Contents

1 text 1
- 1.1 Mathematics . . . 1
- 1.2 Graph Theory . . . 1

2 code 1
- 2.1 2SAT . . . 1
- 2.2 Hopcroft-Karp . . . 1
- 2.3 BCC . . . 2
- 2.4 ConvexHull3D . . . 2
- 2.5 EulerTour . . . 2
- 2.6 DepthArray . . . 2
- 2.7 DMST . . . 3
- 2.8 Match . . . 3
- 2.9 FareyNumber . . . 4
- 2.10 GCD . . . 4
- 2.11 Gusfield . . . 4
- 2.12 Kuhn-Munkres . . . 4
- 2.13 KMP . . . 5
- 2.14 Stoer-Wagner . . . 5
- 2.15 MEC . . . 5
- 2.16 SAP . . . 6
- 2.17 SCC . . . 6
- 2.18 SquareRoot . . . 6
- 2.19 ConvexHull2D . . . 7
- 2.20 SuffixArray . . . 7
- 2.21 Aho-Corasick . . . 7

1 text

1.1 Mathematics

```
V+F=E+2
C(n,m) % p = C(n/p,m/p) * C(n%p,m%p) % p
A=i+b/2-1
```

1.2 Graph Theory

```
Erdos-Gallai Theorem
--------------------
non-negative sequence d: d1 >= d2 >= ... >= dn
sum of di is even
Sigma(i=1,k,di) <= k*(k-1) + Sigma(j=k+1,n,min(k,dj))
     for k=1,2,...,n

Bipartite Graph
---------------
Min Vertex Cover = Max Match
Min Vertex Cover + Max Independent Set = |V|
Max Independent Set = Min Edge Cover

LU Flow
-------
G has solition iff maxflow(G') = sum of min capacity of G

Menger
------
K(x,y) = Lambda(x,y) if x and y is not connected directly.

Ore
---
G has H-cycle iff G+uv has H-cycle if n>=3 and deg(u)+deg(v)>=n
```

2 code

2.1 2SAT

```
// 2-SAT, find a solution
//
// V = 2 * n
int sel[VMAX];
list<int> E_[VMAX];
list<int> _E[VMAX];
int ideg[VMAX];
deque<int> Q;

inline void block(int u) {
  int v; list<int>::iterator it;
  if(sel[u]==2)return;
  sel[u]=2;
  stack[sp++]=u;
  while(sp) {
    u=stack[--sp];
    for(it=E_[u].begin();it!=E_[u].end();++it) {
      if(sel[v=*it]==2)continue;
      sel[v]=2;
      stack[sp++]=v;
    }
  }
}

bool SAT_2() {
  int i,u,v,a,b; list<int>::iterator it;
  SCC();
  for(i=0;i<n;i++)if(scc[i<<1]==scc[(i<<1)|1])return
      false;
  memset(sel,0,sizeof(int)*V);
  memset(ideg,0,sizeof(int)*V);
  for(u=0;u<V;u++) {
    for(it=E[u].begin();it!=E[u].end();++it) {
      v=*it;
      if(scc[v]==scc[u])continue;
      E_[scc[v]].push_back(scc[u]);
      ideg[scc[u]]++;
    }
  }
  for(i=0;i<n;i++) {
    a=scc[i<<1];
    b=scc[(i<<1)|1];
    _E[a].push_back(b);
    _E[b].push_back(a);
  }
  for(u=0;u<V;u++)if(scc[u]==u&&ideg[u]==0)Q.
      push_back(u);
  while(!Q.empty()) {
    u=Q.front();
    Q.pop_front();
    sel[u]=1;
    for(it=_E[u].begin();it!=_E[u].end();++it)block(*
        it);
    for(it=E_[u].begin();it!=E_[u].end();++it) {
      v=*it;
      if(--ideg[v]==0&&sel[v]==0)Q.push_back(v);
    }
  }
  for(u=0;u<V;u++) {
    E_[u].clear();
    _E[u].clear();
  }
  return true;
}
```

2.2 Hopcroft-Karp

```
// V = A + B
int Sa,pair_a[AMAX]; bool flag_a[AMAX];
int Sb,pair_b[BMAX]; bool flag_b[BMAX];

bool HK_DFS_a(int x) {
  flag_a[x]=true;
  foreach(int y: E[x]) {
    if(y!=pair_a[x]&&!flag_b[y]&&HK_DFS_b(y)) {
      pair_a[x]=y;
      pair_b[y]=x;
      return true;
    }
  }
  return false;
}

bool HK_DFS_b(int y) {
```

```
  flag_b[y]=true;
  if(pair_b[y]==-1)return true;
  return !flag_a[pair_b[y]]&&HK_DFS_a(pair_b[y]);
}


bool HK_working(int& ans) {
  bool flag=false;
  memset(flag_a,false,sizeof(bool)*Sa);
  memset(flag_b,false,sizeof(bool)*Sb);
  for(int i=0;i<Sa;i++) {
    if(pair_a[i]==-1&&!flag_a[i]&&HK_DFS_a(i)) {
      ans++;
      flag=true;
    }
  }
  return flag;
}

int HK() {
  int ans=0;
  memset(pair_a,-1,sizeof(int)*Sa);
  memset(pair_b,-1,sizeof(int)*Sb);
  while(HK_working(ans));
  return ans;
}
```

2.3 BCC

```
// Must calculate dfn,low,p before using this
void BCC() {
  int u,pd,pe;
  for(u=0;u<V;u++) {
    if(p[u]==-1)continue;
    pd=dfn[p[u]];
    pe=ID of edge (u,p[u]);
    foreach(edge e: E[u]) {
      if(p[e.v]==u) {
        if(low[v]<=pd) {
          join(pe,e.id);
        }
      } else if(e.id!=pe&&dfn[e.v]<dfn[u]) {
        join(pe,e.id);
      }
    }
  }
}
```

2.4 ConvexHull3D

```
#include<stdio.h>
#include<math.h>
#define SIZE 100
struct Point{
    long long x,y,z;
    Point operator-(Point b)const{return (Point){x-b.
        x,y-b.y,z-b.z};}
    Point operator*(Point b)const{return (Point){y*b.
        z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x};}
}a[SIZE];
int used[SIZE][SIZE][SIZE],t,dn;
struct Face{
    int x,y,z,r;
}d[10000];
long long an;
long long Abs(long long x){return x<0?-x:x;}
long long dot(Point X,Point Y){return X.x*Y.x+X.y*Y.y
    +X.z*Y.z;}
void add_answer(int x,int y,int z,int w){
    Point X,Y,Z;
    X=a[y]-a[x];
    Y=a[z]-a[x];
    Z=a[w]-a[x];
    an+=Abs(X.x*Y.y*Z.z+X.y*Y.z*Z.x+X.z*Y.x*Z.y-X.x*Y
        .z*Z.y-X.y*Y.x*Z.z-X.z*Y.y*Z.x);
}
int NP(Point X,Point Y,Point Z,Point W){
    long long tmp=dot((Y-X)*(Z-X),W-X);
    if(tmp>0)return 1;
    return -1;
}
void First(){
    add_answer(0,1,2,3);
    used[0][1][2]=used[0][1][3]=used[0][2][3]=used
        [1][2][3]=1;
    d[0]=(Face){0,1,2,NP(a[0],a[1],a[2],a[3])};
    d[1]=(Face){0,1,3,NP(a[0],a[1],a[3],a[2])};
    d[2]=(Face){0,2,3,NP(a[0],a[2],a[3],a[1])};
    d[3]=(Face){1,2,3,NP(a[1],a[2],a[3],a[0])};
    dn=4;
}
void delete_face(int x){
    int i;
    for(i=x;i<dn-1;i++)d[i]=d[i+1];
    dn--;
}
void add_point(int x){
    int i,j,n=dn;
    for(i=0;i<n;i++){
        if(NP(a[d[i].x],a[d[i].y],a[d[i].z],a[x])*d[i
            ].r==-1){
            add_answer(d[i].x,d[i].y,d[i].z,x);
            d[dn++]=(Face){d[i].x,d[i].y,x,NP(a[d[i].
                x],a[d[i].y],a[x],a[d[i].z])};
            d[dn++]=(Face){d[i].x,d[i].z,x,NP(a[d[i].
                x],a[d[i].z],a[x],a[d[i].y])};
            d[dn++]=(Face){d[i].y,d[i].z,x,NP(a[d[i].
                y],a[d[i].z],a[x],a[d[i].x])};
            delete_face(i);
            i--;
            n--;
        }
    }
    for(;i<dn;i++){
        for(j=i+1;j<dn;j++){
            if(d[i].x==d[j].x&&d[i].y==d[j].y&&d[i].z
                ==d[j].z){
                delete_face(j);
                delete_face(i);
                i--;
                break;
            }
        }
    }
}
void Final(int N){
    int i;
    for(i=4;i<N;i++)add_point(i);
}
main(){
    int N,i;
    while(scanf("%d",&N)&&N){
        an=0;
        t++;
        for(i=0;i<N;i++)scanf("%lld␣%lld␣%lld",&a[i].
            x,&a[i].y,&a[i].z);
        First();
        Final(N);
        printf("%.2lf\n",an/6.);
    }
}
```

2.5 EulerTour

```
Stack stack;

void euler(int u) {
  while(!E[u].empty())euler(E[u].pop());
  stack.push(u);
}
```

2.6 DepthArray

```
void DepthArray(const int s[],const int sa[],int n,
    int da[]) {
  int* ref=new int[n];
```

```
  int i,j,k=0;
  for(i=0;i<n;i++)ref[sa[i]]=i;
  for(i=0;i<n;i++) {
    if(ref[i]==0){da[0]=(k=0);continue;}
    j=sa[ref[i]-1];
    if(k)k--;
    while(i+k<n&&j+k<n&&s[i+k]==s[j+k])k++;
    da[ref[i]]=k;
  }
  delete[] ref;
}
```

2.7 DMST

```
/*
 * Directed Minimum Spanning Tree
 *
 * input:
 *   n = |V|
 *   cost[][] = weight
 * output:
 *   solve() = minimum cost
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;

#define NMAX 1000
#define INF 2147483647
int n,cost[NMAX][NMAX];
bool alive[NMAX];
int back[NMAX];
int cycle[NMAX],cs;

void select(int u) {
  int v,ov,oc=INF;
  for(v=0;v<n;v++)if(alive[v]&&cost[v][u]<oc){ov=v;oc
      =cost[v][u];}
  back[u]=ov;
}

int shrink() {
  static bool inc[NMAX];
  int u,v,i,j,res=0;
  memset(inc,false,n);
  for(i=0;i<cs;i++)inc[cycle[i]]=true;
  u=cycle[0];i=u;do res+=cost[back[i]][i];while((i=
      back[i])!=u);
  for(j=0;j<n;j++) {
    if(!alive[j]||inc[j])continue;
    if(cost[j][u]!=INF)cost[j][u]-=cost[back[u]][u];
  }
  for(i=1;i<cs;i++) {
    alive[v=cycle[i]]=false;
    for(j=0;j<n;j++) {
      if(!alive[j]||inc[j])continue;
      cost[u][j]=min(cost[u][j],cost[v][j]);
      if(cost[j][v]!=INF)cost[j][u]=min(cost[j][u],
          cost[j][v]-cost[back[v]][v]);
    }
  }
  for(i=1;i<n;i++)if(inc[back[i]])back[i]=u;
  select(u);
  return res;
}

bool find_cycle() {
  static char mark[NMAX];
  int u,v;
  memset(mark,0,n);
  mark[0]=1;
  for(u=1;u<n;u++) {
    if(!alive[u]||mark[u])continue;
    for(v=u;!mark[v];v=back[v])mark[v]=2;
    if(mark[v]==2)break;
    for(v=u;mark[v]==2;v=back[v])mark[v]=1;
  }
  if(u==n)return false;
  for(v=u;mark[v]==2;v=back[v])mark[v]=3;
  cs=0;u=v;do cycle[cs++]=v;while((v=back[v])!=u);
  return true;
}

bool impossible() {
  static int stack[NMAX];
  int sp=0,u,v,cnt=0;
  memset(alive,false,n);
  stack[sp++]=0;
  alive[0]=true;
  while(sp) {
    u=stack[--sp];
    cnt++;
    for(v=0;v<n;v++) {
      if(cost[u][v]!=INF&&!alive[v]) {
        stack[sp++]=v;
        alive[v]=true;
      }
    }
  }
  return cnt!=n;
}

int solve() {
  if(n<=1)return 0;
  if(impossible())return INF;
  int ans=0,i;
  for(i=1;i<n;i++)select(i);
  while(find_cycle())ans+=shrink();
  for(i=1;i<n;i++)if(alive[i])ans+=cost[back[i]][i];
  return ans;
}
```

2.8 Match

```
#include <stdio.h>

#define EVEN(x) (mu[x]==x || (mu[x]!=x && phi[mu[x
    ]]!=mu[x]))
#define ODD(x) (mu[x]!=x && phi[mu[x]]==mu[x] && phi[
    x]!=x)
#define OUTER(x) (mu[x]!=x && phi[mu[x]]==mu[x] &&
    phi[x]==x)

int a[256][256], na[256];
int n;
int mu[256], phi[256], rho[256], scanned[256];
int dx[256], dy[256];

int maximum_matching(void)
{
  for(int i=1;i<=n;i++)
    mu[i] = phi[i] = rho[i] = i, scanned[i] = false;
  for(int x=-1; ;){
    if(x<0){
      for(x=1;x<=n && (scanned[x] || !EVEN(x));++x);
      if(x>n) break;
    }
    int y=-1;
    for(int i=0;i<na[x];i++)
      if(OUTER(a[x][i]) || (EVEN(a[x][i]) && rho[a[x
          ][i]]!=rho[x]))
        y = a[x][i];
    if(y==-1){scanned[x] = true; x = -1;}
    else if(OUTER(y)) phi[y] = x;
    else{
      for(int i=1;i<=n;i++)
        dx[i] = dy[i] = -2;
      for(int k=0, w=x; dx[w]<0; w = k%2? mu[w]:phi[w
          ]) dx[w] = k++;
      for(int k=0, w=y; dy[w]<0; w = k%2? mu[w]:phi[w
          ]) dy[w] = k++;
      bool vertex_disjoint = true;
      for(int v=1;v<=n;v++)
        if(dx[v]>=0 && dy[v]>=0) vertex_disjoint =
            false;
      if(vertex_disjoint){
        for(int v=1;v<=n;v++)
```

```
        if(dx[v]%2) mu[phi[v]] = v, mu[v] = phi[v];
      for(int v=1;v<=n;v++)
        if(dy[v]%2) mu[phi[v]] = v, mu[v] = phi[v];
      mu[x] = y; mu[y] = x; x=-1;
      for(int v=1;v<=n;v++)
        phi[v] = rho[v] = v, scanned[v] = false;
    }else{
      int r = x, d = n, dd = n;
      for(int v=1;v<=n;v++)
        if(dx[v] >=0 && dy[v]>=0 && rho[v]==v && d>
            dx[v])
          d = dx[v], r = v;
      for(int v=1;v<=n;v++)
        if(dy[v] >=0 && dx[v]>=0 && rho[v]==v && dd
            >dy[v])
          dd = dy[v];
      for(int v=1;v<=n;v++)
        if(dx[v]<=d && dx[v]%2 && rho[phi[v]]!=r)
            phi[phi[v]] = v;
      for(int v=1;v<=n;v++)
        if(dy[v]<=dd && dy[v]%2 && rho[phi[v]]!=r)
            phi[phi[v]] = v;
      if(rho[x]!=r) phi[x] = y;
      if(rho[y]!=r) phi[y] = x;
      for(int v=1;v<=n;v++)
        if(dx[rho[v]]<=d && dx[rho[v]]>=0 || dy[rho
            [v]]<=dd && dy[rho[v]]>=0) rho[v] = r;
    }
  }
 }
 int cnt=0;
 for(int v=1;v<=n;v++)
   if(v<mu[v]) ++cnt;
 return cnt;
}

int main(void)
{
 int x, y;
 scanf("%d",&n);
 while(scanf("%d%d",&x,&y)!=EOF)
 {
   a[x][na[x]++] = y;
   a[y][na[y]++] = x;
 }
 x = maximum_matching();
 printf("%d\n",x*2);
 for(x=1;x<=n;x++)
   if(x<mu[x]) printf("%d %d\n",x, mu[x]);
 return 0;
}
```

2.9 FareyNumber

```
#include<stdio.h>
/* a/b< q/p < c/d */
void Go(long long a,long long b,long long c,long long
     d){
    long long p1=1,q1=0,p2=0,q2=1,p,q,k,mp,mq;
    while(1){
        p=p1+p2;
        q=q1+q2;
        if(a*p<b*q&&q*d<c*p)break;
        if(a*p>=b*q){
            k=(a*p1-b*q1)/(b*q2-a*p2);
            mp=p1+k*p2;
            mq=q1+k*q2;
            p1=mp,q1=mq;
        }
        else{
            k=(d*q2-c*p2)/(c*p1-d*q1);
            mp=k*p1+p2;
            mq=k*q1+q2;
            p2=mp,q2=mq;
        }
    }
    printf("%lld/%lld\n",q,p);
}
main(){
    long long a,b,c,d;
    while(scanf("%lld %lld %lld %lld",&a,&b,&c,&d)
        ==4)
        Go(a,b,c,d);
}
```

2.10 GCD

```
// a*x+b*y=z
struct gcd_t {int x,y,z;};

gcd_t gcd(int a,int b) {
  if(b==0)return (gcd_t){1,0,a};
  gcd_t t=gcd(b,a%b);
  return (gcd_t){t.y,t.x-t.y*(a/b),t.z};
}
```

2.11 Gusfield

```
int G[NMAX][NMAX];//the original network
int f[NMAX][NMAX];

int p[NMAX];//the star graph
int a[NMAX][NMAX];//the maxflow between all pairs

void Gusfield() {
  static bool near[NMAX];
  static int stack[NMAX]; int sp;
  int i,j,u,v,w;
  memset(p,0x00,sizeof(int)*n);
  p[0]=-1;
  for(i=0;i<n;i++) {
    for(j=0;j<n;j++)a[i][j]=INF;
    a[i][i]=0;
  }
  for(i=1;i<n;i++) {
    for(j=0;j<n;j++)memcpy(f[j],G[j],sizeof(int)*n);
    SAP(i,p[i]);
    w=maxflow;
    memset(near,false,n);
    sp=0;
    stack[sp++]=i;
    near[i]=true;
    while(sp) {
      u=stack[--sp];
      for(v=0;v<n;v++) {
        if(f[u][v]&&!near[v]) {
          stack[sp++]=v;
          near[v]=true;
        }
      }
    }
    for(j=i+1;j<n;j++)if(near[j]&&p[j]==p[i])p[j]=i;
    a[i][p[i]]=w;
    a[p[i]][i]=w;
    for(j=0;j<i;j++) {
      if(j==p[i])continue;
      a[i][j]=min(w,a[p[i]][j]);
      a[j][i]=a[i][j];
    }
  }
}
```

2.12 Kuhn-Munkres

```
// returns maximum cost
//
// requirement: w all non-negative
int n,w[NMAX][NMAX];
int cx[NMAX];
int cy[NMAX];
int back[NMAX];
bool did[NMAX*2];

bool Hungarian(int u) {
  did[u]=true;
  for(int v=0;v<n;v++) {
    if(cx[u]+cy[v]!=w[u][v])continue;
```

```
    if(back[v]==-1||(!did[back[v]]&&Hungarian(back[v
        ]))) {
      back[v]=u;
      return true;
    }
  }
  return false;
}

int Hungarian() {
  int ans=0,i;
  memset(back,0xff,sizeof(int)*n);
  for(i=0;i<n;i++) {
    memset(did,0,n);
    if(Hungarian(i))ans++;
  }
  return ans;
}

void VertexCover() {
  static int stack[NMAX*2]; int sp=0;
  int i,j,u,v;
  memset(did,0,n*2);
  for(i=0;i<n;i++) {
    for(j=0;j<n;j++)if(back[j]==i)break;
    if(j==n){stack[sp++]=i;did[i]=true;}
  }
  while(sp) {
    u=stack[--sp];
    if(u<n) {
      for(v=0;v<n;v++) {
        if(cx[u]+cy[v]!=w[u][v])continue;
        if(!did[n+v]){stack[sp++]=n+v;did[n+v]=true;}
      }
    } else {
      v=back[u-n];
      if(v!=-1){stack[sp++]=v;did[v]=true;}
    }
  }
}

int Kuhn_Munkres() {
  int i,j,k;
  for(i=0;i<n;i++) {
    cx[i]=w[i][0];
    for(j=1;j<n;j++)cx[i]=max(cx[i],w[i][j]);
  }
  memset(cy,0,sizeof(int)*n);
  while(true) {
    if(Hungarian()==n)break;
    VertexCover();
    k=0x7fffffff;
    for(i=0;i<n;i++) {
      if(!did[i])continue;
      for(j=0;j<n;j++) {
        if(did[n+j])continue;
        k=min(k,cx[i]+cy[j]-w[i][j]);
      }
    }
    for(i=0;i<n;i++)if(did[i])cx[i]-=k;
    for(j=0;j<n;j++)if(did[n+j])cy[j]+=k;
  }
  k=0;
  for(i=0;i<n;i++)k+=cx[i]+cy[i];
  return k;
}
```

2.13 KMP

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
#define SIZE 999
using namespace std;
int kmp_next[SIZE];
void kmp_create(char s[]){
    int i,j;
    kmp_next[0]=-1;
    for(i=1,j=0;s[i];i++,j++){
        kmp_next[i]=j;
        while(j>=0&&s[i]!=s[j])j=kmp_next[j];
    }
    kmp_next[i]=j;
}
int kmp_search(char s2[],char s1[]){
    int i,j,an=0;
    for(i=j=0;s2[i];i++,j++){
        an=max(an,j);
        while(j>=0&&s2[i]!=s1[j])j=kmp_next[j];
    }
    an=max(an,j);
    return an;
}
main(){
    char s1[SIZE],s2[SIZE];
    while(scanf("%s␣%s",s1,s2)==2){
        kmp_create(s1);
        printf("%d\n",kmp_search(s2,s1));
    }
}
```

2.14 Stoer-Wagner

```
// Stoer-Wagner
//
// Input: G=(V,E,W)
void merge(vertex a,vertex b) {
  foreach(vertex v: V) {
    if(v!=a&&v!=b) {
      W[a][v]+=W[b][v];
      W[v][a]+=W[v][b];
    }
  }
  V=V-{b};
}

weight SW() {
  weight answer=INFINITY;
  vertex source,sink;
  while(|V|>=2) {
    answer=min(answer,MAS(&source,&sink));
    merge(source,sink);
  }
  return answer;
}
// Maximum Adjacency Search
//
// Input: G=(V,E,W)
weight W(set S,vertex v) {
  weight result=0;
  foreach(vertex u: S)result+=W[u][v];
  return result;
}

weight MAS(vertex* source,vertex* sink) {
  set S={0};
  while(S!=V) {
    vertex temp=-1;
    foreach(vertex v: V-S)if(temp==-1||W(S,v)>W(S,
        temp))temp=v;
    S=S+{temp};
  }
  *source=S[n-2];
  *sink  =S[n-1];
  S=S-{*sink};
  return W(S,*sink);
}
```

2.15 MEC

```
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;

struct Point {
  double x,y;
```

```
  Point operator-(Point p) {return (Point){x-p.x,y-p.
      y};}
};

double sqr(double x) {return x*x;}
double dist(const Point& a,const Point& b) {return
    sqr(a.x-b.x)+sqr(a.y-b.y);}
double cross(Point a,Point b) {return a.x*b.y-a.y*b.x
    ;}
double dot(Point a,Point b) {return a.x*b.x+a.y*b.y;}
double abs2(Point p) {return p.x*p.x+p.y*p.y;}

struct Circle {
  Point C; double r2;

  Circle() {C.x=0;C.y=0;r2=-1;}
  Circle(Point p1) {C=p1;r2=0;}

  Circle(Point p1,Point p2) {
    C.x=(p1.x+p2.x)/2;
    C.y=(p1.y+p2.y)/2;
    r2=dist(C,p1);
  }

  Circle(Point p1,Point p2,Point p3) {
    double a,b,c,z;
    z=2*sqr(cross(p2-p1,p3-p1));
    a=abs2(p2-p3)*dot(p1-p2,p1-p3)/z;
    b=abs2(p1-p3)*dot(p2-p1,p2-p3)/z;
    c=abs2(p1-p2)*dot(p3-p1,p3-p2)/z;
    C.x=a*p1.x+b*p2.x+c*p3.x;
    C.y=a*p1.y+b*p2.y+c*p3.y;
    r2=abs2(p1-p2)*abs2(p1-p3)*abs2(p2-p3)/(2*z);
  }

  bool contain(Point p) {return dist(C,p)<=r2+1e-8;}
};

Circle findMEC(int n,const Point* p,int m,Point* b) {
  Circle mec;
  if(m==1)mec=Circle(b[0]);
  if(m==2)mec=Circle(b[0],b[1]);
  if(m==3)return Circle(b[0],b[1],b[2]);
  for(int i=0;i<n;i++)if(!mec.contain(p[i])) {
    b[m]=p[i];
    mec=findMEC(i,p,m+1,b);
  }
  return mec;
}

int main() {
  static Point p[100000];
  static Point b[3];
  int n,i; Circle mec;
  while(scanf("%d",&n)==1) {
    if(n==0)break;
    for(i=0;i<n;i++)scanf("%lf%lf",&p[i].x,&p[i].y);
    random_shuffle(p,p+n);
    mec=findMEC(n,p,0,b);
    printf("%.8lf\n",sqrt(mec.r2));
  }
  return 0;
}
```

2.16 SAP

```
int ans,del; bool flag;
int h[NMAX];
int g[NMAX];

void dfs(int u) {
  if(u==sink){ans+=del;flag=true;return;}
  int minh=n-1,temp=del;
  foreach(int v: E[u]) {
    if(h[v]+1==h[u]) {
      del=min(del,f[u][v]);
      dfs(v);
      if(h[source]==n)return;
      if(flag)break;
      del=temp;
    }
    minh=min(minh,h[v]);
  }
  if(flag){f[u][v]-=del;f[v][u]+=del;return;}
  if(--g[h[u]]==0)h[source]=n;
  h[u]=minh+1;
  if(h[u]!=n)g[h[u]]++;
}

void SAP() {
  ans=0;
  memset(h,0,sizeof(int)*n);
  memset(g,0,sizeof(int)*n);
  g[0]=n;
  while(h[source]!=n) {
    del=INFINITY;
    flag=false;
    dfs(source);
  }
}
```

2.17 SCC

```
int visited;
int dfn[VMAX];
int low[VMAX];
Stack stack;
bool inS[VMAX];

void dfs(int u) {
  dfn[u]=visited++;
  low[u]=dfn[u];
  stack.push(u);
  inS[u]=true;
  foreach(int v: E[u]) {
    if(dfn[v]==-1) {
      dfs(v);
      low[u]=min(low[u],low[v]);
    } else if(inS[v]) {
      low[u]=min(low[u],dfn[v]);
    }
  }
  if(low[u]==dfn[u]) {
    do {
      inS[v=stack.pop()]=false;
      // v is with u
    } while(v!=u);
  }
}

void SCC() {
  visited=0;
  memset(dfn,-1,sizeof(int)*V);
  for(int i=0;i<V;i++)if(dfn[i]==-1)dfs(i);
}
```

2.18 SquareRoot

```
#include<stdio.h>
#include<string.h>
#define SIZE 110
void square_root(char s[],int &n,int output[]){
    int a[SIZE],i,j,k,l,b[SIZE],c[SIZE],d[SIZE];
    memset(a,0,sizeof(a));
    memset(c,0,sizeof(c));
    memset(d,0,sizeof(d));
    memset(b,0,sizeof(b));
    for(l=0;s[l]!=0;l++);
    if(l%2==0)
        for(i=1;i<=l;i++)a[i]=s[i-1]-'0';
    else
        for(i=2,a[0]=0,l++;i<=l;i++)a[i]=s[i-2]-'0';
    for(i=1;i*2<=l;i++){
        for(j=1,c[0]=0;j<i;j++)c[j]=b[j]*2;
        for(j=i-1;j>0;j--){
            c[j-1]+=c[j]/10;
            c[j]%=10;
```

```
        }
        for(j=9;j>0;j--){
            c[i]=j;
            for(k=d[0]=0;k<=i;k++)d[k+1]=c[k]*j;
            for(k=i+1;k>0;k--){
                d[k-1]+=d[k]/10;
                d[k]%=10;
            }
            for(k=i+1;k>=0;k--)
                if(d[i+1-k]!=a[2*i-k])break;
            if(d[i+1-k]<a[2*i-k]||k<0)break;
        }
        b[i]=j;
        if(j!=0){
            for(k=i+1;k>=0;k--){
                a[2*i-k]-=d[i+1-k];
                j=0;
                while(a[2*i-k-j]<0){
                    a[2*i-k-j]+=10;
                    j++;
                    a[2*i-k-j]--;
                }
            }
        }
    }
    if(b[0]!=0){
        n=l/2+1;
        b[0]=1;
        for(i=0;i<n;i++)output[i]=b[l/2-i];
    }
    else{
        n=l/2;
        for(i=0;i<n;i++)output[i]=b[l/2-i];
    }
}
```

2.19 ConvexHull2D

```
#include<stdio.h>
#include<algorithm>
#include<math.h>
using namespace std;
/*below need fixed*/
#define SIZE 2400
#define MAX 1e12
#define Err 1e-9
typedef double MYTYPE;
/*above need fixed*/
typedef double MYTYPE;
struct Point{
    MYTYPE x,y;
    bool operator<(Point b)const{
        if(fabs(x*b.y-y*b.x)<Err)return x*x+y*y<=b.x*
            b.x+b.y*b.y;
        return x*b.y>y*b.x;
    }
    MYTYPE operator*(Point b)const{return y*b.x-x*b.y
        ;}
    Point operator-(Point b)const{return (Point){x-b.
        x,y-b.y};}
    Point operator+(Point b)const{return (Point){x+b.
        x,y+b.y};}
}a[SIZE];
void Convex_Hull(int &n){
    int i,j,min_pos;
    MYTYPE min_x,min_y;
    Point tmp;
    min_x=min_y=MAX;
    for(i=0;i<n;i++)
        if(min_x>a[i].x||(min_x==a[i].x&&min_y>a[i].y
            ))min_x=a[i].x,min_y=a[i].y,min_pos=i;
    swap(a[0],a[min_pos]);
    tmp=a[0];
    for(i=0;i<n;i++)a[i]=(a[i]-tmp);
    sort(a+1,a+n);
    for(i=2,j=1;i<n;i++){
        while((a[i]-a[j])*(a[j]-a[j-1])<=0&&j>0)j--;
        a[++j]=a[i];
    }
    n=j+1;
    for(i=0;i<n;i++)a[i]=(a[i]+tmp);
}
```

2.20 SuffixArray

```
inline bool _cmp(const int r[],int a,int b,int c) {
  return r[a]==r[b]&&r[a+c]==r[b+c];
}

void SuffixArray(const int src[],int n,int Z,int sa
    []) {
  int *c=new int[max(n,Z)],*x=new int[n],*y=new int[n
      ];
  int i,j,k;
  memset(c,0,sizeof(int)*max(n,Z));
  for(i=0;i<n;i++)c[x[i]=src[i]]++;
  for(i=1;i<Z;i++)c[i]+=c[i-1];
  for(i=n-1;i>=0;i--)sa[--c[x[i]]]=i;
  for(j=k=1;k<n;j<<=1,Z=k) {
    k=0;
    for(i=n-j;i<n;i++)y[k++]=i;
    for(i=0;i<n;i++)if(sa[i]>=j)y[k++]=sa[i]-j;
    memset(c,0,sizeof(int)*Z);
    for(i=0;i<n;i++)c[x[y[i]]]++;
    for(i=1;i<Z;i++)c[i]+=c[i-1];
    for(i=n-1;i>=0;i--)sa[--c[x[y[i]]]]=y[i];
    swap(x,y);
    x[sa[0]]=0;
    for(i=k=1;i<n;i++)x[sa[i]]=_cmp(y,sa[i-1],sa[i],j
        )?k-1:k++;
  }
  delete[] c;
  delete[] x;
  delete[] y;
}
```

2.21 Aho-Corasick

```
const int ZMAX=26;

struct ACT {
  int alpha;//the last alphabet
  ACT* parent;
  ACT* back;
  ACT* next[ZMAX];
};

void build(ACT* root,int Z) {
  deque<ACT*> Q;
  ACT* u; ACT* ptr;
  int alpha,i;
  Q.push_back(root);
  while(!Q.empty()) {
    u=Q.front();
    Q.pop_front();
    alpha=u->alpha;
    if(u!=root&&u->parent!=root) {
      ptr=u->parent->back;
      while(!ptr->next[alpha]&&ptr!=root)ptr=ptr->
          back;
      u->back=ptr->next[alpha];
      if(!u->back)u->back=root;
    } else {
      u->back=root;
    }
    for(i=0;i<Z;i++)if(u->next[i])Q.push_back(u->next
        [i]);
  }
}
```