HsiaoYeekwan

### [Shandong University of Science and Technology](https://www.baidu.com/link?url=NbFfUnERsWOB_C0lVGBKcyM62tMiiT9rM1oxHJrjP_dLvbxdwTVP3ZcSG4cDpmzV&wd=&eqid=d64b16260000bfc4000000025aeb1f4f)

Code Library

May 2018

目录

[图论 4](#_Toc7816822)

[链式前向星 4](#_Toc7816823)

[最短路 5](#_Toc7816824)

[SPFA 5](#_Toc7816825)

[SLF优化 5](#_Toc7816826)

[判定负环 6](#_Toc7816827)

[Dijkstra 7](#_Toc7816828)

[堆优化 7](#_Toc7816829)

[次短路（长度严格大于最短路） 8](#_Toc7816830)

[最短路and次短路计数 8](#_Toc7816831)

[Floyd 9](#_Toc7816832)

[最小环 9](#_Toc7816833)

[LCA&&RMQ 10](#_Toc7816834)

[倍增在线LCA 10](#_Toc7816835)

[RMQ 10](#_Toc7816836)

[LCA转RMQ 11](#_Toc7816837)

[树 12](#_Toc7816838)

[DP求树直径 12](#_Toc7816839)

[强连通&&双连通 13](#_Toc7816840)

[Tarjan（scc）缩点 13](#_Toc7816841)

[割点与桥 14](#_Toc7816842)

[网络流 18](#_Toc7816843)

[Dinic 18](#_Toc7816844)

[ISAP优化 20](#_Toc7816845)

[有上下界的网络流 20](#_Toc7816846)

[无源无汇可行流（循环流） 20](#_Toc7816847)

[有源有汇可行流 25](#_Toc7816848)

[有源汇网络的最大流 25](#_Toc7816849)

[有源汇网络的最小流 26](#_Toc7816850)

[最小割模型 26](#_Toc7816851)

[输出割集 26](#_Toc7816852)

[最小冲突数 26](#_Toc7816853)

[最大权闭合子图 29](#_Toc7816854)

[最小点权覆盖 30](#_Toc7816855)

[数据结构 30](#_Toc7816856)

[Treap 30](#_Toc7816857)

[Treap实现名次树 30](#_Toc7816858)

[Splay(伸展树) 33](#_Toc7816859)

[Trie 41](#_Toc7816860)

[可持久化字典树 41](#_Toc7816861)

[树状数组 42](#_Toc7816862)

[二维树状数组 42](#_Toc7816863)

[线段树 44](#_Toc7816864)

[区间更新&&查询 44](#_Toc7816865)

[扫描线 45](#_Toc7816866)

[主席树 48](#_Toc7816867)

[线性基： 51](#_Toc7816868)

[计算几何 52](#_Toc7816869)

[基础 52](#_Toc7816870)

[点和线段(直线)相关 54](#_Toc7816871)

[多边形相关(凸包) 56](#_Toc7816872)

[圆和球相关 59](#_Toc7816873)

[极角排序 62](#_Toc7816874)

[半平面交 63](#_Toc7816875)

[裸半平面交求面积： 63](#_Toc7816876)

[字符串 65](#_Toc7816877)

[字符串Hash 65](#_Toc7816878)

[后缀数组 68](#_Toc7816879)

[白书SA + LCP 比较的是（i,i+1） 68](#_Toc7816880)

[蓝书SA+LCP(i-1,i) m大了需要离散 69](#_Toc7816881)

[AC自动机 69](#_Toc7816882)

[KMP 72](#_Toc7816883)

[最小(大)表示法(O(n)) 73](#_Toc7816884)

[数学 74](#_Toc7816885)

[素数相关 74](#_Toc7816886)

[Miller-Rabin 素性测试 74](#_Toc7816887)

[反素数 75](#_Toc7816888)

[线性素数筛(O(n)) 76](#_Toc7816889)

[gcd相关 77](#_Toc7816890)

[EXGCD - 扩展欧几里得定理 77](#_Toc7816891)

[欧拉函数相关 77](#_Toc7816892)

[筛法求欧拉函数 78](#_Toc7816893)

[莫比乌斯反演 78](#_Toc7816894)

[高斯消元 79](#_Toc7816895)

[中国剩余定理 84](#_Toc7816896)

[模数互质 84](#_Toc7816897)

[模数不互质 84](#_Toc7816898)

[组合数学相关 86](#_Toc7816899)

[卡特兰数 86](#_Toc7816900)

[快速傅里叶变换 87](#_Toc7816901)

[动态规划 89](#_Toc7816902)

[LIS打印路径 89](#_Toc7816903)

[数位DP 90](#_Toc7816904)

[杂项 90](#_Toc7816905)

[快速乘法(直接乘超过longlong时使用) 90](#_Toc7816906)

[位运算相关 91](#_Toc7816907)

[希尔伯特(Hilbert)曲线 92](#_Toc7816908)

[快速输入函数 93](#_Toc7816909)

[头文件 94](#_Toc7816910)

# 图论

## 链式前向星

int head[maxn] , Next[maxn] , tot;

struct node//链式前向星

{

int to,cost,Next;

node(){}

node(int tt,int cc,int nn):to(tt),cost(cc),Next(nn){}

};

node edge[maxn];

void init()

{

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

tot = 0;//网络流通过异或取反向边时初始化为2！

}

void addedge(int u,int v,int w)

{

edge[tot].cost = w;

edge[tot].to = v;

edge[tot].Next = head[u];

head[u] = tot++;

}

## 最短路

### SPFA

#### SLF优化

int d[maxn] , vis[maxn];

struct node

{

int to,cost;

node(){}

node(int tt,int cc):to(tt),cost(cc){}

};

vector<node> G[maxn];

void spfa(int s)

{

fill(d , d +maxn , inf);

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

d[s] = 0;

deque<int> q;

q.push\_back(s);

vis[s] = 1;

while(!q.empty()){

int u = q.front();

q.pop\_front();

vis[u] = 0;

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

int v = G[u][i].to;

int cost = G[u][i].cost;

if(d[v] > d[u] +cost){

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

if(!vis[v]){

//SLF优化

if(!q.empty() && d[v] < d[q.front()]){

q.push\_front(v);

}

else{

q.push\_back(v);

}

vis[v] = 1;

}

}

}

}

}

#### 判定负环

**//存在负环返回false，不存在返回true，调用spfa(1)**

bool spfa(int s){ //s为起始点

fill(d , d +maxn , inf);

d[s] = 0;

memset(vis,false,sizeof(vis));

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

queue<node> q;

q.push(node(s , 0));

vis[s] = 1;

while(!q.empty()){

node p = q.front();

q.pop();

int u = p.to;

cnt[u] ++;

vis[u] = 0;

if(cnt[u] >= n)return false;

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

int v = G[u][i].to;

int cost = G[u][i].cost;

if(d[v] > d[u] + cost){

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

if(!vis[v]){

q.push(node(v , d[v]));

vis[v] = true;

}

}

}

}

return true;

}

### Dijkstra

#### 堆优化

struct node{

int to,cost;

node(){}

node(int tt,int cc):to(tt),cost(cc){}

bool operator < (const node &a) const

{

return cost > a.cost;

}

};

int n,m;

vector <node> G[maxn];

int vis[maxn],d[maxn];

void Dijkstra(int s){ //s为起始点

fill(d , d+maxn , inf);

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

d[s] = 0;

priority\_queue<node> q;

q.push(node(s , 0));

while(!q.empty()){

node p = q.top();

int u = p.to;

q.pop();

if(vis[u])continue;

vis[u] = 1;

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

int v = G[u][i].to;

int cost = G[u][i].cost;

if(d[v] > d[u] + cost){

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

q.push(node(v , d[v]));

}

}

}

}

#### 次短路（长度严格大于最短路）

void Dij(){

memset(d,inf,sizeof(d));

d[s][0] = 0;

priority\_queue<P , vector<P> , greater<P> > q; //pair两个参数分别为最短距离、节点编号

q.push(P(0,s));

while(!q.empty()){

P p = q.top();

q.pop();

int u = p.second;

if(d[u][1] < p.first)continue;

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

node e = G[u][i];

int nowcost = p.first + e.cost;

if(d[e.to][0] > nowcost){

int t = nowcost;

nowcost = d[e.to][0];

d[e.to][0] = t;

q.push(P(d[e.to][0] , e.to));

}

if(d[e.to][1] > nowcost && d[e.to][0] < nowcost){

d[e.to][1] = nowcost;

q.push(P(d[e.to][1] , e.to));

}

}

}

}

#### 最短路and次短路计数

cnt[u][0]是最短路条数、cnt[u][1]是次短路条数、

void Cal(){

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

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

cnt[s][0] = 1;

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

q.push(P(0,s));

while(!q.empty()){

P p = q.top();

q.pop();

int u = p.second;

if(vis[u][1])continue;

int now = 0;

if(vis[u][0])now = 1;

vis[u][now] = 1;

if(d[u][now] >= inf)continue;

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

node e = G[u][i];

if(d[e.to][0] == d[u][now] + e.cost){

q.push(P(d[e.to][0] , e.to));

cnt[e.to][0] += cnt[u][now];

}

if(d[e.to][1] == d[u][now] + e.cost){

q.push(P(d[e.to][1] , e.to));

cnt[e.to][1] += cnt[u][now];

}

}

}

}

### Floyd

#### 最小环

void floyd()

{

ans = inf;

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

//floyd求最小环,ans为最小环（最少三个点）长度

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

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

ans = min(ans , d[i][j] + maps[i][k] + maps[k][j]);

}

}

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

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

d[i][j] = min(d[i][j] , d[i][k] + d[k][j]);

}

}

}

}

## LCA&&RMQ

### 倍增在线LCA

int dep[maxn];//节点深度

int fa[maxn][20];

void dfs(int u,int f)

{

dep[u] = dep[f] + 1;

fa[u][0] = f;

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

int v = G[u][i].to;

if(v == f)continue;

dfs(v , u);

}

}

int LCA(int u , int v)

{

if(dep[u] > dep[v]){

swap(u,v);

}

for(int i = dep[v] - dep[u] , j = 0 ; i > 0 ; i >>= 1 , j ++){

if(i & 1){

v = fa[v][j];

}

}

if(u == v)return u;

int k;

for(k = 0 ; (1 << k) <= dep[u]; k ++);

for(; k >= 0 ; k --){

if((1 << k) <= dep[u] && fa[u][k] != fa[v][k]){

u = fa[u][k];

v = fa[v][k];

}

}

return fa[u][0];

}

### RMQ

void ST()

{

int i,j;

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

f[i][0]=i;

for(j=1;j<=(int)(log((double)n)/log(2.0));j++)

{

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

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

}

}

int query(int L,int R)

{

int x=(int)(log(double(R-L+1))/log(2.0));

return min(f[L][x] , f[R-(1<<x)+1][x]);

}

### LCA转RMQ

int head[maxn];

int first[maxn];//首次出现的下标

int dp[maxn\*2][30];

bool vis[maxn];

bool ok[maxn];

int deep[maxn\*2];//深度数组

int ver[maxn\*2];//节点序列

int n,cnt,tot;

struct Edge{

int next;

int to;

}edge[maxn];

void addedge(int u, int v){

edge[cnt].to = v;

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

head[u] = cnt++;

}

void dfs(int u, int step){

vis[u] = true;

ver[tot] = u;

first[u] = tot;

deep[tot++] = step;

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

if(!vis[edge[i].to]){

dfs(edge[i].to,step+1);

ver[tot] = u;

deep[tot++] = step;

}

}

}

void st(int n){

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

dp[i][0] = i;//保存的下标

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

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

int a = dp[i][j-1];

int b = dp[i+(1<<(j-1))][j-1];

dp[i][j] = deep[a] < deep[b] ? a : b;

}

}

int rmq(int x, int y){

int k = (int)(log((y-x+1)\*1.0)/log(2.0));

int a = dp[x][k];

int b = dp[y-(1<<k)+1][k];

if(deep[a] < deep[b]) return a;

return b;

}

int lca(int u, int v){

int x = first[u], y = first[v];//首次出现的下标

if(x > y) swap(x,y);

int ans = rmq(x,y);//得到的是下标

return ver[ans];

}

void init(){

cnt = tot = 0;

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

memset(vis, false, sizeof(vis));

memset(ok, false, sizeof(ok));

}

## 树

### DP求树直径

void DP(int u)

{

vis[u] = 1;

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

int v = edge[i].to;

if(vis[v])continue;

DP(v);

ans = max(ans , d[u] + d[v] + edge[i].cost);

d[u] = max(d[u] , d[v] + edge[i].cost);

}

}

主函数调用：DP(1),ans即为所求树直径。

## 强连通&&双连通

### Tarjan（scc）缩点

struct node

{

int to,Next,cost;

};

node edge[maxn\*5];

int n , m , tot , index;

int head[maxn] , low[maxn] , dfn[maxn] ;

int vis[maxn];//标记是否在栈中

int id[maxn] ;//id[i]表示点i属于的强连通分量的编号

int scc\_cnt ;//强连通分量的数量

void addedge(int u,int v , w)

{

edge[++tot].to = v;

edge[tot].cost = w;

edge[tot].Next = head[u];

head[u] = tot;

}

void init()

{

tot = 1;

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

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

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

index = scc\_cnt = 0;

}

void Tarjan(int u)

{

dfn[u] = low[u] = ++index;

s.push(u);

vis[u] = 1;

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

int v = edge[i].to;

if(!dfn[v]){

Tarjan(v);

low[u] = min(low[u] , low[v]);

}

else if(vis[v]){

low[u]= min(low[u] , dfn[v]);

}

}

int temp;

if(low[u] == dfn[u]){

scc\_cnt ++;

while(1){

temp = s.top();

s.pop();

vis[temp] = false;

id[temp] = scc\_cnt;

if(temp == u)break;

}

}

}

//主函数调用

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

if(!dfn[i]){

Tarjan(i);

}

}

### 割点与桥

#include <stdio.h>

#include <algorithm>

#include <iostream>

#include <string.h>

using namespace std;

/\*

\* 求 无向图的割点和桥

\* 可以找出割点和桥，求删掉每个点后增加的连通块。

\* 需要注意重边的处理，可以先用矩阵存，再转邻接表，或者进行判重

\*/

const int MAXN = 10010;

const int MAXM = 100010;

struct Edge

{

int to,next;

bool cut;//是否为桥的标记

}edge[MAXM];

int head[MAXN],tot;

int Low[MAXN],DFN[MAXN],Stack[MAXN];

int Index,top;

bool Instack[MAXN];

bool cut[MAXN];

int add\_block[MAXN];//删除一个点后增加的连通块

int bridge;

void addedge(int u,int v)

{

edge[tot].to = v;edge[tot].next = head[u];edge[tot].cut = false;

head[u] = tot++;

}

void Tarjan(int u,int pre)

{

int v;

Low[u] = DFN[u] = ++Index;

Stack[top++] = u;

Instack[u] = true;

int son = 0;

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

{

v = edge[i].to;

if(v == pre)continue;

if( !DFN[v] )

{

son++;

Tarjan(v,u);

if(Low[u] > Low[v])Low[u] = Low[v];

/\*

//桥

//一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足DFS(u)<Low(v)。

if(Low[v] > DFN[u])

{

bridge++;

edge[i].cut = true;

edge[i^1].cut = true;

}\*/

//割点

//一个顶点u是割点，当且仅当满足(1)或(2) (1) u为树根，且u有多于一个子树。

//(2) u不为树根，且满足存在(u,v)为树枝边(或称父子边，

//即u为v在搜索树中的父亲)，使得DFS(u)<=Low(v)

if(u != pre && Low[v] >= DFN[u])//不是树根

{

cut[u] = true;

add\_block[u]++;

}

}

else if( Low[u] > DFN[v])

Low[u] = DFN[v];

}

//树根，分支数大于1

if(u == pre && son > 1)cut[u] = true;

if(u == pre)add\_block[u] = son - 1;

Instack[u] = false;

top--;

}

void solve(int N)

{

memset(DFN,0,sizeof(DFN));

memset(Instack,false,sizeof(Instack));

memset(add\_block,0,sizeof(add\_block));

memset(cut,false,sizeof(cut));

Index = top = 0;

bridge = 0;

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

if(!DFN[i])

Tarjan(i,i);

int ans = 0;

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

if(cut[i])

ans++;

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

}

void init()

{

tot = 0;

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

}

int g[110][110];

char buf[1010];

int main()

{

int n;

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

{

gets(buf);

memset(g,0,sizeof(g));

while(gets(buf))

{

if(strcmp(buf,"0")==0)break;

char \*p = strtok(buf," ");

int u;

sscanf(p,"%d",&u);

p = strtok(NULL," ");

int v;

while(p)

{

sscanf(p,"%d",&v);

p = strtok(NULL," ");

g[u][v]=g[v][u]=1;

}

}

init();

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

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

if(g[i][j])

{

addedge(i,j);

addedge(j,i);

}

solve(n);

}

return 0;

}

## 网络流

### Dinic

最坏复杂度O(V^2\*E)，实际跑的很快

const int maxn = 1e3 + 10;//点

const int maxm = 22000//边

struct Edge

{

int from , to , cap , flow;

int l , r;//边容量的上下界

Edge(){}

Edge(int u , int v , int c , int f , int ll , int rr):from(u) , to(v) , cap(c) , flow(f),l(ll) , r(rr){}

};

vector<Edge> edge;

vector<int> G[maxn];

bool vis[maxn];

int d[maxn] , cur[maxn];

int s , t;

void init()

{

edge.clear();

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

G[i].clear();

}

}

void addedge(int u , int v , int c , int l , int r)

{

edge.push\_back(Edge(u , v , c , 0 , l , r));

edge.push\_back(Edge(v , u , 0 , 0 , l , r));

int sz = edge.size();

G[u].push\_back(sz-2);

G[v].push\_back(sz-1);

}

bool bfs()

{

memset(vis , false, sizeof vis);

queue<int> q;

q.push(s);

d[s] = 0;

vis[s] = 1;

while(!q.empty()){

int x = q.front();

q.pop();

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

Edge &e = edge[G[x][i]];

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

vis[e.to] = 1;

d[e.to] = d[x] + 1;

q.push(e.to);

}

}

}

return vis[t];

}

int dfs(int x , int a)

{

if(x == t || !a)return a;

int flow = 0 , f;

for(int &i = cur[x] ; i < G[x].size() ; i ++){

Edge &e = edge[G[x][i]];

if(d[x] + 1 == d[e.to] && (f = dfs(e.to , min(a , e.cap - e.flow))) > 0){

e.flow += f;

edge[G[x][i] ^ 1].flow -= f;

flow += f;

a -= f;

if(!a)break;

}

}

return flow;

}

int dinic()

{

int flow = 0;

while(bfs()){

memset(cur , 0 , sizeof(cur));

flow += dfs(s , inf);

}

return flow;

}

### ISAP优化

### 有上下界的网络流

#### 无源无汇可行流（循环流）

对于每条边上界为R，下界为L，建图过程中使边的下界为0，上界为R-L，增加超级原点s和超级汇点t。

d[i]=in[i]（i节点所有入流下界之和）-out[i]（i节点所有出流下界之和）。

若当du[i]大于0的时候，s到i连一条流量为du[i]的边。

当du[i]小于0的时候，i到t连一条流量为-du[i]的边。

最后对（s，t）求一次最大流即可，当所有附加边全部满流时（即maxflow==所有du[]>0之和），有可行解。原图边的流量等于edge[i].flow + edge[i].L。

SGU-194

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

#include <iostream>

#include <cstring>

#include <cmath>

#include <cstdlib>

#include <cstdio>

#include <string>

#include <algorithm>

#include <queue>

#include <map>

#include <vector>

#include <stack>

#include <cctype>

#include <set>

#include <ctype.h>

#include <string.h>

#define inf 0x3f3f3f3f

#define eps 1e-10

#define key\_value ch[ch[root][1]][0]

using namespace std;

typedef long long ll;

typedef pair<int,int> P;

const int maxn = 1e3 + 10;

struct Edge

{

int from , to , cap , flow;

int l , r;//边容量的上下界

Edge(){}

Edge(int u , int v , int c , int f , int ll , int rr):from(u) , to(v) , cap(c) , flow(f),l(ll) , r(rr){}

};

vector<Edge> edge;

vector<int> G[maxn];

bool vis[maxn];

int d[maxn] , cur[maxn];

int n , m , s , t;

void init()

{

edge.clear();

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

G[i].clear();

}

}

void addedge(int u , int v , int c , int l , int r)

{

edge.push\_back(Edge(u , v , c , 0 , l , r));

edge.push\_back(Edge(v , u , 0 , 0 , l , r));

int sz = edge.size();

G[u].push\_back(sz-2);

G[v].push\_back(sz-1);

}

bool bfs()

{

memset(vis , false, sizeof vis);

queue<int> q;

q.push(s);

d[s] = 0;

vis[s] = 1;

while(!q.empty()){

int x = q.front();

q.pop();

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

Edge &e = edge[G[x][i]];

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

vis[e.to] = 1;

d[e.to] = d[x] + 1;

q.push(e.to);

}

}

}

return vis[t];

}

int dfs(int x , int a)

{

if(x == t || !a)return a;

int flow = 0 , f;

for(int &i = cur[x] ; i < G[x].size() ; i ++){

Edge &e = edge[G[x][i]];

if(d[x] + 1 == d[e.to] && (f = dfs(e.to , min(a , e.cap - e.flow))) > 0){

e.flow += f;

edge[G[x][i] ^ 1].flow -= f;

flow += f;

a -= f;

if(!a)break;

}

}

return flow;

}

int dinic()

{

int flow = 0;

while(bfs()){

memset(cur , 0 , sizeof(cur));

flow += dfs(s , inf);

}

return flow;

}

int in[maxn] , out[maxn];

int main()

{

cin >> n >> m;

init();

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

int u , v , l , r;

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

addedge(u , v , r - l , l , r);

in[v] += l;

in[u] -= l;

}

int sum = 0;

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

if(in[i] > 0){

sum += in[i];

addedge(0 , i , in[i] , 0 , in[i] );

}

else

addedge(i , n+1 , -in[i], 0 , -in[i] );

}

s = 0 , t = n + 1;

if(dinic() == sum){

printf("YES\n");

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

if(i % 2 ==0){

printf("%d\n" , edge[i].flow + edge[i].l);

}

}

}

else{

printf("NO\n");

}

return 0;

}

#### 有源有汇可行流

对于流量有上下界的有源汇网络，原图中存在源点S和汇点T，为了求可行流，先将其转换为无源汇网络。   
    **从T-->S引入一条边，其流量上下界为[0, INF]，此时原图变为了无源汇网络**，然后按照无源汇网络求解可行流的方法来求解。

#### ****有源汇网络的最大流****

要求最大流，先求可行流，通过“有源汇网络的可行流”的求解方法来判断有源汇网络存在可行流。   
    若存在可行流，记从S流出的流量sum1，然后将T-->S的边取消，再次从S到T求解网络的最大流，记从S流出的流量sum2. 那么该有源汇网络的最大流为 sum1 + sum2.   
    其中，sum1是在网络满足流量下界的条件下，从源点S流出的流量；求出sum1之后，网络中可能还有余量可以继续增广，那么再次求解从S到T的最大流，得到sum2，sum1 + sum2即为最终的最大流。

#### ****有源汇网络的最小流****

    求解有源汇网络最小流分为以下几步：   
（1）对SS到TT求一次最大流，即为f1.（在有源汇的情况下，先把整个网络趋向必须边尽量满足的情况）；   
（2）添加一条边T-->S，流量上限为INF，这条边即为P.(构造无源网络）   
（3）对SS到TT再次求最大流，即为f2。（判断此时的网络中存在可行流，同时求出SS-->TT最大流）   
    如果所有必须边都满流，证明存在可行解，原图的最小流为**“流经边P的流量”**（原图已经构造成无源汇网络，对于S同样满足 入流和==出流和，只有新添加的边流向S，而S的出流就是原图的最小流）。

### 最小割模型

#### 输出割集

Dinic后在进行一次bfs,vis[i] = 1代表节点i属于s割，否则属于t割。

#### 最小冲突数

模型：这类问题通常为，初始两个集合，每个人第一意愿为赞成/不赞成，为照顾他人情绪而产生类似“跟票”行为，求最后违反自己意愿人数最少。模型：这类问题通常为，初始两个集合，每个人第一意愿为赞成/不赞成，为照顾他人情绪而产生类似“跟票”行为，求最后违反自己意愿人数最少。

构图方法：   
①： S向每个赞成的连有向边，容量为1；   
②：每个不赞成的向T连有向边，容量为1；   
③：每对朋友之间，若两者意愿不同，则赞成的一方向不赞成的一方连有向边，容量为1；   
即：若有人违背了自己的意愿，割掉。最小割即为答案数。

BZOJ1934

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

#include <iostream>

#include <cstring>

#include <cmath>

#include <cstdlib>

#include <cstdio>

#include <string>

#include <algorithm>

#include <queue>

#include <map>

#include <vector>

#include <stack>

#include <cctype>

#include <set>

#include <ctype.h>

#include <string.h>

#define inf 0x3f3f3f3f

#define eps 1e-10

#define key\_value ch[ch[root][1]][0]

using namespace std;

typedef long long ll;

typedef pair<int,int> P;

const int maxn = 1e3 + 10;

struct Edge

{

int from , to , cap , flow;

// int l , r;//边容量的上下界

Edge(){}

Edge(int u , int v , int c , int f ):from(u) , to(v) , cap(c) , flow(f){}

};

vector<Edge> edge;

vector<int> G[maxn];

bool vis[maxn];

int d[maxn] , cur[maxn];

int n , m , s , t;

void init()

{

edge.clear();

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

G[i].clear();

}

}

void addedge(int u , int v , int c)

{

edge.push\_back(Edge(u , v , c , 0 ));

edge.push\_back(Edge(v , u , 0 , 0 ));

int sz = edge.size();

G[u].push\_back(sz-2);

G[v].push\_back(sz-1);

}

bool bfs()

{

memset(vis , false, sizeof vis);

queue<int> q;

q.push(s);

d[s] = 0;

vis[s] = 1;

while(!q.empty()){

int x = q.front();

q.pop();

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

Edge &e = edge[G[x][i]];

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

vis[e.to] = 1;

d[e.to] = d[x] + 1;

q.push(e.to);

}

}

}

return vis[t];

}

int dfs(int x , int a)

{

if(x == t || !a)return a;

int flow = 0 , f;

for(int &i = cur[x] ; i < G[x].size() ; i ++){

Edge &e = edge[G[x][i]];

if(d[x] + 1 == d[e.to] && (f = dfs(e.to , min(a , e.cap - e.flow))) > 0){

e.flow += f;

edge[G[x][i] ^ 1].flow -= f;

flow += f;

a -= f;

if(!a)break;

}

}

return flow;

}

int dinic()

{

int flow = 0;

while(bfs()){

memset(cur , 0 , sizeof(cur));

flow += dfs(s , inf);

}

return flow;

}

int a[maxn];

int main()

{

cin >> n >> m;

init();

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

int op;

scanf("%d" , &op);

a[i] = op;

if(op == 1){

addedge(0 , i , 1);

}

else{

addedge(i , n + 1 , 1);

}

}

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

int u , v ;

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

if(a[u])

addedge(u , v , 1);

else

addedge(v , u , 1);

}

s = 0 , t = n +1;

int ans = dinic();

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

return 0;

}

#### 最大权闭合子图

* 定义：在一个图中，我们选取一些点构成集合，记为V，且集合中的出边(即集合中的点的向外连出的弧)，所指向的终点(弧头)也在V中，则我们称V为闭合图。最大权闭合图即在所有闭合图中，集合中点的权值之和最大的V，我们称V为最大权闭合图。
* 模型：   
  ①’有向无环图（DAG）问题，反映了事件间的***必要条件***关系：一个事件的发生，它所需要的所有的前提都要发生。EXP：选课，一些课程需要以另一些课程为基础，若给课程打分，最大权闭合图对应获利最大或效率最高的选课计划。   
  ②’带圈的有向图。EXP：工程分期类问题，项目间有***相互依赖***关系，也可以用最大权闭合图解决，将最先应完成的选出来作为第一期。
* 构图方法：   
  ①：S向每个正权点，容量为点权；   
  ②：每个负权点向T，容量为点权绝对值；   
  ③：依赖关系之间单向连接，容量为INF（表示选择了u就一定要选择v，如果割掉了INF，一定不是最小割）；   
  通常正权点代表收益，负权点代表成本。   
  正点权和-最小割=答案。

#### 最小点权覆盖

* 定义：无向(有向)图G中，每个点有点权，选择权值最小的点集覆盖所有的边
* 构图方法：   
  ①：拆点构建二分图。S连x集合中的点，容量为点权，y集合连T，容量为点权  
  ②：原图每条有向边（x,y）,x’到y’’连边，容量为INF；

# 数据结构

## Treap

### Treap实现名次树

//kth查找排名为x的数

//Rank 查询x数的排名(若有多个相同的数，因输出最小的排名)

//Before 和 After 分别查找比x小的最大的数和比x大最小的数

struct node

{

node \*ch[2];

int sz;

int r,v;

node(int vv){

this->v = vv;

ch[1] = ch[0] = NULL;

r = rand();

sz = 1;

}

bool operator < (const node &a) const

{

return r < a.r;

}

int cmp(int x) const

{

if(x == v)return -1;

else return x < v ? 0 : 1;

}

void maintain()

{

sz = 1;

if(ch[0] != NULL)sz += ch[0]->sz;

if(ch[1] != NULL)sz += ch[1]->sz;

}

};

void Rotate(node\* &o , int d)

{

node \*k = o -> ch[d^1];

o -> ch[d^1] = k -> ch[d];

k -> ch[d] = o;

o -> maintain();

k ->maintain();

o = k;

}

void Insert(node\* &o , int x)

{

if(o == NULL)o = new node(x);

else{

int d = (x < o->v ? 0 : 1);

Insert(o->ch[d] , x);

if(o -> ch[d] -> r > o -> r)Rotate(o , d ^ 1);

}

o->maintain();

}

void Remove(node\* &o , int x)

{

int d = o -> cmp(x);

if(d == -1){

node \*u = o;

if(o ->ch[0] != NULL && o -> ch[1] != NULL){

int d2 = (o -> ch[0] -> r > o -> ch[1]->r ? 1: 0);

Rotate(o , d2);

Remove(o -> ch[d2] , x);

}

else{

if(o -> ch[0] == NULL){

o = o ->ch[1];

}

else{

o = o -> ch[0];

}

delete u;

}

}

else{

Remove(o -> ch[d] , x);

}

if(o != NULL)o->maintain();

}

int kth(node \*o , int k)

{

if(o == NULL || k <= 0 || k > o-> sz)return -1;

int s = (o ->ch[0] == NULL ? 0 : o->ch[0]->sz);

if(k == s + 1)return o->v;

if(k <= s){

return kth(o->ch[0] , k);

}

else{

return kth(o->ch[1] , k-s-1);

}

}

int Rank(node \*o , int x)

{

if(o == NULL)return 0;

if(x <= o->v){

return Rank(o->ch[0] , x);

}

int s = (o->ch[0] == NULL ? 0 : o->ch[0]->sz);

return s + 1 + Rank(o->ch[1] , x);

}

int Before(node \*o , int x)

{

int res = -1;

while(o != NULL){

if(x <= o->v){

o = o->ch[0];

}

else{

res = o->v;

o = o->ch[1];

}

}

return res;

}

int After(node \*o , int x)

{

int res = -1;

while(o != NULL){

if(x >= o->v){

o = o->ch[1];

}

else{

res = o->v;

o = o->ch[0];

}

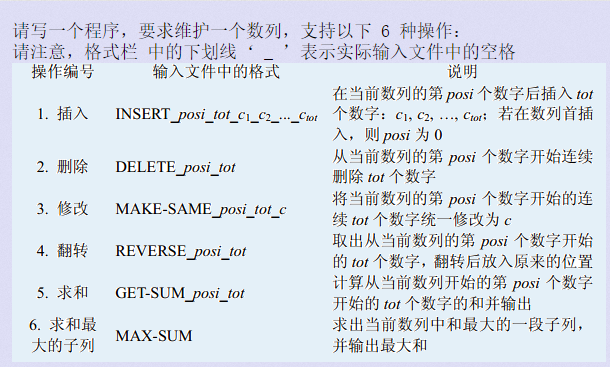
}

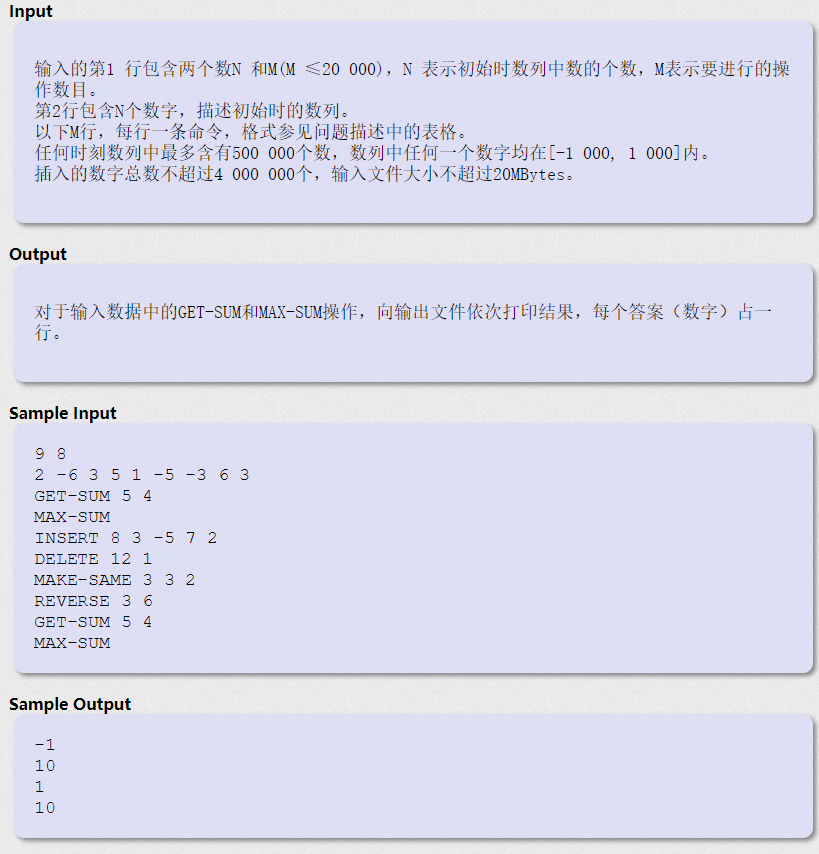
return res;

}

## Splay(伸展树)

Bzoj1500:





#define inf 0x3f3f3f3f

#define key\_value ch[ch[root][1]][0]

const int maxn = 5e5+10;

int pre[maxn] , ch[maxn][2] , key[maxn] , size[maxn];

int root , tot1;

int sum[maxn] , rev[maxn] , same[maxn];

int lx[maxn] , rx[maxn] , mx[maxn];

int s[maxn] , tot2;//内存池和容量

int a[maxn];

int n,q;

void newnode(int &r , int father , int k)

{

if(tot2) r = s[tot2 --];

else r = ++tot1;

pre[r] = father;

ch[r][0] = ch[r][1] = 0;

key[r] = k;

sum[r] = k;

rev[r] = same[r] = 0;

lx[r] = rx[r] = mx[r] = k;

size[r] = 1;

}

void update\_rev(int r)

{

if(!r)return ;

swap(ch[r][0] , ch[r][1]);

swap(lx[r] , rx[r]);

rev[r] ^= 1;

}

void update\_same(int r , int v)

{

if(!r)return;

key[r] = v;

sum[r] = v\*size[r];

lx[r] = rx[r] = mx[r] = max(v , v \* size[r]);

same[r] = 1;

}

void push\_up(int r)

{

int lson = ch[r][0] , rson = ch[r][1];

size[r] = size[lson] + size[rson] + 1;

sum[r]= sum[lson] + sum[rson] + key[r];

lx[r] = max(lx[lson] , sum[lson] + key[r] + max(0 , lx[rson]));

rx[r] = max(rx[rson] , sum[rson] + key[r] + max(0 , rx[lson]));

mx[r] = max(0 , rx[lson]) + key[r] + max(0 , lx[rson]);

mx[r] = max(mx[r] , max(mx[lson] , mx[rson]));

}

void push\_down(int r)

{

if(same[r]){

update\_same(ch[r][0] , key[r]);

update\_same(ch[r][1] , key[r]);

same[r] = 0;

}

if(rev[r]){

update\_rev(ch[r][0]);

update\_rev(ch[r][1]);

rev[r] = 0;

}

}

void build(int &x , int l , int r,int father)

{

if(l > r)return;

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

newnode(x , father , a[mid]);

build(ch[x][0] , l , mid - 1 , x);

build(ch[x][1] , mid + 1 , r , x);

push\_up(x);

}

void init()

{

root = tot1 = tot2 = 0;

ch[root][0] = ch[root][1] = size[root] = pre[root] = 0;

same[root] = rev[root] = sum[root] = key[root] = 0;

lx[root] = rx[root] = mx[root] = -inf;

newnode(root , 0 , -1);

newnode(ch[root][1] , root , -1);

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

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

}

build(key\_value , 0 , n-1 , ch[root][1]);

push\_up(ch[root][1]);

push\_up(root);

}

void Rotate(int x,int kind)//kind->0 左旋 1->右旋

{

int y = pre[x];

push\_down(y);

push\_down(x);

ch[y][!kind] = ch[x][kind];

pre[ch[x][kind]] = y;

if(pre[y]){

ch[pre[y]][ch[pre[y]][1] == y] = x;

}

pre[x] = pre[y];

ch[x][kind] = y;

pre[y] = x;

push\_up(y);

}

void splay(int r , int goal)

{

push\_down(r);

while(pre[r] != goal){

if(pre[pre[r]] == goal){

push\_down(pre[r]);

push\_down(r);

Rotate(r , ch[pre[r]][0] == r);

}

else{

push\_down(pre[pre[r]]);

push\_down(pre[r]);

push\_down(r);

int y = pre[r];

int kind = ch[pre[y]][0] == y;

if(ch[y][kind] == r){

Rotate(r , !kind);

Rotate(r , kind);

}

else{

Rotate(y , kind);

Rotate(r , kind);

}

}

}

push\_up(r);

if(goal == 0)root = r;

}

int get\_kth(int r , int k)

{

push\_down(r);

int t = size[ch[r][0]] + 1;

if(t == k)return r;

if(t > k)return get\_kth(ch[r][0] , k);

else return get\_kth(ch[r][1] , k-t);

}

//在第pos个数后面插入tot个数

void Insert(int pos , int tot)

{

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

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

}

splay(get\_kth(root , pos + 1) , 0);

splay(get\_kth(root , pos + 2) , root);

build(key\_value , 0 , tot-1 , ch[root][1]);

push\_up(ch[root][1]);

push\_up(root);

}

//删除子树

void Erase(int r)

{

if(!r)return;

s[++tot2] = r;

Erase(ch[r][0]);

Erase(ch[r][1]);

}

//在第pos个数开始连续删除tot个数

void Delete(int pos , int tot)

{

splay(get\_kth(root , pos) , 0);

splay(get\_kth(root , pos + tot + 1) , root);

Erase(key\_value);

pre[key\_value] = 0;

key\_value = 0;

push\_up(ch[root][1]);

push\_up(root);

}

//在第pos个数开始连续tot个数修改为c

void make\_same(int pos , int tot , int c)

{

splay(get\_kth(root , pos) , 0);

splay(get\_kth(root , pos + tot + 1) , root);

update\_same(key\_value , c);

push\_up(ch[root][1]);

push\_up(root);

}

//在第pos个数开始连续tot个数反转

void Reverse(int pos , int tot)

{

splay(get\_kth(root , pos) , 0);

splay(get\_kth(root , pos + tot + 1) , root);

update\_rev(key\_value);

push\_up(ch[root][1]);

push\_up(root);

}

//在第pos个数开始连续tot个数的和

int get\_sum(int pos , int tot)

{

splay(get\_kth(root , pos) , 0);

splay(get\_kth(root , pos+tot+1) , root);

return sum[key\_value];

}

//在第pos个数开始连续tot个数的最大字段和

int get\_maxsum(int pos , int tot)

{

splay(get\_kth(root , pos) , 0);

splay(get\_kth(root , pos+tot+1) , root);

return mx[key\_value];

}

void Inorder(int r)

{

if(!r)return ;

push\_down(r);

Inorder(ch[r][0]);

printf("%d " , key[r]);

Inorder(ch[r][1]);

}

int main()

{

while(cin >> n >> q){

init();

while(q --){

char op[20];

int x,y,z;

scanf("%s" , op);

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

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

Insert(x , y);

}

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

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

Delete(x,y);

}

else if(op[0] == 'M'){

if(op[2] == 'K'){

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

make\_same(x,y,z);

}

else{

printf("%d\n" , get\_maxsum(1 , size[root] - 2));

}

}

else if(op[0] == 'R'){

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

Reverse(x,y);

}

else{

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

printf("%d\n" , get\_sum(x,y));

}

}

}

return 0;

}

## Trie

### 可持久化字典树

struct node

{

int ch[maxn \* 32][2];

int sum[maxn\*32];

int cnt;

void init()

{

cnt = 0;

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

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

}

int Insert(int x,int val)

{

int temp,y;

temp = y = ++cnt;

for(int i = 30 ; i >= 0 ; i --){

ch[y][0] = ch[x][0];

ch[y][1] = ch[x][1];

sum[y] = sum[x] + 1;

int t = (val & (1 << i));

t >>= i;

x = ch[x][t];

ch[y][t] = ++cnt;

y = ch[y][t];

}

sum[y] = sum[x] + 1;

return temp;

}

int Query(int l , int r , int val)

{

int res = 0;

for(int i = 30 ; i >= 0 ; i --){

int t = ( val & (1 << i) );

t >>= i;

if(sum[ch[r][t^1]] - sum[ch[l][t^1]]){

l = ch[l][t^1];

r = ch[r][t^1];

res += (1 << i);

}

else{

l = ch[l][t];

r = ch[r][t];

}

}

return res;

}

};

node trie;

//main()调用 [L,R]区间中的字典树信息—root[R] – root[L-1];

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

root[i] = trie.Insert(root[i-1] , val[i]);

}

## 树状数组

### 二维树状数组

关于一维树状数组的区间修改+区间查询：

引入delta数组 delta[i]表示区间 [i, n] 的共同增量 于是修改区间 [l, r] 时修改 delta[l] 和 delta[r + 1] 即可

查询的时候是查询区间 [l, r] 的和 即sum[r] - sum[l - 1]

sum[i] = a[1]+...+a[i] + delta[1]\*i + delta[2]\*(i - 1) + delta[3]\*(i - 2)+...+delta[i]\*1 // a[i]为原始数组

= sigma( a[x] ) + sigma( delta[x] \* (i + 1 - x) )

= sigma( a[x] ) + (i + 1) \* sigma( delta[x] ) - sigma( delta[x] \* x )

于是只需要维护 delta[x] 与 delta[x] \* x 的前缀和（即两个树状数组）

int n,q;

LL sum[maxn];

LL delta[maxn];

LL delta2[maxn];

LL lowbit(LL x){

return x & (-x);

}

void add(LL x, LL c, LL g[]){

while(x <= n){

g[x] += c;

x += lowbit(x);

}

}

LL query(LL x, LL g[]){

LL ans = 0;

while(x > 0){

ans += g[x];

x -= lowbit(x);

}

return ans;

}

int main(){

while(scanf("%d%d",&n,&q) == 2){

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

memset(delta, 0, sizeof(delta));

memset(delta2, 0, sizeof(delta2));

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

LL c;

scanf("%lld",&c);

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

}

while(q--){

getchar();

char op;

scanf("%c",&op);

if(op == 'Q'){

LL l, r;

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

LL sum1 = sum[l-1] + l\*query(l-1,delta) - query(l-1,delta2);

LL sum2 = sum[r] + (r+1)\*query(r,delta) - query(r,delta2);

printf("%lld\n",sum2-sum1);

}

else{

LL l, r ,k;

scanf("%lld%lld%lld",&l,&r,&k);

add(l,k,delta);

add(r+1,-k,delta);

add(l,k\*l,delta2);

add(r+1,-k\*(r+1),delta2);

}

}

}

return 0;

}

## 线段树

### 区间更新&&查询

void push\_down(int l , int r , int o)

{

if(lazy[o]){

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

sum[o\*2] += (mid - l + 1)\*lazy[o];

sum[o\*2+1] += (r - mid)\*lazy[o];

lazy[o\*2] += lazy[o];

lazy[o\*2+1] += lazy[o];

lazy[o] = 0;

}

return ;

}

void build(int o , int l , int r)

{

lazy[o] = 0;

if(l == r){

sum[o] = a[l];

return;

}

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

build(o\*2 , l , mid);

build(o\*2+1 , mid + 1 , r);

sum[o] = sum[o\*2] + sum[o\*2+1];

}

ll query(int o , int l , int r , int ql , int qr)

{

if(ql <= l && r <= qr){

return sum[o];

}

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

push\_down(l , r, o);

ll ans1 = 0;

ll ans2 = 0;

if(ql <= mid)ans1 = query(o\*2 , l , mid , ql , qr);

if(qr > mid)ans2 = query(o\*2 + 1 , mid +1 , r , ql , qr);

return ans1 + ans2;

}

void update(int l , int r , int o , int ql , int qr , ll c)

{

if(ql <= l && r <= qr){

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

lazy[o] += c;

return;

}

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

push\_down(l , r, o);

if(ql <= mid){

update(l , mid , o\*2 , ql , qr , c);

}

if(qr > mid){

update(mid + 1 , r , o\*2+1 , ql , qr , c);

}

sum[o] = sum[o\*2] + sum[o\*2+1];

return ;

}

//主函数调用（下标从1开始）

printf("%lld\n",query(1 , 1 , n , l , r));//[l,r]为查询区间

update(1 , n , 1 , l , r , c);//[l,r]区间所有数+c

### 扫描线

#include <iostream>

#include <cstring>

#include <string>

#include <algorithm>

#include <cstdio>

#include <cmath>

#include <map>

#include <set>

#include <queue>

#include <vector>

#define mod 1000000007

#define INF 0x3f3f3f3f

#define fuck() (cout << "----------------------------------------" << endl)

using namespace std;

const int maxn = 200000 + 5;

double X[maxn];

double sum[maxn];//记录某个区间的下底边长度

int mark[maxn];//记录某个区间的下底边个数

struct node

{

double l,r,h;

int d;

node(){};

node(double x1, double x2, double H, int c)

{

l = x1;

r = x2;

h = H;

d = c;

}

bool operator < (const node &p) const

{

return h < p.h;

}

}nodes[maxn];

void upfather(int n, int left, int right)

{

if(mark[n]) sum[n] = X[right+1] - X[left];//d大于1表示要计算这一段

else if(left == right) sum[n] = 0;

else sum[n] = sum[2\*n] + sum[2\*n+1];//更新父亲包含的线段长度

}

void update(int l, int r, int d, int n, int left, int right)

{

if(l <= left && right <= r)//将区间内的d值更新

{

mark[n] += d;

upfather(n,left,right);//更新底边长

return;

}

int mid = (left + right) / 2;

if(l <= mid) update(l,r,d, 2\*n, left, mid);

if(r > mid) update(l,r,d, 2\*n+1, mid+1, right);

upfather(n,left,right);

}

int search(double key, double \*x, int n)

{

int l = 0, r = n - 1;

while(l <= r)

{

int mid = (l + r) / 2;

if(x[mid] == key) return mid;

else if(x[mid] > key) r = mid - 1;

else l = mid +1 ;

}

return -1;

}

int main()

{

int n, kases = 1;

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

{

int k = 0;

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

{

double x1,x2,y1,y2;

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

X[k] = x1;

nodes[k++] = node(x1,x2,y1,1);//存上边

X[k] = x2;

nodes[k++] = node(x1,x2,y2,-1);//存下边

}

sort(nodes,nodes+k);

sort(X,X+k);

int cnt = 1;

for(int i=1; i<k; i++)//去掉重复点

if(X[i] != X[i-1]) X[cnt++] = X[i];

double ans = 0;

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

{

int L = search(nodes[i].l,X,cnt);

int R = search(nodes[i].r,X,cnt) - 1;

update(L,R,nodes[i].d,1,0,cnt-1);

ans += sum[1] \* (nodes[i+1].h - nodes[i].h);

}

printf("Test case #%d\nTotal explored area: %.2lf\n\n",kases++,ans);

}

return 0;

}

## 主席树

struct node

{

int tot;

int T[maxn] , lson[M] , rson[M] , c[M], t[maxn];

int build(int l , int r)

{

int root = tot ++;

c[root] = 0;

if(l != r){

int mid = l + r >> 1;

lson[root] = build(l , mid);

rson[root] = build(mid + 1 , r);

}

return root;

}

int update(int root , int pos , int val)

{

int newroot = tot ++ , tmp = newroot;

c[newroot] = c[root] + val;

int l = 1 , r = n;

while(l < r)

{

int mid = l + r >>1;

if(pos <= mid){

lson[newroot] = tot ++ ; rson[newroot] = rson[root];

newroot = lson[newroot] ; root = lson[root];

r = mid;

}

else{

rson[newroot] = tot ++ ; lson[newroot] = lson[root];

newroot = rson[newroot] ; root = rson[root];

l = mid + 1;

}

c[newroot] = c[root] + val;

}

return tmp;

}

//查询区间不同数的个数

int query(int root , int pos)

{

int ret = 0;

int l = 1 , r = n;

while(pos < r){

int mid = l + r >>1;

if(pos <= mid){

r = mid;

root = lson[root];

}

else{

ret += c[lson[root]];

root = rson[root];

l = mid + 1;

}

}

return ret + c[root];

}

//查询区间不同数初始化

void init()

{

tot = 0;

T[n+1] = build(1 , n);

map<int,int> mp;

mp.clear();

for(int i = n ; i >= 1 ; i --){

if(mp.find(a[i]) == mp.end()){

T[i] = update(T[i+1] , i , 1);

}

else{

int tmp = update(T[i+1] , mp[a[i]] , -1);

T[i] = update(tmp , i , 1);

}

mp[a[i]] = i;

}

}

void init\_hash()

{

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

t[i] = a[i];

}

sort(t + 1 , t + 1 + n );

m = unique(t + 1 , t + 1 + n ) - t - 1;

}

int Hash(int x)

{

return lower\_bound(t + 1 , t + 1 + m) - t ;

}

//查询区间第k小初始化

void init\_kth()

{

init\_hash();

T[n+1] = build(1 , m);

for(int i = n ; i ; i --){

int pos = Hash(a[i]);

T[i] = update(T[i+1] , pos , 1);

}

}

//查询区间不同数的个数

int ask(int l , int r)

{

return query(T[l] , r);

}

int querykth(int left\_root , int right\_root , int k)

{

int l = 1 , r = n;

while(l < r)

{

int mid = l + r >> 1;

if(c[lson[left\_root]] - c[lson[right\_root]] >= k){

r = mid;

left\_root = lson[left\_root];

right\_root = lson[right\_root];

}

else{

l = mid + 1;

k -= c[lson[left\_root]] - c[lson[right\_root]];

left\_root = rson[left\_root];

right\_root = rson[right\_root];

}

}

return l;

}

//查询第k小

int ask\_kth(int l , int r , k)

{

return querykth(T[l] , T[r+1] , k);

}

};

node tree;

## 线性基：

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

for(int j=62; j>=0; j--){

if(a[i] >> j & 1){

if(b[j]) a[i] ^= b[j];

else{

b[j] = a[i];

for(int k=j-1; k>=0; k--)

if(b[k] && (b[j] >> k & 1)) b[j] ^= b[k];

for(int k=j+1; k<=62; k++)

if(b[k] >> j & 1) b[k] ^= b[j];

break;

}

}

}

}

最大异或值

LL ans = 0;

for(int i=62; i>=0; i--){

if(ans < (ans^b[i])) ans ^= b[i];

}

//第k小线性基

bool zero;

int T,n,m,tot;

ll bin[65],a[10005];

void gauss()

{

tot=zero=0;

for(ll i=bin[60];i;i>>=1)

{

int j=tot+1;

while(!(i&a[j])&&j<=n)j++;

if(j==n+1)continue;

tot++;

swap(a[tot],a[j]);

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

if(k!=tot&&(a[k]&i))

a[k]^=a[tot];

}

if(tot!=n)zero=1;

}

ll query(ll x)

{

ll ans=0;

x-=zero;

if(!x)return 0;

if(x>=bin[tot])return -1;

    for(int i=1;i<=tot;i++)

if(x&bin[tot-i])ans^=a[i];

return ans;

}

# 计算几何

## 基础

struct Point

{

double x,y;

Point(){}

Point(double xx,double yy):x(xx),y(yy){}

Point operator + (const Point &a) const { return Point(x + a.x , y + a.y); }

Point operator - (const Point &a) const { return Point(x - a.x , y - a.y); }

//Point operator \* (const Point &a) const { return x \* a.x + y \* a.y; }

// 向量叉积

double operator ^ (const Point &a) const { return x \* a.y - y \* a.x; }

bool operator == (const Point &p) const { return sign(x-p.x) == 0 && sign(y-p.y) == 0; }

};

Point operator / (Point a, double b){ return Point(a.x/b ,a.y/b); }

typedef Point Vector;

//向量\*数

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

//两点距离

double dist(Point a , Point b){

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

}

// 向量点积

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

// 向量长度

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

// 向量AB夹角

double angle(Vector A , Vector B){ return acos(Dot(A,B) / Length(A) / Length(B) ); }

// 向量叉积

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

// 向量AB和AC组成平行四边形的面积

double Area2(Point A , Point B , Point C){

return Cross(B-A , C-A);

}

// 向量A逆时针旋转rad(弧度)

Vector Rotate(Vector A , double rad)

{

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

A.x \* sin(rad) + A.y \* cos(rad));

}

//两点式表示

struct Line

{

Point s,e;

Line(){}

Line(Point ss, Point ee):s(ss),e(ee){}

/\*

Get two lines' intersection

if the first integer equals to 0 ----> two lines are coincide

if the first integer equals to 1 ----> two lines are parallel

if the first integer queals to 2 ----> get the Point

\*/

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

Point res = s;

if(sign((s-e)^(b.s-b.e)) == 0){

if(sign((s-b.e)^(b.s-b.e)) == 0){

return make\_pair(0 , res);

}

else

return make\_pair(1 , res);

}

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);

}

};

//参数式表示

struct line{

Point p;//线上一点

Vector v;//方向向量

double ang; //极角，从x正半轴旋转到v所需要的角（弧度）

line(Point p, Vector v):p(p),v(v){ang = atan2(v.y,v.x);}

Point point(double t){return p+v\*t;};

bool operator < (const line& L) const{ //排序用的比较运算符

return ang < L.ang;

}

};

//与圆和球有关计算

struct Circle

{

Point c;

double r;

Circle(Point c , double r):c(c),r(r){}

//通过圆心角求圆上任意一点

Point point(double a){

return Point (c.x + cos(a)\*r , c.y+sin(a)\*r);

}

};

### 点和线段(直线)相关

//点到直线距离

double DistanceToLine(Point p , Point A , Point B)

{

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

return fabs(Cross(v1 , v2)) / Length(v1);//如果不取绝对值，得到的是有向距离

}

//点到线段距离

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(sign(Dot(v1,v2)) < 0)return Length(v2);

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

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

}

//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) );

}

//判断线段相交 包含端点

bool segement\_inter\_1(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) &&

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

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

}

//线段相交 不包含端点

bool segement\_inter\_2(Line l1 , Line l2)

{

Point a1 = l1.s , a2 = l1.e , b1 = l2.s , b2 = l2.e;

double c1 = Cross(a2-a1,b1-a1) , c2 = Cross(a2 - a1 , b2 - a1) ,

c3 = Cross(b2 - b1 , a1 - b1) , c4 = Cross(b2 - b1 , a2 - a1);

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

}

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

}

//判断点在线段上 不包含端点

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

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

}struct Point{

int x,y;

Point(){}

Point(int xx, int yy):x(xx),y(yy){}

}pt[maxn];

double cross(Point a, Point b, Point c){

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

}

int dcmp(double x){

if(fabs(x) < eps)

return 0;

if(x < 0) return -1;

else return 1;

}

bool onSegment(Point a, Point b, Point q){

if(dcmp(cross(q,a,b)) == 0){

if(min(a.x, b.x) <= q.x && q.x <= max(a.x, b.x)

&& min(a.y, b.y) <= q.y && q.y <= max(a.y, b.y))

return true;

return false;

}

return false;

}

### 多边形相关(凸包)

//多边形的有向面积 [0,n-1] 凹凸无关

double PolygonArea(Point \*p , int n)

{

double area = 0;

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

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

}

return area/2;

}

//\*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(sign((p[i]-a)^(p[(i+1)%n]-a)) < 0)return -1;

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

}

return 1;

}

//\*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(OnSegment(p,side.s , side.e))return 0;

//如果平行轴则不考虑

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

continue;

if(OnSegment(side.s,ray.s , ray.e))

{

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

}

else if(OnSegment(side.e,ray.s , ray.e))

{

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

}

else if(segement\_inter\_1(ray,side))

cnt++;

}

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

else return -1;

}

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

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

}

return true;

}

/\*

\* 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

\*/

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(sign(tmp) > 0)return true;

else if(sign(tmp) == 0 && sign(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 &&

sign((List[Stack[top-1]]-List[Stack[top-2]])^(List[i]-List[Stack[top-2]])) <=

0)top--;

Stack[top++] = i;

}

}

### 圆和球相关

//直线和圆的交点 返回值为交点个数 交点存在sol中 attention：sol没有清空

int getLineCircleIntersection(line L , Circle C , double &t1 , double &t2 , vector<Point> &sol){

double a=L.v.x, b=L.p.x-C.c.x, c=L.v.y, d=L.p.y-C.c.y;

double e=a\*a+c\*c, f=2\*(a\*b+c\*d), g=b\*b+d\*d-C.r\*C.r;

double delta = f\*f - 4\*e\*g;//判别式

if(sign(delta)<0) return 0; //相离

if(sign(delta)==0){

t1=t2=-f/(2\*e);

sol.push\_back(L.point(t1));

return 1;

}

//相交

t1 = (-f-sqrt(delta))/(2\*e);

sol.push\_back(L.point(t1));

t2 = (-f+sqrt(delta))/(2\*e);

sol.push\_back(L.point(t2));

return 2;

}

//计算向量极角

double angle(Vector v){return atan2(v.y,v.x);}

//两圆相交

int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol){

double d=Length(C1.c-C2.c);

if(sign(d)==0){

if(sign(C1.r-C2.r)==0) return -1; //两圆重合

return 0;

}

if(sign(C1.r+C2.r-d)<0) return 0; //内含

if(sign(fabs(C1.r-C2.r)-d)>0) return 0; //外离

double a = angle(C2.c-C1.c); //向量C1C2的极角

double da = acos((C1.r\*C1.r+d\*d-C2.r\*C2.r)/(2\*C1.r\*d));

//C1C2到C1P1的角

Point p1 = C1.point(a-da), p2 = C1.point(a+da);

sol.push\_back(p1);

if(p1==p2)return 1;

sol.push\_back(p2);

return 2;

}

//过定点作圆切线,v[i]是第i条切线，返回切线条数

int getTangents(Point p, Circle C, Vector\* v){

Vector u = C.c-p;

double dist = Length(u);

if(dist<C.r) return 0;

else if(sign(dist-C.r)==0){ //p在圆上，只有一条切线

v[0] = Rotate(u,PI/2);

return 1;

}

else{

double ang = asin(C.r/dist);

v[0] = Rotate(u, -ang);

v[1] = Rotate(u, +ang);

return 2;

}

}

/\*两圆的公切线

\* 返回切线条数 -1表示两圆重合 无数条公切线

\* a[i]和b[i]表示第i条切线在圆A和圆B上的切点

\*/

int getTangents(Circle A, Circle B, Point\* a, Point\* b){

int cnt=0;

if(sign(A.r-B.r) < 0){

swap(A,B);

swap(a,b);

}

double d=sqrt((A.c.x-B.c.x)\*(A.c.x-B.c.x)+(A.c.y-B.c.y)\*(A.c.y-B.c.y));

double rdiff=A.r-B.r;

double rsum=A.r+B.r;

if(sign(d-rdiff) < 0) return 0; //内含

double base = atan2(B.c.y-A.c.y,B.c.x-A.c.x);

if(sign(d) == 0) return -1; //无限多条切线

if(sign(d -rdiff) == 0){//内切，一条切线

a[cnt]=b[cnt] = A.point(base);

cnt++;

return 1;

}

//有外共切线

double ang = acos((A.r-B.r)/d);

a[cnt] = A.point(base+ang);

b[cnt] = B.point(base+ang);

cnt++;

a[cnt] = A.point(base-ang);

b[cnt] = B.point(base-ang);

cnt++;

if(sign(d - rsum) == 0){

a[cnt]=b[cnt]=A.point(base);

cnt++;

}

else if(sign(d - rsum) > 0){

double ang=acos((A.r+B.r)/d);

a[cnt]=A.point(base+ang);

b[cnt]=B.point(PI+base+ang);

cnt++;

a[cnt]=A.point(base-ang);

b[cnt]=B.point(PI+base-ang);

cnt++;

}

return cnt;

}

//三角形外接圆（三点保证不共线）

Circle CircumscribedCircle(Point p1, Point p2, Point p3){

double Bx = p2.x-p1.x, By = p2.y-p1.y;

double Cx = p3.x-p1.x, Cy = p3.y-p1.y;

double D = 2\*(Bx\*Cy-By\*Cx);

double cx = (Cy\*(Bx\*Bx+By\*By)-By\*(Cx\*Cx+Cy\*Cy))/D+p1.x;

double cy = (Bx\*(Cx\*Cx+Cy\*Cy)-Cx\*(Bx\*Bx+By\*By))/D+p1.y;

Point p = Point(cx,cy);

return Circle(p,Length(p1-p));

}

//三角形内切圆

Circle InscribedCircle(Point p1, Point p2, Point p3){

double a = Length(p2-p3);

double b = Length(p3-p1);

double c = Length(p1-p2);

Point p = (p1\*a+p2\*b+p3\*c)/(a+b+c);

return Circle(p, DistanceToLine(p, p1, p2));

}

### 极角排序

方法一：利用arctan计算计较大小 （范围（-180， 180））

bool cmp(Point p, Point q){

return atan2(p.y, p.x) < atan2(q.y, q.x);

}

方法二： 象限极角排序

bool xy(double x, double y) {return x > y + eps;} //x > y;

bool yx(double x, double y) {return x < y - eps;}//x < y;

bool xyd(double x, double y) {return x > y - eps;} // x >= y;

bool yxd(double x, double y) {return x < y + eps;} // x <= y;

bool dd(double x, double y) {return fabs(x-y) < eps;} // x == y;

int quadrant(Point a){

if(xy(a.x,0) && xyd(a.y,0)) return 1;

if(yxd(a.x,0) && xy(a.y,0)) return 2;

if(yx(a.x,0) && yxd(a.y,0)) return 3;

if(xyd(a.x,0) && yx(a.y,0)) return 4;

}

int cross(Point a, Point b, Point c){//大于0 表示ba向量在ca左侧

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

}

bool cmp(Point a, Point b){

int l1 = quadrant(a);

int l2 = quadrant(b);

if(l1 == l2){//这里也可以用其他方法 比如atan2

Point origin(0,0);

return cross(origin,a,b) > 0;//按逆时针排序

}

return l1 < l2;

} （没仔细测过。。）

## 半平面交

### 裸半平面交求面积：

struct Point{

double x,y;

Point(){}

Point(double xx, double yy):x(xx),y(yy){}

}pt[1005],a[1005];

struct Line{

Point s,e;

double slop;

Line(){}

Line(Point ss, Point ee, double slop\_):s(ss),e(ee),slop(slop\_){}

}l[1005],q[1005];

int cnt, tot;

Point operator-(Point a, Point b){

Point t;

t.x = a.x-b.x;

t.y = a.y-b.y;

return t;

}

double operator \* (Point a, Point b){

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

}

bool operator < (Line a, Line b)

{

if(a.slop != b.slop)

return a.slop < b.slop;

return (a.e - a.s) \* (b.e - a.s) > 0;

}

Point inter(Line a, Line b)

{

double k1,k2,t;

k1 = (b.e - a.s)\*(a.e - a.s);

k2 = (a.e - a.s)\*(b.s - a.s);

t = k1 / (k1+k2);

Point ans;

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

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

return ans;

}

bool judge(Line a, Line b, Line t){

Point p = inter(a,b);

return (t.e - t.s) \* (p - t.s)<0;

}

void hpi(){

sort(l+1,l+cnt+1);

int L = 1, R = 0;

tot = 0;

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

{

if(l[i].slop != l[i-1].slop)

tot++;

l[tot] = l[i];

}

cnt = tot;

tot = 0;

q[++R] = l[1];

q[++R] = l[2];

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

{

while(L<R && judge(q[R-1], q[R], l[i])) R--;

while(L<R && judge(q[L+1], q[L], l[i])) L++;

q[++R] = l[i];

}

while(L<R && judge(q[R-1], q[R], q[L])) R--;

while(L<R && judge(q[L+1], q[L], q[R])) L++;

q[R+1] = q[L];

for(int i=L; i<=R; i++)

a[++tot] = inter(q[i], q[i+1]);

}

double getans(){

double ans = 0;

if(tot < 3) return 0;

a[++tot] = a[1];

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

ans += a[i]\*a[i+1];

ans = fabs(ans)/2;

return ans;

}

int main(){

int T;

scanf("%d",&T);

while(T--){

int n;

scanf("%d",&n);

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

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

}

pt[n+1] = pt[1];

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

l[++cnt].s = pt[i];

l[cnt].e = pt[i+1];

}

}

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

l[i].slop = atan2(l[i].e.y - l[i].s.y, l[i].e.x - l[i].s.x);

}

hpi();

printf("%.3lf\n",getans());

return 0;

}

# 字符串

## 字符串Hash

方法1：   
unsigned long long hash[N];   
定义一个unsigned long long类型的变量，它的范围是在[0, 2^64) 内，这就相当于，当数超不过2^64-1后，它会溢出！这就相当于一个数模2^64的过程。   
那么hash函数可以理解为：   
hash[i]=(hash[i−1] × p + s[i]) % (2^64)

hash[i]=(hash[i−1] × p + s[i]) % (2^64)   
求区间: hash[L,R]=hash[R]−hash[L−1]∗p^(R−L+1)

hash[L,R]=hash[R]−hash[L−1]∗p^(R−L+1)   
P取一个大素数，一般习惯取1e9+7或1e9+9   
安全指数：三星（所以并不是很安全）   
也就是此份模板（这题也就是用这个）：   
大致满足字符串>1e4/1e5，因为相对于方法2，这个时间小。

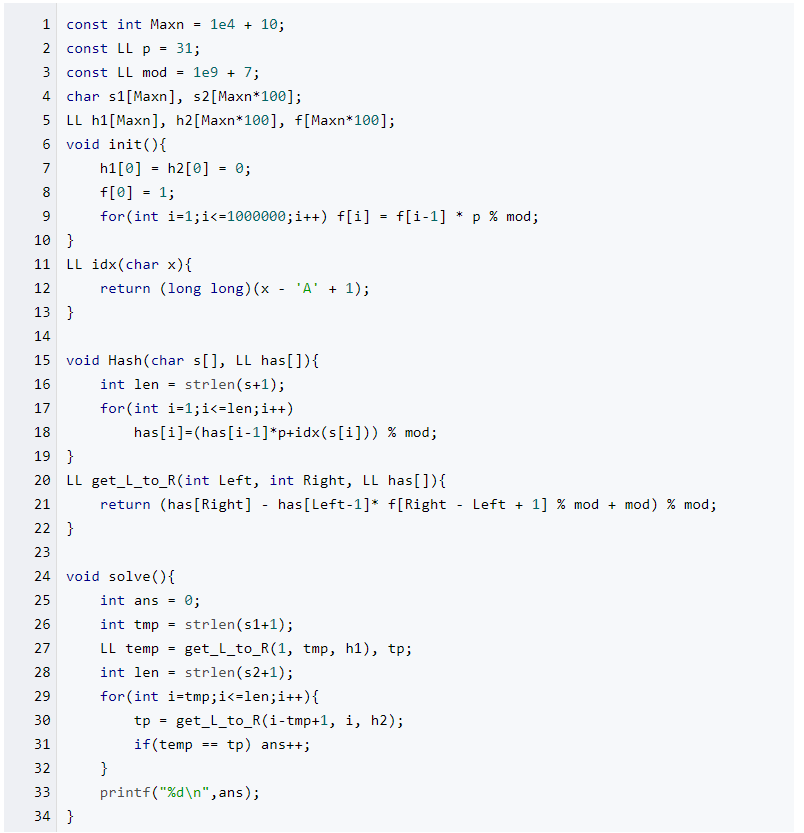


方法2：   
p取一个6到8位的素数，mod取一个大素数，一般取1e9+7或1e9+9   
安全指数：四星 （还可以）   
但是字符串长度>=1e5，时间上就虚了。。   
hash[i] = (hash[i−1] × p + idx(s[i])) % mod

hash[i] = (hash[i−1] × p + idx(s[i])) % mod   
求区间：   
hash[L,R] = hash[R] − hash[L−1] ∗ p(R−L+1)

hash[L,R] = hash[R] − hash[L−1] ∗ p(R−L+1)

但是字符串长度>=1e5，取膜太多，时间上就虚了。。



方法3：   
我感觉。。太暴力了。。   
hash取法三   
double hash   
即取两个mod值，mod1和mod2   
hash1[i] = (hash1[i-1] \* p + idx(s[i])) % mod1   
hash2[i] = (hash2[i-1] \* p + idx(s[i])) % mod2   
mod1一般取1e9+7，mod2一般取1e9+9为什么这么取？   
1000000007和1000000009是一对孪生素数，取它们，冲突的概率极低！   
安全指数：五星！（非常稳！）(稳啊…稳T了…开玩笑)   
但是确实稳，反正不会wa.

## 后缀数组

### 白书SA + LCP 比较的是（i,i+1）

int rank[200005],tmp[200005];

bool cmp(int i,int j){

int ri,rj;

if(rank[i]!=rank[j])

return rank[i] < rank[j];

ri=i+k<=n?rank[i+k]:-1;

rj=j+k<=n?rank[j+k]:-1;

return ri < rj;

}

void get\_suffix(int \*s,int \*sa){

int i;

// n=s.length();

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

sa[i]=i;

rank[i]=i < n?s[i]:-1;

}

for(k=1;k<=n;k\*=2){

sort(sa,sa+n,cmp);

tmp[sa[0]]=0;

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

tmp[sa[i]]=tmp[sa[i-1]]+(cmp(sa[i-1],sa[i])?1:0);

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

rank[i]=tmp[i];

}

}

void get\_lcp(string s, int \*sa, int \*lcp){

int n = s.length();

for(int i=0; i<=n; i++) rank[sa[i]] = i;

int h = 0;

lcp[0] = 0;

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

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

if(h>0) h--;

for(;j+h<n && i+h<n; h++)

if(s[j+h] != s[i+h]) break;

lcp[rank[i]-1] = h;

}

}

### 蓝书SA+LCP(i-1,i) m大了需要离散

void build\_sa(int n, int m, int \*s){

int \*x = t, \*y = t2;

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

for(int i=0; i<n; i++) c[x[i] = s[i]]++;

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

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

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

int p = 0;

for(int i=n-k; i<n; i++) y[p++] = i;

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

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

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

for(int i=0; i<m; i++) c[i] += c[i-1];

for(int i=n-1; i>=0; i--) sa[--c[x[y[i]]]] = y[i];

swap(x,y);

p = 1; x[sa[0]] = 0;

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

x[sa[i]] = y[sa[i-1]] == y[sa[i]] && y[sa[i-1]+k] == y[sa[i]+k] ? p-1 : p++;

if(p >= n) break;

m = p;

}

}

int rank[maxn], lcp[maxn];

void get\_lcp(int \*s){

int i, j, k = 0;

for(i=0; i<n; i++) rank[sa[i]] = i;

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

if(k) k--;

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

while(s[i+k] == s[j+k]) k++;

lcp[rank[i]] = k;

}

}

## AC自动机

struct Aho{

struct node{

int next[26];

int fail,cnt;

}state[maxn];

queue<int> q;

int size;

int idx(char ch) {return ch - 'a';}

void init(){

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

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

memset(state[i].next, 0, sizeof(state[i].next));

state[i].fail = state[i].cnt = 0;

}

size = 1;

}

void insert(char \*s){

int n = (int)strlen(s);

int now = 0;

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

int c = idx(s[i]);

if(!state[now].next[c]){

state[now].next[c] = size++;

}

now = state[now].next[c];

}

vec[now].push\_back(id);

state[now].cnt++;

}

void build(){

state[0].fail = -1;

q.push(0);//0是根节点

while(!q.empty()){

int u = q.front();

q.pop();

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

if(state[u].next[i]){

if(u == 0) state[state[u].next[i]].fail = 0;

else{

int v = state[u].fail;//父亲的fail

while(v != -1){

if(state[v].next[i]){//如果该节点的儿子有这条边

state[state[u].next[i]].fail = state[v].next[i];

break;

}

v = state[v].fail;

}

if(v == -1)

state[state[u].next[i]].fail = 0;

}

q.push(state[u].next[i]);

}

else{//按照蓝书上的话说 是把不存在的fail也补上 导致match时可以不需要不断往上跳

if(u == 0) state[u].next[i] = 0;

else state[u].next[i] = state[state[u].fail].next[i];

}

}

}

}

int get(int now){

int ans = 0;

while(now){

ans += state[now].cnt;

state[now].cnt = 0;

now = state[now].fail;

}

return ans;

}

int match(char \*s){

int n = (int)strlen(s);

int now = 0;

int ans = 0;

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

int c = idx(s[i]);

if(state[now].next[c])

now = state[now].next[c];

else{

int p = state[now].fail;

while(p != -1 && state[p].next[c] == 0) p = state[p].fail;

if(p == -1) now = 0;

else now = state[p].next[c];

}

ans += get(now);

}

return ans;

}

}aho;

## KMP

char s[maxn];

char p[maxn];

int NEXT[maxn];

void getNext(){

int m = (int)strlen(s);

NEXT[0] = -1;

int k = -1;

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

while(k != -1 && s[k+1] != s[i])

k = NEXT[k];

if(s[k+1] == s[i])

k++;

NEXT[i] = k;

}

}

int KMP(){

int k = -1;

int n = (int)strlen(s);

int m = (int)strlen(p);

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

while(k != -1 && p[k+1] != s[i])

k = NEXT[k];

if(p[k+1] == s[i])

k = k+1;

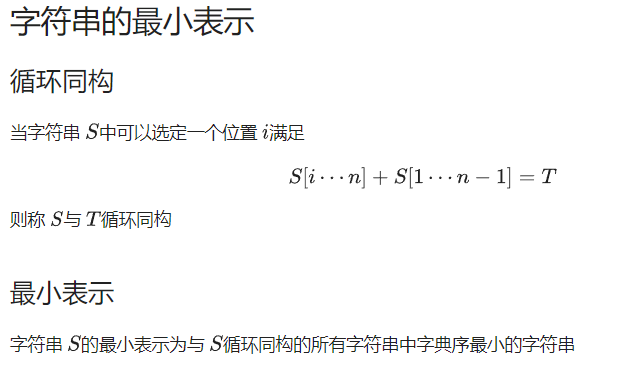
if(k == m-1) return i-m+1;

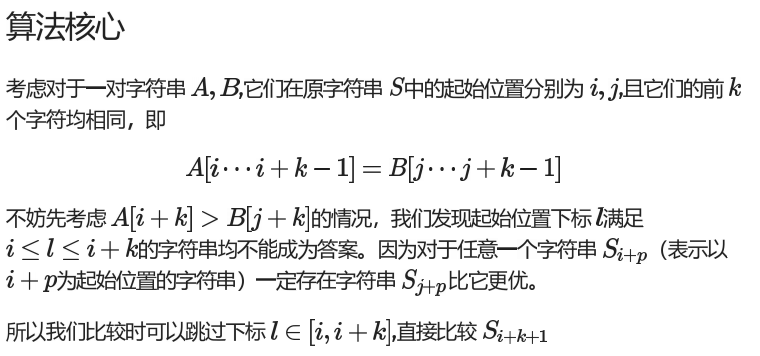
}

return -1;

}

## 最小(大)表示法(O(n))







//最大表示法 返回的是字符串下标

int get\_maxstring(char \*s)

{

int len = strlen(s);

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

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

{

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

if(t==0)

k++;

else

{

if(t > 0)

j+=k+1;

else

i+=k+1;

if(i==j) j++;

k=0;

}

}

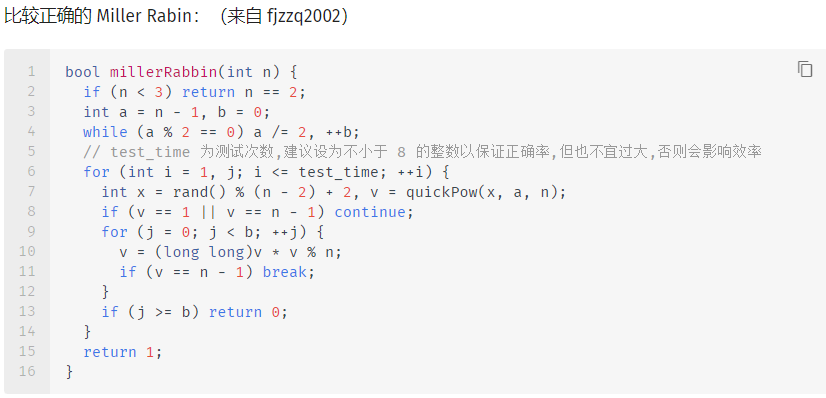
return min(i,j);

}

# 数学

## 素数相关

### Miller-Rabin 素性测试



### 反素数

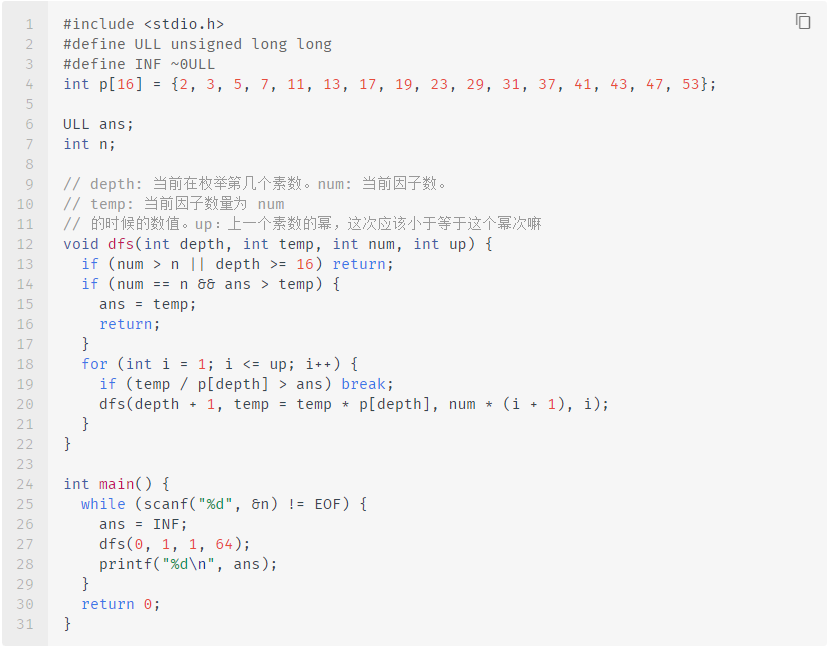
如果某个正整数 满足如下条件，则称为是反素数： 任何小于 的正数的约数个数都小于 的约数个数

注：注意区分[emirp](https://en.wikipedia.org/wiki/Emirp)，它是用来表示从后向前写读是素数的数。

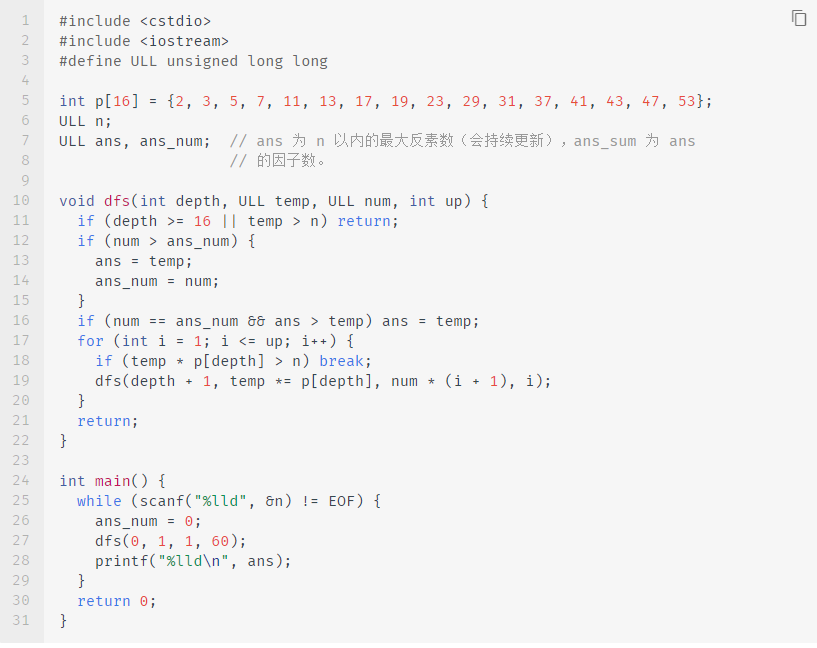
常见题型：

1. 求因子数一定的最小数

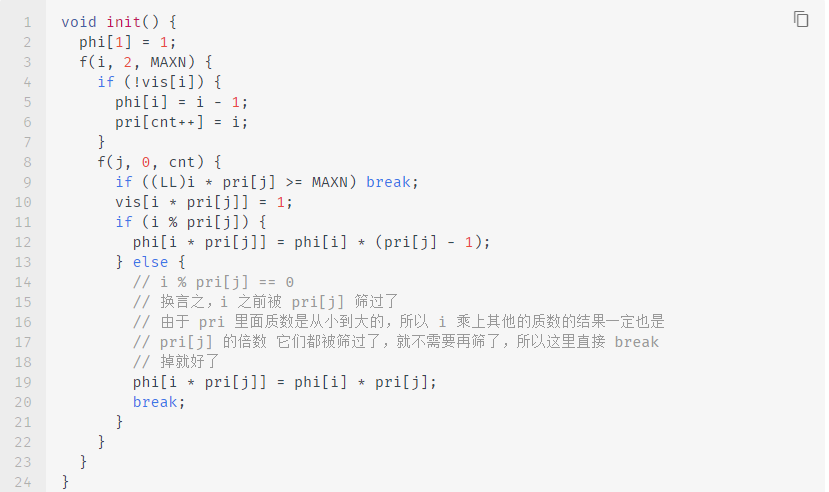
对于这种题，我么只要以因子数为 dfs 的返回条件基准，不断更新找到的最小值就可以了



1. 求n以内因子数最多的数（ZOJ-1562）



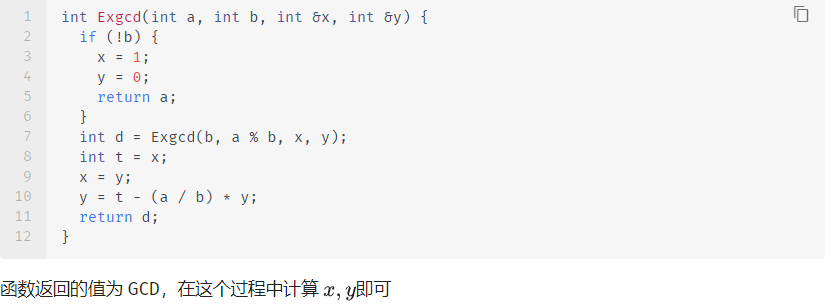
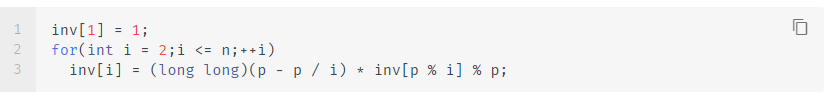
### 线性素数筛(O(n))



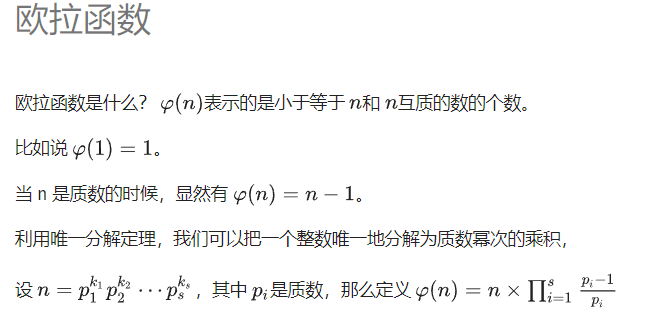
## gcd相关

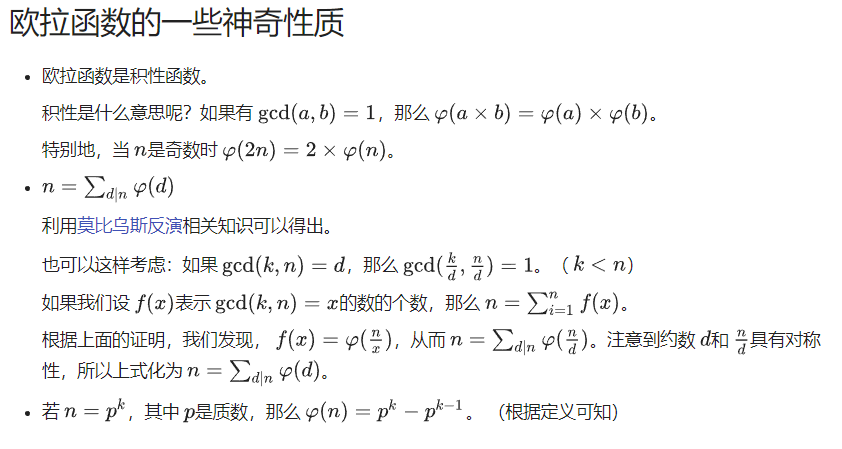
### EXGCD - 扩展欧几里得定理

目的：ax+by=gcd(a,b) 求 的一组可行解

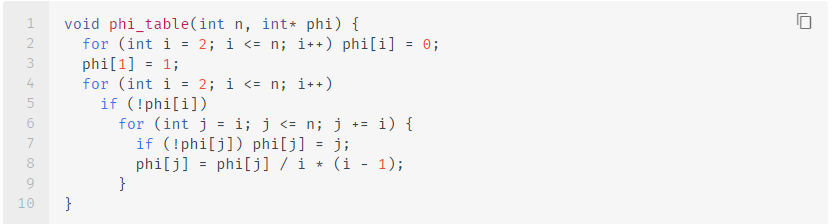
线性递推求逆元

## 欧拉函数相关

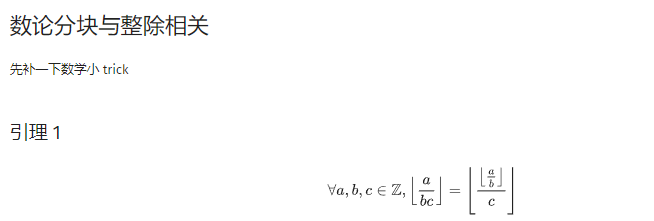


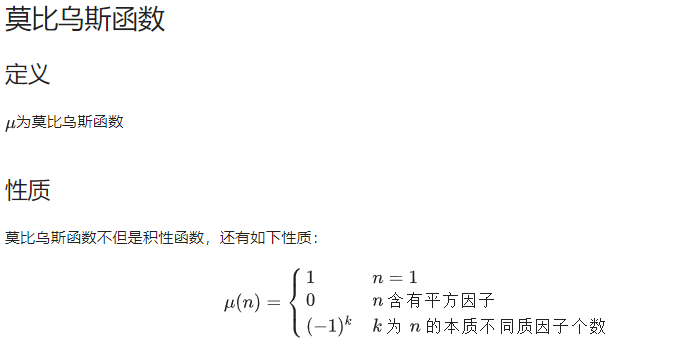


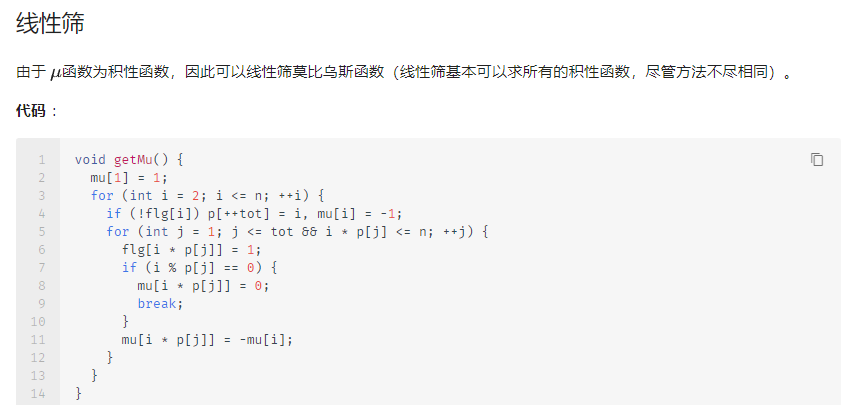
### 筛法求欧拉函数

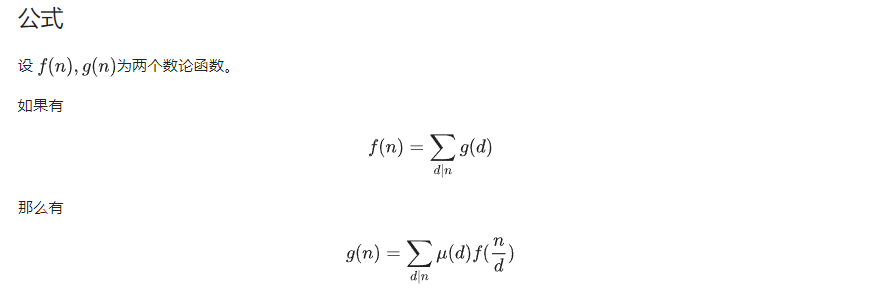


## 莫比乌斯反演









## 高斯消元

1. 高斯消元（浮点数）by kuangbin



1. 整数情况(可能有浮点数解)

#include<stdio.h>

#include<algorithm>

#include<iostream>

#include<string.h>

#include<math.h>

using namespace std;

const int MAXN=50;

int a[MAXN][MAXN];//增广矩阵

int x[MAXN];//解集

bool free\_x[MAXN];//标记是否是不确定的变元

int gcd(int a,int b){

if(b == 0) return a; else return gcd(b,a%b);

}

inline int lcm(int a,int b){

return a/gcd(a,b)\*b;//先除后乘防溢出

}

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

//-1表示无解，0表示唯一解，大于0表示无穷解，并返回自由变元的个数)

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

int Gauss(int equ,int var){

int i,j,k;

int max\_r;// 当前这列绝对值最大的行.

int col;//当前处理的列

int ta,tb;

int LCM;

int temp;

int free\_x\_num;

int free\_index;

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

x[i]=0;

free\_x[i]=true;

}

//转换为阶梯阵.

col=0; // 当前处理的列

for(k = 0;k < equ && col < var;k++,col++){// 枚举当前处理的行.

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

max\_r=k;

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

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

}

if(max\_r!=k){// 与第k行交换.

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

}

if(a[k][col]==0){// 说明该col列第k行以下全是0了，则处理当前行的下一列.

k--;

continue;

}

for(i=k+1;i<equ;i++){// 枚举要删去的行.

if(a[i][col]!=0){

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

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

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

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

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

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

}

}

}

}

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

for (i = k; i < equ; i++){ // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.

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

}

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

// 且出现的行数即为自由变元的个数.

if (k < var){

return var - k; // 自由变元有var - k个.

}

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

// 计算出Xn-1, Xn-2 ... X0.

for (i = var - 1; i >= 0; i--){

temp = a[i][var];

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

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

}

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

x[i] = temp / a[i][i];

}

return 0;

}

int main(void){

// freopen("in.txt", "r", stdin);

// freopen("out.txt","w",stdout);

int i, j;

int equ,var;

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

memset(a, 0, sizeof(a));

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

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

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

}

}

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

if (free\_num == -1) printf("无解!\n");

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

else if (free\_num > 0){

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

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

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

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

}

}else{

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

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

}

}

printf("\n");

}

return 0;

}

1. 简单版本(注意试用条件)

/\*

高斯消元求多项式(取模)的解(整数) 在保证有解的情况下

F(x) = a0+a1\*x^1+a2\*x^2+……+ an\*x^n

已知f(0)%mod,f(1)%mod,……,f(n)%mod （mod是素数）

求a0,a1,……an

\*/

//n为多项式的项数，如果多项式最高次为n次，则应调用gsxy(n+1)

void gsxy(int n)

{

//初始化系数

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

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

a[i][j] = qpow(i,j);

}

}

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

//快速幂求逆元

ll inv = qpow(a[i][i] , mod-2);

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

a[i][j] = a[i][j] \* inv % mod;

}

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

if(j != i){

ll c = a[j][i];

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

a[j][k] = (a[j][k] - a[i][k] \* c % mod + mod) % mod;

}

}

}

}

}

## 中国剩余定理

### 模数互质

//求解模线性方程组x=a\_i(mod m\_i)

ll China\_Reminder(int len, ll\* a, ll\* m)

{

int i;

ll N = 1;

ll result = 0;

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

N = N\*m[i];

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

{

ll tmp = N/m[i];

ll x,y;

exgcd (tmp,m[i],x,y);

x = (x%m[i]+m[i])%m[i];

result = (result + tmp\*a[i]\*x%N)%N;

}

return result;

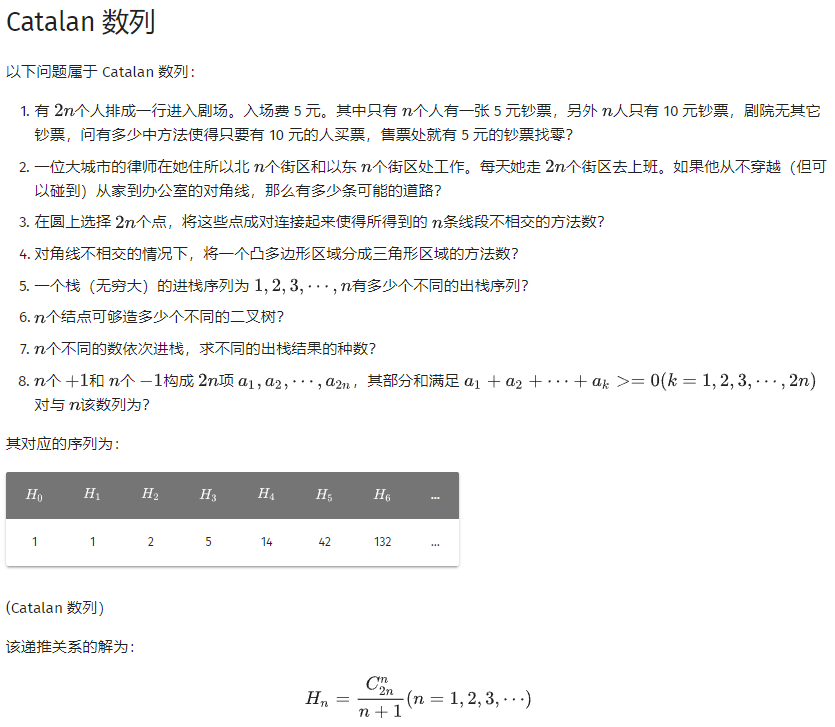
}

### 模数不互质

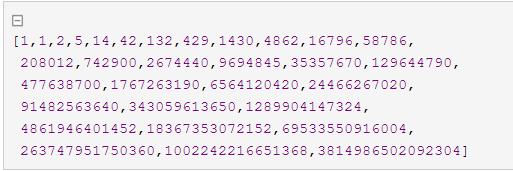
1. int64 Mod;
2. //a在模n乘法下的逆元，没有则返回-1
3. int64 inv(int64 a, int64 n)
4. {
5. int64 x,y;
6. int64 t = exgcd (a,n,x,y);
7. **if**(t != 1)
8. **return** -1;
9. **return** (x%n+n)%n;
10. }
12. //将两个方程合并为一个
13. **bool** merge(int64 a1, int64 n1, int64 a2, int64 n2, int64& a3, int64& n3)
14. {
15. int64 d = gcd(n1,n2);
16. int64 c = a2-a1;
17. **if**(c%d)
18. **return** **false**;
19. c = (c%n2+n2)%n2;
20. c /= d;
21. n1 /= d;
22. n2 /= d;
23. c \*= inv(n1,n2);
24. c %= n2;
25. c \*= n1\*d;
26. c += a1;
27. n3 = n1\*n2\*d;
28. a3 = (c%n3+n3)%n3;
29. **return** **true**;
30. }
32. //求模线性方程组x=ai(mod ni),ni可以不互质
33. int64 China\_Reminder2(**int** len, int64\* a, int64\* n)
34. {
35. int64 a1=a[0],n1=n[0];
36. int64 a2,n2;
37. **for**(**int** i = 1; i < len; i++)
38. {
39. int64 aa,nn;
40. a2 = a[i],n2=n[i];
41. **if**(!merge(a1,n1,a2,n2,aa,nn))
42. **return** -1;
43. a1 = aa;
44. n1 = nn;
45. }
46. Mod = n1;
47. **return** (a1%n1+n1)%n1;
48. }

## 组合数学相关

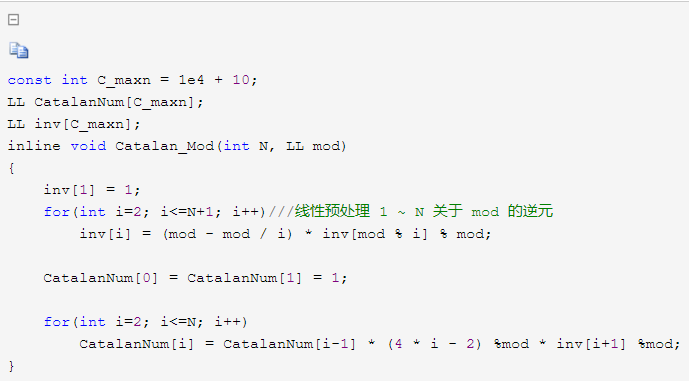
### 卡特兰数



前三十项卡特兰数



卡特兰数求解（取模）：



## 快速傅里叶变换

//UOJ 34 多项式乘法

const int maxn = 1e5+10;

const double PI = acos(-1);

struct Complex

{

double r , i;

Complex(){}

Complex(double rr , double ii):r(rr),i(ii){}

inline Complex operator + (const Complex &a)const { return Complex(r+a.r , i+a.i); }

inline Complex operator - (const Complex &a)const { return Complex(r-a.r , i-a.i); }

inline Complex operator \* (const Complex &a)const { return Complex(r \* a.r - i \* a.i , r \* a.i + i \* a.r); }

};

/\*

\* 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].r /= len;

}

}

}

int n , m;

Complex a[maxn<<4] , b[maxn<<4];//数组大小为nlogn

int main()

{

cin >> n >> m;

//多项式从0次到n次的系数

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

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

}

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

scanf("%lf" , &b[i].r);

}

int len = m + n , cnt;

for(cnt = 1 ; cnt <= len ; cnt <<= 1);

fft(a , cnt , 1);

fft(b , cnt , 1);

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

a[i] = a[i] \* b[i];

}

fft(a , cnt , -1);

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

printf("%d%c" , (int)(a[i].r + 0.5) , i == len ? '\n' : ' ');

}

return 0;

}

# 动态规划

## LIS打印路径

//O（nlogn）

int p[maxn], dp[maxn], who[maxn], pre[maxn], n;

vector<int>path;

int main()

{

scanf("%d",&n);

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

memset(dp,0x7f,sizeof(dp));

int ans=0;

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

{

int u=p[i];

int pos=lower\_bound(dp+1,dp+1+n,u)-dp;

pre[i+1]=who[pos-1];////记录前面

dp[pos]=u;

who[pos]=i+1;/////更新位置

ans=max(pos,ans);

}

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

int now=who[ans];

while(now!=0)

{

path.push\_back(now);

now=pre[now];

}

for(int i=path.size()-1; i>=0; i--)

printf("%d ",path[i]);

return 0;

}

## 数位DP

int dfs(int pos, int sum, bool limit){

if(pos == -1) return sum <= ans;

if(sum > ans) return 0;

if(!limit && dp[pos][ans-sum] != -1) return dp[pos][ans-sum];//记忆化搜索

int up = limit ? arr[pos] : 9;

int tot = 0;

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

tot += dfs(pos-1,sum+i\*(1<<pos),limit && i==arr[pos]);

}

if(!limit) dp[pos][ans-sum] = tot;

return tot;

}

int solve(int a, int b){

int pos = 0;

while(b){

arr[pos++] = b % 10;

b /= 10;

}

return dfs(pos-1,0,true);

}

# 杂项

## 快速乘法(直接乘超过longlong时使用)

快速乘,直接乘会爆ll时需要它，也叫二分乘法。

ll qmul(ll x, ll y, ll mod)

{

ll ret = 0;

while(y) {

if(y & 1)

ret = (ret + x) % mod;

x = x \* 2 % mod;

y >>= 1;

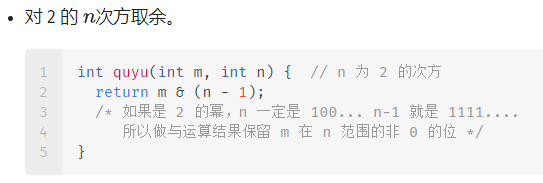
}

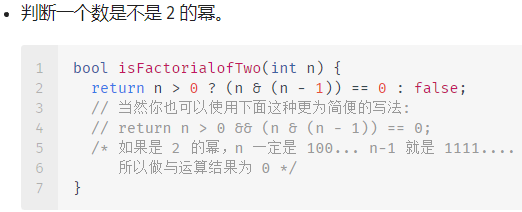
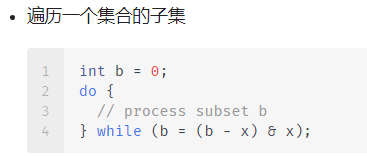
return ret;

}

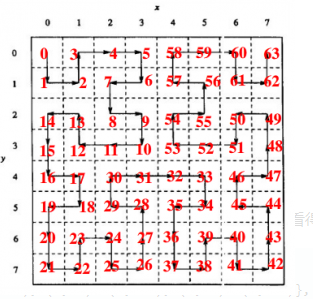
## 位运算相关

一个数的二进制表示可以看作是一个集合（0 表示不在集合中，1 表示在集合中）。比如集合 {1, 3, 4, 8} ，可以表示成 0b00000000000000000000000100011010 ，十进制就是 。而对应的位运算也就可以看作是对集合进行的操作。

## 希尔伯特(Hilbert)曲线



三阶Hilbert曲线：

void rot(int n, int \*x, int \*y, int rx, int ry);

//XY坐标到Hilbert代码转换

int xy2d (int n, int x, int y)

{

int rx, ry, s, d=0;

for (s=n/2; s>0; s/=2)

{

rx = (x & s) > 0;

ry = (y & s) > 0;

d += s \* s \* ((3 \* rx) ^ ry);

rot(s, &x, &y, rx, ry);

}

return d;

}

//Hilbert代码到XY坐标

void d2xy(int n, int d, int \*x, int \*y)

{

int rx, ry, s, t=d;

\*x = \*y = 0;

for (s=1; s<n; s\*=2)

{

rx = 1 & (t/2);

ry = 1 & (t ^ rx);

rot(s, x, y, rx, ry);

\*x += s \* rx;

\*y += s \* ry;

t /= 4;

}

}

void rot(int n, int \*x, int \*y, int rx, int ry)

{

if (ry == 0)

{

if (rx == 1)

{

\*x = n-1 - \*x;

\*y = n-1 - \*y;

}

//Swap x and y

int t = \*x;

\*x = \*y;

\*y = t;

}

}

## 快速输入函数

//读int

inline int readint(){

char ch = getchar();

int f = 1, x = 0;

while(ch < '0'|| ch > '9'){

if(ch=='-')f = -1;

ch = getchar();

}

while(ch >= '0' && ch <= '9'){

x = x \* 10 + ch - '0';

ch = getchar();

}

return f \* x;

}

//读longlong

inline ll readll(){

char ch = getchar();

ll f = 1,x = 0;

while(ch < '0'|| ch > '9'){

if(ch=='-')f = -1;

ch = getchar();

}

while(ch >= '0' && ch <= '9'){

x = x \* 10 + (ll)(ch - '0');

ch = getchar();

}

return f \* x;

}

//调用 n = readint() 相当于 scanf(“%d” , &n);

## 头文件

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

#include <iostream>

#include <cstring>

#include <cmath>

#include <cstdlib>

#include <cstdio>

#include <string>

#include <algorithm>

#include <queue>

#include <map>

#include <vector>

#include <stack>

#include <cctype>

#include <set>

#include <ctype.h>

#include <string.h>