Codebook

1 Basic

1.1 vimrc

```
set nocompatible
filetype plugin indent on
set t_Co=256
set term=screen-256color
set number
set tabstop=4
set shiftwidth=4
set softtabstop=4
set expandtab
set wrap
set showcmd
colorscheme darkblue
map  <F2> :w <CR> :call OP() <CR>
map! <F2> <ESC> :w <CR> :call OP() <CR> <ESC>
map  <F9> :w <CR> :call CP_R() <CR> <ESC>
map! <F9> <ESC> :w <CR> :call CP_R() <CR> <ESC>
map  <HOME> ^
map! <HOME> <ESC>^i
map  <ESC>OH <HOME>
map! <ESC>OH <HOME>
map  <END> $
map  <ESC>OF <END>
map! <ESC>OF <ESC><END>a
function CP_R()

  if( &ft == 'cpp')
    let cpl = 'g++ -w -o "%:r.exe" -std=c++11 "%"' |
        let exc = '"./%:r.exe"'
  elseif( &ft == 'python')
    let exc = 'python "%"'
  endif
  let pause = 'printf "Press any key to continue..." &&
      read -n 1 && exit'
  if !exists('exc')
    echo 'Can''t compile this filetype...'
    return
  endif
  if exists('cpl')
    let cp_r = cpl . ' && time ' . exc
  else
    let cp_r = 'time ' . exc
  endif
    execute '! clear && ' . cp_r . ' && ' . pause
endfunction

function OP()
  execute '!$COLORTERM -x gedit ' . "%" . ";"
endfunction
```

2 Number

2.1 Extended GCD

```
ll ext_gcd(ll a,ll b,ll &x,ll &y){
    ll d=a;
    if(b!=0ll){
        d=ext_gcd(b,a%b,y,x);
        y-=(a/b)*x;
    }
    else x=1ll,y=0ll;
    return d;
}
```

2.2 Modular Inverse

```
/*
 * find the inverse of n modular p
 */
ll mod_inverse(ll n, ll p){
    ll x, y;
    ll d = ext_gcd(n, p, x, y);
    return (p+x%p) % p;
}
```

2.3 Line Modular Equation

```
/*
 * ax = b (mod n)
 * return a set of answer(vector<ll>)
 */
vector<ll> line_mod_equation(ll a,ll b,ll n){
    ll x, y, d;
    d = ext_gcd(a, n, x, y);
    vector<ll> ans;
    if(b%d==0ll){
        x = (x%n + n) % n;
        ans.push_back((x*(b/d))%(n/d));
        for(ll i=1;i<d;i++)
            ans.push_back((ans[0]+i*n/d)%n);
    }
    return ans;
}
```

2.4 Chinese Remainder Theorem

```
/*
 * solve the chinese remainder theorem(CRT)
 * if a.size() != m.size(), return -1
 * return the minimun positive answer of CRT
 * x = a[i] (mod m[i])
 */
int CRT(vector<int> a, vector<int> m) {
    if(a.size() != m.size()) return -1;
    int M = 1;
    for(int i=0;i<(int)m.size();i++)
        M *= m[i];
    int res = 0;
    for(int i=0;i<(int)a.size();i++)
        res = (res + (M/m[i])*mod_inverse(M/m[i], m[i])
            *a[i]) % M;
    return (res + M) % M;
}
```

2.5 C(N,M)

```
/* P is the modular number */
#define P 24851
int fact[P+1];
/* called by Cmod */
int mod_fact(int n,int &e){
    e = 0;
    if(n == 0) return 1;
    int res = mod_fact(n/P, e);
    e += n / P;
    if((n/P)%2 != 0) return res*(P-fact[n%P])%P;
    return res*fact[n%P]%P;
}
/*
 * return C(n, m) mod P
 */
int Cmod(int n,int m){
    /* this section only need to be done once */
    fact[0] = 1;
    for(int i=1;i<=P;i++){
        fact[i] = fact[i-1] * i%P;
    }
    /* end */
    int a1, a2, a3, e1, e2, e3;
    a1 = mod_fact(n, e1);
    a2 = mod_fact(m, e2);
    a3 = mod_fact(n-m, e3);
    if(e1 > e2 + e3)return 0;
    return a1 * mod_inverse(a2 * (a3%P), P) % P;
}
```

2.6 Phi

```
/*
 * gen phi from 1~MAXN
 * store answer in phi
 */
#define MAXN 100
int mindiv[MAXN], phi[MAXN];
void genphi(){
    for(int i=1;i<MAXN;i++)
        mindiv[i] = i;
    for(int i=2;i*i<MAXN;i++)
        if(mindiv[i] == i)
            for(int j=i*i;j<MAXN;j+=i)
                mindiv[j] = i;
    phi[1] = 1;
    for(int i=2;i<MAXN;i++){
        phi[i] = phi[i/mindiv[i]];
        if((i/mindiv[i])%mindiv[i] == 0)
            phi[i] *= mindiv[i];
        else phi[i] *= (mindiv[i]-1);
    }
}
```

2.7 Miller Rabin

```
ll pow_mod(ll x, ll N, ll M) {
    ll res = 1;
    x %= M;
    while(N){
        if(N&1ll) res = mul_mod(res, x, M);
        x = mul_mod(x, x, M);
        N >>= 1;
    }
    return res;
}
bool PrimeTest(ll n, ll a, ll d) {
    if(n == 2 || n == a) return true;
    if((n&1) == 0) return false;
    while((d&1) == 0) d >>= 1;
    ll t = pow_mod(a, d, n);
    while((d!=n-1) && (t!=1) && (t!=n-1)){
        t = mul_mod(t, t, n);
        d <<= 1;
    }
    return (t==n-1) || ((d&1)==1);
}
bool MillerRabin(ll n){
    // test set
    vector<ll> a = {2, 325, 9375, 28178,  450775,
        9780504, 1795265022};
    for(int i=0;i<(int)a.size();i++)
        if(!PrimeTest(n, a[i], n-1)) return false;
    return true;
}
```

2.8 FFT

```
/*
```

```
    vector<Complex> res(a);
    for (int i=1,j=0;i<(int)res.size();i++){
        for(int k=((int)res.size())>>1;!((j^=k)&k);k>>=1);
        if(i > j) swap(res[i], res[j]);
    }
    return res;
}
/*
 * calculate the FFT of sequence
 * a.size() must be 2^k
 * flag = 1  -> FFT(a)
 * falg = -1 -> FFT-1(a)
 * return FFT(a) or FFT-1(a)
 */
vector<Complex> FFT(vector<Complex> a, int flag=1){
    vector<Complex> res = reverse(a);
    for(int k=2;k<=(int)res.size();k<<=1){
        double p0 = -pi / (k>>1) * flag;
        Complex unit_p0(cos(p0), sin(p0));
        for(int j=0;j<(int)res.size();j+=k){
            Complex unit(1.0, 0.0);
            for(int i=j;i<j+k/2;i++,unit*=unit_p0){
                Complex t1 = res[i], t2 = res[i+k/2] * unit;
                res[i] = t1 + t2;
                res[i+k/2] = t1 - t2;
            }
        }
    }
    return res;
}
```

2.9 Function

```
/*
 * class of polynomial function
 * coef is the coefficient
 * f(x) = sigma(c[i]*x^i)
 */
class Function {
public:
    vector<double> coef;
    Function(const vector<double> c=vector<double>()): coef(c){}
    double operator () (const double &rhs) const {
        double res = 0.0;
        double e = 1.0;
        for(int i=0;i<(int)coef.size();i++,e*=rhs)
            res += e * coef[i];
        return res;
    }
    Function derivative() const {
        vector<double> dc((int)this->coef.size()-1);
        for(int i=0;i<(int)dc.size();i++)
            dc[i] = coef[i+1] * (i+1);
        return Function(dc);
    }
    int degree() const {
        return (int)coef.size()-1;
    }
};
/*
 * calculate the integration of f(x) from a to b
 * divided into n piece
 * the bigger the n is, the more accurate the answer is
 */
template<class T>
double simpson(const T &f, double a, double b){
    double c = (a+b) / 2.0;
    return (f(a)+4.0*f(c)+f(b)) * (b-a) / 6.0;
}
template<class T>
double simpson(const T &f, double a, double b, double eps, double A){
    double c = (a+b) / 2.0;
```

```
template<class T>
double simpson(const T &f, double a, double b, double eps){
    return simpson(f, a, b, eps, simpson(f, a, b));
}
```

2.10 Equation

```
/*
 * called by find
 * 1 = positive, -1 = negative, 0 = zero
 */
int sign(double x){
    return x < -EPS ? -1 : x > EPS;
}
/* called by equation */
template<class T>
double find(const T &f, double lo, double hi){
    int sign_lo, sign_hi;
    if((sign_lo=sign(f(lo))) == 0) return lo;
    if((sign_hi=sign(f(hi))) == 0) return hi;
    if(sign_hi * sign_lo > 0) return INF;
    while(hi-lo>EPS){
        double m = (hi+lo) / 2;
        int sign_mid = sign(f(m));
        if(sign_mid == 0) return m;
        if(sign_lo * sign_mid < 0)
            hi = m;
        else lo = m;
    }
    return (lo+hi) / 2;
}
/*
 * return a set of answer of f(x) = 0
 */
template<class T>
vector<double> equation(const T &f){
    vector<double> res;
    if(f.degree() == 1){
        if(sign(f.coef[1]))res.push_back(-f.coef[0]/f.coef[1]);
        return res;
    }
    vector<double> droot = equation(f.derivative());
    droot.insert(droot.begin(), -INF);
    droot.push_back(INF);
    for(int i=0;i<(int)droot.size()-1;i++){
        double tmp = find(f, droot[i], droot[i+1]);
        if(tmp < INF) res.push_back(tmp);
    }
    return res;
}
```

2.11 Permutation

```
/*
 * return the sequence of x-th of n!
 * max(n) = 12
 * 0 of 3! -> 123
 * 5 of 3! -> 321
 */
int factorial[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800, 479001600};
vector<int> idx2permutation(int x, int n){
    vector<bool> used(n+1, false);
    vector<int> res(n);
    for(int i=0;i<n;i++){
        int tmp = x / factorial[n-i-1];
        int j;
        for(j=1;j<=n;j++)if(!used[j]){
            if(tmp == 0) break;
            tmp--;
        }
        res[i] = j, used[j] = true;
        x %= factorial[n-i-1];
```

```
 * 123 of 3! -> 0
 * 321 of 3! -> 5
 */
int permutation2idx(vector<int> a){
    int res = 0;
    for(int i=0;i<(int)a.size();i++){
        int tmp = a[i] - 1;
        for(int j=0;j<i;j++)
            if(a[j] < a[i]) tmp--;
        res += factorial[(int)a.size()-i-1] * tmp;
    }
    return res;
}
```

3 Matrix

3.1 Guass Elimination

```
/*
 * return guass eliminated matrix
 * r will be chenged to the number of the non-free
     variables
 * l[i] will be set to true if i-th variable is not
     free
 * ignore flag
 */
Matrix GuassElimination(int &r, vector<bool> &l, int
    flag=0) {
    l = vector<bool>(C);
    r = 0;
    Matrix res(*this);
    for(int i=0;i<res.C-flag;i++){
        for(int j=r;j<res.R;j++){
            if(fabs(res.at(j, i)) > EPS){
                swap(res.D[r], res.D[j]);
                break;
            }
        }
        if(fabs(res.at(r, i)) < EPS){
            continue;
        }
        for(int j=0;j<res.R;j++){
            if(j != r && fabs(res.at(j, i)) > EPS){
                double tmp = (double)res.at(j, i) / (
                    double)res.at(r, i);
                for(int k=0;k<res.C;k++){
                    res.at(j, k) -= tmp * res.at(r, k);
                }
            }
        }
        r++;
        l[i] = true;
    }
    return res;
}
```

3.2 Solve Matrix (Ax=B)

```
/*
 * Ax = b
 * it will return the answer(x)
 * if row != column or there is any free variable, it
     will return an empty vector
 */
vector<double> Solve(vector<double> a) {
    if(R != C) return vector<double>();
    vector<double> res(R);
    Matrix t(R, C+1);
    for(int i=0;i<R;i++){
        for(int j=0;j<C;j++)
            t.at(i, j) = at(i, j);
        t.at(i, C) = a[i];
    }
    int r = 0;
    vector<bool> l;
    t = t.GuassElimination(r, l, 1);
    if(r != R) return vector<double>();
    for(int i=0;i<C;i++){
        if(l[i])for(int j=0;j<R;j++){
            if(fabs(t.at(j, i)) > EPS)
                res[i] = t.at(j, C) / t.at(j, i);
        }
    }
    return res;
}
```

3.3 Inverse Matrix

```
/*
```

```
    Matrix t(R, R*2);
    for(int i=0;i<R;i++){
        for(int j=0;j<C;j++)
            t.at(i, j) = at(i, j);
        t.at(i, i+R) = 1;
    }
    int r = 0;
    vector<bool> l;
    t = t.GuassElimination(r, l, R);
    if(r != R)return Matrix();
    for(int i=0;i<C;i++){
        if(l[i])for(int j=0;j<R;j++){
            if(fabs(t.at(j, i)) > EPS){
                for(int k=0;k<C;k++)
                    t.at(j, C+k) /= t.at(j, i);
            }
        }
    }
    Matrix res(R, C);
    for(int i=0;i<R;i++)
        for(int j=0;j<C;j++)
            res.at(i, j) = t.at(i, j+C);
    return res;
}
```

4 Graph

4.1 Bridge And Cut

```
/* called by cut_bridge */
void _cut_bridge(int x, int f, int d){
    vis[x] = 1;
    dfn[x] = low[x] = d;
    int children = 0;
    for(int i=0;i<(int)vc[x].size();i++){
        Edge e = vc[x][i];
        if(e.to != f && vis[e.to] == 1)
            low[x] = min(low[x], dfn[e.to]);
        if(vis[e.to] == 0){
            _cut_bridge(e.to, x, d+1);
            children++;
            low[x] = min(low[x], low[e.to]);
            if((f == -1 && children > 1) || (f != -1 &&
                 low[e.to] >= dfn[x]))
                cut[x] = true;
            if(low[e.to] > dfn[x])
                bridge[x][e.to] = bridge[e.to][x] =
                    true;
        }
    }
}
/*
 * solve the cut and bridge
 * store answer in cut(vector<bool>) ans bridge(vector<
    vector<bool> >)
 * cut[i] == true iff i-th node is cut
 * bridge[i][j] == true iff edge between i-th ans j-th
    is bridge
 */
void cut_bridge(){
    vis = vector<int>(N+1, 0);
    dfn = low = vector<int>(N+1);
    cut = vector<bool>(N+1);
    bridge = vector<vector<bool> >(N+1, vector<bool>(N
        +1, false));
    for(int i=0;i<N;i++){
        if(!vis[i])
            _cut_bridge(i, -1, 0);
    }
}
```

4.2 BCC

```
/* called by BCC */
void _BBC(int x, int d){
    stk[++top] = x;
    dfn[x] = low[x] = d;
    for(int i=0;i<(int)vc[x].size();i++){
        Edge e = vc[x][i];
        if(dfn[e.to] == -1){
            _BBC(e.to, d+1);
            if(low[e.to] >= dfn[x]){
                vector<int> l;
                do{
                    l.push_back(stk[top]);
                    top--;
                }while(stk[top+1] != e.to);
                l.push_back(x);
                bcc.push_back(l);
            }
            low[x] = min(low[x], low[e.to]);
        }else low[x] = min(low[x], dfn[e.to]);
    }
}
/*
 * solve the biconnected components(BCC)
 * store answer in bcc(vector<vector<int> >)
 * bbc.size() is the number of BCC
 * bcc[i] is the sequence of a BCC
 */
void BCC(){
```

```
        if(dfn[i] == -1)
            _BBC(i, 0);
}
```

4.3 SCC

```
/* called by SCC */
void _SCC(int x, int d){
    stk[++top] = x;
    dfn[x] = low[x] = d;
    vis[x] = 1;
    for(int i=0;i<(int)vc[x].size();i++){
        Edge e = vc[x][i];
        if(dfn[e.to] != -1){
            if(vis[e.to] == 1)
                low[x] = min(low[x], dfn[e.to]);
        }else{
            _SCC(e.to, d+1);
            low[x] = min(low[x], low[e.to]);
        }
    }
    if(low[x] == dfn[x]){
        while(stk[top] != x){
            scc[stk[top]] = scc_cnt;
            vis[stk[top]] = 2;
            top--;
        }
        scc[stk[top]] = scc_cnt++;
        vis[stk[top]] = 2;
        top--;
    }
}
/*
 * solve the strongly connected component(SCC)
 * store answer in scc(vector<int>)
 * the value of scc[i] means the id of the SCC which i-th node in (id is based 0)
 * scc_cnt id the number of SCC
 */
void SCC(){
    dfn = low = vector<int>(N+1, -1);
    vis = vector<int>(N+1, 0);
    scc = vector<int>(N+1, 0);
    scc_cnt = 0;
    stk = vector<int>(N+1, -1);
    top = -1;
    for(int i=0;i<N;i++)
        if(dfn[i] == -1)
            _SCC(i, 0);
}
```

4.4 Two Sat

```
/*
 * called by TwoSat
 * get the value of i-th
 * 1 = true, 0 = false, -1 = undefined
 */
int TwoSatGet(int x){
    int r = x > N/2 ? x-N/2 : x;
    if(twosatans[r] == -1)
        return -1;
    return x > N/2 ? !twosatans[r] : twosatans[r];
}
/*
 * solve the 2SAT
 * return true if there exists a set of answer
 * store the answer in twosatans
 */
bool TwoSat(){
    SCC();
    twosatans = vector<int>(N/2+1, -1);
    for(int i=0;i<N/2;i++)
        if(scc[i] == scc[i+N/2])
            return false;
    vector<vector<int> > c(scc_cnt+1);
            int x = c[i][j];
            if(TwoSatGet(x) == 0)
                val = 0;
            for(int k=0;k<(int)vc[x].size();k++)
                if(TwoSatGet(vc[x][k].to) == 0)
                    val = 0;
            if(!val)
                break;
        }
        for(int j=0;j<(int)c[i].size();j++){
            if(c[i][j] > N/2)
                twosatans[c[i][j]-N/2] = !val;
            else
                twosatans[c[i][j]] = val;
        }
    }
    return true;
}
```

5 Path

5.1 Kth Shortest

```
int KthShortestPath(int s, int t, int k){
    Graph RG(N);
    for(int i=0;i<N;i++)
        for(int j=0;j<(int)vc[i].size();j++){
            Edge e = vc[i][j];
            RG.add_edge(e.to, Edge(i, e.w));
        }
    RG.AllDijkstra(t);
    dis = RG.dis;
    priority_queue<PI> pq;
    pq.push(PI(-dis[s], s));
    while(!pq.empty()){
        PI v = pq.top();
        pq.pop();
        int real = -v.FF - dis[v.SS];
        if(v.SS == t && (!(--k)))
            return real;
        for(int i=0;i<(int)vc[v.SS].size();i++){
            Edge e = vc[v.SS][i];
            pq.push(PI(-(real+e.w+dis[e.to]), e.to));
        }
    }
    return -1;
}
```

5.2 EulerCircuit

```
#define eid w
void _EulerCircuit(int x){
    for(int i=0;i<(int)vc[x].size();i++){
        Edge e = vc[x][i];
        if(vis[e.eid]) continue;
        vis[e.eid] = 1;
        _EulerCircuit(e.to);
        eulercircuit.push_back(e.eid);
    }
}
bool EulerCircuit(){ // undirected
    if(!Connected()) return false;
    vis = vector<int>(M+1, 0);
    for(int i=0;i<N;i++){
        if(vc[i].size()&1)
            return false;
        //sort
        sort(vc[i].begin(), vc[i].end());
    }
    eulercircuit.clear();
    _EulerCircuit(0);
    //sort
    reverse(eulercircuit.begin(), eulercircuit.end());
    return true;
}
```

6 Flow

6.1 Dinic

```
/*
 * Maximum Flow Dinic
 * Solve() returns answer
 */
class Dinic{
public:
    class Edge{
    public:
        int v1, v2, f, c;
        Edge(int _v1=0, int _v2=0, int _f=0, int _c=0):
             v1(_v1), v2(_v2), f(_f), c(_c){}
    };
    int N;
    vector<vector<int> >vc;
    vector<Edge> E;
    vector<int> dep;

    Dinic(int n=0): N(n), vc(vector<vector<int> >(N+1))
        , dep(vector<int>(N+1)) {}
    void add_edge(int a, int b, int c){
        vc[a].push_back(E.size());
        E.push_back(Edge(a, b, c, c));
        vc[b].push_back(E.size());
        E.push_back(Edge(b, a, 0, c));
    }
    int Bfs(int s, int t){
        fill(dep.begin(), dep.end(), -1);
        dep[s] = 0;
        queue<int> q;
        q.push(s);
        while(!q.empty()){
            int v = q.front(); q.pop();
            for(int i=0;i<(int)vc[v].size();i++){
                Edge e = E[vc[v][i]];
                if(e.f > 0 && dep[e.v2] == -1){
                    dep[e.v2] = dep[v] + 1;
                    q.push(e.v2);
                }
            }
        }
        return dep[t];
    }
    int Dfs(int x, int df, int t){
        if(x == t) return df;
        int res = 0;
        for(int i=0;i<(int)vc[x].size();i++){
            Edge &e = E[vc[x][i]];
            if(e.f > 0 && dep[e.v2] == dep[x] + 1){
                int f = Dfs(e.v2, min(df, e.f), t);
                e.f -= f;
                E[vc[x][i]^1].f += f;
                df -= f;
                res += f;
            }
        }
        return res;
    }
    int Solve(int s, int t){
        int flow = 0;
        while(Bfs(s, t) != -1){
            flow += Dfs(s, 0x3f3f3f3f, t);
        }
        return flow;
    }
};
```

6.2 StoerWanger

```
/*
 * Stoer Wanger
 * Undirected Min Cut
 * Solve() returns answer if graph is connected else 0
```

```
    vector<int> bln, dis;
    StoerWanger(int n=0): N(n), G(vector<vector<int> >(N, vector<int>(N))), bln(vector<int>(N, -1)), dis(vector<int>(N)) {}
    void add_edge(int a, int b, int c){
        G[a][b] += c;
        G[b][a] += c;
    }
    int Mst(int r, int &x, int &y){
        int t;
        bln[t=0] = r;
        for(int i=0;i<wN;i++)
            if(bln[i] != r)
                dis[i] = G[0][i];
        for(int k=0;k<wN-1;k++){
            x = t; t = 0;
            for(int i=0;i<wN;i++)
                if(bln[i] != r && (!t || dis[i] > dis[t]))
                    t = i;
            bln[t] = r;
            for(int i=0;i<wN;i++)
                if(bln[i] != r)
                    dis[i] += G[t][i];
        }
        y = t;
        return dis[t];
    }
    void Merge(int x, int y){
        if(x > y) swap(x, y);
        for(int i=0;i<wN;i++)
            if(i != x && i != y)
                G[i][x] += G[i][y], G[x][i] += G[y][i];
        if(y == wN-1) return;
        for(int i=0;i<wN-1;i++)
            if(i != y)
                swap(G[i][y], G[i][wN-1]), swap(G[y][i], G[wN-1][i]);
    }
    int Solve(){
        wN = N;
        int res = 0x3f3f3f3f;
        for(int i=0;wN>1;i++, wN--){
            int x, y;
            res = min(res, Mst(i, x, y));
            Merge(x, y);
        }
        return res;
    }
};
```

6.3 Mixed Euler

```
/*
 * Mixed Euler
 * Solve() returns if there is a euler circuit or not
 */
class MEuler{
public:
    class Edge{
#define DIRECTED    1
#define UNDIRECTED  0
    public:
        int to, dir;
        Edge(int t=0, int d=0): to(t), dir(d){}
    };
    int N;
    Dinic dinic;
    vector<int> deg;
    vector<vector<Edge> > vc;

    MEuler(int n=0): N(n), dinic(Dinic(N+2)), deg(vector<int>(N, 0)), vc(vector<vector<Edge> >(N)) {}
    void add_edge(int a, int b, int d){
        vc[a].push_back(Edge(b, d));
        deg[a]++, deg[b]--;
        if(d == UNDIRECTED)
            vc[b].push_back(Edge(a, d));
    }
    bool Solve(){
        for(int i=0;i<(int)vc.size();i++){
            for(int j=0;j<(int)vc[i].size();j++){
                Edge e = vc[i][j];
                if(e.dir == UNDIRECTED)
                    dinic.add_edge(i, e.to, 1);
            }
        int ans = 0;
        for(int i=0;i<N;i++){
            if(deg[i] > 0){
                dinic.add_edge(N, i, deg[i]/2);
            }else if(deg[i] < 0){
                dinic.add_edge(i, N+1, -deg[i]/2);
                ans += -deg[i] / 2;
            }
        }
        if(dinic.Solve(N, N+1) < ans) return false;
        return true;
    }
};
```

7 Match

7.1 BiMatch

```cpp
/*
 * BIpartite Matching
 * Nx = number of x nodes
 * Ny = number of y nodes
 * store matching answer in mx, my
 * Solve() returns the number of matching
 */
class BiMatch{
public:
    int Nx, Ny;
    vector<vector<int> > vc;
    vector<int> mx, my;
    vector<int> visy;

    BiMatch(int _x=0, int _y=0): Nx(_x), Ny(_y), vc(
        vector<vector<int> >(Nx+1)){}

    void add(int x, int y){
        vc[x].push_back(y);
    }

    bool Match(int x){
        for(int i=0;i<(int)vc[x].size();i++){
            int y = vc[x][i];
            if(!visy[y]){
                visy[y] = 1;
                if(my[y] == -1 || Match(my[y])){
                    mx[x] = y, my[y] = x;
                    return true;
                }
            }
        }
        return false;
    }
    int Solve(){
        mx = vector<int>(Nx+1, -1);
        my = vector<int>(Ny+1, -1);
        int ans = 0;
        for(int i=0;i<Nx;i++){
            visy = vector<int>(Ny+1, 0);
            ans += Match(i);
        }
        return ans;
    }
};
```

7.2 KM

```cpp
/*
 * solve Maximun Bipartite Matching
 * store matching answer in mx ,my
 * Solve() returns themaximun weight of perfect
     matching
 */
class KM{
public:
#define FF first
#define SS second
    typedef pair<int, int> PI;
    const static int INF = 0x3f3f3f3f;
    int Nx, Ny;
    vector<vector<int> >mp;
    vector<int> visx, visy;
    vector<int> lx, ly, slack;
    vector<int> mx, my;
    KM(int x=0, int y=0): Nx(x), Ny(y), mp(vector<
        vector<int> >(Nx+1, vector<int>(Ny+1, 0))) {}
    void add(int x, int y, int w){
        mp[x][y] = w;
    }

    bool Match(int x){
        visx[x] = 1;
        for(int y=0;y<Ny;y++){
            if(visy[y]) continue;
            if(lx[x] + ly[y] > mp[x][y]){
                slack[y] = min(slack[y], lx[x] + ly[y]
                    - mp[x][y]);
            }else{
                visy[y] = 1;
                if(my[y] == -1 || Match(my[y])){
                    mx[x] = y, my[y] = x;
                    return true;
                }
            }
        }
        return false;
    }

    int Solve(){
        mx = vector<int>(Nx+1, -1);
        my = vector<int>(Ny+1, -1);
        lx = vector<int>(Nx+1, -INF);
        ly = vector<int>(Ny+1, 0);
        for(int i=0;i<Nx;i++)
            for(int j=0;j<Ny;j++)
                lx[i] = max(lx[i], mp[i][j]);
        for(int i=0;i<Nx;i++){
            slack = vector<int>(Ny+1, INF);
            while(true){
                visx = vector<int>(Nx+1, 0);
                visy = vector<int>(Ny+1, 0);
                if(Match(i)) break;
                int d = INF;
                for(int j=0;j<Ny;j++)
                    if(!visy[j]) d = min(d, slack[j]);
                if(d == INF)break;
                for(int i=0;i<Nx;i++)
                    if(visx[i]) lx[i] -= d;
                for(int i=0;i<Ny;i++)
                    if(visy[i]) ly[i] += d;
                    else slack[i] -= d;
            }
        }
        int res = 0;
        for(int i=0;i<Nx;i++)
            if(mx[i] != -1)
                res += mp[i][mx[i]];
        return res;
    }
};
```

7.3 General Match

```cpp
/*
 * Maximun General Graph Matching
 * store answer in m
 * Solve() returns the number of matching
 * important!!!
 * notice the order of disjoint set when unioning
 */
class GMatch{
public:
    int N;
    vector<vector<int> > vc;
    DisjointSet djs;
    vector<int> m, d, c1, c2, p, vis;
    queue<int> q;
    int ts;
    GMatch(int n): N(n), vc(vector<vector<int> >(N+1)),
         djs(DisjointSet(N)), ts(0){}

    void add(int a, int b){
        vc[a].push_back(b);
        vc[b].push_back(a);
    }

    void path(int x, int r){
        if(x==r)return;
        if(d[x] == 0){
            int i = p[x], j = p[p[x]];
            path(j, r);
```

```
            m[i] = j, m[j] = i;
        }
    }

    void blossom(int x, int y, int bi){
        for(int i=djs.find(x);i!=bi;i=djs.find(p[i])){
            djs.U(bi, i);
            if(d[i] == 1)
                c1[i] = x, c2[i] = y, q.push(i);
        }
    }

    int lca(int x,int y,int r){
        ts++;
        vis[r] = ts;
        for(int i=djs.find(x);i!=r;i=djs.find(p[i]))
            vis[i] = ts;
        int b;
        for(b=djs.find(y);vis[b]!=ts;b=djs.find(p[b]));
        return b;
    }

    bool Match(int x){
        djs.init();
        d = vector<int>(N+1, -1);
        d[x] = 0;
        q = queue<int>();
        q.push(x);
        while(!q.empty()){
            int u = q.front(); q.pop();
            for(int i=0;i<(int)vc[u].size();i++){
                int v = vc[u][i];
                if(m[v] != v && djs.find(u) != djs.find
                    (v)){
                    if(d[v] == -1){
                        if(m[v] == -1){
                            path(u, x);
                            m[u] = v, m[v] = u;
                            return true;
                        }else{
                            p[v] = u, p[m[v]] = v;
                            d[v] = 1, d[m[v]] = 0;
                            q.push(m[v]);
                        }
                    }else{
                        if(d[djs.find(v)] == 0){
                            int bi=lca(u, v, x);
                            blossom(u, v, bi);
                            blossom(v, u, bi);
                        }
                    }
                }
            }
        }
        return false;
    }

    int Solve(){
        m = c1 = c2 = d = p = vis = vector<int>(N+1,
            -1);
        int ans = 0;
        for(int i=0;i<N;i++){
            if(m[i] == -1){
                if(Match(i)) ans++;
                else m[i]=i;
            }
        }
        return ans;
    }
};
```

8 MST

8.1 Restricted Minimal Spanning Tree

```
/*
 * Restricted MST
 * r = the node is limited
 * k = the limit
 * notice: <=k or ==k
 * Solve() returns value of rmst if there ia an answer
     else -1
 */
class RMST{
public:
#define to  first.first
#define eid first.second
#define v1  first.first
#define v2  first.second
#define w   second
    const static int INF = 0x3f3f3f3f;
    typedef pair<int, int> PI;
    typedef pair<PI, int> PII;
    int N;
    vector<vector<PII> > vc;
    vector<PII> E;
    DisjointSet djs;
    vector<bool> choose;
    vector<int> best;
    vector<PI> adj;
    RMST(int n=0): N(n), vc(vector<vector<PII> >(N+1)),
         djs(DisjointSet(N)) {}
    void add_edge(int a, int b, int w){
        E.push_back(PII(PI(a, b), w));
    }
    static bool cmp(PII a, PII b){
        return a.w < b.w;
    }
    void dfs(int x, int p, int r){
        for(int i=0;i<(int)vc[x].size();i++){
            PII e = vc[x][i];
            if(choose[e.eid] && e.to != p){
                if(x == r){
                    best[e.to] = -1;
                }else{
                    if(best[x] == -1 || E[best[x]].w <
                        e.w){
                        best[e.to] = e.eid;
                    }else{
                        best[e.to] = best[x];
                    }
                }
                dfs(e.to, x, r);
            }
        }
    }
    int Solve(int r, int k){
        choose = vector<bool>((int)E.size()+1, false);
        best = vector<int>(N+1, -1);
        adj = vector<PI>(N+1, PI(INF, -1));
        sort(E.begin(), E.end(), RMST::cmp);
        int rmst = 0, m = 0;
        for(int i=0;i<(int)E.size();i++){
            PII e = E[i];
            vc[e.v1].push_back(PII(PI(e.v2, i), e.w));
            vc[e.v2].push_back(PII(PI(e.v1, i), e.w));
            if(e.v1 != r && e.v2 != r && djs.find(e.v1)
                 != djs.find(e.v2)){
                choose[i] = true;
                djs.U(e.v1, e.v2);
                rmst += e.w;
            }
        }
        for(int i=0;i<(int)E.size();i++){
            PII e = E[i];
            if(e.v1 == r || e.v2 == r){
```

```
                    djs.U(r, v);
                }
            }
        }
        if(m > k) return -1;
        for(int j=m+1;j<=k;j++){
            fill(best.begin(), best.end(), -1);
            dfs(r, r, r);
            int chid = -1;
            int chmin = INF;
            int vid = -1;
            for(int i=0;i<N;i++){
                if(i != r && adj[i].first != INF &&
                    best[i] != -1){
                    if(chmin > adj[i].first - E[best[i
                        ]].w){
                        chmin = adj[i].first - E[best[i
                            ]].w;
                        chid = adj[i].second;
                        vid = i;
                    }
                }
            }
            /* if ==k
            if(chid == -1) return -1;
            */
            /* if <=k */
            if(chmin >= 0) break;
            /**/
            choose[best[vid]] = false;
            choose[chid] = true;
            rmst += chmin;
        }
        return rmst;
    }
};
```

8.2 Minimal Directed Spanning Tree

```
/*
 * Minimum Directed Spanning Tree
 * Solve() return answer of mdst if there exists else
     -1
 */
class MDST{
public:
#define v1  first.first
#define v2  first.first
#define w   second
    const static int INF = 0x3f3f3f3f;
    typedef pair<int, int> PI;
    typedef pair<PI, int> PII;
    int N;
    vector<PII> E;
    MDST(int n=0): N(n){}
    int Solve(int r){
        vector<bool> mrg(N+1, false);
        vector<int>  dis(N+1, 0);
        vector<int>  vis(N+1, 0);
        vector<int>  pre(N+1, 0);
        vector<int>  bln(N+1, 0);
        int allw = 0, tmpw = 0;
        while(true){
            tmpw = 0;
            fill(dis.begin(), dis.end(), INF);
            fill(vis.begin(), vis.end(), -1);
            fill(bln.begin(), bln.end(), -1);
            for(int i=0;i<(int)E.size();i++){
                PII e = E[i];
                if(e.v1 != e.v2 && e.v2 != r && e.w <
                    dis[e.v2])
                    dis[e.v2] = e.w, pre[e.v2] = e.v1;
            }
            bool tf = false;
            for(int i=0;i<N;i++){
                if(mrg[i]) continue;
                tf = true;
                int j = s;
                do{
                    bln[j] = s;
                    mrg[j] = true;
                    allw += dis[j];
                    j = pre[j];
                }while(j != s);
                mrg[s] = false;
            }
        }
        if(tf == false) break;
        for(int i=0;i<(int)E.size();i++){
            PII &e = E[i];
            if(bln[e.v2] != -1) e.w -= dis[e.v2];
            if(bln[e.v1] != -1) e.v1 = bln[e.v1];
            if(bln[e.v2] != -1) e.v2 = bln[e.v2];
            if(e.v1 == e.v2) {
                e = E.back();
                E.pop_back();
                i--;
            }
        }
    }
    return allw + tmpw;
}
};
```

8.3 Minimal Rational Spanning Tree

```
/*
 * Minimum Ratio Spanning Tree
 * Solve() returns answer of MRST if there exists an
     answer else -1
 * notice: if you want make it faster, move G, wG to
     normal array
 */
class MRST {
public:
#define w  first
#define u  second
    typedef pair<double, double> PD;
    int N;
    vector<vector<PD> > G;
    vector<vector<double> > wG;
    MRST(int n=0): N(n), G(vector<vector<PD> >(N,
        vector<PD>(N))), wG(vector<vector<double> >(N,
         vector<double>(N))) {
    }
    void add_edge(int a, int b, double _w, double _u){
        G[a][b] = PD(_w, _u);
    }
    void build(double chk){
        for(int i=0;i<N;i++)
            for(int j=0;j<N;j++)
                wG[i][j] = G[i][j].w - chk * G[i][j].u;
    }
    double Mst(double chk){
        build(chk);
        vector<bool> vis(N+1, false);
        vector<double> dis(N+1, 1e9);
        vector<int> pre(N+1);
        double W = 0, U = 0;
        int v = 0;
        int times = 0;
        while(++times < N){
            vis[v] = true;
            for(int i=0;i<N;i++)
                if(!vis[i] && dis[i] > wG[v][i])
                    dis[i] = wG[v][i], pre[i] = v;
            double mn = 1e9;
            for(int i=0;i<N;i++)
                if(!vis[i] && mn > dis[i])
                    mn = dis[i], v = i;
            if(mn == 1e9)
                return -1;
```

```
        double last = -1, cur = 0;
        const double EPS = 1e-9;
        while(fabs(last - cur) > EPS){
            last = cur;
            cur = Mst(last);
        }
        return cur;
    }
};
```

9 Geometry

9.1 Point

```
class Point{
public:
    double x ,y;
    Point(double _x=0, double _y=0): x(_x), y(_y) {}
    Point operator + (const Point &rhs) const {
        return Point(x+rhs.x, y+rhs.y);
    }
    Point operator - (const Point &rhs) const {
        return Point(x-rhs.x, y-rhs.y);
    }
    Point operator * (const double &rhs) const {
        return Point(x*rhs, y*rhs);
    }
    Point operator / (const double &rhs) const {
        return Point(x/rhs, y/rhs);
    }
    bool operator == (const Point &rhs) const {
        return x == rhs.x && y == rhs.y;
    }
    double Abs() const {
        return sqrt(x*x + y*y);
    }
    /*
     * range: 0 ~ 2*PI
     */
    double Arg() const {
        double res = atan2(y, x);
        if(cmp(res) < 0) res += PI*2.0;
        return res;
    }
    double Dot(const Point &rhs) const {
        return (x*rhs.x + y*rhs.y);
    }
    double Cross(const Point &rhs) const {
        return (x*rhs.y - y*rhs.x);
    }
    double Dist(const Point &rhs) const {
        return (*this-rhs).Abs();
    }
    /*
     * unit of d is radian
     */
    Point Rotate(double d) const {
        return Rotate(cos(d), sin(d));
    }
    Point Rotate(double cost, double sint) const {
        return Point(x*cost-y*sint, x*sint+y*cost);
    }
    bool operator < (const Point &rhs) const {
        if(x == rhs.x)
            return y < rhs.y;
        return x < rhs.x;
    }
    friend ostream& operator << (ostream &out, const
        Point &rhs){
        out << "(" << rhs.x << ", " << rhs.y << ")";
        return out;
    }
    Point& update(){
        if(cmp(x) == 0)
            x = 0;
        if(cmp(y) == 0)
            y = 0;
        return *this;
    }
}nilPoint(INF, INF);
```

9.2 Line

```
class Line{
public:
```

```
        if(cmp((rhs-b).Dot(a-b)) < 0) return (rhs-b).Abs();
        return fabs((a-rhs).Cross(b-rhs) / a.Dist(b));
    }
    /*
     * the pedal of rhs on line
     */
    Point Proj(const Point &rhs){
        double r = (a-b).Dot(rhs-b) / (a-b).Dot(a-b);
        return b+(a-b)*r;
    }
    bool OnLine(const Point &rhs){
        /* for segment */
        return cmp((rhs-b).Cross(a-b)) == 0 && cmp((rhs-b).Dot(rhs-a)) <= 0;
        /* for line */
        return cmp((rhs-b).Cross(a-b)) == 0;
    }
    bool Parallel(const Line &rhs){
        return !cmp((a-b).Cross(rhs.a-rhs.b));
    }
    bool IsIntersect(const Line &rhs){
        if(cmp((rhs.a-a).Cross(rhs.b-a) * (rhs.a-b).Cross(rhs.b-b)) > 0) return false;
        if(cmp((a-rhs.a).Cross(b-rhs.a) * (a-rhs.b).Cross(b-rhs.b)) > 0) return false;
        return true;
    }
    /* default is line */
    Point Intersection(const Line &rhs, bool flag=false){
        if(Parallel(rhs)) return nilPoint;
        /* for segment */
        if(flag && IsIntersect(rhs) == false) return nilPoint;
        /* end */
        double s1 = (a-rhs.a).Cross(rhs.b-rhs.a);
        double s2 = (b-rhs.a).Cross(rhs.b-rhs.a);
        return (b*s1-a*s2) / (s1-s2);
    }
    /*
     * move d units along the direction of line
     * example: {(0, 0) -> (1, 1)} move _/2 becomes {(1, 1) -> (2, 2)}
     */
    Line Move(const double &d){
        Point tmp = b - a;
        tmp = tmp / tmp.Abs();
        tmp = tmp.Rotate(PI/2);
        return Line(a+tmp*d, b+tmp*d);
    }

    friend ostream& operator << (ostream &out, const Line &rhs){
        out << "[" << rhs.a << ", " << rhs.b << "]";
        return out;
    }
}nilLine(nilPoint, nilPoint);
```

9.3 Polygon

```
/*
 * default is counterclockwise
 */
class Polygon{
#define COUNTERCLOCKWISE 1
#define CLOCKWISE       -1
public:
    int N;
    vector<Point> s;
    vector<double> A;
    Polygon(int n=0): N(n) {}
    Polygon& add(const Point &n){
        s.push_back(n);
        return *this;
    }
    /*
        for(int i=0;i<N&&t==0;i++){
            int a = i, b = (i+1)%N, c = (i+2)%N;
            t = (s[b]-s[a]).Cross(s[c]-s[b]);
        }
        return t;
    }
    double Perimeter(){
        double res = 0;
        for(int i=0;i<N;i++)
            res += s[i].Dist(s[(i+1)%N]);
        return res;
    }
    double Area(){
        double res = 0;
        for(int i=0;i<N;i++)
            res += s[i].Cross(s[(i+1)%N]);
        return fabs(res/2.0);
    }
#define INSIDE  1
#define ONEDGE  2
#define OUTSIDE 0
    int OnPolygon(const Point &n){
        Point rfn = Point(-INF, n.y);
        Line l = Line(n, rfn);
        int cnt = 0;
        for(int i=0;i<N;i++){
            if(Line(s[i], s[(i+1)%N]).OnLine(n))
                return ONEDGE;
            if(cmp(s[i].y - s[(i+1)%N].y) == 0)
                continue;
            if(l.OnLine(s[i])){
                if(cmp(s[i].y - s[(i+1)%N].y) >= 0)
                    cnt++;
            }else if(l.OnLine(s[(i+1)%N])){
                if(cmp(s[(i+1)%N].y - s[i].y) >= 0)
                    cnt++;
            }else if(l.IsIntersect(Line(s[i], s[(i+1)%N])))
                cnt++;
        }
        return (cnt&1);
    }
    bool IsIntersect(const Line &rhs){
        int i = (upper_bound(A.begin(), A.end(), (rhs.b-rhs.a).Arg()) - A.begin()) % N;
        int j = (upper_bound(A.begin(), A.end(), (rhs.a-rhs.b).Arg()) - A.begin()) % N;
        if(cmp((rhs.b-rhs.a).Cross(s[i]-rhs.a)*(rhs.b-rhs.a).Cross(s[j]-rhs.a)) <= 0)
            return true;
        return false;
    }
};
```

9.3.1 Pick's Theorem

```
int PointsOnedge(){
    int res = 0;
    for(int i=0;i<N;i++)
        res += __gcd(abs(int(s[(i+1)%N].x-s[i].x)), abs(int(s[(i+1)%N].y-s[i].y)));
    return res;
}
int PointsInside(){
    return int(Area()) + 1 - PointsOnedge()/2;
}
```

9.3.2 Mass Center

```
Point MassCenter(){
    if(cmp(Area()) == 0)return nilPoint;
    Point res;
    for(int i=0;i<N;i++)
        res = res + (s[i] + s[(i+1)%N]) * s[i].Cross(s[(i+1)%N]);
    return res / Area() / 6.0;
}
```

```
    sort(that.s.begin(), that.s.end());
    that.s.erase(unique(that.s.begin(), that.s.end()),
        that.s.end());
    vector<Point> &w = res.s;
    for(int i=0;i<(int)that.s.size();i++){
        int sz;
        while((sz=w.size()),
                sz > 1 && cmp((w[sz-1]-w[sz-2]).Cross(
                    that.s[i]-w[sz-2])) <= 0)
            w.pop_back();
        w.push_back(that.s[i]);
    }
    int k = w.size();
    for(int i=(int)that.s.size()-2;i>=0;i--){
        int sz;
        while((sz=w.size()),
                sz > k && cmp((w[sz-1]-w[sz-2]).Cross(
                    that.s[i]-w[sz-2])) <= 0)
            w.pop_back();
        w.push_back(that.s[i]);
    }
    if((int)that.s.size() > 1) w.pop_back();
    res.N = w.size();
    res.A = vector<double>(res.N);
    for(int i=0;i<res.N;i++)
        res.A[i] = (res.s[(i+1)%res.N]-res.s[i]).Arg();
    return res;
}
```

9.3.4 OnConvex

```
/*
 * O(lg N)
 */
int OnConvex(const Point &rhs){
    Point rfn = (s[0]+s[N/3]+s[2*N/3]) / 3.0;
    int l = 0, r = N;
    while(l+1 < r){
        int mid = (l+r) / 2;
        if(cmp((s[l]-rfn).Cross(s[mid]-rfn)) > 0){
            if(cmp((s[l]-rfn).Cross(rhs-rfn)) >= 0 &&
                cmp((s[mid]-rfn).Cross(rhs-rfn)) < 0)
                r = mid;
            else l = mid;
        }else{
            if(cmp((s[l]-rfn).Cross(rhs-rfn)) < 0 &&
                cmp((s[mid]-rfn).Cross(rhs-rfn)) >= 0)
                l = mid;
            else r = mid;
        }
    }
    r %= N;
    int z = cmp((s[r]-rhs).Cross(s[l]-rhs));
    if(z == 0) return ONEDGE;
    else if(z == 1) return OUTSIDE;
    else return INSIDE;
}
```

9.3.5 Convex Diameter

```
/*
 * farthest node pair
 */
pair<double, pair<Point, Point> > Diameter(){
    if(N == 1)
        return make_pair(0, make_pair(s[0], s[0]));
    double maxd = 0;
    Point pa, pb;
    for(int i=0,j=1;i<N;i++){
        while(cmp((s[next(i)]-s[i]).Cross(s[j]-s[i])-(s
            [next(i)]-s[i]).Cross(s[next(j)]-s[i])) <
            0)
            j = next(j);
        double d = s[i].Dist(s[j]);
        if(d > maxd)
            maxd = d, pa = s[i], pb = s[j];
```

9.3.6 Circle

```
class Circle{
public:
    Point O;
    double R;
    Circle(const Point &o, const double &r): O(o), R(r)
        {}
    double Area() const {
        return PI * R * R;
    }
    double Perimeter() const {
        return 2.0 * PI * R;
    }
    /*
     * default not includes on the edge
     */
    bool InCircle(const Point &rhs) const {
        return cmp(O.Dist(rhs) - R) < 0;
    }
    /*
     * default is segment
     * if want to change it to line, remove the if
         which judge t
     */
    vector<Point> Intersection(const Line &rhs){
        vector<Point> res;
        Point d1 = rhs.b - rhs.a, d2 = rhs.a - O;
        double A = d1.x*d1.x + d1.y*d1.y;
        double B = 2.0 * d1.Dot(rhs.a-O);
        double C = d2.x*d2.x + d2.y*d2.y - R*R;
        double D = B*B -4*A*C;
        if(cmp(D) >= 0){
            double t1 = (-B - sqrt(max(0.0, D))) /
                (2.0*A);
            double t2 = (-B + sqrt(max(0.0, D))) /
                (2.0*A);
            if(cmp(t1-1) <= 0 && cmp(t1) >= 0)
                res.push_back(rhs.a + d1*t1);
            if(cmp(t1-t2) != 0 && cmp(t2-1) <= 0 && cmp
                (t2) >= 0)
                res.push_back(rhs.a + d1*t2);
        }
        return res;
    }
    /*
     * the intersections of two circle
     */
    pair<Point, Point> Intersection(const Circle &rhs)
        const {
        double d = (O-rhs.O).Abs();
        double cost = (R*R+d*d-rhs.R*rhs.R) / (2.0*R*d)
            ;
        double sint = sqrt(1.0 - cost*cost);
        Point rfn = (rhs.O-O) / d * R;
        return make_pair(O+rfn.Rotate(cost, sint), O+
            rfn.Rotate(cost, -sint));
    }
    friend ostream& operator << (ostream& out, const
        Circle &rhs){
        out << "C{" << rhs.O << ", " << rhs.R << "}";
        return out;
    }
    bool operator < (const Circle &rhs) const {
        if(cmp(R-rhs.R) != 0) return cmp(R-rhs.R) < 0;
        return O < rhs.O;
    }
    bool operator == (const Circle &rhs) const {
        return cmp(R-rhs.R) == 0 && O == rhs.O;
    }
};
```

9.3.7 Circle Polygon Cover

```
double SectorArea(const Point &rhs1, const Point &rhs2)
    {
```

```
/* called by Area(const Polygon&) */
double calc(const Point &rhs1, const Point &rhs2){
    vector<Point> p;
    bool in1 = (cmp(rhs1.Abs()-R) < 0);
    bool in2 = (cmp(rhs2.Abs()-R) < 0);
    if(in1){
        if(in2)
            return fabs(rhs1.Cross(rhs2)) / 2.0;
        else{
            p = Intersection(Line(rhs1, rhs2));
            return SectorArea(rhs2, p[0]) + fabs(rhs1.
                Cross(p[0])) / 2.0;
        }
    }else{
        if(in2){
            p = Intersection(Line(rhs1, rhs2));
            return SectorArea(p[0], rhs1) + fabs(rhs2.
                Cross(p[0])) / 2.0;
        }else{
            p = Intersection(Line(rhs1, rhs2));
            if((int)p.size() == 2){
                return SectorArea(rhs1, p[0]) +
                    SectorArea(p[1], rhs2) + fabs(p
                    [0].Cross(p[1])) / 2.0;
            }else{
                return SectorArea(rhs1, rhs2);
            }
        }
    }
}
/*
 * the area of overlap between circle and polygon
 */
double Area(const Polygon &rhs){
    Polygon that = rhs;
    for(int i=0;i<that.N;i++){
        that.s[i] = that.s[i] - O;
    }
    double res = 0;
    for(int i=0;i<that.N;i++){
        int sng = cmp(that.s[i].Cross(that.s[(i+1)%that
            .N]));
        if(sng){
            res += sng * calc(that.s[i], that.s[(i+1)%
                that.N]);
        }
    }
    return res;
}
```

9.3.8 Minimal Circle Cover

```
/*
 * circumcircle of two points
 */
Circle Center(const Point &rhs1, const Point &rhs2){
    return Circle((rhs1+rhs2)/2.0, rhs1.Dist(rhs2)/2.0)
        ;
}
/*
 * circumcircle of three points
 */
Circle Center(const Point &rhs1, const Point &rhs2,
    const Point &rhs3){
    Circle res(rhs1, 0);
    Point d1 = rhs2 - rhs1, d2 = rhs3 - rhs1;
    double c1 = (d1.x*d1.x+d1.y*d1.y) / 2.0, c2 = (d2.x
        *d2.x+d2.y*d2.y) / 2.0;
    double d = d1.Cross(d2);
    res.O.x += (c1*d2.y-c2*d1.y) / d;
    res.O.y += (c2*d1.x-c1*d2.x) / d;
    res.R = res.O.Dist(rhs1);
    return res;
}
Circle MinCircleCover(vector<Point> rhs){
    random_shuffle(rhs.begin(), rhs.end());
    Circle res(rhs[0], 0);
    for(int i=1;i<(int)rhs.size();i++){
        if(!res.InCircle(rhs[i])){
            res = Center(rhs[i], rhs[0]);
            for(int j=1;j<i;j++){
                if(!res.InCircle(rhs[j])){
                    res = Center(rhs[i], rhs[j
                        ]);
                    for(int k=0;k<j;k++){
                        if(!res.InCircle(rhs[k])){
                            res = Center(rhs[i], rhs[j
                                ], rhs[k]);
                        }
                    }
                }
            }
        }
    }
    return res;
}
```

9.3.9 Halfplane

```
class HalfPlane{
public:
    Point a, b;
    /* a -> b left side */
    HalfPlane(const Point &_a=Point(), const Point &_b=
        Point()): a(_a), b(_b) {}
    double Value(const Point &rhs) const {
        return (rhs-a).Cross(b-a) ;
    }
    bool Satisfy(const Point &rhs) const {
        return cmp(Value(rhs)) <= 0;
    }
    Point Intersection(const Point &rhs1, const Point &
        rhs2){
        return Line(a, b).Intersection(Line(rhs1, rhs2)
            );
    }
    Point Intersection(const HalfPlane &rhs){
        return Line(a, b).Intersection(Line(rhs.a, rhs.
            b));
    }
    /*
     * return the polygon cut by halfplane
     */
    Polygon Cut(const Polygon &rhs){
        Polygon res;
        const vector<Point> &w = rhs.s;
        int N = w.size();
        for(int i=0;i<(int)w.size();i++){
            if(cmp(Value(w[i])) <= 0)
                res.s.push_back(w[i]);
            else{
                if(cmp(Value(w[prev(i)])) < 0)
                    res.s.push_back(Intersection(w[prev
                        (i)], w[i]));
                if(cmp(Value(w[next(i)])) < 0)
                    res.s.push_back(Intersection(w[i],
                        w[next(i)]));
            }
        }
        res.N = res.s.size();
        return res;
    }
    bool operator < (const HalfPlane &rhs) const {
        int res = cmp((b-a).Arg() - (rhs.b-rhs.a).Arg()
            );
        return res == 0 ? rhs.Satisfy(a) : (res<0);
    }
    friend ostream& operator << (ostream& out, const
        HalfPlane &rhs){
        out << "{" << rhs.a << ", " << rhs.b << "}";
        return out;
    }
};
```

9.3.10 Halfplane Set

```
class HalfPlaneSet{
public:
    vector<HalfPlane> s;
    HalfPlaneSet& add(const HalfPlane &rhs){
```

```
     */
    Polygon Solve(){
        Polygon res;
        sort(s.begin(), s.end());
        deque<HalfPlane> q;
        deque<Point> ans;
        q.push_back(s[0]);
        for(int i=1;i<(int)s.size();i++){
            if(cmp((s[i].b-s[i].a).Arg()-(s[i-1].b-s[i-1].a).Arg()) == 0) continue;
            while(ans.size() > 0 && cmp(s[i].Value(ans.back())) >= 0){
                ans.pop_back();
                q.pop_back();
            }
            while(ans.size() > 0 && cmp(s[i].Value(ans.front())) >= 0){
                ans.pop_front();
                q.pop_front();
            }
            ans.push_back(q.back().Intersection(s[i]));
            q.push_back(s[i]);
        }
        while(ans.size() > 0 && cmp(q.front().Value(ans.back())) >= 0){
            ans.pop_back();
            q.pop_back();
        }
        while(ans.size() > 0 && cmp(q.back().Value(ans.front())) >= 0){
            ans.pop_front();
            q.pop_front();
        }
        ans.push_back(q.back().Intersection(q.front()));
        for(int i=0;i<(int)ans.size();i++)
            res.add(ans[i]);
        res.N = res.s.size();
        return res;
    }
};
```

9.3.11 Kernel of Polygon

```
/*
 * the kernel of the polygon
 */
Polygon Kernel(const Polygon &rhs){
    HalfPlaneSet hlps;
    for(int i=0;i<rhs.N;i++)
        hlps.add(HalfPlane(rhs.s[i], rhs.s[(i+1)%rhs.N]));
    return hlps.Solve();
}
```

10 Data Structure

10.1 Splay Tree

```
#include <bits/stdc++.h>
using namespace std;
template <class T>
class SplayTree{
public:
    class Node{
    public:
        Node *L, *R, *P;
        T val;
        int sz;
        Node(const T &rhs=T()):
            L(NULL), R(NULL), P(NULL), val(rhs), sz(1) {}
        void Up(){
            sz = 1 + NodeSize(L) + NodeSize(R);
        }
    };
    static int NodeSize(Node *rhs){
        return rhs?rhs->sz:0;
    }
    Node *root;
    SplayTree(): root(NULL){}
    SplayTree(const T &rhs): root(new Node(rhs)){}
    ~SplayTree(){
    }
    void Free(){
        this->Free(this->root);
    }
    void Free(Node *rhs){
        if(!rhs) return;
        if(rhs->L)Free(rhs->L);
        if(rhs->R)Free(rhs->R);
        delete rhs;
        rhs = NULL;
    }
    int Size() const {
        return NodeSize(root);
    }
    void LeftRotate(Node *rhs){
        Node *x = rhs, *y = x->R;
        x->R = y->L;
        if(y->L)y->L->P = x;
        y->P = x->P;
        if(!x->P)root = y;
        else if(x->P->L == x)x->P->L = y;
        else x->P->R = y;
        y->L = x; x->P = y;
        x->Up(); y->Up();
    }
    void RightRotate(Node *rhs){
        Node *x = rhs, *y = x->L;
        x->L = y->R;
        if(y->R)y->R->P = x;
        y->P = x->P;
        if(!x->P)root = y;
        else if(x->P->L == x)x->P->L = y;
        else x->P->R = y;
        y->R = x; x->P = y;
        x->Up(); y->Up();
    }
    void Splay(Node *rhs){
        while(rhs->P != NULL){
            if(rhs->P->P == NULL){
                if(rhs->P->L == rhs)RightRotate(rhs->P);
                else LeftRotate(rhs->P);
            }else if(rhs->P->L == rhs && rhs->P->P->L == rhs->P){
                RightRotate(rhs->P->P);
                RightRotate(rhs->P);
            }else if(rhs->P->L == rhs && rhs->P->P->R
```

```
                LeftRotate(rhs->P);
            }else{
                LeftRotate(rhs->P);
                RightRotate(rhs->P);
            }
        }
    }
    Node* FindMin() const {
        Node *tr = root;
        while(tr->L)tr = tr->L;
        return tr;
    }
    Node* FindMax() const {
        Node *tr = root;
        while(tr->R)tr = tr->R;
        return tr;
    }
    Node* Find(int k) const {
        Node *tr = root;
        while(tr){
            if(NodeSize(tr->L) >= k)
                tr = tr->L;
            else if(NodeSize(tr->L)+1 == k)
                break;
            else if(tr->R)
                k -= (NodeSize(tr->L)+1), tr = tr->R;
        }
        return tr;
    }
    void Merge(SplayTree rhs){
        if(rhs.Size() == 0)
            return;
        if(this->Size() == 0){
            *this = rhs;
            return;
        }
        this->Splay(this->FindMax());
        this->root->R = rhs.root;
        this->root->R->P = this->root;
        this->root->Up();
    }
    void Insert(const T &rhs){
        this->Merge(SplayTree(rhs));
    }
    void Split(int k, SplayTree &rhs1, SplayTree &rhs2)
        {
        this->Splay(this->Find(k));
        rhs1.root = this->root;
        rhs2.root = this->root->R;
        rhs1.root->R = NULL;
        if(rhs2.root)rhs2.root->P = NULL;
        rhs1.root->Up();
    }
    void Delete(int k){
        this->Splay(this->Find(k));
        SplayTree a, b;
        a.root = this->root->L;
        b.root = this->root->R;
        if(a.root)a.root->P = NULL;
        if(b.root)b.root->P = NULL;
        delete this->root;
        a.Merge(b);
        this->root = a.root;
    }
    void Print() const {
        print(this->root);
        puts("");
    }
    void print(Node *rhs, int a=0) const {
        if(rhs == NULL)return;
        print(rhs->L, a+1);
        cout << rhs->val << " ";
        print(rhs->R, a+1);
    }
};
int main(){
    const int size = 10;
        do{
            a = rand()%size;
            b = rand()%size;
        }while(a == b);
        s[a].Merge(s[b]);
        s[b].root = NULL;
    }
    for(int i=0;i<size;i++){
        printf("%d\n", i);
    }
    return 0;
}
```

11 String

11.1 Z Value

```
vector<int> z_value(string s){
    int len = s.size();
    vector<int> z(0, len);
    int l=0, r=0;
    z[0] = len;
    for(int i=1,j;i<len;z[i]=j,i++){
        j = max(min(z[i-l], r-i), 0);
        while(i+j<len&&s[i+j]==s[j])j++;
        if(i+z[i]>r)r=(l=i)+z[i];
    }
    return z;
}
```

11.2 Z Value Longest Palindrome

```
vector<int> zvaule_pali(string s1){
    int len1=s1.size(), len2=len1*2-1;
    vector<int> z(len2, 0);
    string s2(len2, '@');
    for(int i=0;i<len2;i++)
        if(!(i&1))s2[i] = s1[i/2];
    z[0] = 1;
    int l=0, r=0;
    for(int i=1;i<len2;i++){
        if(i>r){
            l = r = i;
            while(l>0&&r<len2-1&&s2[l-1]==s2[r+1])l--,
                r++;
            z[i] = r-l+1;
        }else{
            z[i] = z[((l+r)&(~1))-i];
            int nr = i+z[i]/2;
            if(nr==r){
                l = i*2-r;
                while(l>0&&r<len2-1&&s2[l-1]==s2[r+1])l
                    --, r++;
                z[i] = r-l+1;
            }else if(nr>r){
                z[i] = (r-i)*2+1;
            }
        }
    }
    return z;
}
```