**Template**

#include <bits/stdc++.h>  
#define LL long long  
#define ULL unsigned long long  
#define FOR(i,a,b) for(int i=a;i<=b;i++)  
#define FO(i,a,b) for(int i=a;i<b;i++)  
#define FOD(i,a,b) for(int i=a;i>b;i--)  
#define FORD(i,a,b) for(int i=a;i>=b;i--)  
#define FORV(i,a) for(typeof(a.begin()) i = a.begin(); i != a.end(); i++)  
#define st first  
#define nd second  
#define pb push\_back  
#define mp make\_pair  
using namespace std;  
  
typedef pair<int,int>II;  
template<class T> int getbit(T s, int i) { return (s >> i) & 1; }  
template<class T> T onbit(T s, int i) { return s | (T(1) << i); }  
template<class T> T offbit(T s, int i) { return s & (~(T(1) << i)); }  
template<class T> int cntbit(T s) { return \_\_builtin\_popcount(s);}  
template<class T> T gcd(T a, T b){ T r; while (b != 0) { r = a % b; a = b; b = r; } return a;}  
  
typedef pair<LL,LL> PLL;

**Disjoint Set**

int P[MAXN];

int rank[MAXN];

int findSet(int x){ if(x != P[x]) P[x] = findSet(P[x]); return P[x]; }

void merge(int x, int y){ int PX = findSet(x); int PY = findSet(y); P[PY] = PX; }

**RMQ**

int f[MAXN][LOGMAXN];

int n;

void preRMQ() {

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

f[i][0] = i;

int n1, n2;

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

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

int x = f[i][j - 1];

int y = f[i + (1 << (j - 1))][j - 1];

f[i][j] = a[x] < a[y] ? x : y;

}

}

int get(int x, int y) {

int k = log2(y - x + 1);

int n1 = f[x][k];

int n2 = f[y - (1 << k) + 1][k];

return a[n1] < a[n2] ? n1: n2;

}

**BIT**

|  |  |
| --- | --- |
| 1. BIT 1D - Range query/update   int n;  LL B1[MAXN], B2[MAXN];  LL query(int i){  LL mul = 0, add = 0;  LL at = i;  while(i){  mul += B1[i];  add += B2[i];  i -= i & (-i);  }  return mul \* at - add;  }  LL range\_query(int a, int b){  return query(b) - query(a - 1);  }  void update(LL \*t, int i, LL v){  while(i <= n){  t[i] += v;  i = i + (i & (-i));  }  }  //add all elements from index a -> b  void range\_update(int a, int b, LL v){  update(B1, a, v);  update(B1, b + 1, -v);  update(B2, a, v \* (a - 1));  update(B2, b + 1, - b \* v);  } | 2. BIT 2D -Point Query  int sum[MAXN+3][MAXN+3],a[MAXN+3][MAXN+3];  void insert\_t(int x,int y, int val) {  while (x <= MAXN) {  int y1 = y;  while (y1 <= MAXN) {  sum[x][y1] += val;  y1 += (y1&(-y1));  }  x += (x&(-x));  }  }  int sum\_t(int x, int y) {  int t = 0;  while (x>0) {  int y1 = y;  while (y1 > 0) {  t += sum[x][y1];  y1 &= (y1-1);  }  x &= (x-1);  }  return t;  }  void init() {  FOR (i,1,MAXN)  FOR (j,1,MAXN) {  sum[i][j] = 0; a[i][j] = 0;  }  } |

**LCA**

int n, m; int const MAXN = 1e5 + 10;

int level[MAXN];//level[i]: muc cua i

int cnt[MAXN]; //cnt[i]: trong luong cua cay con goc i

int T[MAXN]; //T[i]: cha cua i

int P[MAXN][20];//P[i][j]: cha thu 2^j cua i

bool mark[MAXN];

vector < vector <int> > adj(MAXN);

void DFS(int node, int height) {

cnt[node] = 1;

mark[node] = true; level[node] = height;

for(int i = 0; i < adj[node].size(); ++i) {

int v = adj[node][i];

if(!mark[v]) {

T[v] = node;

DFS(v, height + 1);

cnt[node] += cnt[v];

}

}

}

void preLCA() {

for (int i = 1; i <= n; i++) P[i][0] = T[i];

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

if(P[i][j-1] != 0)

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

}

int LCA(int x, int y) {

if(level[x] < level[y]) swap(x, y);

int log = log2(level[x]);

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

if(level[x] - (1 << i) >= level[y]) {

x = P[x][i];

}

}

if(x == y) return x;

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

if(P[x][i] != 0 && P[y][i] != P[x][i]) {

x = P[x][i]; y = P[y][i];

}

}

return T[x];

}

//tim cha thu k cua x

int k\_th\_node(int x, int k) {

if(k == 0)

return x;

int log = log2(k);

for(int i = log; i >= 0; i--)

if(1 & (k >> i)) {

x = P[x][i];

k = k - (1 << i);

}

return x;

}

**Left-Right O(n)**

int a[MAXN], n;

int L[MAXN], R[MAXN]; //L[i] ( hoặc R[i]) vị trí đầu tiên bên trái (hoặc bên phải) nhỏ hơn a[i];

stack <int> s; s.push(0);

a[0] = a[n+1] = 0;

FOR(i,1,n+1){

while(a[i] < a[s.top()]){

R[s.top()] = i;

s.pop();

}

if (a[i] == a[s.top()]) L[i] = L[s.top()];

else L[i] = s.top();

s.push(i);

}

**Euler Totient Function**

//counts the totatives of n, that is, the positive integers less than or equal to n that are relatively prime to n.

long long int phi(long long x) {

long long int ret = 1,i,pow;

for (i = 2; x != 1; i++) {

pow = 1;

if(i \* i> x)break;

while (!(x%i)) {

x /= i;

pow \*= i;

}

ret \*= (pow - (pow/i));

}

if(x!=1)ret\*=(x-1);

return ret;

}

**Extend\_euclide**

//ax + by = gcd(a,b)

**LL extend\_euclide(LL a, LL b){**

**LL xa = 1; LL ya = 0;**

**LL xb = 0; LL yb = 1;**

**while(b){**

**LL q = a / b;**

**LL r = a % b; a = b; b = r;**

**LL xr = xa - q \* xb;**

**LL yr = ya - q \* yb;**

**xa = xb; ya = yb;**

**xb = xr; yb = yr;**

**}**

**return (xa + MOD) % MOD;**

**}**

**Powerful tricks with modulo**

1. **(A / B) % MOD = (A % (MOD \* B)) / B**

**Conditions: none**

1. **(A / B) % MOD = ((A % MOD) \* (Bphi(MOD) - 1 % MOD)) % MOD**

**where phi is [Euler's totient function](https://en.wikipedia.org/wiki/Euler%27s_totient_function)**

***Conditions:* B and MOD are coprimes.**

1. **(A / B) % MOD = ((A % MOD) \* (BMOD - 2 % MOD)) % MOD**

**Conditions: B and MOD are coprimes, MOD is a prime number.**

1. **AN % MOD = AN % phi(MOD) % MOD**

**Conditions: A and MOD are coprimes.**

**Matrix**

struct matrix {

int x[maxn + 5][maxn + 5];

int sz;

matrix() {

memset(x, 0, sizeof(x));

}

matrix unit() {

matrix res;

res.sz = sz;

for (int i = 1; i <= sz; i++) res.x[i][i] = 1;

return res;

}

matrix operator + (matrix A) {

matrix res;

res.sz = sz;

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

res.x[i][j] = (x[i][j] + A.x[i][j]) % MOD;

}

return res;

}

matrix operator \* (matrix A) {

matrix res;

res.sz = sz;

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

res.x[i][j] = (res.x[i][j] + 1ll \* x[i][k] \* A.x[k][j]) % MOD;

}

return res;

}

matrix operator ^ (long long k) {

if (!k) return unit();

matrix res, tmp;

res.sz = tmp.sz = sz;

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

res.x[i][j] = tmp.x[i][j] = x[i][j];

}

k--;

while (k) {

if (k & 1) res = res \* tmp;

tmp = tmp \* tmp;

k >>= 1;

}

return res;

}

matrix sumpower(matrix A, long long k) {

if (k == 1) return A;

if (k & 1) return (A ^ k) + sumpower(A, k - 1);

k >>= 1;

matrix T;

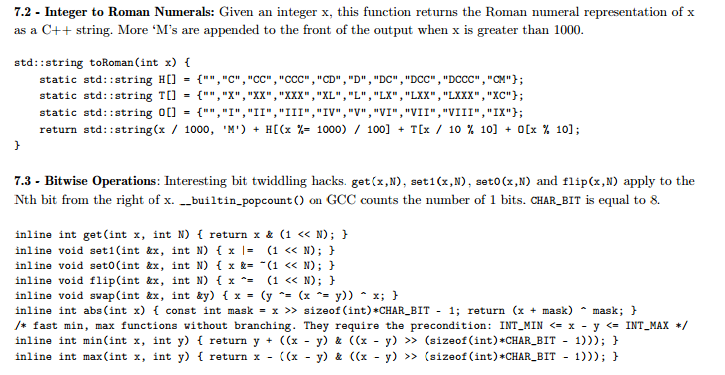
T.sz = sz;

T = sumpower(A, k);

return T + (T \* (A ^ k));

}

};



**Miller-Rabin**

**long long longRand(long long p) {  
 long long result = (long long) rand() << 32 + rand();  
 long long mod = p - 1;  
 return ((result % mod) + mod) % mod + 1;  
}  
  
long long mul(long long a, long long b, long long m) {  
 if(b == 0) return 0;  
 long long ans = mul(a, (b >> 1), m);  
 if(b & 1) {  
 return ((ans + ans) % m + a) % m;  
 }  
 return (ans + ans) % m;  
}  
  
long long power(long long a, long long b, long long m) {  
 long long ans = 1, y = a;  
 while(b > 0) {  
 if(b & 1) {  
 ans = mul(ans, y, m);  
 }  
 y = mul(y, y, m);  
 b >>= 1;  
 }  
 return ans;  
}  
  
bool miller(long long p, int iteration) {  
 if(p == 2) return true;  
 if(p < 2 || p % 2 == 0) return false;  
 long long d = p - 1;  
 while(d % 2 == 0) {  
 d >>= 1;  
 }   
 long long s = (p - 1) / d;  
 for (int i = 1; i <= iteration; i++) {  
 long long a = longRand(p);  
 long long mod = power(a, d, p);  
 if(mod == 1) continue;  
 for(int i = 0; i < s; i++) {  
 if(mod == p - 1) break;  
 mod = mul(mod, mod, p);  
 }  
 if(mod != p - 1) return false;  
 }  
 return true;  
}**

**Trie**

|  |  |
| --- | --- |
| struct Trie {  int value;  Trie \*child[2];  };  Trie \*newTrie() {  Trie \*trie = new Trie;  trie->value = 0;  trie->child[0] = NULL;  trie->child[1] = NULL;  return trie;  }  void insert(Trie \*root, int x) {  Trie \*tmp = root;  for(int i = 30; i >= 0; i--) {  int val = getBit(x, i);  if(tmp->child[val] == NULL) {  tmp->child[val] = newTrie();  }  tmp = tmp->child[val];  }  tmp->value = x;  }  int query(Trie \*root, int x) {  Trie \*tmp = root;  for(int i = 30; i >= 0; i--) {  int val = getBit(x, i);  if(tmp->child[1 - val] != NULL) {  tmp = tmp->child[1-val];  } else {  if(tmp->child[val] != NULL)  tmp = tmp->child[val];  }  }  return x ^ (tmp->value);  } | int const MAXN = 1e5 + 10;  int const NUM\_CHILDS = 2;  int trie[MAXN][NUM\_CHILDS];  int value[MAXN];  int n;  int index = 0;  int newTrie() {  index++;  return index;  }  void insert(int root, int x) {  int tmp = root;  for(int i = 30; i >= 0; i--) {  int val = getBit(x, i);  if(trie[tmp][val] == 0) {  trie[tmp][val] = newTrie();  }  tmp = trie[tmp][val];  }  value[tmp] = x;  }  int query(int root, int x) {  int tmp = root;  for(int i = 30; i >= 0; i--) {  int val = getBit(x, i);  if(trie[tmp][1-val] != 0) {  tmp = trie[tmp][1-val];  } else {  tmp = trie[tmp][val];  }  }  return x ^ value[tmp];  } |

**Suffix automation**/\*

A suffix automaton is a data structure to efficiently represent the suffixes of a string. It can be considered a compressed version of a suffix tree. The data structure supports querying for substrings within the text from with the automaton is constructed in linear

time. It also supports computation of the longest common substring in linear time.

Time Complexity: O(n \* ALPHABET\_SIZE) for construction, and O(n) for find\_all(), as well as longest\_common\_substring().

Space Complexity: O(n \* ALPHABET\_SIZE) auxiliary.

\*/

|  |  |
| --- | --- |
| #include<bits/stdc++.h>  using namespace std;  struct suffix\_automaton {  static const int ALPHABET\_SIZE = 26;  static int map\_alphabet(char c) {  return (int)(c - 'a');  }  struct state\_t {  int length, suffix\_link;  int firstpos, next[ALPHABET\_SIZE];  std::vector<int> invlinks;  state\_t() {  length = 0;  suffix\_link = 0;  firstpos = -1;  for (int i = 0; i < ALPHABET\_SIZE; i++)  next[i] = -1;  }  }; | std::vector<state\_t> states;  suffix\_automaton(const std::string & s) {  int n = s.size();  states.resize(std::max(2, 2 \* n - 1));  states[0].suffix\_link = -1;  int last = 0;  int size = 1;  for (int i = 0; i < n; i++) {  int c = map\_alphabet(s[i]);  int curr = size++;  states[curr].length = i + 1;  states[curr].firstpos = i;  int p = last;  while (p != -1 && states[p].next[c] == -1) {  states[p].next[c] = curr;  p = states[p].suffix\_link;  }  if (p == -1) {  states[curr].suffix\_link = 0;  } else {  int q = states[p].next[c];  if (states[p].length + 1 == states[q].length) {  states[curr].suffix\_link = q;  } else {  int clone = size++;  states[clone].length = states[p].length + 1;  for (int i = 0; i < ALPHABET\_SIZE; i++)  states[clone].next[i] = states[q].next[i];  states[clone].suffix\_link = states[q].suffix\_link;  while (p != -1 && states[p].next[c] == q) {  states[p].next[c] = clone;  p = states[p].suffix\_link;  }  states[q].suffix\_link = clone;  states[curr].suffix\_link = clone;  }  }  last = curr;  }  for (int i = 1; i < size; i++)  states[states[i].suffix\_link].invlinks.push\_back(i);  states.resize(size);  } |
| std::vector<int> find\_all(const std::string & s) {  std::vector<int> res;  int node = 0;  for (int i = 0; i < (int)s.size(); i++) {  int next = states[node].next[map\_alphabet(s[i])];  if (next == -1) return res;  node = next;  }  std::queue<int> q;  q.push(node);  while (!q.empty()) {  int curr = q.front();  q.pop();  if (states[curr].firstpos != -1)  res.push\_back(states[curr].firstpos - (int)s.size() + 1);  for (int j = 0; j < (int)states[curr].invlinks.size(); j++)  q.push(states[curr].invlinks[j]);  }  return res;  }  std::string longest\_common\_substring(const std::string & s) {  int len = 0, bestlen = 0, bestpos = -1;  for (int i = 0, cur = 0; i < (int)s.size(); i++) {  int c = map\_alphabet(s[i]);  if (states[cur].next[c] == -1) {  while (cur != -1 && states[cur].next[c] == -1)  cur = states[cur].suffix\_link;  if (cur == -1) {  cur = len = 0;  continue;  }  len = states[cur].length;  }  len++;  cur = states[cur].next[c];  if (bestlen < len) {  bestlen = len;  bestpos = i;  }  }  return s.substr(bestpos - bestlen + 1, bestlen);  }  }; | int main() {  {  suffix\_automaton sa("bananas");  vector<int> pos\_a, pos\_an, pos\_ana;  int ans\_a[] = {1, 3, 5};  int ans\_an[] = {1, 3};  int ans\_ana[] = {1, 3};  pos\_a = sa.find\_all("a");  for(int i = 0; i < pos\_a.size(); i++) {  cout << pos\_a[i] << " ";  }  cout << endl;  pos\_an = sa.find\_all("an");  pos\_ana = sa.find\_all("ana");  assert(equal(pos\_a.begin(), pos\_a.end(), ans\_a));  assert(equal(pos\_an.begin(), pos\_an.end(), ans\_an));  assert(equal(pos\_ana.begin(), pos\_ana.end(), ans\_ana));  }  {  suffix\_automaton sa("bbbabca");  assert(sa.longest\_common\_substring("aababcd")  == "babc");  }  return 0;  } |

**Suffix Array and LCP Array**

const int MAXN = 1e3 + 7;

string s;

int n, gap;

int sa[MAXN], pos[MAXN], tmp[MAXN], lcp[MAXN];

bool cmp(int i, int j) {

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

return pos[i] < pos[j];

i += gap;

j += gap;

return (i < n && j < n) ? pos[i] < pos[j] : i > j;

}

void buildSA() {

n = s.length();

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

sa[i] = i;

pos[i] = s[i];

}

for (gap = 1;; gap \*= 2) {

sort(sa, sa + n, cmp);

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

tmp[i + 1] = tmp[i] + cmp(sa[i], sa[i + 1]);

}

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

pos[sa[i]] = tmp[i];

}

if (tmp[n - 1] == n - 1) break;

}

}

void buildLCP() {

int k = 0;

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

if (pos[i] != n - 1) {

int j = sa[pos[i] + 1];

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

lcp[pos[i]] = k;

if (k)--k;

}

}

}

//number of dictinct strings

int ans = 0;

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

ans=ans+(n-sa[i])-lcp[i];

}

**MST**

|  |  |
| --- | --- |
| vector <pair<ll,ll> > a[MAXN];  vector <pair<ll,ll> >::iterator it;  ll m,n,ans;  pair<ll,ll> d[MAXN];  set <pair<ll,ll> > s;  bool Free[MAXN];  void input() {  scanf("%d %d",&n,&m);  FOR (i,1,m) {  LL u,v,c;  scanf("%d %d %d",&u,&v,&c);  a[u].push\_back(make\_pair(c,v));  a[v].push\_back(make\_pair(c,u));  }  }  void init() {  FOR (i,1,n) {  d[i].first=maxmax;  d[i].second=i;  }  d[1].first=0;  FOR (i,1,n) s.insert(d[i]);  }  void prim() {  ll u,v,dd;  while (!s.empty()) {  u=s.begin()->second;  dd=s.begin()->first;  Free[u]=1;  ans+=dd;  s.erase(s.begin());  for(it=a[u].begin();it!=a[u].end();it++) {  v=it->second; //canh ke voi u  dd=it->first; //do dai u-->v  if (!Free[v] && d[v].first>dd) {  s.erase(d[v]); //Xoa bo khoi set  d[v].first=dd; //Cap nhat lai d[v]  s.insert(d[v]);  }  }  }  } | struct edge{  int u,v;  long long w;  }a[100005];  bool cmp(edge s,edge t){  return s.w<t.w;  }  int find\_far(int x){  if(far[x]==x)return x;  return (far[x]=find\_far(far[x]));  }  void kruskal(){  long long w=0;  int d=0;  for(int i=1;i<=n;i++)far[i]=i;  for(int i=1;i<=m;i++){  if(find\_far(a[i].u)==find\_far(a[i].v))  continue;  w+=a[i].w;  far[find\_far(a[i].u)]=find\_far(a[i].v);  d++;  if(d==n-1)break;  }  cout<<w;  } |

**Vertex and Bridges**

vector<vector<int> >Adj(10001);

int n, m;

int Vertexs, Bridges;

bool Mark[10001]; int Num[10001], Low[100001], nc[100001];

int Count;

void DFS(int u){

++Count; Low[u] = n+1;

Num[u] = Count;

nc[u] = 0;

Mark[u] = false;

FO(i,0,Adj[u].size()){

int v = Adj[u][i];

FO(j,0,Adj[v].size())

if(Adj[v][j] == u){

Adj[v].erase(Adj[v].begin()+j);

break;

}

if(Num[v] == 0){

nc[u]++;

DFS(v);

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

++Bridges;

Mark[u] = Mark[u] | (Low[v] >= Num[u]);

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

}

else

Low[u] = min(Low[u], Num[v]);

}

}

int main(){

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

while(m--)

{

int u, v;

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

Adj[u].pb(v);

Adj[v].pb(u);

}

FOR(i,1,n)

if(Num[i] == 0){

DFS(i);

if(nc[i] < 2)

Mark[i] = false;

}

FOR(i,1,n)

if(Mark[i]) ++Vertexs;

printf("%d %d", Vertexs, Bridges);

}

**Hình học**

## **1. Các kiến thức cơ bản:**

int cmp(double a, double b){

if(abs(a-b) < eps) return 0;

return (a > b ? 1 : -1);

}

struct point{

double x;

double y;

point(double \_x = 0, double \_y = 0){

x = \_x; y = \_y;

}

bool operator == (const point& that) const{

return (cmp(x, that.x) == 0 && cmp(y, that.y) == 0);

}

bool operator < (const point& that) const{

if(cmp(x, that.x) != 0) return cmp(x, that.x) < 0;

return cmp(y, that.y) < 0;

}

};

int ccw(point p0, point p1, point p2){ /// vector p0p1, p0p2

double dx1 = p1.x - p0.x, dy1 = p1.y - p0.y;

double dx2 = p2.x - p0.x, dy2 = p2.y - p0.y;

return cmp(dx1\*dy2 - dx2\*dy1, 0);

///0 thẳng hàng 1 huong duong -1 huong am

}

### **// Hai đoạn thẳng thực sự cắt nhau**

int isRealCut(point p0, point p1, point p2, point p3){

return ccw(p0,p1,p2)\*ccw(p0,p1,p3)<0 && ccw(p2,p3,p0)\*ccw(p2,p3,p1)<0;

}

### **// Vị trí của p0 so với đoạn thẳng p1p2**

int segmentPos(point p0, point p1, point p2){

if(p1 == p2) return -2;

else if(ccw(p0,p1,p2) != 0) return -1; /// khong thang hang

else if (p2 < p1) swap(p1,p2);

if(p0 < p1) return 1; /// nam ngoai gan phia p1

else if(p2 < p0) return 2; /// nam ngoai gan phia p2

return 0; /// nam trong doan

}

### **// Hai đoạn thẳng cắt nhau**

int isSegmentCut(point p0, point p1, point p2, point p3){

if(isRealCut(p0,p1,p2,p3)) return 1;

if(segmentPos(p0,p2,p3) || segmentPos(p1,p2,p3)) return 1;

if(segmentPos(p2,p0,p1) || segmentPos(p3,p0,p1)) return 1;

return 0;

}

### **// Phương trình đường thẳng**

int getLine(point p0, point p1, double &a, double &b, double &c){

if(p0 == p1) return 0;

a = p1.y - p0.y;

b = p0.x - p1.x;

c = -(a\*p0.x + b\*p0.y);

return 1;

}

### **// Khoảng cách giữa hai điểm**

double dist(point p0, point p1){

double dx = p1.x - p0.x, dy = p1.y - p0.y;

return sqrt(dx \* dx + dy \* dy);

}

### **// Giao điểm của hai đoạn thẳng**

int getIntersection(point p0, point p1, point p2, point p3, point &p4){

double a0,b0,c0,a1,b1,c1;

getLine(p0,p1,a0,b0,c0);

getLine(p2,p3,a1,b1,c1);

double d = a0\*b1 - a1\*b0;

double dx = b0\*c1 - b1\*c0;

double dy = -a0\*b1 + a1\*c0;

if(cmp(d,0) == 0){

if(cmp(dx,0) == 0 && cmp(dy,0) == 0) return -1; /// trung nhau

else return 0; /// song song

}

p4.x = dx/d; p4.y = dy/d;

return 1;

}

### **// Diện tích đa giác**

double areaPolygon(point P[], int n){

P[n+1] = P[1];

double ans = 0;

FOR(i,1,n){

ans += P[i].x\*P[i+1].y - P[i+1].x\*P[i].y;

}

return ans/2;

}

### **// Kiểm tra một điểm nằm trong đa giác**

int insidePolygon(point P[], int n, point p0){

P[n+1] = P[1];

int x1 = 0, x2 = 0;

FOR(i,1,n){

if(ccw(P[i], P[i + 1], p0) == 0) return 0;

else if(ccw(P[i], P[i + 1], p0) == -1) x1++;

else x2++;

}

return (!x1 || !x2);

}

### **// Thuật toán bao lồi Graham Scan**

point O;

int degreeCmp( point p1, point p2){

int d = ccw(O,p1,p2);

if(d != 0) return (d > 0);

return cmp(dist(O, p1), dist(O, p2) < 0);

}

///Tim bao loi cua tap p[] dpt O(n log n)

/// Sap xep lai tap diem p[] va tra ve so diem thuoc bao loi, cap nhat lai n

void grahamScan(point P[], int &n){

int j = 1;

FOR(i,2,n){

if(cmp(P[j].y, P[i].y) > 0 || (cmp(P[j].y, P[i].y) == 0 && cmp(P[j].x, P[i].x) < 0))

j = i;

}

swap(P[1],P[j]);

///tim diem P[j] co hoanh do lon nhat trong nhung diem co tung do nho nhat

O = P[1];

sort(P+2, P+n+1,degreeCmp);

int k = 3;

FOR(i,3,n){

while(k > 2 && ccw(P[i], P[k-1], P[k-2]) >= 0)

k--;

swap(P[k++], P[i]);

}

n = k - 1;

}

### **// Kiểm tra hai đa giác có điểm chung**

int intersectionPolygon(point P[], int n1, point Q[], int n2){

P[n1+1] = P[1]; Q[n2+1] = Q[1];

/// check a point in a polygon

FOR(i,1,n1) if(insidePolygon(Q,n2,P[i])) return 1;

FOR(i,1,n2) if(insidePolygon(P,n1,Q[i])) return 1;

/// check line intersect

FOR(i,1,n1)

FOR(j,1,n2)

if(isRealCut(P[i],P[i+1],Q[j],Q[j+1]) == 1) return 1;

/// in case same point

FOR(i,1,n1)

FOR(j,1,n2){

if(isSegmentCut(P[i],P[i+1],Q[j],Q[j+1])) return 1;

}

return 0;

}

### **// Tìm giao của hai đa giác lồi**

void get2ConvexPolygon(point P[], int n1, point Q[], int n2, point ans[], int &dem){

dem = 1;

P[n1+1] = P[1]; Q[n2+1] = Q[1];

FOR(i,1,n1)

if(insidePolygon(Q,n2,P[i])) ans[dem++] = P[i];

FOR(i,1,n2)

if(insidePolygon(P,n1,Q[i])) ans[dem++] = Q[i];

FOR(i,1,n1){

FOR(j,1,n2){

point p3;

getIntersection(P[i],P[i+1],Q[i],Q[i+1],p3);

if(segmentPos(p3,P[i],P[i+1]) == 0)

if(segmentPos(p3,Q[j],Q[j+1]) == 0)

ans[dem++] = p3;

}

}

dem--;

grahamScan(ans,dem);

}

### **// Diện tích tam giác**

double areaTriangle(point A, point B, point C){

double ans = abs(A.x\*(B.y-C.y) + B.x\*(C.y-A.y) + C.x\*(A.y-B.y));

return ans/2;

}

### **// Tính góc BAC theo radian**

double getAngle(point A, point B, point C){

double a = dist(B,C);

double b = dist(C,A);

double c = dist(A,B);

double agoc = (b\*b + c\*c - a\*a) / (2\*b\*c);

return acos(agoc);

}

### **// Kiểm tra điểm p0 nằm trong tam giác ABC**

int insideTriangle(point p0, point A, point B, point C){

double S1 = areaTriangle(p0,A,B);

double S2 = areaTriangle(p0,B,C);

double S3 = areaTriangle(p0,C,A);

double Sum = areaTriangle(A,B,C);

if(cmp(Sum, S1+S2+S3) == 0) return 1;

return 0;

}

### **// Kiểm tra điểm p0 nằm trong góc tạo bởi tia AB, AC**

int insideAngle(point p0, point A, point B, point C){

if(p0 == A) return 1;

if(ccw(p0,A,B) \* ccw(p0,A,C) > 0) return 0;

//return (getAngle(A,p0,B) + getAngle(A,p0,C) < PI+eps);

if(cmp(getAngle(A,B,C), getAngle(A,p0,B) + getAngle(A,p0,C)) == 0) return 1;

return 0;

}

### **// Kiểm tra điểm p0 nằm trong đa giác O(log n)**

int insideConvexPolygonLogN(point P[], int n, point p0){

if(insideAngle(p0,P[1],P[2],P[n]) == 0) return 0;

int low = 2, high = n;

while(high - low > 1){

int mid = (low+high) / 2;

if(insideAngle(p0,P[1],P[low],P[mid]))

high = mid;

else low = mid;

}

return insideTriangle(p0,P[1],P[low],P[high]);

}

**2. Tâm đường tròn nội tiếp.**

xI = ( a.xA + b.xB + c.xC )/(a + b + c)

yI = ( a.yA + b.yB + c.yC )/(a + b + c)

## **3. Tâm đường tròn ngoại tiếp**

void timtam(point p1, point p2, point p3)

{

double a1,b1,c1,a2,b2,c2,dx,dy,D;

a1 = 2 \* (p1.x-p2.x);

b1 = 2 \* (p1.y-p2.y);

c1 = p2.x\*p2.x + p2.y\*p2.y - p1.x\*p1.x - p1.y\*p1.y; c1=-c1;

a2 = 2 \* (p1.x-p3.x);

b2 = 2 \* (p1.y-p3.y);

c2 = p3.x\*p3.x + p3.y\*p3.y - p1.x\*p1.x - p1.y\*p1.y; c2=-c2;

D = a1\*b2 - a2\*b1;

Dx = c1\*b2 - c2\*b1;

Dy = a1\*c2 - a2\*c1;

xO = dx/D;

yO = dy/D;

//cout << a1 << " " << b1 << " " << c1<<endl;

//cout << a2 << " " << b2 << " " << c2<<endl;

//cout << dx << " " << dy << " " << D<<endl;

cout << xO << endl << yO << endl;

}

## **3. Khoảng cách 1 điểm tới đường thẳng.**

**Các hằng số:**

E=2.718281828

Thư viện Cmath

Sin, cos, tan, asin, acos, atan, atan2(góc giữa 2 véc tơ)

Log: ln

Log10: lg

**Phép quay góc a:**

X = x cos a – y sin a

Y = x sin a + y cos a

**Tọa độ cầu:**

X = r cos phi sin theta

Y = r sin phi sin theta

Z = r cos theta

### **// Tính tổ hợp chập k của m**

LL calC(int m, int k){

if(c[m][k] != 0) return c[m][k];

if(k == 0 || k == m) return c[m][k] = 1;

return c[m][k] = calC(m - 1, k) + calC(m-1, k-1);

}

### **// Số các số từ 0 -> (a-1) có đúng k bit 1**

LL bit(LL a, int k){

if(k < 0) return 0;

int m = getBit(a);

if(m < k) return 0LL;

return calC(m, k) + f(a & ((1<<m)-1), k-1);

}

### **// Số các số từ a -> b có đúng k bit 1:** f(b+1, k) - f(a, k)

## **4. Đồ thị:**

### **// Dijkstra:**

LL dist[maxn]; /// chi phi min khi den diem i

LL par[maxn]; /// danh dau duong di de backtrack

vector<PII> adj[maxn]; ///chi phi giua 2 dinh

void dijkstra(int ii, int jj) {

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

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

dist[ii] = 0;

priority\_queue<PII> q;

q.push(mp(0, ii));

while(!q.empty()) {

int top = q.top().se;

LL dis = -q.top().fi;

q.pop();

if(dis == dist[top]) {

for(int i = 0; i < adj[top].size(); i++) {

int v = adj[top][i].fi;

LL w = adj[top][i].se;

if(dist[v] == -1 || dist[v] > dis + w) {

dist[v] = dis + w;

q.push(mp(-dist[v], v));

par[v] = top;

}

}

}

}

}

### **// Chu trình Euler**

stack := (1); *//Ngăn xếp ban đầu chỉ chứa một đỉnh bất kỳ, chẳng hạn đỉnh 1*

repeat

u := Get; *//Đọc phần tử ở đỉnh ngăn xếp*

if $(u, v) ∈ E then *//Từ u còn đi tiếp được*

begin

Push(v);

E := E – {(u, v)}; *//Xóa cạnh (u, v) khỏi đồ thị*

end;

else *//Từ u không đi đâu được nữa*

begin

u := Pop; *//Lấy u khỏi ngăn xếp*

Output ¬ u; *//In ra u*

end;

until stack = Æ; *//Lặp tới khi ngăn xếp rỗng*