**目录**

[1 排序 3](#_Toc27144306)

[1.1 快速排序 3](#_Toc27144307)

[1.2 归并排序 3](#_Toc27144308)

[1.3 堆排序 4](#_Toc27144309)

[2 动态规划 5](#_Toc27144310)

[2.1 背包 5](#_Toc27144311)

[2.2 最长上升子序列（LIS） 6](#_Toc27144312)

[2.3 最长公共子序列（LCS） 7](#_Toc27144313)

[2.4 最长上升公共子序列（LCIS） 9](#_Toc27144314)

[2.5 区间最值查询（RMQ） 10](#_Toc27144315)

[2.6 N重ALS 11](#_Toc27144316)

[2.7 矩阵链乘法（MCM） 12](#_Toc27144317)

[2.8 最大子段和 13](#_Toc27144318)

[2.9 最小二叉搜索树（OBST） 14](#_Toc27144319)

[2.10 树形DP-二叉树 15](#_Toc27144320)

[2.11 树形DP初步-真树 16](#_Toc27144321)

[3 图论 17](#_Toc27144322)

[3.1 Dijkstra 17](#_Toc27144323)

[3.2 Floyd 19](#_Toc27144324)

[3.3 网络流 20](#_Toc27144325)

[3.4 二分图匹配 22](#_Toc27144326)

[3.5 判定二分图 23](#_Toc27144327)

[4 计算几何 24](#_Toc27144328)

[4.1 凸包、面积、周长 24](#_Toc27144329)

[4.2 线段交点 26](#_Toc27144330)

[\*1. 基本函数 28](#_Toc27144331)

[(1) 点的定义 28](#_Toc27144332)

[(2) 线段的定义 29](#_Toc27144333)

[(3) 两点间距离 29](#_Toc27144334)

[(4) 判断：线段相交 29](#_Toc27144335)

[(5) 判断：直线和段相交 30](#_Toc27144336)

[(6) 点到直线距离 30](#_Toc27144337)

[(7) 点到线段距离 30](#_Toc27144338)

[(8) 计算多边形面积 31](#_Toc27144339)

[(9) 判断点在线段上 31](#_Toc27144340)

[(10) 判断点在凸多边形内 31](#_Toc27144341)

[(11) 判断点在任意多边形内 判断点在任意多边形内 32](#_Toc27144342)

[(12) 判断凸多边形 33](#_Toc27144343)

[\*2. 凸包 33](#_Toc27144344)

[\*3. 平面最近点对 34](#_Toc27144345)

[\*4. 旋转卡壳 36](#_Toc27144346)

[(1) 求解平面最远点对 36](#_Toc27144347)

[(2) 求解平面点集最大三角形 38](#_Toc27144348)

[\*5. 平面点逆时针排序 39](#_Toc27144349)

[\*6. 平面线段交点 40](#_Toc27144350)

[5 FFT 43](#_Toc27144351)

[5.1 多项式乘 43](#_Toc27144352)

[5.2 高精乘 44](#_Toc27144353)

[\*1. 求高精度乘法 46](#_Toc27144354)

[\*2. 给出n条线段长度，问任取3根组成三角形的概率 49](#_Toc27144355)

[6 字符串 50](#_Toc27144356)

[6.1 KMP 50](#_Toc27144357)

[6.2 AC自动机 51](#_Toc27144358)

[\*1. 经典题目：POJ 3167 53](#_Toc27144359)

[\*2. 扩展 KMP 56](#_Toc27144360)

[\*3. Manacher 最长回文子串 57](#_Toc27144361)

# 1 排序

## 1.1 快速排序

#include<cstdio>

#include<algorithm>

const int N = 1000010;

int n, m, a[N];

void qs(int dep, int l, int r) {

    int i = l, j = r, x = a[l + r + 1 >> 1];

    while (i <= j) {

        while (a[i] < x) ++i;

        while (a[j] > x) --j;

        if (i <= j) {

            std::swap(a[i], a[j]);

            i++, j--;

        }

    }

    if (dep == 2) {

        for (; i <= r; ++i) printf("%d ", a[i]);

        exit(0);

    }

    if (l < j) qs(dep + 1, l, j);

    if (i < r) qs(dep + 1, i, r);

}

int main() {

    scanf("%d", &n);

    for (int i = 1; i <= n; ++i) scanf("%d", a + i);

    qs(1, 1, n);

    return 0;

}

## 1.2 归并排序（）

#include<stdio.h>

int a[1000005],l[1000005],r[1000005];

long long cnt;

void mergesort(int lo,int hi){

    int i,j,k;

    if(hi-lo<2) return;

    int mi=(lo+hi)>>1;

    mergesort(lo,mi),mergesort(mi,hi);

    for(i=lo;i<mi;i++) l[i]=a[i];

    for(i=mi;i<hi;i++) r[i]=a[i];

    i=lo,j=mi,k=lo;

    while(i<mi&&j<hi){

        if(l[i]<=r[j]) a[k++]=l[i++];

        else {a[k++]=r[j++];cnt+=mi-i;}

    }

    while(i<mi) a[k++]=l[i++];

    while(j<hi) a[k++]=r[j++];

}

int main(){

    int n;

    scanf("%d",&n);

    for(int i=0;i<n;i++){

        scanf("%d",a+i);

    }

    mergesort(0,n);//[0,n)

    printf("%lld\n",cnt);

}

## 1.3 堆排序

#include <cstdio>

#include <queue>//优先队列

using namespace std;

priority\_queue <int, vector<int>, greater<int> > q;

int main() {

int n, x, y;

long long ans;

while (~scanf("%d", &n)) {

ans = 0;

while (!q.empty()) q.pop();

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

scanf("%d", &x);

q.push(x);

}

for (int i = 0; i < n - 1; i++) {

x = q.top(); q.pop();

y = q.top(); q.pop();

ans += x + y;

q.push(x + y);

}

printf("%lld\n", ans);

}

}

# 2 动态规划

## 2.1 背包

#include<cstdio>

#include<algorithm>

#include<cstring>

using namespace std;

int c,w,m;

int f[100005];

int n,v,ans;

void comp(int c,int w){

    f[c]=max(f[c],f[0]+w);

    for(int i=c+1;i<=v;i++){

        if(f[f[i-c]])f[i]=max(f[i],f[i-c]+w);

    }

}

void zero(int c,int w){

    for(int i=v;i>=c+1;i--){

        if(f[i-c])f[i]=max(f[i],f[i-c]+w);

    }

    f[c]=max(f[c],f[0]+w);

}

void multi(int c,int w,int m){

    if(c\*m>=v){

        comp(c,w);

    }

    int k=1;

    while(k<m){

        zero(k\*c,k\*w);

        m-=k;

        k<<=1;

    }

    zero(c\*m,w\*m);

}

int main(){

    while(~scanf("%d%d",&n,&v)){

        ans=0;

        memset(f,0,sizeof(f));

        for(int i=1;i<=n;i++){

            scanf("%d%d",&c,&w);

            //multi(c,w,m);

            zero(c,w);

            //comp(c,w);

        }

        for(int i=1;i<=v;i++){

            ans=max(ans,f[i]);

        }

        printf("%d\n",ans);

    }

}

## 2.2 最长上升子序列（LIS）

#include <iostream>

#include <random>

using namespace std;

const int MAX = 99999999;

int ar[MAX], br[MAX];

int Search(int num, int left, int right) {

int mid;

while (left <= right) {

mid = (left + right) / 2;

if (num > br[mid])

left = mid + 1;

else

right = mid - 1;

}

return left;

}

int DP(int n) {

int i, len, pos;

br[1] = ar[1];

len = 1;

for (i = 2; i <= n; i++) {

if (ar[i] > br[len]) {

len = len + 1;

br[len] = ar[i];

}

else {

pos = Search(ar[i], 1, len);

br[pos] = ar[i];

}

}

return len;

}

int main() {

int n, i;

scanf("%d", &n);

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

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

printf("%d\n", DP(n));

}

//二分

#include<cstdio>

#include<algorithm>

#include<cstring>

using namespace std;

#define LL long long

#define INF 0x3f3f3f3f

#define MAXN 40005

int n;

int a[MAXN];

int d[MAXN];

int main(){

scanf("%d",&n);

int len=1;

scanf("%d",&a[1]);

d[1]=a[1];

for(int i=2;i<=n;i++){

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

if(a[i]>d[len])

len++,d[len]=a[i];

else{

int pos=lower\_bound(d+1,d+len+1,a[i])-d;

d[pos]=a[i];

}

}

printf("%d\n",len);

return 0;

}

## 2.3 最长公共子序列（LCS）

#include <stdio.h>

#include <string.h>

#define MAXLEN 50

void LCSLength(char \*x, char \*y, int m, int n, int c[][MAXLEN]) {

int i, j;

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

c[i][0] = 0;

for (j = 1; j <= n; j++)

c[0][j] = 0;

for (i = 1; i <= m; i++) {

for (j = 1; j <= n; j++) {

if (x[i - 1] == y[j - 1]) { //仅仅去掉了对b数组的使用，其它都没变

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

}

else if (c[i - 1][j] >= c[i][j - 1]) {

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

}

else {

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

}

}

}

}

/\*

void PrintLCS(int c[][MAXLEN], char \*x, int i, int j) { //非递归版PrintLCS

static char s[MAXLEN];

int k=c[i][j];

s[k]='\0';

while(k>0){

if(c[i][j]==c[i-1][j]) i--;

else if(c[i][j]==c[i][j-1]) j--;

else{

s[--k]=x[i-1];

i--;j--;

}

}

printf("%s",s);

}

\*/

void PrintLCS(int c[][MAXLEN], char \*x, int i, int j) {

if (i == 0 || j == 0)

return;

if (c[i][j] == c[i - 1][j]) {

PrintLCS(c, x, i - 1, j);

}

else if (c[i][j] == c[i][j - 1])

PrintLCS(c, x, i, j - 1);

else {

PrintLCS(c, x, i - 1, j - 1);

printf("%c ", x[i - 1]);

}

}

int main() {

char x[MAXLEN] = { "ABCBDAB" };

char y[MAXLEN] = { "BDCABA" };

//char x[MAXLEN] = {"ACCGGTCGAGTGCGCGGAAGCCGGCCGAA"}; //算法导论上222页的DNA的碱基序列匹配

//char y[MAXLEN] = {"GTCGTTCGGAATGCCGTTGCTCTGTAAA"};

int c[MAXLEN][MAXLEN]; //仅仅使用一个c表

int m, n;

m = strlen(x);

n = strlen(y);

LCSLength(x, y, m, n, c);

PrintLCS(c, x, m, n);

return 0;

}

## 2.4 最长上升公共子序列（LCIS）

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

int n,m,ans;

int A[10005],B[10005];

short int dp[10005];//dp[i][j]：以A串的前i个整数与B串的前j个整数且以B[j]为结尾构成的LCIS的长度

int main(){

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

memset(dp,0,sizeof(dp));

ans=0;

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

scanf("%d",A+i);

}

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

scanf("%d",B+i);

}

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

int mmax=0;

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

if(A[i]==B[j]){

dp[j]=mmax+1;

}

if(A[i]>B[j]){

mmax=max(mmax,(int)dp[j]);

}

}

}

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

if(ans<dp[j]) ans=dp[j];

}

printf("%d\n",ans);

}

}

## 2.5 区间最值查询（RMQ）

#include<cstdio>

#include<cstring>

#include<cmath>

using namespace std;

int num[200005];//用来存储数组里的每一个数

int logn[200005];//预处理不大于每个数2的n次方

int f[200005][35];//f[i][j]记录第i个数向左数，长度为j的区间中的最大值

int gg[36];//预处理2的n次方为多少

int n, m;

void pre(){

logn[1] = 0;

logn[2] = 1;

for (int i = 3; i <= n; i++){

logn[i] = logn[i / 2] + 1;

}

}

void pre1(){

int j = 0;

gg[0] = 1;

int cnt = 1;

while (cnt <= n){

cnt = cnt \* 2;

j++;

gg[j] = cnt;

}

}

int maxn(int a, int b){

return a > b ? a : b;

}

void deal()//预处理每个区间的最大值

{

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

int j = 0;

int k = i - 1;

f[i][0] = num[i];//长度为2的一次方的区间最大值为那个数本身

while (k > 0){

j++;

f[i][j] = maxn(f[i][j - 1], f[i - gg[j - 1]][j - 1]);//每个长度为2^n的区间最大值为两个长度为2^(n-1)的区间所合成

k = k - gg[j];//保证不会处理到超过区间长度

}

}

}

int ques(int l, int r){

int ll = r - l + 1;//区间长度

int lx = logn[ll];

int res = maxn(f[l + gg[lx] - 1][lx], f[r][lx]);//合成两小区间到大区间

return res;

}

int main(){

scanf("%d", &n);

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

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

}

scanf("%d", &m);

pre();

pre1();

deal();//三发预处理

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

int l, r;

scanf("%d%d", &l, &r);

int res = ques(l, r);

printf("%d\n", res);

}

return 0;

}

## 2.6 N重ALS

#include<iostream>

using namespace std;

#define INF 0xFFFFFF;

int n, m, mini;

int p[507][507], t[507][507], T[507][507];

int main()

{

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

{

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

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

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

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

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

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

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

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

T[i][j] = INF;

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

T[i][0] = p[i][0];

for (int k = 1; k < m; k++)

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

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

if (T[i][k] > T[j][k - 1] + p[i][k] + t[j][i])

T[i][k] = T[j][k - 1] + p[i][k] + t[j][i];

mini = INF;

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

if (mini > T[i][m - 1])

mini = T[i][m - 1];

printf("%d\n", mini);

}

}

## 2.7 矩阵链乘法（MCM）

#include <cstdio>

#include <cstring>

#define INF 0xFFFFFF;

using namespace std;

int divPos[305][305];

int p[305];

int m[305][305];

void printResult(int i, int j)

{

if (i == j)

printf("A%d", i);

else

{

printf("(");

printResult(i, divPos[i][j]);

printResult(divPos[i][j] + 1, j);

printf(")");

}

}

int main()

{

int n;

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

{

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

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

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

m[i][i] = 0;

for (int len = 2; len <= n; len++)

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

{

int j = i + len - 1;

m[i][j] = INF;

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

{

int temp = m[i][k] + m[k + 1][j] + p[i - 1] \* p[k] \* p[j];

if (m[i][j] >= temp)

{

m[i][j] = temp;

divPos[i][j] = k;

}

}

}

printf("%d\n", m[1][n]);

printResult(1, n);

printf("\n");

}

}

## 2.8 最大子段和

#include <iostream>

using namespace std;

int stock[1000007], n;

long long profit, present;

int main()

{

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

{

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

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

profit = present = 0;

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

if (present < 0)

present = 0;

present += stock[i];

if (present > profit)

profit = present;

}

printf("%lld\n", profit);

}

}

## 2.9 最小二叉搜索树（OBST）

#include <iostream>

#include <algorithm>

#include <cmath>

#include <cfloat>

using namespace std;

int n;

const int maxn = 510;

double e[maxn][maxn];

double w[maxn][maxn];

double p[maxn];

double q[maxn];

int main(){

while (~scanf("%d", &n)){

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

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

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

scanf("%lf", &q[i]);

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

e[i][i - 1] = q[i - 1];

w[i][i - 1] = q[i - 1];

}

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

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

int j = i + l - 1;

e[i][j] = DBL\_MAX;

w[i][j] = w[i][j - 1] + p[j] + q[j];

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

double t = e[i][r - 1] + e[r + 1][j] + w[i][j];

if (t < e[i][j]) {

e[i][j] = t;

}

}

}

} printf("%0.0lf\n", e[1][n]);

}

return 0;

}

## 2.10 树形DP-二叉树

最长链为这棵二叉树中一条最长的简单路径，即不经过重复结点的一条路径。可以容易证明，二叉树中最长链的起始、结束结点均为叶子结点。

现给出一棵N(N<=100000)个结点二叉树，问这棵二叉树中最长链的长度为多少，保证了1号结点为二叉树的根。

**输入**

输入第1行为包含了一个正整数N，为这棵二叉树的结点数，结点标号由1至N。

接下来N行，这N行中的第i行包含两个正整数l[i], r[i]，表示了结点i的左儿子与右儿子编号。如果l[i]为0，表示结点i没有左儿子，同样地，如果r[i]为0则表示没有右儿子。

#include <iostream>

#include <cstdio>

#include<algorithm>

using namespace std;

int n;

int son[100001][2];

int dp[100001];

int ans;

void find(int a){

if (son[a][0])

find(son[a][0]);

if (son[a][1])

find(son[a][1]);

dp[a]=max(dp[son[a][0]],dp[son[a][1]])+1;

//int b=dp[son[a][0]]+dp[son[a][1]];

//if (b>ans) ans=b;

}

void found(int a){

int b=dp[son[a][0]]+dp[son[a][1]];

ans=max(b,ans);

}

int main(){

scanf("%d",&n);

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

scanf("%d%d",&son[i][0],&son[i][1]);

if (son[1][0])

find(son[1][0]);

if (son[1][1])

find(son[1][1]);

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

found(i);

printf("%d",ans);

}

## 2.11 树形DP初步-真树

每只白兔都有它们自己唯一的整数编号（范围在1到N之间），并且对应一个参加聚会所得的开心值。为了使每个参加聚会的白兔都巨开心，老白兔想让每只白兔和他的上一代白兔不会同时参加聚会。求参加聚会的白兔获得的最大总开心值。

输入的第一行是一个整数N，1<= N <= 6000。以下的N行是对应的N个白兔的开心值（开心值是一个从-128到127之间的整数）。接着是白兔的家族树，树的每一行格式如下： 每行输入一对整数L,K。表示第K个白兔是第L个白兔的上一代。 输入以0 0表示结束

#include <cstdio>

#include <cstring>

#include <iostream>

#define maxn 6007

using namespace std;

int n;

int dp[maxn][2],father[maxn];

void tree\_dp(int node){

int i;

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

if(father[i]==node){

tree\_dp(i);

dp[node][1]+=dp[i][0];

dp[node][0]+=max(dp[i][0],dp[i][1]);

}

}

}

int main(){

int a,b;

int root=0;

scanf("%d",&n);

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

scanf("%d",&dp[i][1]);

}

while(scanf("%d%d",&a,&b)==2){

if(a==0&&b==0) break;

father[a]=b;

}

root=b;

while(father[root]){

root=father[root];

}

//从根节点开始dp

tree\_dp(root);

printf("%d\n",max(dp[root][0],dp[root][1]));

}

# 3 图论

## 3.1 Dijkstra

#include<cstdio>

#include<queue>

#include<vector>

#include<cstring>

using namespace std;

const int INF=0x3f3f3f3f;

const int MAXN=100010;

struct qnode{

    int v;

    int c;

    qnode(int \_v=0,int \_c=0):v(\_v),c(\_c){}

    bool operator <(const qnode &r)const{

        return c>r.c;

    }

};

struct Edge{

    int v,cost;

    Edge(int \_v=0,int \_cost=0):v(\_v),cost(\_cost){}

};

vector<Edge>E[MAXN];

bool vis[MAXN];

int dist[MAXN];

//int pre[MAXN];

void dj(int n,int start){

    memset(vis,false,sizeof(vis));

    //memset(pre,-1,sizeof(pre));

    for(int i=1;i<=n;i++)dist[i]=INF;

    priority\_queue<qnode>que;

    while(!que.empty())que.pop();

    dist[start]=0;

    que.push(qnode(start,0));

    qnode tmp;

    while(!que.empty()){

        tmp=que.top();

        que.pop();

        int u=tmp.v;

        if(vis[u])continue;

        vis[u]=true;

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

            int v=E[tmp.v][i].v;

            int cost=E[u][i].cost;

            if(!vis[v]&&dist[v]>dist[u]+cost){

                dist[v]=dist[u]+cost;

                que.push(qnode(v,dist[v]));

                //pre[v]=u;

            }

        }

    }

}

void addedge(int u,int v,int w){

    E[u].push\_back(Edge(v,w));

}

int main(){

    int n,m,s,t,u,v,w;

    scanf("%d%d%d%d",&n,&m,&s,&t);

    for(int i=1;i<=m;i++){

        scanf("%d%d%d",&u,&v,&w);

        addedge(u,v,w);

        addedge(v,u,w);

    }

    dj(n,s);

    printf("%d\n",dist[t]);

    /\*逆序输出路径

    int node=t;

    printf("%d ",node)

    while(pre[node]!=-1){

        printf("%d ",pre[node]);

        node=pre[node];

    }

    printf("\n");\*/

}

## 3.2 Floyd

#include<cstdio>

#include<algorithm>

#include<cstring>

using namespace std;

#define INF 0x3f3f3f3f

int map[1005][1005];

int main(){

    int k,i,j,n,m;

    scanf("%d%d",&n,&m);

    for (i=1;i<=n;i++){

        for (j=1;j<=n;j++){

            if(i==j) map[i][j] = 0;

            else map[i][j] = INF;

        }

    }

    int a,b,c;

    for (i=1;i<=m;i++){

        scanf("%d%d%d",&a,&b,&c);

        map[a][b]=c;

    }

    for (k = 1; k <= n; k++){

        for (i = 1; i <= n; i++){

            for (j = 1; j <= n; j++){

                if (map[i][j] > map[i][k] + map[k][j]){

                    map[i][j] = map[i][k] + map[k][j];

                }

            }

        }

    }

    for(i=1;i<=n;i++){

        for(j=1;j<=n;j++){

            printf("%d\n",map[i][j]);

        }

    }

    return 0;

}

## 3.3 网络流

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <queue>

typedef long long LL;

using namespace std;

const int MAXN = 100000 + 50;

const int INF = 1e8;

int head[MAXN], dist[MAXN], vis[MAXN];

int cur[MAXN];

int n, m, q;

int top = 0;

struct Edge {

    int to, cap, flow, next;

}edge[MAXN \* 20];

void init() {

    top = 0;

    memset(head, -1, sizeof(head));

    memset(vis, 0, sizeof(vis));

}

void addedge(int a, int b, int c) {

    Edge E1 = {b, c, 0, head[a]};

    edge[top] = E1;

    head[a] = top++;

    Edge E2 = {a, 0, 0, head[b]};

    edge[top] = E2;

    head[b] = top++;

}

bool BFS(int st, int ed) {

    memset(dist, -1, sizeof(dist));

    memset(vis, 0, sizeof(vis));

    queue<int> que;

    que.push(st);

    vis[st] = 1;

    dist[st] = 0;

    while(!que.empty()) {

        int u = que.front();

        que.pop();

        for(int i = head[u]; i != -1; i = edge[i].next) {

            Edge E = edge[i];

            if(!vis[E.to] && E.cap > E.flow) {

                dist[E.to] = dist[u] + 1;

                vis[E.to] = 1;

                if(E.to == ed) return true;

                que.push(E.to);

            }

        }

    }

    return false;

}

int DFS(int x, int a, int ed) {

    if(x == ed || a == 0) return a;

    int flow = 0, f;

    for(int& i = cur[x]; i != -1; i = edge[i].next) {

        Edge& E = edge[i];

        if(dist[E.to] == dist[x] + 1 && (f = DFS(E.to, min(a, E.cap - E.flow), ed)) > 0) {

            E.flow += f;

            edge[i^1].flow -= f;

            flow += f;

            a -= f;

            if(a == 0) break;

        }

    }

    return flow;

}

int Maxflow(int st, int ed) {

    int flow = 0;

    while(BFS(st, ed)) {

        memcpy(cur, head, sizeof(head));

        flow += DFS(st, INF, ed);

    }

    return flow;

}

int main(){

    init();

    int i,j;

    int n,m,a,b,c;

    scanf("%d%d", &n, &m);

    for (i = 0; i < m; ++i){

        scanf("%d%d%d", &a, &b, &c);

        addedge(a,b,c);

        addedge(b,a,c);

    }

    int ans = Maxflow(1, n);

    printf("%d\n", ans);

    return 0;

}

## 3.4 二分图匹配

#include<cstdio>

#include<vector>

#include<cstring>

#include<algorithm>

using namespace std;

vector<int>a[200005];

int match[200005];

bool vis[200005];

void add\_edge(int u,int v){

//a[v].push\_back(u);

a[u].push\_back(v);

}

bool dfs(int v){

for(int i=0;i<a[v].size();i++){

int u=a[v][i],m=match[u];

if(!vis[u]){

vis[u]=true;

if(m==-1||dfs(m)){

match[u]=v;

//match[v]=u;

return true;

}

}

}

return false;

}

int main(){

int n,m,res=0;

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

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

int T;

scanf("%d", &T);

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

int num;

scanf("%d", &num);

add\_edge(i,num);

}

}

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

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

memset(vis,0,sizeof(vis));

//if(match[i]==-1)

res+=dfs(i);

}

printf("%d\n",res);

}

## 3.5 判定二分图

# include<iostream>

# include<vector>

using namespace std;

const int MAXV = 1005;

int V,E;

vector<int> G[MAXV];

//新建了一个图，G数组中的每一个元素都是一个顶点，对应着一个vector，其中包含了它指向的顶点。

//没有设定边的属性，仅用一个int表示指向的顶点。

int color[MAX\_V];//记录顶点的颜色（只有两种，1或-1）

bool dfs(int v,int c){

    color[v] = c;//将顶点v染成颜色c

    for (int i = 0; i < G[v].size(); i++){

        if (color[G[v][i]] == c)

            return false;

        /\*首先需要理解G[v][i]的含义！是指顶点v的第i条边指向的顶点！

        上面的表示，如果相邻的顶点同色，就剪掉这一枝，返回false\*/

        if (color[G[v][i]] == 0 && !dfs(G[v][i], -c))

            return false;

        //如果相邻的顶点还没有染色就把它染成-c

    }

    return true;

    //如果都染色了返回true

}

void solve(){

    for (int i = 0; i < V; i++){

        if (color[i] == 0){

            if (!dfs(i, 1)){

                printf("No\n");

                return;

            }

        }

    }

    printf("Yes\n");

}

int main(){

    cin >> V >> E;

    for (int i = 0; i < E; i++){

        int s, t;

        cin >> s >> t;

        G[s].push\_back(t);//从s向t连边

        G[t].push\_back(s);//因为是无向图反过来再连接一次

    }

    solve();

    return 0;

}

# 4 计算几何

## 4.1 凸包、面积、周长

#include<algorithm>

#include<cmath>

#include<cstdio>

using namespace std;

const double eps=1e-8,pi=acos(-1.0);

int n;

int dcmp(double x){

    if(fabs(x)<eps)return 0;

    else return x>0?1:-1;

}

struct Point{

    double x,y;

    inline Point(double x=0,double y=0):x(x),y(y){}

}p[100005],ch[100005];

bool myCmp(Point A,Point B){

    if(A.x!=B.x)return A.x<B.x;

    else return A.y<B.y;

}

Point operator + (Point A,Point B){

    return Point(A.x+B.x,A.y+B.y);

}

Point operator - (Point A,Point B){

    return Point(A.x-B.x,A.y-B.y);

}

double operator ^ (Point A,Point B){

    return A.x\*B.y-A.y\*B.x;

}

bool operator == (const Point &A,const Point &B){

    return dcmp(A.x-B.x)==0&&dcmp(A.y-B.y)==0;

}

int ConvexHull(){

    sort(p,p+n,myCmp);

    int m=0;

    for(int i=0;i<n;i++){

        while(m>1&&dcmp((ch[m-1]-ch[m-2])^(p[i]-ch[m-2]))<=0)m--;

        ch[m++]=p[i];

    }

    int k=m;

    for(int i=n-2;i>=0;i--){

        while(m>k&&dcmp((ch[m-1]-ch[m-2])^(p[i]-ch[m-2]))<=0)m--;

        ch[m++]=p[i];

    }

    if(n>1)m--;

    return m;

}

double Dis(Point A,Point B){

    return sqrt((A.x-B.x)\*(A.x-B.x)+(A.y-B.y)\*(A.y-B.y));

}

int main(){

    int m;

    double ans;

    scanf("%d",&n);

    for(int i=0;i<n;i++){

        int a,b;

        scanf("%d%d",&a,&b);

        p[i]=Point(a,b);

    }

    m=ConvexHull();

    ans=0.0;

    for(int i=0;i<m-1;i++){

        ans+=Dis(ch[i],ch[i+1]);

    }

    ans+=Dis(ch[m-1],ch[0]);

    printf("%.2f ",ans);

    ans=0.0;

    ch[m]=ch[0];

    for(int i=0;i<m;i++){

        ans+=ch[i]^ch[i+1];

    }

    ans=ans/2;

    printf("%.2f\n",ans);

}

## 4.2 线段交点

#include<algorithm>

#include<cmath>

#include<cstdio>

using namespace std;

const double eps=1e-8,pi=acos(-1.0);

int n;

int dcmp(double x){

    if(fabs(x)<eps)return 0;

    else return x>0?1:-1;

}

struct Point{

    double x,y;

    inline Point(double x=0,double y=0):x(x),y(y){}

}line1[5],line2[5];

bool myCmp(Point A,Point B){

    if(A.x!=B.x)return A.x<B.x;

    else return A.y<B.y;

}

Point operator + (Point A,Point B){

    return Point(A.x+B.x,A.y+B.y);

}

Point operator - (Point A,Point B){

    return Point(A.x-B.x,A.y-B.y);

}

double operator ^ (Point A,Point B){

    return A.x\*B.y-A.y\*B.x;

}

bool operator == (const Point &A,const Point &B){

    return dcmp(A.x-B.x)==0&&dcmp(A.y-B.y)==0;

}

int check(){//线段相交

    double a=((line1[1]-line2[1])^(line2[2]-line2[1]))\*((line1[2]-line2[1])^(line2[2]-line2[1]));

    double b=((line2[2]-line1[1])^(line1[2]-line1[1]))\*((line2[1]-line1[1])^(line1[2]-line1[1]));

    if(a<=0&&b<=0)return 1;

    return 0;

}

double abscal(Point a){

    return sqrt(a.x\*a.x+a.y\*a.y);

}

Point get\_cross(){

    Point base=line2[2]-line2[1];

    double d1=abscal(base^(line1[1]-line2[1]));

    double d2=abscal(base^(line1[2]-line2[1]));

    double t=d1/(d1+d2);

    Point tmp=line1[2]-line1[1];

    tmp.x\*=t;

    tmp.y\*=t;

    return line1[1]+tmp;

}

double Dis(Point A,Point B){

    return sqrt((A.x-B.x)\*(A.x-B.x)+(A.y-B.y)\*(A.y-B.y));

}

int main(){

    int a,b,c,d;

    while(~scanf("%d%d%d%d",&a,&b,&c,&d)){

        line1[1]=Point(a,b);

        line1[2]=Point(c,d);

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

        line2[1]=Point(a,b);

        line2[2]=Point(c,d);

        sort(line1+1,line1+3,myCmp);

        sort(line2+1,line2+3,myCmp);

        if((line1[2].y-line1[1].y)\*(line2[2].x-line2[1].x)==(line1[2].x-line1[1].x)\*(line2[2].y-line2[1].y)){

            if(line1[2]==line2[1]){

                printf("%lf %lf\n",line1[2].x,line1[2].y);

            }

            else printf("none\n");

        }

        else{

            if(check()){

                Point tmp=get\_cross();

                printf("%lf %lf\n",tmp.x,tmp.y);

            }

            else printf("none\n");

        }

    }

}

## \*1. 基本函数

### (1) 点的定义

const double eps = 1e-8;

const double PI = acos(-1.0);

int sgn(double x)

{

if (fabs(x) < eps)return 0;

if (x < 0)return -1;

else return 1;

}

struct Point

{

double x, y;

Point() {}

Point(double \_x, double \_y)

{

x = \_x; y = \_y;

}

Point operator -(const Point &b)const

{

return Point(x - b.x, y - b.y);

}

//叉积

double operator ^(const Point &b)const

{

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

}

//点积

double operator \*(const Point &b)const

{

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

}

//绕原点旋转角度B（弧度值），后x,y的变化

void transXY(double B)

{

double tx = x, ty = y;

x = tx \* cos(B) - ty \* sin(B);

y = tx \* sin(B) + ty \* cos(B);

}

};

### (2) 线段的定义

struct Line

{

Point s, e;

Line() {}

Line(Point \_s, Point \_e)

{

s = \_s; e = \_e;

}

//两直线相交求交点

//第一个值为0表示直线重合，为1表示平行，为0表示相交,为2是相交

//只有第一个值为2时，交点才有意义

pair<int, Point> operator &(const Line &b)const

{

Point res = s;

if (sgn((s - e) ^ (b.s - b.e)) == 0)

{

if (sgn((s - b.e) ^ (b.s - b.e)) == 0)

return make\_pair(0, res);//重合

else return make\_pair(1, res);//平行

}

double t = ((s - b.s) ^ (b.s - b.e)) / ((s - e) ^ (b.s - b.e));

res.x += (e.x - s.x)\*t;

res.y += (e.y - s.y)\*t;

return make\_pair(2, res);

}

};

### (3) 两点间距离

//\*两点间距离

double dist(Point a, Point b)

{

return sqrt((a - b)\*(a - b));

}

### (4) 判断：线段相交

//\*判断线段相交

bool inter(Line l1, Line l2)

{

return

max(l1.s.x, l1.e.x) >= min(l2.s.x, l2.e.x) &&

max(l2.s.x, l2.e.x) >= min(l1.s.x, l1.e.x) &&

max(l1.s.y, l1.e.y) >= min(l2.s.y, l2.e.y) &&

max(l2.s.y, l2.e.y) >= min(l1.s.y, l1.e.y) &&

sgn((l2.s - l1.e) ^ (l1.s - l1.e))\*sgn((l2.e - l1.e) ^ (l1.s - l1.e)) <= 0 &&

sgn((l1.s - l2.e) ^ (l2.s - l2.e))\*sgn((l1.e - l2.e) ^ (l2.s - l2.e)) <= 0;

}

### (5) 判断：直线和段相交

//判断直线和线段相交

bool Seg\_inter\_line(Line l1, Line l2) //判断直线l1和线段l2是否相交

{

return sgn((l2.s - l1.e) ^ (l1.s - l1.e))\*sgn((l2.e - l1.e) ^ (l1.s - l1.e)) <= 0;

}

### (6) 点到直线距离

//点到直线距离

//返回为result,是点到直线最近的点

Point PointToLine(Point P, Line L)

{

Point result;

double t = ((P - L.s)\*(L.e - L.s)) / ((L.e - L.s)\*(L.e - L.s));

result.x = L.s.x + (L.e.x - L.s.x)\*t;

result.y = L.s.y + (L.e.y - L.s.y)\*t;

return result;

}

### (7) 点到线段距离

//点到线段的距离

//返回点到线段最近的点

Point NearestPointToLineSeg(Point P, Line L)

{

Point result;

double t = ((P - L.s)\*(L.e - L.s)) / ((L.e - L.s)\*(L.e - L.s));

if (t >= 0 && t <= 1)

{

result.x = L.s.x + (L.e.x - L.s.x)\*t;

result.y = L.s.y + (L.e.y - L.s.y)\*t;

}

else

{

if (dist(P, L.s) < dist(P, L.e))

result = L.s;

else result = L.e;

}

return result;

}

### (8) 计算多边形面积

//计算多边形面积

//点的编号从0~n-1

double CalcArea(Point p[], int n)

{

double res = 0;

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

res += (p[i] ^ p[(i + 1) % n]) / 2;

return fabs(res);

}

### (9) 判断点在线段上

//\*判断点在线段上

bool OnSeg(Point P, Line L)

{

return

sgn((L.s - P) ^ (L.e - P)) == 0 &&

sgn((P.x - L.s.x) \* (P.x - L.e.x)) <= 0 &&

sgn((P.y - L.s.y) \* (P.y - L.e.y)) <= 0;

}

### (10) 判断点在凸多边形内

//\*判断点在凸多边形内

//点形成一个凸包，而且按逆时针排序（如果是顺时针把里面的<0改为>0）

//点的编号:0~n-1

//返回值：

//-1:点在凸多边形外

//0:点在凸多边形边界上

//1:点在凸多边形内

int inConvexPoly(Point a, Point p[], int n)

{

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

{

if (sgn((p[i] - a) ^ (p[(i + 1) % n] - a)) < 0)return -1;

else if (OnSeg(a, Line(p[i], p[(i + 1) % n])))return 0;

}

return 1;

}

### (11) 判断点在任意多边形内 判断点在任意多边形内

//\*判断点在任意多边形内

//射线法，poly[]的顶点数要大于等于3,点的编号0~n-1

//返回值

//-1:点在凸多边形外

//0:点在凸多边形边界上

//1:点在凸多边形内

int inPoly(Point p, Point poly[], int n)

{

int cnt;

Line ray, side;

cnt = 0;

ray.s = p;

ray.e.y = p.y;

ray.e.x = -100000000000.0;//-INF,注意取值防止越界

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

{

side.s = poly[i];

side.e = poly[(i + 1) % n];

if (OnSeg(p, side))return 0;

//如果平行轴则不考虑

if (sgn(side.s.y - side.e.y) == 0)

continue;

if (OnSeg(side.s, ray))

{

if (sgn(side.s.y - side.e.y) > 0)cnt++;

}

else if (OnSeg(side.e, ray))

{

if (sgn(side.e.y - side.s.y) > 0)cnt++;

}

else if (inter(ray, side))

cnt++;

}

if (cnt % 2 == 1)return 1;

else return -1;

}

### (12) 判断凸多边形

//判断凸多边形

//允许共线边

//点可以是顺时针给出也可以是逆时针给出

//点的编号1~n-1

bool isconvex(Point poly[], int n)

{

bool s[3];

memset(s, false, sizeof(s));

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

{

s[sgn((poly[(i + 1) % n] - poly[i]) ^ (poly[(i + 2) % n] - poly[i])) + 1] = true;

if (s[0] && s[2])return false;

}

return true;

}

## \*2. 凸包

/\*

\* 求凸包，Graham算法

\* 点的编号0~n-1

\* 返回凸包结果Stack[0~top-1]为凸包的编号

\*/

const int MAXN = 1010;

Point list[MAXN];

int Stack[MAXN], top;

//相对于list[0]的极角排序

bool \_cmp(Point p1, Point p2)

{

double tmp = (p1 - list[0]) ^ (p2 - list[0]);

if (sgn(tmp) > 0)return true;

else if (sgn(tmp) == 0 && sgn(dist(p1, list[0]) - dist(p2, list[0])) <= 0)

return true;

else return false;

}

void Graham(int n)

{

Point p0;

int k = 0;

p0 = list[0];

//找最下边的一个点

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

{

if ((p0.y > list[i].y) || (p0.y == list[i].y && p0.x > list[i].x))

{

p0 = list[i];

k = i;

}

}

swap(list[k], list[0]);

sort(list + 1, list + n, \_cmp);

if (n == 1)

{

top = 1;

Stack[0] = 0;

return;

}

if (n == 2)

{

top = 2;

Stack[0] = 0;

Stack[1] = 1;

return;

}

Stack[0] = 0;

Stack[1] = 1;

top = 2;

for (int i = 2; i < n; i++)

{

while (top > 1 && sgn((list[Stack[top - 1]] - list[Stack[top - 2]]) ^ (list[i] - list[Stack[top - 2]])) <= 0)

top--;

Stack[top++] = i;

}

}

## \*3. 平面最近点对

#include <stdio.h>

#include <string.h>

#include <algorithm>

#include <iostream>

#include <math.h>

using namespace std;

const double eps = 1e-6;

const int MAXN = 100010;

const double INF = 1e20;

struct Point

{

double x, y;

};

double dist(Point a, Point b)

{

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

}

Point p[MAXN];

Point tmpt[MAXN];

bool cmpxy(Point a, Point b)

{

if (a.x != b.x)return a.x < b.x;

else return a.y < b.y;

}

bool cmpy(Point a, Point b)

{

return a.y < b.y;

}

double Closest\_Pair(int left, int right)

{

double d = INF;

if (left == right)return d;

if (left + 1 == right)

return dist(p[left], p[right]);

int mid = (left + right) / 2;

double d1 = Closest\_Pair(left, mid);

double d2 = Closest\_Pair(mid + 1, right);

d = min(d1, d2);

int k = 0;

for (int i = left; i <= right; i++)

{

if (fabs(p[mid].x - p[i].x) <= d)

tmpt[k++] = p[i];

}

sort(tmpt, tmpt + k, cmpy);

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

{

for (int j = i + 1; j < k && tmpt[j].y - tmpt[i].y < d; j++)

{

d = min(d, dist(tmpt[i], tmpt[j]));

}

}

return d;

}

int main()

{

int n;

while (scanf("%d", &n) == 1 && n)

{

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

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

sort(p, p + n, cmpxy);

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

}

return 0;

}

## \*4. 旋转卡壳

### (1) 求解平面最远点对

struct Point

{

int x, y;

Point(int \_x = 0, int \_y = 0)

{

x = \_x; y = \_y;

}

Point operator -(const Point &b)const

{

return Point(x - b.x, y - b.y);

}

int operator ^(const Point &b)const

{

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

}

int operator \*(const Point &b)const

{

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

}

void input()

{

scanf("%d%d", &x, &y);

}

};

//距离的平方

int dist2(Point a, Point b)

{

return (a - b)\*(a - b);

}

//\*\*\*\*\*\*二维凸包，int\*\*\*\*\*\*\*\*\*\*\*

const int MAXN = 50010;

Point list[MAXN];

int Stack[MAXN], top;

bool \_cmp(Point p1, Point p2)

{

int tmp = (p1 - list[0]) ^ (p2 - list[0]);

if (tmp > 0)return true;

else if (tmp == 0 && dist2(p1, list[0]) <= dist2(p2, list[0]))

return true;

else return false;

}

void Graham(int n)

{

Point p0;

int k = 0;

p0 = list[0];

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

if (p0.y > list[i].y || (p0.y == list[i].y && p0.x > list[i].x))

{

p0 = list[i];

k = i;

}

swap(list[k], list[0]);

sort(list + 1, list + n, \_cmp);

if (n == 1)

{

top = 1;

Stack[0] = 0;

return;

}

if (n == 2)

{

top = 2;

Stack[0] = 0; Stack[1] = 1;

return;

}

Stack[0] = 0; Stack[1] = 1;

top = 2;

for (int i = 2; i < n; i++)

{

while (top > 1 && ((list[Stack[top - 1]] - list[Stack[top - 2]]) ^ (list[i] - list[Stack[top - 2]])) <= 0)

top--;

Stack[top++] = i;

}

}

//旋转卡壳，求两点间距离平方的最大值

int rotating\_calipers(Point p[], int n)

{

int ans = 0;

Point v;

int cur = 1;

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

{

v = p[i] - p[(i + 1) % n];

while ((v ^ (p[(cur + 1) % n] - p[cur])) < 0)

cur = (cur + 1) % n;

ans = max(ans, max(dist2(p[i], p[cur]), dist2(p[(i + 1) % n], p[(cur + 1) % n])));

}

return ans;

}

Point p[MAXN];

int main()

{

int n;

while (scanf("%d", &n) == 1)

{

for (int i = 0; i < n; i++)list[i].input();

Graham(n);

for (int i = 0; i < top; i++)p[i] = list[Stack[i]];

printf("%d\n", rotating\_calipers(p, top));

}

return 0;

}

### (2) 求解平面点集最大三角形

//旋转卡壳计算平面点集最大三角形面积

int rotating\_calipers(Point p[], int n)

{

int ans = 0;

Point v;

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

{

int j = (i + 1) % n;

int k = (j + 1) % n;

while (j != i && k != i)

{

ans = max(ans, abs((p[j] - p[i]) ^ (p[k] - p[i])));

while (((p[i] - p[j]) ^ (p[(k + 1) % n] - p[k])) < 0)

k = (k + 1) % n;

j = (j + 1) % n;

}

}

return ans;

}

Point p[MAXN];

int main()

{

int n;

while (scanf("%d", &n) == 1)

{

if (n == -1)break;

for (int i = 0; i < n; i++)list[i].input();

Graham(n);

for (int i = 0; i < top; i++)p[i] = list[Stack[i]];

printf("%.2f\n", (double)rotating\_calipers(p, top) / 2);

}

return 0;

}

## \*5. 平面点逆时针排序

#include <cstdio>

#include <string>

#include <algorithm>

using namespace std;

#define MAX 100000

#define ll long long

struct Point

{

char s[100];

int x, y;

bool operator < (const Point r) const { return ((ll)x \* (ll)r.y - (ll)r.x \* (ll)y > 0 || (ll)x \* (ll)r.y - (ll)r.x \* (ll)y == 0 && (ll)y < (ll)r.y); }

} p[MAX];

int n, x, y;

string s;

int main()

{

while (~scanf("%d", &n))

{

for (int i = 0;i < n;i++) scanf("%s%d%d", &p[i].s, &p[i].x, &p[i].y);

sort(p, p + n);

for (int i = 0;i < n;i++) printf("%s\n", p[i].s);

printf("\n");

}

}

## \*6. 平面线段交点

#include <cstdio>

#include <algorithm>

#include <vector>

#include <cmath>

using namespace std;

int cmp(double x) {

if (fabs(x) < 1e-9)return 0;

if (x > 0)return 1;

else return -1;

}

struct Point

{

double x, y;

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

{

Point p;

p.x = a.x - b.x;

p.y = a.y - b.y;

return p;

}

}t[1000007];

struct stline

{

Point a, b;

} l[507];

double det(const Point &a, const Point &b) {return a.x\*b.y - a.y\*b.x;}

bool parallel(stline a, stline b) { return !cmp(det(a.a-a.b,b.a-b.b)); }

bool line\_make\_point(stline a, stline b,Point &p)

{

if (parallel(a, b))return false;

double s1 = det(a.a - b.a, b.b - b.a);

double s2 = det(a.b - b.a, b.b - b.a);

p.x = (s1\*a.b.x - s2 \* a.a.x) / (s1 - s2);

p.y = (s1\*a.b.y - s2 \* a.a.y) / (s1 - s2);

return true;

}

bool cp(Point a,Point b)

{

int d1 = cmp(a.x - b.x);

int d2 = cmp(a.y - b.y);

if (d1 != 0)

return d1<0;

else return d2 < 0;

}

bool isSame(Point a, Point b)

{

int d1 = cmp(a.x - b.x);

int d2 = cmp(a.y - b.y);

if (d1 == 0 && d2 == 0)

return true;

else return false;

}

bool isLegal(Point a, Point b, Point p)

{

if (p.x >= min(a.x, b.x) && p.x <= max(a.x, b.x) && p.y >= min(a.y, b.y) && p.y <= max(a.y, b.y))

return true;

else return false;

}

vector<Point> vec;

int main()

{

int n, num;

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

{

vec.clear();

Point p;

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

scanf("%lf%lf%lf%lf", &l[i].a.x, &l[i].a.y, &l[i].b.x, &l[i].b.y);

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

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

{

if (line\_make\_point(l[i], l[j], p))

{

if (isLegal(l[i].a, l[i].b, p)&&isLegal(l[j].a, l[j].b, p))

vec.push\_back(p);

}

}

sort(vec.begin(), vec.end(), cp);

if (vec.empty())

printf("0\n");

else

{

num = 1;

for (int i = 1; i < vec.size(); i++)

if (!isSame(vec[i], vec[i - 1]))

num++;

printf("%d\n", num);

}

}

}

G:

#include <cstdio>

int main()

{

long long x1, y1, z1, x2, y2, z2;

long long result;

while(scanf("%lld%lld%lld%lld%lld%lld",&x1,&y1,&z1,&x2,&y2,&z2)!=EOF)

{

printf("%lld\n", x1\*x2 + y1 \* y2 + z1 \* z2);

printf("%lld %lld %lld\n", y1\*z2 - y2 \* z1, x2\*z1 - x1 \* z2, x1\*y2 - x2 \* y1);

}

}

# 5 FFT

## 5.1 多项式乘

#include <complex>

#include <cstdio>

using namespace std;

typedef complex<double> comp;

const int N = (1 << 21) + 10;

const double PI = acos(-1);

int read() {

    int x = 0; char c = getchar();

    for(; c < '0' || c > '9'; c = getchar()) ;

    for(; c >= '0' && c <= '9'; c = getchar())

        x = x \* 10 + (c & 15);

    return x;

}

int n, m, lim, r[N];

comp a[N], b[N];

void fft(comp \* a, int type) {

    for(int i = 0; i < lim; i ++)

        if(i < r[i]) swap(a[i], a[r[i]]);

    for(int i = 1; i < lim; i <<= 1) {

        comp x(cos(PI / i), type \* sin(PI / i));

        for(int j = 0; j < lim; j += (i << 1)) {

            comp y(1, 0);

            for(int k = 0; k < i; k ++, y \*= x) {

                comp p = a[j + k], q = y \* a[j + k + i];

                a[j + k] = p + q; a[j + k + i] = p - q;

            }

        }

    }

}

int main() {

    n = read(), m = read();

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

    for(int i = 0; i <= m; i ++) b[i] = read();

    int l = 0;

    for(lim = 1; lim <= n + m; lim <<= 1) ++ l;

    for(int i = 0; i < lim; i ++)

        r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));

    fft(a, 1), fft(b, 1);

    for(int i = 0; i <= lim; i ++) a[i] \*= b[i];

    fft(a, -1);

    for(int i = 0; i <= n + m; i ++)

        printf("%d ", (int)(0.5 + a[i].real() / lim));

    return 0;

}

## 5.2 高精乘

#include<cstdio>

#include<cstdlib>

#include<cmath>

#include<algorithm>

#include<cstring>

#include<complex>

using namespace std;

typedef complex<double> cd;//复数类的定义

const int maxl=3000000;

const double PI=acos(-1.0);

cd a[maxl],b[maxl];//用于储存变换的中间结果

int rev[maxl];//用于储存二进制反转的结果

void getrev(int bit){

    for(int i=0;i<(1<<bit);i++){//高位决定二进制数的大小

        rev[i]=(rev[i>>1]>>1)|((i&1)<<(bit-1));

    }//能保证(x>>1)<x,满足递推性质

}

void fft(cd\* a,int n,int dft){//变换主要过程

    for(int i=0;i<n;i++){//按照二进制反转

        if(i<rev[i])//保证只把前面的数和后面的数交换,(否则数组会被翻回来)

            swap(a[i],a[rev[i]]);

    }

    for(int step=1;step<n;step<<=1){//枚举步长的一半

        cd wn=exp(cd(0,dft\*PI/step));//计算单位复根

        for(int j=0;j<n;j+=step<<1){//对于每一块

            cd wnk(1,0);//!!每一块都是一个独立序列,都是以零次方位为起始的

            for(int k=j;k<j+step;k++){//蝴蝶操作处理这一块

                cd x=a[k];

                cd y=wnk\*a[k+step];

                a[k]=x+y;

                a[k+step]=x-y;

                wnk\*=wn;//计算下一次的复根

            }

        }

    }

    if(dft==-1){//如果是反变换,则要将序列除以n

        for(int i=0;i<n;i++)

            a[i]/=n;

    }

}

int output[maxl];

char s1[maxl],s2[maxl];

int main(){

    while(~scanf("%s%s",s1,s2)){

        int l1=strlen(s1),l2=strlen(s2);

        int jd=0;

        int t;

        if(s1[0]=='-'){

            for(t=0;t<l1-1;t++){

                s1[t]=s1[t+1];

            }

            s1[t]='\0';

            l1--;

            jd++;

        }

        if(s2[0]=='-'){

            for(t=0;t<l2-1;t++){

                s2[t]=s2[t+1];

            }

            s2[t]='\0';

            l2--;

            jd++;

        }

        memset(output,0,sizeof(output));

        memset(a,0,sizeof(a));

        memset(b,0,sizeof(b));

        memset(rev,0,sizeof(rev));

        int bit=1,s=2;//s表示分割之前整块的长度

        for(bit=1;(1<<bit)<l1+l2-1;bit++){

            s<<=1;//找到第一个二的整数次幂使得其可以容纳这两个数的乘积

        }

        for(int i=0;i<l1;i++){//第一个数装入a

            a[i]=(double)(s1[l1-i-1]-'0');

        }

        for(int i=0;i<l2;i++){//第二个数装入b

            b[i]=(double)(s2[l2-i-1]-'0');

        }

        getrev(bit);fft(a,s,1);fft(b,s,1);//dft

        for(int i=0;i<s;i++)a[i]\*=b[i];//对应相乘

        fft(a,s,-1);//idft

        for(int i=0;i<s;i++){//还原成十进制数

            output[i]+=(int)(a[i].real()+0.5);//注意精度误差

            output[i+1]+=output[i]/10;

            output[i]%=10;

        }

        int i;

        for(i=l1+l2;!output[i]&&i>=0;i--);//去掉前导零

        if(i==-1)printf("0");//特判长度为0的情况

        if(jd==1){

            printf("-");

        }

        for(;i>=0;i--){//输出这个十进制数

            printf("%d",output[i]);

        }

        putchar('\n');

    }

    return 0;

}

## \*1. 求高精度乘法

#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;

}

## \*2. 给出n条线段长度，问任取3根组成三角形的概率

//n<=10^5 用 FFT 求可以组成三角形的取法有几种

const int MAXN = 400040;

Complex x1[MAXN];

int a[MAXN / 4];

long long num[MAXN];//100000\*100000会超int

long long sum[MAXN];

int main()

{

int T;

int n;

scanf("%d", &T);

while (T--)

{

scanf("%d", &n);

memset(num, 0, sizeof(num));

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

{

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

num[a[i]]++;

}

sort(a, a + n);

int len1 = a[n - 1] + 1;

int len = 1;

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

for (int i = 0; i < len1; i++)

x1[i] = Complex(num[i], 0);

for (int i = len1; i < len; i++)

x1[i] = Complex(0, 0);

fft(x1, len, 1);

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

x1[i] = x1[i] \* x1[i];

fft(x1, len, -1);

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

num[i] = (long long)(x1[i].x + 0.5);

len = 2 \* a[n - 1];

//减掉取两个相同的组合

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

num[a[i] + a[i]]--;

for (int i = 1; i <= len; i++)num[i] /= 2;

sum[0] = 0;

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

sum[i] = sum[i - 1] + num[i];

long long cnt = 0;

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

{

cnt += sum[len] - sum[a[i]];

//减掉一个取大，一个取小的

cnt -= (long long)(n - 1 - i)\*i;

//减掉一个取本身，另外一个取其它

cnt -= (n - 1);

cnt -= (long long)(n - 1 - i)\*(n - i - 2) / 2;

}

long long tot = (long long)n\*(n - 1)\*(n - 2) / 6;

printf("%.7lf\n", (double)cnt / tot);

}

return 0;

}

# 6 字符串

## 6.1 KMP

#include<cstdio>

#include<iostream>

#include<algorithm>

#include<string>

#include<cstring>

#include<cmath>

using namespace std;

char a1[2000000],a2[2000000];

int kmp[2000000],nxt[2000000];

int main(){

    scanf("%s%s",a1,a2);

    kmp[0]=kmp[1]=0;//前一位，两位失配了，都只可能将第一位作为新的开头

    int len1=strlen(a1),len2=strlen(a2);

    int k;

    k=0;

    for(int i=1;i<len2;i++){

        while(k&&a2[i]!=a2[k])k=kmp[k];

        //找到最长的前后缀重叠长度

        kmp[i+1]=a2[i]==a2[k]?++k:0;

        //不相等的情况，即无前缀能与后缀重叠，直接赋值位0（注意是给下一位，因为匹配的是下一位适失配的情况）

    }

    k=0;

    for(int i=0;i<len1;i++){

        while(k&&a1[i]!=a2[k])k=kmp[k];//如果不匹配，则将利用kmp数组往回跳

        k+=a1[i]==a2[k]?1:0;//如果相等了，则匹配下一位

        if(k==len2)printf("%d\n",i-len2+2);//如果已经全部匹配完毕，则输出初始位置

    }

    for(int i=1;i<=len2;i++)printf("%d ",kmp[i]);

    printf("\n");

    for(int i=2;i<=len2;i++)nxt[i]=kmp[i-1]+1;

    for(int i=1;i<=len2;i++)printf("%d ",nxt[i]);

    printf("\n");

    return 0;

}

## 6.2 AC自动机

/\*给你一个文本串S和n个模式串T1..n​，请你分别求出每个模式串在S中出现的次数。

第一行包含一个正整数n表示模式串的个数。

接下来n行，第i行包含一个由小写英文字母构成的字符串 TiT\_iTi​。

最后一行包含一个由小写英文字母构成的字符串S。

输出包含n行，其中第i行包含一个非负整数表示Ti在S中出现的次数。\*/

#include<cstdio>

#include<cstring>

#include<queue>

#include<algorithm>

#define maxn 2000001

using namespace std;

char s[maxn],T[maxn];

int n,cnt,vis[200051],ans,in[maxn],Map[maxn];

struct kkk{

    int son[26],fail,flag,ans;

    void clear(){memset(son,0,sizeof(son)),fail=flag=ans=0;}

}trie[maxn];

queue<int>q;

void insert(char\* s,int num){

    int u=1,len=strlen(s);

    for(int i=0;i<len;i++){

        int v=s[i]-'a';

        if(!trie[u].son[v])trie[u].son[v]=++cnt;

        u=trie[u].son[v];

    }

    if(!trie[u].flag)trie[u].flag=num;

    Map[num]=trie[u].flag;

}

void getFail(){

    for(int i=0;i<26;i++)trie[0].son[i]=1;

    q.push(1);

    while(!q.empty()){

        int u=q.front();q.pop();

        int Fail=trie[u].fail;

        for(int i=0;i<26;i++){

            int v=trie[u].son[i];

            if(!v){trie[u].son[i]=trie[Fail].son[i];continue;}

            trie[v].fail=trie[Fail].son[i]; in[trie[v].fail]++;

            q.push(v);

        }

    }

}

void topu(){

    for(int i=1;i<=cnt;i++)

    if(in[i]==0)q.push(i);

    while(!q.empty()){

        int u=q.front();q.pop();vis[trie[u].flag]=trie[u].ans;

        int v=trie[u].fail;in[v]--;

        trie[v].ans+=trie[u].ans;

        if(in[v]==0)q.push(v);

    }

}

void query(char\* s){

    int u=1,len=strlen(s);

    for(int i=0;i<len;i++)

    u=trie[u].son[s[i]-'a'],trie[u].ans++;

}

int main(){

    scanf("%d",&n); cnt=1;

    for(int i=1;i<=n;i++){

        scanf("%s",s);

        insert(s,i);

    }getFail();scanf("%s",T);

    query(T);topu();

    for(int i=1;i<=n;i++)printf("%d\n",vis[Map[i]]);

}

## \*1. 经典题目：POJ 3167

/\*

\* POJ 3167 Cow Patterns

\* 模式串可以浮动的模式匹配问题

\* 给出模式串的相对大小，需要找出模式串匹配次数和位置

\* 比如说模式串：1，4，4，2，3，1 而主串：5,6,2,10,10,7,3,2,9

\* 那么2,10,10,7,3,2就是匹配的

\*

\* 统计比当前数小，和于当前数相等的，然后进行kmp

\*/

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <algorithm>

#include <vector>

using namespace std;

const int MAXN = 100010;

const int MAXM = 25010;

int a[MAXN];

int b[MAXN];

int n, m, s;

int as[MAXN][30];

int bs[MAXM][30];

void init()

{

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

{

if (i == 0)

{

for (int j = 1; j <= 25; j++)as[i][j] = 0;

}

else

{

for (int j = 1; j <= 25; j++)as[i][j] = as[i - 1][j];

}

as[i][a[i]]++;

}

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

{

if (i == 0)

{

for (int j = 1; j <= 25; j++)bs[i][j] = 0;

}

else

{

for (int j = 1; j <= 25; j++)bs[i][j] = bs[i - 1][j];

}

bs[i][b[i]]++;

}

}

int next[MAXM];

void kmp\_pre()

{

int i, j;

j = next[0] = -1;

i = 0;

while (i < m)

{

int t11 = 0, t12 = 0, t21 = 0, t22 = 0;

for (int k = 1; k < b[i]; k++)

{

if (i - j > 0)t11 += bs[i][k] - bs[i - j - 1][k];

else t11 += bs[i][k];

}

if (i - j > 0)t12 = bs[i][b[i]] - bs[i - j - 1][b[i]];

else t12 = bs[i][b[i]];

for (int k = 1; k < b[j]; k++)

{

t21 += bs[j][k];

}

t22 = bs[j][b[j]];

if (j == -1 || (t11 == t21 && t12 == t22))

{

next[++i] = ++j;

}

else j = next[j];

}

}

vector<int>ans;

void kmp()

{

ans.clear();

int i, j;

kmp\_pre();

i = j = 0;

while (i < n)

{

int t11 = 0, t12 = 0, t21 = 0, t22 = 0;

for (int k = 1; k < a[i]; k++)

{

if (i - j > 0)t11 += as[i][k] - as[i - j - 1][k];

else t11 += as[i][k];

}

if (i - j > 0)t12 = as[i][a[i]] - as[i - j - 1][a[i]];

else t12 = as[i][a[i]];

for (int k = 1; k < b[j]; k++)

{

t21 += bs[j][k];

}

t22 = bs[j][b[j]];

if (j == -1 || (t11 == t21 && t12 == t22))

{

i++; j++;

if (j >= m)

{

ans.push\_back(i - m + 1);

j = next[j];

}

}

else j = next[j];

}

}

int main()

{

while (scanf("%d%d%d", &n, &m, &s) == 3)

{

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

{

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

}

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

{

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

}

init();

kmp();

printf("%d\n", ans.size());

for (int i = 0; i < ans.size(); i++)

printf("%d\n", ans[i]);

}

return 0;

}

## \*2. 扩展 KMP

/\*

\* 扩展KMP算法

\*/

//next[i]:x[i...m-1]与x[0...m-1]的最长公共前缀

//extend[i]:y[i...n-1]与x[0...m-1]的最长公共前缀

void pre\_EKMP(char x[], int m, int next[])

{

next[0] = m;

int j = 0;

while (j + 1 < m && x[j] == x[j + 1])j++;

next[1] = j;

int k = 1;

for (int i = 2; i < m; i++)

{

int p = next[k] + k - 1;

int L = next[i - k];

if (i + L < p + 1)next[i] = L;

else

{

j = max(0, p - i + 1);

while (i + j < m && x[i + j] == x[j])j++;

next[i] = j;

k = i;

}

}

}

void EKMP(char x[], int m, char y[], int n, int next[], int extend[])

{

pre\_EKMP(x, m, next);

int j = 0;

while (j < n && j < m && x[j] == y[j])j++;

extend[0] = j;

int k = 0;

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

{

int p = extend[k] + k - 1;

int L = next[i - k];

if (i + L < p + 1)extend[i] = L;

else

{

j = max(0, p - i + 1);

while (i + j < n && j < m && y[i + j] == x[j])j++;

extend[i] = j;

k = i;

}

}

}

## \*3. Manacher 最长回文子串

/\*

\* 求最长回文子串

\*/

const int MAXN = 110010;

char Ma[MAXN \* 2];

int Mp[MAXN \* 2];

void Manacher(char s[], int len)

{

int l = 0;

Ma[l++] = '$';

Ma[l++] = '#';

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

{

Ma[l++] = s[i];

Ma[l++] = '#';

}

Ma[l] = 0;

int mx = 0, id = 0;

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

{

Mp[i] = mx > i ? min(Mp[2 \* id - i], mx - i) : 1;

while (Ma[i + Mp[i]] == Ma[i - Mp[i]])Mp[i]++;

if (i + Mp[i] > mx)

{

mx = i + Mp[i];

id = i;

}

}

}

/\*

\* abaaba

\* i: 0 1 2 3 4 5 6 7 8 9 10 11 12 13

\* Ma[i]: $ # a # b # a # a $ b # a #

\* Mp[i]: 1 1 2 1 4 1 2 7 2 1 4 1 2 1

\*/

char s[MAXN];

int main()

{

while (scanf("%s", s) == 1)

{

int len = strlen(s);

Manacher(s, len);

int ans = 0;

for (int i = 0; i < 2 \* len + 2; i++)

ans = max(ans, Mp[i] - 1);

printf("%d\n", ans);

}

return 0;

}