

## Codebook

1	Basic	
1.1	vimrc	
2	Number	
2.1	Extended GCD	
2.2	Modular Inverse	
2.3	Line Modular Equation	
2.4	Chinese Remainder Theorem	
2.5	C(N,M)	
2.6	Phi	
2.7	Miller Rabin	
2.8	FFT	
2.9	Function	
2.10	Equation	
2.11	Permutation	
3	Matrix	
3.1	Guass Elimination	
3.2	Solve Matrix (Ax=B)	
3.3	Inverse Matrix	
4	Graph	
4.1	Bridge And Cut	
4.2	BCC	
4.3	SCC	
4.4	Two Sat	
5	Path	
5.1	Kth Shortest	
5.2	EulerCircuit	
6	Flow	
6.1	Dinic	
6.2	StoerWanger	
6.3	Mixed Euler	
7	Match	
7.1	BiMatch	
7.2	KM	
7.3	General Match	
8	MST	
8.1	Restricted Minimal Spanning Tree	
8.2	Minimal Directed Spanning Tree	
8.3	Minimal Rational Spanning Tree	
9	Geometry	
9.1	Point	
9.2	Line	
9.3	Polygon	
9.3.1	Pick's Theorem	
9.3.2	Mass Center	
9.3.3	Convex	
9.3.4	OnConvex	
9.3.5	Convex Diameter	
9.3.6	Circle	
9.3.7	Circle Polygon Cover	
9.3.8	Minimal Circle Cover	
9.3.9	Halfplane	
9.3.10	Halfplane Set	
9.3.11	Kernel of Polygon	
10	Data Structure	
10.1	Splay Tree	
11	String	
11.12	Value	
11.22	Value Longest Palindrome	

## 1 Basic

## 1.1 vimrc

```

1 1 set nocompatible
2 2 filetype plugin indent on
2 3 set t_Co=256
2 4 set term=screen-256color
2 5 set number
2 6 set tabstop=4
2 7 set shiftwidth=4
3 8 set softtabstop=4
3 9 set expandtab
3 10 set wrap
4 11 set showcmd
4 12 colorscheme darkblue
4 13 map <F2> :w <CR> :call OP() <CR>
4 14 map! <F2> <ESC> :w <CR> :call OP() <CR> <ESC>
15 map <F9> :w <CR> :call CP_R() <CR> <ESC>
5 16 map! <F9> <ESC> :w <CR> :call CP_R() <CR> <ESC>
5 17 map <HOME> ^
6 18 map! <HOME> <ESC>^i
6 19 map <ESC>OH <HOME>
20 map! <ESC>OH <HOME>
7 21 map <END> $
7 22 map <ESC>OF <END>
23 map! <ESC>OF <ESC><END>a
7 24 function CP_R()
7 25
7 26   if( &ft == 'cpp')
7 27       let cpl = 'g++ -w -o "%:r.exe" -std=c++11 "%" |
8       let exc = '"./%:r.exe"'
9
9 28   elseif( &ft == 'python')
9 29       let exc = 'python "%"'
9 30   endif
10 31   let pause = 'printf "Press any key to continue..." &&
10 32       read -n 1 && exit'
11 33   if !exists('exc')
11 34       echo 'Can''t compile this filetype...'
12 35       return
12 36   endif
12 37   if exists('cpl')
13 38       let cp_r = cpl . ' && time ' . exc
13 39   else
13 40       let cp_r = 'time ' . exc
14 41   endif
14 42   execute '! clear && ' . cp_r . ' && ' . pause
14 43 endfunction
15 44 function OP()
15 45     execute '!$COLORTERM -x gedit ' . "%" . ";"
16 46 endfunction

```

## 2 Number

### 2.1 Extended GCD

```

1 ll ext_gcd(ll a, ll b, ll &x, ll &y){
2     ll d=a;
3     if(b!=0ll){
4         d=ext_gcd(b, a%b, y, x);
5         y--=(a/b)*x;
6     }
7     else x=1ll, y=0ll;
8     return d;
9 }

```

### 2.2 Modular Inverse

```

1 /*
2  * find the inverse of n modular p
3  */
4 ll mod_inverse(ll n, ll p){
5     ll x, y;
6     ll d = ext_gcd(n, p, x, y);
7     return (p+x%p) % p;
8 }

```

### 2.3 Line Modular Equation

```

1 /*
2  * ax = b (mod n)
3  * return a set of answer(vector<ll>)
4  */
5 vector<ll> line_mod_equation(ll a, ll b, ll n){
6     ll x, y, d;
7     d = ext_gcd(a, n, x, y);
8     vector<ll> ans;
9     if(b%d==0ll){
10         x = (x%n + n) % n;
11         ans.push_back((x*(b/d))%(n/d));
12         for(ll i=1; i<d; i++){
13             ans.push_back((ans[0]+i*n/d)%n);
14         }
15         return ans;
16 }

```

### 2.4 Chinese Remainder Theorem

```

1 /*
2  * solve the chinese remainder theorem(CRT)
3  * if a.size() != m.size(), return -1
4  * return the minimum positive answer of CRT
5  * x = a[i] (mod m[i])
6  */
7 int CRT(vector<int> a, vector<int> m) {
8     if(a.size() != m.size()) return -1;
9     int M = 1;
10    for(int i=0; i<(int)m.size(); i++)
11        M *= m[i];
12    int res = 0;
13    for(int i=0; i<(int)a.size(); i++)
14        res = (res + (M/m[i])*mod_inverse(M/m[i], m[i])
15            *a[i]) % M;
16    return (res + M) % M;
17 }

```

### 2.5 C(N,M)

```

1 /* P is the modular number */
2 #define P 24851
3 int fact[P+1];
4 /* called by Cmod */
5 int mod_fact(int n, int &e){
6     e = 0;
7     if(n == 0) return 1;
8     int res = mod_fact(n/P, e);
9     e += n / P;
10    if((n/P) % 2 == 0)
11        return res * (fact[n%P]%P);
12    return res * ((P-fact[n%P])%P);
13 }
14 /*

```

```

15  * return C(n, m) mod P
16  */
17 int Cmod(int n, int m){
18     /* this section only need to be done once */
19     fact[0] = 1;
20     for(int i=1; i<=P; i++){
21         fact[i] = fact[i-1] * i%P;
22     }
23     /* end */
24     int a1, a2, a3, e1, e2, e3;
25     a1 = mod_fact(n, e1);
26     a2 = mod_fact(m, e2);
27     a3 = mod_fact(n-m, e3);
28     if(e1 > e2 + e3) return 0;
29     return a1 * mod_inverse(a2 * (a3%P), P) % P;
30 }

```

### 2.6 Phi

```

1 /*
2  * gen phi from 1~MAXN
3  * store answer in phi
4  */
5 #define MAXN 100
6 int mindiv[MAXN], phi[MAXN];
7 void genphi(){
8     for(int i=1; i<MAXN; i++)
9         mindiv[i] = i;
10    for(int i=2; i<MAXN; i++){
11        if(mindiv[i] == i)
12            for(int j=i; j<MAXN; j+=i)
13                mindiv[j] = i;
14    }
15    phi[1] = 1;
16    for(int i=2; i<MAXN; i++){
17        phi[i] = phi[i/mindiv[i]];
18        if((i/mindiv[i])%mindiv[i] == 0)
19            phi[i] *= mindiv[i];
20        else phi[i] *= (mindiv[i]-1);
21    }
22 }

```

### 2.7 Miller Rabin

```

1 ll pow_mod(ll x, ll N, ll M) {
2     ll res = 1;
3     x %= M;
4     while(N){
5         if(N&1ll) res = mul_mod(res, x, M);
6         x = mul_mod(x, x, M);
7         N >>= 1;
8     }
9     return res;
10 }
11 bool PrimeTest(ll n, ll a, ll d) {
12     if(n == 2 || n == a) return true;
13     if((n&1) == 0) return false;
14     while((d&1) == 0) d >>= 1;
15     ll t = pow_mod(a, d, n);
16     while((d!=n-1) && (t!=1) && (t!=n-1)){
17         t = mul_mod(t, t, n);
18         d <<= 1;
19     }
20     return (t==n-1) || ((d&1)==1);
21 }
22 bool MillerRabin(ll n){
23     // test set
24     vector<ll> a = {2, 325, 9375, 28178, 450775,
25         9780504, 1795265022};
26     for(int i=0; i<(int)a.size(); i++)
27         if(!PrimeTest(n, a[i], n-1)) return false;
28     return true;
29 }

```

### 2.8 FFT

```

1 /*
2  * called by FFT
3  * build the sequence of a that used to calculate FFT
4  * return a reversed sequence
5  */
6 vector<Complex> reverse(vector<Complex> a){

```

```

7   vector<Complex> res(a);
8   for (int i=1,j=0;i<(int)res.size();i++){
9       for(int k=((int)res.size())>>1;!(j^=k)&k;k
10          >>=1);
11          if(i > j) swap(res[i], res[j]);
12      }
13      return res;
14  }
15  /*
16  * calculate the FFT of sequence
17  * a.size() must be 2^k
18  * flag = 1 -> FFT(a)
19  * falg = -1 -> FFT-1(a)
20  * return FFT(a) or FFT-1(a)
21  */
22  vector<Complex> FFT(vector<Complex> a, int flag=1){
23      vector<Complex> res = reverse(a);
24      for(int k=2;k<=(int)res.size();k<=1){
25          double p0 = -pi / (k>>1) * flag;
26          Complex unit_p0(cos(p0), sin(p0));
27          for(int j=0;j<(int)res.size();j+=k){
28              Complex unit(1.0, 0.0);
29              for(int i=j;i<j+k/2;i++,unit*=unit_p0){
30                  Complex t1 = res[i], t2 = res[i+k/2] *
31                      unit;
32                  res[i] = t1 + t2;
33                  res[i+k/2] = t1 - t2;
34              }
35          }
36      }
37      return res;
38  }

```

## 2.9 Function

```

1  /*
2  * class of polynomial function
3  * coef is the coefficient
4  * f(x) = sigma(c[i]*x^i)
5  */
6  class Function {
7  public:
8      vector<double> coef;
9      Function(const vector<double> c=vector<double>()):
10         coef(c){}
11      double operator () (const double &rhs) const {
12          double res = 0.0;
13          double e = 1.0;
14          for(int i=0;i<(int)coef.size();i++,e*=rhs)
15              res += e * coef[i];
16          return res;
17      }
18      Function derivative() const {
19          vector<double> dc((int)this->coef.size()-1);
20          for(int i=0;i<(int)dc.size();i++)
21              dc[i] = coef[i+1] * (i+1);
22          return Function(dc);
23      }
24      int degree() const {
25          return (int)coef.size()-1;
26      }
27  };
28  /*
29  * calculate the integration of f(x) from a to b
30  * divided into n piece
31  * the bigger the n is, the more accurate the answer is
32  */
33  template<class T>
34  double simpson(const T &f, double a, double b){
35      double c = (a+b) / 2.0;
36      return (f(a)+4.0*f(c)+f(b)) * (b-a) / 6.0;
37  }
38  template<class T>
39  double simpson(const T &f, double a, double b, double
40      eps, double A){
41      double c = (a+b) / 2.0;
42      double L = simpson(f, a, c), R = simpson(f, c, b);
43      if(fabs(A-L-R) <= 15.0*eps) return L + R + (A-L-R)
44          / 15.0;
45      return simpson(f, a, c, eps/2, L) + simpson(f, c, b
46          , eps/2, R);
47  }

```

```

44 template<class T>
45 double simpson(const T &f, double a, double b, double
46     eps){
47     return simpson(f, a, b, eps, simpson(f, a, b));
48 }

```

## 2.10 Equation

```

1  /*
2  * called by find
3  * 1 = positive, -1 = negative, 0 = zero
4  */
5  int sign(double x){
6      return x < -EPS ? -1 : x > EPS;
7  }
8  /* called by equation */
9  template<class T>
10 double find(const T &f, double lo, double hi){
11     int sign_lo, sign_hi;
12     if((sign_lo=sign(f(lo))) == 0) return lo;
13     if((sign_hi=sign(f(hi))) == 0) return hi;
14     if(sign_hi * sign_lo > 0) return INF;
15     while(hi-lo>EPS){
16         double m = (hi+lo) / 2;
17         int sign_mid = sign(f(m));
18         if(sign_mid == 0) return m;
19         if(sign_lo * sign_mid < 0)
20             hi = m;
21         else lo = m;
22     }
23     return (lo+hi) / 2;
24 }
25 /*
26 * return a set of answer of f(x) = 0
27 */
28 template<class T>
29 vector<double> equation(const T &f){
30     vector<double> res;
31     if(f.degree() == 1){
32         if(sign(f.coef[1]))res.push_back(-f.coef[0]/f.
33             coef[1]);
34         return res;
35     }
36     vector<double> droot = equation(f.derivative());
37     droot.insert(droot.begin(), -INF);
38     droot.push_back(INF);
39     for(int i=0;i<(int)droot.size()-1;i++){
40         double tmp = find(f, droot[i], droot[i+1]);
41         if(tmp < INF) res.push_back(tmp);
42     }
43     return res;
44 }

```

## 2.11 Permutation

```

1  /*
2  * return the sequence of x-th of n!
3  * max(n) = 12
4  * 0 of 3! -> 123
5  * 5 of 3! -> 321
6  */
7  int factorial[] = {1, 1, 2, 6, 24, 120, 720, 5040,
8      40320, 362880, 3628800, 39916800, 479001600};
9  vector<int> idx2permutation(int x, int n){
10     vector<bool> used(n+1, false);
11     vector<int> res(n);
12     for(int i=0;i<n;i++){
13         int tmp = x / factorial[n-i-1];
14         int j;
15         for(j=1;j<=n;j++)if(!used[j]){
16             if(tmp == 0) break;
17             tmp--;
18         }
19         res[i] = j, used[j] = true;
20         x %= factorial[n-i-1];
21     }
22     return res;
23 }
24 /*
25 * a is x-th of n!
26 * return x(0~n!)

```

```

26 * 123 of 3! -> 0
27 * 321 of 3! -> 5
28 */
29 int permutation2idx(vector<int> a){
30     int res = 0;
31     for(int i=0;i<(int)a.size();i++){
32         int tmp = a[i] - 1;
33         for(int j=0;j<i;j++){
34             if(a[j] < a[i]) tmp--;
35         }
36         res += factorial[(int)a.size()-i-1] * tmp;
37     }
38     return res;
39 }

```

## 3 Matrix

### 3.1 Gauss Elimination

```

1 /*
2  * return gauss eliminated matrix
3  * r will be changed to the number of the non-free
4  * variables
5  * l[i] will be set to true if i-th variable is not
6  * free
7  * ignore flag
8  */
9 Matrix GaussElimination(int &r, vector<bool> &l, int
10 flag=0) {
11     l = vector<bool>(C);
12     r = 0;
13     Matrix res(*this);
14     for(int i=0;i<res.C-flag;i++){
15         for(int j=r;j<res.R;j++){
16             if(fabs(res.at(j, i)) > EPS){
17                 swap(res.D[j], res.D[r]);
18                 break;
19             }
20         }
21         if(fabs(res.at(r, i)) < EPS){
22             continue;
23         }
24         for(int j=0;j<res.R;j++){
25             if(j != r && fabs(res.at(j, i)) > EPS){
26                 double tmp = (double)res.at(j, i) / (
27                     double)res.at(r, i);
28                 for(int k=0;k<res.C;k++){
29                     res.at(j, k) -= tmp * res.at(r, k);
30                 }
31             }
32         }
33         r++;
34         l[i] = true;
35     }
36     return res;
37 }

```

### 3.2 Solve Matrix (Ax=B)

```

1 /*
2  * Ax = b
3  * it will return the answer(x)
4  * if row != column or there is any free variable, it
5  * will return an empty vector
6  */
7 vector<double> Solve(vector<double> a) {
8     if(R != C) return vector<double>();
9     vector<double> res(R);
10    Matrix t(R, C+1);
11    for(int i=0;i<R;i++){
12        for(int j=0;j<C;j++){
13            t.at(i, j) = a[j];
14        }
15        t.at(i, C) = a[i];
16    }
17    int r = 0;
18    vector<bool> l;
19    t = t.GaussElimination(r, l, 1);
20    if(r != R) return vector<double>();
21    for(int i=0;i<C;i++){
22        if(l[i])for(int j=0;j<R;j++){
23            if(fabs(t.at(j, i)) > EPS)
24                res[i] = t.at(j, C) / t.at(j, i);
25        }
26    }
27    return res;
28 }

```

### 3.3 Inverse Matrix

```

1 /*
2  * return an inverse matrix
3  * if row != column or the inverse matrix doesn't exist
4  * , it will return an empty matrix
5  */
6 Matrix Inverse() {
7     if(R != C) return Matrix();

```

```

7| Matrix t(R, R*2);
8| for(int i=0;i<R;i++){
9|     for(int j=0;j<C;j++){
10|         t.at(i, j) = at(i, j);
11|         t.at(i, i+R) = 1;
12|     }
13|     int r = 0;
14|     vector<bool> l;
15|     t = t.GuassElimination(r, l, R);
16|     if(r != R) return Matrix();
17|     for(int i=0;i<C;i++){
18|         if(l[i]) for(int j=0;j<R;j++){
19|             if(fabs(t.at(j, i)) > EPS){
20|                 for(int k=0;k<C;k++){
21|                     t.at(j, C+k) /= t.at(j, i);
22|                 }
23|             }
24|         }
25|     }
26|     Matrix res(R, C);
27|     for(int i=0;i<R;i++){
28|         for(int j=0;j<C;j++){
29|             res.at(i, j) = t.at(i, j+C);
30|         }
31|     }

```

## 4 Graph

### 4.1 Bridge And Cut

```

1| /* called by cut_bridge */
2| void _cut_bridge(int x, int f, int d){
3|     vis[x] = 1;
4|     dfn[x] = low[x] = d;
5|     int children = 0;
6|     for(int i=0;i<(int)vc[x].size();i++){
7|         Edge e = vc[x][i];
8|         if(e.to != f && vis[e.to] == 1)
9|             low[x] = min(low[x], dfn[e.to]);
10|         if(vis[e.to] == 0){
11|             _cut_bridge(e.to, x, d+1);
12|             children++;
13|             low[x] = min(low[x], low[e.to]);
14|             if((f == -1 && children > 1) || (f != -1 &&
15|                 low[e.to] >= dfn[x]))
16|                 cut[x] = true;
17|             if(low[e.to] > dfn[x])
18|                 bridge[x][e.to] = bridge[e.to][x] =
19|                     true;
20|         }
21|     }
22| }
23| /*
24| * solve the cut and bridge
25| * store answer in cut(vector<bool>) and bridge(vector<
26| * vector<bool> >)
27| * cut[i] == true iff i-th node is cut
28| * bridge[i][j] == true iff edge between i-th and j-th
29| * is bridge
30| */
31| void cut_bridge(){
32|     vis = vector<int>(N+1, 0);
33|     dfn = low = vector<int>(N+1);
34|     cut = vector<bool>(N+1);
35|     bridge = vector<vector<bool>>(N+1, vector<bool>(N
36|         +1, false));
37|     for(int i=0;i<N;i++){
38|         if(!vis[i])
39|             _cut_bridge(i, -1, 0);
40|     }
41| }

```

### 4.2 BCC

```

1| /* called by BCC */
2| void _BBC(int x, int d){
3|     stk[++top] = x;
4|     dfn[x] = low[x] = d;
5|     for(int i=0;i<(int)vc[x].size();i++){
6|         Edge e = vc[x][i];
7|         if(dfn[e.to] == -1){
8|             _BBC(e.to, d+1);
9|             if(low[e.to] >= dfn[x]){
10|                 vector<int> l;
11|                 do{
12|                     l.push_back(stk[top]);
13|                     top--;
14|                 }while(stk[top+1] != e.to);
15|                 l.push_back(x);
16|                 bcc.push_back(l);
17|             }
18|             low[x] = min(low[x], low[e.to]);
19|         }else low[x] = min(low[x], dfn[e.to]);
20|     }
21| }
22| /*
23| * solve the biconnected components(BCC)
24| * store answer in bcc(vector<vector<int> >)
25| * bcc.size() is the number of BCC
26| * bcc[i] is the sequence of a BCC
27| */
28| void BCC(){
29|     dfn = low = vector<int>(N+1, -1);
30|     bcc = vector<vector<int>>();
31|     stk = vector<int>(N+1, -1);
32|     top = -1;
33|     for(int i=0;i<N;i++){

```

```

34     if(dfn[i] == -1)
35         _BBC(i, 0);
36 }

```

### 4.3 SCC

```

1  /* called by SCC */
2  void _SCC(int x, int d){
3      stk[++top] = x;
4      dfn[x] = low[x] = d;
5      vis[x] = 1;
6      for(int i=0; i<(int)vc[x].size(); i++){
7          Edge e = vc[x][i];
8          if(dfn[e.to] != -1){
9              if(vis[e.to] == 1)
10                 low[x] = min(low[x], dfn[e.to]);
11             }else{
12                 _SCC(e.to, d+1);
13                 low[x] = min(low[x], low[e.to]);
14             }
15         }
16         if(low[x] == dfn[x]){
17             while(stk[top] != x){
18                 scc[stk[top]] = scc_cnt;
19                 vis[stk[top]] = 2;
20                 top--;
21             }
22             scc[stk[top]] = scc_cnt++;
23             vis[stk[top]] = 2;
24             top--;
25         }
26     }
27     /*
28     * solve the strongly connected component(SCC)
29     * store answer in scc(vector<int>)
30     * the value of scc[i] means the id of the SCC which i-
31       th node in (id is based 0)
32     * scc_cnt id the number of SCC
33     */
34     void SCC(){
35         dfn = low = vector<int>(N+1, -1);
36         vis = vector<int>(N+1, 0);
37         scc = vector<int>(N+1, 0);
38         scc_cnt = 0;
39         stk = vector<int>(N+1, -1);
40         top = -1;
41         for(int i=0; i<N; i++){
42             if(dfn[i] == -1)
43                 _SCC(i, 0);
44         }
45     }
46 }

```

### 4.4 Two Sat

```

1  /*
2  * called by TwoSat
3  * get the value of i-th
4  * 1 = true, 0 = false, -1 = undefined
5  */
6  int TwoSatGet(int x){
7      int r = x > N/2 ? x-N/2 : x;
8      if(twosatans[r] == -1)
9          return -1;
10     return x > N/2 ? !twosatans[r] : twosatans[r];
11 }
12 /*
13 * solve the 2SAT
14 * return true if there exists a set of answer
15 * store the answer in twosatans
16 */
17 bool TwoSat(){
18     SCC();
19     twosatans = vector<int>(N/2+1, -1);
20     for(int i=0; i<N/2; i++){
21         if(scc[i] == scc[i+N/2])
22             return false;
23     }
24     vector<vector<int>> c(scc_cnt+1);
25     for(int i=0; i<N; i++){
26         c[scc[i]].push_back(i);
27     }
28     for(int i=0; i<scc_cnt; i++){
29         int val = 1;
30         for(int j=0; j<(int)c[i].size(); j++){
31             int x = c[i][j];
32             if(TwoSatGet(x) == 0)
33                 val = 0;
34             for(int k=0; k<(int)vc[x].size(); k++){
35                 if(TwoSatGet(vc[x][k].to) == 0)
36                     val = 0;
37             }
38             if(!val)
39                 break;
40         }
41         for(int j=0; j<(int)c[i].size(); j++){
42             if(c[i][j] > N/2)
43                 twosatans[c[i][j]-N/2] = !val;
44             else
45                 twosatans[c[i][j]] = val;
46         }
47     }
48     return true;
49 }

```

## 5 Path

### 5.1 Kth Shortest

```

1 int KthShortestPath(int s, int t, int k){
2     Graph RG(N);
3     for(int i=0;i<N;i++){
4         for(int j=0;j<(int)vc[i].size();j++){
5             Edge e = vc[i][j];
6             RG.add_edge(e.to, Edge(i, e.w));
7         }
8     }
9     RG.AllDijkstra(t);
10    dis = RG.dis;
11    priority_queue<PI> pq;
12    pq.push(PI(-dis[s], s));
13    while(!pq.empty()){
14        PI v = pq.top();
15        pq.pop();
16        int real = -v.FF - dis[v.SS];
17        if(v.SS == t && (!(--k)))
18            return real;
19        for(int i=0;i<(int)vc[v.SS].size();i++){
20            Edge e = vc[v.SS][i];
21            pq.push(PI(-(real+e.w+dis[e.to]), e.to));
22        }
23    }
24    return -1;
}

```

### 5.2 EulerCircuit

```

1 #define eid w
2 void _EulerCircuit(int x){
3     for(int i=0;i<(int)vc[x].size();i++){
4         Edge e = vc[x][i];
5         if(vis[e.eid]) continue;
6         vis[e.eid] = 1;
7         _EulerCircuit(e.to);
8         eulercircuit.push_back(e.eid);
9     }
10 }
11 bool EulerCircuit(){ // undirected
12     if(!Connected()) return false;
13     vis = vector<int>(M+1, 0);
14     for(int i=0;i<N;i++){
15         if(vc[i].size()&1)
16             return false;
17         //sort
18         sort(vc[i].begin(), vc[i].end());
19     }
20     eulercircuit.clear();
21     _EulerCircuit(0);
22     //sort
23     reverse(eulercircuit.begin(), eulercircuit.end());
24     return true;
25 }

```

## 6 Flow

### 6.1 Dinic

```

1 /*
2  * Maximum Flow Dinic
3  * Solve() returns answer
4  */
5 class Dinic{
6 public:
7     class Edge{
8     public:
9         int v1, v2, f, c;
10        Edge(int _v1=0, int _v2=0, int _f=0, int _c=0):
11            v1(_v1), v2(_v2), f(_f), c(_c){}
12    };
13    int N;
14    vector<vector<int>> >vc;
15    vector<Edge> E;
16    vector<int> dep;
17
18    Dinic(int n=0): N(n), vc(vector<vector<int>>(N+1))
19        , dep(vector<int>(N+1)) {}
20    void add_edge(int a, int b, int c){
21        vc[a].push_back(E.size());
22        E.push_back(Edge(a, b, c, c));
23        vc[b].push_back(E.size());
24        E.push_back(Edge(b, a, 0, c));
25    }
26    int Bfs(int s, int t){
27        fill(dep.begin(), dep.end(), -1);
28        dep[s] = 0;
29        queue<int> q;
30        q.push(s);
31        while(!q.empty()){
32            int v = q.front(); q.pop();
33            for(int i=0;i<(int)vc[v].size();i++){
34                Edge e = E[vc[v][i]];
35                if(e.f > 0 && dep[e.v2] == -1){
36                    dep[e.v2] = dep[v] + 1;
37                    q.push(e.v2);
38                }
39            }
40        }
41        return dep[t];
42    }
43    int Dfs(int x, int df, int t){
44        if(x == t) return df;
45        int res = 0;
46        for(int i=0;i<(int)vc[x].size();i++){
47            Edge &e = E[vc[x][i]];
48            if(e.f > 0 && dep[e.v2] == dep[x] + 1){
49                int f = Dfs(e.v2, min(df, e.f), t);
50                e.f -= f;
51                E[vc[x][i]^1].f += f;
52                df -= f;
53                res += f;
54            }
55        }
56        return res;
57    }
58    int Solve(int s, int t){
59        int flow = 0;
60        while(Bfs(s, t) != -1){
61            flow += Dfs(s, 0x3f3f3f3f, t);
62        }
63        return flow;
64    }
65 };

```

### 6.2 StoerWanger

```

1 /*
2  * Stoer Wanger
3  * Undirected Min Cut
4  * Solve() returns answer if graph is connected else 0
5  */
6 class StoerWanger{
7 public:
8     int N, wN;
9     vector<vector<int>> >G;

```

```

10 vector<int> bln, dis;
11 StoerWanger(int n=0): N(n), G(vector<vector<int>> >(
    N, vector<int>(N))), bln(vector<int>(N, -1)),
    dis(vector<int>(N)) {}
12 void add_edge(int a, int b, int c){
13     G[a][b] += c;
14     G[b][a] += c;
15 }
16 int Mst(int r, int &x, int &y){
17     int t;
18     bln[t=0] = r;
19     for(int i=0;i<wN;i++){
20         if(bln[i] != r)
21             dis[i] = G[0][i];
22     }
23     for(int k=0;k<wN-1;k++){
24         x = t; t = 0;
25         for(int i=0;i<wN;i++){
26             if(bln[i] != r && (!t || dis[i] > dis[t])){
27                 t = i;
28             }
29         }
30         bln[t] = r;
31         for(int i=0;i<wN;i++){
32             if(bln[i] != r)
33                 dis[i] += G[t][i];
34         }
35         y = t;
36         return dis[t];
37     }
38 void Merge(int x, int y){
39     if(x > y) swap(x, y);
40     for(int i=0;i<wN;i++){
41         if(i != x && i != y)
42             G[i][x] += G[i][y], G[x][i] += G[y][i];
43     }
44     if(y == wN-1) return;
45     for(int i=0;i<wN-1;i++){
46         if(i != y)
47             swap(G[i][y], G[i][wN-1]), swap(G[y][i], G[wN-1][i]);
48     }
49 }
50 int Solve(){
51     wN = N;
52     int res = 0x3f3f3f3f;
53     for(int i=0;i<wN;i++){
54         int x, y;
55         res = min(res, Mst(i, x, y));
56         Merge(x, y);
57     }
58     return res;
59 }
60 };

```

### 6.3 Mixed Euler

```

1 /*
2  * Mixed Euler
3  * Solve() returns if there is a euler circuit or not
4  */
5 class MEuler{
6 public:
7     class Edge{
8     #define DIRECTED 1
9     #define UNDIRECTED 0
10    public:
11        int to, dir;
12        Edge(int t=0, int d=0): to(t), dir(d){}
13    };
14    int N;
15    Dinic dinic;
16    vector<int> deg;
17    vector<vector<Edge>> > vc;
18
19    MEuler(int n=0): N(n), dinic(Dinic(N+2)), deg(
        vector<int>(N, 0)), vc(vector<vector<Edge>> >(N
        )) {}
20    void add_edge(int a, int b, int d){
21        vc[a].push_back(Edge(b, d));
22        deg[a]++, deg[b]--;
23    }
24    bool Solve(){
25        for(int i=0;i<N;i++){
26            if(abs(deg[i])>1) return false;
27        }
28        for(int i=0;i<N;i++){
29            if(deg[i] != 0) return false;
30        }
31        return true;
32    }
33 };

```



## 7 Match

### 7.1 BiMatch

```

1  /*
2  * Bipartite Matching
3  * Nx = number of x nodes
4  * Ny = number of y nodes
5  * store matching answer in mx, my
6  * Solve() returns the number of matching
7  */
8  class BiMatch{
9  public:
10     int Nx, Ny;
11     vector<vector<int>> > vc;
12     vector<int> mx, my;
13     vector<int> visy;
14
15     BiMatch(int _x=0, int _y=0): Nx(_x), Ny(_y), vc(
16         vector<vector<int>> >(Nx+1)){}
17
18     void add(int x, int y){
19         vc[x].push_back(y);
20     }
21
22     bool Match(int x){
23         for(int i=0; i<(int)vc[x].size(); i++){
24             int y = vc[x][i];
25             if(!visy[y]){
26                 visy[y] = 1;
27                 if(my[y] == -1 || Match(my[y])){
28                     mx[x] = y, my[y] = x;
29                     return true;
30                 }
31             }
32         }
33         return false;
34     }
35
36     int Solve(){
37         mx = vector<int>(Nx+1, -1);
38         my = vector<int>(Ny+1, -1);
39         int ans = 0;
40         for(int i=0; i<Nx; i++){
41             visy = vector<int>(Ny+1, 0);
42             ans += Match(i);
43         }
44     };

```

### 7.2 KM

```

1  /*
2  * solve Maximun Bipartite Matching
3  * store matching answer in mx, my
4  * Solve() returns themaximun weight of perfect
5  * matching
6  */
7  class KM{
8  public:
9      #define FF first
10     #define SS second
11     typedef pair<int, int> PI;
12     const static int INF = 0x3f3f3f3f;
13     int Nx, Ny;
14     vector<vector<int>> > mp;
15     vector<int> visx, visy;
16     vector<int> lx, ly, slack;
17     vector<int> mx, my;
18     KM(int x=0, int y=0): Nx(x), Ny(y), mp(vector<
19         vector<int>> >(Nx+1, vector<int>(Ny+1, 0))) {}
20
21     void add(int x, int y, int w){
22         mp[x][y] = w;
23     }
24
25     bool Match(int x){
26         visx[x] = 1;
27         for(int i=0; i<Ny; i++){
28             int y = i;
29             if(visy[y]) continue;
30             if(lx[x] + ly[y] > mp[x][y])

```

```

28         slack[y] = min(slack[y], lx[x] + ly[y]
29             - mp[x][y]);
30         else{
31             visy[y] = 1;
32             if(my[y] == -1 || Match(my[y])){
33                 mx[x] = y, my[y] = x;
34                 return true;
35             }
36         }
37     }
38     return false;
39 }
40
41 int Solve(){
42     mx = vector<int>(Nx+1, -1);
43     my = vector<int>(Ny+1, -1);
44     lx = vector<int>(Nx+1, -INF);
45     ly = vector<int>(Ny+1, 0);
46     for(int i=0; i<Nx; i++){
47         for(int j=0; j<Ny; j++){
48             lx[i] = max(lx[i], mp[i][j]);
49         }
50         for(int i=0; i<Nx; i++){
51             slack = vector<int>(Ny+1, INF);
52             while(true){
53                 visx = vector<int>(Nx+1, 0);
54                 visy = vector<int>(Ny+1, 0);
55                 if(Match(i)) break;
56                 int d = INF;
57                 for(int j=0; j<Ny; j++){
58                     if(!visy[j]) d = min(d, slack[j]);
59                 }
60                 if(d == INF) break;
61                 for(int i=0; i<Nx; i++){
62                     if(visx[i]) lx[i] -= d;
63                 }
64                 for(int i=0; i<Ny; i++){
65                     if(visy[i]) ly[i] += d;
66                     else slack[i] -= d;
67                 }
68             }
69         }
70     }
71     int res = 0;
72     for(int i=0; i<Nx; i++){
73         if(mx[i] != -1)
74             res += mp[i][mx[i]];
75     }
76     return res;
77 }

```

### 7.3 General Match

```

1  /*
2  * Maximun General Graph Matching
3  * store answer in m
4  * Solve() returns the number of matching
5  * important!!!
6  * notice the order of disjoint set when unioning
7  */
8  class GMatch{
9  public:
10     int N;
11     vector<vector<int>> > vc;
12     DisjointSet djs;
13     vector<int> m, d, c1, c2, p, vis;
14     queue<int> q;
15     int ts;
16     GMatch(int n): N(n), vc(vector<vector<int>> >(N+1)),
17         djs(DisjointSet(N)), ts(0){}
18
19     void add(int a, int b){
20         vc[a].push_back(b);
21         vc[b].push_back(a);
22     }
23
24     void path(int x, int r){
25         if(x==r) return;
26         if(d[x] == 0){
27             int i = p[x], j = p[p[x]];
28             path(j, r);
29             m[i] = j, m[j] = i;
30         }
31         else if(d[x] == 1){
32             int i = c1[x], j = c2[x];
33             path(i, m[x]);
34             path(i, r);

```

```

34     m[i] = j, m[j] = i;
35 }
36 }
37
38 void blossom(int x, int y, int bi){
39     for(int i=djs.find(x);i!=bi;i=djs.find(p[i])){
40         djs.U(bi, i);
41         if(d[i] == 1)
42             c1[i] = x, c2[i] = y, q.push(i);
43     }
44 }
45
46 int lca(int x,int y,int r){
47     ts++;
48     vis[r] = ts;
49     for(int i=djs.find(x);i!=r;i=djs.find(p[i]))
50         vis[i] = ts;
51     int b;
52     for(b=djs.find(y);vis[b]!=ts;b=djs.find(p[b]));
53     return b;
54 }
55
56 bool Match(int x){
57     djs.init();
58     d = vector<int>(N+1, -1);
59     d[x] = 0;
60     q = queue<int>();
61     q.push(x);
62     while(!q.empty()){
63         int u = q.front(); q.pop();
64         for(int i=0;i<(int)vc[u].size();i++){
65             int v = vc[u][i];
66             if(m[v] != v && djs.find(u) != djs.find
67                 (v)){
68                 if(d[v] == -1){
69                     if(m[v] == -1){
70                         path(u, x);
71                         m[u] = v, m[v] = u;
72                         return true;
73                     }
74                     p[v] = u, p[m[v]] = v;
75                     d[v] = 1, d[m[v]] = 0;
76                     q.push(m[v]);
77                 }
78                 else{
79                     if(d[djs.find(v)] == 0){
80                         int bi=lca(u, v, x);
81                         blossom(u, v, bi);
82                         blossom(v, u, bi);
83                     }
84                 }
85             }
86         }
87     }
88     return false;
89 }
90
91 int Solve(){
92     m = c1 = c2 = d = p = vis = vector<int>(N+1,
93         -1);
94     int ans = 0;
95     for(int i=0;i<N;i++){
96         if(m[i] == -1){
97             if(Match(i)) ans++;
98             else m[i]=i;
99         }
100     }
101     return ans;
102 }

```

## 8 MST

### 8.1 Restricted Minimal Spanning Tree

```

1 /*
2  * Restricted MST
3  * r = the node is limited
4  * k = the limit
5  * notice: <=k or ==k
6  * Solve() returns value of rmst if there ia an answer
7   else -1
8 */
9 class RMST{
10 public:
11     #define to first.first
12     #define eid first.second
13     #define v1 first.first
14     #define v2 first.second
15     #define w second
16     const static int INF = 0x3f3f3f3f;
17     typedef pair<int, int> PI;
18     typedef pair<PI, int> PII;
19     int N;
20     vector<vector<PII>> vc;
21     vector<PII> E;
22     DisjointSet djs;
23     vector<bool> choose;
24     vector<int> best;
25     vector<PI> adj;
26     RMST(int n=0): N(n), vc(vector<vector<PII>>(N+1)),
27         djs(DisjointSet(N)) {}
28     void add_edge(int a, int b, int w){
29         E.push_back(PII(PI(a, b), w));
30     }
31     static bool cmp(PII a, PII b){
32         return a.w < b.w;
33     }
34     void dfs(int x, int p, int r){
35         for(int i=0;i<(int)vc[x].size();i++){
36             PII e = vc[x][i];
37             if(choose[e.eid] && e.to != p){
38                 if(x == r){
39                     best[e.to] = -1;
40                 }
41                 else{
42                     if(best[x] == -1 || E[best[x]].w <
43                         e.w){
44                         best[e.to] = e.eid;
45                     }
46                     else{
47                         best[e.to] = best[x];
48                     }
49                 }
50                 dfs(e.to, x, r);
51             }
52         }
53     }
54     int Solve(int r, int k){
55         choose = vector<bool>((int)E.size()+1, false);
56         best = vector<int>(N+1, -1);
57         adj = vector<PI>(N+1, PI(INF, -1));
58         sort(E.begin(), E.end(), RMST::cmp);
59         int rmst = 0, m = 0;
60         for(int i=0;i<(int)E.size();i++){
61             PII e = E[i];
62             vc[e.v1].push_back(PII(PI(e.v2, i), e.w));
63             vc[e.v2].push_back(PII(PI(e.v1, i), e.w));
64             if(e.v1 != r && e.v2 != r && djs.find(e.v1)
65                 != djs.find(e.v2)){
66                 choose[i] = true;
67                 djs.U(e.v1, e.v2);
68                 rmst += e.w;
69             }
70         }
71         for(int i=0;i<(int)E.size();i++){
72             PII e = E[i];
73             if(e.v1 == r || e.v2 == r){
74                 int v = (e.v1 == r ? e.v2 : e.v1);
75                 adj[v] = min(adj[v], PI(e.w, i));
76                 if(djs.find(r) != djs.find(v)){
77                     choose[i] = true;
78                     rmst += e.w;
79                     m++;
80                 }
81             }
82         }
83     }
84 }

```

```

74         djs.U(r, v);
75     }
76 }
77 }
78 if(m > k) return -1;
79 for(int j=m+1; j<=k; j++){
80     fill(best.begin(), best.end(), -1);
81     dfs(r, r, r);
82     int chid = -1;
83     int chmin = INF;
84     int vid = -1;
85     for(int i=0; i<N; i++){
86         if(i != r && adj[i].first != INF &&
87            best[i] != -1){
88             if(chmin > adj[i].first - E[best[i]
89                ].w){
90                 chmin = adj[i].first - E[best[i]
91                    ].w;
92                 chid = adj[i].second;
93                 vid = i;
94             }
95         }
96     }
97     /* if ==k
98     if(chid == -1) return -1;
99     */
100    /* if <=k */
101    if(chmin >= 0) break;
102    /**/
103    choose[best[vid]] = false;
104    choose[chid] = true;
105    rmst += chmin;
106 }
return rmst;
}
};

```

## 8.2 Minimal Directed Spanning Tree

```

1 /*
2  * Minimum Directed Spanning Tree
3  * Solve() return answer of mdst if there exists else
4  * -1
5  */
6 class MDST{
7 public:
8 #define v1 first.first
9 #define v2 first.first
10 #define w second
11 const static int INF = 0x3f3f3f3f;
12 typedef pair<int, int> PI;
13 typedef pair<PI, int> PII;
14 int N;
15 vector<PII> E;
16 MDST(int n=0): N(n){}
17 int Solve(int r){
18     vector<bool> mrg(N+1, false);
19     vector<int> dis(N+1, 0);
20     vector<int> vis(N+1, 0);
21     vector<int> pre(N+1, 0);
22     vector<int> bln(N+1, 0);
23     int allw = 0, tmpw = 0;
24     while(true){
25         tmpw = 0;
26         fill(dis.begin(), dis.end(), INF);
27         fill(vis.begin(), vis.end(), -1);
28         fill(bln.begin(), bln.end(), -1);
29         for(int i=0; i<(int)E.size(); i++){
30             PII e = E[i];
31             if(e.v1 != e.v2 && e.v2 != r && e.w <
32                dis[e.v2]){
33                 dis[e.v2] = e.w, pre[e.v2] = e.v1;
34             }
35         }
36         bool tf = false;
37         for(int i=0; i<N; i++){
38             if(mrg[i]) continue;
39             if(pre[i] == -1 && i != r) return -1;
40             if(pre[i] != -1) tmpw += dis[i];
41             int s;
42             for(s=i; s!=-1 && vis[s]==-1; s=pre[s])
43                 vis[s] = i;
44             if(s != -1 && vis[s] == i){

```

```

42         tf = true;
43         int j = s;
44         do{
45             bln[j] = s;
46             mrg[j] = true;
47             allw += dis[j];
48             j = pre[j];
49         }while(j != s);
50         mrg[s] = false;
51     }
52 }
53 if(tf == false) break;
54 for(int i=0; i<(int)E.size(); i++){
55     PII &e = E[i];
56     if(bln[e.v2] != -1) e.w -= dis[e.v2];
57     if(bln[e.v1] != -1) e.v1 = bln[e.v1];
58     if(bln[e.v2] != -1) e.v2 = bln[e.v2];
59     if(e.v1 == e.v2) {
60         e = E.back();
61         E.pop_back();
62         i--;
63     }
64 }
65 }
66 return allw + tmpw;
67 }
68 };

```

## 8.3 Minimal Rational Spanning Tree

```

1 /*
2  * Minimum Ratio Spanning Tree
3  * Solve() returns answer of MRST if there exists an
4  * answer else -1
5  * notice: if you want make it faster, move G, wG to
6  * normal array
7  */
8 class MRST {
9 public:
10 #define w first
11 #define u second
12 typedef pair<double, double> PD;
13 int N;
14 vector<vector<PD>> G;
15 vector<vector<double>> wG;
16 MRST(int n=0): N(n), G(vector<vector<PD>>(N,
17     vector<PD>(N))), wG(vector<vector<double>>(N,
18     vector<double>(N))) {
19 }
20 void add_edge(int a, int b, double _w, double _u){
21     G[a][b] = PD(_w, _u);
22 }
23 void build(double chk){
24     for(int i=0; i<N; i++){
25         for(int j=0; j<N; j++){
26             wG[i][j] = G[i][j].w - chk * G[i][j].u;
27         }
28     }
29 }
30 double Mst(double chk){
31     build(chk);
32     vector<bool> vis(N+1, false);
33     vector<double> dis(N+1, 1e9);
34     vector<int> pre(N+1);
35     double W = 0, U = 0;
36     int v = 0;
37     int times = 0;
38     while(++times < N){
39         vis[v] = true;
40         for(int i=0; i<N; i++){
41             if(!vis[i] && dis[i] > wG[v][i]){
42                 dis[i] = wG[v][i], pre[i] = v;
43             }
44         }
45         double mn = 1e9;
46         for(int i=0; i<N; i++){
47             if(!vis[i] && mn > dis[i])
48                 mn = dis[i], v = i;
49         }
50         if(mn == 1e9)
51             return -1;
52         W += G[pre[v]][v].w;
53         U += G[pre[v]][v].u;
54     }
55     return W / U;
56 }
57 double Solve(){

```

```

49     double last = -1, cur = 0;
50     const double EPS = 1e-9;
51     while(fabs(last - cur) > EPS){
52         last = cur;
53         cur = Mst(last);
54     }
55     return cur;
56 }
57 };

```

## 9 Geometry

### 9.1 Point

```

1 class Point{
2 public:
3     double x, y;
4     Point(double _x=0, double _y=0): x(_x), y(_y) {}
5     Point operator + (const Point &rhs) const {
6         return Point(x+rhs.x, y+rhs.y);
7     }
8     Point operator - (const Point &rhs) const {
9         return Point(x-rhs.x, y-rhs.y);
10    }
11    Point operator * (const double &rhs) const {
12        return Point(x*rhs, y*rhs);
13    }
14    Point operator / (const double &rhs) const {
15        return Point(x/rhs, y/rhs);
16    }
17    bool operator == (const Point &rhs) const {
18        return x == rhs.x && y == rhs.y;
19    }
20    double Abs() const {
21        return sqrt(x*x + y*y);
22    }
23    /*
24     * range: 0 ~ 2*PI
25     */
26    double Arg() const {
27        double res = atan2(y, x);
28        if(cmp(res) < 0) res += PI*2.0;
29        return res;
30    }
31    double Dot(const Point &rhs) const {
32        return (x*rhs.x + y*rhs.y);
33    }
34    double Cross(const Point &rhs) const {
35        return (x*rhs.y - y*rhs.x);
36    }
37    double Dist(const Point &rhs) const {
38        return (*this-rhs).Abs();
39    }
40    /*
41     * unit of d is radian
42     */
43    Point Rotate(double d) const {
44        return Rotate(cos(d), sin(d));
45    }
46    Point Rotate(double cost, double sint) const {
47        return Point(x*cost-y*sint, x*sint+y*cost);
48    }
49    bool operator < (const Point &rhs) const {
50        if(x == rhs.x)
51            return y < rhs.y;
52        return x < rhs.x;
53    }
54    friend ostream& operator << (ostream &out, const
55        Point &rhs){
56        out << "(" << rhs.x << ", " << rhs.y << ")";
57        return out;
58    }
59    Point& update(){
60        if(cmp(x) == 0)
61            x = 0;
62        if(cmp(y) == 0)
63            y = 0;
64        return *this;
65    }
66 } nilPoint(INF, INF);

```

### 9.2 Line

```

1 class Line{
2 public:
3     Point a, b;
4     Line(Point _a=Point(), Point _b=Point()): a(_a), b(
5         _b) {}
6     double Dist(const Point &rhs){
7         if(cmp((rhs-a).Dot(b-a)) < 0) return (rhs-a).
8             Abs();

```

```

7   if(cmp((rhs-b).Dot(a-b)) < 0) return (rhs-b).
   Abs();
8   return fabs((a-rhs).Cross(b-rhs) / a.Dist(b));
9 }
10 /*
11  * the pedal of rhs on line
12  */
13 Point Proj(const Point &rhs){
14     double r = (a-b).Dot(rhs-b) / (a-b).Dot(a-b);
15     return b+(a-b)*r;
16 }
17 bool OnLine(const Point &rhs){
18     /* for segment */
19     return cmp((rhs-b).Cross(a-b)) == 0 && cmp((rhs
20 -b).Dot(rhs-a)) <= 0;
21     /* for line */
22     return cmp((rhs-b).Cross(a-b)) == 0;
23 }
24 bool Parallel(const Line &rhs){
25     return !cmp((a-b).Cross(rhs.a-rhs.b));
26 }
27 bool IsIntersect(const Line &rhs){
28     if(cmp((rhs.a-a).Cross(rhs.b-a) * (rhs.a-b).
29 Cross(rhs.b-b)) > 0) return false;
30     if(cmp((a-rhs.a).Cross(b-rhs.a) * (a-rhs.b).
31 Cross(b-rhs.b)) > 0) return false;
32     return true;
33 }
34 /* default is line */
35 Point Intersection(const Line &rhs, bool flag=false
36 ){
37     if(Parallel(rhs)) return nilPoint;
38     /* for segment */
39     if(flag && IsIntersect(rhs) == false) return
40 nilPoint;
41     /* end */
42     double s1 = (a-rhs.a).Cross(rhs.b-rhs.a);
43     double s2 = (b-rhs.a).Cross(rhs.b-rhs.a);
44     return (b*s1-a*s2) / (s1-s2);
45 }
46 /*
47  * move d units along the direction of line
48  * example: {(0, 0) -> (1, 1)} move _/2 becomes
49  * {(1, 1) -> (2, 2)}
50 */
51 Line Move(const double &d){
52     Point tmp = b - a;
53     tmp = tmp / tmp.Abs();
54     tmp = tmp.Rotate(PI/2);
55     return Line(a+tmp*d, b+tmp*d);
56 }
57 friend ostream& operator << (ostream &out, const
58 Line &rhs){
59     out << "[" << rhs.a << ", " << rhs.b << "]";
60     return out;
61 }
62 }
63 nilLine(nilPoint, nilPoint);

```

### 9.3 Polygon

```

1  /*
2  * default is counterclockwise
3  */
4  class Polygon{
5  #define COUNTERCLOCKWISE 1
6  #define CLOCKWISE -1
7  public:
8      int N;
9      vector<Point> s;
10     vector<double> A;
11     Polygon(int n=0): N(n) {}
12     Polygon& add(const Point &n){
13         s.push_back(n);
14         return *this;
15     }
16     /*
17     * counterclockwise or clockwise
18     * defined as above
19     */
20     int Order(){
21         int t = 0;

```

```

22     for(int i=0;i<N&&t==0;i++){
23         int a = i, b = (i+1)%N, c = (i+2)%N;
24         t = (s[b]-s[a]).Cross(s[c]-s[b]);
25     }
26     return t;
27 }
28 double Perimeter(){
29     double res = 0;
30     for(int i=0;i<N;i++)
31         res += s[i].Dist(s[(i+1)%N]);
32     return res;
33 }
34 double Area(){
35     double res = 0;
36     for(int i=0;i<N;i++)
37         res += s[i].Cross(s[(i+1)%N]);
38     return fabs(res/2.0);
39 }
40 #define INSIDE 1
41 #define ONEDGE 2
42 #define OUTSIDE 0
43 int OnPolygon(const Point &n){
44     Point rfn = Point(-INF, n.y);
45     Line l = Line(n, rfn);
46     int cnt = 0;
47     for(int i=0;i<N;i++){
48         if(Line(s[i], s[(i+1)%N]).OnLine(n))
49             return ONEDGE;
50         if(cmp(s[i].y - s[(i+1)%N].y) == 0)
51             continue;
52         if(l.OnLine(s[i])){
53             if(cmp(s[i].y - s[(i+1)%N].y) >= 0)
54                 cnt++;
55         }else if(l.OnLine(s[(i+1)%N])){
56             if(cmp(s[(i+1)%N].y - s[i].y) >= 0)
57                 cnt++;
58         }else if(l.IsIntersect(Line(s[i], s[(i+1)%N
59 ])))
60             cnt++;
61     }
62     return (cnt&1);
63 }
64 bool IsIntersect(const Line &rhs){
65     int i = (upper_bound(A.begin(), A.end(), (rhs.b
66 -rhs.a).Arg()) - A.begin()) % N;
67     int j = (upper_bound(A.begin(), A.end(), (rhs.a
68 -rhs.b).Arg()) - A.begin()) % N;
69     if(cmp((rhs.b-rhs.a).Cross(s[i]-rhs.a)*(rhs.b-
70 rhs.a).Cross(s[j]-rhs.a)) <= 0)
71         return true;
72     return false;
73 }
74 }
75 };

```

#### 9.3.1 Pick's Theorem

```

1  int PointsOnedge(){
2      int res = 0;
3      for(int i=0;i<N;i++)
4          res += __gcd(abs(s[(i+1)%N].x-s[i].x), abs
5              (s[(i+1)%N].y-s[i].y));
6      return res;
7  }
8  int PointsInside(){
9      return int(Area()) + 1 - PointsOnedge()/2;
10 }

```

#### 9.3.2 Mass Center

```

1  Point MassCenter(){
2      if(cmp(Area()) == 0) return nilPoint;
3      Point res;
4      for(int i=0;i<N;i++)
5          res = res + (s[i] + s[(i+1)%N]) * s[i].Cross(s
6              [(i+1)%N]);
7      return res / Area() / 6.0;
8  }

```

#### 9.3.3 Convex

```

1  Polygon ConvexHull(){
2      Polygon res. that = *this;

```

```

3   sort(that.s.begin(), that.s.end());
4   that.s.erase(unique(that.s.begin(), that.s.end()),
5   that.s.end());
6   vector<Point> &w = res.s;
7   for(int i=0; i<(int)that.s.size(); i++){
8       int sz;
9       while((sz=w.size()),
10          sz > 1 && cmp((w[sz-1]-w[sz-2]).Cross(
11          that.s[i]-w[sz-2])) <= 0)
12          w.pop_back();
13       w.push_back(that.s[i]);
14   }
15   int k = w.size();
16   for(int i=(int)that.s.size()-2; i>=0; i--){
17       int sz;
18       while((sz=w.size()),
19          sz > k && cmp((w[sz-1]-w[sz-2]).Cross(
20          that.s[i]-w[sz-2])) <= 0)
21          w.pop_back();
22       w.push_back(that.s[i]);
23   }
24   if((int)that.s.size() > 1) w.pop_back();
25   res.N = w.size();
26   res.A = vector<double>(res.N);
27   for(int i=0; i<res.N; i++){
28       res.A[i] = (res.s[(i+1)%res.N]-res.s[i]).Arg();
29   }
30   return res;
31 }

```

### 9.3.4 OnConvex

```

1  /*
2  * O(lg N)
3  */
4  int OnConvex(const Point &rhs){
5      Point rfn = (s[0]+s[N/3]+s[2*N/3]) / 3.0;
6      int l = 0, r = N;
7      while(l+1 < r){
8          int mid = (l+r) / 2;
9          if(cmp((s[l]-rfn).Cross(s[mid]-rfn)) > 0){
10             if(cmp((s[l]-rfn).Cross(rhs-rfn)) >= 0 &&
11                cmp((s[mid]-rfn).Cross(rhs-rfn)) < 0)
12                 r = mid;
13             else l = mid;
14         }else{
15             if(cmp((s[l]-rfn).Cross(rhs-rfn)) < 0 &&
16                cmp((s[mid]-rfn).Cross(rhs-rfn)) >= 0)
17                 l = mid;
18             else r = mid;
19         }
20     }
21     r %= N;
22     int z = cmp((s[r]-rhs).Cross(s[l]-rhs));
23     if(z == 0) return ONEDGE;
24     else if(z == 1) return OUTSIDE;
25     else return INSIDE;
26 }

```

### 9.3.5 Convex Diameter

```

1  /*
2  * farthest node pair
3  */
4  pair<double, pair<Point, Point> > Diameter(){
5      if(N == 1)
6          return make_pair(0, make_pair(s[0], s[0]));
7      double maxd = 0;
8      Point pa, pb;
9      for(int i=0, j=1; i<N; i++){
10         while(cmp((s[next(i)]-s[i]).Cross(s[j]-s[i]))-(s
11            [next(i)]-s[i]).Cross(s[next(j)]-s[i])) <
12            0)
13             j = next(j);
14         double d = s[i].Dist(s[j]);
15         if(d > maxd)
16             maxd = d, pa = s[i], pb = s[j];
17         d = s[next(i)].Dist(s[next(j)]);
18         if(d > maxd)
19             maxd = d, pa = s[next(i)], pb = s[next(j)];
20     }
21     return make_pair(maxd, make_pair(pa, pb));
22 }

```

### 9.3.6 Circle

```

1  class Circle{
2  public:
3      Point O;
4      double R;
5      Circle(const Point &o, const double &r): O(o), R(r)
6      {}
7      double Area() const {
8          return PI * R * R;
9      }
10     double Perimeter() const {
11         return 2.0 * PI * R;
12     }
13     /*
14     * default not includes on the edge
15     */
16     bool InCircle(const Point &rhs) const {
17         return cmp(O.Dist(rhs) - R) < 0;
18     }
19     /*
20     * default is segment
21     * if want to change it to line, remove the if
22     * which judge t
23     */
24     vector<Point> Intersection(const Line &rhs){
25         vector<Point> res;
26         Point d1 = rhs.b - rhs.a, d2 = rhs.a - O;
27         double A = d1.x*d1.x + d1.y*d1.y;
28         double B = 2.0 * d1.Dot(rhs.a-O);
29         double C = d2.x*d2.x + d2.y*d2.y - R*R;
30         double D = B*B - 4*A*C;
31         if(cmp(D) >= 0){
32             double t1 = (-B - sqrt(max(0.0, D))) /
33                 (2.0*A);
34             double t2 = (-B + sqrt(max(0.0, D))) /
35                 (2.0*A);
36             if(cmp(t1-1) <= 0 && cmp(t1) >= 0)
37                 res.push_back(rhs.a + d1*t1);
38             if(cmp(t1-1) != 0 && cmp(t2-1) <= 0 && cmp
39                 (t2) >= 0)
40                 res.push_back(rhs.a + d1*t2);
41         }
42         return res;
43     }
44     /*
45     * the intersections of two circle
46     */
47     pair<Point, Point> Intersection(const Circle &rhs)
48     const {
49         double d = (O-rhs.O).Abs();
50         double cost = (R*R+d*d-rhs.R*rhs.R) / (2.0*R*d);
51         double sint = sqrt(1.0 - cost*cost);
52         Point rfn = (rhs.O-O) / d * R;
53         return make_pair(O+rfn.Rotate(cost, sint), O+
54             rfn.Rotate(cost, -sint));
55     }
56     friend ostream& operator << (ostream& out, const
57         Circle &rhs){
58         out << "C{" << rhs.O << ", " << rhs.R << "}";
59         return out;
60     }
61     bool operator < (const Circle &rhs) const {
62         if(cmp(R-rhs.R) != 0) return cmp(R-rhs.R) < 0;
63         return 0 < rhs.O;
64     }
65     bool operator == (const Circle &rhs) const {
66         return cmp(R-rhs.R) == 0 && 0 == rhs.O;
67     }
68 }

```

### 9.3.7 Circle Polygon Cover

```

1  double SectorArea(const Point &rhs1, const Point &rhs2)
2  {
3      double theta = rhs1.Arg() - rhs2.Arg();
4      while(cmp(theta) <= 0) theta += 2.0 * PI;
5      while(cmp(theta - 2.0*PI) > 0) theta -= 2.0 * PI;
6      theta = min(theta, 2.0*PI - theta);
7      return R * R * theta / 2.0;
8  }

```



```

8  /* called by Area(const Polygon&) */
9  double calc(const Point &rhs1, const Point &rhs2){
10     vector<Point> p;
11     bool in1 = (cmp(rhs1.Abs()-R) < 0);
12     bool in2 = (cmp(rhs2.Abs()-R) < 0);
13     if(in1){
14         if(in2)
15             return fabs(rhs1.Cross(rhs2)) / 2.0;
16         else{
17             p = Intersection(Line(rhs1, rhs2));
18             return SectorArea(rhs2, p[0]) + fabs(rhs1.
19                 Cross(p[0])) / 2.0;
20         }
21     }else{
22         if(in2){
23             p = Intersection(Line(rhs1, rhs2));
24             return SectorArea(p[0], rhs1) + fabs(rhs2.
25                 Cross(p[0])) / 2.0;
26         }else{
27             p = Intersection(Line(rhs1, rhs2));
28             if((int)p.size() == 2){
29                 return SectorArea(rhs1, p[0]) +
30                     SectorArea(p[1], rhs2) + fabs(p
31                         [0].Cross(p[1])) / 2.0;
32             }else{
33                 return SectorArea(rhs1, rhs2);
34             }
35         }
36     }
37 }
38 /*
39  * the area of overlap between circle and polygon
40  */
41 double Area(const Polygon &rhs){
42     Polygon that = rhs;
43     for(int i=0;i<that.N;i++){
44         that.s[i] = that.s[i] - 0;
45     }
46     double res = 0;
47     for(int i=0;i<that.N;i++){
48         int sng = cmp(that.s[i].Cross(that.s[(i+1)%that
49             .N]));
50         if(sng){
51             res += sng * calc(that.s[i], that.s[(i+1)%
52                 that.N]);
53         }
54     }
55     return res;
56 }

```

### 9.3.8 Minimal Circle Cover

```

1  /*
2  * circumcircle of two points
3  */
4  Circle Center(const Point &rhs1, const Point &rhs2){
5      return Circle((rhs1+rhs2)/2.0, rhs1.Dist(rhs2)/2.0)
6      ;
7  }
8  /*
9  * circumcircle of three points
10 */
11 Circle Center(const Point &rhs1, const Point &rhs2,
12     const Point &rhs3){
13     Circle res(rhs1, 0);
14     Point d1 = rhs2 - rhs1, d2 = rhs3 - rhs1;
15     double c1 = (d1.x*d1.x+d1.y*d1.y) / 2.0, c2 = (d2.x
16         *d2.x+d2.y*d2.y) / 2.0;
17     double d = d1.Cross(d2);
18     res.O.x += (c1*d2.y-c2*d1.y) / d;
19     res.O.y += (c2*d1.x-c1*d2.x) / d;
20     res.R = res.O.Dist(rhs1);
21     return res;
22 }
23 Circle MinCircleCover(vector<Point> rhs){
24     random_shuffle(rhs.begin(), rhs.end());
25     Circle res(rhs[0], 0);
26     for(int i=1;i<(int)rhs.size();i++){
27         if(!res.InCircle(rhs[i])){
28             res = Circle(rhs[i], 0);
29             for(int j=0;j<i;j++){
30                 if(!res.InCircle(rhs[j])){

```

```

28         res = Center(rhs[i], rhs[j]);
29         for(int k=0;k<j;k++){
30             if(!res.InCircle(rhs[k])){
31                 res = Center(rhs[i], rhs[j]
32                     , rhs[k]);
33             }
34         }
35     }
36 }
37 return res;
38 }
39 }

```

### 9.3.9 Halfplane

```

1  class HalfPlane{
2  public:
3      Point a, b;
4      /* a -> b left side */
5      HalfPlane(const Point &a=Point(), const Point &b=
6          Point()): a(_a), b(_b) {}
7      double Value(const Point &rhs) const {
8          return (rhs-a).Cross(b-a) ;
9      }
10     bool Satisfy(const Point &rhs) const {
11         return cmp(Value(rhs)) <= 0;
12     }
13     Point Intersection(const Point &rhs1, const Point &
14         rhs2){
15         return Line(a, b).Intersection(Line(rhs1, rhs2)
16             );
17     }
18     Point Intersection(const HalfPlane &rhs){
19         return Line(a, b).Intersection(Line(rhs.a, rhs.
20             b));
21     }
22     /*
23     * return the polygon cut by halfplane
24     */
25     Polygon Cut(const Polygon &rhs){
26         Polygon res;
27         const vector<Point> &w = rhs.s;
28         int N = w.size();
29         for(int i=0;i<(int)w.size();i++){
30             if(cmp(Value(w[i])) <= 0)
31                 res.s.push_back(w[i]);
32             else{
33                 if(cmp(Value(w[prev(i)])) < 0)
34                     res.s.push_back(Intersection(w[prev
35                         (i)], w[i]));
36                 if(cmp(Value(w[next(i)])) < 0)
37                     res.s.push_back(Intersection(w[i],
38                         w[next(i)]));
39             }
40         }
41         res.N = res.s.size();
42         return res;
43     }
44     bool operator < (const HalfPlane &rhs) const {
45         int res = cmp((b-a).Arg() - (rhs.b-rhs.a).Arg()
46             );
47         return res == 0 ? rhs.Satisfy(a) : (res<0);
48     }
49     friend ostream& operator << (ostream& out, const
50         HalfPlane &rhs){
51         out << "{" << rhs.a << ", " << rhs.b << "}";
52         return out;
53     }
54 };

```

### 9.3.10 Halfplane Set

```

1  class HalfPlaneSet{
2  public:
3      vector<HalfPlane> s;
4      HalfPlaneSet& add(const HalfPlane &rhs){
5          s.push_back(rhs);
6          return *this;
7      }
8      /*
9      * return the polygon that satisfies all halfplanes

```

```

10  */
11  Polygon Solve(){
12      Polygon res;
13      sort(s.begin(), s.end());
14      deque<HalfPlane> q;
15      deque<Point> ans;
16      q.push_back(s[0]);
17      for(int i=1;i<(int)s.size();i++){
18          if(cmp((s[i].b-s[i].a).Arg()-(s[i-1].b-s[i-1].a).Arg()) == 0) continue;
19          while(ans.size() > 0 && cmp(s[i].Value(ans.back())) >= 0){
20              ans.pop_back();
21              q.pop_back();
22          }
23          while(ans.size() > 0 && cmp(s[i].Value(ans.front())) >= 0){
24              ans.pop_front();
25              q.pop_front();
26          }
27          ans.push_back(q.back().Intersection(s[i]));
28          q.push_back(s[i]);
29      }
30      while(ans.size() > 0 && cmp(q.front().Value(ans.back())) >= 0){
31          ans.pop_back();
32          q.pop_back();
33      }
34      while(ans.size() > 0 && cmp(q.back().Value(ans.front())) >= 0){
35          ans.pop_front();
36          q.pop_front();
37      }
38      ans.push_back(q.back().Intersection(q.front()));
39      ;
40      for(int i=0;i<(int)ans.size();i++)
41          res.add(ans[i]);
42      res.N = res.s.size();
43      return res;
44  };

```

### 9.3.11 Kernel of Polygon

```

1  /*
2  * the kernel of the polygon
3  */
4  Polygon Kernel(const Polygon &rhs){
5      HalfPlaneSet hpls;
6      for(int i=0;i<rhs.N;i++)
7          hpls.add(HalfPlane(rhs.s[i], rhs.s[(i+1)%rhs.N]));
8      return hpls.Solve();
9  }

```

## 10 Data Structure

### 10.1 Splay Tree

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  template <class T>
4  class SplayTree{
5  public:
6      class Node{
7      public:
8          Node *L, *R, *P;
9          T val;
10         int sz;
11         Node(const T &rhs=T()):
12             L(NULL), R(NULL), P(NULL), val(rhs), sz(1)
13         {}
14         void Up(){
15             sz = 1 + NodeSize(L) + NodeSize(R);
16         }
17     };
18     static int NodeSize(Node *rhs){
19         return rhs?rhs->sz:0;
20     }
21     Node *root;
22     SplayTree(): root(NULL){}
23     SplayTree(const T &rhs): root(new Node(rhs)){}
24     ~SplayTree(){}
25     void Free(){
26         this->Free(this->root);
27     }
28     void Free(Node *rhs){
29         if(!rhs) return;
30         if(rhs->L) Free(rhs->L);
31         if(rhs->R) Free(rhs->R);
32         delete rhs;
33         rhs = NULL;
34     }
35     int Size() const {
36         return NodeSize(root);
37     }
38     void LeftRotate(Node *rhs){
39         Node *x = rhs, *y = x->R;
40         x->R = y->L;
41         if(y->L)y->L->P = x;
42         y->P = x->P;
43         if(!x->P)root = y;
44         else if(x->P->L == x)x->P->L = y;
45         else x->P->R = y;
46         y->L = x; x->P = y;
47         x->Up(); y->Up();
48     }
49     void RightRotate(Node *rhs){
50         Node *x = rhs, *y = x->L;
51         x->L = y->R;
52         if(y->R)y->R->P = x;
53         y->P = x->P;
54         if(!x->P)root = y;
55         else if(x->P->L == x)x->P->L = y;
56         else x->P->R = y;
57         y->R = x; x->P = y;
58         x->Up(); y->Up();
59     }
60     void Splay(Node *rhs){
61         while(rhs->P != NULL){
62             if(rhs->P->P == NULL){
63                 if(rhs->P->L == rhs)RightRotate(rhs->P);
64                 else LeftRotate(rhs->P);
65             }else if(rhs->P->L == rhs && rhs->P->P->L == rhs->P){
66                 RightRotate(rhs->P->P);
67                 RightRotate(rhs->P);
68             }else if(rhs->P->L == rhs && rhs->P->P->R == rhs->P){
69                 RightRotate(rhs->P);
70                 LeftRotate(rhs->P);
71             }else if(rhs->P->R == rhs && rhs->P->P->R == rhs->P){
72                 LeftRotate(rhs->P->P);

```



```

73         LeftRotate(rhs->P);
74     }else{
75         LeftRotate(rhs->P);
76         RightRotate(rhs->P);
77     }
78 }
79 }
80 Node* FindMin() const {
81     Node *tr = root;
82     while(tr->L)tr = tr->L;
83     return tr;
84 }
85 Node* FindMax() const {
86     Node *tr = root;
87     while(tr->R)tr = tr->R;
88     return tr;
89 }
90 Node* Find(int k) const {
91     Node *tr = root;
92     while(tr){
93         if(NodeSize(tr->L) >= k)
94             tr = tr->L;
95         else if(NodeSize(tr->L)+1 == k)
96             break;
97         else if(tr->R)
98             k -= (NodeSize(tr->L)+1), tr = tr->R;
99     }
100    return tr;
101 }
102 void Merge(SplayTree rhs){
103     if(rhs.Size() == 0)
104         return;
105     if(this->Size() == 0){
106         *this = rhs;
107         return;
108     }
109     this->Splay(this->FindMax());
110     this->root->R = rhs.root;
111     this->root->R->P = this->root;
112     this->root->Up();
113 }
114 void Insert(const T &rhs){
115     this->Merge(SplayTree(rhs));
116 }
117 void Split(int k, SplayTree &rhs1, SplayTree &rhs2)
118 {
119     this->Splay(this->Find(k));
120     rhs1.root = this->root;
121     rhs2.root = this->root->R;
122     rhs1.root->R = NULL;
123     if(rhs2.root)rhs2.root->P = NULL;
124     rhs1.root->Up();
125 }
126 void Delete(int k){
127     this->Splay(this->Find(k));
128     SplayTree a, b;
129     a.root = this->root->L;
130     b.root = this->root->R;
131     if(a.root)a.root->P = NULL;
132     if(b.root)b.root->P = NULL;
133     delete this->root;
134     a.Merge(b);
135     this->root = a.root;
136 }
137 void Print() const {
138     print(this->root);
139     puts("");
140 }
141 void print(Node *rhs, int a=0) const {
142     if(rhs == NULL)return;
143     print(rhs->L, a+1);
144     cout << rhs->val << " ";
145     print(rhs->R, a+1);
146 }
147 int main(){
148     const int size = 10;
149     const int time = 100000000;
150     SplayTree<int> s[size];
151     for(int i=0;i<time;i++){
152         s[rand()%size].Insert(rand());
153         int a,b;
154         do{
155             a = rand()%size;
156             b = rand()%size;
157         }while(a == b);
158         s[a].Merge(s[b]);
159         s[b].root = NULL;
160     }
161     for(int i=0;i<size;i++){
162         printf("%d\n", i);
163     }
164     return 0;
165 }

```

## 11 String

### 11.1 Z Value

```

1 vector<int> z_value(string s){
2     int len = s.size();
3     vector<int> z(0, len);
4     int l=0, r=0;
5     z[0] = len;
6     for(int i=1,j;i<len;z[i]=j,i++){
7         j = max(min(z[i-1], r-i), 0);
8         while(i+j<len&&s[i+j]==s[j])j++;
9         if(i+z[i]>r)r=(l=i)+z[i];
10    }
11    return z;
12 }
```

### 11.2 Z Value Longest Palindrome

```

1 vector<int> zvaule_pali(string s1){
2     int len1=s1.size(), len2=len1*2-1;
3     vector<int> z(len2, 0);
4     string s2(len2, '@');
5     for(int i=0;i<len2;i++)
6         if(!(i&1))s2[i] = s1[i/2];
7     z[0] = 1;
8     int l=0, r=0;
9     for(int i=1;i<len2;i++){
10        if(i>r){
11            l = r = i;
12            while(l>0&&r<len2-1&&s2[l-1]==s2[r+1])l--,
13                r++;
14            z[i] = r-l+1;
15        }else{
16            z[i] = z[((l+r)&(~1))-i];
17            int nr = i+z[i]/2;
18            if(nr==r){
19                l = i*2-r;
20                while(l>0&&r<len2-1&&s2[l-1]==s2[r+1])l
21                    --, r++;
22                z[i] = r-l+1;
23            }else if(nr>r){
24                z[i] = (r-i)*2+1;
25            }
26        }
27    }
28    return z;
29 }
```