y) scanf("%d%d",&x,&y)

#define sf3(x,y,z) scanf("%d%d%d",&x,&y,&z)

typedef long long ll;

const int mx = 1005;

void fre()

{

    freopen("1003.in","r", stdin);

    freopen("1003my.out","w", stdout);

}

const long long mod = 1e9+7;

using namespace std;

#include <bits/stdc++.h>

#define debug puts("xxxxxxx");

#define pi (acos(-1.0))

#define eps (1e-8)

#define inf 0x3f3f3f3f

#define mx 200005

#define mem(x) memset(x,0,sizeof(x))

typedef long long ll;

using namespace std;

**freopen**("input.txt","r", stdin);**freopen**("output.txt","w", stdout);

void fre()

{

    freopen("a.in","r", stdin);

    freopen("a.out","w", stdout);

}

#pragma comment(linker, "/STACK:102400000,102400000")

template <class T>

inline bool rd(T &ret) {

char c; int sgn;

if (c = getchar(), c == EOF) return 0;

while (c != '-' && (c<'0' || c>'9')) c = getchar();

sgn = (c == '-') ? -1 : 1;

ret = (c == '-') ? 0 : (c - '0');

while (c = getchar(), c >= '0'&&c <= '9') ret = ret \* 10 + (c - '0');

ret \*= sgn;

return 1;

}

template <class T>

inline void pt(T x) {

if (x < 0) {

putchar('-');

x = -x;

}

if (x > 9) pt(x / 10);

putchar(x % 10 + '0');

}

## 2 STL

### 1 map

STL中map用法详解

一．Map概述  
Map是STL的一个关联容器，它提供一对一（其中第一个可以称为关键字，每个关键字只能在map中出现一次，第二个可能称为该关键字的值）的数据处理能力，由于这个特性，它完成有可能在我们处理一对一数据的时候，在编程上提供快速通道。这里说下map内部数据的组织，map内部自建一颗红黑树(一种非严格意义上的平衡二叉树)，这颗树具有对数据自动排序的功能，所以在map内部所有的数据都是有序的，后边我们会见识到有序的好处。  
下面举例说明什么是一对一的数据映射。比如一个班级中，每个学生的学号跟他的姓名就存在着一一映射的关系，这个模型用map可能轻易描述，很明显学号用int描述，姓名用字符串描述(本篇文章中不用char \*来描述字符串，而是采用STL中string来描述),下面给出map描述代码：  
Map<int, string> mapStudent;  
map的构造函数  
map共提供了6个构造函数，这块涉及到内存分配器这些东西，略过不表，在下面我们将接触到一些map的构造方法，这里要说下的就是，我们通常用如下方法构造一个map：  
Map<int, string> mapStudent;  
数据的插入  
在构造map容器后，我们就可以往里面插入数据了。这里讲三种插入数据的方法：  
第一种：用insert函数插入pair数据，下面举例说明(以下代码虽然是随手写的，应该可以在VC和GCC下编译通过，大家可以运行下看什么效果，在VC下请加入这条语句，屏蔽4786警告  ＃pragma warning (disable:4786) )  
#include <map>  
#include <string>  
#include <iostream>  
Using namespace std;  
Int main()  
{  
 Map<int, string> mapStudent;  
 mapStudent.insert(pair<int, string>(1, “student\_one”));  
 mapStudent.insert(pair<int, string>(2, “student\_two”));  
 mapStudent.insert(pair<int, string>(3, “student\_three”));  
 map<int, string>::iterator  iter;  
 for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)  
{  
 Cout<<iter->first<<”   ”<<iter->second<<end;  
}  
}  
第二种：用insert函数插入value\_type数据，下面举例说明  
#include <map>  
#include <string>  
#include <iostream>  
Using namespace std;  
Int main()  
{  
 Map<int, string> mapStudent;  
 mapStudent.insert(map<int, string>::value\_type (1, “student\_one”));  
 mapStudent.insert(map<int, string>::value\_type (2, “student\_two”));  
 mapStudent.insert(map<int, string>::value\_type (3, “student\_three”));  
 map<int, string>::iterator  iter;  
 for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)  
{  
 Cout<<iter->first<<”   ”<<iter->second<<end;  
}  
}  
第三种：用数组方式插入数据，下面举例说明  
#include <map>  
#include <string>  
#include <iostream>  
Using namespace std;  
Int main()  
{  
 Map<int, string> mapStudent;  
 mapStudent[1] =  “student\_one”;  
 mapStudent[2] =  “student\_two”;  
 mapStudent[3] =  “student\_three”;  
 map<int, string>::iterator  iter;  
 for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)  
{  
 Cout<<iter->first<<”   ”<<iter->second<<end;  
}  
}  
以上三种用法，虽然都可以实现数据的插入，但是它们是有区别的，当然了第一种和第二种在效果上是完成一样的，用insert函数插入数据，在数据的插入上涉及到集合的唯一性这个概念，即当map中有这个关键字时，insert操作是插入数据不了的，但是用数组方式就不同了，它可以覆盖以前该关键字对应的值，用程序说明  
mapStudent.insert(map<int, string>::value\_type (1, “student\_one”));  
mapStudent.insert(map<int, string>::value\_type (1, “student\_two”));  
上面这两条语句执行后，map中1这个关键字对应的值是“student\_one”，第二条语句并没有生效，那么这就涉及到我们怎么知道insert语句是否插入成功的问题了，可以用pair来获得是否插入成功，程序如下  
Pair<map<int, string>::iterator, bool> Insert\_Pair;  
Insert\_Pair = mapStudent.insert(map<int, string>::value\_type (1, “student\_one”));  
我们通过pair的第二个变量来知道是否插入成功，它的第一个变量返回的是一个map的迭代器，如果插入成功的话Insert\_Pair.second应该是true的，否则为false。  
下面给出完成代码，演示插入成功与否问题  
#include <map>  
#include <string>  
#include <iostream>  
Using namespace std;  
Int main()  
{  
 Map<int, string> mapStudent;  
Pair<map<int, string>::iterator, bool> Insert\_Pair;  
 Insert\_Pair ＝ mapStudent.insert(pair<int, string>(1, “student\_one”));  
 If(Insert\_Pair.second == true)  
 {  
  Cout<<”Insert Successfully”<<endl;  
 }  
 Else  
 {  
  Cout<<”Insert Failure”<<endl;  
 }  
 Insert\_Pair ＝ mapStudent.insert(pair<int, string>(1, “student\_two”));  
 If(Insert\_Pair.second == true)  
 {  
  Cout<<”Insert Successfully”<<endl;  
 }  
 Else  
 {  
  Cout<<”Insert Failure”<<endl;  
 }  
 map<int, string>::iterator  iter;  
 for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)  
{  
 Cout<<iter->first<<”   ”<<iter->second<<end;  
}  
}  
大家可以用如下程序，看下用数组插入在数据覆盖上的效果  
#include <map>  
#include <string>  
#include <iostream>  
Using namespace std;  
Int main()  
{  
 Map<int, string> mapStudent;  
 mapStudent[1] =  “student\_one”;  
 mapStudent[1] =  “student\_two”;  
 mapStudent[2] =  “student\_three”;  
 map<int, string>::iterator  iter;  
 for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)  
{  
 Cout<<iter->first<<”   ”<<iter->second<<end;  
}  
}  
map的大小  
在往map里面插入了数据，我们怎么知道当前已经插入了多少数据呢，可以用size函数，用法如下：  
Int nSize = mapStudent.size();  
数据的遍历  
这里也提供三种方法，对map进行遍历  
第一种：应用前向迭代器，上面举例程序中到处都是了，略过不表  
第二种：应用反相迭代器，下面举例说明，要体会效果，请自个动手运行程序  
#include <map>  
#include <string>  
#include <iostream>  
Using namespace std;  
Int main()  
{  
 Map<int, string> mapStudent;  
 mapStudent.insert(pair<int, string>(1, “student\_one”));  
 mapStudent.insert(pair<int, string>(2, “student\_two”));  
 mapStudent.insert(pair<int, string>(3, “student\_three”));  
 map<int, string>::reverse\_iterator  iter;  
 for(iter = mapStudent.rbegin(); iter != mapStudent.rend(); iter++)  
{  
 Cout<<iter->first<<”   ”<<iter->second<<end;  
}  
}  
第三种：用数组方式，程序说明如下  
#include <map>  
#include <string>  
#include <iostream>  
Using namespace std;  
Int main()  
{  
 Map<int, string> mapStudent;  
 mapStudent.insert(pair<int, string>(1, “student\_one”));  
 mapStudent.insert(pair<int, string>(2, “student\_two”));  
 mapStudent.insert(pair<int, string>(3, “student\_three”));  
 int nSize = mapStudent.size()  
 for(int nIndex = 0; nIndex < nSize; nIndex++)  
{  
 Cout<<mapStudent[nIndex]<<end;  
}  
}  
数据的查找（包括判定这个关键字是否在map中出现）  
在这里我们将体会，map在数据插入时保证有序的好处。  
要判定一个数据（关键字）是否在map中出现的方法比较多，这里标题虽然是数据的查找，在这里将穿插着大量的map基本用法。  
这里给出三种数据查找方法  
第一种：用count函数来判定关键字是否出现，其缺点是无法定位数据出现位置,由于map的特性，一对一的映射关系，就决定了count函数的返回值只有两个，要么是0，要么是1，出现的情况，当然是返回1了  
第二种：用find函数来定位数据出现位置，它返回的一个迭代器，当数据出现时，它返回数据所在位置的迭代器，如果map中没有要查找的数据，它返回的迭代器等于end函数返回的迭代器，程序说明  
#include <map>  
#include <string>  
#include <iostream>  
Using namespace std;  
Int main()  
{  
 Map<int, string> mapStudent;  
 mapStudent.insert(pair<int, string>(1, “student\_one”));  
 mapStudent.insert(pair<int, string>(2, “student\_two”));  
 mapStudent.insert(pair<int, string>(3, “student\_three”));  
 map<int, string>::iterator iter;  
 iter = mapStudent.find(1);  
if(iter != mapStudent.end())  
{  
 Cout<<”Find, the value is ”<<iter->second<<endl;  
}  
Else  
{  
 Cout<<”Do not Find”<<endl;  
}  
}  
第三种：这个方法用来判定数据是否出现，是显得笨了点，但是，我打算在这里讲解  
Lower\_bound函数用法，这个函数用来返回要查找关键字的下界(是一个迭代器)  
Upper\_bound函数用法，这个函数用来返回要查找关键字的上界(是一个迭代器)  
例如：map中已经插入了1，2，3，4的话，如果lower\_bound(2)的话，返回的2，而upper-bound（2）的话，返回的就是3  
Equal\_range函数返回一个pair，pair里面第一个变量是Lower\_bound返回的迭代器，pair里面第二个迭代器是Upper\_bound返回的迭代器，如果这两个迭代器相等的话，则说明map中不出现这个关键字，程序说明  
#include <map>  
#include <string>  
#include <iostream>  
Using namespace std;  
Int main()  
{  
 Map<int, string> mapStudent;  
 mapStudent[1] =  “student\_one”;  
 mapStudent[3] =  “student\_three”;  
 mapStudent[5] =  “student\_five”;  
 map<int, string>::iterator  iter;  
iter = mapStudent.lower\_bound(2);  
{  
 //返回的是下界3的迭代器  
 Cout<<iter->second<<endl;  
}  
iter = mapStudent.lower\_bound(3);  
{  
 //返回的是下界3的迭代器  
 Cout<<iter->second<<endl;  
}

iter = mapStudent.upper\_bound(2);  
{  
 //返回的是上界3的迭代器  
 Cout<<iter->second<<endl;  
}  
iter = mapStudent.upper\_bound(3);  
{  
 //返回的是上界5的迭代器  
 Cout<<iter->second<<endl;  
}

Pair<map<int, string>::iterator, map<int, string>::iterator> mapPair;  
mapPair = mapStudent.equal\_range(2);  
if(mapPair.first == mapPair.second)  
 {  
 cout<<”Do not Find”<<endl;  
}  
Else  
{  
Cout<<”Find”<<endl;  
}  
mapPair = mapStudent.equal\_range(3);  
if(mapPair.first == mapPair.second)  
 {  
 cout<<”Do not Find”<<endl;  
}  
Else  
{  
Cout<<”Find”<<endl;  
}  
}  
数据的清空与判空  
清空map中的数据可以用clear()函数，判定map中是否有数据可以用empty()函数，它返回true则说明是空map  
数据的删除  
这里要用到erase函数，它有三个重载了的函数，下面在例子中详细说明它们的用法  
#include <map>  
#include <string>  
#include <iostream>  
Using namespace std;  
Int main()  
{  
 Map<int, string> mapStudent;  
 mapStudent.insert(pair<int, string>(1, “student\_one”));  
 mapStudent.insert(pair<int, string>(2, “student\_two”));  
 mapStudent.insert(pair<int, string>(3, “student\_three”));

//如果你要演示输出效果，请选择以下的一种，你看到的效果会比较好  
 //如果要删除1,用迭代器删除  
 map<int, string>::iterator iter;  
 iter = mapStudent.find(1);  
 mapStudent.erase(iter);

 //如果要删除1，用关键字删除  
 Int n = mapStudent.erase(1);//如果删除了会返回1，否则返回0

 //用迭代器，成片的删除  
 //一下代码把整个map清空  
 mapStudent.earse(mapStudent.begin(), mapStudent.end());  
 //成片删除要注意的是，也是STL的特性，删除区间是一个前闭后开的集合

 //自个加上遍历代码，打印输出吧  
}  
其他一些函数用法  
这里有swap,key\_comp,value\_comp,get\_allocator等函数，感觉到这些函数在编程用的不是很多，略过不表，有兴趣的话可以自个研究  
排序  
这里要讲的是一点比较高深的用法了,排序问题，STL中默认是采用小于号来排序的，以上代码在排序上是不存在任何问题的，因为上面的关键字是int型，它本身支持小于号运算，在一些特殊情况，比如关键字是一个结构体，涉及到排序就会出现问题，因为它没有小于号操作，insert等函数在编译的时候过不去，下面给出两个方法解决这个问题  
第一种：小于号重载，程序举例  
#include <map>  
#include <string>  
Using namespace std;  
Typedef struct tagStudentInfo  
{  
 Int      nID;  
 String   strName;  
}StudentInfo, \*PStudentInfo;  //学生信息

Int main()  
{  
 //用学生信息映射分数  
 Map<StudentInfo, int>mapStudent;  
 StudentInfo studentInfo;  
 studentInfo.nID = 1;  
 studentInfo.strName = “student\_one”;  
 mapStudent.insert(pair<StudentInfo, int>(studentInfo, 90));  
 studentInfo.nID = 2;  
 studentInfo.strName = “student\_two”;  
mapStudent.insert(pair<StudentInfo, int>(studentInfo, 80));  
}  
以上程序是无法编译通过的，只要重载小于号，就OK了，如下：  
Typedef struct tagStudentInfo  
{  
 Int      nID;  
 String   strName;  
 Bool operator < (tagStudentInfo const& \_A) const  
 {  
  //这个函数指定排序策略，按nID排序，如果nID相等的话，按strName排序  
  If(nID < \_A.nID)  return true;  
  If(nID == \_A.nID) return strName.compare(\_A.strName) < 0;  
  Return false;  
 }  
}StudentInfo, \*PStudentInfo;  //学生信息  
第二种：仿函数的应用，这个时候结构体中没有直接的小于号重载，程序说明  
#include <map>  
#include <string>  
Using namespace std;  
Typedef struct tagStudentInfo  
{  
 Int      nID;  
 String   strName;  
}StudentInfo, \*PStudentInfo;  //学生信息

Classs sort  
{  
 Public:  
 Bool operator() (StudentInfo const &\_A, StudentInfo const &\_B) const  
 {  
  If(\_A.nID < \_B.nID) return true;  
  If(\_A.nID == \_B.nID) return \_A.strName.compare(\_B.strName) < 0;  
  Return false;  
 }  
};

Int main()  
{  
 //用学生信息映射分数  
 Map<StudentInfo, int, sort>mapStudent;  
 StudentInfo studentInfo;  
 studentInfo.nID = 1;  
 studentInfo.strName = “student\_one”;  
 mapStudent.insert(pair<StudentInfo, int>(studentInfo, 90));  
 studentInfo.nID = 2;  
 studentInfo.strName = “student\_two”;  
mapStudent.insert(pair<StudentInfo, int>(studentInfo, 80));  
}  
另外  
由于STL是一个统一的整体，map的很多用法都和STL中其它的东西结合在一起，比如在排序上，这里默认用的是小于号，即less<>，如果要从大到小排序呢，这里涉及到的东西很多，在此无法一一加以说明。  
还要说明的是，map中由于它内部有序，由红黑树保证，因此很多函数执行的时间复杂度都是log2N的，如果用map函数可以实现的功能，而STL  Algorithm也可以完成该功能，建议用map自带函数，效率高一些。  
下面说下，map在空间上的特性，否则，估计你用起来会有时候表现的比较郁闷，由于map的每个数据对应红黑树上的一个节点，这个节点在不保存你的数据时，是占用16个字节的，一个父节点指针，左右孩子指针，还有一个枚举值（标示红黑的，相当于平衡二叉树中的平衡因子），我想大家应该知道，这些地方很费内存了吧，不说了……

### 2 set

set<int> s;

 //插入了 5 个元素，但由于 8 有重复，第二次插入的 8 并没有执行

 s.insert(8);//第一次插入 8，可以插入

 s.insert(1);

 s.insert(8);//第二次插入 8，重复元素，不会插入

 //中序遍历集合中的元素

 set<int>::iterator it;//定义前向迭代器

 //中序遍历集合中的所有元素

 for(it=s.begin();it!=s.end();it++)

 {

 cout<<\*it<<" ";

 }

 s.erase(1);

c++ stl容器set成员函数:begin()--返回指向第一个元素的迭代器

c++ stl容器set成员函数:clear()--清除所有元素

c++ stl容器set成员函数:count()--返回某个值元素的个数

c++ stl容器set成员函数:empty()--如果集合为空，返回true

c++ stl容器set成员函数:end()--返回指向最后一个元素的迭代器

c++ stl容器set成员函数:equal\_range()--返回集合中与给定值相等的上下限的两个迭代器

c++ stl容器set成员函数:erase()--删除集合中的元素

c++ stl容器set成员函数:find()--返回一个指向被查找到元素的迭代器

c++ stl容器set成员函数:get\_allocator()--返回集合的分配器

c++ stl容器set成员函数:insert()--在集合中插入元素

c++ stl容器set成员函数:lower\_bound()--返回指向大于（或等于）某值的第一个元素的迭代器

c++ stl容器set成员函数:key\_comp()--返回一个用于元素间值比较的函数

c++ stl容器set成员函数:max\_size()--返回集合能容纳的元素的最大限值

c++ stl容器set成员函数:rbegin()--返回指向集合中最后一个元素的反向迭代器

c++ stl容器set成员函数:rend()--返回指向集合中第一个元素的反向迭代器

c++ stl容器set成员函数:size()--集合中元素的数目

c++ stl容器set成员函数:swap()--交换两个集合变量

c++ stl容器set成员函数:upper\_bound()--返回大于某个值元素的迭代器

c++ stl容器set成员函数:value\_comp()--返回一个用于比较元素间的值的函数

### 3 string

string s1 = "abcdeg";  
const char \*k = s1.c\_str();  
const char \*t = s1.data();  
printf("%s%s",k,t);  
cout<<k<<t<<endl;

getline(cin,s7,'a')

string s;

s="aaaaa";

s1="bb";

s=s+s1;

s.length(),s.size();

采用 find()方法可查找字符串中的第一个字符元素（ char， 用单引号界定） 或者子串（用

双引号界定） ， 如果查到， 则返回下标值（从 0 开始计数） ， 如果查不到， 则返回 4294967295。

find函数采用kmp方法实现，比暴力查找快。（kmp自行谷歌）

s="cat dog cat";

 //查找第一个字符‘c’，返回下标值

 cout<<s.find('c')<<endl;

 //查找第一个子串“c”，返回下标值

 cout<<s.find("c")<<endl;

 //查找第一个子串“cat”，返回下标值

 string 比较大小 字典序 a<b

reverse倒置字符串函数reverse：用于倒置字符串s中的各个字符的位置，如原来字符串中如果初始值为123456，则通过reverse函数可将其倒置为654321

accumulate 函数

<http://www.cnblogs.com/heyonggang/p/3241878.html>

Codeforces:Problem 394A - Counting Sticks   (string与substr函数使用）

A. Counting Sticks

time limit per test

 1 second

memory limit per test

 256 megabytes

input

 standard input

output

 standard output

When new students come to the Specialized Educational and Scientific Centre (SESC) they need to start many things from the beginning. Sometimes the teachers say (not always unfairly) that we cannot even count. So our teachers decided to teach us arithmetics from the start. And what is the best way to teach students add and subtract? — That's right, using counting sticks! An here's our new task:

An expression of counting sticks is an expression of type:

[ *A* sticks][sign +][*B* sticks][sign =][*C* sticks] (1 ≤ *A*, *B*, *C*).

Sign + consists of two crossed sticks: one vertical and one horizontal. Sign = consists of two horizontal sticks. The expression is arithmetically correct if *A* + *B* = *C*.

We've got an expression that looks like *A* + *B* = *C* given by counting sticks. Our task is to shift at most one stick (or we can shift nothing) so that the expression became arithmetically correct. Note that we cannot remove the sticks from the expression, also we cannot shift the sticks from the signs + and =.

We really aren't fabulous at arithmetics. Can you help us?

**Input**

The single line contains the initial expression. It is guaranteed that the expression looks like *A* + *B* = *C*, where 1 ≤ *A*, *B*, *C* ≤ 100.

**Output**

If there isn't a way to shift the stick so the expression becomes correct, print on a single line "Impossible" (without the quotes). If there is a way, print the resulting expression. Follow the format of the output from the test samples. Don't print extra space characters.

If there are multiple correct answers, print any of them. For clarifications, you are recommended to see the test samples.

**Sample test(s)**

**input**

||+|=|||||

**output**

|||+|=||||

**input**

|||||+||=||

**output**

Impossible

**input**

|+|=||||||

**output**

Impossible

**input**

||||+||=||||||

**output**

||||+||=||||||

**Note**

In the first sample we can shift stick from the third group of sticks to the first one.

In the second sample we cannot shift vertical stick from + sign to the second group of sticks. So we cannot make a - sign.

There is no answer in the third sample because we cannot remove sticks from the expression.

In the forth sample the initial expression is already arithmetically correct and that is why we don't have to shift sticks.

这题理解错了……

Note that we cannot remove the sticks from the expression, also we cannot shift the sticks from the signs + and =.这句话没理解啊，英语好烂！！

以前用到过substr函数，那时觉得很有用，那时也觉得记住了，没想到忘了，所以还是写这篇以防下次又忘了……

substr函数：比如：string s="abcdef"，string a=s.substr(起始位置，获取长度），注意的是起始位置是从0开始的。如：a=s.substr(1,4)="bcde".

**[cpp]** [view plaincopy](http://blog.csdn.net/u011466175/article/details/19680307)[在CODE上查看代码片](https://code.csdn.net/snippets/199406)

1. #include <iostream>
2. #include <cstdio>
3. #include <fstream>
4. #include <algorithm>
5. #include <cmath>
6. #include <deque>
7. #include <vector>
8. #include <list>
9. #include <queue>
10. #include <string>
11. #include <cstring>
12. #include <map>
13. #include <stack>
14. #include <set>
15. #define PI acos(-1.0)
16. #define mem(a,b) memset(a,b,sizeof(a))
17. #define sca(a) scanf("%d",&a)
18. #define pri(a) printf("%d\n",a)
19. #define f(i,a,n) for(i=a;i<n;i++)
20. #define F(i,a,n) for(i=a;i<=n;i++)
21. #define MM 1000002
22. #define MN 3005
23. #define INF 16843009000000
24. **using** **namespace** std;
25. **typedef** **long** **long** ll;
26. **int** main()
27. {
28. string s,s1,s2,s3;
29. **int** a,b,c,d;
30. cin>>s;
31. a=s.find("+"); //"+"的左边有多少个"|"
32. s1=s.substr(0,a); //取出"+"左边的所有"|"，下同
33. b=s.find("=")-a-1;
34. s2=s.substr(a+1,b);
35. c=a+b; //左边总共的"|"
36. d=s.size()-s.find("=")-1; //右边总共的"|"
37. s3=s.substr(s.find("=")+1,d);
38. **if**(c==d) cout<<s<<endl;
39. **else** **if**(abs(c-d)!=2) puts("Impossible");
40. **else** **if**(abs(c-d)==2)
41. {
42. **if**(c-d==2)
43. {
44. **if**(a==1) cout<<s1<<'+'<<s2.substr(0,b-1);
45. **else** cout<<s1.substr(0,a-1)<<'+'<<s2;
46. cout<<'='<<s3<<'|'<<endl;
47. }
48. **else** cout<<s1<<"|+"<<s2<<'='<<s3.substr(0,d-1)<<endl;
49. }
50. **return** 0;
51. }

string

string类的查找函数：

#include <string>

#include <iostream>

using namespace std;

int find(char c, int pos = 0) const;//从pos开始查找字符c在当前字符串的位置

int find(const char \*s, int pos = 0) const;//从pos开始查找字符串s在当前串中的位置

int find(const char \*s, int pos, int n) const;//从pos开始查找字符串s中前n个字符在当前串中的位置

int find(const string &s, int pos = 0) const;//从pos开始查找字符串s在当前串中的位置

//查找成功时返回所在位置，失败返回string::npos的值

int rfind(char c, int pos = npos) const;//从pos开始从后向前查找字符c在当前串中的位置

int rfind(const char \*s, int pos = npos) const;

int rfind(const char \*s, int pos, int n = npos) const;

int rfind(const string &s,int pos = npos) const;

//从pos开始从后向前查找字符串s中前n个字符组成的字符串在当前串中的位置，成功返回所在位置，失败时返回string::npos的值

int find\_first\_of(char c, int pos = 0) const;//从pos开始查找字符c第一次出现的位置

int find\_first\_of(const char \*s, int pos = 0) const;

int find\_first\_of(const char \*s, int pos, int n) const;

int find\_first\_of(const string &s,int pos = 0) const;

//从pos开始查找当前串中第一个在s的前n个字符组成的数组里的字符的位置。查找失败返回string::npos

int find\_first\_not\_of(char c, int pos = 0) const;

int find\_first\_not\_of(const char \*s, int pos = 0) const;

int find\_first\_not\_of(const char \*s, int pos,int n) const;

int find\_first\_not\_of(const string &s,int pos = 0) const;

//从当前串中查找第一个不在串s中的字符出现的位置，失败返回string::npos

int find\_last\_of(char c, int pos = npos) const;

int find\_last\_of(const char \*s, int pos = npos) const;

int find\_last\_of(const char \*s, int pos, int n = npos) const;

int find\_last\_of(const string &s,int pos = npos) const;

int find\_last\_not\_of(char c, int pos = npos) const;

int find\_last\_not\_of(const char \*s, int pos = npos) const;

int find\_last\_not\_of(const char \*s, int pos, int n) const;

int find\_last\_not\_of(const string &s,int pos = npos) const;

//find\_last\_of和find\_last\_not\_of与find\_first\_of和find\_first\_not\_of相似，只不过是从后向前查找

string类的替换函数：

string &replace(int p0, int n0,const char \*s);//删除从p0开始的n0个字符，然后在p0处插入串s

string &replace(int p0, int n0,const char \*s, int n);//删除p0开始的n0个字符，然后在p0处插入字符串s的前n个字符

string &replace(int p0, int n0,const string &s);//删除从p0开始的n0个字符，然后在p0处插入串s

string &replace(int p0, int n0,const string &s, int pos, int n);//删除p0开始的n0个字符，然后在p0处插入串s中从pos开始的n个字符

string &replace(int p0, int n0,int n, char c);//删除p0开始的n0个字符，然后在p0处插入n个字符c

string &replace(iterator first0, iterator last0,const char \*s);//把[first0，last0）之间的部分替换为字符串s

string &replace(iterator first0, iterator last0,const char \*s, int n);//把[first0，last0）之间的部分替换为s的前n个字符

string &replace(iterator first0, iterator last0,const string &s);//把[first0，last0）之间的部分替换为串s

string &replace(iterator first0, iterator last0,int n, char c);//把[first0，last0）之间的部分替换为n个字符c

string &replace(iterator first0, iterator last0,const\_iterator first, const\_iterator last);//把[first0，last0）之间的部分替换成[first，last）之间的字符串

string类的插入函数：

string &insert(int p0, const char \*s);

string &insert(int p0, const char \*s, int n);

string &insert(int p0,const string &s);

string &insert(int p0,const string &s, int pos, int n);

//前4个函数在p0位置插入字符串s中pos开始的前n个字符

string &insert(int p0, int n, char c);//此函数在p0处插入n个字符c

iterator insert(iterator it, char c);//在it处插入字符c，返回插入后迭代器的位置

void insert(iterator it, const\_iterator first, const\_iterator last);//在it处插入[first，last）之间的字符

void insert(iterator it, int n, char c);//在it处插入n个字符c

string类的删除函数

iterator erase(iterator first, iterator last);//删除[first，last）之间的所有字符，返回删除后迭代器的位置

iterator erase(iterator it);//删除it指向的字符，返回删除后迭代器的位置

string &erase(int pos = 0, int n = npos);//删除pos开始的n个字符，返回修改后的字符串

### 4 list

#include<iostream>

#include<cstdio>

#include<queue>

#include<cstring>

#include<string>

#include<list>

#include<cmath>

#include<algorithm>

#include<assert.h>

#include<stdlib.h>

#include<time.h>

#include<set>

#include<stack>

#include<vector>

#include<map>

#define pi acos(-1.0)

#define inf 1<<20

#define debug puts("\*\*\*\*\*\*\*\*\*\*")

#define mx 200005

typedef long long ll;

using namespace std;

int main()

{

    char c;

    while(~scanf("%c",&c))

    {

        list<char>s;

        list<char>::iterator it=s.begin();

        if(!(c=='['||c==']'))

        {

            s.insert(it++,c);

        }

        while(~scanf("%c",&c))

        {

            if(c=='\n')break;

            if(c=='[')

                it=s.begin();

            else

                if(c==']')

                it=s.end();

            else

                s.insert(it,c);

        }

        for(list<char>::iterator i=s.begin(); i!=s.end(); i++)

            cout<<\*i;

        cout<<endl;

    }

}

### 5 unique 和 unique\_copy

类属性算法unique的作用是从输入序列中“**删除”所有相邻的重复元素**。

该算法删除相邻的重复元素，然后重新排列输入范围内的元素，并且返回一个迭代器（容器的长度没变，只是元素顺序改变了），表示无重复的值范围得结束。

在STL中unique函数是一个去重函数， unique的功能是去除相邻的重复元素(只保留一个),其实它并不真正把重复的元素删除，是把重复的元素移到后面去了，然后依然保存到了原数组中，然后 返回去重后最后一个元素的地址，因为unique去除的是相邻的重复元素，所以一般用之前都会要排一下序。

#include <iostream>

#include <cassert>

#include <algorithm>

#include <vector>

#include <string>

#include <iterator>

 using namespace std;

 int main()

{

    //cout<<"Illustrating the generic unique algorithm."<<endl;

    const int N=11;

    int array1[N]={1,2,0,3,3,0,7,7,7,0,8};

    vector<int> vector1;

    for (int i=0;i<N;++i)

        vector1.push\_back(array1[i]);

    vector<int>::iterator new\_end;

    new\_end=unique(vector1.begin(),vector1.end());    //"删除"相邻的重复元素

    assert(vector1.size()==N);

    vector1.erase(new\_end,vector1.end());  //删除（真正的删除）重复的元素

    copy(vector1.begin(),vector1.end(),ostream\_iterator<int>(cout," "));

    cout<<endl;

    return 0;

}

**unique\_copy**

算法标准库定义了一个名为unique\_copy的函数，其操作类似于unique。

唯一的区别在于：**前者接受第三个迭代器实参，用于指定复制不重复元素的目标序列。**

**unique\_copy根据字面意思就是去除重复元素再执行copy运算。**

**//使用unique\_copy算法**

**//将一个list对象中不重复的元素赋值到一个空的vector对象中**

**#include<iostream>**

**#include<list>**

**#include<vector>**

**#include<algorithm>**

**using namespace std;**

**int main()**

**{**

**int ia[7] = {5 , 2 , 2 , 2 , 100 , 5 , 2};**

**list<int> ilst(ia , ia + 7);**

**vector<int> ivec;**

**//将list对象ilst中不重复的元素复制到空的vector对象ivec中**

**//sort(ilst.begin() , ilst.end());  //不能用此种排序，会报错**

**ilst.sort();  //在进行复制之前要先排序，切记**

**unique\_copy(ilst.begin() , ilst.end() , back\_inserter(ivec));**

**//输出vector容器**

**cout<<"vector: "<<endl;**

**for(vector<int>::iterator iter = ivec.begin() ; iter != ivec.end() ; ++iter)**

**cout<<\*iter<<" ";**

**cout<<endl;**

**return 0;**

**}**

### 6 vector

vector<int> v;

vector<int> v(10);

vector<int> v(10,0);

v.push\_back(2);

v[1],v[2];

vector<int>::iterator it;

for(it=v.begin(); it!=v.end(); it++)

{

   cout<<\*it<<" ";

}

排序 sort(v.begin(),v.end());

清空 v.clear();

大小 v.size();vector容器是一个模板类，可以存放任何类型的对象（但必须是同一类对象）。vector对象可以在运行时高效地添加元素，并且vector中元素是连续存储的。  
vector的构造  
  
函数原型：  
template<typename T>  
 explicit vector(); // 默认构造函数，vector对象为空  
 explicit vector(size\_type n, const T& v = T()); // 创建有n个元素的vector对象  
 vector(const vector& x);  
 vector(const\_iterator first, const\_iterator last);  
  
注：vector容器内存放的所有对象都是经过初始化的。如果没有指定存储对象的初始值，那么对于内置类型将用0初始化，对于类类型将调用其默认构造函数进行初始化（如果有其它构造函数而没有默认构造函数，那么此时必须提供元素初始值才能放入容器中）。  
  
举例：  
vector<string> v1; // 创建空容器，其对象类型为string类  
vector<string> v2(10); // 创建有10个具有初始值（即空串）的string类对象的容器  
vector<string> v3(5, "hello"); // 创建有5个值为“hello”的string类对象的容器  
vector<string> v4(v3.begin(), v3.end()); // v4是与v3相同的容器（完全复制）  
  
vector的操作（下面的函数都是成员函数）  
  
bool empty() const; // 如果为容器为空，返回true；否则返回false  
size\_type max\_size() const; // 返回容器能容纳的最大元素个数  
size\_type size() const; // 返回容器中元素个数   
size\_type capacity() const; // 容器能够存储的元素个数，有：capacity() >= size()   
void reserve(size\_type n); // 确保capacity() >= n  
void resize(size\_type n, T x = T()); // 确保返回后，有：size() == n；如果之前size()<n，那么用元素x的值补全。  
  
reference front(); // 返回容器中第一个元素的引用（容器必须非空）  
const\_reference front() const;   
reference back(); // 返回容器中最后一个元素的引用（容器必须非空）  
const\_reference back() const;  
  
reference operator[](size\_type pos); // 返回下标为pos的元素的引用（下标从0开始；如果下标不正确，则属于未定义行为。  
const\_reference operator[](size\_type pos) const;   
reference at(size\_type pos); // 返回下标为pos的元素的引用；如果下标不正确，则抛出异常out\_of\_range  
const\_reference at(size\_type pos) const;  
  
void push\_back(const T& x); // 向容器末尾添加一个元素   
void pop\_back(); // 弹出容器中最后一个元素（容器必须非空）  
  
// 注：下面的插入和删除操作将发生元素的移动（为了保持连续存储的性质），所以之前的迭代器可能失效  
iterator insert(iterator it, const T& x = T()); // 在插入点元素之前插入元素（或者说在插入点插入元素）  
void insert(iterator it, size\_type n, const T& x); // 注意迭代器可能不再有效（可能重新分配空间）  
void insert(iterator it, const\_iterator first, const\_iterator last);  
  
iterator erase(iterator it); // 删除指定元素，并返回删除元素后一个元素的位置（如果无元素，返回end()）  
iterator erase(iterator first, iterator last); // 注意：删除元素后，删除点之后的元素对应的迭代器不再有效。  
  
void clear() const; // 清空容器，相当于调用erase( begin(), end())  
  
void assign(size\_type n, const T& x = T()); // 赋值，用指定元素序列替换容器内所有元素  
void assign(const\_iterator first, const\_iterator last);  
  
const\_iterator begin() const; // 迭代序列  
iterator begin();  
const\_iterator end() const;  
iterator end();  
  
const\_reverse\_iterator rbegin() const;  
reverse\_iterator rbegin();  
const\_reverse\_iterator rend() const;   
reverse\_iterator rend();  
  
vector对象的比较（非成员函数）  
  
 针对vector对象的比较有六个比较运算符：operator==、operator!=、operator<、operator<=、operator>、operator>=。  
  
 其中，对于operator==和operator!=，如果vector对象拥有相同的元素个数，并且对应位置的元素全部相等，则两个vector对象相等；否则不等。  
 对于operator<、operator<=、operator>、operator>=，采用字典排序策略比较。  
  
注：其实只需要实现operator==和operator!=就可以了，其它可以根据这两个实现。因为，operator!= (lhs, rhs) 就是 !(lhs == rhs)，operator<=(lhs, rhs) 就是 !(rhs < lhs)，operator>(lhs, rhs) 就是 (rhs < lhs)，operator>=（lhs, rhs) 就是 !(lhs, rhs)。  
  
vector类的迭代器  
  
 vector类的迭代器除了支持通用的前缀自增运算符外，还支持算术运算：it + n、it - n、it2 - it1。注意it2 - it1返回值为difference\_type（signed类型）。  
  
 注意，任何改变容器大小的操作都可能造成以前的迭代器失效。  
  
应用示例  
  
#include <iostream>  
#include <cassert>  
#include <vector>  
  
using namespace std;  
  
int main()  
{  
 vector<string> v(5, "hello");  
 vector<string> v2(v.begin(), v.end());  
  
 assert(v == v2);  
  
 cout<<"> Before operation"<<endl;  
 for(vector<string>::const\_iterator it = v.begin(); it < v.end(); ++it)  
 cout<<\*it<<endl;  
  
 v.insert(v.begin() + 3, 4, "hello, world");  
 cout<<"> After insert"<<endl;  
 for(vector<string>::size\_type i = 0; i < v.size(); ++i)  
 cout<<v[i]<<endl;  
  
 vector<string>::iterator it = v.erase(v.begin() + 3, v.begin() + 6);  
 assert(\*it == "hello, world");  
 cout<<"> After erase"<<endl;  
 for(vector<string>::size\_type i = 0; i != v.size(); ++i)  
 cout<<v[i]<<endl;  
  
 assert(v.begin() + v.size() == v.end());  
 assert(v.end() - v.size() == v.begin());  
 assert(v.begin() - v.end() == -vector<string>::difference\_type(v.size()));  
  
 return 0;  
}  
程序说明：上面程序中用了三个循环输出容器中的元素，每个循环的遍历方式是不一样的。特别需要说明的是，第二个循环在条件判断中使用了size() 函数，而不是在循环之前先保存在变量中再使用。之所以这样做，有两个原因：其一，如果将来在修改程序时，在循环中修改了容器元素个数，这个循环仍然能很好 地工作，而如果先保存size()函数值就不正确了；其二，由于这些小函数（其实现只需要一条返回语句）基本上都被声明为inline，所以不需要考虑效率问题。

### 7 next\_permutation and prev\_permutation

next\_permutation

（1）如果给你组数：1 2 3 4 5

（2）如果给出的条件是让你从小到大求出下一个排列，当然就是：1 2 3 5 4

（3）如果再下一个就是：1 2 4 3 5   ；再下一个是：1 2 4 5 3

（4）如果问你求的是下n个排列是什么？你该怎么快速求出呢？？？

当然，在（2）中的答案可以从（1）中的倒数第一位与第二位比较，如果倒数第一位比第二位大，那么交换位置，然后下一个排列就是交换后的了，即：1 2 3 5 4；

然后依次下去用这种思想可以求出下面的排列……

但是你想想每位都要比较判断，如果相等又往前推，依次这样的话，判断语句又多，代码也不会短，你不烦我还烦呢……  ^\_^  嘿嘿，想偷懒就得想个法子解决问题……

所以懒也是有原因的……next\_permutation出场的时候就到了……

使用方法： next\_permutation(数组头地址，数组尾地址); 若下一个排列存在，则返回真，如果不存在则返回假。

举例：（1）数据数组

Int a[6]={1,2,3,5,4};

If(next\_permutation(a,a+5))

{

for(int i=0;i<5;i++)

cout<<a[i]<<‘ ‘;

}

输出：1 2 4 5 3;也就是1 2 3 5 4的下一个排列咯，是不是很快……

（2）字符串（字母数组）

String str=“abcde”;  //输出下5个排列

Int k=0;

While(next\_permutation(str.begin(),str.end())) //begin()与end()是string类的迭代器

{

k++;

cout<<str<<endl;

if(k==5) break;

}

输出：

abced

abdce

abdec

abecd

Abedc

问题又来了，既然next\_permutation可以求下一个排列，那有没有函数可以求上一个排列的？当然有啦！就是prev\_permutation。用法与next\_permutation是一样的。

例：

String str=“abecd”;

if(prev\_permutation(str.begin(),str.end())) cout<<str<<endl;

输出：abdec; //看上面例子

### 8 stack

stack栈

Stack<int>s;

包含以下几个成员函数：  
s.empty() 判断栈是否为空

s.push() 在栈顶增加元素

s.top() 返回栈顶元素   
s.pop() 删除栈顶元素  
s.size() 返回栈中元素数目

#include <iostream>

#include <stack>

using namespace std;

int main()

{

    stack<int>s;

    for(int i=1;i<10;i++) s.push(i); //入栈

    cout<<s.size()<<endl; //栈中有多少元素

    while(!s.empty())

    {cout<<s.top()<<' '; //读取栈顶元素

        s.pop();//删除栈顶元素

    }

    return 0;

}

### 9 queue

stack栈

Stack<int>s;

包含以下几个成员函数：  
s.empty() 判断栈是否为空

s.push() 在栈顶增加元素

s.top() 返回栈顶元素   
s.pop() 删除栈顶元素  
s.size() 返回栈中元素数目

#include <iostream>

#include <stack>

using namespace std;

int main()

{

    stack<int>s;

    for(int i=1;i<10;i++) s.push(i); //入栈

    cout<<s.size()<<endl; //栈中有多少元素

    while(!s.empty())

    {cout<<s.top()<<' '; //读取栈顶元素

        s.pop();//删除栈顶元素

    }

    return 0;

}

## 3 基本算法

### 1 二分

///二分（单调增）

double l=0,r=100,mid;

while(r-l>eps)

{

mid=(l+r)/2;

if (f(mid)>0)

r=mid;

else

l=mid;

}

///二分（精确binary\_search）

int l=0,r=10000,mid;

while(l<=r)

{

mid=l+r>>1;

if (a[mid]>key)

r=mid-1;

else if (a[mid]<key)

l=mid+1;

else

break;

}

if (a[mid]==key);

///二分（lower\_bound）

int l=0,r=10000,mid;

while(l<r)

{

mid=l+r>>1;

if (a[mid]>=key)

r=mid;

else if (a[mid]<key)

l=mid+1;

}

if (a[l]==key);

### 2 三分

///（极小值）

double left,right,mid1,mid2;

left=-1e7;

right=1e7;

while(fabs(left-right)>eps)

{

mid1=(left+right)/2;

mid2=(mid1+right)/2;

if (f(mid1)<f(mid2))

right=mid2;

else

left=mid1;

}

## 4 数据结构

### 1 树桩数组

//一定要给N赋值，一定要清空数组

int N;

const int NV=100005;

int c[NV];

inline int lowbit(int t)

{

    return t&(-t);

}

void update(int x,int v)

{

    while(x<=N)

    {

        c[x]+=v;

        x+=lowbit(x);

    }

}

int query(int x)

{

    int ans=0;

    while(x>0)

    {

        ans+=c[x];

        x-=lowbit(x);

    }

    return ans;

}

int findkth(int k)

{

    int idx = 0;

    for(int i=20; i>=0; i--)

    {

        idx |= 1 << i;

        if(idx <= N && c[idx] < k)

            k -= c[idx];

        else idx ^= 1 << i;

    }

    return idx + 1;

}

**A - 敌兵布阵**

**Time Limit:**1000MS     **Memory Limit:**32768KB     **64bit IO Format:**%I64d & %I64u

Submit [Status](http://acm.hust.edu.cn/vjudge/contest/view.action?cid=78286#status//A/0) [Practice](http://acm.hust.edu.cn/vjudge/problem/viewProblem.action?id=16216) [HDU 1166](http://acm.hust.edu.cn/vjudge/problem/visitOriginUrl.action?id=16216)

**Description**

C国的死对头A国这段时间正在进行军事演习，所以C国间谍头子Derek和他手下Tidy又开始忙乎了。A国在海岸线沿直线布置了N个工兵营地,Derek和Tidy的任务就是要监视这些工兵营地的活动情况。由于采取了某种先进的监测手段，所以每个工兵营地的人数C国都掌握的一清二楚,每个工兵营地的人数都有可能发生变动，可能增加或减少若干人手,但这些都逃不过C国的监视。   
中央情报局要研究敌人究竟演习什么战术,所以Tidy要随时向Derek汇报某一段连续的工兵营地一共有多少人,例如Derek问:“Tidy,马上汇报第3个营地到第10个营地共有多少人!”Tidy就要马上开始计算这一段的总人数并汇报。但敌兵营地的人数经常变动，而Derek每次询问的段都不一样，所以Tidy不得不每次都一个一个营地的去数，很快就精疲力尽了，Derek对Tidy的计算速度越来越不满:"你个死肥仔，算得这么慢，我炒你鱿鱼!”Tidy想：“你自己来算算看，这可真是一项累人的工作!我恨不得你炒我鱿鱼呢!”无奈之下，Tidy只好打电话向计算机专家Windbreaker求救,Windbreaker说：“死肥仔，叫你平时做多点acm题和看多点算法书，现在尝到苦果了吧!”Tidy说："我知错了。。。"但Windbreaker已经挂掉电话了。Tidy很苦恼，这么算他真的会崩溃的，聪明的读者，你能写个程序帮他完成这项工作吗？不过如果你的程序效率不够高的话，Tidy还是会受到Derek的责骂的.

**Input**

第一行一个整数T，表示有T组数据。   
每组数据第一行一个正整数N（N<=50000）,表示敌人有N个工兵营地，接下来有N个正整数,第i个正整数ai代表第i个工兵营地里开始时有ai个人（1<=ai<=50）。   
接下来每行有一条命令，命令有4种形式：   
(1) Add i j,i和j为正整数,表示第i个营地增加j个人（j不超过30）   
(2)Sub i j ,i和j为正整数,表示第i个营地减少j个人（j不超过30）;   
(3)Query i j ,i和j为正整数,i<=j，表示询问第i到第j个营地的总人数;   
(4)End 表示结束，这条命令在每组数据最后出现;   
每组数据最多有40000条命令

**Output**

对第i组数据,首先输出“Case i:”和回车,   
对于每个Query询问，输出一个整数并回车,表示询问的段中的总人数,这个数保持在int以内。

**Sample Input**

1

10

1 2 3 4 5 6 7 8 9 10

Query 1 3

Add 3 6

Query 2 7

Sub 10 2

Add 6 3

Query 3 10

End

**Sample Output**

Case 1:

6

33

59

#include<iostream>

#include<cstdio>

#include<queue>

#include<cstring>

#include<string>

#include<cmath>

#include<algorithm>

#include<assert.h>

#include<stdlib.h>

#include<time.h>

#include<set>

#include<stack>

#include<vector>

#include<map>

#define pi acos(-1.0)

#define inf 1<<20

#define debug puts("\*\*\*\*\*\*\*\*\*\*");

#define mem(x) memset(x,0,sizeof(x))

typedef long long ll;

using namespace std;

const int mod = 1e9+7;

const int maxn= 100010;

ll ans = 0;

int a[maxn]= {0};

int temp[maxn]= {0};

void Merge(int l,int m,int r)

{

    int i=l,j=m+1,k=l;

    while(i<=m&&j<=r)

    {

        if(a[i]>a[j])

        {

            temp[k++]=a[j++];

            ans+=m-i+1;

            //printf("l=%d m=%d r=%d i=%d j=%d a[i]=%d a[j]=%d\n",l,m,r,i,j-1,a[i],a[j]);

        }

        else

        {

            temp[k++]=a[i++];

        }

    }

    while(i<=m)temp[k++]=a[i++];

    while(j<=r)temp[k++]=a[j++];

    for(i=l; i<=r; i++)

        a[i]=temp[i];

}

void Merge\_sort(int l,int r)

{

    if(l<r)

    {

        int m=(l+r)/2;

        Merge\_sort(l,m);

        Merge\_sort(m+1,r);

        Merge(l,m,r);

    }

}

int main()

{

    int T,I;

    scanf("%d",&T);

    for(I=1; I<=T; I++)

    {

        mem(a);

        mem(temp);

        ans=0;

        int n;

        scanf("%d",&n);

        int i;

        for(i=0; i<n; i++)

            scanf("%d",a+i);

        Merge\_sort(0,n-1);

        printf("Scenario #%d:\n%I64d\n\n",I,ans);

    }

}

#include<iostream>

#include<cstdio>

#include<queue>

#include<cstring>

#include<string>

#include<cmath>

#include<algorithm>

#include<assert.h>

#include<stdlib.h>

#include<time.h>

#include<stack>

#include<vector>

#include<map>

#define pi acos(-1.0)

#define inf 1<<29

typedef long long ll;

using namespace std;

int n,num=1;

int c[50005];

int lowbit(int x)

{

    return x&(-x);

}

void  update(int i,  int x)

{

    while(i<=n)  {c[i] = c[i]+x; i += lowbit(i);}

}

int sum(int i)

{

    int  ans = 0;

    while(i>0) { ans += c[i]; i -= lowbit(i);}

    return ans;

}

int main()

{

    int t,i,j,l;

    scanf("%d",&t);

    while(t--)

    {

        scanf("%d",&n);

        memset(c,0,sizeof(c));

        for(i=1;i<=n;i++)

        {

            scanf("%d",&l);

            update(i,l);

        }

        printf("Case %d:\n",num++);

        char s[10];

        while(scanf("%s",s)!=EOF)

        {

            if(s[0]=='E')break;scanf("%d%d",&i,&j);

            switch(s[0])

            {

                case 'A':update(i,j);break;

                case 'S':update(i,-j);break;

                case 'Q':printf("%d\n",sum(j)-sum(i-1));break;

                default : printf("wrroy");

            }

        }

    }

    return 0;

}

二维：

#include<stdio.h>

#include<string.h>

#include<iostream>

using namespace std;

#define N 505

int a[N][N];

int sum[N][N];

int n,m,q;

int lowbit(int x)

{

return x&(-x);

}

void update(int x,int y,int k)

{

for(int i=x;i<=n;i+=lowbit(i)){

for(int j=y;j<=m;j+=lowbit(j)){

sum[i][j]^=k;

}

}

}

int query(int x,int y)

{

int ans=0;

for(int i=x;i>0;i-=lowbit(i)){

for(int j=y;j>0;j-=lowbit(j)){

ans^=sum[i][j];

}

}

return ans;

}

int main()

{

int T,x1,x2,y1,y2,k;

scanf("%d",&T);

while(T--){

memset(sum,0,sizeof(sum));

scanf("%d%d%d",&n,&m,&q);

for(int i=1;i<=n;i++){

for(int j=1;j<=m;j++){

scanf("%d",&a[i][j]);

update(i,j,a[i][j]);

}

}

while(q--){

scanf("%d",&k);

if(k==1){

scanf("%d %d %d %d", &x1, &y1, &x2, &y2);

int ans = 0;

ans ^= query(x2, y2);

if(y1 > 1) ans ^= query(x2, y1 - 1);

if(x1 > 1) ans ^= query(x1 - 1, y2);

if(x1 > 1 && y1 > 1) ans ^= query(x1 - 1, y1 - 1);

if(ans == 0) {

printf("No\n");

} else {

printf("Yes\n");

}

}

else{

scanf("%d%d%d",&x1,&y1,&x2);

update(x1,y1,a[x1][y1]^x2);

a[x1][y1]=x2;

}

}

}

return 0;

}

### 2 RMQ

/\*

下标从1开始

\*/

const int NV = 255;

const int NVB = 20;

int mx[NV][NVB],mn[NV][NVB],a[NV];

void init(int data[],int n)

{

    int k = log2(n);

    for(int i=1; i<=n; i++)

        mx[i][0] = mn[i][0] = data[i];

    for(int j=1; j<=k; j++)

    {

        for(int i=1; i+(1<<j)-1<=n; i++)

        {

            mx[i][j] = max(mx[i][j-1],mx[i+(1<<(j-1))][j-1]);

            mn[i][j] = min(mn[i][j-1],mn[i+(1<<(j-1))][j-1]);

        }

    }

}

int query(int l,int r,int flag)

{

    int k = log2(r-l+1);

    if(flag)

        return max(mx[l][k],mx[r-(1<<k)+1][k]);

    else

        return min(mn[l][k],mn[r-(1<<k)+1][k]);

}

二维

int val[255][255];

int mm[255];

int dpmin[255][255][8][8];//最小值

int dpmax[255][255][8][8];//最大值

void initRMQ(int n,int m)

{

    for(int i = 1;i <= n;i++)

        for(int j = 1;j <= m;j++)

            dpmin[i][j][0][0] = dpmax[i][j][0][0] = val[i][j];

    for(int ii = 0; ii <= mm[n]; ii++)

        for(int jj = 0; jj <= mm[m]; jj++)

            if(ii+jj)

                for(int i = 1; i + (1<<ii) - 1 <= n; i++)

                    for(int j = 1; j + (1<<jj) - 1 <= m; j++)

                    {

                        if(ii)

                        {

                            dpmin[i][j][ii][jj] = min(dpmin[i][j][ii-1][jj],dpmin[i+(1<<(ii-1))][j][ii-1][jj]);

                            dpmax[i][j][ii][jj] = max(dpmax[i][j][ii-1][jj],dpmax[i+(1<<(ii-1))][j][ii-1][jj]);

                        }

                        else

                        {

                            dpmin[i][j][ii][jj] = min(dpmin[i][j][ii][jj-1],dpmin[i][j+(1<<(jj-1))][ii][jj-1]);

                            dpmax[i][j][ii][jj] = max(dpmax[i][j][ii][jj-1],dpmax[i][j+(1<<(jj-1))][ii][jj-1]);

                        }

                    }

}

//查询矩形的最大值

int rmq1(int x1,int y1,int x2,int y2)

{

    int k1 = mm[x2-x1+1];

    int k2 = mm[y2-y1+1];

    x2 = x2 - (1<<k1) + 1;

    y2 = y2 - (1<<k2) + 1;

    return max(max(dpmax[x1][y1][k1][k2],dpmax[x1][y2][k1][k2]),max(dpmax[x2][y1][k1][k2],dpmax[x2][y2][k1][k2]));

}

//查询矩形的最小值

int rmq2(int x1,int y1,int x2,int y2)

{

    int k1 = mm[x2-x1+1];

    int k2 = mm[y2-y1+1];

    x2 = x2 - (1<<k1) + 1;

    y2 = y2 - (1<<k2) + 1;

    return min(min(dpmin[x1][y1][k1][k2],dpmin[x1][y2][k1][k2]),min(dpmin[x2][y1][k1][k2],dpmin[x2][y2][k1][k2]));

}

### 3 线段树

#include <stdio.h>  
#include <math.h>  
#include <iostream>  
  
using namespace std;  
#define lson l,m,rt<<1  
#define rson m+1,r,rt<<1|1  
const int NV = 100005;  
#define INF 0x3f3f3f3f  
int n,m;  
  
  
struct node  
{  
    long long oo,oj,jo,jj;  
    node():oo(-INF),oj(-INF),jj(-INF),jo(-INF) {}  
}sum[NV<<2];  
  
long long max(long long a,long long b)  
{  
    if(a > b) return a;  
    else return b;  
}  
  
void PushUp(int rt)  
{  
    sum[rt].jj = max(max(sum[rt<<1].jj,sum[rt<<1|1].jj),max(sum[rt<<1].jj+sum[rt<<1|1].oj,sum[rt<<1].jo+sum[rt<<1|1].jj));  
  
    sum[rt].oo = max(max(sum[rt<<1].oo,sum[rt<<1|1].oo),max(sum[rt<<1].oo+sum[rt<<1|1].jo,sum[rt<<1].oj+sum[rt<<1|1].oo));  
  
    sum[rt].jo = max(max(sum[rt<<1].jo,sum[rt<<1|1].jo),max(sum[rt<<1].jj+sum[rt<<1|1].oo,sum[rt<<1].jo+sum[rt<<1|1].jo));  
  
    sum[rt].oj = max(max(sum[rt<<1].oj,sum[rt<<1|1].oj),max(sum[rt<<1].oo+sum[rt<<1|1].jj,sum[rt<<1].oj+sum[rt<<1|1].oj));  
  
  
}  
void build(int l,int r,int rt=1)  
{  
    if (l == r)  
    {  
        if(l % 2 == 1)  
        {  
        scanf("%I64d",&sum[rt].jj);  
        sum[rt].oo = sum[rt].jo = sum[rt].oj = -INF;  
  
        }  
        else  
        {  
            scanf("%I64d",&sum[rt].oo);  
            sum[rt].jj = sum[rt].oj = sum[rt].jo = -INF;  
  
        }  
            //此处为sum[rt]赋值，即为scanf("%d",&sum[rt]);  
        return ;  
    }  
    int m = (l + r) >> 1;  
    build(lson);  
    build(rson);  
    PushUp(rt);  
}  
void update(long long L,long long c,int l,int r,int rt=1)  //L为插入位置，c为增加的值  
{  
    if (L == l && l == r)  
    {  
  
        if(l % 2 == 1)  
        {  
            sum[rt].jj = c;  
            sum[rt].oj = sum[rt].oo = sum[rt].jo = -INF;  
        }  
        else  
        {  
            sum[rt].oo = c;  
            sum[rt].jj = sum[rt].oj = sum[rt].jo = -INF;  
        }  
        return ;  
    }  
    int m = (l + r) >> 1;  
    if (L <= m) update(L , c , lson);  
    else update(L , c , rson);  
    PushUp(rt);  
}  
  
  
  
node query(long long  L,long long R,int l,int r,int rt=1)//(L,R)为要查询的区间  
{  
    node temp,ltemp,rtemp;  
    if (L <= l && r <= R)  
        {  
            temp.jj = sum[rt].jj;  
            temp.oo = sum[rt].oo;  
            temp.oj = sum[rt].oj;  
            temp.jo = sum[rt].jo;  
            return temp;  
        }  
    int m = (l + r) >> 1;  
    if (L <= m) ltemp = query(L , R , lson);  
    if (m < R) rtemp = query(L , R , rson);      //注：两个判断都为if  
  
     temp.jj=max(max(max(ltemp.jj+rtemp.oj,ltemp.jo+rtemp.jj),ltemp.jj),rtemp.jj);  
     temp.oo=max(max(max(ltemp.oo+rtemp.jo,ltemp.oj+rtemp.oo),ltemp.oo),rtemp.oo);  
     temp.jo=max(max(max(ltemp.jo+rtemp.jo,ltemp.jj+rtemp.oo),ltemp.jo),rtemp.jo);  
     temp.oj=max(max(max(ltemp.oj+rtemp.oj,ltemp.oo+rtemp.jj),ltemp.oj),rtemp.oj);  
      return temp;  
}  
  
int main()  
{  
    int t;  
    long long b,c;  
    int a;  
    scanf("%d",&t);  
    while(t--){  
     scanf("%d%d",&n,&m);  
     build(1,n,1);  
     for(int i = 0 ; i < m ; i++)  
     {  
         scanf("%d%I64d%I64d",&a,&b,&c);  
         if(a == 1)  
         {  
             update(b,c,1,n);  
         }  
         else  
         {  
             node tempc = query(b,c,1,n);  
             long long ans = max(max(max(tempc.oo,tempc.jj),tempc.jo),tempc.oj);  
             printf("%I64d\n",ans);  
         }  
     }  
  
    }  
    return  0;  
  
}

### 4 树链刨分（没验证过）

Hdu3966

const int NV=50005;

//num为初始数组，rank只在为初始数组build时使用

int num[NV],siz[NV],top[NV],son[NV];

int dep[NV],w[NV],rank[NV],fa[NV];

int he[NV],to[2\*NV],next[2\*NV],ecnt,tot;

void adde(int u,int v)

{

to[ecnt]=v,next[ecnt]=he[u],he[u]=ecnt++;

to[ecnt]=u,next[ecnt]=he[v],he[v]=ecnt++;

}

void dfs1(int u,int father,int d)

{

dep[u]=d;

fa[u]=father;

siz[u]=1;

for(int i=he[u]; ~i; i=next[i])

{

int v=to[i];

if(v!=father)

{

dfs1(v,u,d+1);

siz[u]+=siz[v];

if(son[u]==-1||siz[v]>siz[son[u]])

son[u]=v;

}

}

}

void dfs2(int u,int tp)

{

top[u]=tp;

w[u]=++tot;

rank[w[u]]=u;

if(son[u]==-1) return;

dfs2(son[u],tp);

for(int i=he[u]; ~i; i=next[i])

{

int v=to[i];

if(v!=son[u]&&v!=fa[u])

dfs2(v,v);

}

}

#define lson l,m,rt<<1

#define rson m+1,r,rt<<1|1

int add[NV<<2],sum[NV<<2];

void PushUp(int rt)

{

sum[rt]=sum[rt<<1]+sum[rt<<1|1];

}

void PushDown(int rt,int m)

{

if (add[rt])

{

add[rt<<1] += add[rt];

add[rt<<1|1] += add[rt];

sum[rt<<1] += add[rt] \* (m - (m >> 1));

sum[rt<<1|1] += add[rt] \* (m >> 1);

add[rt] = 0;

}

}

void build(int l,int r,int rt=1)

{

add[rt] = 0;

if (l == r)

{

sum[rt]=num[rank[l]];

return ;

}

int m = (l + r) >> 1;

build(lson);

build(rson);

PushUp(rt);

}

void update(int L,int R,int c,int l,int r,int rt=1)

{

if (L <= l && r <= R)

{

add[rt] += c;

sum[rt] += c \* (r - l + 1);

return ;

}

PushDown(rt , r - l + 1);

int m = (l + r) >> 1;

if (L <= m) update(L , R , c , lson);

if (m < R) update(L , R , c , rson);

PushUp(rt);

}

int query(int L,int l,int r,int rt=1)

{

if (L == l && l == r)

{

return sum[rt];

}

PushDown(rt , r - l + 1);

int m = (l + r) >> 1;

if (L <= m) return query(L , lson);

return query(L , rson);

}

void change(int x,int y,int l,int r,int c)

{

while(top[x]!=top[y])

{

if(dep[top[x]]<dep[top[y]]) swap(x,y);

update(w[top[x]],w[x],c,l,r);

x=fa[top[x]];

}

if(dep[x]>dep[y]) swap(x,y);

update(w[x],w[y],c,l,r);

}

void init(int n)

{

memset(he,-1,sizeof(he));

memset(son,-1,sizeof(son));

tot=0;

ecnt=0;

for(int i=1; i<=n; i++)

scanf("%d",&num[i]);

for(int i=1; i<n; i++)

{

int u,v;

scanf("%d%d",&u,&v);

adde(u,v);

}

dfs1(1,0,0);

dfs2(1,1);

build(1,n);

}

int main()

{

int n,q;

while(~scanf("%d%\*d%d",&n,&q))

{

init(n);

while(q--)

{

int a,b,c;

char op[5];

scanf("%s",op);

if(op[0]=='Q')

{

scanf("%d",&a);

printf("%d\n",query(w[a],1,n));

}

else

{

scanf("%d%d%d",&a,&b,&c);

if(op[0]=='D') c=-c;

change(a,b,1,n,c);

}

}

}

return 0;

}

### 5 离散化

void discrete(int \*aa,int nn)

{

    memcpy(cp,aa,sizeof(cp));

    sort(cp,cp+nn);

    int tot=unique(cp,cp+nn)-cp;

    for(int i=0;i<nn;i++)

        aa[i]=lower\_bound(cp,cp+tot,aa[i])-cp+1;

}

### 6 并查集

http://blog.csdn.net/lalor/article/details/7388805

模板一

const int LEN = 50010;

int f[LEN],rk[LEN];

//int finds(int x) //非递归

//{

//    int k,j,r;

//    r=x;

//    while(r!=f[r])

//        r=f[r];

//    k=x;

//    while(k!=r)

//    {

//        j=f[k];

//        f[k]=r;

//        k=j;

//    }

//    return r;

//}

int finds(int x) //递归

{

    return f[x]==x?x:f[x]=finds(f[x]);

}

void merge(int a,int b)

{

    a=finds(a);

    b=finds(b);

    if (a==b) return;

    if (rk[a]>rk[b]) f[b]=a;

    else

    {

        if (rk[a]==rk[b]) rk[b]++;

        f[a]=b;

    }

}

void init(int n)

{

    memset(rk,0,sizeof(rk));

    for (int i=0; i<=n; i++)

        f[i]=i;

}

模板二

int fa[x];  
int zip(int x)  
{  
    if(x==fa[x])return x;  
    return fa[x]=zip(fa[x]);  
}  
int main()  
{  
    int x,y;  
    for(int i=0;i<mx;i++)  
        fa[i]=i;  
    if(zip(x)!=zip(y))fa[zip(x)]=zip(y);  
}

## 5 字符串

### 1 KMP

#define len 500000

int next[len];

int s[len];

int sub[len];

void getnext(int sub[],int next[],int len2)

{

    int i=0,j=-1;

    next[0]=-1;

    while(i<len2)

    {

        if (j==-1||sub[i]==sub[j])

        {

            i++;

            j++;

            next[i]=j;

        }

        else

        {

            j=next[j];

        }

    }

}

int kmp(int s[],int sub[],int len1,int len2,int next[])

{

    int i,j;

    i=0;

    j=0;

    while(i<len1&&j<len2)

    {

        if(j==-1||s[i]==sub[j])

        {

            i++;

            j++;

        }

        else

        {

            j=next[j];

        }

    }

    if(j>=len2)

    {

        return i-len2;

    }

    else

    {

        return -1;

    }

}

### 2 字典树

const int NS=1005\*25;

int c[NS][128];

struct trie

{

// int val[NS];

int cnt[NS];

int sz;

trie()

{

sz=1;

memset(c[0],0,sizeof(c[0]));

memset(cnt,0,sizeof(cnt));

}

void insert(char s[],int v=0)

{

int u=0;

for (int i=0; s[i]; i++)

{

if (!c[u][s[i]])

{

memset(c[sz],0,sizeof(c[sz]));

// val[sz]=0;

c[u][s[i]]=sz++;

}

u=c[u][s[i]];

cnt[u]++;

}

// val[u]=v;

}

int query(char s[])

{

int u=0,n=strlen(s);

for (int i=0; i<n; i++)

{

if (!c[u][s[i]]||u!=0&&cnt[u]<=1)

return i;

u=c[u][s[i]];

}

return n;

// return cnt[u];

}

};

### 3 AC自动机

#include<iostream>

#include<cstdio>

#include<queue>

#include<cstring>

#include<string>

#include<cmath>

#include<algorithm>

#include<assert.h>

#include<stdlib.h>

#include<time.h>

#include<set>

#include<stack>

#include<vector>

#include<map>

#define pi acos(-1.0)

#define inf 1<<29

#define debug puts("\*\*\*\*\*\*\*\*\*\*");

#define mem(x) memset(x,0,sizeof(x))

const int mx = 20010;

typedef long long ll;

using namespace std;

int tot=0;

char aim[100010][10010];

int num[100010];

int n,m;

 struct trie

{

    int next[500010][26],fail[500010],ed[500010];

    int root,L;

    int newnode()

    {

        for(int i = 0; i < 26; i++)

            next[L][i] = -1;

        ed[L++] = 0;

        return L-1;

    }

    void init()

    {

        L = 0;

        root = newnode();

    }

    void insert(char buf[])

    {

        int len = strlen(buf);

        int now = root;

        for(int i = 0; i < len; i++)

        {

            if(next[now][buf[i]-'a'] == -1)

                next[now][buf[i]-'a'] = newnode();

                now = next[now][buf[i]-'a'];

        }

        if(ed[now]==-1)

        {

            ed[now]=1;

        }

        else

        ed[now]++;

    }

    void build()

    {

        queue<int> q;

        fail[root] = root;

        for(int i = 0; i < 26; i++)

            if(next[root][i] == -1)

                next[root][i] = root;

            else

            {

                fail[next[root][i]] = root;

                q.push(next[root][i]);

            }

        while(!q.empty())

        {

            int now = q.front();

            q.pop();

            for(int i = 0; i < 26; i++)

                if(next[now][i] == -1)

                    next[now][i] = next[fail[now]][i];

                else

                {

                    fail[next[now][i]]=next[fail[now]][i];

                    q.push(next[now][i]);

                }

        }

    }

    int query(char buf[])

    {

        memset(num,0,sizeof(num));

        int len = strlen(buf);

        int now = root;

        int res = 0;

        for(int i = 0; i < len; i++)

        {

            now = next[now][buf[i]-'a'];

            int temp = now;

            while(temp != root)

            {

                if(ed[temp]!=-1)

                {

                    num[ed[temp]]+=ed[temp];

                    //res += ed[temp];

                }

                //tot++;

                //printf("%d====",res);

                //ed[temp] = 0;

                temp = fail[temp];

            }

        }

    return res;

    }

    void Debug()

    {

        for(int i = 0; i < L; i++)

        {

            printf("id = %3d,fail = %3d,end = %3d,chi = [",i,fail[i],ed[i]);

            for(int j = 0; j < 26; j++)

                printf("%2d",next[i][j]);

            printf("]\n");

        }

    }

};

char buf[1000010];

trie ac;

int main()

{

    int t;

    //int n;

    scanf("%d",&t);

    while(t--)

    {

        //int n,m;

        scanf("%d%d",&n,&m);

        ac.init();

        for(int i = 0; i < n; i++)

        {

            scanf("%s",aim[i]);

            //ac.insert(aim[i]);

        }

        for(int i=0;i<m;i++)

        {

        scanf("%s",buf);

        ac.insert(buf);

        //printf("%d\n",ac.query(buf,dex));

        //printf("%d\n",tot);

        }

        ac.build();

        for(int i=0;i<n;i++)

        {

            ac.query(aim[i]);

            int ans=0;

            for(int i=0;i<m;i++)

            {

                ans+=num[i];

            }

            printf("%d\n",ans);

        }

    }

    return 0;

}

### 4 字符串的最小表示法（没验证过）

int getmin(char s[],int len)

{

int i=0,j=1,k=0;

while(i<len&&j<len&&k<len)

{

int t=s[(i+k)%len]-s[(j+k)%len];

if (!t) k++;

else

{

if (t>0) i+=k+1;

else j+=k+1;

if (i==j) j++;

k=0;

}

}

return min(i,j);

}

### 5 二叉树

//给先序中序求位置

#include<stdio.h>

#include<string.h>

struct node {

    int l,r,val;

}tree[100005];

int tot=1;

void init(int tot)

{

    tree[tot].l=tree[tot].r=0;

}

void insert(int x,int val)

{

   if(tree[x].l&&val<tree[x].val)insert(tree[x].l,val);

   else if(tree[x].r&&val>tree[x].val)insert(tree[x].r,val);

   else {

       init(tot);

       tree[tot].val=val;

       if(tree[x].val>val)tree[x].l=tot;

       else tree[x].r=tot;

       tot++;

   }

}

void query(int x,int val)

{

    if(val==tree[x].val){printf("\n");return;}

    if(tree[x].val>val){

        printf("E");

        query(tree[x].l,val);

    }

    else {

        printf("W");

        query(tree[x].r,val);

    }

}

int main()

{

    int T,i,j,k,n,a[1005],b[1005],q,x;

    scanf("%d",&T);

    while(T--)

    {

         tot=1;

        scanf("%d",&n);

        for(i=1;i<=n;i++)

        {

            scanf("%d",&a[i]);

            if(i==1){

                init(tot);

                tree[tot].val=a[i];

                tot++;

            }

            else insert(1,a[i]);

        }

        scanf("%d",&q);

        while(q--)

        {

            scanf("%d",&x);

            query(1,x);

        }

    }

    return 0;

}

## 6 动态规划

### 1 最长上升子序列(LIS)

//导弹拦截代码

#include<bits/stdc++.h>

using namespace std;

int main()

{

    int n;

    cin>>n;

    int i,k=1,j,a[30]={0},b[30]={0};

    for(i=1;i<=n;i++)

        cin>>a[i];

    b[1]=a[1];

    for(i=2;i<=n;i++)

        if(a[i]<=b[k])

        {

            b[k+1]=a[i];

            k++;

        }

        else

        {

            int p[k];

            for(j=0;j<k;j++)

                p[j]=b[k-j];

            int m=lower\_bound(p,p+k,a[i])-p;

            b[k-m+1]=a[i];

        }

        cout<<k;

}

//求最长上升子序列

#include<bits/stdc++.h>

using namespace std;

int main()

{

    int n;

    cin>>n;

    int i,k=0,a[n],b[n];

    for(i=0;i<n;i++)

        cin>>a[i];

    b[0]=a[0];

    for(i=1;i<n;i++)

         if(a[i]>b[k])

            b[++k]=a[i];

        else

        {

            int m=lower\_bound(b,b+k+1,a[i])-b;

            if(b[m]!=a[i])

            b[m]=a[i];

        }

        for(i=0;i<k+1;i++)

            cout<<b[i]<<endl;

        cout<<k+1;

}

#include<bits/stdc++.h>

using namespace std;

int main()

{

    int n;

    cin>>n;

    int a[n],l[n],i,j;

    for(i=0; i<n; i++)

        cin>>a[i];

    for(i=0; i<n; i++)

    {

        l[i]=1;

        for(j=0; j<i; j++)

            if(a[i]>a[j])  
        l[i]=max(l[j]+1,l[i]);

    }

    int m=0;

    for(i=0; i<n; i++)

        if(l[i]>m)

            m=l[i];

    cout<<m;

    return 0;

}

### 2 01背包

#include<stdio.h>

#include<cstring>

#include<cmath>

#include<algorithm>

#include<iostream>

using namespace std;

int main()

{

    int bag[2000];

    int w[2000],h[2000];

    int n,m;

    int t;

    scanf("%d",&t);

    while(t--){

        scanf("%d%d",&n,&m);

          memset(bag,0,sizeof(bag));

    for(int i=0;i<n;i++)

        scanf("%d",&h[i]);

    for(int i=0;i<n;i++)

        scanf("%d",&w[i]);

    for(int i=0;i<n;i++)

        for(int k=m;k>=w[i];k--)//k代表能容下k kg的背包

        bag[k]=max(bag[k-w[i]]+h[i],bag[k]);

    printf("%d\n",bag[m]);

    }

    return 0;

}

#include<iostream>

#include<stdio.h>

#include<queue>

#include<cstring>

#include<string>

#include<cmath>

#include<algorithm>

#include<assert.h>

#include<stdlib.h>

#include<time.h>

#include<stack>

#include<vector>

#include<map>

#define pi acos(-1.0)

#define inf 1<<29

typedef long long ll;

using namespace std;

int main()

{

     int bag[2000];

    int w[2000],j[2000];

    int n,m;

    int t;

        scanf("%d%d",&m,&n);

          memset(bag,0,sizeof(bag));

    for(int i=0;i<n;i++)

        scanf("%d%d",&w[i],&j[i]);

    for(int i=0;i<n;i++)

        for(int k=m;k>=w[i];k--)//k代表能容下k kg的背包

        bag[k]=max(bag[k-w[i]]+j[i],bag[k]);

    printf("%d",bag[m]);

    return 0;

}

**J - FATE**

**Time Limit:**1000MS     **Memory Limit:**32768KB     **64bit IO Format:**%I64d & %I64u

Submit [Status](http://acm.hust.edu.cn/vjudge/contest/view.action?cid=99543#status//J/0) [Practice](http://acm.hust.edu.cn/vjudge/problem/viewProblem.action?id=20469) [HDU 2159](http://acm.hust.edu.cn/vjudge/problem/visitOriginUrl.action?id=20469)

**Description**

最近xhd正在玩一款叫做FATE的游戏，为了得到极品装备，xhd在不停的杀怪做任务。久而久之xhd开始对杀怪产生的厌恶感，但又不得不通过杀怪来升完这最后一级。现在的问题是，xhd升掉最后一级还需n的经验值，xhd还留有m的忍耐度，每杀一个怪xhd会得到相应的经验，并减掉相应的忍耐度。当忍耐度降到0或者0以下时，xhd就不会玩这游戏。xhd还说了他最多只杀s只怪。请问他能升掉这最后一级吗？

**Input**

输入数据有多组，对于每组数据第一行输入n，m，k，s(0 < n,m,k,s < 100)四个正整数。分别表示还需的经验值，保留的忍耐度，怪的种数和最多的杀怪数。接下来输入k行数据。每行数据输入两个正整数a，b(0 < a,b < 20)；分别表示杀掉一只这种怪xhd会得到的经验值和会减掉的忍耐度。(每种怪都有无数个)

**Output**

输出升完这级还能保留的最大忍耐度，如果无法升完这级输出-1。

**Sample Input**

10 10 1 10

1 1

10 10 1 9

1 1

9 10 2 10

1 1

2 2

**Sample Output**

0

-1

1

//二维背包

#include<stdio.h>

#include<string.h>

int dp[101][101];      //dp[i][j] 表示消耗i的忍耐度和杀j个怪物得到的最大经验值

struct node

{

    int e;     //经验值

    int r;     //忍耐度

}a[101];

int main()

{

    int n,m,k,s,i,j,t;

    while(scanf("%d%d%d%d",&n,&m,&k,&s)!=EOF)

    {

        for(i=1;i<=k;++i)

            scanf("%d%d",&a[i].e,&a[i].r);

        memset(dp,0,sizeof(dp));

        for(i=1;i<=k;++i)   //k表示怪物种类---对不同怪物遍历一遍

            for(j=a[i].r;j<=m;++j)  //m表示保留的忍耐度

                for(t=1;t<=s;++t)    // s表示杀的怪物数

                {

                    if(dp[j][t]<dp[j-a[i].r][t-1]+a[i].e)

                    {

                        dp[j][t]=dp[j-a[i].r][t-1]+a[i].e;

                    }

                }

        if(dp[m][s]>=n)     //表示能过升级

        {

                for(i=0;i<=m;++i)   //寻找能够升级所消耗的最小忍耐度,只用找消耗相同忍耐度的情况下，令杀怪数量最多，

                    if(dp[i][s]>=n) //那么d[i][s]一定是消耗i忍耐度的情况下，获得的最大经验值

                    {

                        printf("%d\n",m-i);

                        break;

                    }

        }

        else

            printf("-1\n");

    }

    return 0;

}

//自己的代码

#include<iostream>

#include<cstdio>

#include<queue>

#include<cstring>

#include<string>

#include<cmath>

#include<algorithm>

#include<assert.h>

#include<stdlib.h>

#include<time.h>

#include<stack>

#include<vector>

#include<map>

#define pi acos(-1.0)

#define inf 1<<29

#define bug puts("\*\*\*\*\*\*\*\*\*\*");

#define mx 1010

typedef long long ll;

using namespace std;

#define mod 1000000009

int main()

{

    int n,m,k,s;

    int dp[110][110],st[110],e[110];

    while(~scanf("%d%d%d%d",&n,&m,&k,&s))

    {

        memset(dp,0,sizeof(dp));

        memset(st,0,sizeof(st));

        memset(e,0,sizeof(e));

        int i,j,l;

        for(i=0;i<k;i++)

            scanf("%d%d",&e[i],&st[i]);

        for(i=0;i<k;i++)

            for(j=st[i];j<=m;j++)

            for(l=1;l<=s;l++)

            if(dp[j][l]<dp[j-st[i]][l-1]+e[i])

            dp[j][l]=dp[j-st[i]][l-1]+e[i];

        if(dp[m][s]>=n)

        {

            for(i=0;i<=m;i++)

                if(dp[i][s]>=n)

            {

                printf("%d\n",m-i);

                break;

            }

        }

        else

            printf("-1\n");

    }

    return 0;

}

### 3 数位dp（没验证过）

int dfs(int pos, int s, bool e) //e==1表示满了（当前pos之前的数位都相同）

{

if (pos==-1) return s==target\_s; //返回 当前状态==目标状态

if (~f[pos][s] && !e ) return f[pos][s]; //如果值已经算过并且没满则返回记录的值

int ans = 0;

int u = e ? bit[pos] : 9; //9可以相应改成进制

for (int i = 0; i <= u; ++i)

ans += dfs(pos-1, new\_s(s, i), e&&i==u); //new\_s表示新的状态。另外，“肯定不符合目标状态”的状态可以直接continue掉

return e ? ans : f[pos][s]=ans; //如果满了就直接返回，否则记录再返回

}

int solve(int n)

{

memset(f,-1,sizeof(f));

int pos=-1;

while(n)

{

bit[++pos]=n%10; //10为进制

n/=10;

}

return dfs(pos,0,1); //初始状态

}

int f[20][5];

int bit[20];

int dfs(int pos, int s, bool e)

{

if (pos==-1) return s==0||s==1;

if (~f[pos][s]&&!e) return f[pos][s];

int ans=0;

int u=e?bit[pos]:9;

for (int i=0; i<=u; ++i)

{

if(i==4||s==1&&i==2)

continue;

ans+=dfs(pos-1,i==6,e&&i==u);

}

return e?ans:f[pos][s]=ans;

}

int solve(int n)

{

memset(f,-1,sizeof(f));

int pos=-1;

while(n)

{

bit[++pos]=n%10;

n/=10;

}

return dfs(pos,0,1);

}

## 7 数论

### 1 判断素数

int isprime(int n)

{

    if (n==1)

        return 0;

    int x=sqrt(n\*1.0);

    for (int i=2; i<=x; i++)

        if (n%i==0)

            return 0;

    return 1;

}

### 2 素数打表

const int limit = 20000000;

bool visit[limit];

int prime[limit];

int primelen;

void getprim()

{

    primelen=0;

    visit[0]=visit[1]=true;

    for (int i = 2; i <= limit; ++i)

    {

        if (visit[i] == false)

        {

            prime[primelen++] = i;

        }

        for (int j = 0; ((j < primelen) && (i \* prime[j] <limit));  ++j)

        {

            visit[i \* prime[j]] = true;

            if (i % prime[j] == 0) break;

        }

    }

}

### 3 分解质因数

int a[1000000]= {},pcnt=0;

void pdec(int n)

{

    int x=sqrt(n);

    for (int i=1; prime[i]<=x; i++)

        if (n%prime[i]==0)

        {

            a[++pcnt]=prime[i];

            n/=prime[i];

            i--;

        }

    if (n!=1)

        a[++pcnt]=n;

}

### 4 快速幂和快速乘

//快速幂

typedef long long mytype;

const long long mod = 1e9+7;

mytype quickpow(mytype a,mytype n)

{

    mytype res=1;

    while(n)

    {

        if(n&1)

            res=res\*a%mod;

        a=a\*a%mod;

        n>>=1;

    }

    return res;

}

//快速幂加快速乘

typedef long long mytype;

const long long mod = 1e9+7;

mytype quickmult(mytype a,mytype b)

{

    mytype res=0;

    while(b)

    {

        if(b&1)

        {

            res+=a;

            if(res>=mod)res-=mod;

        }

        a+=a;

        b>>=1LL;

        if(a>=mod)a-=mod;

    }

    return res;

}

mytype quickpow(mytype a,mytype n)

{

    mytype res=1;

    while(n)

    {

        if(n&1)

            res=quickmult(res,a);

        a=quickmult(a,a);

        n>>=1;

    }

    return res;

}

//快速乘

typedef long long mytype;

const long long mod = 1e9+7;

mytype quickmult(mytype a,mytype b)

{

    mytype res=0;

    while(b)

    {

        if(b&1)

        {

            res+=a;

            if(res>=mod)res-=mod;

        }

        a+=a;

        b>>=1LL;

        if(a>=mod)a-=mod;

    }

    return res;

}

### 5 逆元

long long quickpow(long long a, long long b) {

    if (b < 0) return 0;

    long long ret = 1;

    a %= mod;

    while(b) {

        if (b & 1) ret = (ret \* a) % mod;

        b >>= 1;

        a = (a \* a) % mod;

    }

    return ret;

}

//费马小定理求逆元（M必须是质数）

long long inv(long long a) {

    return quickpow(a, mod - 2);

}

a/b)%Mod=c;    (b\*p)%Mod=1;    ==》   (a/b)\*(b\*p) %Mod=c;    ==》    (a\*p)%Mod=c;  其中p是逆元

const int mod = 1000000009;

新学的一个求逆元的方法：

inv[i] = ( MOD - MOD / i ) \* inv[MOD%i] % MOD

证明：

设t = MOD / i , k = MOD % i

则有 t \* i + k == 0 % MOD

有 -t \* i == k % MOD

两边同时除以ik得到

-t \* inv[k] == inv[i] % MOD

即

inv[i] == -MOD / i \* inv[MOD%i]

即

inv[i] == ( MOD - MOD / i) \* inv[MOD%i]

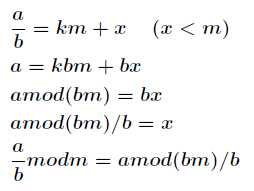
证毕

适用于MOD是质数的情况，能够O(n)时间求出1~n对模MOD的逆

### 6 数模计算公式

a / b mod x=a mod bx / b

证明



### 7 米勒罗宾素数测试

// 18位素数：154590409516822759

// 19位素数：2305843009213693951 (梅森素数)

// 19位素数：4384957924686954497

ll prime[6] = {2, 3, 5, 233, 331};

ll qmul(ll x, ll y, ll mod) { // 乘法防止溢出， 如果p \* p不爆LL的话可以直接乘； O(1)乘法或者转化成二进制加法

    return (x \* y - (long long)(x / (long double)mod \* y + 1e-3) \*mod + mod) % mod;

    /\*

    LL ret = 0;

    while(y) {

        if(y & 1)

            ret = (ret + x) % mod;

        x = x \* 2 % mod;

        y >>= 1;

    }

    return ret;

    \*/

}

ll qpow(ll a, ll n, ll mod) {

    ll ret = 1;

    while(n) {

        if(n & 1) ret = qmul(ret, a, mod);

        a = qmul(a, a, mod);

        n >>= 1;

    }

    return ret;

}

bool Miller\_Rabin(ll p) {

    if(p < 2) return 0;

    if(p != 2 && p % 2 == 0) return 0;

    ll s = p - 1;

    while(! (s & 1)) s >>= 1;

    for(int i = 0; i < 5; ++i) {

        if(p == prime[i]) return 1;

        ll t = s, m = qpow(prime[i], s, p);

        while(t != p - 1 && m != 1 && m != p - 1) {

            m = qmul(m, m, p);

            t <<= 1;

        }

        if(m != p - 1 && !(t & 1)) return 0;

    }

    return 1;

}

### 8 Lucas定理

long long mod=110119;

typedef long long mytype;

mytype quickmult(mytype a,mytype b)

{

    mytype res=0;

    while(b)

    {

        if(b&1)

        {

            res+=a;

            if(res>=mod)res-=mod;

        }

        a+=a;

        b>>=1LL;

        if(a>=mod)a-=mod;

    }

    return res;

}

long long PowMod(long long a,long long b)

{

    long long ret=1;

    while(b)

    {

        if(b&1)

            ret=quickmult(ret,a)%mod;

        a=quickmult(a,a)%mod;

        b>>=1;

    }

    return ret;

}

long long fac[1000005];

long long Get\_Fact(long long p)

{

    fac[0]=1;

    for(int i=1; i<=p; i++)

        fac[i]=quickmult(fac[i-1],i)%p;

}

long long Lucas(long long n,long long m,long long p)

{

    long long ret=1;

    while(m)

    {

        long long a=n%mod,b=m%mod;

        if(a<b) return 0;

        ret=(ret\*fac[a]\*PowMod(fac[b]\*fac[a-b]%p,p-2))%p;

        n/=p;

        m/=p;

    }

    return ret;

}

### 9 中国剩余定理

//不互质

void crt()

{

    int t;

    while(cin>>t)

    {

        int flag=1;

        long long n1,a1;

        if (t) scanf("%lld%lld",&n1,&a1),t--;

        while(t--)

        {

            long long n2,a2,k1,k2;

            scanf("%lld%lld",&n2,&a2);

            if (flag==0)

                continue;

            long long d=exgcd(n1,n2,k1,k2);

            if ((a2-a1)%d!=0)

                flag=0;

            if (flag)

            {

                k1=(k1\*(a2-a1)/d%(n2/d)+n2/d)%(n2/d);

                long long a=n1\*k1+a1;

                long long n=n1/d\*n2;

                n1=n;

                a1=a;

            }

        }

        if (flag) return a1;

        else return -1;

    }

}

//互质

ll China(ll r) {

    ll M = 1, ans = 0;

    for (ll i = 0; i < r; ++i)

        M \*= m[i];

    for(ll i = 0;i < r;i++) {

        ll N = M/m[i];

        ll x, y;

        extend\_gcd(N, m[i], x, y);

        x = (x%m[i]+m[i])%m[i];

        ans = ((ans+a[i]\*N%M\*x%M)%M + M)%M;

    }

    return ans;

}

int China(int r) {

int M = 1, ans = 0;

for (int i = 0; i < r; ++i)

M \*= m[i];

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

int N = M/m[i];

int x, y;

extend\_gcd(N, m[i], x, y);  
 x = (x%m[i]+m[i])%m[i];

ans = ((ans+a[i]\*N%M\*x%M)%M + M)%M;

}

return ans;

}

### 10 指数循环结

D:\Users\Dean Wang\AppData\Local\YNote\data\99893492@qq.com\10935343c54f437b93434053d8e944da\13151829390.jpeg

<http://blog.csdn.net/wust_zzwh/article/details/51966450>

#include<vector>

#include<iostream>

#include<stdio.h>

#include<queue>

#include<cstring>

#include<cmath>

#include<map>

#include <string>

#include<algorithm>

#include<set>

#include<stack>

#define debug puts("xxxxxxx");

#define pi (acos(-1.0))

#define eps (1e-5)

#define inf 0x3f3f3f3f

#define mem(x) memset(x,0,sizeof(x))

typedef long long ll;

const int mx = 10000005;

const int mod = 1e9+7;

using namespace std;

int pri[mx],phi[mx],tot;

ll sum[mx];

bool vis[mx];

void init()

{

    int n=mx;

    tot=0;

    memset(vis,false,sizeof vis);

    phi[1]=1;

    for(int i=2; i<n; i++)

    {

        if(!vis[i])

        {

            pri[tot++]=i;

            phi[i]=i-1;

        }

        for(int j=0; j<tot && i\*pri[j]<n; j++)

        {

            vis[i\*pri[j]]=true;

            if(i%pri[j]==0)

            {

                phi[i\*pri[j]]=phi[i]\*pri[j];

                break;

            }

            else phi[i\*pri[j]]=phi[i]\*(pri[j]-1);

        }

    }

    sum[0]=0;

    for(int i=1;i<mx;i++)

        sum[i]=(sum[i-1]+phi[i])%mod;

}

int rear;

int a[200];

void resolve(ll n)

{

    int i,j;

    rear=0;

    for(i=0;i<tot;i++)

    {

        if(!vis[n])

        {

            a[rear++]=n;

            break;

        }

        if(n%pri[i]==0)

        {

            a[rear++]=pri[i];

            n/=pri[i];

        }

    }

}

ll f(int pos,ll n,ll m)

{

//cout<<pos<<" "<<n<<' '<<m<<endl;

    if(n==1)

        return sum[m];

    if(m==0)return 0;

    if(m==1)

        return phi[n];

    ll p=a[pos];

    return ((p-1)\*f(pos-1,n/p,m)+f(pos,n,m/p))%mod;

}

//ll f(int pos,ll n,ll m)

//{

//    ll M=mod;

//

//    //pos即每个素数，一次一个就行了

//    if(n==1) return sum[m];//n为1结果就是欧拉值的前缀和

//    if(m==0)return 0;

//    return ((a[pos]-1)\*f(pos-1,n/a[pos],m)%M+f(pos,n,m/a[pos]))%M;

//}

typedef long long mytype;

mytype quickpow(mytype a,mytype n,mytype mmod)

{

    mytype res=1;

    while(n)

    {

        if(n&1)

            res=res\*a%mmod;

        a=a\*a%mmod;

        n>>=1;

    }

    if(res==0) res+=mmod;

    return res;

}

ll solve(ll k,ll q)

{

    if(q==1)

        return q;

    ll num=solve(k,phi[q]);

    return quickpow(k,num,q);

}

int main()

{

//    freopen("1006.in","r",stdin);

//    freopen("1006my.out","w",stdout);

    init();

    ll n,m,q;

    while(~scanf("%I64d%I64d%I64d",&n,&m,&q))

    {

        resolve(n);

        ll k=f(rear-1,n,m);

        ll ans=solve(k,q);

        printf("%I64d\n",ans%q);

    }

}

### 11欧拉函数

欧拉函数是指：对于一个正整数n，小于n且和n互质的正整数（包括1）的个数，记作φ(n) 。

通式：φ(x)=x\*(1-1/p1)\*(1-1/p2)\*(1-1/p3)\*(1-1/p4)…..(1-1/pn),其中p1, p2……pn为x的所有质因数，x是不为0的整数。φ(1)=1（唯一和1互质的数就是1本身）。

对于质数p，φ(p) = p - 1。注意φ(1)=1.

欧拉定理：对于互质的正整数a和n，有aφ(n) ≡ 1 mod n。

欧拉函数是[积性函数](http://baike.baidu.com/view/2046973.htm)——若m,n互质，φ(mn)=φ(m)φ(n)。

                                 若n是质数p的k次幂，φ(n)=p^k-p^(k-1)=(p-1)p^(k-1)，因为除了p的倍数外，其他数都跟n互质。

特殊性质：当n为奇数时，φ(2n)=φ(n)

欧拉函数还有这样的性质：

**设a为N的质因数，若(N % a == 0 && (N / a) % a == 0) 则有E(N)=E(N / a) \* a；若(N % a == 0 && (N / a) % a != 0) 则有：E(N) = E(N / a) \* (a - 1)。**

通式：

D:\Users\Dean Wang\AppData\Local\YNote\data\99893492@qq.com\8f1963852e8e4833a1fc62f79b4f668a\d72a6059a7a7.png

,其中p1, p2……pn为x的所有质因数，x是不为0的整数。φ(1)=1（唯一和1[互质](http://baike.baidu.com/view/731400.htm)的数(小于等于1)就是1本身）。 (注意：每种质因数只一个。比如12=2\*2\*3那么φ（12）=12\*（1-1/2）\*(1-1/3)=4

若n是质数p的k次幂，

D:\Users\Dean Wang\AppData\Local\YNote\data\99893492@qq.com\743924566f0941d9ac402a2f12cd32e5\b31c8701e4b1.png

，因为除了p的倍数外，其他数都跟n互质。

设n为正整数，以 φ(n)表示不超过n且与n互

素的正整数的个数，称为n的欧拉函数值，这里函数

φ：N→N，n→φ(n)称为欧拉函数。

欧拉函数是[积性函数](http://baike.baidu.com/view/2046973.htm)——若m,n互质，

D:\Users\Dean Wang\AppData\Local\YNote\data\99893492@qq.com\a09075f6cf9d41f5af5ac32bf59848c5\4134970a5a1c.png

特殊性质：当n为奇数时，

D:\Users\Dean Wang\AppData\Local\YNote\data\99893492@qq.com\0fe63ebd913a44ce96c7e6ab3a679e77\d109b3de492e.png

, 证明与上述类似。

若n为质数则

D:\Users\Dean Wang\AppData\Local\YNote\data\99893492@qq.com\a33063665edc45afa801a1048cd2e31c\51da80cb3ece.png

∑*d*|*nφ*(*d*)=*n*

欧拉函数

long long phi(long long n)

{

    long long ans=n;

    long long x=sqrt(n);

    for (long long i=2; i<=x; i++)

    {

        if (n%i==0)

        {

            while(n%i==0)

                n/=i;

            ans=ans/i\*(i-1);

        }

    }

    if (n>1)

        ans=ans/n\*(n-1);

    return ans;

}

欧拉函数打表

const int MAXN=10005;

long long phi[MAXN];

void getphi()

{

    for (int i=1; i<MAXN; i++)

        phi[i]=i;

    for (int i=2; i<MAXN; i++)

        if (phi[i]==i)

            for (int j=i; j<MAXN; j+=i)

                phi[j]=phi[j]/i\*(i-1);

}

筛选素数的过程中求出欧拉函数：

int pri[mx],phi[mx],tot;

bool vis[mx];

void init()

{

    int n=mx;

    tot=0;

    memset(vis,false,sizeof vis);

    phi[1]=1;

    for(int i=2; i<n; i++)

    {

        if(!vis[i])

        {

            pri[tot++]=i;

            phi[i]=i-1;

        }

        for(int j=0; j<tot && i\*pri[j]<n; j++)

        {

            vis[i\*pri[j]]=true;

            if(i%pri[j]==0)

            {

                phi[i\*pri[j]]=phi[i]\*pri[j];

                break;

            }

            else phi[i\*pri[j]]=phi[i]\*(pri[j]-1);

        }

    }

}

### 12 欧拉常数

这是一个有名的调和级数：  
当n->∞，1+1/2+1/3+1/4+1/5+...+1/n->∞，是个发散级数，而极限却是收敛的  
当n很大时，有个近似公式：1+1/2+1/3+1/4+1/5+...+1/n=γ+ln(n)   
γ是欧拉常数，γ=0.57721566490153286060651209...

### 13 fft

匡斌模板

const double PI = acos(-1.0);

//复数结构体

struct Complex

{

    double x,y;//实部和虚部  x+yi

    Complex(double \_x = 0.0,double \_y = 0.0)

    {

        x = \_x;

        y = \_y;

    }

    Complex operator -(const Complex &b)const

    {

        return Complex(x-b.x,y-b.y);

    }

    Complex operator +(const Complex &b)const

    {

        return Complex(x+b.x,y+b.y);

    }

    Complex operator \*(const Complex &b)const

    {

        return Complex(x\*b.x-y\*b.y,x\*b.y+y\*b.x);

    }

};

/\*

\*    进行FFT和IFFT前的反转变换。

\*    位置i和 （i二进制反转后位置）互换

\*    len必须去2的幂

 \*/

void change(Complex y[],int len)

{

    int i,j,k;

    for(i = 1, j = len/2; i <len-1; i++)

    {

        if(i < j)swap(y[i],y[j]);

        //交换互为小标反转的元素，i<j保证交换一次         //i做正常的+1，j左反转类型的+1,始终保持i和j是反转的

        k = len/2;

        while(j >= k)

        {

            j -= k;

            k /= 2;

        }

        if(j < k)j += k;

    }

}

/\*

\*    做FFT

\*    len必须为2^k形式，

\*    on==1时是DFT，on==-1时是IDFT

 \*/ void fft(Complex y[],int len,int on)

{

    change(y,len);

    for(int h = 2; h <= len; h <<= 1)

    {

        Complex wn(cos(-on\*2\*PI/h),sin(-on\*2\*PI/h));

        for(int j = 0; j < len; j+=h)

        {

            Complex w(1,0);

            for(int k = j; k < j+h/2; k++)

            {

                Complex u = y[k];

                Complex t = w\*y[k+h/2];

                y[k] = u+t;

                y[k+h/2] = u-t;

                w = w\*wn;

            }

        }

    }

    if(on == -1)

        for(int i = 0; i < len; i++)

            y[i].x /= len;

}

A\*B

#include <stdio.h>

#include <string.h>

#include <iostream>

#include <algorithm>

#include <math.h>

using namespace std;

const double PI = acos(-1.0);

//复数结构体

struct Complex

{

    double x,y;//实部和虚部  x+yi

    Complex(double \_x = 0.0,double \_y = 0.0)

    {

        x = \_x;

        y = \_y;

    }

    Complex operator -(const Complex &b)const

    {

        return Complex(x-b.x,y-b.y);

    }

    Complex operator +(const Complex &b)const

    {

        return Complex(x+b.x,y+b.y);

    }

    Complex operator \*(const Complex &b)const

    {

        return Complex(x\*b.x-y\*b.y,x\*b.y+y\*b.x);

    }

};

/\*

\*    进行FFT和IFFT前的反转变换。

\*    位置i和 （i二进制反转后位置）互换

\*    len必须去2的幂

 \*/ void change(Complex y[],int len)

{

    int i,j,k;

    for(i = 1, j = len/2; i <len-1; i++)

    {

        if(i < j)swap(y[i],y[j]);

        //交换互为小标反转的元素，i<j保证交换一次         //i做正常的+1，j左反转类型的+1,始终保持i和j是反转的

        k = len/2;

        while(j >= k)

        {

            j -= k;

            k /= 2;

        }

        if(j < k)j += k;

    }

}

/\*

\*    做FFT

\*    len必须为2^k形式，

\*    on==1时是DFT，on==-1时是IDFT

 \*/ void fft(Complex y[],int len,int on)

{

    change(y,len);

    for(int h = 2; h <= len; h <<= 1)

    {

        Complex wn(cos(-on\*2\*PI/h),sin(-on\*2\*PI/h));

        for(int j = 0; j < len; j+=h)

        {

            Complex w(1,0);

            for(int k = j; k < j+h/2; k++)

            {

                Complex u = y[k];

                Complex t = w\*y[k+h/2];

                y[k] = u+t;

                y[k+h/2] = u-t;

                w = w\*wn;

            }

        }

    }

    if(on == -1)         for(int i = 0; i < len; i++)             y[i].x /= len;

}

const int MAXN = 200010;

Complex x1[MAXN],x2[MAXN];

char str1[MAXN/2],str2[MAXN/2];

int sum[MAXN];

int main()

{

    while(scanf("%s%s",str1,str2)==2)

    {

        int len1 = strlen(str1);

        int len2 = strlen(str2);

        int len = 1;

        while(len < len1\*2 || len < len2\*2)len<<=1;

        for(int i = 0; i < len1; i++)             x1[i] = Complex(str1[len1-1-i]-'0',0);

        for(int i = len1; i < len; i++)             x1[i] = Complex(0,0);

        for(int i = 0; i < len2; i++)             x2[i] = Complex(str2[len2-1-i]-'0',0);

        for(int i = len2; i < len; i++)             x2[i] = Complex(0,0);

        //求DFT

        fft(x1,len,1);

        fft(x2,len,1);

        for(int i = 0; i < len; i++)             x1[i] = x1[i]\*x2[i];

        fft(x1,len,-1);

        for(int i = 0; i < len; i++)             sum[i] = (int)(x1[i].x+0.5);

        for(int i = 0; i < len; i++)

        {

            sum[i+1]+=sum[i]/10;

            sum[i]%=10;

        }

        len = len1+len2-1;

        while(sum[len] <= 0 && len > 0)len--;

        for(int i = len; i >= 0; i--)             printf("%c",sum[i]+'0');

        printf("\n");

    }

    return 0;

}

### 14 雷德算法和倒位序

void rader(complex \*F,int len)

{

    int j=len/2;///模拟二进制反转进位的的位置

    for(int i=1; i<len-1; i++)

    {

        if(i<j)swap(F[i],F[j]);///该出手时就出手

        int k=len/2;

        while(j>=k)

        {

            j-=k;

            k>>=1;

        }

        if(j<k)j+=k;

    }

### 15 原根及快速求原根

设m是正整数，a是整数，若a模m的阶等于φ(m)，则称a为模m的一个原根。

求原根目前的做法只能是从2开始枚举，然后暴力判断g^(P-1) = 1 (mod P)是否当且当指数为P-1的时候成立

快速求原根

#include <iostream>

#include <string.h>

#include <algorithm>

#include <stdio.h>

using namespace std;

#define LL long long

const int N = 1000010;

bool prime[N];

LL p[N];

LL pr[N];

LL k=0,c;

void isprime()

{

    LL i,j;

    memset(prime,true,sizeof(prime));

    for(i=2; i<N; i++)

    {

        if(prime[i])

        {

            p[k++]=i;

            for(j=i+i; j<N; j+=i)

            {

                prime[j]=false;

            }

        }

    }

}

void cal(LL n)

{

    LL t=n,i,a;

    c=0;

    for(i=0; p[i]\*p[i]<=n; i++)

    {

        if(n%p[i]==0)

        {

            pr[c]=p[i];

            while(n%p[i]==0) n/=p[i];

            c++;

        }

    }

    if(n>1)

    {

        pr[c]=n;

        c++;

    }

}

LL quick\_mod(LL a,LL b,LL m)

{

    LL ans=1;

    a%=m;

    while(b)

    {

        if(b&1)

        {

            ans=ans\*a%m;

            b--;

        }

        b>>=1;

        a=a\*a%m;

    }

    return ans;

}

int main()

{

    LL P,i,t,g,root;

    isprime();

    while(cin>>P)

    {

        cal(P-1);

        for(g=2; g<P; g++)

        {

            bool flag=true;

            for(i=0; i<c; i++)

            {

                t=(P-1)/pr[i];

                if(quick\_mod(g,t,P)==1)

                {

                    flag=false;

                    break;

                }

            }

            if(flag)

            {

                root=g;

                cout<<root<<endl;

            }

        }

    }

    return 0;

}

<http://blog.csdn.net/z690933166/article/details/11620267>

设m是正整数，a是整数，若a模m的阶等于φ(m)，则称a为模m的一个原根。（其中φ(m)表示m的欧拉函数）

给出1个质数P，找出P最小的原根。

**Input**

输入1个质数P(3 <= P <= 10^9)

**Output**

输出P最小的原根。

#include <cstdio>

#include <iostream>

#include <cstring>

using namespace std;

const int maxn = 100000+10;

typedef long long LL;

bool prime[maxn];

int prim[maxn],k;

void make\_prime()

{

    memset(prime,0,sizeof(prime));

    k=0;

    for(int i=2; i<maxn; i++)

        if(!prime[i])

        {

            prim[k++]=i;

            for(int j=0; j<k&&prim[j]<maxn/i; j++)

            {

                prime[i\*prim[j]]=1;

                if(i%prim[j]==0) break;

            }

        }

}

int fac[50],num;

void divide(LL n)

{

    num=0;

    for(int i=0; prim[i]\*prim[i]<=n; i++)

        if(n%prim[i]==0)

        {

            fac[num++]=prim[i];

            while(n%prim[i]==0) n/=prim[i];

        }

    if(n>1) fac[num++]=n;

}

LL pow\_mod(LL a,LL b,LL mod)

{

    LL r=1;

    while(b)

    {

        if(b&1)

            r=r\*a%mod;

        a=a\*a%mod;

        b>>=1;

    }

    return r;

}

int main()

{

    LL p;

    make\_prime();

    while(cin>>p)

    {

        divide(p-1);

        for(int i=2; i<p; i++)

        {

            bool flag=true;

            for(int j=0; j<num; j++)

            {

                int x=(p-1)/fac[j];

                if(pow\_mod(i,x,p)==1)

                {

                    flag=false;

                    break;

                }

            }

            if(flag)

            {

                printf("%d\n",i);

                break;

            }

        }

    }

    return 0;

}

### 16 本福特定律

本福特定律说明在b进位制中，以数n起头的数出现的机率为logb(n + 1) − logb(n) .本福特定律不但适用于个位数字，连多位的数也可用。

## 8 图论

### 1 拓扑排序

#include <iostream>

#include <stdio.h>

#include <cstring>

#include <cmath>

#include <queue>

#include <algorithm>

#include <vector>

#define debug puts("----------")

//#define INF 0x3f3f3f3f

#define maxn 300005

using namespace std;

priority\_queue<int> q;      //优先队列

vector<int> vec[maxn];

int b[maxn],num[maxn];

int m,n;

void init()

{

    memset(num,0,sizeof(num));

    for(int i = 1 ; i <= n ; i++)

        vec[i].clear();

}

int main()

{

    int t,x,y;

    while(~scanf("%d%d",&n,&m))

    {

        init();

        int flag = 0;

        for(int i = 0 ; i < m ; i++)

        {

            scanf("%d%d",&x,&y);

            num[x]++;

            vec[y].push\_back(x);

        }

        for(int i = 1 ; i <= n ; i++)

        {

            if(num[i] == 0) q.push(i);

        }

        int temp  =  n;

        while(!q.empty())

        {

            temp--;

            int u = q.top();

            b[temp] = u;

            q.pop();

            for(int i = 0 ; i < vec[u].size() ; i++)

            {

                int v = vec[u][i];

                num[v]--;

                if(!num[v]) q.push(v);

            }

        }

        for(int i = 1 ; i <= n ; i++)

            if(num[i])

        {

            flag = 1;

            break;

        }

        if(flag) cout <<"NO"<<endl;

        else cout << "YES"<<endl;

    }

    return 0;

}

### 2 最小生成树

typedef int mytype;

const int NV=1005;

int pre[NV],vis[NV];

mytype dis[NV],g[NV][NV];

void init(int n,int m,int s)

{

    memset(pre,0,sizeof(pre));

    memset(vis,0,sizeof(vis));

    for (int i=0; i<=n; i++)

        dis[i]=inf;

    dis[s]=0;

    for (int i=1; i<=n; i++)

        for (int j=1; j<=n; j++)

            g[i][j]=inf;

    for (int i=1; i<=m; i++)

    {

        int u,v,l;

        scanf("%d%d%d",&u,&v,&l);

        g[u][v]=l;

        g[v][u]=l;

    }

}

mytype prim(int n)

{

    mytype ans=0;

    for (int i=1; i<=n; i++)

    {

        int u=0;

        for (int j=1; j<=n; j++)

            if (!vis[j]&&dis[j]<dis[u])

                u=j;

        vis[u]=1;

        ans+=dis[u];

        for (int j=1; j<=n; j++)

            if (!vis[j]&&g[u][j]<dis[j])

            {

                dis[j]=g[u][j];

                pre[j]=u;

            }

    }

    return ans;

}

bool judge(int n)

{

    int cnt=0;

    for (int i=1; i<=n; i++)

        cnt+=vis[i];

    return cnt==n;

}

typedef int mytype;

const int NV=105;

const int NE=10005;

struct edge

{

    int u,v;

    mytype l;

    bool operator<(const edge e) const

    {

        return l<e.l;

    }

} E[NE];

int f[NV],rk[NV];

int finds(int x)

{

    return f[x]==x?x:f[x]=finds(f[x]);

}

void uni(int a,int b)

{

    a=finds(a);

    b=finds(b);

    if (a==b) return;

    if (rk[a]>rk[b]) f[b]=a;

    else

    {

        if (rk[a]==rk[b]) rk[b]++;

        f[a]=b;

    }

}

void init(int n,int m)

{

    memset(rk,0,sizeof(rk));

    for (int i=1; i<=n; i++)

        f[i]=i;

    for (int i=1; i<=m; i++)

        scanf("%d%d%d",&E[i].u,&E[i].v,&E[i].l);

}

mytype kruskal(int n,int m)

{

    sort(E+1,E+m+1);

    mytype ans=0;

    for (int i=1; i<=m; i++)

        if (finds(E[i].u)!=finds(E[i].v))

        {

            uni(E[i].u,E[i].v);

            ans+=E[i].l;

        }

    return ans;

}

bool judge(int n)

{

    int flag=0;

    for (int i=1; i<=n; i++)

        if (finds(i)==i)

            flag++;

    return flag==1;

}

### 3 dijk

dijk

#include<bits/stdc++.h>

using namespace std;

#define INF 1<<29

#define debug puts("\*\*\*\*\*\*\*\*\*\*\n")

#define mx 1005

int mapp[mx][mx];

int dis[mx],mark[mx];

void init()

{

    int i,j;

    for(i=0;i<mx;i++)

        for(j=0;j<mx;j++)

        mapp[i][j]=INF;

    memset(mark,0,sizeof(0));

}

void dijk(int u,int n)

{

    int m,i,j,min,v;

    memset(mark,0,mx);

    for(i=1;i<=n;i++)

        dis[i]=mapp[u][i];

    mark[u]=1;

    dis[u]=0;

    while(1)

    {

        min=INF;

        for(i=1;i<=n;i++)

            if(!mark[i]&&min>dis[i])

            min=dis[i],v=i;

        if(min==INF)

            break;

        mark[v]=1;

        for(i=1;i<=n;i++)

            if(!mark[i]&&dis[v]+mapp[v][i]<dis[i])

            dis[i]=dis[v]+mapp[v][i];

    }

}

int main()

{

    int n,m;

    while(~scanf("%d%d",&n,&m))

    {

        int i,j,a,b,c;

        init();

        for(i=0;i<m;i++)

        {

            scanf("%d%d%d",&a,&b,&c);

            mapp[a][b]=mapp[b][a]=c;

        }

        dijk(1,n);

        printf("%d\n",dis[n]);

    }

}

### 4 SPFA

typedef int mytype;

const int NV=105;

const int NE=10005\*2;

mytype dis[NV];

int pre[NV],vis[NV],vcnt[NV],he[NV],ecnt,flag;

struct edge

{

    int v,next;

    mytype l;

} E[NE];

void adde(int u,int v,mytype l)

{

    E[++ecnt].v=v;

    E[ecnt].l=l;

    E[ecnt].next=he[u];

    he[u]=ecnt;

}

void init(int n,int m,int s)

{

    ecnt=0;

    memset(pre,0,sizeof(pre));

    memset(vis,0,sizeof(vis));

    memset(vcnt,0,sizeof(vcnt));

    memset(he,-1,sizeof(he));

    for (int i=0; i<=n; i++)

        dis[i]=inf;

    dis[s]=0;

    for (int i=1; i<=m; i++)

    {

        int u,v;

        mytype l;

        scanf("%d%d%d",&u,&v,&l);

        adde(u,v,l);

        adde(v,u,l);

    }

}

void spfa(int n,int m,int s)

{

    queue<int> q;

    vis[s]=1;

    q.push(s);

    flag=0;

    while(!q.empty())

    {

        int u=q.front();

        q.pop();

        vis[u]=0;

        if(vcnt[u]>=n)

        {

            flag=1;

            break;

        }

        for (int i=he[u]; i!=-1; i=E[i].next)

            if (dis[u]+E[i].l<dis[E[i].v])

            {

                dis[E[i].v]=dis[u]+E[i].l;

                pre[E[i].v]=u;

                if (!vis[E[i].v])

                {

                    vis[E[i].v]=1;

                    q.push(E[i].v);

                    vcnt[E[i].v]++;

                }

            }

    }

}

### 5网络流

网络流（最大流）

#include<stdio.h>  
#include<cstring>  
#include<algorithm>  
#include<iostream>  
#include<cmath>  
#include<vector>  
using namespace std;  
#define mem(x,y) memset(x,y,sizeof(x))  
#define inf 0x3f3f3f3f  
#define debug puts("-----")  
#define maxn 10000+5  
#define ll long long  
#define LL long long  
#define NE 3000  
#define mx 10005  
#define ep 1e-2  
#define pi acos(-1.0)    
#define mod 1000000007   
///===================================////  
#define INF 0x3f3f3f3f  
const int maxm = 120900;  
int n;  
struct Node {  
    int v;    // vertex  
    int cap;    // capacity  
    int flow;   // current flow in this arc  
    int nxt;  
} e[maxm \* 2];  
int g[maxn], fcnt;  
int st, ed;  
void add\_(int u, int v, int c) {  
    e[++fcnt].v = v;  
    e[fcnt].cap = c;  
    e[fcnt].flow = 0;  
    e[fcnt].nxt = g[u];  
    g[u] = fcnt;  
    e[++fcnt].v = u;  
    e[fcnt].cap = 0;  
    e[fcnt].flow = 0;  
    e[fcnt].nxt = g[v];  
    g[v] = fcnt;  
}  
void init\_maxflow(int src,int sink,int n\_) {  
    memset(g, 0, sizeof(g));  
    fcnt = 1;  
n=n\_;    //n为包括源点和汇点的总点个数  
    st = src, ed = sink;/\*修改\*/  
    //n = n + 3; //这里可以将n适当扩大一些  
/\*什么JB玩意儿\*/  
}  
int dist[maxn], numbs[maxn], q[maxn];  
void rev\_bfs() {  
    int font = 0, rear = 1;  
    for (int i = 0; i <= n; i++) { //n为总点数  
        dist[i] = maxn;  
        numbs[i] = 0;  
    }  
    q[font] = ed;  
    dist[ed] = 0;  
    numbs[0] = 1;  
    while(font != rear) {  
        int u = q[font++];  
        for (int i = g[u]; i; i = e[i].nxt) {  
            if (e[i ^ 1].cap == 0 || dist[e[i].v] < maxn) continue;  
            dist[e[i].v] = dist[u] + 1;  
            ++numbs[dist[e[i].v]];  
            q[rear++] = e[i].v;  
        }  
    }  
}  
int maxflow() {  
    rev\_bfs();  
    int u, totalflow = 0;  
    int curg[maxn], revpath[maxn];  
    for(int i = 0; i <= n; ++i) curg[i] = g[i];  
    u = st;  
    while(dist[st] < n) {  
        if(u == ed) {   // find an augmenting path  
            int augflow = INF;  
            for(int i = st; i != ed; i = e[curg[i]].v)  
                augflow = min(augflow, e[curg[i]].cap);  
            for(int i = st; i != ed; i = e[curg[i]].v) {  
                e[curg[i]].cap -= augflow;  
                e[curg[i] ^ 1].cap += augflow;  
                e[curg[i]].flow += augflow;  
                e[curg[i] ^ 1].flow -= augflow;  
            }  
            totalflow += augflow;  
            u = st;  
        }  
        int i;  
        for(i = curg[u]; i; i = e[i].nxt)  
            if(e[i].cap > 0 && dist[u] == dist[e[i].v] + 1) break;  
        if(i) {   // find an admissible arc, then Advance  
            curg[u] = i;  
            revpath[e[i].v] = i ^ 1;  
            u = e[i].v;  
        } else {    // no admissible arc, then relabel this vertex  
            if(0 == (--numbs[dist[u]])) break;    // GAP cut, Important!  
            curg[u] = g[u];  
            int mindist = n;  
            for(int j = g[u]; j; j = e[j].nxt)  
                if(e[j].cap > 0) mindist = min(mindist, dist[e[j].v]);  
            dist[u] = mindist + 1;  
            ++numbs[dist[u]];  
            if(u != st)  
                u = e[revpath[u]].v;    // Backtrack  
        }  
    }  
    return totalflow;  
}  
///================分割线=============///  
//最大流的输出路径  
int out[maxm][3], tot = 0;  
void get\_out() {  
    for (int u = 1; u < ed; u++) {  
        for (int i = g[u]; i; i = e[i].nxt) {  
            if (e[i].v != ed && e[i].flow > 0) {  
                out[tot][0] = u;  
                out[tot][1] = e[i].v;  
                out[tot++][2] = e[i].flow;  
            }  
        }  
    }  
}  
  
//n为包括源点和汇点的总点个数

网络流（最小费用最大流）

#include<iostream>  
#include<cstring>  
#include<cstdio>  
#include<queue>  
#define inf (1<<30)  
using namespace std;  
int st,en;  
struct Edge  
{  
    int u,v,w,next;  
    int cost;  
}e[50000];  
int ecnt,pre[250];  
void adde(int u,int v,int w,int cost)  
{  
    e[ecnt].u=u;  
    e[ecnt].v=v;  
    e[ecnt].w=w;  
    e[ecnt].cost=cost;  
    e[ecnt].next=pre[u];  
    pre[u]=ecnt++;  
    e[ecnt].u=v;  
    e[ecnt].v=u;  
    e[ecnt].w=0;  
    e[ecnt].cost=-cost;  
    e[ecnt].next=pre[v];  
    pre[v]=ecnt++;  
}  
int f,v;  
void init()  
{  
    ecnt=0;  
    memset(pre,-1,sizeof(pre));  
    st=0,en=f+v+1;  
}  
bool vis[250];  
int p[250];  
int dis[250];  
bool spfa()  
{  
    queue<int>que;  
    memset(p,-1,sizeof(p));  
    memset(vis,0,sizeof(vis));  
    for(int i=st; i<=en; i++)  
    {  
        dis[i]=inf;  
    }  
    dis[st]=0;  
    vis[st]=1;  
    que.push(st);  
    while(!que.empty())  
    {  
        int u=que.front();  
        que.pop();  
        vis[u]=0;  
        for(int i=pre[u]; i!=-1; i=e[i].next)  
        {  
            int v=e[i].v;  
            if(e[i].w>0&&dis[v]>dis[u]+e[i].cost)  
            {  
                dis[v]=dis[u]+e[i].cost;  
                p[v]=i;  
                if(vis[v]==0)  
                {  
                    que.push(v);  
                    vis[v]=1;  
                }  
            }  
        }  
    }  
    if(dis[en]==inf)  
        return false;  
    return true;  
}  
int MCMF()  
{  
    int flow=0,mincost=0;  
    int minflow;  
    while(spfa())  
    {  
        minflow=inf;  
        for(int i=p[en]; i!=-1; i=p[e[i].u])  
        {  
            minflow=min(minflow,e[i].w);  
        }  
        for(int i=p[en]; i!=-1; i=p[e[i].u])  
        {  
            e[i].w-=minflow;  
            e[i^1].w+=minflow;  
        }  
        flow+=minflow;  
        mincost+=minflow\*dis[en];  
    }  
    return mincost;  
}  
int main()  
{  
    cin>>f>>v;  
    init();  
    int a;  
    for(int i=1; i<=f; i++)  
    {  
        for(int j=1; j<=v; j++)  
        {  
            scanf("%d",&a);  
            adde(i,f+j,1,-a);  
        }  
    }  
    for(int i=1; i<=f; i++)  
    {  
        adde(st,i,1,0);  
    }  
    for(int i=1; i<=v; i++)  
    {  
        adde(f+i,en,1,0);  
    }  
    int ans=MCMF();  
    cout<<-ans;  
    return 0;  
}  
  
注意其中的ecnt必须从0开始

## 9 数学

### 1 高斯消元

#include<stdio.h>

#include<algorithm>

#include<iostream>

#include<string.h>

#include<math.h>

using namespace std;

const int MAXN=50;

int a[MAXN][MAXN];//增广矩阵

int x[MAXN];//解集

bool free\_x[MAXN];//标记是否是不确定的变元

/\*

void Debug(void)

{

    int i, j;

    for (i = 0; i < equ; i++)

    {

        for (j = 0; j < var + 1; j++)

        {

            cout << a[i][j] << " ";

        }

        cout << endl;

    }

    cout << endl;

}

\*/

inline int gcd(int a,int b)

{

    int t;

    while(b!=0)

    {

        t=b;

        b=a%b;

        a=t;

    }

    return a;

}

inline int lcm(int a,int b)

{

    return a/gcd(a,b)\*b;//先除后乘防溢出

}

// 高斯消元法解方程组(Gauss-Jordan elimination).(-2表示有浮点数解，但无整数解，

//-1表示无解，0表示唯一解，大于0表示无穷解，并返回自由变元的个数)

//有equ个方程，var个变元。增广矩阵行数为equ,分别为0到equ-1,列数为var+1,分别为0到var.

int Gauss(int equ,int var)

{

    int i,j,k;

    int max\_r;// 当前这列绝对值最大的行.

    int col;//当前处理的列

    int ta,tb;

    int LCM;

    int temp;

    int free\_x\_num;

    int free\_index;

    for(int i=0; i<=var; i++)

    {

        x[i]=0;

        free\_x[i]=true;

    }

    //转换为阶梯阵.

    col=0; // 当前处理的列

    for(k = 0; k < equ && col < var; k++,col++)

    {

        // 枚举当前处理的行.

// 找到该col列元素绝对值最大的那行与第k行交换.(为了在除法时减小误差)

        max\_r=k;

        for(i=k+1; i<equ; i++)

        {

            if(abs(a[i][col])>abs(a[max\_r][col])) max\_r=i;

        }

        if(max\_r!=k)

        {

            // 与第k行交换.

            for(j=k; j<var+1; j++) swap(a[k][j],a[max\_r][j]);

        }

        if(a[k][col]==0)

        {

            // 说明该col列第k行以下全是0了，则处理当前行的下一列.

            k--;

            free\_x[free\_num++] = col;//这个是自由变元

            continue;

        }

        for(i=k+1; i<equ; i++)

        {

            // 枚举要删去的行.

            if(a[i][col]!=0)

            {

                LCM = lcm(abs(a[i][col]),abs(a[k][col]));

                ta = LCM/abs(a[i][col]);

                tb = LCM/abs(a[k][col]);

                if(a[i][col]\*a[k][col]<0)tb=-tb;//异号的情况是相加

                for(j=col; j<var+1; j++)

                {

                    a[i][j] = a[i][j]\*ta-a[k][j]\*tb;

                }

            }

        }

    }

    //  Debug();

    // 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).

    for (i = k; i < equ; i++)

    {

        // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.

        if (a[i][col] != 0) return -1;

    }

    // 2. 无穷解的情况: 在var \* (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行，即说明没有形成严格的上三角阵.

    // 且出现的行数即为自由变元的个数.

    if (k < var)

    {

        // 首先，自由变元有var - k个，即不确定的变元至少有var - k个.

        for (i = k - 1; i >= 0; i--)

        {

            // 第i行一定不会是(0, 0, ..., 0)的情况，因为这样的行是在第k行到第equ行.

            // 同样，第i行一定不会是(0, 0, ..., a), a != 0的情况，这样的无解的.

            free\_x\_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过1个，则无法求解，它们仍然为不确定的变元.

            for (j = 0; j < var; j++)

            {

                if (a[i][j] != 0 && free\_x[j]) free\_x\_num++, free\_index = j;

            }

            if (free\_x\_num > 1) continue; // 无法求解出确定的变元.

            // 说明就只有一个不确定的变元free\_index，那么可以求解出该变元，且该变元是确定的.

            temp = a[i][var];

            for (j = 0; j < var; j++)

            {

                if (a[i][j] != 0 && j != free\_index) temp -= a[i][j] \* x[j];

            }

            x[free\_index] = temp / a[i][free\_index]; // 求出该变元.

            free\_x[free\_index] = 0; // 该变元是确定的.

        }

        return var - k; // 自由变元有var - k个.

    }

    // 3. 唯一解的情况: 在var \* (var + 1)的增广阵中形成严格的上三角阵.

    // 计算出Xn-1, Xn-2 ... X0.

    for (i = var - 1; i >= 0; i--)

    {

        temp = a[i][var];

        for (j = i + 1; j < var; j++)

        {

            if (a[i][j] != 0) temp -= a[i][j] \* x[j];

        }

        if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.

        x[i] = temp / a[i][i];

    }

    return 0;

}

int main(void)

{

    freopen("in.txt", "r", stdin);

    freopen("out.txt","w",stdout);

    int i, j;

    int equ,var;

    while (scanf("%d %d", &equ, &var) != EOF)

    {

        memset(a, 0, sizeof(a));

        for (i = 0; i < equ; i++)

        {

            for (j = 0; j < var + 1; j++)

            {

                scanf("%d", &a[i][j]);

            }

        }

//        Debug();

        int free\_num = Gauss(equ,var);

        if (free\_num == -1) printf("无解!\n");

        else if (free\_num == -2) printf("有浮点数解，无整数解!\n");

        else if (free\_num > 0)

        {

            printf("无穷多解! 自由变元个数为%d\n", free\_num);

            for (i = 0; i < var; i++)

            {

                if (free\_x[i]) printf("x%d 是不确定的\n", i + 1);

                else printf("x%d: %d\n", i + 1, x[i]);

            }

        }

        else

        {

            for (i = 0; i < var; i++)

            {

                printf("x%d: %d\n", i + 1, x[i]);

            }

        }

        printf("\n");

    }

    return 0;

}

精简版

int equ,var;

int a[110][110];

int x[110];

int free\_x[110];

int free\_num;

//返回值为-1表示无解，为0是唯一解，否则返回自由变元个数

int Gauss()

{

    int max\_r, col, k;

    free\_num = 0;

    for(k = 0, col = 0; k < equ && col < var; k++, col++)

    {

        max\_r = k;

        for(int i = k+1 ; i < equ; i++)

            if(abs(a[i][col]) > abs(a[max\_r][col]))

                max\_r = i;

        if(a[max\_r][col] == 0)

        {

            k--;

            free\_x[free\_num++] = col; //自由变元

            continue;

        }

        if(max\_r != k)

        {

            for(int j = col; j < var+1; j++)

                swap(a[k][j],a[max\_r][j]);

        }

        for(int i = k+1; i < equ;i++)

            if(a[i][col] != 0)

                for(int j = col; j < var+1;j++)

                    a[i][j] ^= a[k][j];

    }

    for(int i = k;i < equ;i++)

        if(a[i][col] != 0)

            return -1;

    if(k < var)return var-k;

    for(int i = var-1; i >= 0;i--)

    {

        x[i] = a[i][var];

        for(int j = i+1; j < var;j++)

            x[i] ^= (a[i][j] && x[j]);

    }

    return 0;

}

异或模板

//对2取模的01方程组

const int MAXN = 300;

//有equ个方程，var个变元。增广矩阵行数为equ,列数为var+1,分别为0到var

int equ,var;

int a[MAXN][MAXN]; //增广矩阵

int x[MAXN]; //解集

int free\_x[MAXN];//用来存储自由变元（多解枚举自由变元可以使用）

int free\_num;//自由变元的个数

//返回值为-1表示无解，为0是唯一解，否则返回自由变元个数

int Gauss()

{

    int max\_r,col,k;

    free\_num = 0;

    for(k = 0, col = 0 ; k < equ && col < var ; k++, col++)

    {

        max\_r = k;

        for(int i = k+1;i < equ;i++)

        {

            if(abs(a[i][col]) > abs(a[max\_r][col]))

                max\_r = i;

        }

        if(a[max\_r][col] == 0)

        {

            k--;

            free\_x[free\_num++] = col;//这个是自由变元

            continue;

        }

        if(max\_r != k)

        {

            for(int j = col; j < var+1; j++)

                swap(a[k][j],a[max\_r][j]);

        }

        for(int i = k+1;i < equ;i++)

        {

            if(a[i][col] != 0)

            {

                for(int j = col;j < var+1;j++)

                    a[i][j] ^= a[k][j];

            }

        }

    }

    for(int i = k;i < equ;i++)

        if(a[i][col] != 0)

            return -1;//无解

    if(k < var) return var-k;//自由变元个数

    //唯一解，回代

    for(int i = var-1; i >= 0;i--)

    {

        x[i] = a[i][var];

        for(int j = i+1;j < var;j++)

            x[i] ^= (a[i][j] && x[j]);

    }

    return 0;

}

<http://www.cppblog.com/menjitianya/archive/2014/06/08/207226.html>

<http://www.cnblogs.com/kuangbin/p/3265270.html>

### 2 组合数学

二项式

const int CN=20;

long long c[CN][CN]= {};

void cinit()

{

    for(int i=0; i<CN; i++)

    {

        c[i][0]=c[i][i]=1;

        for(int j=1; j<i; j++)

            c[i][j]=c[i-1][j]+c[i-1][j-1];

    }

}

//小的数据可以，大的数据不行，要用逆元处理除

ll cc(ll n,ll m)

{

    if(m>n)

        return 0;

    ll i,ans=1;

    for(i=m+1;i<=n;i++)

        ans=ans\*i;

    for(i=2;i<=n-m;i++)

        ans=ans/i;

    return ans;

}

//处理大数据的

ll cc(ll n,ll m)

{

    if(m>n)

        return 0;

    ll a=1, b=1;

    if(m>n) return 0;

    while(m)

    {

        a=(a\*n)%mod;

        b=(b\*m)%mod;

        m--;

        n--;

    }

    return (ll)a\*inv(b)%mod;

}

普通母函数

for(i=0; i<n; i++)

            {

                for(j=0; j<=m[i]; j++)

                    for(k=0; k+j\*p[i]<=num; k++)

                        tmp[k+j\*p[i]]+=ans[k];

                for(j=0; j<=num; j++)

                {

                    ans[j]=+tmp[j];

                    tmp[j]=0;

                    ans[j]%=10000;

                }

            }

指数型母函数

double tmp[mx\*mx]= {0},ans[mx\*mx]= {0},a[mx]= {0},arrange[mx\*mx]= {1};

ll n,m,i,j,k,num=0;

    for(i=1; i<mx\*mx; i++)

        arrange[i]=i\*arrange[i-1];

    while(~scanf("%I64d%I64d",&n,&m))

    {

        memset(tmp,0,sizeof(tmp));

        memset(ans,0,sizeof(ans));

        memset(a,0,sizeof(a));

        for(i=0; i<n; i++)

            scanf("%lf",&a[i]);

        ans[0]=1;

        for(i=0; i<n; i++)

        {

            for(j=0; j<=a[i]; j++)

                for(k=0; k+j<=m; k++)

                    tmp[k+j]+=ans[k]/arrange[j];

            memcpy(ans,tmp,sizeof(ans));

            memset(tmp,0,sizeof(tmp));

        }

        printf("%.lf\n",ans[m]\*arrange[m]);

    }

c(k, 1) + c(k, 3) + c(k, 5) +…… = 2^（k - 1)

c(k, 0) + c(k, 2) + c(k, 3) + c(k, 5) +…… = 2^（k - 1)

c(k, 0) + c(k, 1) + c(k, 2) +…… = 2^k

1\*C(n,1)+2\*C(n,2)+3\*C(n,3)+…+n\*C(n,n)=n\*2^(n-1)

### 3 卡特兰数

其前几项为 : 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452, ...

令h(0)=1,h(1)=1，catalan数满足递推式[1]  ：

**h(n)= h(0)\*h(n-1)+h(1)\*h(n-2) + ... + h(n-1)h(0) (n>=2)**

例如：h(2)=h(0)\*h(1)+h(1)\*h(0)=1\*1+1\*1=2

h(3)=h(0)\*h(2)+h(1)\*h(1)+h(2)\*h(0)=1\*2+1\*1+2\*1=5

另类递推式[2]  ：

**h(n)=h(n-1)\*(4\*n-2)/(n+1);**

递推关系的解为：

**h(n)=C(2n,n)/(n+1) (n=0,1,2,...)**

递推关系的另类解为：

**h(n)=c(2n,n)-c(2n,n-1)(n=0,1,2,...)**

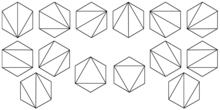
**应用：**

矩阵连乘： P=a1×a2×a3×……×an，依据乘法结合律，不改变其顺序，只用括号表示成对的乘积，试问有几种括号化的方案？(h(n-1)种)

一个栈(无穷大)的[进栈](http://baike.baidu.com/view/346788.htm)序列为1，2，3，…，n，有多少个不同的[出栈](http://baike.baidu.com/view/346791.htm)序列?

有2n个人排成一行进入剧场。入场费5元。其中只有n个人有一张5元钞票，另外n人只有10元钞票，剧院无其它钞票，问有多少中方法使得只要有10元的人买票，售票处就有5元的钞票找零？(将持5元者到达视作将5元入栈，持10元者到达视作使栈中某5元出栈)

在一个[凸多边形](http://baike.baidu.com/view/363944.htm)中，通过若干条互不相交的对角线，把这个多边形划分成了若干个三角形。任务是键盘上输入凸多边形的边数n，求不同划分的方案数f（n）。比如当n=6时，f（6）=14



一位大城市的律师在她住所以北n个街区和以东n个街区处工作。每天她走2n个街区去上班。如果她从不穿越（但可以碰到）从家到办公室的对角线，那么有多少条可能的道路？

在[圆上](http://baike.baidu.com/view/2300579.htm)选择2n个点,将这些点成对连接起来使得所得到的n条线段不相交的方法数？

给定N个[节点](http://baike.baidu.com/view/47398.htm)，能构成多少种不同的[二叉树](http://baike.baidu.com/view/88806.htm)？

### 4 归并排序

归并排序求逆序数

题目网址

<http://poj.org/problem?id=1804>

详解博客

<http://blog.csdn.net/acdreamers/article/details/16849761>

实际上归并排序的交换次数就是这个数组的逆序对个数，为什么呢？

我们可以这样考虑：

归并排序是将数列a[l,h]分成两半a[l,mid]和a[mid+1,h]分别进行归并排序，然后再将这两半合并起来。

在合并的过程中（设l<=i<=mid，mid+1<=j<=h），当a[i]<=a[j]时，并不产生逆序数；当a[i]>a[j]时，在

前半部分中比a[i]大的数都比a[j]大，将a[j]放在a[i]前面的话，逆序数要加上mid+1-i。因此，可以在归并

排序中的合并过程中计算逆序数.

#include <iostream>

#include <string.h>

#include <stdio.h>

using namespace std;

const int N = 1005;

int a[N],tmp[N];

int ans;

void Merge(int l,int m,int r)

{

    int i = l;

    int j = m + 1;

    int k = l;

    while(i <= m && j <= r)

    {

        if(a[i] > a[j])

        {

            tmp[k++] = a[j++];

            ans += m - i + 1;

        }

        else

        {

            tmp[k++] = a[i++];

        }

    }

    while(i <= m) tmp[k++] = a[i++];

    while(j <= r) tmp[k++] = a[j++];

    for(int i=l;i<=r;i++)

        a[i] = tmp[i];

}

void Merge\_sort(int l,int r)

{

    if(l < r)

    {

        int m = (l + r) >> 1;

        Merge\_sort(l,m);

        Merge\_sort(m+1,r);

        Merge(l,m,r);

    }

}

int main()

{

    int n,T,tt=1;

    scanf("%d",&T);

    while(T--)

    {

        scanf("%d",&n);

        for(int i=0;i<n;i++)

            scanf("%d",&a[i]);

        ans = 0;

        Merge\_sort(0,n-1);

        printf("Scenario #%d:\n%d\n\n",tt++,ans);

    }

    return 0;

}

### 5 高精度

const int ra=10;

int ten[4]= {1,ra,ra\*ra,ra\*ra\*ra};

int radix=ra\*ra\*ra\*ra;

const int NV=1000;

struct integer

{

int d[NV];

integer()

{

\*this=integer(0);

}

integer(int x)

{

for (int i=0; i<NV; i++) d[i]=0;

if (!x) d[0]=1;

while(x)

{

d[++d[0]]=x%radix;

x/=radix;

}

}

integer(long long x)

{

for (int i=0; i<NV; i++) d[i]=0;

if (!x) d[0]=1;

while(x)

{

d[++d[0]]=x%radix;

x/=radix;

}

}

integer(const char s[])

{

int len=strlen(s),i,j,k;

d[0]=(len-1)/4+1;

for (i=1; i<NV; i++) d[i]=0;

for (i=len-1; i>=0; i--)

{

j=(len-i-1)/4+1;

k=(len-i-1)%4;

d[j]+=ten[k]\*(s[i]-'0');

}

while(d[0]>1&&d[d[0]]==0) d[0]--;

}

string tostring()

{

string s;

int i,j,temp;

for (i=3; i>=1; i--) if (d[d[0]]>=ten[i]) break;

temp=d[d[0]];

for (j=i; j>=0; j--)

{

s+=(char) (temp/ten[j]+'0');

temp%=ten[j];

}

for (i=d[0]-1; i>0; i--)

{

temp=d[i];

for (j=3; j>=0; j--)

{

s+=(char) (temp/ten[j]+'0');

temp%=ten[j];

}

}

return s;

}

void output()

{

int k=d[0];

printf("%d",d[k--]);

while(k) printf("%04d",d[k--]);

putchar('\n');

}

} d,mid1[15];

bool operator <(const integer &a,const integer &b)

{

if (a.d[0]!=b.d[0]) return a.d[0]<b.d[0];

for (int i=a.d[0]; i>0; i--)

if (a.d[i]!=b.d[i])

return a.d[i]<b.d[i];

return 0;

}

integer operator +(const integer &a,const integer &b)

{

integer c;

c.d[0]=max(a.d[0],b.d[0]);

int i,x=0;

for (i=1; i<=c.d[0]; i++)

{

x+=a.d[i]+b.d[i];

c.d[i]=x%radix;

x/=radix;

}

while(x)

{

c.d[++c.d[0]]=x%radix;

x/=radix;

}

return c;

}

integer operator -(const integer &a,const integer &b)

{

integer c;

c.d[0]=a.d[0];

int i,x=0;

for (i=1; i<=c.d[0]; i++)

{

x+=radix+a.d[i]-b.d[i];

c.d[i]=x%radix;

x=x/radix-1;

}

while(c.d[0]>1&&c.d[c.d[0]]==0) c.d[0]--;

return c;

}

integer operator \*(const integer &a,const integer &b)

{

integer c;

c.d[0]=a.d[0]+b.d[0];

int i,j,x=0;

for (i=1; i<=a.d[0]; i++)

{

x=0;

for (j=1; j<=b.d[0]; j++)

{

x=a.d[i]\*b.d[j]+x+c.d[i+j-1];

c.d[i+j-1]=x%radix;

x/=radix;

}

c.d[i+b.d[0]]=x;

}

while(c.d[0]>1&&c.d[c.d[0]]==0) c.d[0]--;

return c;

}

integer operator \*(const integer &a,const long long &k)

{

integer c;

c.d[0]=a.d[0];

int i;

long long x=0;

for (i=1; i<=a.d[0]; i++)

{

x+=a.d[i]\*k;

c.d[i]=x%radix;

x/=radix;

}

while(x>0)

{

c.d[++c.d[0]]=x%radix;

x/=radix;

}

while(c.d[0]>1&&c.d[c.d[0]]==0) c.d[0]--;

return c;

}

long long rem;

integer operator /(const integer &a,const long long &k)

{

integer c;

c.d[0]=a.d[0];

long long x=0;

for (int i=a.d[0]; i>=1; i--)

{

x+=a.d[i];

c.d[i]=x/k;

x%=k;

rem=x;

x\*=radix;

}

while(c.d[0]>1&&c.d[c.d[0]]==0) c.d[0]--;

return c;

}

bool smaller(const integer &a,const integer &b,int delta)

{

if (a.d[0]+delta!=b.d[0]) return a.d[0]+delta<b.d[0];

for (int i=a.d[0]; i>0; i--)

if (a.d[i]!=b.d[i+delta])

return a.d[i]<b.d[i+delta];

return 1;

}

void Minus(integer &a,const integer &b,int delta)

{

int i,x=0;

for (i=1; i<=a.d[0]-delta; i++)

{

x+=radix+a.d[i+delta]-b.d[i];

a.d[i+delta]=x%radix;

x=x/radix-1;

}

while(a.d[0]>1&&a.d[a.d[0]]==0) a.d[0]--;

}

integer operator /(const integer &a,const integer &b)

{

integer c;

d=a;

int i,j,temp;

mid1[0]=b;

for (i=1; i<=13; i++) mid1[i]=mid1[i-1]\*2;

for (i=a.d[0]-b.d[0]; i>=0; i--)

{

temp=8192;

for (j=13; j>=0; j--)

{

if (smaller(mid1[j],d,i))

{

Minus(d,mid1[j],i);

c.d[i+1]+=temp;

}

temp/=2;

}

}

c.d[0]=max(1,a.d[0]-b.d[0]+1);

while(c.d[0]>1&&c.d[c.d[0]]==0) c.d[0]--;

return c;

}

bool operator ==(const integer &a,const integer &b)

{

if (a.d[0]!=b.d[0]) return 0;

for (int i=1; i<=a.d[0]; i++)

if (a.d[i]!=b.d[i])

return 0;

return 1;

}

void init(int b) ///将大数切换至任意<=10进制

{

for (int i=1; i<=3; i++)

ten[i]=ten[i-1]\*b;

radix=b\*b\*b\*b;

}

### 6 矩阵

typedef long long mytype;

const int SZ=105;

const long long M=1000000007;

long long quickpow(long long a, long long b)

{

    if(b < 0) return 0;

    long long ret = 1;

    a %= M;

    for (; b; b >>= 1, a = (a \* a) % M)

        if (b & 1)

            ret = (ret \* a) % M;

    return ret;

}

long long inv(long long a)

{

    return quickpow(a,M-2);

}

struct mat

{

    int n,m;

    mytype a[SZ][SZ];

    void init()

    {

        memset(a,0,sizeof(a));

    }

    mat(int n=SZ,int m=SZ):n(n),m(m) {}

    mat unit()

    {

        mat t(n,n);

        t.init();

        for (int i=0; i<n; i++)

            t.a[i][i]=1;

        return t;

    }

    mytype \*operator [](int n)

    {

        return \*(a+n);

    }

    mat operator +(const mat &b)

    {

        mat t(n,m);

        for (int i=0; i<n; i++)

            for (int j=0; j<m; j++)

                t.a[i][j]=(a[i][j]+b.a[i][j]+M)%M;

        return t;

    }

    mat operator -(const mat &b)

    {

        mat t(n,m);

        for (int i=0; i<n; i++)

            for (int j=0; j<m; j++)

                t.a[i][j]=(a[i][j]-b.a[i][j]+M)%M;

        return t;

    }

    mat operator \*(const mat &b)

    {

        mat t(n,m);

        for(int i=0; i<n; i++)

            for(int j=0; j<m; j++)

            {

                t.a[i][j]=0;

                for(int k=0; k<m; k++)

                    t.a[i][j]=(t.a[i][j]+(a[i][k]\*b.a[k][j])%M)%M;

            }

        return t;

    }

    mat operator \*(const mytype &b)

    {

        mat t(n,m);

        for(int i=0; i<n; i++)

            for(int j=0; j<m; j++)

                t.a[i][j]=a[i][j]\*b%M;

        return t;

    }

    mat operator /(const mytype &b)

    {

        mat t(n,m);

        for(int i=0; i<n; i++)

            for(int j=0; j<m; j++)

                t.a[i][j]=a[i][j]\*inv(b)%M;

        return t;

    }

    mat operator !()

    {

        mat t(n,m);

        for(int i=0; i<n; i++)

            for(int j=0; j<m; j++)

                t.a[i][j]=a[j][i];

        return t;

    }

    mytype det()

    {

    }

    mat invm(mat &a)

    {

    }

    friend mat quickpow(mat a, mytype b)

    {

        if(b<0) return a.unit();

        mat ret=a.unit();

        for (; b; b>>=1,a=a\*a)

            if (b&1)

                ret=ret\*a;

        return ret;

    }

    void in()

    {

        for (int i=0; i<n; i++)

            for (int j=0; j<m; j++)

                scanf("%lld",&a[i][j]);

    }

    void out()

    {

        for (int i=0; i<n; i++)

            for (int j=0; j<m; j++)

                printf("%lld%c",a[i][j]," \n"[j==m-1]);

    }

};

矩阵快速幂

#define mod 10000007

const int maxn = 10;

const int maxm = 10;

ll k;

int n;

int num[100005];

struct Matrix {

    int n, m;

    long long a[maxn][maxm];

    void clear() {

        n = m = 0;

        memset(a, 0, sizeof(a));

    }

    Matrix operator \* (const Matrix &b) const { //实现矩阵乘法

        Matrix tmp;

        tmp.n = n;

        tmp.m = b.m;

        for (int i = 0; i < n; i++)

            for (int j = 0; j < b.m; j++) tmp.a[i][j] = 0;

        for (int i = 0; i < n; i++)

            for (int j = 0; j < m; j++) {

                if (!a[i][j]) continue;

                for (int k = 0; k < b.m; k++)

                    tmp.a[i][k] += a[i][j] \* b.a[j][k]%mod , tmp.a[i][k] %= mod;

            }

        return tmp;

    }

    void Copy(const Matrix &b) {

        n = b.n, m = b.m;

        for (int i = 0; i < n; i++)

            for(int j = 0; j < m; j++) a[i][j] = b.a[i][j];

    }

    void unit(int sz) {

        n = m = sz;

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) a[i][j] = 0;

            a[i][i] = 1;

        }

    }

};

Matrix A, B;

void init() {

    A.clear(); //矩阵A是构造的矩阵

    A.n = A.m = 3;

    A.a[0][0] = 1;

    A.a[0][1] = 1;

    A.a[1][0] = 1;

    A.a[2][0] = 1;

    A.a[2][1] = 1;

    A.a[2][2] = 1;

    B.clear();

    B.n = 3;

    B.m = 1;

}

//Matrix Matrix\_pow(Matrix A,ll k, ll mod) {

Matrix Matrix\_pow(Matrix A,ll k)

{

    Matrix res;

    res.clear();

    res.n = res.m = 3;

    for (int i = 0; i < 10; i++) res.a[i][i] = 1;

    while(k) {

        if (k & 1) res.Copy(A \* res);

        k >>= 1;

        A.Copy(A \* A);

    }

    return res;

}

## 10 博弈

### 1 巴士博弈

### 2 尼姆博弈

### 3 威佐夫博弈

int wzf(int n, int m)

{

if(n > m)

swap(n, m);

int k = m-n;

int a = (k \* (1.0 + sqrt(5.0))/2.0);

if(a == n)

return 0;

else

return 1;

}

### 4 sg图

int sg[mx];

    int v[mx];

    int i,j;

    for(i=1; i<mx; i++)

    {

        int x=i;

        mem(v);

        for(j=1; j<=x; j\*=2)

            v[sg[x-j]]++;

        for(j=0; j<mx; j++)

            if(v[j]==0)

                break;

        sg[i]=j;

    }

## 11 计算几何

### 1 三角形

内心：(欧拉定理)△ABC中，R和r分别为外接圆为和内切圆的半径，O和I分别为其外心和内心，则OI2=R2-2Rr

重心：

1、重心到顶点的距离与重心到对边中点的距离之比为2：1。

2、重心和三角形3个顶点组成的3个三角形面积相等。

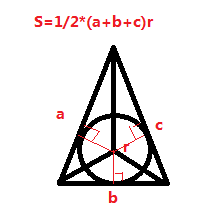
3、重心到三角形3个顶点距离平方的和最小。 (等边三角形）

4、在平面直角坐标系中，重心的坐标是顶点坐标的算术平均数，

5、三角形内到三边距离之积最大的点。

6、在△ABC中，若MA向量+MB向量+MC向量=0（向量） ，则M点为△ABC的重心，反之也成立。

7、设△ABC重心为G点，所在平面有一点O，则向量OG=1/3（向量OA+向量OB+向量OC）

**S=1/2\*Lr,其中r为三角形内切圆半径.  
**

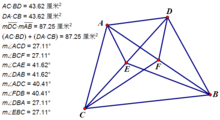
### 2 四边形

调和四边形

定义

对边乘积相等的圆内接四边形叫做调和四边形。

性质



1、调和四边形的其中一条对角线，与过其余两点的四边形外接圆的两条切线，这三条直线共点；

2、设调和四边形ABCD中，对角线AC中点为M，则△AMB∽△DMA∽△DCB，△BMC∽△CMD∽△BAD；

3、设调和四边形ABCD中，对角线AC与过B、D两点的四边形ABCD外接圆的切线所共的点记为P，记AP交BD于Q，则AQ为△ABD的一条陪位中线，A、Q、C、P四点为调和点列；取对角线AC中点M，设四边形ABCD外接圆圆心为O，则B、P、D、O、M五点共圆；

判定

1、利用定义：对边乘积相等的圆内接四边形是调和四边形；

2、过圆外一点作圆的两条切线与一条割线，与圆相交的四点构成的凸四边形为调和四边形。

托勒密定理

圆内接凸四边形两对对边乘积的和等于两条对角线的乘积。

托勒密定理的逆定理同样成立：一个凸四边形两对对边乘积的和等于两条对角线的乘积，则这个凸四边形内接于一圆

### 3 最近点对

const int MAXN=100005;

// 分治算法求最近点对

struct point

{

    double x,y;

};

point p[MAXN];

int index[MAXN];  //保存筛选的坐标点的索引

int cmpx(const point &a,const point &b)

{

    return a.x<b.x;

}

int cmpy(int a,int b)  //这里用的是下标索引

{

    return p[a].y<p[b].y;

}

inline double dis(point &a , point &b)

{

    return sqrt( (a.x-b.x)\*(a.x-b.x) + (a.y-b.y)\*(a.y-b.y) );

}

inline double min(double a , double b)

{

    return a<b?a:b;

}

double closest(int low , int high)

{

    if(low+1==high)

        return dis(p[low],p[high]);

    if(low+2==high)

        return min( dis(p[low],p[high]) , min(dis(p[low],p[low+1]),dis(p[low+1],p[high])) );

    int mid = (low + high)>>1;//求中点

    double ans = min( closest(low,mid),closest(mid+1,high) );  //分治法进行递归求解

    int cnt = 0;

    for(int i=low; i<=high; i++) //把x坐标在p[mid].x-ans~p[mid].x+ans范围内的点取出来

    {

        if(p[i].x >= p[mid].x - ans && p[i].x <= p[mid].x + ans)

            index[cnt++]=i;     //保存的是下标索引

    }

    sort(index,index+cnt,cmpy);   //按y坐标进行升序排序

    for(int i=0; i<cnt; i++)

        for(int j=i+1; j<cnt; j++)

        {

            if(p[index[j]].y - p[index[i]].y >= ans)   //注意下标索引

                break;

            ans = min( ans,dis(p[index[i]],p[index[j]]) );

        }

    return ans;

}

void init(int n)

{

    for (int i=0; i<n; i++)

        scanf("%lf%lf",&p[i].x,&p[i].y);

    sort(p,p+n,cmpx);

    return;

}

int main()

{

    int n;

    while(cin>>n,n!=0)

    {

        init(n);

        printf("%.2lf\n",closest(0,n-1)/2);

    }

    return 0;

}

### 4 凸包

#include<cstdio>

#include<cmath>

#include<iostream>

#include<algorithm>

using namespace std;

const int MAXN=1000;

struct point    //结构体储存点集Q

{

    int x,y;

};

point list[MAXN];

int stack[MAXN],top;

int cross(point p0,point p1,point p2){ //计算叉积  （向量p0p1与向量p0p2的叉积）

    return (p1.x-p0.x)\*(p2.y-p0.y)-(p1.y-p0.y)\*(p2.x-p0.x);

}

double dis(point p1,point p2){  //计算 p1p2的 距离

    return sqrt((double)(p2.x-p1.x)\*(p2.x-p1.x)+(p2.y-p1.y)\*(p2.y-p1.y));

}

bool cmp(point p1,point p2){ //极角排序函数 ， 角度相同则距离小的在前面

    int tmp=cross(list[0],p1,p2);

    if(tmp>0)

        return true;

    else if(tmp==0&&dis(list[0],p1)<dis(list[0],p2))

        return true;

    else

        return false;

}

void init(int n){ //输入，并把最左下方的点放在 list[0]。并且进行极角排序

    int i,k=0;

    point p0;

    scanf("%d%d",&list[0].x,&list[0].y);

    p0.x=list[0].x;

    p0.y=list[0].y;

    for(i=1;i<n;i++){

        scanf("%d%d",&list[i].x,&list[i].y);

        if( (p0.y>list[i].y) || ((p0.y==list[i].y)&&(p0.x>list[i].x)) ){//找纵坐标最小的点，如果纵坐标相同，取横坐标最小的点

            p0.x=list[i].x;

            p0.y=list[i].y;

            k=i;

        }

    }

    list[k]=list[0];

    list[0]=p0;

    sort(list+1,list+n,cmp);//极角排序

}

void graham(int n){

    int i;

    if(n==1){

        top=0;

        stack[0]=0;

    }

    if(n==2){

        top=1;

        stack[0]=0;

        stack[1]=1;

    }

    if(n>2){

        for(i=0;i<=1;i++) stack[i]=i;

        top=1;

        for(i=2;i<n;i++){

            while(top>0&&cross(list[stack[top-1]],list[stack[top]],list[i])<=0)

            top--;

            top++;

            stack[top]=i;

        }

    }

}

int main()

{

    int i,n;

    cin>>n;

    init(n);

    graham(n);

    for(i=0;i<=top;i++)

        cout<<list[stack[i]].x<<' '<<list[stack[i]].y<<endl;

    return 0;

}

### 5 公式

二维

求三角形面积

S = sqrt(s1\* (s1-a) \* (s1-b)\*(s1-c))

其中s1表示三角形周长的一半

三维

四面体内切球半径

s为四个面的面积，其中共点的三个向量a,b,c

r=(a×b)\*c/2.0/s

已知任意四面体（三棱锥）六条棱的棱长，求其体积。  
不妨记同一顶点引出的三条棱棱长的平方分别为a，b，c，它们的对棱棱长的平方分别为d，e，f，则四面体的体积V满足：  
V=sqrt[ad(b+c+e+f-a-d)+be(a+c+d+f-b-e)+cf(a+b+d+e-c-f)-abf-bcd-cae-def)]/12

用向量

混合积 [**a b c**] = (**a**×**b**)·**c**可以得到以**a**，**b**，**c**为棱的平行六面体的体积。

正四面体内切球半径为r=√6a/12,外接球半径为 r=√6a/4

四面体内切球半径为r=v/2/s(v为四面体体积，s为四面体表面积)

向量叉积

三维坐标叉乘

point Cross(point a, point b)

{

point s;

s.x = a.y\*b.z - a.z\*b.y;

s.y = a.z\*b.x - a.x\*b.z;

s.z = a.x\*b.y - b.x\*a.y;

return s;

}

三维坐标点乘

double dot(point a, point b)

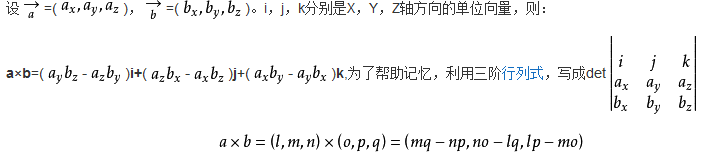
{

double s = a.x\*b.x+a.y\*b.y+a.z\*b.z;

return s;

}

E:\Users\Dequan\AppData\Local\YNote\data\99893492@qq.com\812259fa8db946089c4b964dd2c3e6a5\33fa828b4758.png



反交换律：

a×b= -b×a

加法的分配律：

a× (b+c) =a×b+a×c

与标量乘法兼容：

(ra) ×b=a× (rb) = r(a×b)

不满足结合律，但满足雅可比恒等式：

a× (b×c) +b× (c×a) +c× (a×b) =0

分配律，线性性和雅可比恒等式别表明：具有向量加法和叉积的 R3 构成了一个李代数。

两个非零向量a和b平行，当且仅当a×b=0[2]

混合积 [a b c] = (a×b)·c可以得到以a，b，c为棱的平行六面体的体积。

拉格朗日公式

a×（b×c）=b(a·c) -c(a·b),

三个点确定一个平面

///求平面方程 A\*x + B\*y + C\*z + D = 0

struct point

{

double x, y, z;

} a[10];

double A = ((a[2].y-a[1].y)\*(a[3].z-a[1].z)-(a[2].z-a[1].z)\*(a[3].y-a[1].y));

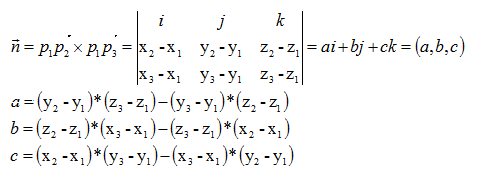
double B = ((a[2].z-a[1].z)\*(a[3].x-a[1].x)-(a[2].x-a[1].x)\*(a[3].z-a[1].z));

double C = ((a[2].x-a[1].x)\*(a[3].y-a[1].y)-(a[2].y-a[1].y)\*(a[3].x-a[1].x));

double D = -(A \* a[1].x + B \* a[1].y + C \* a[1].z);

已知三点p1（x1,y1,z1），p2(x2,y2,z2)，p3(x3,y3,z3)，要求确定的平面方程

关键在于求出平面的一个法向量，为此做向量p1p2（x2-x1,y2-y1,z2-z1), p1p3(x3-x1,y3-y1,z3-z1),平面法线和这两个向量垂直，因此法向量n：



平面方程：a(x-x1)+b(y-y1)+ c(z-z1)=0；

d=-a\*x1-b\*y1-c\*z1。

平面平面方程为ax+by+cz+d=0。

### 6 计算几何模板

#include<cstdio>

#include<cstdlib>

#include<cstring>

#include<cmath>

#include<ctime>

#include<cassert>

#include<climits>

#include<iostream>

#include<algorithm>

#include<string>

#include<vector>

#include<deque>

#include<list>

#include<set>

#include<map>

#include<stack>

#include<queue>

#include<numeric>

#include<iomanip>

#include<bitset>

#include<sstream>

#include<fstream>

#define debug puts("-----")

#define pi (acos(-1.0))

#define eps (1e-8)

#define inf (1<<30)

#define INF (1ll<<62)

using namespace std;

inline double sqr(const double &x)

{

return x \* x;

}

inline int sgn(const double &x)

{

return x < -eps ? -1 : x > eps;

}

struct point

{

double x, y;

point(const double &x = 0, const double &y = 0): x(x), y(y) {}

friend point operator + (const point &a, const point &b)

{

return point(a.x + b.x, a.y + b.y);

}

friend point operator - (const point &a, const point &b)

{

return point(a.x - b.x, a.y - b.y);

}

friend point operator \* (const point &a, const double &b)

{

return point(a.x \* b, a.y \* b);

}

friend point operator \* (const double &a, const point &b)

{

return point(a \* b.x, a \* b.y);

}

friend point operator / (const point &a, const double &b)

{

return point(a.x / b, a.y / b);

}

friend bool operator == (const point &a, const point &b)

{

return !sgn(a.x - b.x) && !sgn(a.y - b.y);

}

friend bool operator < (const point &a, const point &b)

{

return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);

}

double norm()

{

return sqrt(sqr(x) + sqr(y));

}

friend double det(const point &a, const point &b)

{

return a.x \* b.y - a.y \* b.x;

}

friend double dot(const point &a, const point &b)

{

return a.x \* b.x + a.y \* b.y;

}

friend double dist(const point &a, const point &b)

{

return (a - b).norm();

}

double arg()

{

return atan2(y, x);

// double res = atan2(y, x); //(-pi, pi]

// if(res < -pi / 2 + eps) res += 2 \* pi; //eps修正精度

// return res;

}

//逆时针旋转angle弧度

point rotate(const double &angle)

{

return rotate(cos(angle), sin(angle));

}

point rotate(const point &p, const double &angle)

{

return (\*this-p).rotate(angle)+p;

}

point rotate(const double &cosa, const double &sina)

{

return point(x \* cosa - y \* sina, x \* sina + y \* cosa);

}

int in()

{

return scanf("%lf %lf", &x, &y);

}

void out()

{

printf("%.2f %.2f\n", x, y);

}

};

struct line

{

point s, t;

line(const point &s = point(), const point &t = point()): s(s), t(t) {}

point vec() const

{

return t - s;

}

double norm() const

{

return vec().norm();

}

//点在直线上

bool ispointonline(const point &p) const

{

return sgn(det(p - s, t - s)) == 0;

}

//点在线段上

bool ispointonseg(const point &p) const

{

return ispointonline(p) && sgn(dot(p - s, p - t)) <= 0;

}

//点在线段上不含端点

bool ispointonsegex(const point &p)

{

return ispointonline(p) && sgn(dot(p - s, p - t)) < 0;

}

//点到直线的垂足

point pointprojline(const point &p)

{

return s + vec() \* ((dot(p - s, vec()) / norm()) / (norm()));

}

//点到直线的距离

double pointdistline(const point &p)

{

return fabs(det(p - s, vec()) / norm());

}

//点到线段的距离

double pointdistseg(const point &p)

{

if (sgn(dot(p - s, t - s)) < 0) return (p - s).norm();

if (sgn(dot(p - t, s - t)) < 0) return (p - t).norm();

return pointdistline(p);

}

//判断两直线是否平行

friend bool parallel(const line &l1, const line &l2)

{

return !sgn(det(l1.vec(), l2.vec()));

}

//判断两个点是否在直线的同一侧

friend bool sameside(const line &l, const point &a, const point &b)

{

return sgn(det(b - l.s, l.vec())) \* sgn(det(a - l.s, l.vec())) > 0;

}

//两直接的交点

friend point linexline(const line l1, const line l2) //利用相似三角形对应边成比例

{

double s1 = det(l1.s - l2.s, l2.vec());

double s2 = det(l1.t - l2.s, l2.vec());

return (l1.t \* s1 - l1.s \* s2) / (s1 - s2);

}

//判断线段交

friend bool issegxseg(const line &l1, const line &l2)

{

if(!sgn(det(l2.s - l1.s, l1.vec())) && !sgn(det(l2.t - l1.s, l1.vec())))

{

return l1.ispointonseg(l2.s) ||

l1.ispointonseg(l2.t) ||

l2.ispointonseg(l1.s) ||

l2.ispointonseg(l1.t);

}

return !sameside(l1, l2.s, l2.t) && !sameside(l2, l1.s, l1.t);

}

//直线沿法线方向移动d距离

friend line move(const line &l, const double &d)

{

point t = l.vec();

t = t / t.norm();

t = t.rotate(pi / 2);

return line(l.s + t \* d, l.t + t \* d);

}

int in()

{

s.in();

return t.in();

}

void out()

{

s.out(), t.out();

}

};

struct polygon

{

#define next(i) ((i+1)%n)

int n;

vector<point> p;

polygon(int n = 0): n(n)

{

p.resize(n);

}

// polygon(vector<point> &v):p(v){}

//多边形周长

double perimeter()

{

double sum = 0;

for (int i = 0; i < n; i++)

sum += (p[next(i)] - p[i]).norm();

return sum;

}

//多边形面积

double area()

{

double sum = 0;

for (int i = 0; i < n; i++)

sum += det(p[i], p[next(i)]);

return sum / 2 + eps; //要加eps吗？

}

//判断点与多边形的位置关系，0外, 1内,2边上

int pointin(const point &t)

{

int num = 0;

for (int i = 0; i < n; i++)

{

if (line(p[i], p[next(i)]).ispointonseg(t))

return 2;

int k = sgn(det(p[next(i)] - p[i], t - p[i]));

int d1 = sgn(p[i].y - t.y);

int d2 = sgn(p[next(i)].y - t.y);

if (k > 0 && d1 <= 0 && d2 > 0) num++;

if (k < 0 && d2 <= 0 && d1 > 0) num--;

}

return num != 0;

}

//多边形重心

point masscenter()

{

point ans;

if (sgn(area()) == 0) return ans;

for (int i = 0; i < n; i++)

ans = ans + (p[i] + p[next(i)]) \* det(p[i], p[next(i)]);

return ans / area() / 6 + eps; //要加eps吗？

}

//多边形边界上格点的数量

int borderpointnum()

{

int num = 0;

for (int i = 0; i < n; i++)

{

int a = fabs(p[next(i)].x - p[i].x);

int b = fabs(p[next(i)].y - p[i].y);

num += \_\_gcd(a, b);

}

return num;

}

//多边形内格点数量

int insidepointnum()

{

return int(area()) + 1 - borderpointnum() / 2;

}

void in()

{

for (int i = 0; i < n; i++)

p[i].in();

}

void out()

{

for (int i = 0; i < n; i++)

p[i].out();

}

};

struct convex : public polygon

{

convex(int n = 0): polygon(n) {}

// convex(vector<point> &v):polygon(v){}

//需要先求凸包，若凸包每条边除端点外都有点，则可唯一确定凸包

bool isunique(vector<point> &v)

{

if (sgn(area()) == 0)

return 0;

for (int i = 0; i < n; i++)

{

line l(p[i], p[next(i)]);

bool flag = 0;

for (int j = 0; j < v.size(); j++)

if (l.ispointonsegex(v[j]))

{

flag = 1;

break;

}

if (!flag)

return 0;

}

return 1;

}

//O(n)时间内判断点是否在凸包内

bool containon(const point &a)

{

int sign = 0;

for (int i = 0; i < n; i++)

{

int x = sgn(det(p[i] - a, p[next(i)] - a));

if (x)

{

if (sign)

{

if (sign != x)

return 0;

}

else

sign = x;

}

}

return 1;

}

//O(logn)时间内判断点是否在凸包内

bool containologn(const point &a)

{

point g = (p[0] + p[n / 3] + p[2 \* n / 3]) / 3.0;

int l = 0, r = n;

while(l + 1 < r)

{

int m = (l + r) / 2;

if (sgn(det(p[l] - g, p[m] - g)) > 0)

{

if (sgn(det(p[l] - g, a - g)) >= 0 && sgn(det(p[m] - g, a - g)) < 0)

r = m;

else

l = m;

}

else

{

if (sgn(det(p[l] - g, a - g)) < 0 && sgn(det(p[m] - g, a - g)) >= 0)

l = m;

else

r = m;

}

}

return sgn(det(p[r % n] - a, p[l] - a)) - 1;

}

//最远点对（直径）

int first, second; //最远的两个点对应标号

double diameter()

{

double mx = 0;

if (n == 1)

{

first = second = 0;

return mx;

}

for (int i = 0, j = 1; i < n; i++)

{

while(sgn(det(p[next(i)] - p[i], p[j] - p[i]) -

det(p[next(i)] - p[i], p[next(j)] - p[i])) < 0)

j = next(j);

double d = dist(p[i], p[j]);

if (d > mx)

{

mx = d;

first = i;

second = j;

}

d = dist(p[next(i)], p[next(j)]);

if (d > mx)

{

mx = d;

first = next(i);

second = next(j);

}

}

return mx;

}

//凸包是否与直线有交点O(log(n)), 需要On的预处理, 适合判断与直线集是否有交点

vector<double> ang; //角度

bool isinitangle;

int finda(const double &x)

{

return upper\_bound(ang.begin(), ang.end(), x) - ang.begin();

}

double getangle(const point &p) //获取向量角度[0, 2pi]

{

double res = atan2(p.y, p.x); //(-pi, pi]

// if (res < 0) res += 2 \* pi; //为何不可以

if(res < -pi / 2 + eps) res += 2 \* pi; //eps修正精度

return res;

}

void initangle()

{

for(int i = 0; i < n; i++)

ang.push\_back(getangle(p[next(i)] - p[i]));

isinitangle = 1;

}

bool isxline(const line &l)

{

if(!isinitangle) initangle();

int i = finda(getangle(l.t - l.s));

int j = finda(getangle(l.s - l.t));

if(sgn(det(l.t - l.s, p[i] - l.s) \* det(l.t - l.s, p[j] - l.s) >= 0))

return 0;

return 1;

}

};

convex convexhull(vector<point> &a)

{

//从一个vector获取凸包

convex res(2 \* a.size() + 5); //为何？经测试好像只需要a.size()？

sort(a.begin(), a.end());

a.erase(unique(a.begin(), a.end()), a.end());

int m = 0;

for (int i = 0; i < a.size(); i++)

{

//<=0则不含边界，<0则含边界

while(m > 1 && sgn(det(res.p[m - 1] - res.p[m - 2], a[i] - res.p[m - 2])) <= 0)

m--;

res.p[m++] = a[i];

}

int k = m;

for (int i = a.size() - 2; i >= 0; i--)

{

while(m > k && sgn(det(res.p[m - 1] - res.p[m - 2], a[i] - res.p[m - 2])) <= 0)

m--;

res.p[m++] = a[i];

}

if (m > 1) m--;

res.p.resize(m);

res.n = m;

return res;

}

struct halfplane: public line

{

//ax+by+c<=0

double a, b, c;

//s->t的左侧表示半平面

halfplane(const point &s = point(), const point &t = point()): line(s, t)

{

a = t.y - s.y;

b = s.x - t.x;

c = det(t, s);

}

halfplane(double a, double b, double c): a(a), b(b), c(c) {}

//求点p带入直线方程的值

double calc(const point &p) const

{

return p.x \* a + p.y \* b + c;

}

//好像跟linexline一样，那个是4个点计算。这个是用abc与两点进行计算

friend point halfxline(const halfplane &h, const line &l)

{

point res;

double t1 = h.calc(l.s), t2 = h.calc(l.t);

res.x = (t2 \* l.s.x - t1 \* l.t.x) / (t2 - t1);

res.y = (t2 \* l.s.y - t1 \* l.t.y) / (t2 - t1);

return res;

}

//用abc进行计算 尚未测试

friend point halfxhalf(const halfplane &h1, const halfplane&h2)

{

return point((h1.b \* h2.c - h1.c \* h2.b) / (h1.a \* h2.b - h2.a \* h1.b) + eps,

(h1.a \* h2.c - h2.a \* h1.c) / (h1.b \* h2.a - h1.a \* h2.b) + eps);

}

//凸多边形与半平面交(cut)

friend convex halfxconvex(const halfplane &h, const convex &c)

{

convex res;

for (int i = 0; i < c.n; i++)

{

if (h.calc(c.p[i]) < -eps)

res.p.push\_back(c.p[i]);

else

{

int j = i - 1;

if (j < 0) j = c.n - 1;

if (h.calc(c.p[j]) < -eps)

res.p.push\_back(halfxline(h, line(c.p[j], c.p[i])));

j = i + 1;

if (j == c.n) j = 0;

if (h.calc(c.p[j]) < -eps)

res.p.push\_back(halfxline(h, line(c.p[i], c.p[j])));

}

}

res.n = res.p.size();

return res;

}

//点在半平面内

friend int satisfy(const point &p, const halfplane &h)

{

return sgn(det(p - h.s, h.t - h.s)) <= 0;

}

friend bool operator <(const halfplane &h1, const halfplane &h2)

{

int res = sgn(h1.vec().arg() - h2.vec().arg());

return res == 0 ? satisfy(h1.s, h2) : res < 0;

}

//半平面交出的凸多边形

friend convex halfx(vector<halfplane> &v)

{

sort(v.begin(), v.end());

deque<halfplane> q;

deque<point> ans;

q.push\_back(v[0]);

for (int i = 1; i < v.size(); i++)

{

if (sgn(v[i].vec().arg() - v[i - 1].vec().arg()) == 0)

continue;

while(ans.size() > 0 && !satisfy(ans.back(), v[i]))

{

ans.pop\_back();

q.pop\_back();

}

while(ans.size() > 0 && !satisfy(ans.front(), v[i]))

{

ans.pop\_front();

q.pop\_front();

}

ans.push\_back(linexline(q.back(), v[i]));

q.push\_back(v[i]);

}

while(ans.size() > 0 && !satisfy(ans.back(), q.front()))

{

ans.pop\_back();

q.pop\_back();

}

while(ans.size() > 0 && !satisfy(ans.front(), q.back()))

{

ans.pop\_front();

q.pop\_front();

}

ans.push\_back(linexline(q.back(), q.front()));

convex c(ans.size());

int i = 0;

for (deque<point>::iterator it = ans.begin(); it != ans.end(); it++, i++)

c.p[i] = \*it;

return c;

}

};

//多边形的核，逆时针

convex core(const polygon &a)

{

convex res;

res.p.push\_back(point(-inf, -inf));

res.p.push\_back(point(inf, -inf));

res.p.push\_back(point(inf, inf));

res.p.push\_back(point(-inf, inf));

res.n = 4;

for (int i = 0; i < a.n; i++)

res = halfxconvex(halfplane(a.p[i], a.p[(i + 1) % a.n]), res);

return res;

}

//凸多边形交出的凸多边形

convex convexxconvex(convex &c1, convex &c2)

{

vector<halfplane> h;

for (int i = 0; i < c1.p.size(); i++)

h.push\_back(halfplane(c1.p[i], c1.p[(i + 1) % c1.p.size()]));

for (int i = 0; i < c2.p.size(); i++)

h.push\_back(halfplane(c2.p[i], c2.p[(i + 1) % c2.p.size()]));

return halfx(h);

}

double mysqrt(double n) //防止出现sqrt(-eps)的情况

{

return sqrt(max(0.0, n));

}

struct circle

{

point o;

double r;

circle(point o = point(), double r = 0): o(o), r(r) {}

bool operator ==(const circle &c)

{

return o == c.o && !sgn(r - c.r);

}

double area()

{

return pi \* r \* r;

}

//圆与线段交 用参数方程表示直线：P=A+t\*(B-A)，带入圆的方程求解t

friend vector<point> cirxseg(const circle &c, const line &l)

{

double dx = l.t.x - l.s.x, dy = l.t.y - l.s.y;

double A = dx \* dx + dy \* dy;

double B = 2 \* dx \* (l.s.x - c.o.x) + 2 \* dy \* (l.s.y - c.o.y);

double C = sqr(l.s.x - c.o.x) + sqr(l.s.y - c.o.y) - sqr(c.r);

double delta = B \* B - 4 \* A \* C;

vector<point> res;

if(sgn(delta) >= 0) //or delta > -eps ?

{

//可能需要注意delta接近-eps的情况，所以使用mysqrt

double w1 = (-B - mysqrt(delta)) / (2 \* A);

double w2 = (-B + mysqrt(delta)) / (2 \* A);

if(sgn(w1 - 1) <= 0 && sgn(w1) >= 0)

res.push\_back(l.s + w1 \* (l.t - l.s));

if(sgn(w2 - 1) <= 0 && sgn(w2) >= 0)

res.push\_back(l.s + w2 \* (l.t - l.s));

}

return res;

}

//圆与直线交

friend vector<point> cirxline(const circle &c, const line &l)

{

double dx = l.t.x - l.s.x, dy = l.t.y - l.s.y;

double A = dx \* dx + dy \* dy;

double B = 2 \* dx \* (l.s.x - c.o.x) + 2 \* dy \* (l.s.y - c.o.y);

double C = sqr(l.s.x - c.o.x) + sqr(l.s.y - c.o.y) - sqr(c.r);

double delta = B \* B - 4 \* A \* C;

vector<point> res;

if(sgn(delta) >= 0) //or delta > -eps ?

{

double w1 = (-B - mysqrt(delta)) / (2 \* A);

double w2 = (-B + mysqrt(delta)) / (2 \* A);

res.push\_back(l.s + w1 \* (l.t - l.s));

res.push\_back(l.s + w2 \* (l.t - l.s));

}

return res;

}

//扇形面积 a->b

double sectorarea(const point &a, const point &b) const

{

double theta = atan2(a.y, a.x) - atan2(b.y, b.x);

while (theta < 0) theta += 2 \* pi;

while (theta > 2 \* pi) theta -= 2 \* pi;

theta = min(theta, 2 \* pi - theta);

return sgn(det(a, b)) \* theta \* r \* r / 2.0; //此处交大板子有误

}

//与线段AB的交点计算面积 a->b

double calcarea(const point &a, const point &b) const

{

vector<point> p = cirxseg(\*this, line(a, b));

bool ina = sgn((a - o).norm() - r) < 0;

bool inb = sgn((b - o).norm() - r) < 0;

if (ina)

{

if (inb) return det(a - o, b - o) / 2;

else return det(a - o, p[0] - o) / 2 + sectorarea(p[0] - o, b - o);

}

else

{

if (inb) return det(p[0] - o, b - o) / 2 + sectorarea(a - o, p[0] - o);

else

{

if (p.size() == 2)

return sectorarea(a - o, p[0] - o) + sectorarea(p[1] - o, b - o)

+ det(p[0] - o, p[1] - o) / 2;

else

return sectorarea(a - o, b - o);

}

}

}

//圆与多边形交，结果可以尝试+eps

friend double cirxpolygon(const circle &c, const polygon &a)

{

int n = a.p.size();

double ans = 0;

for(int i = 0; i < n; i++)

{

if(sgn(det(a.p[i] - c.o, a.p[next(i)] - c.o)) == 0)

continue;

ans += c.calcarea(a.p[i], a.p[next(i)]);

}

return fabs(ans);

}

//点在圆内，不包含边界

bool pointin(const point &p)

{

return sgn((p - o).norm() - r) < 0;

}

};

//三点求圆

circle getcircle3(const point &p0, const point &p1, const point &p2)

{

double a1 = p1.x - p0.x, b1 = p1.y - p0.y, c1 = (a1 \* a1 + b1 \* b1) / 2;

double a2 = p2.x - p0.x, b2 = p2.y - p0.y, c2 = (a2 \* a2 + b2 \* b2) / 2;

double d = a1 \* b2 - a2 \* b1;

point o(p0.x + (c1 \* b2 - c2 \* b1) / d, p0.y + (a1 \* c2 - a2 \* c1) / d);

return circle(o, (o - p0).norm());

}

//直径上两点求圆

circle getcircle2(const point &p0, const point &p1)

{

point o((p0.x + p1.x) / 2, (p0.y + p1.y) / 2);

return circle(o, (o - p0).norm());

}

//最小圆覆盖 用之前可以随机化random\_shuffle

circle mincircover(vector<point> &a)

{

int n = a.size();

circle c(a[0], 0);

for (int i = 1; i < n; i++)

if (!c.pointin(a[i]))

{

c.o = a[i];

c.r = 0;

for (int j = 0; j < i; j++)

if (!c.pointin(a[j]))

{

c = getcircle2(a[i], a[j]);

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

if (!c.pointin(a[k]))

c = getcircle3(a[i], a[j], a[k]);

}

}

return c;

}

int main()

{

return 0;

}