**ACM简陋模板**

**2017.12.24**

1.1头文件

2.1 最短路

2.2最小生成树

2.3网络流

3.1线段树

4.1最长上升子序列

5.1 欧几里得

5.2求逆元

5.3素数

6.1计算几何

**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;

**2.1 最短路**

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

/\*第一个点从0开始计算\*/

int cost[maxn][maxn];

int dis[maxn];

int vis[maxn];

void dij(int x){

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

vis[x]=1;

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

dis[x]=0;

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

int min=INF;int u=-1;

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

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

min=dis[j];u=j;

}

}

vis[u]=1;

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

dis[i]=INF;

dis[x]=0;

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

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

dis[i]=INF;

dis[x]=0;

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

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

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

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

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

**2.2最小生成树**

/\*\*节点编号均从0开始\*\*/

/\*\*\***并查集**\*\*\*\*/

void init(){

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

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

lowcost[0]=-1;

int sum=0;

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

int min=INF;int u;

for(int j=0;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=0;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;

}

**3.1线段树**

/\*\*\*\*区间更新 单点查询 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**

/\*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; }

**5.1 欧几里得**

/\*\*\*\*\***辗转相除法**\*\*\*\*/

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;

}

**5.2求逆元**

/\*\*\*\*拓欧求逆元\*\*\*\*/

/\*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);

}

**5.3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**

/\*\*\*\***Point 定义**\*\*\*\*\*/

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;

}

struct Point {

double x,y;

Point(){}

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

Point operator -(const Point &b)const {

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

**//叉积**

double operator ^(const Point &b)const{

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

**//点积**

double operator \*(const Point &b)const{

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

**//绕原点旋转角度B（弧度值），后x,y的变化**

void transXY(double B) {

double tx = x,ty = y;

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

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

};

/\*\*\*\***Line** 定义\*\*\*\*/

struct Line{

Point s,e;

Line(){}

Line(Point \_s,Point \_e) { s = \_s;e = \_e; }

//**两直线相交求交点**

//第一个值为0表示直线重合，为1表示平行，为0表示相交,为2是相交

//只有第一个值为2时，交点才有意义

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

Point res = s;

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

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

return make\_pair(0,res);//重合

else return make\_pair(1,res);//平行

}

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

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

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

return make\_pair(2,res);}

};

/\*\*\***判断线段相交**\*\*\*\*/

/\*返回1为相交\*/

bool inter(Line l1,Line l2) {

return

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

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

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

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

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

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

}

/\*\*\***计算多边形面积**\*\*\*/

/\*点的编号从0~n-1\*/

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

double res = 0;

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

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

return fabs(res);

}

/\*\*\*\***判断点在任意多边形内**\*\*\*\*/

//射线法，poly[]的顶点数要大于等于3,点的编号0~n-1

//返回值 -1:点在凸多边形外 0:点在凸多边形边界上 1:点在凸多边形内

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

{

int cnt;

Line ray,side; cnt = 0;

ray.s = p;

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

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

{

side.s = poly[i];

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

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

//如果平行轴则不考虑

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

if(OnSeg(side.s,ray))

{

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

}

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

{

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

}

else if(inter(ray,side))

cnt++;

}

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

else return -1; }