目录

[图论 2](#_Toc468640717)

[spfa（最小费用最大流） 2](#_Toc468640718)

[双联通分量 3](#_Toc468640719)

[割点割边 3](#_Toc468640720)

[二分图最大匹配 4](#_Toc468640721)

[SPFA判断是否有正环 4](#_Toc468640722)

[IASP 5](#_Toc468640723)

[SAP求此图的最大流 5](#_Toc468640724)

[floyed 6](#_Toc468640725)

[字符串 6](#_Toc468640726)

[字符串——exkmp 6](#_Toc468640727)

[利用kmp的next数组求循环节 7](#_Toc468640728)

[后缀数组 7](#_Toc468640729)

[manachar 8](#_Toc468640730)

[AC自动机 8](#_Toc468640731)

[数据结构 9](#_Toc468640732)

[线段树——查询比a大的第k个数 9](#_Toc468640733)

[主席树——树上两点路径第k小，树上主席树，动态lca 10](#_Toc468640734)

[树状数组 12](#_Toc468640735)

[树链剖分 12](#_Toc468640736)

[STL堆 14](#_Toc468640737)

[SPLAY 14](#_Toc468640738)

[TREAP 15](#_Toc468640739)

[博弈 16](#_Toc468640740)

[博弈——威佐夫 16](#_Toc468640741)

[博弈——antiNim 16](#_Toc468640742)

[博弈——比胜态走法 17](#_Toc468640743)

[博弈——Multinum，sg 17](#_Toc468640744)

[数学 18](#_Toc468640745)

[错排问题 18](#_Toc468640746)

[卡特兰数 18](#_Toc468640747)

[高斯消元 18](#_Toc468640748)

[欧拉函数 & 素数表 19](#_Toc468640749)

[模线性方程 20](#_Toc468640750)

[矩阵快速幂 20](#_Toc468640751)

[线性筛法求素数 21](#_Toc468640752)

[线性求中位数 21](#_Toc468640753)

[求素数个数 21](#_Toc468640754)

[快速幂 21](#_Toc468640755)

[归并排序求逆序数 22](#_Toc468640756)

[计算几何 22](#_Toc468640757)

[计算几何基础模板 22](#_Toc468640758)

[求多边形面积 23](#_Toc468640759)

[判断两直线是否相交 23](#_Toc468640760)

[求两个圆相交面积 24](#_Toc468640761)

[凸包 24](#_Toc468640762)

[其他 25](#_Toc468640763)

[二层魔方 25](#_Toc468640764)

[从数组取n个元素组合 25](#_Toc468640765)

# 图论

## spfa（最小费用最大流）

const int INF = 0x3fffffff;

bool inq[maxNode];

char org[105][105];

int pre[maxNode], res[maxNode][maxNode], cost[maxNode][maxNode], d[maxNode];

struct node{

int x, y;

}h[maxNode], m[maxNode];

bool SPFA(int s, int t){

queue<int> q;

memset(inq, 0, sizeof(inq));

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

inq[s] = 1;

q.push(s);

for(int i = s; i <= t; i ++)

d[i] = INF;

d[s] = 0;

while(!q.empty()){

int u = q.front();

q.pop();

inq[u] = 0;

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

if(res[u][i] && d[u] + cost[u][i] < d[i]){

d[i] = d[u] + cost[u][i];

pre[i] = u;

if(!inq[i]) {

inq[i] = 1;

q.push(i);

}

}

}

}

if(pre[t] == -1)

return false;

return true;

}

int MCMF(int s, int t){

int mincost = 0;

while(SPFA(s, t)) {

int v = t;

while(v != -1) {

res[pre[v]][v] -= 1;

res[v][pre[v]] += 1;

v = pre[v];

}

mincost += d[t];

}

return mincost;

}

int main (){

int r, c;

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

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

scanf("%s", org[i]+1);

}

int house = 0, man = 0;

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

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

if(org[i][j] == 'H') {

h[house].x = i;

h[house].y = j;

house ++;

}

if(org[i][j] == 'm'){

m[man].x = i;

m[man].y = j;

man ++;

}

}

}

memset(res, 0, sizeof(res));

memset(cost, 0, sizeof(cost));

int s = 0, t = house + man + 1;

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

res[s][i] = 1;

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

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

int dis = abs(h[i].x - m[j].x) + abs(h[i].y - m[j].y);

res[i + 1][j + house + 1] = 1;

cost[i + 1][j + house + 1] = dis;

cost[j + house + 1][i + 1] = -dis;

}

}

for(int i = house + 1; i < t; i ++)

res[i][t] = 1;

printf("%d\n", MCMF(s, t));

}

return 0;

}

## 双联通分量

//此题利用tarjan求加多少条边可以得到双连通分量

struct node{

int to, next;

}edge[3000];

int dfn[1005], vis[1005], low[1004], head[1005], in[1005];

int time, n, edge\_total;

void addEdge(int a, int b){

edge[edge\_total].to = a;

edge[edge\_total].next = head[b];

head[b] = edge\_total ++;

edge[edge\_total].to = b;

edge[edge\_total].next = head[a];

head[a] = edge\_total ++;

}

void tarjan\_init(){

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

memset(dfn, 0, sizeof(dfn));

memset(in, 0, sizeof(in));

time = 1;

}

void dfs(int id, int fa){

dfn[id] = low[id] = time ++;

vis[id] = 1;

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

int t = edge[i].to;

if(t == fa)

continue;

//因为建边的时候建的是双向边，因此必须检测这条边是否指向他的父亲

if(!vis[t]){

dfs(t, id);

low[id] = min(low[id], low[t]);

}

else{

low[id] = min(low[id], dfn[t]);

}

}

}

int tarjan(){

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

if(!vis[i])

dfs(i, i);

}

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

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

if(low[i] != low[edge[j].to])

in[low[i]] ++;

}

}

int ans = 0;

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

if(in[i] == 1)

ans ++;

}

return (ans + 1) / 2;

}

int main (){

int r, a, b;

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

edge\_total = 0;

memset(head, -1, sizeof(head[0]) \* (n+1));

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

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

addEdge(a, b);

}

tarjan\_init();

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

}

return 0;

}

## 割点割边

int dfn[106], vis[105], low[105], head[106], flag[105];

int time, total, ans;

struct node{

int to, next;

}edge[10000006];

int min(int a, int b){

return a>b? b: a;

}

void add(int a, int b){

edge[total].to = b;

edge[total].next = head[a];

head[a] = total ++;

}

void dfs(int id){

time ++;

low[id] = dfn[id] = time;

vis[id] = 1;

int cnum = 0;

for(int i = head[id]; i; i = edge[i].next){

int temp = edge[i].to;

if(vis[temp]){

low[id] = min(low[id], dfn[temp]);

}

else{

cnum++;

dfs(temp);

low[id] = min(low[id], low[temp]);

if(id == 1 && cnum > 1)

flag[id] = 1;

if(id != 1 && low[temp] >= dfn[id])

flag[id] = 1;

}

}

}

int main (){

int N, a, b;

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

ans = time = 0;

total = 1;

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

memset(head, 0, sizeof(head));

//head=0表示没有该边。因此edge的下标必须从1开始

memset(flag, 0, sizeof(flag));

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

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

add(a, b);

add(b, a);

if(getchar()=='\n')

break;

}

}

dfs(1);

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

ans += flag[i];

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

}

}

## 二分图最大匹配

int g[107][107], msp[1007][1007], msw[1007][1007], mpw[1005][1005], used[1005], linker[1004];

int uN, vN;

bool dfs(int u){

for(int v=0; v<vN; v++){

if(g[u][v]&&!used[v]){

used[v]=true;

if(linker[v]==-1||dfs(linker[v])){

linker[v]=u;

return true;

}

}

}

return false;

}

int hungary()

{

int res=0, u;

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

for(u=0; u<uN; u++){

memset(used, 0, sizeof(used));

if(dfs(u))res++;

}

return res;

}

## SPFA判断是否有正环

int e;

int head[10500], vis[10005], cnt[10050];

double dis[10050];

//dis可能是小数

struct node{

int v, next;

double r, c;

}edge[1500];

void add(int a, int b, double r, double c){

edge[e].v = b;

edge[e].r = r;

edge[e].c = c;

edge[e].next = head[a];

head[a] = e ++;

}

void SPFA\_init(){

e = 0;

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

memset(dis, 0, sizeof(dis));

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

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

}

int SPFA(int source, double much, int N){

queue<int> q;

q.push(source);

vis[source] = 1;

dis[source] = much;

cnt[source] ++;

while(!q.empty()){

int first = q.front();

q.pop();

vis[first] = 0;

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

int v = edge[i].v;

double tempdis = (dis[first] - edge[i].c) \* edge[i].r;

if(dis[v] < tempdis){

dis[v] = tempdis;

if(!vis[v]){

q.push(v);

vis[v] = 1;

}

cnt[v] ++;

if(cnt[v] > N + 1)

return -1;

}

}

}

return 1;

}

int main (){

int N, M, a, b, source;

double much, rab, rba, cba, cab;

scanf("%d%d%d%lf", &N, &M, &source, &much);

SPFA\_init();

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

scanf("%d%d%lf%lf%lf%lf", &a, &b, &rab, &cab, &rba, &cba);

add(a, b, rab, cab);

add(b, a, rba, cba);

}

int ans = SPFA(source, much, N);

if(ans == -1)

printf("YES\n");

else

printf("NO\n");

}

## IASP

const int inf = 0x3fffffff;

struct Iasp{

int top;

int head[ver], d[ver], gap[edg], pre[edg];

struct Edge{

int v, next;

int c, f;

}edges[edg];

void init(){

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

memset(gap, 0, sizeof(gap));

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

top = 0;

}

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

edges[top].v = v;

edges[top].c = c;

edges[top].f = 0;

edges[top].next = head[u];

head[u] = top ++;

}

void add(int u, int v, int c) {

add\_edge(u, v, c);

add\_edge(v, u, 0);

}

//为d数组赋值，求出每个点所在的层次。

//汇点处于0层

void set\_d(int t){

queue<int> q;

d[t] = 0;

q.push(t);

while(!q.empty()){

int v = q.front();

q.pop();

gap[d[v]] ++;

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

int u = edges[i].v;

if(d[u] == -1){

d[u] = d[v] + 1;

q.push(u);

}

}

}

}

}

## **SAP求此图的最大流**

int sap(int s, int t){

set\_d(t);

int ans = 0, u = s;

int flow = inf;

while(d[s] <= t){

int i;

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

int v = edges[i].v;

if(edges[i].c > edges[i].f && d[u] == d[v] + 1) {

u = v;

pre[v] = i;

flow = min(flow, edges[i].c - edges[i].f);

if(u == t){

while(u != s){

int j = pre[u];

edges[j].f += flow;

edges[j^1].f -= flow;

u = edges[j ^ 1].v;

}

ans += flow;

flow = inf;

}

break;

}

}

if(i == -1){

if(--gap[d[u]] == 0)

break;

int dmin = t;

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

if(edges[j].c > edges[j].f)

dmin = min(dmin, d[edges[j].v]);

}

d[u] = dmin + 1;

gap[d[u]] ++;

if(u != s)

u = edges[pre[u] ^ 1].v;

}

}

return ans;

}

}Sap;

int main (){

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

int s = 0;

int t = n \* 2 + f + d + 1;

Sap.init();

//先建立原点与食物的边，容量为1

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

Sap.add(s, i, 1);

}

//建立饮料与汇点的边，容量为1；

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

Sap.add(2\*n+f+i, t, 1);

}

int ans = Sap.sap(s, t);

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

}

## floyed

int flag = 0;

for(int k = 1; k <= N && !flag; k ++){

for(int i = 1; i <= N && !flag; i ++){

for(int j = 1; j <= N && !flag; j ++){

int t=w[i][k]+w[k][j];

if(w[i][j]>t)w[i][j]=t;

}

if(w[i][i] < 0)

flag = 1;

}

}

# 字符串

## 字符串——exkmp

#define next nxt

int next[50010];

int ex[50010];

void exkmp(char s1[], char s2[], int next[], int ex[]){

int i, j, p;

for (i = 0, j = 0, p = -1; s1[i] != '\0'; i++, j++, p--){

if(p == -1){

j = 0;

do

p++;

while(s1[i+p] != '\0' && s1[i+p] == s2[j+p]);

ex[i] = p;

}

else if(next[j] < p)

ex[i] = next[j];

else if(next[j] > p)

ex[i] = p;

else{

j = 0;

while(s1[i + p] != '\0' && s1[i+p] == s2[j+p]) p++;

ex[i] = p;

}

}

ex[i] = 0;

}

int main(){

char s1\_p[50010];

char s2\_s[50010];

while(~scanf("%s", s1\_p)){

scanf("%s", s2\_s);

next[0] = 0;

exkmp(s1\_p + 1, s1\_p, next, next+1);

exkmp(s2\_s, s1\_p, next, ex);

int len = strlen(s2\_s);

int mx = 0, mx\_index = -1;

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

if(ex[i] == len - i){

if(ex[i] > mx){

mx = ex[i];

mx\_index = i;

}

}

}

if(mx == 0){

printf("0\n");

}

else{

for (int i = mx\_index; i < len; i++){

printf("%c", s2\_s[i]);

}

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

}

}

return 0;

}

## 利用kmp的next数组求循环节

void get(void){

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

int j = next[i - 1];

while(j && in[j] != in[i - 1])

j = next[j];

next[i] = in[i - 1] == in[j] ? j + 1: 0;

}

}

void work(void){

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

if(i % (i - next[i]) == 0 && i / (i - next[i]) > 1)

printf("%d %d\n", i, i / (i - next[i]));

}

## 后缀数组

int wa[maxn], wb[maxn], wsf[maxn], wv[maxn], sa[maxn];

int rank[maxn], height[maxn], s[maxn];

char str[maxn], str1[maxn];

int cmp(int \*r, int a, int b, int k){

return r[a] == r[b] && r[a + k] == r[b + k];

}

void get\_sa(int \*r, int \*sa, int n, int m){

int \*x = wa, \*y = wb, \*t, i, j, p;

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

for(i = 0; i < n; i ++) wsf[x[i] = r[i]] ++;

for(i = 1; i < m; i ++) wsf[i] += wsf[i - 1];

for(i = n - 1; i >= 0; i --) sa[-- wsf[x[i]]] = i;

p = 1, j = 1;

for(; p < n; j \*= 2, m = p) {

for(p = 0, i = n - j; i < n; i ++) y[p ++] = sa[i] - j;

for(i = 0; i < n; i ++) if(sa[i] >= j) y[p ++] = sa[i] - j;

for(i = 0; i < n; i ++) wv[i] = x[y[i]];

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

for(i = 0; i < n; i ++) wsf[wv[i]] ++;

for(i = 1; i < m; i ++) wsf[i] += wsf[i - 1];

for(i = n - 1; i >= 0; i --) sa[--wsf[wv[i]]] = y[i];

t = x;

x = y;

y = t;

x[sa[0]] = 0;

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

x[sa[i]] == cmp(y, sa[i - 1], sa[i], j)? p - 1: p ++;

}

}

void getheight(int \*r, int n){

int i, j, k = 0;

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

rank[sa[i]] = i;

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

if(k)

k --;

j = sa[rank[i] - 1];

while(r[i + k] == r[j + k])

k ++;

height[rank[i]] = k;

}

}

int main(){

int T, n;

scanf("%d", &T);

while(T --){

scanf("%d", &n);

scanf("%s", str);

strcpy(str1, str);

strcat(str1, str1);

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

str[i] = str1[n - 1 - i];

strcat(str, str);

n \*= 2;

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

s[i] = str[i] - 'a';

}

s[n ++] = 28;

get\_sa(s, sa, n + 1, 30);

getheight(s, n);

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

if(height[i] == n / 2){

ans = i;

break;

}

}

}

return 0;

}

## manachar

char str[maxn], str1[maxn \* 2];

int dp[maxn \* 2], n, maxx = 0;

void Manacher(){

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

int mx = 0, id;

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

if(mx > i)

dp[i] = min(dp[2 \* id - i], mx - i);

else

dp[i] = 1;

for(; str1[i - dp[i]] == str1[i + dp[i]]; dp[i] ++);

maxx = max(maxx, dp[i]);

if(i + dp[i] > mx) {

mx = i + dp[i];

id = i;

}

}

}

void pre(){

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

str1[0] = '$';

while(str[i] != '\0'){

str1[k ++] = t? str[i ++] : '#';

t ^= 1

}

str1[k ++] = '#';

str1[k] = '\0';

n = k;

}

int main (){

while(~scanf("%s", str)) {

maxx = 0;

pre();

Manacher();

printf("%d\n", maxx - 1);

## AC自动机

int ch[500005][26], fail[500005], val[500005], total[500005];

int AC\_total;

char s[1000006];

void AC\_init(){

memset(ch, 0, sizeof(ch));

memset(fail, 0, sizeof(fail));

memset(val, 0, sizeof(val));

memset(total, 0, sizeof(total));

AC\_total = 1;

}

void AC\_insert(){

int len = strlen(s), id;

int u = 0;

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

id = s[i] - 'a';

if(ch[u][id] == 0)

ch[u][id] = AC\_total ++;

u = ch[u][id];

}

val[u] ++;

}

void AC\_build(){

queue<int> q;

while(!q.empty())

q.pop();

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

if(ch[0][i])

q.push(ch[0][i]);

while(!q.empty()){

int u = q.front();

q.pop();

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

int temp = ch[u][i];

if(temp != 0){

int v = fail[u];

while(v && !ch[v][i])

v = fail[v];

fail[temp] = ch[v][i];

q.push(temp);

}

}

}

}

int AC\_find(){

int n = strlen(s);

int j = 0, ans = 0;

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

int c = s[i] - 'a';

while(j && !ch[j][c])

j = fail[j];

j = ch[j][c];

int temp = j;

while(temp && val[temp] != -1){

ans += val[temp];

val[temp] = -1;

temp = fail[temp];

}

}

return ans;

}

int main (){

AC\_init();

scanf("%d", &n);

while(n --){

scanf("%s", s);

AC\_insert();

}

scanf("%s", s);

AC\_build();

int ans = AC\_find();

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

}

}

# 数据结构

## 线段树——查询比a大的第k个数

#define maxn 100010

int m;

int sum[maxn\*4];

void build(){

memset(sum, 0, sizeof sum);

}

void maintain(int o){

int lc=o<<1, rc=o<<1|1;

sum[o] = sum[lc] + sum[rc];

}

bool update(int o, int l, int r, int pos, int val){//正常的更新函数

if(l == r){

if(sum[o] == 0 && val < 0) return false;

sum[o] += val;

return true;

}

int mid=(l+r)>>1, lc=o<<1, rc=o<<1|1;

bool res = false;

if(pos <= mid) res = update(lc, l, mid, pos, val);

else res = update(rc, mid+1, r, pos, val);

maintain(o);

return res;

}

int ret;//存储大于等于pos的第k个数是几

void query(int o, int l, int r, int pos, int& rk){//查询pos（包含）之后的第k个数是多少。题目要求是求比pos大的第k个数，见第85行

if(rk <= 0 || sum[o] == 0) return;//要查的rk值是0或者此区间内没有数

if(l == r){//每次到根节点就更新ret值

ret = l;

rk -= sum[o];

return;

}

if(pos <= l && rk > sum[o]){//如果该区间在pos后并且该区间的数之和小于rk，那么rk直接减去，因为要查的值一定不在该区间。

rk -= sum[o];

return;

}

int mid = (l+r)>>1, lc=o<<1, rc=o<<1|1;

if(pos <= mid) query(lc, l, mid, pos, rk);//如果pos在左区间，那么要查左区间

query(rc, mid+1, r, pos, rk);//否则直接查找右区间

}

int main(){

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

build();

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

int op1, op2, op3;

scanf("%d%d", &op1, &op2);

if(op1 == 0){

update(1, 1, 100000, op2, 1);

}

else if(op1 == 1){

if(update(1, 1, 100000, op2, -1) == false){

printf("No Elment!\n");

}

}

else if(op1 == 2){

scanf("%d", &op3);

ret = 0;

query(1, 1, 100000, op2+1, op3);//本行在传递的时候已经处理了比pos大的第k个数，即求大于等于op2+1的第k个数

if(ret == 0 || op3 > 0) printf("Not Find!\n");//因为每次到叶节点就更新，如果op3的值最后小于等于0了，说名找到了，如果没有的话，说明此时找的这个ret不是结果，没找到。

else printf("%d\n", ret);

}

}

}

}

## 主席树——树上两点路径第k小，树上主席树，动态lca

//本题是让求树上两点路径之间的第k大，树上主席树+动态lca+输入外挂

//bzoj上需要输入外挂，而且要用边集数组来存边，用vector可能会TLE

//本题的思路是每个节点对应的线段树是从其父亲节点的线段树修改过来的，这样每个节点的线段树的结果，都是到根节点的路径上所有点的结果

//在进行查询的时候，只需要两个节点都对他们的lca节点做差就是u，v路径上所有点的线段树的结果。

//

#define maxn 100005

#define maxm 100005

int n, m;

int u, v;

int x, y, z;

int val[maxn], valc[maxn];

int len;//离散后的长度

int tot, trcnt;

int tr[maxn];

int lc[maxn\*30], rc[maxn\*30], sum[maxn\*30];

//输入外挂

inline int iread() {

int f = 1; ll x = 0; char ch = getchar();

for(; ch < '0' || ch > '9'; ch = getchar()) f = ch == '-' ? -1 : 1;

for(; ch >= '0' && ch <= '9'; ch = getchar()) x = x \* 10 + ch - '0';

return f \* x;

}

struct Edge{

int to, next;

}edge[maxn\*2];

int totedge, head[maxn];

void init\_edge(){

totedge = 0;

memset(head, -1, sizeof head);

}

void add\_edge(int u, int v){//无向图要加两条边

edge[totedge].to = v;

edge[totedge].next = head[u];

head[u] = totedge++;

}

//-----------presistent seg tree--------------

void init\_segtree(){//建树之前的初始化

tot = trcnt = 0;

memcpy(valc, val, sizeof val);

sort(valc, valc+n);

len = unique(valc, valc+n)-valc;

}

int build\_segtree(int l, int r){//建树操作

int root = tot++;

sum[root] = 0;

if(l != r){

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

lc[root] = build\_segtree(l, mid);

rc[root] = build\_segtree(mid+1, r);

}

return root;

}

int update\_segtree(int Lroot, int l, int r, int pos){//主席树正常的更新操作，根据Lroot创建一个新的树，与Lroot公用节点

int newroot = tot++;

sum[newroot] = sum[Lroot] + 1;

if(l != r){

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

if(pos <= mid){

lc[newroot] = update\_segtree(lc[Lroot], l, mid, pos);

rc[newroot] = rc[Lroot];

}

else{

lc[newroot] = lc[Lroot];

rc[newroot] = update\_segtree(rc[Lroot], mid+1, r, pos);

}

}

return newroot;

}

void dfs\_update(int child, int fa){//主席树的dfs更新操作，用子节点的树通过父节点的树来建，与父节点的树公用节点。

int hash = lower\_bound(valc, valc+len, val[child-1])-valc;//这样的话，每个节点存储的是从根节点到当前节点的修改操作后的结果。查询操作好维护。

tr[child] = update\_segtree(tr[fa], 0, len-1, hash);

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

int to = edge[i].to;

if(to == fa) continue;

dfs\_update(to, child);

}

}

int query\_segtree(int one, int two, int father, int l, int r, int rk, int fatherhash){//查询u，v路径上的第k小。

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

if(l == r) return l;

int tmp = sum[lc[one]] - sum[lc[father]] + sum[lc[two]] - sum[lc[father]];//u，v路径上不包括公共祖先的点的线段树在左子树上的结果。与静态区间第k大思路相仿。

if(fatherhash >= l && fatherhash <= mid) tmp++;//看公共祖先这个点的哈希值是否应该包含在线段树的左树上，是就加上

if(tmp >= rk) return query\_segtree(lc[one], lc[two], lc[father], l, mid, rk, fatherhash);//如果在排名在左子树上的点的数量大于等于k，左子上寻找

else return query\_segtree(rc[one], rc[two], rc[father], mid+1, r, rk-tmp, fatherhash);//否则就在右子上寻找

}

//----------------lca dynamic------------------kuangbin模板

int rmq[2\*maxn];

struct ST{

int mm[2\*maxn];

int dp[2\*maxn][20];

void init(int n){

mm[0] = -1;

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

mm[i] = ((i&(i-1)) == 0) ? mm[i-1]+1 : mm[i-1];

dp[i][0] = i;

}

for (int j = 1; j <= mm[n]; j++)

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

dp[i][j] = rmq[dp[i][j-1]] < rmq[dp[i+(1<<(j-1))][j-1]] ? dp[i][j-1]:dp[i+(1<<(j-1))][j-1];

}

int query(int a, int b){

if(a > b) swap(a, b);

int k = mm[b-a+1];

return rmq[dp[a][k]] <= rmq[dp[b-(1<<k)+1][k]] ? dp[a][k] : dp[b-(1<<k)+1][k];

}

};

int F[maxn\*2];//欧拉序列

int P[maxn];

int cnt;

ST st;

void dfs\_lca(int u, int pre, int dep){//求三个序列

F[++cnt] = u;

rmq[cnt] = dep;

P[u] = cnt;

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

int v = edge[i].to;

if(v == pre) continue;

dfs\_lca(v, u, dep+1);

F[++cnt] = u;

rmq[cnt] = dep;

}

}

void init\_lca(int root, int node\_num){

cnt = 0;

dfs\_lca(root, root, 0);

st.init(2\*node\_num-1);

}

int query\_lca(int u, int v){

return F[st.query(P[u],P[v])];

}

int main(){

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

int pre = 0;//题目要求，要求记录前一个值，并且要与下一个询问的前边的那个范围做异或操作

for (int i = 0; i < n; i++) val[i] = iread();

init\_segtree();//建树

tr[0] = build\_segtree(0, len-1);

init\_edge();//读图

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

u = iread();

v = iread();

add\_edge(u, v);

add\_edge(v, u);

}

dfs\_update(1, 1);//继续建主席树

init\_lca(1, n);//lca询问初始化

while(m--){

x = iread(),y = iread(), z = iread();

x = x^pre;

int father = query\_lca(x, y);

int fatherhash = lower\_bound(valc, valc+len, val[father-1])-valc;

pre = valc[query\_segtree(tr[x], tr[y], tr[father], 0, len-1, z, fatherhash)];

printf("%d", pre);

if(m != 0) printf("\n");

}

}

}

## 树状数组

int a, b, ans[150005], s[32005]n;

int lowbit(int x){

return x &(-x);

}

void add(int x, int val){

for(int i = x; i <= 32003; i += lowbit(i)){

s[i] += val;

}

}

int sum(int x){

int re = 0;

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

re += s[i];

}

return re;

}

## **树链剖分**

//树上节点的权值，以该节点为根的子树节点个数，节点所在重链的头，节点重链上的子节点

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

//节点的深度，节点对应线段树上的位置下标，线段树上位置对应的节点下标，节点的父节点

int dep[N], tid[N], \_rank[N], fa[N];

//建图所用

int head[N], to[N \* 2], \_next[N \* 2], edge;

//线段树上每个节点所需维护的值，线段树上节点是否有更改操作

int sum[N \* 4], col[N \* 4];

//当前深度，树的总结点树（线段树的最右端点）

int tim, n;

void init(){

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

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

tim = 1;

edge = 0;

}

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

to[edge] = v;

\_next[edge] = head[u];

head[u] = edge ++;

to[edge] = u;

\_next[edge] = head[v];

head[v] = edge ++;

}

//当前结点，父节点，深度

void dfs1(int u, int f, int d){

dep[u] = d;

fa[u] = f;

siz[u] = 1;

for(int i = head[u]; i != -1; i = \_next[i]){

int v = to[i];

if(v != f){

dfs1(v, u, d + 1);

siz[u] += siz[v];

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

son[u] = v;

}

}

}

//当前节点，所在重链

void dfs2(int u, int tp){

top[u] = tp;

tid[u] = tim;

\_rank[tim ++] = u;

if(son[u] == -1)

return ;

dfs2(son[u], tp);

for(int i = head[u]; i != -1; i = \_next[i]){

int v = to[i];

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

dfs2(v, v);

}

}

//由ｒｔ节点的两个儿子节点更新ｒｔ

void push\_up(int rt){

sum[rt] = max(sum[rt << 1], sum[rt << 1 | 1]);

}

//rt 点的lazy 操作

void push\_down(int rt, int m){

if(col[rt]){

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

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

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

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

col[rt] = 0;

}

}

//线段树建树

void build(int l, int r, int rt){

col[rt] = 0;

if(l == r) {

sum[rt] = num[\_rank[l]];

return ;

}

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

build(l, mid, rt << 1);

build(mid + 1, r, rt << 1 | 1);

push\_up(rt);

}

//线段树更新

void update(int l, int r, int v, int ll, int rr, int rt){

if(l <= ll && r >= rr) {

col[rt] += v;

sum[rt] += v \* (rr - ll + 1);

return ;

}

push\_down(rt, rr - ll + 1);

int mid = (ll + rr ) >> 1;

if(l <= mid)

update(l, r, v, ll, mid, rt << 1);

if(r > mid)

update(l, r, v, mid + 1, rr, rt << 1 | 1);

push\_up(rt);

}

//线段树查询

int query(int l, int r, int rt, int val){

if(l == r)

return sum[rt];

push\_down(rt, r - l + 1);

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

int ret = 0;

if(val <= mid)

ret = query(l, mid, rt << 1, val);

if(val > mid)

ret = query(mid + 1, r, rt << 1 | 1, val);

push\_up(rt);

return ret;

}

//树链更新

void change(int x, int y, int val){

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

if(dep[top[x]] < dep[top[y]])

swap(x, y);

update(tid[top[x]], tid[x], val, 1, n, 1);

x = fa[top[x]];

}

if(dep[x] > dep[y])

swap(x, y);

update(tid[x], tid[y], val, 1, n, 1);

}

int main (){

int a, b, c, m, q;

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

init();

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

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

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

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

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

add\_edge(a, b);

}

dfs1(1, 0, 0);

dfs2(1, 1);

build(1, n, 1);

char op[20];

while(q --){

scanf("%s", op);

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

scanf("%d", &a);

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

}

else{

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

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

c = -c;

change(a, b, c);

}

}

}

}

## STL堆

make\_heap(a, a + m);

pop\_heap(a, a + m);

push\_heap(a, a + m);

## SPLAY

#define N 500000

#define lc (tr[id].c[0])

#define rc (tr[id].c[1])

#define KEY (tr[tr[root].c[1]].c[0])//根的右孩子的左孩子

struct Tr {

int fa, sum, val, c[2], lz;

}tr[N];

int newtr(int k, int f) {//新建立一个节点

tr[tot].sum = 1, tr[tot].val = k;

tr[tot].c[0] = tr[tot].c[1] = -1;

tr[tot].lz = 0;

tr[tot].fa = f;

return tot++;

}

void Push(int id) {

int lsum, rsum;

lsum = (lc == -1)?0:tr[lc].sum;

rsum = (rc == -1)?0:tr[rc].sum;

tr[id].sum = lsum+rsum+1;

}

void lazy(int id) {//flip专属懒操作

if (tr[id].lz) {

swap(lc, rc);

tr[lc].lz ^= 1, tr[rc].lz ^= 1;

tr[id].lz = 0;

}

}

int build(int l, int r, int f) {//建树

if (r < l) return-1;

int mid = l+r>>1;

int ro = newtr(data[mid], f);

tr[ro].c[0] = build(l, mid-1, ro);

tr[ro].c[1] = build(mid+1, r, ro);

Push(ro);

return ro;

}

void Rotate(int x, int k) {//k=1右旋,k=0左旋

if (tr[x].fa == -1) return;

int fa = tr[x].fa, w;

lazy(fa), lazy(x);

tr[fa].c[!k] = tr[x].c[k];

if (tr[x].c[k] != -1) tr[tr[x].c[k]].fa = fa;

tr[x].fa = tr[fa].fa, tr[x].c[k] = fa;

if (tr[fa].fa != -1) {

w = tr[tr[fa].fa].c[1]==fa;

tr[tr[fa].fa].c[w] = x;

}

tr[fa].fa = x;

Push(fa);

Push(x);

}

void Splay(int x, int goal) {//将x节点转到goal的儿子上

if (x == -1) return;

lazy(x);

while (tr[x].fa != goal) {

int y = tr[x].fa;

lazy(tr[y].fa), lazy(y), lazy(x);

bool w = x==tr[y].c[1];

if (tr[y].fa != goal && w == (y==tr[tr[y].fa].c[1]))

Rotate(y, !w);

Rotate(x, !w);

}

if (goal == -1) root = x;

Push(x);

}

int find(int k) {//找到第k个节点的ID

int id = root;

while (id != -1) {

lazy(id);

int lsum = (lc==-1)?0:tr[lc].sum;

if (lsum >= k) {

id = lc;

}

else if (lsum+1 == k) break;

else {

k = k-lsum-1;

id = rc;

}

}

return id;

}

int Index(int l, int r) {//将区间(l+1, r-1)化成一颗子树

Splay(find(l), -1);

Splay(find(r),root);

}

int Getnext(int id) {//寻找后继节点

lazy(id);

int p = tr[id].c[1];

if (p == -1) return id;

lazy(p);

while (tr[p].c[0] != -1) {

p = tr[p].c[0];

lazy(p);

}

return p;

}

int del(int l, int r) {//将【l,r】切掉,返回切掉子树的根节点

Index(l-1, r+1);

int ro = KEY;

tr[KEY].fa = -1;

KEY = -1;

Push(tr[root].c[1]);

Push(root);

return ro;

}

void cut(int k, int ro) {//将子树ro接到第k个树之后

Index(k, k+1);

KEY = ro;

tr[ro].fa = tr[root].c[1];

Push(tr[root].c[1]);

Push(root);

}

void filp(int l, int r) {//对区间【l,r】反转

Index(l-1, r+1);

lazy(root), lazy(tr[root].c[1]);

tr[KEY].lz ^= 1;

}

void Add(int l, int r, int d) {//区间【l,r】的数加上d

Index(l-1, r+1);

tr[KEY].add += d;

tr[KEY].mi += d;

tr[KEY].val += d;

Push(tr[root].c[1]);

Push(root);

}

void Delete(int x) {//删除第x个数

Index(x-1, x+1);

tr[KEY].fa = -1;

tr[tr[root].c[1]].c[0] = -1;

Push(tr[root].c[1]);

Push(root);

}

void Insert(int l, int x) {//在l之后插入x

Index(l, l+1);

int ro;

ro = newtr(x, tr[root].c[1]);

KEY = ro;

Push(tr[root].c[1]);

Push(root);

}

void Revolve(int l, int r, int d) {//【l, r】整体右移d位

int ro = del(r+1-d, r);

cut(l-1, ro);

}

## TREAP

struct treap{

treap \*left, \*right;

int val, pri;

int size;

treap (int vv){

left = right = NULL;

pri = rand();

val = vv;

}

}\*root;

void print(treap \*p){

if(!p)

return;

print(p->left);

printf("%d ", p->val);

print(p->right);

}

int lsize(treap \*p){

return p->left ? p->left->size : 0;

}

int rsize(treap \*p){

return p->right ? p->right->size : 0;

}

void l\_rotate(treap \*&p){

treap \*temp = p->right;

p->right = temp->left;

temp->left = p;

temp->size = p->size;

p->size = lsize(p) + rsize(p) + 1;

p = temp;

}

void r\_rotate(treap \*&p){

treap \*temp = p->left;

p->left = temp->right;

temp->right = p;

temp->size = p->size;

p->size = lsize(p) + rsize(p) + 1;

p = temp;

}

void insert(treap \*&p, int val) {

if(!p){

p = new treap(val);

p->size = 1;

}

else if(val <= p->val){

p->size ++;

insert(p->left, val);

if(p->left->pri < p->pri)

r\_rotate(p);

}

else{

p->size ++;

insert(p->right, val);

if(p->right->pri < p->pri)

l\_rotate(p);

}

}

int find(int k, treap \*p){

int temp = lsize(p);

if(k == temp + 1)

return p->val;

else if(k <= temp)

return find(k, p->left);

else return find(k - temp - 1, p->right);

}

int main (){

int m, n, num[30005];

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

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

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

int temp = 1, len, ans;

root = NULL;

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

scanf("%d", &len);

for(; temp <= len; temp ++){

insert(root, num[temp]);

}

ans = find(i, root);

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

}

return 0;

}

# 博弈

## 博弈——威佐夫

int main(){

int a, b;

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

if (a > b) swap(a, b);

int k = b-a;

int tmpa = (int)((1 + sqrt(5.0)) / 2 \* k);

int tmpb = tmpa + k;

if(tmpa == a && tmpb == b){

printf("0\n");

}

else printf("1\n");

}

return 0;

}

## 博弈——antiNim

int t, n;

int num;

int main(){

scanf("%d", &t);

while(t--){

scanf("%d", &n);

bool flag = false;

int res = 0;

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

scanf("%d", &num);

if(num > 1) flag = true;

res ^= num;

}

if(res == 0 && !flag)

printf("John\n");

else if(res != 0 && flag)

printf("John\n");

else printf("Brother\n");

}

return 0;

}

## 博弈——比胜态走法

int n;

int num[200010];

int main(){

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

int res = 0;

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

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

res ^= num[i];

}

sort(num, num + n);

if(res != 0){

printf("Yes\n");

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

if((res ^ num[i]) <= num[i]){

printf("%d %d\n", num[i], res^num[i]);

}

}

}

else{

printf("No\n");

}

}

}

## 博弈——Multinum，sg

int sg[1000];

bool visit[1000];

void getsg(){

memset(sg, 0, sizeof sg);

sg[0] = 0;

sg[1] = 1;

sg[2] = 2;

for (int i = 3; i < 1000; i++){

memset(visit, 0, sizeof visit);

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

visit[sg[j]] = true;

}

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

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

int l = i-j-k;

int yihuo = sg[j]^sg[k]^sg[l];

visit[yihuo] = true;

}

}

for (int j = 0; j < 1000; j++){

if(!visit[j]){

sg[i] = j;

break;

}

}

}

for (int i = 0; i < 100; i++){

cout << "sg[i] = " << i << " " << sg[i] << endl;

}

}

int t, n, num;

int main(){

scanf("%d", &t);

while(t--){

scanf("%d", &n);

int res = 0;

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

scanf("%d", &num);

if(num > 0 && num % 8 == 0){

res ^= num-1;

}

else if(num > 0 && num % 8 == 7){

res ^= num+1;

}

else res ^= num;

}

if(res != 0)

printf("First player wins.\n");

else printf("Second player wins.\n");

}

return 0;

}

# 数学

## **错排问题**

求给定排列中错排出现的概率。

给定一个排列，若１－ｎ没有出现在原来的位置，则说明这是一个错排

这里算的贼麻烦，用的容斥原理

错排的递推公式为D(n) = (n - 1) \* (D(n - 1) + D(n - 2))

最终的公式是：（欧拉大神做出来的）ｎ的错排数量为[(n!/e)+0.5] 这里[]表示向下取整

ll zuhe(ll a, ll b){

if(a > b - a)

a = b - a;

ll res = 1;

for(ll i = 1; i <= a; i ++){

res = res \* (b - i + 1 )/i;

}

return res;

}

ll jie(ll n){

ll res = 1;

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

res \*= i;

}

return res;

}

## **卡特兰数**

对于包含ｎ个Ｘ和ｎ个Ｙ的字符串，满足任一前缀中Ｘ的数量大于Ｙ的数量的个数

卡特兰数公式：Cn = (2n)! / ((n+1)! \* n!)

递推公式：Cn = (2 \* (2n - 1) / (n + 1)) \* Cn-1 C0 = 1

ll mul(ll a, ll b){

ll res = 1;

while(b > 0){

if(b&1) res \*= a;

res %= mod;

a \*= a;

a %= mod;

b >>= 1;

}

res %= mod;

return res;

}

//直接使用公式计算

ll solve(ll n){

ll res = 1;

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

res \*= (n+i);

res %= mod;

res \*= mul(i, mod - 2);

res %= mod;

}

res \*= mul(n+1, mod - 2);

res %= mod;

return res;

}

ll ans[10110];

## 高斯消元

TYPE: 高斯消元 异或消元

DETAIL: 给n个数，保证每个数的质因子最大不会超过2000，问有多少中取法能够使得取出的数的

乘机是完全平方数

TATICS: 将每一个数质因子分解，形成一个0,1矩阵，0表示当前质因子出现偶数次，否则出现奇

数次，然后进行高斯消元，注意消元的时候使用异或消元。

求出秩r，结果为 2^r - 1

#define CL(a) memset(a, 0, sizeof(a))

const int inf = 1e9+7;

const double eps = 1e-6;

const int maxn = 305, maxe = 2001;

int prime[303], isNotPrime[maxe], cnt;

void init(){

CL(prime);

CL(isNotPrime);

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

if(!isNotPrime[i]){

prime[cnt++] = i;

}

for(int j = 0; j < cnt && prime[j] \* i < maxe; j ++){

isNotPrime[i \* prime[j]] = 1;

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

break;

}

}

}

struct Mat{

int mat[maxn][maxn];

int n, m;

Mat(){

n = maxn, m = maxn;

}

Mat(int \_n):n(\_n), m(maxn){}

void init(){

CL(mat);

}

//求秩

int Rank(){

int i, j, k, col, max\_r, res = 0;

for(k = 0, col = 0; k < n && col < m; k ++, col++){

max\_r = k;

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

if(mat[i][col] > mat[max\_r][col]){

max\_r = i;

}

if(mat[max\_r][col] == 0){

k --;

continue;

}

if(k != max\_r){

for(j = col; j < m; j ++){

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

}

}

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

if(mat[i][col]){

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

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

}

}

}

res ++;

}

return res;

}

void print(){

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

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

printf("%d%c", mat[i][j], j == m - 1 ? '\n' : ' ');

}

}

}

};

ll mul(ll a, ll b){

ll res = 1;

while(b > 0){

if(b & 1) res \*= a;

res %= mod;

a \*= a;

a %= mod;

b >>= 1;

}

return res;

}

int main()

{

init();

for(i = 0; i < cnt; i ++){

if(prime[i] > 2000)

break;

printf("%d %d", prime[i-1], i); int T, n, tt = 1;

ll tmp;

scanf("%d", &T);

while(T --){

scanf("%d", &n);

Mat m(n);

m.init();

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

scanf("%I64d", &tmp);

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

if(tmp % prime[j] == 0){

int ct = 0;

while(tmp % prime[j] == 0){

tmp /= prime[j];

ct ++;

}

m.mat[i][j] = ct & 1;

}

}

}

int r = n - m.Rank();

ll ans = mul(2, r) - 1;

ans = (ans + mod) % mod;

printf("Case #%d:\n%I64d\n",tt ++, ans);

}

return 0;

}

## 欧拉函数 & 素数表

int prime[N], isNotPrime[N], cnt, phi[N]; //prime存放素数，phi为欧拉函数

void init() {

memset(prime, 0, sizeof(prime));

memset(isNotPrime, 0, sizeof(isNotPrime));

memset(phi, 0, sizeof(phi));

cnt = 0;

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

if (!isNotPrime[i]) {

prime[cnt++] = i;

phi[i] = i - 1;

}

for (int j = 0; j < cnt && prime[j] \* i < N; j++) {

isNotPrime[prime[j] \* i] = 1;

if (i % prime[j] != 0) {

phi[i\*prime[j]] = phi[i] \* (prime[j] - 1); //根据欧拉定理可得，当i是j的倍数时，phi[i\*j] = phi[i]\*(j-1); 其中i是素数

}

else{

phi[i\*prime[j]] = phi[i] \* prime[j]; //当不是倍数时，phi[i\*j] = phi[i]\*(j); 其中i是素数

break; // 保证每个数只被筛了一次

}

}

}

}

## 模线性方程

ll exgcd(ll a, ll b, ll& x, ll& y) { //扩展欧几里得算法，求解ax + by = d = gcd(a, b)的解

if (b == 0) {

x = 1;

y = 0;

return a;

}

ll d = exgcd(b, a%b, x, y), tmp;

tmp = x;

x = y;

y = tmp - a / b\*y;

return d;

}

//求解模线性方程，ax = b (mod n)

//该方程当且仅当 b|d 时有d个解，每个解之间相差n/d

void modular\_linear(ll a, ll b, ll n) {

if (b == 0) {

puts("0");

return;

}

ll d, x, y;

d = exgcd(a, n, x, y);

if (b % d != 0) {

printf("FOREVER\n");

}

else {

x = (x\*b / d) % n;

x = (x % (n / d) + n / d) % (n / d); //求得最小的正数解 其余解为 (x+ i \* n / d) % n (i = 1,2,3...)

printf("%I64d\n", x);

}

}

## 矩阵快速幂

const int N = 2;

const int mod = 10000;

struct mat {

int m[N][N];

void init() {

memset(m, 0, sizeof(m));

}

};

mat mul(mat a, mat b) {

mat c;

c.init();

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

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

for (int k = 0; k < N; k++) {

c.m[i][j] += a.m[i][k] \* b.m[k][j];

c.m[i][j] %= mod;

}

return c;

}

mat multi(mat a,int n) {

mat c;

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

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

c.m[i][j] = (i == j);

if (n == 0) {

return c;

}

while (n > 0) { //与乘法快速幂类似

if (n & 1) c = mul(c, a);

a = mul(a,a);

n >>= 1;

}

return c;

}

## 线性筛法求素数

bool a[N];

int prime[N], num;

//a[i] = 0表示i为素数

//prime[i]存储第i个素数

//num存储一共多少个素数

//n表示最大界,但是不包括n

void Prime(int n) {

memset(a, 0, n \* sizeof(a[0]));

num = 0;

a[0] = a[1] = 1;

//不要冒昧的吧<改成<=

//不然会错。亲测

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

if(!(a[i])) prime[num ++] = i;

for(int j = 0; j < num && i \* prime[j] < n; ++j){

a[i \* prime[j]] = 1;

if(!(i % prime[j])) break;

}

}

}

## 线性求中位数

//下标从零开始

int find\_mid(int arr[], int left, int right, int x){

if(left >= right){

return arr[left + x];

}

int mid = arr[left];

int i = left;

int j = right;

while(i < j){

while(i < j && arr[j] >= mid) j--;

arr[i] = arr[j];

while(i < j && arr[i] <= mid) i++;

arr[j] = arr[i];

}

arr[j] = mid;

if(i - left == x)

return arr[i];

if(i - left < x)

return find\_mid(arr, i + 1, right, x - (i - left + 1));

else

return find\_mid(arr, left, i - 1, x);

}

## 求素数个数

long long f[340000], g[340000], n;

void init(){

long long i, j, m;

for(m = 1; m \* m <= n; m ++)

f[m] = n / m - 1;

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

g[i] = i - 1;

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

if(g[i] == g[i - 1])

continue;

for(j = 1; j <= min(m - 1, n / i / i); j ++){

if(i \* j < m)

f[j] -= f[i \* j] - g[i - 1];

else

f[j] -= g[n / i / j] - g[i - 1];

}

for(j = m; j >= i \* i; -- j)

g[j] -= g[j / i] - g[i - 1];

}

}

## 快速幂

long long multi(long long a, long long b, long long mod){

long long ret;

ret = 1;

while(b > 0) {

if(b & 1)

ret = ret \* a % mod;

a = (a \* a) % mod;

b = b >> 1;

}

return ret;

}

## 归并排序求逆序数

int b[500005], a[500005];

long long ans;

void merge(int l, int r, int mid){

int last = mid + 1, temp = l;

while(l <= mid && last <= r){

if(a[l] <= a[last])

b[temp ++] = a[l ++];

else{

ans += mid - l + 1;

b[temp ++] = a[last ++];

}

}

while(l <= mid)

b[temp ++] = a[l ++];

while(last <= r)

b[temp ++] = a[last ++];

}

void mergesort(int l, int r){

if(l >= r)

return ;

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

mergesort(l, mid);

mergesort(mid + 1, r);

merge(l, r, mid);

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

a[i] = b[i];

}

# 计算几何

## **计算几何基础模板**

#define CL(a) memset(a, 0, sizeof(a))

const int inf = 1e9+7;

const int mod = 1e9+7;

const int maxn = 1e6+7;

const double eps = 1e-9;

struct Point{

double x, y;

Point(double \_x = 0, double \_y = 0):x(\_x), y(\_y){}

};

//向量与点等价，表示从原点到这个点的向量

typedef Point Vector;

Vector operator + (Vector A, Vector B){return Vector(A.x+B.x, A.y+B.y);}

Vector operator - (Vector A, Vector B){return Vector(A.x-B.x, A.y-B.y);}

Vector operator \* (Vector A, double p){return Vector(A.x\*p, A.y\*p);}

Vector operator / (Vector A, double p){return Vector(A.x/p, A.y/p);}

bool operator < (const Vector& A, const Vector& B){

return A.x < B.x || (A.x == B.x && A.y < B.y);

}

//判断浮点数的正负

int dcmp(double x){

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

else return x < 0 ? -1 : 1;

}

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

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

}

/\* 求极角

向量(x, y) 的级角为 atan2(y, x);

\*/

//点积

double Dot(Vector A, Vector B){return A.x\*B.x + A.y\*B.y;}

//利用点积求长度

double Length(Vector A){return sqrt(Dot(A, A));}

//求两个向量的角度

double Angle(Vector A, Vector B){

return acos(Dot(A, B)/ Length(A)/Length(B));

}

//叉积

double Cross(Vector A, Vector B){return A.x\*B.y - A.y\*B.x;}

double Area(Point A, Point B, Point C){return Cross((B-A), C-A)/2;}

//将向量旋转一定的角度

Vector Rotate(Vector A, double rad){

return Vector(A.x\*cos(rad)-A.y\*sin(rad) ,A.x\*sin(rad) + A.y\*cos(rad));

}

//求一个向量的法线，即旋转90°再单位化

Vector Normal(Vector A){

double L = Length(A);

return Vector(-A.y/L, A.x/L);

}

//直线可以表示成一个起点P和方向向量v l: P+tv t为参数

//若已知直线上的两个点，则参数方程为A+(B-A)t

//对于上面的参数方程，线段 0 < t < 1 射线 t > 0

//求直线交点

//调用前确保两条之前有且只有一个交点，当且仅当Cross(v, w) != 0;

Point GetLineIntersection(Point P, Vector v, Point Q, Vector w){

Vector u = P - Q;

double t = Cross(w, u) / Cross(v, w);

return P + v \* t;

}

//点到直线距离，利用叉积

double DistanceToLine(Point P, Point A, Point B){

Vector v1 = B - A, v2 = P - A;

return fabs(Cross(v1, v2)) / Length(v1);

}

//点到线段距离，需要考虑点到线段的垂线是否在线段上

double DistanceToSegment(Point P, Point A, Point B){

if(A == B) return Length(P-A);

Vector v1 = B - A, v2 = P - A, v3 = P - B;

if(dcmp(Dot(v1, v2)) < 0) return Length(v2);

else if (dcmp(Dot(v1, v3)) > 0) return Length(v3);

else return fabs(Cross(v1, v2)) / Length(v1);

}

//求点在直线上的投影

Point GetLineProjection(Point P, Point A, Point B){

Vector v = B - A;

return A + v\*(Dot(v, P-A) / Dot(v, v));

}

//判断点是否在线段上（可以在端点）

bool onSegment(Point p, Point a1, Point a2){

return dcmp(Cross(a1-p, a2-p) == 0) && dcmp(Dot(a1-p, a2-p)) <= 0;

}

//判断两条线段是否相交，不包含端点相交的情况

bool IsSegmaIntersection(Point a1, Point a2, Point b1, Point b2){

double c1 = Cross(a2-a1, b1-a1);

double c2 = Cross(a2-a1, b2-a1);

double c3 = Cross(b2-b1, a1-b1);

double c4 = Cross(b2-b1, a2-b1);

return dcmp(c1)\*dcmp(c2) < 0 && dcmp(c3) \* dcmp(c4) < 0;

}

//多边形求面积，适用于凸多边形和凹多边形

double ConvexPolygonArea(Point \*p, int n){

double res = 0;

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

res += Cross(p[i] - p[0], p[i+1] - p[0]);

return res / 2;

}

//多边形求面积，适用于凸多边形和凹多边形

double ConvexPolygonArea(Point \*p, int n){

double res = 0;

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

res += Cross(p[i] - p[0], p[i+1] - p[0]);

return res / 2;

}

## **求多边形面积**

//任选一个点，按照逆时针或顺时针方向，与每两个点依次做叉积，所得结果除以2就是多边形的面积

## **判断两直线是否相交**

//判断两个直线是否相交，只需要没个线段两个端点分别位于另一条线段的两侧。

//对于AB CD 两条线段，若AB向量与AC向量叉乘大于零，则说明C点位于AB直线的逆时针方向，若等于零说明C点在AB之线上，若小于零说明C点在AB直线的顺时针方向，通过这个特点来判断两点是否位于一条直线的同一侧。

bool intersect(point x1, point y1, point x2, point y2){

double d1, d2, d3, d4;

//d1,d2表示x1，y1是否位于x2->y2这条向量的两侧

d1 = (y2-x2)\*(x1-x2);

d2 = (y2-x2)\*(y1-x2);

//d3,d4同理

d3 = (y1-x1)\*(x2-x1);

d4 = (y1-x1)\*(y2-x1);

if(d1\*d2<0 && d3\*d4<0)

return true;

return false;

}

## **求两个圆相交面积**

const int N = 30;

const double eps = 1e-8;

const double Pi = acos(-1.0);

struct point{

double x, y;

point(){}

point(double \_x, double \_y):x(\_x), y(\_y){}

point operator + (point t){

return point(x+t.x, y+t.y);

}

point operator - (point t){

return point(x-t.x, y-t.y);

}

point operator \* (double a){

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

}

point operator / (double a){

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

}

double operator \* (point t){ // 向量叉积，返回向量|a|\*|b|\*sin(θ)方向垂直u,v 且遵循右手定则

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

}

double operator ^ (point t){ // 向量点积，返回值 |a|\*|b|\*cos(θ)

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

}

double len(){

return sqrt( x\*x + y\*y );

}

};

struct circle{

//圆心

point o;

//半径

double r;

circle(){}

circle(point \_o, double \_r):o(\_o), r(\_r){}

double area(){

return Pi\*r\*r;

}

//求两个圆相交面积

double interarea(circle t){

double d = (t.o-o).len();

//相离的情况

if(d > r + t.r + eps )

return 0;

//内含的情况

if(t.r+d < r - eps)

return t.area();

if(r+d < t.r - eps)

return area();

double xita = acos((r\*r+d\*d-t.r\*t.r)/(2\*r\*d));

double arf = acos((t.r\*t.r+d\*d-r\*r)/(2\*t.r\*d));

//a圆对应扇形的面积

double S1 = xita\*r\*r;

//b圆对应扇形的面积

double S2 = arf\*t.r\*t.r;

//两圆圆心和交点构成的四边行的面积

double S3 = d\*r\*sin(xita);

//相交面积

return S1+S2-S3;

}

}cr[N];

## 凸包

//凸包求法，按照x轴排序，则第一个点一定是凸包的顶点。枚举后面每一个点，若当前栈中只有一个点，则直接入栈，否则判断是否发生右旋，若发生右旋，将栈顶的元素弹出，直到发生左旋，讲当前点入栈，从前向后扫一边，从后向前扫一边就能求出凸包

void Convex(point \*p, int& n){

int i, j, r, top, m;

sort(p, p+n, cmp);

//先入队前两个点

s[0] = p[0];

s[1] = p[1];

top = 1;

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

//这里判断是否发生左旋的时候>=0，若等于0说明当前点是凸包边上的一个顶点，可以不用考虑，如果要求凸包所有的顶点的话则改成>0

while(top>0 && (p[i]-s[top-1])\*(s[top]-s[top-1]) >= 0)top--;

s[++top] = p[i];

}

m = top;

//最后一个点当前一定位于栈顶

s[++top] = p[n-2];

for(i = n-3; i >= 0; i --){

while(top>m && (p[i]-s[top-1]) \* (s[top]-s[top-1]) >= 0)top--;

s[++top] = p[i];

}

//当前的栈顶是第一个点，top就是凸包的顶点数

n = top;

}

int main(){

int n;

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

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

scanf("%lf%lf", &no[i].x, &no[i].y);

}

Convex(no, n);

double ans = 0;

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

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

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

ans = max( fabs((s[i]-s[j]) \* (s[i]-s[k])) / 2, ans);

}

}

}

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

}

return 0;

}

# 其他

## 二层魔方

int B[6][24]={ {6,1,12,3,5,11,16,7,8,9,4,10,18,13,14,15,20,17,22,19,0,21,2,23}, //ok

{20,1,22,3,10,4,0,7,8,9,11,5,2,13,14,15,6,17,12,19,16,21,18,23}, //ok

{1,3,0,2,23,22,4,5,6,7,10,11,12,13,14,15,16,17,18,19,20,21,9,8}, //ok

{2,0,3,1,6,7,8,9,23,22,10,11,12,13,14,15,16,17,18,19,20,21,5,4}, //ok

{0,1,8,14,4,3,7,13,17,9,10,2,6,12,16,15,5,11,18,19,20,21,22,23}, //ok

{0,1,11,5,4,16,12,6,2,9,10,17,13,7,3,15,14,8,18,19,20,21,22,23} //ok

};

## 从数组取n个元素组合

void combine\_increase(const int \*numbers, int \*result, const int arrysize,const int elements, int current = 0, int start = 0){

for(int i = start; i <= arrysize - elements + current; i ++){

result[current] = i;

if(elements - current - 1){

combine\_increase(numbers, result, arrysize, elements, current + 1, i + 1);

}

else{

for(int j = current; j >= 0; j --){

printf("%d\t", numbers[result[current - j]]);

}

printf("\n");

}

}

}