**ACM简陋模板**

**2018.3.12**

1.1头文件

2.1 最短路

2.2最小生成树

2.3网络流

2.4 其他的图算法

3.1线段树

4.1最长上升子序列

5.1 数论

5.2素数

6.1计算几何

7.1字符串

**1.1头文件**

#include<cstdio>

#include<iostream>

#include<queue>

#include<cmath>

#include<cstring>

#include<algorithm>

#define ll long long

using namespace std;

const int INF=0x3f3f3f3f;

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

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

**2.1 最短路**

/\*起点标号均已改为从1开始\*/

/\*\*\*\*\***Dijkstra** POJ - 2387\*\*\*\*\*/

/\*注意重复边的输入情况\*/

int cost[maxn][maxn];

int dis[maxn];

int vis[maxn];

void dij(int x){

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

vis[x]=1;

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

dis[i]=cost[x][i];

dis[x]=0;

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

int min=INF;int u=-1;

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

if(!vis[j]&&dis[j]<min) {

min=dis[j];u=j;

}

}

vis[u]=1;

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

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

dis[j]=cost[u][j]+dis[u];

}

}

}

/\*\*\*\***Bellman-Ford**\*\*\*\*/

/\*单源负环\*/

/\*o(nm)\*/

struct edge{

int s,e,w

}edges[maxm];

int dis[maxn];

void bell(int x){

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

dis[i]=INF;

dis[x]=0;

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

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

if(edges[i].w!=INF&&dis[edges[i].e]>dis[edges[i].s]+edges[i].w)

dis[edges[i].e]=dis[edges[i].s]+edges[i].w;

}

}

}

/\*再判断有无负环回路\*/

/\*返回1则存在负环回路\*/

struct edge{

int s,e,w

}edges[maxm];

int dis[maxn];

int bell(int x){

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

dis[i]=INF;

dis[x]=0;

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

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

if(edges[i].w!=INF&&dis[edges[i].e]>dis[edges[i].s]+edges[i].w){

dis[edges[i].e]=dis[edges[i].s]+edges[i].w;

if(k==n) return 1;/\*若第n次循环的时候路径还能在缩短，说明存在负环回路\*/

}

}

}

return 0;

}

/\*\*\*\***SPFA**\*\*\*\*/

/\*若存在负环回路则返回1\*/

struct edge{

int to,cost;

edge(int To,int Cost):to(To),cost(Cost){};

};

vector<edge>edges[maxn];

int dis[maxn];

int vis[maxn]//标记每个点是否在队列里

int cnt[maxn]//判断是否存在负环回路;若有点更新超过n次，则存在负环

int spfa(int x){

memset(dis,0x3f,sizeof(dis));

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

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

dis[x]=0;

queue<int>que;

que.push(x);

vis[x]=1;cnt[x]=1;

while(!que.empty()){

int t=que.top;que.pop();vis[t]=0;

for(int i=0;i<edges[t].size;i++){

node e=edges[t][i];

if(dis[e.to]>dis[t]+e.cost){

dis[e.to]=dis[t]+e.cost;

if(!vis[e.to]){

que.push(e.to);

vis[e.to]=1;

if(++cnt[e.to]>n) return 1;

}

}

}

}

return 0;}

/\*\*\*\***floyd**\*\*\*\*/

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

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

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

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

**2.2最小生成树**

/\*顶点均已改为从1开始\*/

/\*\*\***并查集**\*\*\*\*/

void init(){

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

p[i]=i;/\*根节点初始化为自己\*/

}

int find(int x){

return x==p[x]?x:find(p[x]);

}

void unin(int x,int y){

x=find(x);y=find(y);

if(x!=y) p[x]=y;

}

/\*\*\*\*最小生成树**Kruskal** poj2421\*\*\*\*/

/\*时间复杂度取决于边数\*/

#define maxm 55//最多边数

#define maxn 11//最多顶点数

struct edge{

int u,v,w;

}edges[maxm];

int p[maxn];

int n,m;

void init(){

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

p[i]=i;

}

int find(int x){

return x==p[x]?x:find(p[x]);

}

void unin(int x,int y){

x=find(x);y=find(y);

if(x!=y) p[x]=y;

}

int cmp(edge a,edge b){

return a.w<b.w;

}

void kruskal(){

int sumweight=0;//生成树的总权值

int num=0;//已选用的边的数目

init();//初始化

sort(edges,edges+m,cmp);

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

int u=edges[i].u;int v=edges[i].v;int w=edges[i].w;

if(find(u)!=find(v))

{unin(u,v);num++;sumweight+=w;

}

if(num==n-1) break;

}

}

/\*\*\*\*\*最小生成树**Prim** \*\*\*\*\*/

/\*时间复杂度取决于顶点数\*/

int cost[maxn][maxn];

int lowcost[maxn];

int prim(){

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

lowcost[i]=cost[0][i];

lowcost[1]=-1;

int sum=0;

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

int min=INF;int u;

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

if(lowcost[j]!=-1&&lowcost[j]<min){

min=lowcost[j];u=j;

}}

if(min==INF) return -1;

lowcost[u]=-1;

sum+=min;

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

if(lowcost[j]!=-1&&lowcost[j]>cost[u][j]){

lowcost[j]=cost[u][j];

}

}

}

return sum;

}

**2.3网络流**

/\*\*\***dinic**算法 HDU - 1532\*\*\*\*/

/\*复杂度o(n^2\*m)\*/

struct edge{

int to,cap;

int rev;//反向边的序号

};

vector<edge>edges[maxn];

int level[maxn];

void addnode(int u,int v,int cap){

edges[u].push\_back((edge){v,cap,edges[v].size()});

edges[v].push\_back((edge){u,0,edges[u].size()-1});

}

void bfs(int s){

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

level[s]=0;

queue<int>que;

que.push(s);

while(!que.empty()){

int u=que.front();que.pop();

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

edge e=edges[u][i];

if(e.cap>0&&level[e.to]==-1){

level[e.to]=level[u]+1;

que.push(e.to);}

}

}

}

int dfs(int u,int t,int f){

if(u==t) return f;

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

edge e=edges[u][i];

if(level[e.to]==level[u]+1&&e.cap>0){

int d=dfs(e.to,t,min(f,e.cap));

if(d>0){

edges[u][i].cap-=d;

edges[e.to][e.rev].cap+=d;

return d;

}

}

}

return 0;

}

int max\_flow(int s,int t){

int flow=0;

while(1){

bfs(s);

if(level[t]<0) return flow;

int f;

while((f=dfs(s,t,INF))>0)

flow+=f;

}

return flow;

}

**2.4其他的图算法**

/\*\*\*\***Tarjan求割点**\*\*\*\*/

int edge[1010][1010];

int vis[1010];

int n;int tdfn;

int son;//根结点的子女结点个数，son>=2表示根结点为割点

int dfn[1010];//结点dfs的编号

int low[1010];//每个结点最低能回到哪里

int subn[1010];//记录去掉某个节点后的连通分量数（需再+1），若为0，则表示不是割点

void dfs(int u){

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

if(edge[u][v]){

if(!vis[v]){

vis[v]=1;

tdfn++;dfn[v]=low[v]=tdfn;

dfs(v);

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

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

if(u!=1) subn[u]++;

if(u==1) son++;

}

}

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

}

}

}

void init(){

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

low[1]=dfn[1]=1;

tdfn=1;son=0;

vis[1]=1;

memset(subn,0,sizeof(subn));

}

/\***强连通分量**\*/

/\* Tarjan算法 \* 复杂度O(N+M)\*/

const int MAXN = 10010;//点数

const int MAXM = 100010;//边数

struct Edge {

int to,next;

}edge[MAXM];

int head[MAXN],tot;

int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong数组的值是1~scc

int Index,top;

int scc;//强连通分量的个数S

bool Instack[MAXN]; int num[MAXN];//各个强连通分量包含点的个数，数组编号1~scc //num数组不一定需要，结合实际情况

void addedge(int u,int v) {

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

}

void Tarjan(int u) {

int v;

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

Stack[top++] = u;

Instack[u] = true;

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

v = edge[i].to;

if(!DFN[v]){

Tarjan(v);

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

Low[u] = Low[v]; }

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

Low[u] = DFN[v]; }

if(Low[u] == DFN[u]) {

scc++;

do {

v = Stack[--top];

Instack[v] = false;

Belong[v] = scc;

num[scc]++;}while( v != u); } }

void solve(int N) {

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

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

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

Index = scc = top = 0;

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

if(!DFN[i])

Tarjan(i); }

void init() {

tot = 0; memset(head,-1,sizeof(head)); }

/\*\*\*\***二分图最大匹配** 匈牙利算法\*\*\*\*/

int line[50][50];

int used[50];//标记这条边有没有用过

int match[50];//标记右侧的点是否被匹配，以及匹配的是左侧哪个点

int nl;

int nr;

bool find(int x){

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

if(line[x][i]&&!used[i]){

used[i]=1;

if(match[i]==0||find(match[i])){

match[i]=x;

return true;

}

}

}

return false;

}

int hungarian()

{

int ans = 0;

memset(match,0,sizeof(match));

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

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

if(find(i)) ans++;

}

return ans;

}

/\*\***拓扑排序**\*\*/

queue<int>q;

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

//优先队列的话，会按照数值大小有顺序的输出

//此处为了理解，暂时就用简单队列

int ind[maxn];//入度

int topo()

{

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

{

if(ind[i]==0)

{

q.push(i);

}

}

int temp;

while(!q.empty())

{

temp=q.front();

q.pop();

//此处可输出排序结果temp

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

{

if(edge[temp][i])

{

ind[i]--;

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

}

}

}

}

**3.1线段树**

/\*\*\*\***单点更新** 区间查询 HDU - 1166\*\*\*\*/

int tree[maxn<<2];

void pushup(int rt){

tree[rt]=tree[rt\*2]+tree[rt\*2+1];

}

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

if(l==r) {scanf("%d",&tree[rt]);return;}

int mid=(l+r)/2;

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

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

pushup(rt);

}

int query(int l,int r,int ll,int rr,int rt){//l,r为当前边界

if(l>=ll&&r<=rr) return tree[rt];

int ans=0;int mid=(l+r)/2;

if(ll<=mid){

ans+=query(l,mid,ll,rr,rt\*2);

}

if(rr>mid){

ans+=query(mid+1,r,ll,rr,rt\*2+1);

}

return ans;

}

void update(int l,int r,int index,int add,int rt){

if(l==r) {

tree[rt]+=add;return;

}

int mid=(l+r)/2;

if(index<=mid){

update(l,mid,index,val,rt\*2);

}

else update(mid+1,r,index,val,rt\*2+1);

pushup(rt);

}

/\*\*\*\***区间更新** poj-3468\*\*\*\*/

int tree[maxn<<2];

int seg[maxn<<2];

void pushup(int rt){

tree[rt]=tree[rt\*2]+tree[rt\*2+1];

}

void pushdown(int len,int rt){

if(seg[rt]){

seg[rt\*2]+=seg[rt];

seg[rt\*2+1]+=seg[rt];

tree[rt\*2]+=(len-len/2)\*seg[rt];

tree[rt\*2+1]+=len/2\*seg[rt];

seg[rt]=0;

}

}

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

seg[rt]=0;

if(l==r) {

scanf("%d",&tree[rt]);return;}

int mid=(l+r)/2;

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

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

pushup(rt);

}

void update(int l,int r,int ll,int rr,int add,int rt){//l,r为当前区间，ll,rr为更新的区间

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

seg[rt]+=add;tree[rt]+=(r-l+1)\*add;

return;

}

pushdown(r-l+1,rt);

int mid=(l+r)/2;

if(ll<=mid) {

update(l,mid,ll,rr,add,rt\*2);

}

if(rr>mid){

update(mid+1,r,ll,rr,add,rt\*2+1);

}

pushup(rt);

}

int query(int l,int r,int ll,int rr,int rt){//ll,rr为查询的区间

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

return tree[rt];

}

int ans=0;

pushdown(r-l+1,rt);

int mid=(l+r)/2;

if(ll<=mid) ans+=query(l,mid,ll,rr,rt\*2);

if(rr>mid) ans+=query(mid+1,r,ll,rr,rt\*2+1);

return ans;

}

**4.1最长上升子序列等dp算法**

/\***最长上升子序列**o nlogn\*/

const int MAXN=500010; int a[MAXN],b[MAXN];

//用二分查找的方法找到一个位置，使得num>b[i-1] 并且num<b[i],并用num代替b[i]

int Search(int num,int low,int high) {

int mid;

while(low<=high) {

mid=(low+high)/2;

if(num>=b[mid]) low=mid+1;

else high=mid-1;

}

return low;

}

int DP(int n) {

int i,len,pos;

b[1]=a[1];

len=1;

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

if(a[i]>=b[len])//如果a[i]比b[]数组中最大还大直接插入到后面即可

{

len=len+1; b[len]=a[i]; }

else//用二分的方法在b[]数组中找出第一个比a[i]大的位置并且让a[i]替代这个位置

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

b[pos]=a[i];

}}

return len; }

/\*0-1**背包**\*/

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

for(int j=v;j>=c[i];j--)

dp[j]=max(dp[j],dp[j-c[i]]+w[i]);

/\*完全背包\*/

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

for(int j=c[i];j<=v;j++)

dp[j]=max(dp[j],dp[j-c[i]]+w[i]);

/\*

**组合数**： c(m,n)=c(m-1,n)+c(m-1,n-1);

\*/

**5.1 数论**

/\*\*\*\*\***欧几里得辗转相除**\*\*\*\*/

ll gcd(ll a,ll b){

if(a < b){

long long temp;

temp = a;

a = b;

b = temp;

}

while(a % b){

ll r = a % b;

a = b;

b = r;

r = a % b;

}

return b;

}

/\*\*\*\***拓展欧几里得**\*\*\*\*/

/\*求ax+by=c的xy 其中c%gcd(a,b)==0时才有解

解出的为特解x0,y0 \*/

ll ex\_gcd(ll a,ll b,ll &x,ll &y)

{

if(a==0&&b==0) return -1;//无最大公约数

if(b==0){x=1;y=0;return a;}

ll d=ex\_gcd(b,a%b,y,x);

y-=a/b\*x;

return d;

}

/\***求最小正整数解**x;若无解返回false\*/

/\*已求出特解 x0,y0 设t=c/d,s=b/d,则通解为 x=t\*x0+s\*k; y=t\*y0-s\*k; k为任意整数

最小正整数解 x=x0\*t; x=(x%s+s)%s;\*/

bool min\_x(int a,int b,int &x,int &y,int c)

{

int d=ex\_gcd(a,b,x,y);

if(c%d) return false;

x=x\*c/d;

int s=b/d;

s=s>0?s:-1\*s;

x=(x%s+s)%s;

y=(c-a\*x)/b;

//cout<<"x = "<<x<<endl;

//cout<<"y = "<<y<<endl;

return true;

}

/\*\*\*\***求逆元素**\*\*\*\*/

/\*ax = 1(mod n) 即求ax+ny=1中x的解\*/

ll mod\_reverse(ll a,ll n) {

ll x,y;

ll d=ex\_gcd(a,n,x,y);

if(d==1) return (x%n+n)%n;

else return -1;

}

/\*\*\*\*费马小定理求逆元\*\*\*\*/

/\*求ax = 1( mod m) 的x值 要求 0<a<m 且a,m互质 \*/

ll inv(ll a,ll m){

if(a == 1) return 1;

return inv(m%a,m)\*(m-m/a)%m;

}

/\*\*\*\*利用欧拉函数求逆元\*\*\*\*/

/\*mod为素数,而且a和m互质\*/

ll inv(ll a,ll mod) {

return pow\_m(a,mod-2,mod);

}

/\***中国剩余定理**\*/

/\*x=bi(mod m0i)\*/

/\*x0+=biMiyi\*/

ll ChinaRemainder(ll m0[],ll b[],int n){

ll m=1,a=0;

int i;

for (i=1; i<= n ; i++) m=m\*m0[i];

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

MM=m/m0[i];

ll x=mod\_reverse(MM,mo[i]);

a=(a+MM\*x\*b[i]) % m;

}

return a;

}

/\*\*\*\***快速幂**\*\*\*\*/

ll quickmod(ll a,ll b,ll c){

ll res=1;

while(b){

if(b&1)

res=res\*a%c;

a=a\*a%c;

b>>=1;

}

return res;

}

/\*\*\*\***矩阵快速幂**\*\*\*\*/

typedef vector<int> vec;

typedef vector<vec> mat;

const int MOD=10000;

//计算A\*B

mat mul(mat &A,mat &B){

mat C(A.size(),vec(B[0].size()));

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

for(int k=0;k<B.size();k++)

for(int j=0;j<B[0].size();j++)

C[i][j]=(C[i][j]+A[i][k]\*B[k][j])%MOD;

return C;

}

//计算 A^n

mat pow(mat A,ll n){

mat B(A.size(),vec(A.size()));

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

B[i][i]=1;

while(n>0){

if(n&1) B=mul(B,A);

A=mul(A,A);

n>>=1;

}

return B;

}

//输入

ll n;

void solve(){

mat A(2,vec(2));//行数，列数

A[0][0]=1;A[0][1]=1;

A[1][0]=1;A[1][1]=0;

A=pow(A,n);

cout<<A[1][0]<<endl;

}

**5.2素数**

/\*\*\*\*一个比较快的**素数判断**\*\*\*\*/

/\*可以判断一个 < 2^63 的数是不是素数 \*/

/\*用法：若 miller\_rabin(n)为1，则n为素数 \*/

ll mul(ll x,ll y,ll Z)//大数相乘取模

{

ll tmp=x/(long double)Z\*y+1e-3;

return (x\*y+Z-tmp\*Z)%Z;

}

ll MUL(ll x,ll p,ll Z)

{

ll y=1;

while(p)

{

if(p&1)y=mul(y,x,Z);

x=mul(x,x,Z);

p>>=1;

}

return y;

}

bool miller\_rabin(ll n)

{

if(n<=1)return 0;

if(n==2)return 1;

if(n%2==0)return 0;

ll p=n-1;

srand(time(NULL));

int TIMES=8;

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

{

ll x=rand()%(n-1)+1;

if(MUL(x,p,n)!=1)return 0;

}

return 1;

}

/\*\*\*\***素数打表**\*\*\*\*/

/\*notprime为true表示不是素数 \*/

const int MAXN=1000010;

bool notprime[MAXN];

void init() {

memset(notprime,false,sizeof(notprime));

notprime[0]=notprime[1]=true;

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

if(!notprime[i]) {

if(i>MAXN/i)continue;//防止溢出,或者i,j用long long

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

notprime[j]=true;}

}

/\*\*\*\***素数筛选**\*\*\*\*/

/\*prime数组存小于等于MAXN的素数 prime[0] 存的是素数的个数 \*/

const int MAXN=10000;

int prime[MAXN+1];

void getPrime() {

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

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

if(!prime[i])

prime[++prime[0]]=i;

for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++){

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

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

} }

}

**6.1计算几何**

const double eps = 1e-8;

const double PI = acos(-1.0);

int sgn(double x) {/\*判断x的正负性\*/

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

if(x < 0)return -1;

else return 1;

}

/\*\*\*\***Point 定义**\*\*\*\*\*/

struct Point {

double x,y;

Point(){};

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

Point operator- (const Point &b){

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

}

double operator^ (const Point &b){

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

}

double operator\*(const Point &b){

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

}

};

/\*判断**点c是否在线段ab上** 已验证\*/

bool onseg(Point a,Point b,Point c){

if(sgn((b-a)^(c-a))==0&&

sgn((c.x-a.x)\*(c.x-b.x))<= 0 && sgn((c.y-a.y)\*(c.y-b.y))<= 0)

return 1;

return 0;

}

/\*\*\*\***Line 定义**\*\*\*\*/

struct Line{

Point s,t;

Line(){};

Line(Point \_s,Point \_t){s=\_s;t=\_t;}

};

/\***判断线段是否相交** 51Nod - 1264 已验证\*/

/\*为1则相交\*/

bool inter(Line l1,Line l2){

if(sgn((l1.s-l2.s)^(l1.s-l1.t))\*sgn((l1.s-l1.t)^(l1.s-l2.t))<0) return 0;

if(sgn((l2.s-l1.s)^(l2.s-l2.t))\*sgn((l2.s-l2.t)^(l2.s-l1.t))<0) return 0;

if(min(l1.s.x,l1.t.x)<=max(l2.s.x,l2.t.x)&&min(l1.s.y,l1.t.y)<=max(l2.s.y,l2.t.y)

&&min(l2.s.x,l2.t.x)<=max(l1.s.x,l1.t.x)&&min(l2.s.y,l2.t.y)<=max(l1.s.y,l1.t.y))

return 1;

return 0;

}

/\***计算多边形面积**\*/

double calarea(Point p[],int n){

double res=0;

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

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

}

return fabs(res);

}

int gcd ( int a,int b){

if(b==0) return a;

else return gcd(b,a%b);

}

/\***多边形边上整点数**\*/

int onedge(Point p[],int n){

int ans=0;

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

ans+=gcd(fabs(p[(i+1)%n].x-p[i].x),fabs(p[(i+1)%n].y-p[i].y));

}

return ans;

}

/\***多边形内部整点数** s（面积）=l/2（边上整点数/2）+n(内部整点数）-1\*/

int inside(Point p[],int n){

return calarea(p,n)+1-onedge(p,n)/2;

}

/\***点在多边形内** ZOJ - 1081 已验证\*/

/\*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){

Point p1(-INF+100,p.y);

Line ray(p,p1);

Line side;

int cnt=0;

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

side.s=poly[i];

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

if(onseg(side.s,side.t,p)) return 0;

if(sgn(side.s.y-side.t.y)==0) continue;//平行轴不考虑

if(onseg(ray.s,ray.t,side.s)){

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

else if(onseg(ray.s,ray.t,side.t)){

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

else if(inter(side,ray)) cnt++;

}

if(cnt%2) return 1;

return -1;

}

/\***判断凸多边形**\*/

/\*点的编号1~n-1 凸边形为true\*/

bool isconvex(Point poly[],int n) {

bool s[3];

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

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

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

if(s[0] && s[2])

return false; }

return true; }

double dissq(Point a,Point b){

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

}

/\***求凸包，Graham算法**\*/

/\*返回凸包结果st[0~top-1]为凸包边上的点编号 top为凸包边上的点的总数\*/

Point lis[maxn];

int st[maxn],top;

bool cmp(Point a,Point b){

int flag=sgn((a-lis[0])^(b-lis[0]));

if(flag>0) return 1;

if(flag==0&&sgn(dissq(a,lis[0])-dissq(b,lis[0]))<=0) return 1;

return 0;

}

void graham(int n){

Point p0;

p0=lis[0];int k=0;

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

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

p0=lis[i];

k=i;

}

}

lis[k]=lis[0];

lis[0]=p0;

sort(lis+1,lis+n,cmp);

if(n==1){

st[0]=0;

top=1;

return;

}

if(n==2){

st[0]=0;

st[1]=1;

top=2;

return;

}

st[0]=0;

st[1]=1;

top=2;

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

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

top--;

st[top++]=i;

}

}

7.1字符串

/\*\*\*\***kmp**(手码未验证版）\*\*\*\*/

const int maxn=100;

int f[maxn];

void getft(const string &a){

memset(f,0,sizeof(f));

f[0]=-1;

int i=0,k=-1;

while(i<a.size()-1){

if(k==-1||a[i]==a[k])

f[++i]=f[++k];

else k=f[k];

}

}

int kmp(const string &t,const string &pat,int pos=0){

getft(pat);

int i=0,j=0;

while(i<t.size()&&j<pat.size()&&pat.size()-j<=t.size()-i){

if(j==-1||t[i]==pat[j]){

i++;j++;

}

else{

j=f[j];

}

}

if(j<pat.size())

return -1;

else return i-j;

}

/\*\*\*\***AC自动机**\*\*\*\*/

struct Aho{

struct state{

int next[26];

int fail,cnt;

}statetable[maxn];

int size;

queue<int>que;

void init(){

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

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

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

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

}

size=1;

}

void insert(char \*s){

int n=strlen(s);

int now=0;

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

char c=s[i];

if(!statetable[now].next[c-'a'])

statetable[now].next[c-'a']=size++;

now=statetable[now].next[c-'a'];

}

stateble[now].cnt++;

}

void build(){

statetable[0].fail=-1;

que.push(0);

while(!que.empty()){

int u=que.front();

que.pop();

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

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

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

else {v=statetable[u].fail;

while(v!=-1){

if(statetable[v].next[i]){

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

break;

}

v=statetable[v].fail;

}

if(v==-1) statetable[statetable[u].next[i]]=0;

}

que.push(statetable[u].next[i]);

}

}}

}

int match(char \*s){

int n=strlen(s);

int res=0;int now=0;

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

char c=s[i];

if(statetable[now].next[c-'a']){

now=statetable[now].next[c-'a'];

}

else{

int p=statetable[now].fail;

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

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

else now=statetable[p].next[c-'a'];

}

//if(statetable[now].cnt) res=res+get(now);

}

}

}aho;