Contents

1 Basic

1.1 .vimrc

```
syn on
se ai nu ru cul mouse=a
se cin et ts=2 sw=2 sts=2
so $VIMRUNTIME/mswin.vim
colo desert
se gfn=Monospace\ 14
```

1.2 Increase Stack Size

```
//stack resize
asm( "mov %0,%%esp\n" ::"g"(mem+10000000) );
//change esp to rsp if 64-bit system

//stack resize (linux)
#include <sys/resource.h>
void increase_stack_size() {
  const rlim_t ks = 64*1024*1024;
  struct rlimit rl;
  int res=getrlimit(RLIMIT_STACK, &rl);
  if(res==0){
    if(rl.rlim_cur<ks){
      rl.rlim_cur=ks;
      res=setrlimit(RLIMIT_STACK, &rl);
    }
  }
}
```

1.3 Misc

```
#include <random>
mt19937 rng(0x5EED);
int randint(int lb, int ub)
{ return uniform_int_distribution<int>(lb, ub)(rng); }

#define SECs (clock() / CLOCKS_PER_SEC)

struct KeyHasher {
  size_t operator()(const Key& k) const {
    return k.first + k.second * 100000;
  }
};
typedef unordered_map<Key,int,KeyHasher> map_t;
```

2 flow

2.1 ISAP

```
#define SZ(c) ((int)(c).size())
struct Maxflow {
  static const int MAXV = 20010;
  static const int INF  = 1000000;
  struct Edge {
    int v, c, r;
    Edge(int _v, int _c, int _r):
      v(_v), c(_c), r(_r) {}
  };
  int s, t;
  vector<Edge> G[MAXV*2];
  int iter[MAXV*2], d[MAXV*2], gap[MAXV*2], tot;
  void init(int x) {
    tot = x+2;
    s = x+1, t = x+2;
    for(int i = 0; i <= tot; i++) {
      G[i].clear();
      iter[i] = d[i] = gap[i] = 0;
    }
  }
  void addEdge(int u, int v, int c) {
```

```
    G[u].push_back(Edge(v, c, SZ(G[v]) ));
    G[v].push_back(Edge(u, 0, SZ(G[u]) - 1));
  }
  int dfs(int p, int flow) {
    if(p == t) return flow;
    for(int &i = iter[p]; i < SZ(G[p]); i++) {
      Edge &e = G[p][i];
      if(e.c > 0 && d[p] == d[e.v]+1) {
        int f = dfs(e.v, min(flow, e.c));
        if(f) {
          e.c -= f;
          G[e.v][e.r].c += f;
          return f;
        }
      }
    }
    if( (--gap[d[p]]) == 0) d[s] = tot;
    else {
      d[p]++;
      iter[p] = 0;
      ++gap[d[p]];
    }
    return 0;
  }
  int solve() {
    int res = 0;
    gap[0] = tot;
    for(res = 0; d[s] < tot; res += dfs(s, INF));
    return res;
  }
} flow;
```

2.2 MinCostFlow

```
struct MinCostMaxFlow{
typedef int Tcost;
  static const int MAXV = 20010;
  static const int INFf = 1000000;
  static const Tcost INFc  = 1e9;
  struct Edge{
    int v, cap;
    Tcost w;
    int rev;
    Edge(){}
    Edge(int t2, int t3, Tcost t4, int t5)
    : v(t2), cap(t3), w(t4), rev(t5) {}
  };
  int V, s, t;
  vector<Edge> g[MAXV];
  void init(int n){
    V = n+2;
    s = n+1, t = n+2;
    for(int i = 0; i <= V; i++) g[i].clear();
  }
  void addEdge(int a, int b, int cap, Tcost w){
    g[a].push_back(Edge(b, cap, w, (int)g[b].size()));
    g[b].push_back(Edge(a, 0, -w, (int)g[a].size()-1));
  }
  Tcost d[MAXV];
  int id[MAXV], mom[MAXV];
  bool inqu[MAXV];
  queue<int> q;
  Tcost solve(){
    int mxf = 0; Tcost mnc = 0;
    while(1){
      fill(d, d+1+V, INFc);
      fill(inqu, inqu+1+V, 0);
      fill(mom, mom+1+V, -1);
      mom[s] = s;
      d[s] = 0;
      q.push(s); inqu[s] = 1;
      while(q.size()){
        int u = q.front(); q.pop();
        inqu[u] = 0;
        for(int i = 0; i < (int) g[u].size(); i++){
          Edge &e = g[u][i];
          int v = e.v;
          if(e.cap > 0 && d[v] > d[u]+e.w){
            d[v] = d[u]+e.w;
            mom[v] = u;
            id[v] = i;
            if(!inqu[v]) q.push(v), inqu[v] = 1;
          }
        }
      }
      if(mom[t] == -1) break ;
      int df = INFf;
      for(int u = t; u != s; u = mom[u])
        df = min(df, g[mom[u]][id[u]].cap);
      for(int u = t; u != s; u = mom[u]){
        Edge &e = g[mom[u]][id[u]];
        e.cap             -= df;
        g[e.v][e.rev].cap += df;
      }
      mxf += df;
      mnc += df*d[t];
    }
    return mnc;
  }
} flow;
```

2.3 Dinic

```
struct Dinic{
  static const int MXN = 10000;
  struct Edge{ int v,f,re; };
  int n,s,t,level[MXN];
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t){
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v, int f){
    E[u].PB({v,f,SZ(E[v])});
    E[v].PB({u,0,SZ(E[u])-1});
  }
  bool BFS(){
    for (int i=0; i<n; i++) level[i] = -1;
    queue<int> que;
    que.push(s);
    level[s] = 0;
    while (!que.empty()){
      int u = que.front(); que.pop();
      for (auto it : E[u]){
        if (it.f > 0 && level[it.v] == -1){
          level[it.v] = level[u]+1;
          que.push(it.v);
        }
      }
    }
    return level[t] != -1;
  }
  int DFS(int u, int nf){
    if (u == t) return nf;
    int res = 0;
    for (auto &it : E[u]){
      if (it.f > 0 && level[it.v] == level[u]+1){
        int tf = DFS(it.v, min(nf,it.f));
        res += tf; nf -= tf; it.f -= tf;
        E[it.v][it.re].f += tf;
        if (nf == 0) return res;
      }
    }
    if (!res) level[u] = -1;
    return res;
  }
  int flow(int res=0){
    while ( BFS() )
      res += DFS(s,2147483647);
    return res;
  }
}flow;
```

2.4 Kuhn Munkres

```
struct KM{
// Maximum Bipartite Weighted Matching (Perfect Match)
  static const int MXN = 650;
  static const int INF = 2147483647; // long long
  int n,match[MXN],vx[MXN],vy[MXN];
  int edge[MXN][MXN],lx[MXN],ly[MXN],slack[MXN];
  // ^^^^ long long
  void init(int _n){
    n = _n;
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
        edge[i][j] = 0;
  }
  void add_edge(int x, int y, int w){ // long long
    edge[x][y] = w;
  }
  bool DFS(int x){
    vx[x] = 1;
    for (int y=0; y<n; y++){
      if (vy[y]) continue;
      if (lx[x]+ly[y] > edge[x][y]){
        slack[y] = min(slack[y], lx[x]+ly[y]-edge[x][y
            ]);
      } else {
        vy[y] = 1;
        if (match[y] == -1 || DFS(match[y])){
          match[y] = x;
          return true;
        }
      }
    }
    return false;
  }
  int solve(){
    fill(match,match+n,-1);
    fill(lx,lx+n,-INF);
    fill(ly,ly+n,0);
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
        lx[i] = max(lx[i], edge[i][j]);
    for (int i=0; i<n; i++){
      fill(slack,slack+n,INF);
      while (true){
        fill(vx,vx+n,0);
        fill(vy,vy+n,0);
        if ( DFS(i) ) break;
        int d = INF; // long long
        for (int j=0; j<n; j++)
          if (!vy[j]) d = min(d, slack[j]);
        for (int j=0; j<n; j++){
          if (vx[j]) lx[j] -= d;
          if (vy[j]) ly[j] += d;
          else slack[j] -= d;
        }
      }
    }
    int res=0;
    for (int i=0; i<n; i++)
      res += edge[match[i]][i];
    return res;
  }
}graph;
```

2.5 DMST

```
/*
 * Edmond's algoirthm for Directed MST
 * runs in O(VE)
 */
const int MAXV = 10010;
const int MAXE = 10010;
const int INF  = 2147483647;
struct Edge{
  int u, v, c;
  Edge(){}
  Edge(int x, int y, int z) :
    u(x), v(y), c(z){}
};
int V, E, root;
Edge edges[MAXE];
inline int newV(){
  V++;
  return V;
}
inline void addEdge(int u, int v, int c){
  E++;
  edges[E] = Edge(u, v, c);
}
bool con[MAXV];
int mnInW[MAXV], prv[MAXV], cyc[MAXV], vis[MAXV];
inline int DMST(){
  fill(con, con+V+1, 0);
  int r1 = 0, r2 = 0;
  while(1){
    fill(mnInW, mnInW+V+1, INF);
    fill(prv, prv+V+1, -1);
    REP(i, 1, E){
      int u=edges[i].u, v=edges[i].v, c=edges[i].c;
      if(u != v && v != root && c < mnInW[v])
        mnInW[v] = c, prv[v] = u;
    }
    fill(vis, vis+V+1, -1);
    fill(cyc, cyc+V+1, -1);
    r1 = 0;
    bool jf = 0;
    REP(i, 1, V){
      if(con[i]) continue ;
      if(prv[i] == -1 && i != root) return -1;
      if(prv[i] > 0) r1 += mnInW[i];
      int s;
      for(s = i; s != -1 && vis[s] == -1; s = prv[s])
        vis[s] = i;
      if(s > 0 && vis[s] == i){
         // get a cycle
        jf = 1;
        int v = s;
        do{
          cyc[v] = s, con[v] = 1;
          r2 += mnInW[v];
          v = prv[v];
        }while(v != s);
        con[s] = 0;
      }
    }
    if(!jf) break ;
    REP(i, 1, E){
      int &u = edges[i].u;
      int &v = edges[i].v;
      if(cyc[v] > 0) edges[i].c -= mnInW[edges[i].v];
      if(cyc[u] > 0) edges[i].u = cyc[edges[i].u];
      if(cyc[v] > 0) edges[i].v = cyc[edges[i].v];
      if(u == v) edges[i--] = edges[E--];
    }
  }
  return r1+r2;
}
```

2.6 SW min-cut

```
// global min cut
struct SW{ // O(V^3)
  static const int MXN = 514;
  int n,vst[MXN],del[MXN];
  int edge[MXN][MXN],wei[MXN];
  void init(int _n){
    n = _n;
    FZ(edge);
    FZ(del);
  }
  void add_edge(int u, int v, int w){
    edge[u][v] += w;
    edge[v][u] += w;
  }
  void search(int &s, int &t){
    FZ(vst); FZ(wei);
    s = t = -1;
    while (true){
```

```
      int mx=-1, cur=0;
      for (int i=0; i<n; i++)
        if (!del[i] && !vst[i] && mx<wei[i])
          cur = i, mx = wei[i];
      if (mx == -1) break;
      vst[cur] = 1;
      s = t;
      t = cur;
      for (int i=0; i<n; i++)
        if (!vst[i] && !del[i]) wei[i] += edge[cur][i];
    }
  }
  int solve(){
    int res = 2147483647;
    for (int i=0,x,y; i<n-1; i++){
      search(x,y);
      res = min(res,wei[y]);
      del[y] = 1;
      for (int j=0; j<n; j++)
        edge[x][j] = (edge[j][x] += edge[y][j]);
    }
    return res;
  }
}graph;
```

2.7 Max Cost Circulation

```
struct MaxCostCirc {
  static const int MAXN = 33;
  int n , m;
  struct Edge {
    int v , w , c , r;
  };
  vector<Edge> g[ MAXN ];
  int dis[ MAXN ] , prv[ MAXN ] , prve[ MAXN ];
  bool vis[ MAXN ];
  int ans;
  void init( int _n , int _m ) : n(_n), m(_m) {}
  void adde( int u , int v , int w , int c ) {
    g[ u ].push_back( { v , w , c , SZ( g[ v ] ) } );
    g[ v ].push_back( { u , -w , 0 , SZ( g[ u ] )-1 } )
        ;
  }
  bool poscyc() {
    fill( dis , dis+n+1 , 0 );
    fill( prv , prv+n+1 , 0 );
    fill( vis , vis+n+1 , 0 );
    int tmp = -1;
    FOR( t , n+1 ) {
      REP( i , 1 , n ) {
        FOR( j , SZ( g[ i ] ) ) {
          Edge& e = g[ i ][ j ];
          if( e.c && dis[ e.v ] < dis[ i ]+e.w ) {
            dis[ e.v ] = dis[ i ]+e.w;
            prv[ e.v ] = i;
            prve[ e.v ] = j;
            if( t == n ) {
              tmp = i;
              break;
            }
          }
        }
      }
    }
    if( tmp == -1 ) return 0;
    int cur = tmp;
    while( !vis[ cur ] ) {
      vis[ cur ] = 1;
      cur = prv[ cur ];
    }
    int now = cur;
    int cost = 0 , df = 100000;
    do{
      Edge &e = g[ prv[ now ] ][ prve[ now ] ];
      df = min( df , e.c );
      cost += e.w;
      now = prv[ now ];
    }while( now != cur );
    ans += df*cost;
    now = cur;
    do{
      Edge &e = g[ prv[ now ] ][ prve[ now ] ];
      Edge &re = g[ now ][ e.r ];
      e.c -= df;
      re.c += df;
      now = prv[ now ];
    }while( now != cur );
    return 1;
  }
} circ;
```

2.8 Max flow with lower/upper bound

```
// Max flow with lower/upper bound on edges
// source = 1 , sink = n
int in[ N ] , out[ N ];
int l[ M ] , r[ M ] , a[ M ] , b[ M ];
int solve(){
  flow.init( n );
  for( int i = 0 ; i < m ; i ++ ){
    in[ r[ i ] ] += a[ i ];
    out[ l[ i ] ] += a[ i ];
    flow.addEdge( l[ i ] , r[ i ] , b[ i ] - a[ i ] );
    // flow on edge from l[ i ] to r[ i ] should
    // be in [a[ i ], b[ i ]].
  }
  int nd = 0;
  for( int i = 1 ; i <= n ; i ++ ){
    if( in[ i ] < out[ i ] ){
      flow.addEdge( i , flow.t , out[ i ] - in[ i ] );
      nd += out[ i ] - in[ i ];
    }
    if( out[ i ] < in[ i ] )
      flow.addEdge( flow.s , i , in[ i ] - out[ i ] );
  }
  // original sink to source
  flow.addEdge( n , 1 , INF );
  if( flow.maxflow() != nd )
    // no solution
    return -1;
  int ans = flow.G[ 1 ].back().c; // source to sink
  flow.G[ 1 ].back().c = flow.G[ n ].back().c = 0;
  // take out super source and super sink
  for( size_t i = 0 ; i < flow.G[ flow.s ].size() ; i
      ++ ){
    flow.G[ flow.s ][ i ].c = 0;
    Edge &e = flow.G[ flow.s ][ i ];
    flow.G[ e.v ][ e.r ].c = 0;
  }
  for( size_t i = 0 ; i < flow.G[ flow.t ].size() ; i
      ++ ){
    flow.G[ flow.t ][ i ].c = 0;
    Edge &e = flow.G[ flow.t ][ i ];
    flow.G[ e.v ][ e.r ].c = 0;
  }
  flow.addEdge( flow.s , 1 , INF );
  flow.addEdge( n , flow.t , INF );
  flow.reset();
  return ans + flow.maxflow();
}
```

2.9 Flow Method

```
Maximize c^T x subject to Ax ≤ b, x ≥ 0;
with the corresponding symmetric dual problem,
Minimize b^T y subject to A^T y ≥ c, y ≥ 0.

Maximize c^T x subject to Ax ≤ b;
with the corresponding asymmetric dual problem,
Minimize b^T y subject to A^T y = c, y ≥ 0.


Minimum vertex cover on bipartite graph =
Maximum matching on bipartite graph =
Max flow with source to one side, other side to sink

To reconstruct the minimum vertex cover, dfs from each
unmatched vertex on the left side and with unused edges
```

```
only. Equivalently, dfs from source with unused edges
only and without visiting sink. Then, a vertex is
    chosen
iff. it is on the left side and without visited or on
the right side and visited through dfs.


Maximum density subgraph ( \sum W_e + \sum W_v ) / |V|

Binary search on answer:
For a fixed D, construct a Max flow model as follow:
Let S be Sum of all weight( or inf)
1. from source to each node with cap = S
2. For each (u,v,w) in E, (u->v,cap=w), (v->u,cap=w)
3. For each node v, from v to sink with cap = S + 2 * D
     - deg[v] - 2 * (W of v)
where deg[v] = \sum weight of edge associated with v
If maxflow < S * |V|, D is an answer.

Requiring subgraph: all vertex can be reached from
    source with
edge whose cap > 0.
```

3 Math

3.1 FFT

```
// const int MAXN = 262144;
// (must be 2^k)
// before any usage, run pre_fft() first
//
// To implement poly. multiply:
//
// fft( n , a );
// fft( n , b );
// for( int i = 0 ; i < n ; i++ )
//   c[ i ] = a[ i ] * b[ i ];
// fft( n , c , 1 );
//
// then you have the result in c :: [cplx]
typedef long double ld;
typedef complex<ld> cplx;
const ld PI = acosl(-1);
const cplx I(0, 1);
cplx omega[MAXN+1];
void pre_fft(){
  for(int i=0; i<=MAXN; i++)
    omega[i] = exp(i * 2 * PI / MAXN * I);
}
// n must be 2^k
void fft(int n, cplx a[], bool inv=false){
  int basic = MAXN / n;
  int theta = basic;
  for (int m = n; m >= 2; m >>= 1) {
    int mh = m >> 1;
    for (int i = 0; i < mh; i++) {
      cplx w = omega[inv ? MAXN-(i*theta%MAXN)
                         : i*theta%MAXN];
      for (int j = i; j < n; j += m) {
        int k = j + mh;
        cplx x = a[j] - a[k];
        a[j] += a[k];
        a[k] = w * x;
      }
    }
    theta = (theta * 2) % MAXN;
  }
  int i = 0;
  for (int j = 1; j < n - 1; j++) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(a[i], a[j]);
  }
  if (inv)
    for (i = 0; i < n; i++)
      a[i] /= n;
}
```

3.2 NTT

```
typedef long long LL;
// Remember coefficient are mod P
/* p=a*2^n+1
   n     2^n          p           a    root
   5     32           97          3    5
   6     64           193         3    5
   7     128          257         2    3
   8     256          257         1    3
   9     512          7681        15   17
   10    1024         12289       12   11
   11    2048         12289       6    11
   12    4096         12289       3    11
   13    8192         40961       5    3
   14    16384        65537       4    3
   15    32768        65537       2    3
   16    65536        65537       1    3
   17    131072       786433      6    10
   18    262144       786433      3    10 (605028353,
       2308, 3)
   19    524288       5767169     11   3
   20    1048576      7340033     7    3
   21    2097152      23068673    11   3
   22    4194304      104857601   25   3
   23    8388608      167772161   20   3
   24    16777216     167772161   10   3
   25    33554432     167772161   5    3 (1107296257, 33,
       10)
   26    67108864     469762049   7    3
   27    134217728    2013265921  15   31 */
// (must be 2^k)
// To implement poly. multiply:
// NTT<P, root, MAXN> ntt;
// ntt( n , a ); // or ntt.tran( n , a );
// ntt( n , b );
// for( int i = 0 ; i < n ; i++ )
//   c[ i ] = a[ i ] * b[ i ];
// ntt( n , c , 1 );
//
// then you have the result in c :: [LL]
template<LL P, LL root, int MAXN>
struct NTT{
  static LL bigmod(LL a, LL b) {
    LL res = 1;
    for (LL bs = a; b; b >>= 1, bs = (bs * bs) % P) {
      if(b&1) res=(res*bs)%P;
    }
    return res;
  }
  static LL inv(LL a, LL b) {
    if(a==1)return 1;
    return (((LL)(a-inv(b%a,a))*b+1)/a)%b;
  }
  LL omega[MAXN+1];
  NTT() {
    omega[0] = 1;
    LL r = bigmod(root, (P-1)/MAXN);
    for (int i=1; i<=MAXN; i++)
      omega[i] = (omega[i-1]*r)%P;
  }
  // n must be 2^k
  void tran(int n, LL a[], bool inv_ntt=false){
    int basic = MAXN / n;
    int theta = basic;
    for (int m = n; m >= 2; m >>= 1) {
      int mh = m >> 1;
      for (int i = 0; i < mh; i++) {
        LL w = omega[i*theta%MAXN];
        for (int j = i; j < n; j += m) {
          int k = j + mh;
          LL x = a[j] - a[k];
          if (x < 0) x += P;
          a[j] += a[k];
          if (a[j] > P) a[j] -= P;
          a[k] = (w * x) % P;
        }
      }
      theta = (theta * 2) % MAXN;
    }
    int i = 0;
```

```
    for (int j = 1; j < n - 1; j++) {
      for (int k = n >> 1; k > (i ^= k); k >>= 1);
      if (j < i) swap(a[i], a[j]);
    }
    if (inv_ntt) {
      LL ni = inv(n,P);
      reverse( a+1 , a+n );
      for (i = 0; i < n; i++)
        a[i] = (a[i] * ni) % P;
    }
  }
  void operator()(int n, LL a[], bool inv_ntt=false) {
    tran(n, a, inv_ntt);
  }
};
const LL P=2013265921,root=31;
const int MAXN=4194304;
NTT<P, root, MAXN> ntt;
```

3.3 Fast Walsh Transform

```
/* xor convolution:
 * x = (x0,x1) , y = (y0,y1)
 * z = ( x0y0 + x1y1 , x0y1 + x1y0 )
 * =>
 * x' = ( x0+x1 , x0-x1 ) , y' = ( y0+y1 , y0-y1 )
 * z' = ( ( x0+x1 )( y0+y1 ) , ( x0-x1 )( y0-y1 ) )
 * z = (1/2) * z''
 * or convolution:
 * x = (x0, x0+x1), inv = (x0, x1-x0) w/o final div
 * and convolution:
 * x = (x0+x1, x1), inv = (x0-x1, x1) w/o final div */
typedef long long LL;
const int MAXN = (1<<20)+10;
const LL MOD = 1e9+7;
inline LL pw( LL x , LL k ) {
  LL res = 1;
  for( LL bs = x ; k ; k >>= 1, bs = (bs * bs)%MOD ){
    if( k&1 ) res = ( res * bs ) % MOD;
  }
  return res;
}
inline LL inv( LL x ) {
  return pw( x , MOD-2 );
}
inline void fwt( LL x[ MAXN ] , int N , bool inv=0 ) {
  for( int d = 1 ; d < N ; d <<= 1 ) {
    int d2 = d<<1;
    for( int s = 0 ; s < N ; s += d2 ) {
      for( int i = s , j = s+d ; i < s+d ; i++, j++ ){
        LL ta = x[ i ] , tb = x[ j ];
        x[ i ] = ta+tb;
        x[ j ] = ta-tb;
        if( x[ i ] >= MOD ) x[ i ] -= MOD;
        if( x[ j ] < 0 ) x[ j ] += MOD;
      }
    }
  }
  if( inv )
    for( int i = 0 ; i < N ; i++ ) {
      x[ i ] *= inv( N );
      x[ i ] %= MOD;
    }
}
```

3.4 Linear Recurrence

```
LL n, m;
LL dp[ N + N ];
void pre_dp(){
  dp[ 0 ] = 1;
  LL bdr = min( m + m , n );
  for( LL i = 1 ; i <= bdr ; i ++ )
    for( LL j = i - 1 ; j >= max(0LL , i - m) ; j -- )
      dp[ i ] = add( dp[ i ] , dp[ j ] );
}
vector<LL> Mul( vector<LL>& v1, vector<LL>& v2 ){
  int _sz1 = (int)v1.size();
  int _sz2 = (int)v2.size();
  assert( _sz1 == m );
  assert( _sz2 == m );
  vector<LL> _v( m + m );
  for( int i = 0 ; i < m + m ; i ++ ) _v[ i ] = 0;
// expand
  for( int i = 0 ; i < _sz1 ; i ++ )
    for( int j = 0 ; j < _sz2 ; j ++ )
      _v[ i + j + 1 ] = add( _v[ i + j + 1 ] ,
                             mul(v1[ i ] , v2[ j ]) );
// shrink
  for( int i = 0 ; i < m ; i ++ )
    for( int j = 1 ; j <= m ; j ++ )
      _v[ i + j ] = add( _v[ i + j ] , _v[ i ] );
  for( int i = 0 ; i < m ; i ++ )
    _v[ i ] = _v[ i + m ];
  _v.resize( m );
  return _v;
}
vector<LL> I, A;
void solve(){
  pre_dp();
  if( n <= m + m )
  { printf( "%lld\n" , dp[ n ] ); exit( 0 ); }
  I.resize( m ); A.resize( m );
  for( int i = 0 ; i < m ; i ++ ) I[ i ] = A[ i ] = 1;
// dp[ n ] = /Sum_{i=0}^{m-1} A_i * dp[ n - i - 1 ]
  LL dlt = ( n - m ) / m, rdlt = dlt * m;
  while( dlt ){
    if( dlt & 1LL ) I = Mul( I , A );
    A = Mul( A , A );
    dlt >>= 1;
  }
  LL ans = 0;
  for( int i = 0 ; i < m ; i ++ )
    ans = add(ans, mul(I[i], dp[n - i - 1 - rdlt]));
  printf( "%lld\n" , ans );
}
```

3.5 Miller Rabin

```
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n-2] if
// you want to use magic.
bool witness(LL a,LL n,LL u,int t){
  LL x=mypow(a,u,n);
  for(int i=0;i<t;i++) {
    LL nx=mul(x,x,n);
    if(nx==1&&x!=1&&x!=n-1) return 1;
    x=nx;
  }
  return x!=1;
}
bool miller_rabin(LL n,int s=100) {
  // iterate s times of witness on n
  // return 1 if prime, 0 otherwise
  if(n<2) return 0;
  if(!(n&1)) return n == 2;
  LL u=n-1; int t=0;
  // n-1 = u*2^t
  while(!(u&1)) u>>=1, t++;
  while(s--){
    LL a=randll()%(n-1)+1;
    if(witness(a,n,u,t)) return 0;
  }
  return 1;
}
```

3.6 Simplex

```
const int MAXN = 111;
const int MAXM = 111;
const double eps = 1E-10;
double a[MAXN][MAXM], b[MAXN], c[MAXM], d[MAXN][MAXM];
```

```
double x[MAXM];
int ix[MAXN + MAXM]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b,x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
// usage :
// value = simplex(a, b, c, N, M);
double simplex(double a[MAXN][MAXM], double b[MAXN],
               double c[MAXM], int n, int m){
  ++m;
  int r = n, s = m - 1;
  memset(d, 0, sizeof(d));
  for (int i = 0; i < n + m; ++i) ix[i] = i;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
    d[i][m - 1] = 1;
    d[i][m] = b[i];
    if (d[r][m] > d[i][m]) r = i;
  }
  for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
  d[n + 1][m - 1] = -1;
  for (double dd;; ) {
    if (r < n) {
      int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
      d[r][s] = 1.0 / d[r][s];
      for (int j = 0; j <= m; ++j)
        if (j != s) d[r][j] *= -d[r][s];
      for (int i = 0; i <= n + 1; ++i) if (i != r) {
        for (int j = 0; j <= m; ++j) if (j != s)
          d[i][j] += d[r][j] * d[i][s];
        d[i][s] *= d[r][s];
      }
    }
    r = -1; s = -1;
    for (int j = 0; j < m; ++j)
      if (s < 0 || ix[s] > ix[j]) {
        if (d[n + 1][j] > eps ||
            (d[n + 1][j] > -eps && d[n][j] > eps))
          s = j;
      }
    if (s < 0) break;
    for (int i = 0; i < n; ++i) if (d[i][s] < -eps) {
      if (r < 0 ||
          (dd = d[r][m] / d[r][s] - d[i][m] / d[i][s])
              < -eps ||
          (dd < eps && ix[r + m] > ix[i + m]))
        r = i;
    }
    if (r < 0) return -1; // not bounded
  }
  if (d[n + 1][m] < -eps) return -1; // not executable
  double ans = 0;
  for(int i=0; i<m; i++) x[i] = 0;
  for (int i = m; i < n + m; ++i) { // the missing
      enumerated x[i] = 0
    if (ix[i] < m - 1){
      ans += d[i - m][m] * c[ix[i]];
      x[ix[i]] = d[i-m][m];
    }
  }
  return ans;
}
```

3.7 Faulhaber

```
/* faulhaber's formula -
 * cal power sum formula of all p=1~k in O(k^2) */
#define MAXK 2500
const int mod = 1000000007;
int b[MAXK];
// bernoulli number
int inv[MAXK+1];
// inverse
int cm[MAXK+1][MAXK+1]; // combinactories
int co[MAXK][MAXK+2];
// coeeficient of x^j when p=i
inline int getinv(int x) {
  int a=x,b=mod,a0=1,a1=0,b0=0,b1=1;
  while(b) {
    int q,t;
    q=a/b; t=b; b=a-b*q; a=t;
    t=b0; b0=a0-b0*q; a0=t;
    t=b1; b1=a1-b1*q; a1=t;
  }
  return a0<0?a0+mod:a0;
}
inline void pre() {
  /* combinational */
  for(int i=0;i<=MAXK;i++) {
    cm[i][0]=cm[i][i]=1;
    for(int j=1;j<i;j++)
      cm[i][j]=add(cm[i-1][j-1],cm[i-1][j]);
  }
  /* inverse */
  for(int i=1;i<=MAXK;i++) inv[i]=getinv(i);
  /* bernoulli */
  b[0]=1; b[1]=getinv(2); // with b[1] = 1/2
  for(int i=2;i<MAXK;i++) {
    if(i&1) { b[i]=0; continue; }
    b[i]=1;
    for(int j=0;j<i;j++)
      b[i]=sub(b[i], mul(cm[i][j],mul(b[j], inv[i-j+1])
          );
  }
  /* faulhaber */
  // sigma_x=1~n {x^p} = 1/(p+1) * sigma_j=0~p { C(p+1,
      j) * Bj * n^(p-j+1)}
  for(int i=1;i<MAXK;i++) {
    co[i][0]=0;
    for(int j=0;j<=i;j++)
      co[i][i-j+1]= mul(inv[i+1], mul(cm[i+1][j], b[j])
          )
  }
}
/* sample usage: return f(n,p) = sigma_x=1~n (x^p) */
inline int solve(int n,int p) {
  int sol=0,m=n;
  for(int i=1;i<=p+1;i++) {
    sol=add(sol,mul(co[p][i],m));
    m = mul(m, n);
  }
  return sol;
}
```

3.8 Chinese Remainder

```
int pfn;
// number of distinct prime factors
int pf[MAXN]; // prime factor powers
int rem[MAXN]; // corresponding remainder
int pm[MAXN];
inline void generate_primes() {
  int i,j;
  pnum=1;
  prime[0]=2;
  for(i=3;i<MAXVAL;i+=2) {
    if(nprime[i]) continue;
    prime[pnum++]=i;
    for(j=i*i;j<MAXVAL;j+=i) nprime[j]=1;
  }
}
inline int inverse(int x,int p) {
  int q,tmp,a=x,b=p;
  int a0=1,a1=0,b0=0,b1=1;
  while(b) {
    q=a/b; tmp=b; b=a-b*q; a=tmp;
    tmp=b0; b0=a0-b0*q; a0=tmp;
    tmp=b1; b1=a1-b1*q; a1=tmp;
  }
  return a0;
}
inline void decompose_mod() {
  int i,p,t=mod;
  pfn=0;
  for(i=0;i<pnum&&prime[i]<=t;i++) {
    p=prime[i];
    if(t%p==0) {
      pf[pfn]=1;
      while(t%p==0) {
        t/=p;
```

```
      pf[pfn]*=p;
    }
    pfn++;
  }
 }
 if(t>1) pf[pfn++]=t;
}
inline int chinese_remainder() {
 int i,m,s=0;
 for(i=0;i<pfn;i++) {
  m=mod/pf[i];
  pm[i]=(LL)m*inverse(m,pf[i])%mod;
  s=(s+(LL)pm[i]*rem[i])%mod;
 }
 return s;
}
```

3.9 Pollard Rho

```
// does not work when n is prime
LL f(LL x, LL mod){
 return add(mul(x,x,mod),1,mod);
}
LL pollard_rho(LL n) {
 if(!(n&1)) return 2;
 while(true){
  LL y=2, x=rand()%(n-1)+1, res=1;
  for(int sz=2; res==1; sz*=2) {
   for(int i=0; i<sz && res<=1; i++) {
    x = f(x, n);
    res = __gcd(abs(x-y), n);
   }
   y = x;
  }
  if (res!=0 && res!=n) return res;
 }
}
```

3.10 Poly Generator

```
struct PolyGen{
 /* for a nth-order polynomial f(x), *
  * given f(0), f(1), ..., f(n) *
  * express f(x) as sigma_i{c_i*C(x,i)} */
 int n;
 vector<LL> coef;
 // initialize and calculate f(x), vector _fx should
 // be filled with f(0) to f(n)
 PolyGen(int _n,vector<LL> _fx):n(_n),coef(_fx){
  for(int i=0;i<n;i++)
   for(int j=n;j>i;j--)
    coef[j]-=coef[j-1];
 }
 // evaluate f(x), runs in O(n)
 LL eval(int x){
  LL m=1, ret=0;
  for(int i=0;i<=n;i++){
   ret+=coef[i]*m;
   m=m*(x-i)/(i+1);
  }
  return ret;
 }
};
```

3.11 Matrix Pseudo Inverse

```
Mat pinv( Mat m ){
 Mat res = I;
 FZ( used );
 for( int i = 0 ; i < W ; i ++ ){
  int piv = -1;
  for( int j = 0 ; j < W ; j ++ ){
   if( used[ j ] ) continue;
   if( abs( m.v[ j ][ i ] ) > EPS ){
    piv = j;
    break;
   }
  }
  if( piv == -1 ) continue;
  used[ i ] = true;
  swap( m.v[ piv ], m.v[ i ] );
  swap( res.v[ piv ], res.v[ i ] );
  LD rat = m.v[ i ][ i ];
  for( int j = 0 ; j < W ; j ++ ){
   m.v[ i ][ j ] /= rat;
   res.v[ i ][ j ] /= rat;
  }
  for( int j = 0 ; j < W ; j ++ ){
   if( j == i ) continue;
   rat = m.v[ j ][ i ];
   for( int k = 0 ; k < W ; k ++ ){
    m.v[ j ][ k ] -= rat * m.v[ i ][ k ];
    res.v[ j ][ k ] -= rat * res.v[ i ][ k ];
   }
  }
 }
 for( int i = 0 ; i < W ; i ++ ){
  if( used[ i ] ) continue;
  for( int j = 0 ; j < W ; j ++ )
   res.v[ i ][ j ] = 0;
 }
 return res;
}
```

3.12 ax+by=gcd

```
PII gcd(int a, int b){
 if(b == 0) return {1, 0};
 PII q = gcd(b, a % b);
 return {q.second, q.first - q.second * (a / b)};
}
```

3.13 Discrete sqrt

```
void calcH(int &t, int &h, const int p) {
 int tmp=p-1; for(t=0;(tmp&1)==0;tmp/=2) t++; h=tmp;
}
// solve equation x^2 mod p = a
bool solve(int a, int p, int &x, int &y) {
 if(p == 2) { x = y = 1; return true; }
 int p2 = p / 2, tmp = mypow(a, p2, p);
 if (tmp == p - 1) return false;
 if ((p + 1) % 4 == 0) {
  x=mypow(a,(p+1)/4,p); y=p-x; return true;
 } else {
  int t, h, b, pb; calcH(t, h, p);
  if (t >= 2) {
   do {b = rand() % (p - 2) + 2;
   } while (mypow(b, p / 2, p) != p - 1);
   pb = mypow(b, h, p);
  } int s = mypow(a, h / 2, p);
  for (int step = 2; step <= t; step++) {
   int ss = (((LL)(s * s) % p) * a) % p;
   for(int i=0;i<t-step;i++) ss=mul(ss,ss,p);
   if (ss + 1 == p) s = (s * pb) % p;
   pb = ((LL)pb * pb) % p;
  } x = ((LL)s * a) % p; y = p - x;
 } return true;
}
```

3.14 SchreierSims

```
// time: O(n^2 lg^3 |G| + t n lg |G|)
// mem : O(n^2 lg |G| + tn)
// t : number of generator
namespace SchreierSimsAlgorithm{
 typedef vector<int> Permu;
 Permu inv( const Permu& p ){
  Permu ret( p.size() );
  for( int i = 0; i < int(p.size()); i ++ )
   ret[ p[ i ] ] = i;
  return ret;
```

```
}
Permu operator*( const Permu& a, const Permu& b ){
  Permu ret( a.size() );
  for( int i = 0 ; i < (int)a.size(); i ++ )
    ret[ i ] = b[ a[ i ] ];
  return ret;
}
typedef vector<Permu> Bucket;
typedef vector<int> Table;
typedef pair<int,int> pii;
int n, m;
vector<Bucket> bkts, bktsInv;
vector<Table> lookup;
int fastFilter( const Permu &g, bool addToG = 1 ){
  n = bkts.size();
  Permu p;
  for( int i = 0 ; i < n ; i ++ ){
    int res = lookup[ i ][ p[ i ] ];
    if( res == -1 ){
      if( addToG ){
        bkts[ i ].push_back( p );
        bktsInv[ i ].push_back( inv( p ) );
        lookup[ i ][ p[i] ] = (int)bkts[i].size()-1;
      }
      return i;
    }
    p = p * bktsInv[i][res];
  }
  return -1;
}
long long calcTotalSize(){
  long long ret = 1;
  for( int i = 0 ; i < n ; i ++ )
    ret *= bkts[i].size();
  return ret;
}
bool inGroup( const Permu &g ){
  return fastFilter( g, false ) == -1;
}
void solve( const Bucket &gen, int _n ){
  n = _n, m = gen.size(); // m perm[0..n-1]s
  {//clear all
    bkts.clear();
    bktsInv.clear();
    lookup.clear();
  }
  for(int i = 0 ; i < n ; i ++ ){
    lookup[i].resize(n);
    fill(lookup[i].begin(), lookup[i].end(), -1);
  }
  Permu id( n );
  for(int i = 0 ; i < n ; i ++ ) id[i] = i;
  for(int i = 0 ; i < n ; i ++ ){
    bkts[i].push_back(id);
    bktsInv[i].push_back(id);
    lookup[i][i] = 0;
  }
  for(int i = 0 ; i < m ; i ++)
    fastFilter( gen[i] );
  queue< pair<pii,pii> > toUpd;
  for(int i = 0; i < n; i ++)
    for(int j = i; j < n; j ++)
      for(int k = 0; k < (int)bkts[i].size(); k ++)
        for(int l = 0; l < (int)bkts[j].size(); l ++)
          toUpd.push( {pii(i,k), pii(j,l)} );
  while( !toUpd.empty() ){
    pii a = toUpd.front().first;
    pii b = toUpd.front().second;
    toUpd.pop();
    int res = fastFilter(bkts[a.first][a.second] *
                         bkts[b.first][b.second]);
    if(res == -1) continue;
    pii newPair(res, (int)bkts[res].size() - 1);
    for(int i = 0; i < n; i ++)
      for(int j = 0; j < (int)bkts[i].size(); ++j){
        if(i <= res)
          toUpd.push(make_pair(pii(i , j), newPair));
        if(res <= i)
          toUpd.push(make_pair(newPair, pii(i, j)));
      }
  }
}
```

3.15 Prefix Inverse

```
void solve( int m ){
  inv[ 1 ] = 1;
  for( int i = 2 ; i < m ; i ++ )
    inv[ i ] = ((LL)(m - m / i) * inv[m % i]) % m;
}
```

3.16 Roots of Polynomial

```
const double eps = 1e-12;
const double inf = 1e+12;
double a[ 10 ], x[ 10 ];
int n;
int sign( double x ){
  return (x < -eps)?(-1):(x>eps);
}
double f(double a[], int n, double x){
  double tmp=1,sum=0;
  for(int i=0;i<=n;i++){
    sum=sum+a[i]*tmp;
    tmp=tmp*x;
  }
  return sum;
}
double binary(double l,double r,double a[],int n){
  int sl=sign(f(a,n,l)),sr=sign(f(a,n,r));
  if(sl==0) return l;
  if(sr==0) return r;
  if(sl*sr>0) return inf;
  while(r-l>eps){
    double mid=(l+r)/2;
    int ss=sign(f(a,n,mid));
    if(ss==0) return mid;
    if(ss*sl>0) l=mid; else r=mid;
  }
  return l;
}
void solve(int n,double a[],double x[],int &nx){
  if(n==1){
    x[1]=-a[0]/a[1];
    nx=1;
    return;
  }
  double da[10], dx[10];
  int ndx;
  for(int i=n;i>=1;i--) da[i-1]=a[i]*i;
  solve(n-1,da,dx,ndx);
  nx=0;
  if(ndx==0){
    double tmp=binary(-inf,inf,a,n);
    if (tmp<inf) x[++nx]=tmp;
    return;
  }
  double tmp;
  tmp=binary(-inf,dx[1],a,n);
  if(tmp<inf) x[++nx]=tmp;
  for(int i=1;i<=ndx-1;i++){
    tmp=binary(dx[i],dx[i+1],a,n);
    if(tmp<inf) x[++nx]=tmp;
  }
  tmp=binary(dx[ndx],inf,a,n);
  if(tmp<inf) x[++nx]=tmp;
}
int main() {
  scanf("%d",&n);
  for(int i=n;i>=0;i--) scanf("%lf",&a[i]);
  int nx;
  solve(n,a,x,nx);
  for(int i=1;i<=nx;i++) printf("%.6f\n",x[i]);
}
```

3.17 Mod

```
/// _fd(a,b)  floor(a/b).
/// _rd(a,m)  a-floor(a/m)*m.
/// _pv(a,m,r) largest x s.t x<=a && x%m == r.
/// _nx(a,m,r) smallest x s.t x>=a && x%m == r.
/// _ct(a,b,m,r) |A| , A = { x : a<=x<=b && x%m == r }.
int _fd(int a,int b){ return a<0?(-~a/b-1):a/b; }
int _rd(int a,int m){ return a-_fd(a,m)*m; }
int _pv(int a,int m,int r){
    r=(r%m+m)%m;
    return _fd(a-r,m)*m+r;
}
int _nt(int a,int m,int r){
    m=abs(m);
    r=(r%m+m)%m;
    return _fd(a-r-1,m)*m+r+m;
}
int _ct(int a,int b,int m,int r){
    m=abs(m);
    a=_nt(a,m,r);
    b=_pv(b,m,r);
    return (a>b)?0:((b-a+m)/m);
}
```

3.18 Primes and μ function

```
/* 12721, 13331, 14341, 75577, 123457, 222557, 556679
* 999983, 1097774749, 1076767633, 100102021, 999997771
* 1001010013, 1000512343, 987654361, 999991231
* 999888733, 98789101, 987777733, 999991921, 1010101333
* 1010102101, 1000000000039, 1000000000000037
* 2305843009213693951, 4611686018427387847
* 9223372036854775783, 18446744073709551557 */
int mu[ N ] , p_tbl[ N ];
vector<int> primes;
void sieve() {
  mu[ 1 ] = p_tbl[ 1 ] = 1;
  for( int i = 2 ; i < N ; i ++ ){
    if( !p_tbl[ i ] ){
      p_tbl[ i ] = i;
      primes.push_back( i );
      mu[ i ] = -1;
    }
    for( int p : primes ){
      int x = i * p;
      if( x >= M ) break;
      p_tbl[ x ] = p;
      mu[ x ] = -mu[ i ];
      if( i % p == 0 ){
        mu[ x ] = 0;
        break;
      }
    }
  }
}
vector<int> factor( int x ){
  vector<int> fac{ 1 };
  while( x > 1 ){
    int fn = SZ(fac), p = p_tbl[ x ], pos = 0;
    while( x % p == 0 ){
      x /= p;
      for( int i = 0 ; i < fn ; i ++ )
        fac.PB( fac[ pos ++ ] * p );
    }
  }
  return fac;
}
```

3.19 Result

```
/*
Lucas' Theorem:
  For non-negative integer n,m and prime P,
  C(m,n) mod P = C(m/M,n/M) * C(m%M,n%M) mod P
  = mult_i ( C(m_i,n_i) )
  where m_i is the i-th digit of m in base P.

Pick's Theorem
  A = i + b/2 - 1

Kirchhoff's theorem
  A_{ii} = deg(i), A_{ij} = (i,j) \in E ? -1 : 0
  Deleting any one row, one column, and cal the det(A)
*/
```

4 Geometry

4.1 halfPlaneIntersection

4.2 Intersection of 2 lines

```
Pt interPnt( Line l1, Line l2, bool &res ){
  Pt p1, p2, q1, q2;
  tie(p1, p2) = l1;
  tie(q1, q2) = l2;
  double f1 = (p2 - p1) ^ (q1 - p1);
  double f2 = (p2 - p1) ^ (p1 - q2);
  double f = (f1 + f2);
  if( fabs(f) < eps)
  { res = false; return {0, 0}; }
  res = true;
  return q1 * (f2 / f) + q2 * (f1 / f);
}
bool isin( Line l0, Line l1, Line l2 ){
  // Check inter(l1, l2) in l0
  bool res;
  Pt p = interPnt(l1, l2, res);
  return ( (l0.SE - l0.FI) ^ (p - l0.FI) ) > eps;
}
/* If no solution, check: 1. ret.size() < 3
 * Or more precisely, 2. interPnt(ret[0], ret[1])
 * in all the lines. (use (l.S - l.F) ^ (p - l.F) > 0
 */
/* --^-- Line.FI --^-- Line.SE --^-- */
vector<Line> halfPlaneInter( vector<Line> lines ){
  int sz = lines.size();
  vector<double> ata(sz), ord(sz);
  for( int i=0; i<sz; i++) {
    ord[i] = i;
    Pt d = lines[i].SE - lines[i].FI;
    ata[i] = atan2(d.Y, d.X);
  }
  sort( ord.begin(), ord.end(), [&](int i, int j) {
    if( fabs(ata[i] - ata[j]) < eps )
      return ( (lines[i].SE - lines[i].FI) ^
               (lines[j].SE - lines[i].FI) ) < 0;
    return ata[i] < ata[j];
  });
  vector<Line> fin;
  for (int i=0; i<sz; i++)
    if (!i or fabs(ata[ord[i]] - ata[ord[i-1]]) > eps)
      fin.PB(lines[ord[i]]);
  deque<Line> dq;
  for (int i=0; i<(int)(fin.size()); i++) {
    while((int)(dq.size()) >= 2 and
        not isin(fin[i], dq[(int)(dq.size())-2],
                         dq[(int)(dq.size())-1]))
      dq.pop_back();
    while((int)(dq.size()) >= 2 and
        not isin(fin[i], dq[0], dq[1]))
      dq.pop_front();
    dq.push_back(fin[i]);
  }
  while( (int)(dq.size()) >= 3 and
      not isin(dq[0], dq[(int)(dq.size())-2],
                      dq[(int)(dq.size())-1]))
    dq.pop_back();
  while( (int)(dq.size()) >= 3 and
      not isin(dq[(int)(dq.size())-1], dq[0], dq[1]))
    dq.pop_front();
  vector<Line> res(dq.begin(),dq.end());
  return res;
}
```

4.3 Intersection of 2 segments

```
int ori( const PLL& o , const PLL& a , const PLL& b ){
  LL ret = ( a - o ) ^ ( b - o );
  return ret / max( 1ll , abs( ret ) );
}
// p1 == p2 || q1 == q2 need to be handled
bool banana( const PLL& p1 , const PLL& p2 ,
             const PLL& q1 , const PLL& q2 ){
  if( ( ( p2 - p1 ) ^ ( q2 - q1 ) ) == 0 ){ // parallel
    if( ori( p1 , p2 , q1 ) ) return false;
    return ( ( p1 - q1 ) * ( p2 - q1 ) ) <= 0 ||
           ( ( p1 - q2 ) * ( p2 - q2 ) ) <= 0 ||
           ( ( q1 - p1 ) * ( q2 - p1 ) ) <= 0 ||
           ( ( q1 - p2 ) * ( q2 - p2 ) ) <= 0;
  }
  return (ori( p1, p2, q1 ) * ori( p1, p2, q2 )<=0) &&
         (ori( q1, q2, p1 ) * ori( q1, q2, p2 )<=0);
}
```

4.4 Intersection of 2 circles

4.5 Circle cover

```
#define N 1021
struct CircleCover{
  int C; Circle c[ N ];
  bool g[ N ][ N ], overlap[ N ][ N ];
  // Area[i] : area covered by at least i circles
  D Area[ N ];
  void init( int _C ){ C = _C; }
  bool CCinter( Circle& a , Circle& b , Pt& p1 , Pt& p2
      ){
    Pt o1 = a.O , o2 = b.O;
    D r1 = a.R , r2 = b.R;
    D d2 = ( o1 - o2 ) * ( o1 - o2 );
    D d = sqrt(d2);
    if( d > r1 + r2 ) return false;
    Pt u = (o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2))
        ;
    D A = sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(-r1+r2+d)
        );
    Pt v = Pt( o1.Y-o2.Y , -o1.X + o2.X ) * A / (2*d2);
    p1 = u + v; p2 = u - v;
    return true;
  }
  struct Tevent {
    Pt p; D ang; int add;
    Tevent() {}
    Tevent(Pt _a, D _b, int _c): p(_a), ang(_b), add(_c
        ) {}
    bool operator<(const Tevent &a)const
    {return ang < a.ang;}
  }eve[ N * 2 ];
  // strict: x = 0, otherwise x = -1
  bool disjuct( Circle& a, Circle &b, int x ){
    return sign( norm( a.O - b.O ) - a.R - b.R ) > x;
  }
  bool contain( Circle& a, Circle &b, int x ){
    return sign( a.R - b.R - norm( a.O - b.O ) ) > x;
  }
  bool contain(int i, int j){ /* c[j] is non-strictly
      in c[i]. */
    return (sign(c[i].R - c[j].R) > 0 ||
            (sign(c[i].R - c[j].R) == 0 && i < j) ) &&
                contain(c[i], c[j], -1);
  }
  void solve(){
    for( int i = 0 ; i <= C + 1 ; i ++ )
      Area[ i ] = 0;
    for( int i = 0 ; i < C ; i ++ )
      for( int j = 0 ; j < C ; j ++ )
        overlap[i][j] = contain(i, j);
    for( int i = 0 ; i < C ; i ++ )
      for( int j = 0 ; j < C ; j ++ )
        g[i][j] = !(overlap[i][j] || overlap[j][i] ||
                    disjuct(c[i], c[j], -1));
    for( int i = 0 ; i < C ; i ++ ){
      int E = 0, cnt = 1;
      for( int j = 0 ; j < C ; j ++ )
        if( j != i && overlap[j][i] )
          cnt ++;
      for( int j = 0 ; j < C ; j ++ )
        if( i != j && g[i][j] ){
          Pt aa, bb;
          CCinter(c[i], c[j], aa, bb);
          D A = atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X)
              ;
          D B = atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X)
              ;
          eve[E ++] = Tevent(bb, B, 1);
          eve[E ++] = Tevent(aa, A, -1);
          if(B > A) cnt ++;
        }
      if( E == 0 ) Area[ cnt ] += pi * c[i].R * c[i].R;
      else{
        sort( eve , eve + E );
        eve[E] = eve[0];
        for( int j = 0 ; j < E ; j ++ ){
          cnt += eve[j].add;
          Area[cnt] += (eve[j].p ^ eve[j + 1].p) * .5;
          D theta = eve[j + 1].ang - eve[j].ang;
          if (theta < 0) theta += 2. * pi;
          Area[cnt] += ( theta - sin(theta) ) * c[i].R
              * c[i].R * .5;
        }
      }
    }
  }
};
```

4.6 Convex Hull trick

```
/* Given a convexhull, answer querys in O(\lg N)
CH should not contain identical points, the area should
be > 0, min pair(x, y) should be listed first */
double det( const Pt& p1 , const Pt& p2 )
{ return p1.X * p2.Y - p1.Y * p2.X; }
struct Conv{
  int n;
  vector<Pt> a;
  vector<Pt> upper, lower;
  Conv(vector<Pt> _a) : a(_a){
    n = a.size();
    int ptr = 0;
    for(int i=1; i<n; ++i) if (a[ptr] < a[i]) ptr = i;
    for(int i=0; i<=ptr; ++i) lower.push_back(a[i]);
    for(int i=ptr; i<n; ++i) upper.push_back(a[i]);
    upper.push_back(a[0]);
  }
  int sign( LL x ){ // fixed when changed to double
    return x < 0 ? -1 : x > 0 }
  pair<LL,int> get_tang(vector<Pt> &conv, Pt vec){
    int l = 0, r = (int)conv.size() - 2;
    for( ; l + 1 < r; ){
      int mid = (l + r) / 2;
      if(sign(det(conv[mid+1]-conv[mid],vec))>0)r=mid;
      else l = mid;
    }
    return max(make_pair(det(vec, conv[r]), r),
               make_pair(det(vec, conv[0]), 0));
  }
  void upd_tang(const Pt &p, int id, int &i0, int &i1){
    if(det(a[i0] - p, a[id] - p) > 0) i0 = id;
    if(det(a[i1] - p, a[id] - p) < 0) i1 = id;
  }
  void bi_search(int l, int r, Pt p, int &i0, int &i1){
    if(l == r) return;
    upd_tang(p, l % n, i0, i1);
    int sl=sign(det(a[l % n] - p, a[(l + 1) % n] - p));
    for( ; l + 1 < r; ) {
      int mid = (l + r) / 2;
      int smid=sign(det(a[mid%n]-p, a[(mid+1)%n]-p));
      if (smid == sl) l = mid;
      else r = mid;
    }
    upd_tang(p, r % n, i0, i1);
  }
  int bi_search(Pt u, Pt v, int l, int r) {
    int sl = sign(det(v - u, a[l % n] - u));
    for( ; l + 1 < r; ) {
      int mid = (l + r) / 2;
```

```
      int smid = sign(det(v - u, a[mid % n] - u));
      if (smid == sl) l = mid;
      else r = mid;
    }
    return l % n;
  }
  // 1. whether a given point is inside the CH
  bool contain(Pt p) {
    if (p.X < lower[0].X || p.X > lower.back().X)
        return 0;
    int id = lower_bound(lower.begin(), lower.end(), Pt
        (p.X, -INF)) - lower.begin();
    if (lower[id].X == p.X) {
      if (lower[id].Y > p.Y) return 0;
    }else if(det(lower[id-1]-p,lower[id]-p)<0)return 0;
    id = lower_bound(upper.begin(), upper.end(), Pt(p.X
        , INF), greater<Pt>()) - upper.begin();
    if (upper[id].X == p.X) {
      if (upper[id].Y < p.Y) return 0;
    }else if(det(upper[id-1]-p,upper[id]-p)<0)return 0;
    return 1;
  }
  // 2. Find 2 tang pts on CH of a given outside point
  // return true with i0, i1 as index of tangent points
  // return false if inside CH
  bool get_tang(Pt p, int &i0, int &i1) {
    if (contain(p)) return false;
    i0 = i1 = 0;
    int id = lower_bound(lower.begin(), lower.end(), p)
        - lower.begin();
    bi_search(0, id, p, i0, i1);
    bi_search(id, (int)lower.size(), p, i0, i1);
    id = lower_bound(upper.begin(), upper.end(), p,
        greater<Pt>()) - upper.begin();
    bi_search((int)lower.size() - 1, (int)lower.size()
        - 1 + id, p, i0, i1);
    bi_search((int)lower.size() - 1 + id, (int)lower.
        size() - 1 + (int)upper.size(), p, i0, i1);
    return true;
  }
  // 3. Find tangent points of a given vector
  // ret the idx of vertex has max cross value with vec
  int get_tang(Pt vec){
    pair<LL, int> ret = get_tang(upper, vec);
    ret.second = (ret.second+(int)lower.size()-1)%n;
    ret = max(ret, get_tang(lower, vec));
    return ret.second;
  }
  // 4. Find intersection point of a given line
  // return 1 and intersection is on edge (i, next(i))
  // return 0 if no strictly intersection
  bool get_intersection(Pt u, Pt v, int &i0, int &i1){
   int p0 = get_tang(u - v), p1 = get_tang(v - u);
   if(sign(det(v-u,a[p0]-u))*sign(det(v-u,a[p1]-u))<0){
     if (p0 > p1) swap(p0, p1);
     i0 = bi_search(u, v, p0, p1);
     i1 = bi_search(u, v, p1, p0 + n);
     return 1;
   }
   return 0;
  }
};
```

4.7 Tangent line of two circles

```
vector<Line> go( const Circle& c1 , const Circle& c2 ){
  vector<Line> ret;
  double d_sq = norm2( c1.O - c2.O );
  if( d_sq < eps ) return ret;
  double d = sqrt( d_sq );
  Pt v = ( c2.O - c1.O ) / d;
  for( int sign1 = 1 ; sign1 >= -1 ; sign1 -= 2 ){
    double c = ( c1.R - sign1 * c2.R ) / d;
    if( c * c > 1 ) continue;
    double h = sqrt( max( 0.0 , 1.0 - c * c ) );
    for( int sign2 = 1 ; sign2 >= -1 ; sign2 -= 2 ){
      Pt n;
      n.X = v.X * c - sign2 * h * v.Y;
      n.Y = v.Y * c + sign2 * h * v.X;
      Pt p1 = c1.O + n * c1.R;
      Pt p2 = c2.O + n * ( c2.R * sign1 );
      if( fabs( p1.X - p2.X ) < eps and
          fabs( p1.Y - p2.Y ) < eps )
        p2 = p1 + perp( c2.O - c1.O );
      ret.push_back( { p1 , p2 } );
    }
  }
  return ret;
}
```

4.8 KD Tree

```
const int MXN = 100005;

struct KDTree {
  struct Node {
    int x,y,x1,y1,x2,y2;
    int id,f;
    Node *L, *R;
  }tree[MXN];
  int n;
  Node *root;

  long long dis2(int x1, int y1, int x2, int y2) {
    long long dx = x1-x2;
    long long dy = y1-y2;
    return dx*dx+dy*dy;
  }
  static bool cmpx(Node& a, Node& b){ return a.x<b.x; }
  static bool cmpy(Node& a, Node& b){ return a.y<b.y; }
  void init(vector<pair<int,int>> ip) {
    n = ip.size();
    for (int i=0; i<n; i++) {
      tree[i].id = i;
      tree[i].x = ip[i].first;
      tree[i].y = ip[i].second;
    }
    root = build_tree(0, n-1, 0);
  }
  Node* build_tree(int L, int R, int dep) {
    if (L>R) return nullptr;
    int M = (L+R)/2;
    tree[M].f = dep%2;
    nth_element(tree+L, tree+M, tree+R+1, tree[M].f ?
        cmpy : cmpx);
    tree[M].x1 = tree[M].x2 = tree[M].x;
    tree[M].y1 = tree[M].y2 = tree[M].y;

    tree[M].L = build_tree(L, M-1, dep+1);
    if (tree[M].L) {
      tree[M].x1 = min(tree[M].x1, tree[M].L->x1);
      tree[M].x2 = max(tree[M].x2, tree[M].L->x2);
      tree[M].y1 = min(tree[M].y1, tree[M].L->y1);
      tree[M].y2 = max(tree[M].y2, tree[M].L->y2);
    }

    tree[M].R = build_tree(M+1, R, dep+1);
    if (tree[M].R) {
      tree[M].x1 = min(tree[M].x1, tree[M].R->x1);
      tree[M].x2 = max(tree[M].x2, tree[M].R->x2);
      tree[M].y1 = min(tree[M].y1, tree[M].R->y1);
      tree[M].y2 = max(tree[M].y2, tree[M].R->y2);
    }

    return tree+M;
  }
  int touch(Node* r, int x, int y, long long d2){
    long long dis = sqrt(d2)+1;
    if (x<r->x1-dis || x>r->x2+dis || y<r->y1-dis || y>
        r->y2+dis)
      return 0;
    return 1;
  }
  void nearest(Node* r, int x, int y, int &mID, long
      long &md2) {
    if (!r || !touch(r, x, y, md2)) return;
    long long d2 = dis2(r->x, r->y, x, y);
    if (d2 < md2 || (d2 == md2 && mID < r->id)) {
      mID = r->id;
      md2 = d2;
```

```
    }
    // search order depends on split dim
    if ((r->f == 0 && x < r->x) ||
        (r->f == 1 && y < r->y)) {
      nearest(r->L, x, y, mID, md2);
      nearest(r->R, x, y, mID, md2);
    } else {
      nearest(r->R, x, y, mID, md2);
      nearest(r->L, x, y, mID, md2);
    }
  }
  int query(int x, int y) {
    int id = 1029384756;
    long long d2 = 10293847561234567 8LL;
    nearest(root, x, y, id, d2);
    return id;
  }
}tree;
```

4.9 Lower Concave Hull

```
/****
  maintain a "concave hull" that support the following
  1. insertion of a line
  2. query of height(y) on specific x on the hull
 ****/
/* set as needed */
typedef long double LD;
const LD eps=1e-9;
const LD inf=1e19;
class Seg {
 public:
  LD m,c,x1,x2; // y=mx+c
  bool flag;
  Seg(
    LD _m,LD _c,LD _x1=-inf,LD _x2=inf,bool _flag=0)
    :m(_m),c(_c),x1(_x1),x2(_x2),flag(_flag) {}
  LD evaly(LD x) const {
    return m*x+c;
  }
  const bool operator<(LD x) const {
    return x2-eps<x;
  }
  const bool operator<(const Seg &b) const {
    if(flag||b.flag) return *this<b.x1;
    return m+eps<b.m;
  }
};
class LowerConcaveHull { // maintain a hull like: \__/
 public:
  set<Seg> hull;
  /* functions */
  LD xintersection(Seg a,Seg b) {
    return (a.c-b.c)/(b.m-a.m);
  }
  inline set<Seg>::iterator replace(set<Seg> &
      hull,set<Seg>::iterator it,Seg s) {
    hull.erase(it);
    return hull.insert(s).first;
  }
  void insert(Seg s) {
    // insert a line and update hull
    set<Seg>::iterator it=hull.find(s);
    // check for same slope
    if(it!=hull.end()) {
      if(it->c+eps>=s.c) return;
      hull.erase(it);
    }
    // check if below whole hull
    it=hull.lower_bound(s);
    if(it!=hull.end()&&
       s.evaly(it->x1)<=it->evaly(it->x1)+eps) return;
    // update right hull
    while(it!=hull.end()) {
      LD x=xintersection(s,*it);
      if(x>=it->x2-eps) hull.erase(it++);
      else {
        s.x2=x;
        it=replace(hull,it,Seg(it->m,it->c,x,it->x2));
        break;
      }
    }
    // update left hull
    while(it!=hull.begin()) {
      LD x=xintersection(s,*(--it));
      if(x<=it->x1+eps) hull.erase(it++);
      else {
        s.x1=x;
        it=replace(hull,it,Seg(it->m,it->c,it->x1,x));
        break;
      }
    }
    // insert s
    hull.insert(s);
  }
  void insert(LD m,LD c) { insert(Seg(m,c)); }
  LD query(LD x) { // return y @ given x
    set<Seg>::iterator it =
      hull.lower_bound(Seg(0.0,0.0,x,x,1));
    return it->evaly(x);
  }
};
```

4.10 Delaunay Triangulation

```
/* Delaunay Triangulation:
Given a sets of points on 2D plane, find a
triangulation such that no points will strictly
inside circumcircle of any triangle.

find : return a triangle contain given point
add_point : add a point into triangulation

A Triangle is in triangulation iff. its has_chd is 0.
Region of triangle u: iterate each u.edge[i].tri,
each points are u.p[(i+1)%3], u.p[(i+2)%3]

calculation involves O(|V|^6) */
const int N = 100000 + 5;
const type inf = 2e3;
type eps = 1e-6; // 0 when integer
type sqr(type x) { return x*x; }
// return p4 is in circumcircle of tri(p1,p2,p3)
bool in_cc(const Pt& p1, const Pt& p2, const Pt& p3,
    const Pt& p4){
  type u11 = p1.X - p4.X; type u12 = p1.Y - p4.Y;
  type u21 = p2.X - p4.X; type u22 = p2.Y - p4.Y;
  type u31 = p3.X - p4.X; type u32 = p3.Y - p4.Y;
  type u13 = sqr(p1.X)-sqr(p4.X)+sqr(p1.Y)-sqr(p4.Y);
  type u23 = sqr(p2.X)-sqr(p4.X)+sqr(p2.Y)-sqr(p4.Y);
  type u33 = sqr(p3.X)-sqr(p4.X)+sqr(p3.Y)-sqr(p4.Y);
  type det = -u13*u22*u31 + u12*u23*u31 + u13*u21*u32
             -u11*u23*u32 - u12*u21*u33 + u11*u22*u33;
  return det > eps;
}
type side(const Pt& a, const Pt& b, const Pt& p)
{ return (b - a) ^ (p - a); }
typedef int SdRef;
struct Tri;
typedef Tri* TriRef;
struct Edge {
  TriRef tri; SdRef side;
  Edge():tri(0), side(0){}
  Edge(TriRef _tri, SdRef _side):tri(_tri), side(_side)
      {}
};
struct Tri {
  Pt p[3];
  Edge edge[3];
  TriRef chd[3];
  Tri() {}
  Tri(const Pt& p0, const Pt& p1, const Pt& p2) {
    p[0] = p0; p[1] = p1; p[2] = p2;
    chd[0] = chd[1] = chd[2] = 0;
  }
  bool has_chd() const { return chd[0] != 0; }
  int num_chd() const {
    return chd[0] == 0 ? 0
         : chd[1] == 0 ? 1
         : chd[2] == 0 ? 2 : 3;
```

```
  }
  bool contains(Pt const& q) const {
    for( int i = 0 ; i < 3 ; i ++ )
      if( side(p[i], p[(i + 1) % 3] , q) < -eps )
        return false;
    return true;
  }
} pool[ N * 10 ], *tris;
void edge( Edge a, Edge b ){
  if(a.tri) a.tri->edge[a.side] = b;
  if(b.tri) b.tri->edge[b.side] = a;
}
struct Trig { // Triangulation
  Trig(){
    the_root = // Tri should at least contain all
        points
      new(tris++)Tri(Pt(-inf,-inf),Pt(+inf+inf,-inf),Pt
          (-inf,+inf+inf));
  }
  TriRef find(Pt p)const{ return find(the_root,p); }
  void add_point(const Pt& p){ add_point(find(the_root,
      p),p); }
  TriRef the_root;
  static TriRef find(TriRef root, const Pt& p) {
    while( true ){
      if( !root->has_chd() )
        return root;
      for( int i = 0; i < 3 && root->chd[i] ; ++i )
        if (root->chd[i]->contains(p)) {
          root = root->chd[i];
          break;
        }
    }
    assert( false ); // "point not found"
  }
  void add_point(TriRef root, Pt const& p) {
    TriRef tab,tbc,tca;
    /* split it into three triangles */
    tab=new(tris++) Tri(root->p[0],root->p[1],p);
    tbc=new(tris++) Tri(root->p[1],root->p[2],p);
    tca=new(tris++) Tri(root->p[2],root->p[0],p);
    edge(Edge(tab,0), Edge(tbc,1));
    edge(Edge(tbc,0), Edge(tca,1));
    edge(Edge(tca,0), Edge(tab,1));
    edge(Edge(tab,2), root->edge[2]);
    edge(Edge(tbc,2), root->edge[0]);
    edge(Edge(tca,2), root->edge[1]);
    root->chd[0] = tab;
    root->chd[1] = tbc;
    root->chd[2] = tca;
    flip(tab,2);
    flip(tbc,2);
    flip(tca,2);
  }
  void flip(TriRef tri, SdRef pi) {
    TriRef trj = tri->edge[pi].tri;
    int pj = tri->edge[pi].side;
    if (!trj) return;
    if (!in_cc(tri->p[0],tri->p[1],tri->p[2],trj->p[pj
        ])) return;
    /* flip edge between tri,trj */
    TriRef trk = new(tris++) Tri(tri->p[(pi+1)%3], trj
        ->p[pj], tri->p[pi]);
    TriRef trl = new(tris++) Tri(trj->p[(pj+1)%3], tri
        ->p[pi], trj->p[pj]);
    edge(Edge(trk,0), Edge(trl,0));
    edge(Edge(trk,1), tri->edge[(pi+2)%3]);
    edge(Edge(trk,2), trj->edge[(pj+1)%3]);
    edge(Edge(trl,1), trj->edge[(pj+2)%3]);
    edge(Edge(trl,2), tri->edge[(pi+1)%3]);
    tri->chd[0]=trk; tri->chd[1]=trl; tri->chd[2]=0;
    trj->chd[0]=trk; trj->chd[1]=trl; trj->chd[2]=0;
    flip(trk,1); flip(trk,2);
    flip(trl,1); flip(trl,2);
  }
};
vector<TriRef> triang;
set<TriRef> vst;
void go( TriRef now ){
  if( vst.find( now ) != vst.end() )
    return;
  vst.insert( now );
  if( !now->has_chd() ){
    triang.push_back( now );
    return;
  }
  for( int i = 0 ; i < now->num_chd() ; i ++ )
    go( now->chd[ i ] );
}
void build( int n , Pt* ps ){
  tris = pool;
  random_shuffle(ps, ps + n);
  Trig tri;
  for(int i = 0; i < n; ++ i)
    tri.add_point(ps[i]);
  go( tri.the_root );
}
```

4.11 Min Enclosing Circle

```
struct Mec{
  // return pair of center and r
  static const int N = 101010;
  int n;
  Pt p[ N ], cen;
  double r2;
  void init( int _n , Pt _p[] ){
    n = _n;
    memcpy( p , _p , sizeof(Pt) * n );
  }
  double sqr(double a){ return a*a; }
  Pt center(Pt p0, Pt p1, Pt p2) {
    Pt a = p1-p0;
    Pt b = p2-p0;
    double c1=norm2( a ) * 0.5;
    double c2=norm2( b ) * 0.5;
    double d = a ^ b;
    double x = p0.X + (c1 * b.Y - c2 * a.Y) / d;
    double y = p0.Y + (a.X * c2 - b.X * c1) / d;
    return Pt(x,y);
  }
  pair<Pt,double> solve(){
    random_shuffle(p,p+n);
    r2=0;
    for (int i=0; i<n; i++){
      if (norm2(cen-p[i]) <= r2) continue;
      cen = p[i];
      r2 = 0;
      for (int j=0; j<i; j++){
        if (norm2(cen-p[j]) <= r2) continue;
        cen=Pt((p[i].X+p[j].X)/2,(p[i].Y+p[j].Y)/2);
        r2 = norm2(cen-p[j]);
        for (int k=0; k<j; k++){
          if (norm2(cen-p[k]) <= r2) continue;
          cen = center(p[i],p[j],p[k]);
          r2 = norm2(cen-p[k]);
        }
      }
    }
    return {cen,sqrt(r2)};
  }
} mec;
```

4.12 Min Enclosing Ball

```
// Pt : { x , y , z }
#define N 202020
int n, nouter; Pt pt[ N ], outer[4], res;
double radius,tmp;
void ball() {
  Pt q[3]; double m[3][3], sol[3], L[3], det;
  int i,j; res.x = res.y = res.z = radius = 0;
  switch ( nouter ) {
    case 1: res=outer[0]; break;
    case 2: res=(outer[0]+outer[1])/2; radius=norm2(res
        , outer[0]); break;
    case 3:
      for (i=0; i<2; ++i) q[i]=outer[i+1]-outer[0];
      for (i=0; i<2; ++i) for(j=0; j<2; ++j) m[i][j]=(q
          [i] * q[j])*2;
```

```
      for (i=0; i<2; ++i) sol[i]=(q[i] * q[i]);
      if (fabs(det=m[0][0]*m[1][1]-m[0][1]*m[1][0])<eps
          ) return;
      L[0]=(sol[0]*m[1][1]-sol[1]*m[0][1])/det;
      L[1]=(sol[1]*m[0][0]-sol[0]*m[1][0])/det;
      res=outer[0]+q[0]*L[0]+q[1]*L[1];
      radius=norm2(res, outer[0]);
      break;
    case 4:
      for (i=0; i<3; ++i) q[i]=outer[i+1]-outer[0], sol
          [i]=(q[i] * q[i]);
      for (i=0;i<3;++i) for(j=0;j<3;++j) m[i][j]=(q[i]
          * q[j])*2;
      det= m[0][0]*m[1][1]*m[2][2]
        + m[0][1]*m[1][2]*m[2][0]
        + m[0][2]*m[2][1]*m[1][0]
        - m[0][2]*m[1][1]*m[2][0]
        - m[0][1]*m[1][0]*m[2][2]
        - m[0][0]*m[1][2]*m[2][1];
      if ( fabs(det)<eps ) return;
      for (j=0; j<3; ++j) {
        for (i=0; i<3; ++i) m[i][j]=sol[i];
        L[j]=( m[0][0]*m[1][1]*m[2][2]
               + m[0][1]*m[1][2]*m[2][0]
               + m[0][2]*m[2][1]*m[1][0]
               - m[0][2]*m[1][1]*m[2][0]
               - m[0][1]*m[1][0]*m[2][2]
               - m[0][0]*m[1][2]*m[2][1]
             ) / det;
        for (i=0; i<3; ++i) m[i][j]=(q[i] * q[j])*2;
      } res=outer[0];
      for (i=0; i<3; ++i ) res = res + q[i] * L[i];
      radius=norm2(res, outer[0]);
}}
void minball(int n){ ball();
  if( nouter < 4 ) for( int i = 0 ; i < n ; i ++ )
    if( norm2(res, pt[i]) - radius > eps ){
      outer[ nouter ++ ] = pt[ i ]; minball(i); --
          nouter;
      if(i>0){ Pt Tt = pt[i];
        memmove(&pt[1], &pt[0], sizeof(Pt)*i); pt[0]=Tt
            ;
}}}
void solve{
  // n points in pt
  random_shuffle(pt, pt+n); radius=-1;
  for(int i=0;i<n;i++) if(norm2(res,pt[i])-radius>eps)
    nouter=1, outer[0]=pt[i], minball(i);
  printf("%.5f\n",sqrt(radius));
}
```

4.13 Minkowski sum

```
vector<Pt> minkowski(vector<Pt> p, vector<Pt> q){
  int n = p.size() , m = q.size();
  Pt c = Pt(0, 0);
  for( int i = 0; i < m; i ++) c = c + q[i];
  c = c / m;
  for( int i = 0; i < m; i ++) q[i] = q[i] - c;
  int cur = -1;
  for( int i = 0; i < m; i ++)
    if( (q[i] ^ (p[0] - p[n-1])) > -eps)
      if( cur == -1 ||
          (q[i] ^ (p[0] - p[n-1])) >
          (q[cur] ^ (p[0] - p[n-1])) )
        cur = i;
  vector<Pt> h;
  p.push_back(p[0]);
  for( int i = 0; i < n; i ++)
    while( true ){
      h.push_back(p[i] + q[cur]);
      int nxt = (cur + 1 == m ? 0 : cur + 1);
      if((q[cur] ^ (p[i+1] - p[i])) < -eps) cur = nxt;
      else if( (q[nxt] ^ (p[i+1] - p[i])) >
               (q[cur] ^ (p[i+1] - p[i])) ) cur = nxt;
      else break;
    }
  for(auto &&i : h) i = i + c;
  return convex_hull(h);
}
```

4.14 Heart of Triangle

```
Pt inCenter( Pt &A,  Pt &B,  Pt &C) { // 內心
  double a = norm(B-C), b = norm(C-A), c = norm(A-B);
  return (A * a + B * b + C * c) / (a + b + c);
}
Pt circumCenter( Pt &a,  Pt &b,  Pt &c) { // 外心
  Pt bb = b - a, cc = c - a;
  double db=norm2(bb), dc=norm2(cc), d=2*(bb ^ cc);
  return a-Pt(bb.Y*dc-cc.Y*db, cc.X*db-bb.X*dc) / d;
}
Pt othroCenter( Pt &a,  Pt &b,  Pt &c) { // 垂心
  Pt ba = b - a, ca = c - a, bc = b - c;
  double Y = ba.Y * ca.Y * bc.Y,
    A = ca.X * ba.Y - ba.X * ca.Y,
    x0= (Y+ca.X*ba.Y*b.X-ba.X*ca.Y*c.X) / A,
    y0= -ba.X * (x0 - c.X) / ba.Y + ca.Y;
  return Pt(x0, y0);
}
```

5 Graph

5.1 HeavyLightDecomp

```
#define SZ(c) (int)(c).size()
#define ALL(c) (c).begin(), (c).end()
#define REP(i, s, e) for(int i = (s); i <= (e); i++)
#define REPD(i, s, e) for(int i = (s); i >= (e); i--)
typedef tuple< int , int > tii;
const int MAXN = 100010;
const int LOG  = 19;
struct HLD{
  int n;
  vector<int> g[MAXN];
  int sz[MAXN], dep[MAXN];
  int ts, tid[MAXN], tdi[MAXN], tl[MAXN], tr[MAXN];
  //  ts : timestamp , useless after yutruli
  //  tid[ u ] : pos. of node u in the seq.
  //  tdi[ i ] : node at pos i of the seq.
  //  tl , tr[ u ] : subtree interval in the seq. of
      node u
  int mom[MAXN][LOG], head[MAXN];
  // head[ u ] : head of the chain contains u
  void dfssz(int u, int p){
    dep[u] = dep[p] + 1;
    mom[u][0] = p;
    sz[u] = 1;
    head[u] = u;
    for(int& v:g[u]) if(v != p){
      dep[v] = dep[u] + 1;
      dfssz(v, u);
      sz[u] += sz[v];
    }
  }
  void dfshl(int u){
    //printf("dfshl %d\n", u);
    ts++;
    tid[u] = tl[u] = tr[u] = ts;
    tdi[tid[u]] = u;
    sort(ALL(g[u]),
         [&](int a, int b){return sz[a] > sz[b];});
    bool flag = 1;
    for(int& v:g[u]) if(v != mom[u][0]){
      if(flag) head[v] = head[u], flag = 0;
      dfshl(v);
      tr[u] = tr[v];
    }
  }
  inline int lca(int a, int b){
    if(dep[a] > dep[b]) swap(a, b);
    //printf("lca %d %d\n", a, b);
    int diff = dep[b] - dep[a];
    REPD(k, LOG-1, 0) if(diff & (1<<k)){
      //printf("b %d\n", mom[b][k]);
      b = mom[b][k];
    }
    if(a == b) return a;
    REPD(k, LOG-1, 0) if(mom[a][k] != mom[b][k]){
```

```
      a = mom[a][k];
      b = mom[b][k];
    }
    return mom[a][0];
  }
  void init( int _n ){
    n = _n;
    REP( i , 1 , n ) g[ i ].clear();
  }
  void addEdge( int u , int v ){
    g[ u ].push_back( v );
    g[ v ].push_back( u );
  }
  void yutruli(){
    dfssz(1, 0);
    ts = 0;
    dfshl(1);
    REP(k, 1, LOG-1) REP(i, 1, n)
      mom[i][k] = mom[mom[i][k-1]][k-1];
  }
  vector< tii > getPath( int u , int v ){
    vector< tii > res;
    while( tid[ u ] < tid[ head[ v ] ] ){
      res.push_back( tii(tid[ head[ v ] ] , tid[ v ]) )
          ;
      v = mom[ head[ v ] ][ 0 ];
    }
    res.push_back( tii( tid[ u ] , tid[ v ] ) );
    reverse( ALL( res ) );
    return res;
    /*
     * res : list of intervals from u to v
     * u must be ancestor of v
     * usage :
     * vector< tii >& path = tree.getPath( u , v )
     * for( tii tp : path ) {
     *   int l , r;tie( l , r ) = tp;
     *   upd( l , r );
     *   uu = tree.tdi[ l ] , vv = tree.tdi[ r ];
     *   uu ~> vv is a heavy path on tree
     * }
     */
  }
} tree;
```

5.2 DominatorTree

```
const int MAXN = 100010;
struct DominatorTree{
#define REP(i,s,e) for(int i=(s);i<=(e);i++)
#define REPD(i,s,e) for(int i=(s);i>=(e);i--)
  int n , m , s;
  vector< int > g[ MAXN ] , pred[ MAXN ];
  vector< int > cov[ MAXN ];
  int dfn[ MAXN ] , nfd[ MAXN ] , ts;
  int par[ MAXN ];
  int sdom[ MAXN ] , idom[ MAXN ];
  int mom[ MAXN ] , mn[ MAXN ];
  inline bool cmp( int u , int v )
  { return dfn[ u ] < dfn[ v ]; }
  int eval( int u ){
    if( mom[ u ] == u ) return u;
    int res = eval( mom[ u ] );
    if(cmp( sdom[ mn[ mom[ u ] ] ] , sdom[ mn[ u ] ] ))
      mn[ u ] = mn[ mom[ u ] ];
    return mom[ u ] = res;
  }
  void init( int _n , int _m , int _s ){
    ts = 0; n = _n; m = _m; s = _s;
    REP( i, 1, n ) g[ i ].clear(), pred[ i ].clear();
  }
  void addEdge( int u , int v ){
    g[ u ].push_back( v );
    pred[ v ].push_back( u );
  }
  void dfs( int u ){
    ts++;
    dfn[ u ] = ts;
    nfd[ ts ] = u;
    for( int v : g[ u ] ) if( dfn[ v ] == 0 ){
      par[ v ] = u;
      dfs( v );
    }
  }
  void build(){
    REP( i , 1 , n ){
      dfn[ i ] = nfd[ i ] = 0;
      cov[ i ].clear();
      mom[ i ] = mn[ i ] = sdom[ i ] = i;
    }
    dfs( s );
    REPD( i , n , 2 ){
      int u = nfd[ i ];
      if( u == 0 ) continue ;
      for( int v : pred[ u ] ) if( dfn[ v ] ){
        eval( v );
        if( cmp( sdom[ mn[ v ] ] , sdom[ u ] ) )
          sdom[ u ] = sdom[ mn[ v ] ];
      }
      cov[ sdom[ u ] ].push_back( u );
      mom[ u ] = par[ u ];
      for( int w : cov[ par[ u ] ] ){
        eval( w );
        if( cmp( sdom[ mn[ w ] ] , par[ u ] ) )
          idom[ w ] = mn[ w ];
        else idom[ w ] = par[ u ];
      }
      cov[ par[ u ] ].clear();
    }
    REP( i , 2 , n ){
      int u = nfd[ i ];
      if( u == 0 ) continue ;
      if( idom[ u ] != sdom[ u ] )
        idom[ u ] = idom[ idom[ u ] ];
    }
  }
} domT;
```

5.3 MaxClique

```
struct MaxClique {
  static const int MV = 210;
  int V , ans , dp[MV];
  int el[MV][MV/30+1], s[MV][MV/30+1];
  vector<int> sol;
  void init(int v) {
    V = v; ans = 0;
    FZ(el); FZ(dp);
  }
  /* Zero Base */
  void addEdge(int u, int v) {
    if(u > v) swap(u, v);
    if(u == v) return;
    el[u][v/32] |= (1<<(v%32));
  }
  bool dfs(int v, int k) {
    int c = 0, d = 0;
    for(int i=0; i<(V+31)/32; i++) {
      s[k][i] = el[v][i];
      if(k != 1) s[k][i] &= s[k-1][i];
      c += __builtin_popcount(s[k][i]);
    }
    if(c == 0) {
      if(k > ans) {
        ans = k;
        sol.clear();
        sol.push_back(v);
        return 1;
      }
      return 0;
    }
    for(int i=0; i<(V+31)/32; i++) {
      for(int a = s[k][i]; a ; d++) {
        if(k + (c-d) <= ans) return 0;
        int lb = a&(-a), lg = 0;
        a ^= lb;
        while(lb!=1) {
          lb = (unsigned int)(lb) >> 1;
          lg ++;
        }
```

```
        int u = i*32 + lg;
        if(k + dp[u] <= ans) return 0;
        if(dfs(u, k+1)) {
          sol.push_back(v);
          return 1;
        }
      }
    }
    return 0;
  }
  int solve() {
    for(int i=V-1; i>=0; i--) {
      dfs(i, 1);
      dp[i] = ans;
    }
    return ans;
  }
};
```

5.4 Number of Maximal Clique

```
// bool g[][] : adjacent array indexed from 1 to n
void dfs(int sz){
  int i, j, k, t, cnt, best = 0;
  if(ne[sz]==ce[sz]){ if (ce[sz]==0) ++ans; return; }
  for(t=0, i=1; i<=ne[sz]; ++i){
    for (cnt=0, j=ne[sz]+1; j<=ce[sz]; ++j)
    if (!g[lst[sz][i]][lst[sz][j]]) ++cnt;
    if (t==0 || cnt<best) t=i, best=cnt;
  } if (t && best<=0) return;
  for (k=ne[sz]+1; k<=ce[sz]; ++k) {
    if (t>0){ for (i=k; i<=ce[sz]; ++i)
        if (!g[lst[sz][t]][lst[sz][i]]) break;
      swap(lst[sz][k], lst[sz][i]);
    } i=lst[sz][k]; ne[sz+1]=ce[sz+1]=0;
    for (j=1; j<k; ++j)if (g[i][lst[sz][j]])
        lst[sz+1][++ne[sz+1]]=lst[sz][j];
    for (ce[sz+1]=ne[sz+1], j=k+1; j<=ce[sz]; ++j)
    if (g[i][lst[sz][j]]) lst[sz+1][++ce[sz+1]]=lst[sz
        ][j];
    dfs(sz+1); ++ne[sz]; --best;
    for (j=k+1, cnt=0; j<=ce[sz]; ++j) if (!g[i][lst[sz
        ][j]]) ++cnt;
    if (t==0 || cnt<best) t=k, best=cnt;
    if (t && best<=0) break;
}}
void work(){
  ne[0]=0; ce[0]=0;
  for(int i=1; i<=n; ++i) lst[0][++ce[0]]=i;
  ans=0; dfs(0);
}
```

5.5 Strongly Connected Component

```
struct Scc{
  int n, nScc, vst[MXN], bln[MXN];
  vector<int> E[MXN], rE[MXN], vec;
  void init(int _n){
    n = _n;
    for (int i=0; i<MXN; i++){
      E[i].clear();
      rE[i].clear();
    }
  }
  void add_edge(int u, int v){
    E[u].PB(v);
    rE[v].PB(u);
  }
  void DFS(int u){
    vst[u]=1;
    for (auto v : E[u])
      if (!vst[v]) DFS(v);
    vec.PB(u);
  }
  void rDFS(int u){
    vst[u] = 1;
    bln[u] = nScc;
    for (auto v : rE[u])
      if (!vst[v]) rDFS(v);
  }
  void solve(){
    nScc = 0;
    vec.clear();
    FZ(vst);
    for (int i=0; i<n; i++)
      if (!vst[i]) DFS(i);
    reverse(vec.begin(),vec.end());
    FZ(vst);
    for (auto v : vec){
      if (!vst[v]){
        rDFS(v);
        nScc++;
      }
    }
  }
};
```

5.6 Dynamic MST

```
/* Dynamic MST O( Q lg^2 Q )
 (qx[i], qy[i])->chg weight of edge No.qx[i] to qy[i]
 delete an edge: (i, \infty)
 add an edge: change from \infty to specific value
 */
const int SZ=M+3*MXQ;
int a[N],*tz;
int find(int xx){
  int root=xx; while(a[root]) root=a[root];
  int next; while((next=a[xx])){a[xx]=root; xx=next; }
  return root;
}
bool cmp(int aa,int bb){ return tz[aa]<tz[bb]; }
int kx[N],ky[N],kt, vd[N],id[M], app[M];
bool extra[M];
void solve(int *qx,int *qy,int Q,int n,int *x,int *y,
    int *z,int m1,long long ans){
  if(Q==1){
    for(int i=1;i<=n;i++) a[i]=0;
    z[ qx[0] ]=qy[0]; tz = z;
    for(int i=0;i<m1;i++) id[i]=i;
    sort(id,id+m1,cmp); int ri,rj;
    for(int i=0;i<m1;i++){
      ri=find(x[id[i]]); rj=find(y[id[i]]);
      if(ri!=rj){ ans+=z[id[i]]; a[ri]=rj; }
    }
    printf("%lld\n",ans);
    return;
  }
  int ri,rj;
  //contract
  kt=0;
  for(int i=1;i<=n;i++) a[i]=0;
  for(int i=0;i<Q;i++){
    ri=find(x[qx[i]]); rj=find(y[qx[i]]); if(ri!=rj) a[
        ri]=rj;
  }
  int tm=0;
  for(int i=0;i<m1;i++) extra[i]=true;
  for(int i=0;i<Q;i++) extra[ qx[i] ]=false;
  for(int i=0;i<m1;i++) if(extra[i]) id[tm++]=i;
  tz=z; sort(id,id+tm,cmp);
  for(int i=0;i<tm;i++){
    ri=find(x[id[i]]); rj=find(y[id[i]]);
    if(ri!=rj){
      a[ri]=rj; ans += z[id[i]];
      kx[kt]=x[id[i]]; ky[kt]=y[id[i]]; kt++;
    }
  }
  for(int i=1;i<=n;i++) a[i]=0;
  for(int i=0;i<kt;i++) a[ find(kx[i]) ]=find(ky[i]);
  int n2=0;
  for(int i=1;i<=n;i++) if(a[i]==0)
  vd[i]=++n2;
  for(int i=1;i<=n;i++) if(a[i])
  vd[i]=vd[find(i)];
  int m2=0, *Nx=x+m1, *Ny=y+m1, *Nz=z+m1;
  for(int i=0;i<m1;i++) app[i]=-1;
  for(int i=0;i<Q;i++) if(app[qx[i]]==-1){
```

```
    Nx[m2]=vd[ x[ qx[i] ] ]; Ny[m2]=vd[ y[ qx[i] ] ];
        Nz[m2]=z[ qx[i] ];
    app[qx[i]]=m2; m2++;
  }
  for(int i=0;i<Q;i++){ z[ qx[i] ]=qy[i]; qx[i]=app[qx[
      i]]; }
  for(int i=1;i<=n2;i++) a[i]=0;
  for(int i=0;i<tm;i++){
    ri=find(vd[ x[id[i]] ]);  rj=find(vd[ y[id[i]] ]);
    if(ri!=rj){
      a[ri]=rj; Nx[m2]=vd[ x[id[i]] ];
      Ny[m2]=vd[ y[id[i]] ]; Nz[m2]=z[id[i]]; m2++;
    }
  }
  int mid=Q/2;
  solve(qx,qy,mid,n2,Nx,Ny,Nz,m2,ans);
  solve(qx+mid,qy+mid,Q-mid,n2,Nx,Ny,Nz,m2,ans);
}
int x[SZ],y[SZ],z[SZ],qx[MXQ],qy[MXQ],n,m,Q;
void init(){
  scanf("%d%d",&n,&m);
  for(int i=0;i<m;i++) scanf("%d%d%d",x+i,y+i,z+i);
  scanf("%d",&Q);
  for(int i=0;i<Q;i++){ scanf("%d%d",qx+i,qy+i); qx[i
      ]--; }
}
void work(){ if(Q) solve(qx,qy,Q,n,x,y,z,m,0); }
int main(){init(); work(); }
```

5.7 Maximum General graph Matching

```
const int N = 514, E = (2e5) * 2;
struct Graph{
  int to[E],bro[E],head[N],e;
  int lnk[N],vis[N],stp,n;
  void init( int _n ){
    stp = 0; e = 1; n = _n;
    for( int i = 1 ; i <= n ; i ++ )
      lnk[i] = vis[i] = 0;
  }
  void add_edge(int u,int v){
    to[e]=v,bro[e]=head[u],head[u]=e++;
    to[e]=u,bro[e]=head[v],head[v]=e++;
  }
  bool dfs(int x){
    vis[x]=stp;
    for(int i=head[x];i;i=bro[i]){
      int v=to[i];
      if(!lnk[v]){
        lnk[x]=v,lnk[v]=x;
        return true;
      }else if(vis[lnk[v]]<stp){
        int w=lnk[v];
        lnk[x]=v,lnk[v]=x,lnk[w]=0;
        if(dfs(w)){
          return true;
        }
        lnk[w]=v,lnk[v]=w,lnk[x]=0;
      }
    }
    return false;
  }
  int solve(){
    int ans = 0;
    for(int i=1;i<=n;i++)
      if(!lnk[i]){
        stp++; ans += dfs(i);
      }
    return ans;
  }
} graph;
```

5.8 Minimum General Weighted Matching

```
struct Graph {
  // Minimum General Weighted Matching (Perfect Match)
  static const int MXN = 105;
  int n, edge[MXN][MXN];
  int match[MXN],dis[MXN],onstk[MXN];
  vector<int> stk;
  void init(int _n) {
    n = _n;
    for( int i = 0 ; i < n ; i ++ )
      for( int j = 0 ; j < n ; j ++ )
        edge[ i ][ j ] = 0;
  }
  void add_edge(int u, int v, int w)
  { edge[u][v] = edge[v][u] = w; }
  bool SPFA(int u){
    if (onstk[u]) return true;
    stk.PB(u);
    onstk[u] = 1;
    for (int v=0; v<n; v++){
      if (u != v && match[u] != v && !onstk[v]){
        int m = match[v];
        if (dis[m] > dis[u] - edge[v][m] + edge[u][v]){
          dis[m] = dis[u] - edge[v][m] + edge[u][v];
          onstk[v] = 1;
          stk.PB(v);
          if (SPFA(m)) return true;
          stk.pop_back();
          onstk[v] = 0;
        }
      }
    }
    onstk[u] = 0;
    stk.pop_back();
    return false;
  }
  int solve() {
    // find a match
    for (int i=0; i<n; i+=2){
      match[i] = i+1;
      match[i+1] = i;
    }
    while (true){
      int found = 0;
      for( int i = 0 ; i < n ; i ++ )
        onstk[ i ] = dis[ i ] = 0;
      for (int i=0; i<n; i++){
        stk.clear();
        if (!onstk[i] && SPFA(i)){
          found = 1;
          while (SZ(stk)>=2){
            int u = stk.back(); stk.pop_back();
            int v = stk.back(); stk.pop_back();
            match[u] = v;
            match[v] = u;
          }
        }
      }
      if (!found) break;
    }
    int ret = 0;
    for (int i=0; i<n; i++)
      ret += edge[i][match[i]];
    ret /= 2;
    return ret;
  }
}graph;
```

5.9 Maximum General Weighted Matching

```
struct WeightGraph {
  static const int INF = INT_MAX;
  static const int N = 514;
  struct edge{
    int u,v,w; edge(){}
    edge(int ui,int vi,int wi)
      :u(ui),v(vi),w(wi){}
  };
  int n,n_x;
  edge g[N*2][N*2];
  int lab[N*2];
  int match[N*2],slack[N*2],st[N*2],pa[N*2];
  int flo_from[N*2][N+1],S[N*2],vis[N*2];
  vector<int> flo[N*2];
  queue<int> q;
```

```
int e_delta(const edge &e){
  return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
}
void update_slack(int u,int x){
  if(!slack[x]||e_delta(g[u][x])<e_delta(g[slack[x]][
      x]))slack[x]=u;
}
void set_slack(int x){
  slack[x]=0;
  for(int u=1;u<=n;++u)
    if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)
      update_slack(u,x);
}
void q_push(int x){
  if(x<=n)q.push(x);
  else for(size_t i=0;i<flo[x].size();i++)
    q_push(flo[x][i]);
}
void set_st(int x,int b){
  st[x]=b;
  if(x>n)for(size_t i=0;i<flo[x].size();++i)
    set_st(flo[x][i],b);
}
int get_pr(int b,int xr){
  int pr=find(flo[b].begin(),flo[b].end(),xr)-flo[b].
      begin();
  if(pr%2==1){
    reverse(flo[b].begin()+1,flo[b].end());
    return (int)flo[b].size()-pr;
  }else return pr;
}
void set_match(int u,int v){
  match[u]=g[u][v].v;
  if(u<=n) return;
  edge e=g[u][v];
  int xr=flo_from[u][e.u],pr=get_pr(u,xr);
  for(int i=0;i<pr;++i)set_match(flo[u][i],flo[u][i
      ^1]);
  set_match(xr,v);
  rotate(flo[u].begin(),flo[u].begin()+pr,flo[u].end
      ());
}
void augment(int u,int v){
  for(;;){
    int xnv=st[match[u]];
    set_match(u,v);
    if(!xnv)return;
    set_match(xnv,st[pa[xnv]]);
    u=st[pa[xnv]],v=xnv;
  }
}
int get_lca(int u,int v){
  static int t=0;
  for(++t;u||v;swap(u,v)){
    if(u==0)continue;
    if(vis[u]==t)return u;
    vis[u]=t;
    u=st[match[u]];
    if(u)u=st[pa[u]];
  }
  return 0;
}
void add_blossom(int u,int lca,int v){
  int b=n+1;
  while(b<=n_x&&st[b])++b;
  if(b>n_x)++n_x;
  lab[b]=0,S[b]=0;
  match[b]=match[lca];
  flo[b].clear();
  flo[b].push_back(lca);
  for(int x=u,y;x!=lca;x=st[pa[y]])
    flo[b].push_back(x),flo[b].push_back(y=st[match[x
        ]]),q_push(y);
  reverse(flo[b].begin()+1,flo[b].end());
  for(int x=v,y;x!=lca;x=st[pa[y]])
    flo[b].push_back(x),flo[b].push_back(y=st[match[x
        ]]),q_push(y);
  set_st(b,b);
  for(int x=1;x<=n_x;++x)g[b][x].w=g[x][b].w=0;
  for(int x=1;x<=n;++x)flo_from[b][x]=0;
  for(size_t i=0;i<flo[b].size();++i){
    int xs=flo[b][i];
    for(int x=1;x<=n_x;++x)
      if(g[b][x].w==0||e_delta(g[xs][x])<e_delta(g[b
          ][x]))
        g[b][x]=g[xs][x],g[x][b]=g[x][xs];
    for(int x=1;x<=n;++x)
      if(flo_from[xs][x])flo_from[b][x]=xs;
  }
  set_slack(b);
}
void expand_blossom(int b){
  for(size_t i=0;i<flo[b].size();++i)
    set_st(flo[b][i],flo[b][i]);
  int xr=flo_from[b][g[b][pa[b]].u],pr=get_pr(b,xr);
  for(int i=0;i<pr;i+=2){
    int xs=flo[b][i],xns=flo[b][i+1];
    pa[xs]=g[xns][xs].u;
    S[xs]=1,S[xns]=0;
    slack[xs]=0,set_slack(xns);
    q_push(xns);
  }
  S[xr]=1,pa[xr]=pa[b];
  for(size_t i=pr+1;i<flo[b].size();++i){
    int xs=flo[b][i];
    S[xs]=-1,set_slack(xs);
  }
  st[b]=0;
}
bool on_found_edge(const edge &e){
  int u=st[e.u],v=st[e.v];
  if(S[v]==-1){
    pa[v]=e.u,S[v]=1;
    int nu=st[match[v]];
    slack[v]=slack[nu]=0;
    S[nu]=0,q_push(nu);
  }else if(S[v]==0){
    int lca=get_lca(u,v);
    if(!lca)return augment(u,v),augment(v,u),true;
    else add_blossom(u,lca,v);
  }
  return false;
}
bool matching(){
  memset(S+1,-1,sizeof(int)*n_x);
  memset(slack+1,0,sizeof(int)*n_x);
  q=queue<int>();
  for(int x=1;x<=n_x;++x)
    if(st[x]==x&&!match[x])pa[x]=0,S[x]=0,q_push(x);
  if(q.empty())return false;
  for(;;){
    while(q.size()){
      int u=q.front();q.pop();
      if(S[st[u]]==1)continue;
      for(int v=1;v<=n;++v)
        if(g[u][v].w>0&&st[u]!=st[v]){
          if(e_delta(g[u][v])==0){
            if(on_found_edge(g[u][v]))return true;
          }else update_slack(u,st[v]);
        }
    }
    int d=INF;
    for(int b=n+1;b<=n_x;++b)
      if(st[b]==b&&S[b]==1)d=min(d,lab[b]/2);
    for(int x=1;x<=n_x;++x)
      if(st[x]==x&&slack[x]){
        if(S[x]==-1)d=min(d,e_delta(g[slack[x]][x]));
        else if(S[x]==0)d=min(d,e_delta(g[slack[x]][x
            ])/2);
      }
    for(int u=1;u<=n;++u){
      if(S[st[u]]==0){
        if(lab[u]<=d)return 0;
        lab[u]-=d;
      }else if(S[st[u]]==1)lab[u]+=d;
    }
    for(int b=n+1;b<=n_x;++b)
      if(st[b]==b){
        if(S[st[b]]==0)lab[b]+=d*2;
        else if(S[st[b]]==1)lab[b]-=d*2;
      }
    q=queue<int>();
    for(int x=1;x<=n_x;++x)
```

```
        if(st[x]==x&&slack[x]&&st[slack[x]]!=x&&e_delta
            (g[slack[x]][x])==0)
          if(on_found_edge(g[slack[x]][x]))return true;
      for(int b=n+1;b<=n_x;++b)
        if(st[b]==b&&S[b]==1&&lab[b]==0)expand_blossom(
            b);
    }
    return false;
  }
  pair<long long,int> solve(){
    memset(match+1,0,sizeof(int)*n);
    n_x=n;
    int n_matches=0;
    long long tot_weight=0;
    for(int u=0;u<=n;++u)st[u]=u,flo[u].clear();
    int w_max=0;
    for(int u=1;u<=n;++u)
      for(int v=1;v<=n;++v){
        flo_from[u][v]=(u==v?u:0);
        w_max=max(w_max,g[u][v].w);
      }
    for(int u=1;u<=n;++u)lab[u]=w_max;
    while(matching())++n_matches;
    for(int u=1;u<=n;++u)
      if(match[u]&&match[u]<u)
        tot_weight+=g[u][match[u]].w;
    return make_pair(tot_weight,n_matches);
  }
  void add_edge( int ui , int vi , int wi ){
    g[ui][vi].w = g[vi][ui].w = wi;
  }
  void init( int _n ){
    n = _n;
    for(int u=1;u<=n;++u)
      for(int v=1;v<=n;++v)
        g[u][v]=edge(u,v,0);
  }
} graph;
```

5.10 Minimum Steiner Tree

```
// Minimum Steiner Tree
// O(V 3^T + V^2 2^T)
struct SteinerTree{
#define V 33
#define T 8
#define INF 1023456789
  int n , dst[V][V] , dp[1 << T][V] , tdst[V];
  void init( int _n ){
    n = _n;
    for( int i = 0 ; i < n ; i ++ ){
      for( int j = 0 ; j < n ; j ++ )
        dst[ i ][ j ] = INF;
      dst[ i ][ i ] = 0;
    }
  }
  void add_edge( int ui , int vi , int wi ){
    dst[ ui ][ vi ] = min( dst[ ui ][ vi ] , wi );
    dst[ vi ][ ui ] = min( dst[ vi ][ ui ] , wi );
  }
  void shortest_path(){
    for( int k = 0 ; k < n ; k ++ )
      for( int i = 0 ; i < n ; i ++ )
        for( int j = 0 ; j < n ; j ++ )
          dst[ i ][ j ] = min( dst[ i ][ j ],
                dst[ i ][ k ] + dst[ k ][ j ] );
  }
  int solve( const vector<int>& ter ){
    int t = (int)ter.size();
    for( int i = 0 ; i < ( 1 << t ) ; i ++ )
      for( int j = 0 ; j < n ; j ++ )
        dp[ i ][ j ] = INF;
    for( int i = 0 ; i < n ; i ++ )
      dp[ 0 ][ i ] = 0;
    for( int msk = 1 ; msk < ( 1 << t ) ; msk ++ ){
      if( msk == ( msk & (-msk) ) ){
        int who = __lg( msk );
        for( int i = 0 ; i < n ; i ++ )
          dp[ msk ][ i ] = dst[ ter[ who ] ][ i ];
        continue;
      }
      for( int i = 0 ; i < n ; i ++ )
        for( int submsk = ( msk - 1 ) & msk ; submsk ;
                 submsk = ( submsk - 1 ) & msk )
          dp[ msk ][ i ] = min( dp[ msk ][ i ],
                               dp[ submsk ][ i ] +
                               dp[ msk ^ submsk ][ i ] );
      for( int i = 0 ; i < n ; i ++ ){
        tdst[ i ] = INF;
        for( int j = 0 ; j < n ; j ++ )
          tdst[ i ] = min( tdst[ i ],
                      dp[ msk ][ j ] + dst[ j ][ i ] );
      }
      for( int i = 0 ; i < n ; i ++ )
        dp[ msk ][ i ] = tdst[ i ];
    }
    int ans = INF;
    for( int i = 0 ; i < n ; i ++ )
      ans = min( ans , dp[ ( 1 << t ) - 1 ][ i ] );
    return ans;
  }
} solver;
```

5.11 BCC based on vertex

```
struct BccVertex {
  int n,nScc,step,dfn[MXN],low[MXN];
  vector<int> E[MXN],sccv[MXN];
  int top,stk[MXN];
  void init(int _n) {
    n = _n;
    nScc = step = 0;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v) {
    E[u].PB(v);
    E[v].PB(u);
  }
  void DFS(int u, int f) {
    dfn[u] = low[u] = step++;
    stk[top++] = u;
    for (auto v:E[u]) {
      if (v == f) continue;
      if (dfn[v] == -1) {
        DFS(v,u);
        low[u] = min(low[u], low[v]);
        if (low[v] >= dfn[u]) {
          int z;
          sccv[nScc].clear();
          do {
            z = stk[--top];
            sccv[nScc].PB(z);
          } while (z != v);
          sccv[nScc].PB(u);
          nScc++;
        }
      } else {
        low[u] = min(low[u],dfn[v]);
      }
    }
  }
  vector<vector<int>> solve() {
    vector<vector<int>> res;
    for (int i=0; i<n; i++) {
      dfn[i] = low[i] = -1;
    }
    for (int i=0; i<n; i++) {
      if (dfn[i] == -1) {
        top = 0;
        DFS(i,i);
      }
    }
    REP(i,nScc) res.PB(sccv[i]);
    return res;
  }
}graph;
```

5.12 Graph Hash

```
$$F_t(i) =
  (F_{t-1}(i) \times A +
  \sum_{i\rightarrow j} F_{t-1}(j) \times B +
  \sum_{j\rightarrow i} F_{t-1}(j) \times C +
  D \times (i = a))\ mod\ P
$$
for each node i, iterate t times.
t, A, B, C, D, P are hash parameter
```

6 String

6.1 PalTree

```
const int MAXN = 200010;
struct PalT{
  struct Node{
    int nxt[ 33 ] , len , fail;
    ll cnt;
  };
  int tot , lst;
  Node nd[ MAXN * 2 ];
  char* s;
  int newNode( int l , int _fail ){
    int res = ++tot;
    memset( nd[ res ].nxt , 0 , sizeof nd[ res ].nxt );
    nd[ res ].len = l;
    nd[ res ].cnt = 0;
    nd[ res ].fail = _fail;
    return res;
  }
  void push( int p ){
    int np = lst;
    int c = s[ p ] - 'a';
    while( p - nd[ np ].len - 1 < 0
        || s[ p ] != s[ p - nd[ np ].len - 1 ] )
      np = nd[ np ].fail;

    if( nd[ np ].nxt[ c ] ){
      nd[ nd[ np ].nxt[ c ] ].cnt++;
      lst = nd[ np ].nxt[ c ];
      return ;
    }
    int nq = newNode( nd[ np ].len + 2 , 0 );
    nd[ nq ].cnt++;
    nd[ np ].nxt[ c ] = nq;
    lst = nq;
    if( nd[ nq ].len == 1 ){
      nd[ nq ].fail = 2;
      return ;
    }
    int tf = nd[ np ].fail;
    while( p - nd[ tf ].len - 1 < 0
        || s[ p ] != s[ p - nd[ tf ].len - 1 ] )
      tf = nd[ tf ].fail;

    nd[ nq ].fail = nd[ tf ].nxt[ c ];
    return ;
  }
  void init( char* _s ){
    s = _s;
    tot = 0;
    newNode( -1 , 1 );
    newNode( 0 , 1 );
    lst = 2;
    for( int i = 0 ; s[ i ] ; i++ )
      push( i );
  }
  void yutruli(){
#define REPD(i, s, e) for(int i = (s); i >= (e); i--)
    REPD( i , tot , 1 )
      nd[ nd[ i ].fail ].cnt += nd[ i ].cnt;
    nd[ 1 ].cnt = nd[ 2 ].cnt = 0ll;
  }
} pA;
int main(){ pA.init( sa ); }
```

6.2 SAIS

```
const int N = 300010;
struct SA{
#define REP(i,n) for ( int i=0; i<int(n); i++ )
#define REP1(i,a,b) for ( int i=(a); i<=int(b); i++ )
  bool _t[N*2];
  int _s[N*2], _sa[N*2], _c[N*2], x[N], _p[N], _q[N*2],
      hei[N], r[N];
  int operator [] (int i){ return _sa[i]; }
  void build(int *s, int n, int m){
    memcpy(_s, s, sizeof(int) * n);
    sais(_s, _sa, _p, _q, _t, _c, n, m);
    mkhei(n);
  }
  void mkhei(int n){
    REP(i,n) r[_sa[i]] = i;
    hei[0] = 0;
    REP(i,n) if(r[i]) {
      int ans = i>0 ? max(hei[r[i-1]] - 1, 0) : 0;
      while(_s[i+ans] == _s[_sa[r[i]-1]+ans]) ans++;
      hei[r[i]] = ans;
    }
  }
  void sais(int *s, int *sa, int *p, int *q, bool *t,
      int *c, int n, int z){
    bool uniq = t[n-1] = true, neq;
    int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s + n,
        lst = -1;
#define MS0(x,n) memset((x),0,n*sizeof(*(x)))
#define MAGIC(XD) MS0(sa, n); \
    memcpy(x, c, sizeof(int) * z); \
    XD; \
    memcpy(x + 1, c, sizeof(int) * (z - 1)); \
    REP(i,n) if(sa[i] && !t[sa[i]-1]) sa[x[s[sa[i
        ]-1]]++] = sa[i]-1; \
    memcpy(x, c, sizeof(int) * z); \
    for(int i = n - 1; i >= 0; i--) if(sa[i] && t[sa[i
        ]-1]) sa[--x[s[sa[i]-1]]] = sa[i]-1;
    MS0(c, z);
    REP(i,n) uniq &= ++c[s[i]] < 2;
    REP(i,z-1) c[i+1] += c[i];
    if (uniq) { REP(i,n) sa[--c[s[i]]] = i; return; }
    for(int i = n - 2; i >= 0; i--) t[i] = (s[i]==s[i
        +1] ? t[i+1] : s[i]<s[i+1]);
    MAGIC(REP1(i,1,n-1) if(t[i] && !t[i-1]) sa[--x[s[i
        ]]]=p[q[i]=nn++]=i);
    REP(i, n) if (sa[i] && t[sa[i]] && !t[sa[i]-1]) {
      neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]]+1]-sa
          [i])*sizeof(int));
      ns[q[lst=sa[i]]]=nmxz+=neq;
    }
    sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmxz
        + 1);
    MAGIC(for(int i = nn - 1; i >= 0; i--) sa[--x[s[p[
        nsa[i]]]]] = p[nsa[i]]);
  }
}sa;
int H[ N ], SA[ N ];
void suffix_array(int* ip, int len) {
  // should padding a zero in the back
  // ip is int array, len is array length
  // ip[0..n-1] != 0, and ip[len] = 0
  ip[len++] = 0;
  sa.build(ip, len, 128);
  for (int i=0; i<len; i++) {
    H[i] = sa.hei[i + 1];
    SA[i] = sa._sa[i + 1];
  }
  // resulting height, sa array \in [0,len)
}
```

6.3 SuffixAutomata

```
const int MAXM = 1000010;
struct SAM{
  int tot, root, lst, mom[MAXM], mx[MAXM];
  int acc[MAXM], nxt[MAXM][33];
  int newNode(){
```

```
    int res = ++tot;
    fill(nxt[res], nxt[res]+33, 0);
    mom[res] = mx[res] = acc[res] = 0;
    return res;
  }
  void init(){
    tot = 0;
    root = newNode();
    mom[root] = 0, mx[root] = 0;
    lst = root;
  }
  void push(int c){
    int p = lst;
    int np = newNode();
    mx[np] = mx[p]+1;
    for(; p && nxt[p][c] == 0; p = mom[p])
      nxt[p][c] = np;
    if(p == 0) mom[np] = root;
    else{
      int q = nxt[p][c];
      if(mx[p]+1 == mx[q]) mom[np] = q;
      else{
        int nq = newNode();
        mx[nq] = mx[p]+1;
        for(int i = 0; i < 33; i++)
          nxt[nq][i] = nxt[q][i];
        mom[nq] = mom[q];
        mom[q] = nq;
        mom[np] = nq;
        for(; p && nxt[p][c] == q; p = mom[p])
          nxt[p][c] = nq;
      }
    }
    lst = np;
  }
  void push(char *str){
    for(int i = 0; str[i]; i++)
      push(str[i]-'a'+1);
  }
} sam;
```

6.4 Aho-Corasick

```
struct ACautomata{
  struct Node{
    int cnt,dp;
    Node *go[26], *fail;
    Node (){
      cnt = 0; dp = -1; fail = 0;
      memset(go,0,sizeof(go));
    }
  };
  Node *root, pool[1048576];
  int nMem;
  Node* new_Node(){
    pool[nMem] = Node();
    return &pool[nMem++];
  }
  void init()
  { nMem = 0; root = new_Node(); }
  void add(const string &str)
  { insert(root,str,0); }
  void insert(Node *cur, const string &str, int pos){
    if (pos >= (int)str.size())
    { cur->cnt++; return; }
    int c = str[pos]-'a';
    if (cur->go[c] == 0)
      cur->go[c] = new_Node();
    insert(cur->go[c],str,pos+1);
  }
  void make_fail(){
    queue<Node*> que;
    que.push(root);
    while (!que.empty()){
      Node* fr=que.front();
      que.pop();
      for (int i=0; i<26; i++){
        if (fr->go[i]){
          Node *ptr = fr->fail;
          while (ptr && !ptr->go[i]) ptr = ptr->fail;
          if (!ptr) fr->go[i]->fail = root;
          else fr->go[i]->fail = ptr->go[i];
          que.push(fr->go[i]);
        }
      }
    }
  }
};
```

6.5 Z Value

```
char s[MAXN];
int len,z[MAXN];
void Z_value() {
  int i,j,left,right;
  left=right=0; z[0]=len;
  for(i=1;i<len;i++) {
    j=max(min(z[i-left],right-i),0);
    for(;i+j<len&&s[i+j]==s[j];j++);
    z[i]=j;
    if(i+z[i]>right) {
      right=i+z[i];
      left=i;
    }
  }
}
```

6.6 BWT

```
struct BurrowsWheeler{
#define SIGMA 26
#define BASE 'a'
  vector<int> v[ SIGMA ];
  void BWT(char* ori, char* res){
    // make ori -> ori + ori
    // then build suffix array
  }
  void iBWT(char* ori, char* res){
    for( int i = 0 ; i < SIGMA ; i ++ )
      v[ i ].clear();
    int len = strlen( ori );
    for( int i = 0 ; i < len ; i ++ )
      v[ ori[i] - BASE ].push_back( i );
    vector<int> a;
    for( int i = 0 , ptr = 0 ; i < SIGMA ; i ++ )
      for( auto j : v[ i ] ){
        a.push_back( j );
        ori[ ptr ++ ] = BASE + i;
      }
    for( int i = 0 , ptr = 0 ; i < len ; i ++ ){
      res[ i ] = ori[ a[ ptr ] ];
      ptr = a[ ptr ];
    }
    res[ len ] = 0;
  }
} bwt;
```

6.7 ZValue Palindrome

```
int len, zv[MAX*2];
char ip[MAX], op[MAX*2];
int main(){
  cin >> ip; len = strlen(ip);
  int l2 = len*2 - 1;
  for(int i=0; i<l2; i++)
    if(i&1) op[i] = '@';
    else op[i] = ip[i/2];
  int l=0, r=0; zv[0] = 1;
  for(int i=1; i<l2; i++){
    if( i > r ){
      l = r = i;
      while( l>0 && r<l2-1 && op[l-1] == op[r+1] )
        l --, r ++;
      zv[i] = (r-l+1);
    }else{
      int md = (l+r)/2, j = md + md - i;
```

```
      zv[i] = zv[j];
      int q = zv[i] / 2, nr = i + q;
      if( nr == r ){
        l = i + i - r;
        while( l>0 && r<l2-1 && op[l-1] == op[r+1] )
          l --, r ++;
        zv[i] = r - l + 1;
      }else if( nr > r )
        zv[i] = (r - i) * 2 + 1;
    }
  }
}
```

6.8 Smallest Rotation

```
string mcp(string s){
  int n = s.length();
  s += s;
  int i=0, j=1;
  while (i<n && j<n){
    int k = 0;
    while (k < n && s[i+k] == s[j+k]) k++;
    if (s[i+k] <= s[j+k]) j += k+1;
    else i += k+1;
    if (i == j) j++;
  }
  int ans = i < n ? i : j;
  return s.substr(ans, n);
}
```

7 Data Structure

7.1 Treap

```
struct Treap{
  int sz , val , pri , tag;
  Treap *l , *r;
  Treap( int _val ){
    val = _val; sz = 1;
    pri = rand(); l = r = NULL; tag = 0;
  }
};
void push( Treap * a ){
  if( a->tag ){
    Treap *swp = a->l; a->l = a->r; a->r = swp;
    int swp2;
    if( a->l ) a->l->tag ^= 1;
    if( a->r ) a->r->tag ^= 1;
    a->tag = 0;
  }
}
int Size( Treap * a ){ return a ? a->sz : 0; }
void pull( Treap * a ){
  a->sz = Size( a->l ) + Size( a->r ) + 1;
}
Treap* merge( Treap *a , Treap *b ){
  if( !a || !b ) return a ? a : b;
  if( a->pri > b->pri ){
    push( a );
    a->r = merge( a->r , b );
    pull( a );
    return a;
  }else{
    push( b );
    b->l = merge( a , b->l );
    pull( b );
    return b;
  }
}
void split( Treap *t , int k , Treap*&a , Treap*&b ){
  if( !t ){ a = b = NULL; return; }
  push( t );
  if( Size( t->l ) + 1 <= k ){
    a = t;
    split( t->r , k - Size( t->l ) - 1 , a->r , b );
    pull( a );
  }else{
    b = t;
    split( t->l , k , a , b->l );
    pull( b );
  }
}
```

7.2 Link-Cut Tree

```
const int MXN = 100005;
const int MEM = 100005;
struct Splay {
  static Splay nil, mem[MEM], *pmem;
  Splay *ch[2], *f;
  int val, rev, size;
  Splay () : val(-1), rev(0), size(0)
  { f = ch[0] = ch[1] = &nil; }
  Splay (int _val) : val(_val), rev(0), size(1)
  { f = ch[0] = ch[1] = &nil; }
  bool isr()
  { return f->ch[0] != this && f->ch[1] != this; }
  int dir()
  { return f->ch[0] == this ? 0 : 1; }
  void setCh(Splay *c, int d){
    ch[d] = c;
    if (c != &nil) c->f = this;
    pull();
  }
  void push(){
    if( !rev ) return;
    swap(ch[0], ch[1]);
    if (ch[0] != &nil) ch[0]->rev ^= 1;
    if (ch[1] != &nil) ch[1]->rev ^= 1;
    rev=0;
  }
  void pull(){
    size = ch[0]->size + ch[1]->size + 1;
    if (ch[0] != &nil) ch[0]->f = this;
    if (ch[1] != &nil) ch[1]->f = this;
  }
} Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::
    mem;
Splay *nil = &Splay::nil;
void rotate(Splay *x){
  Splay *p = x->f;
  int d = x->dir();
  if (!p->isr()) p->f->setCh(x, p->dir());
  else x->f = p->f;
  p->setCh(x->ch[!d], d);
  x->setCh(p, !d);
  p->pull(); x->pull();
}
vector<Splay*> splayVec;
void splay(Splay *x){
  splayVec.clear();
  for (Splay *q=x;; q=q->f){
    splayVec.push_back(q);
    if (q->isr()) break;
  }
  reverse(begin(splayVec), end(splayVec));
  for (auto it : splayVec) it->push();
  while (!x->isr()) {
    if (x->f->isr()) rotate(x);
    else if (x->dir()==x->f->dir())
      rotate(x->f),rotate(x);
    else rotate(x),rotate(x);
  }
}
Splay* access(Splay *x){
  Splay *q = nil;
  for (;x!=nil;x=x->f){
    splay(x);
    x->setCh(q, 1);
    q = x;
  }
  return q;
}
void evert(Splay *x){
  access(x);
  splay(x);
```

```
  x->rev ^= 1;
  x->push(); x->pull();
}
void link(Splay *x, Splay *y){
//  evert(x);
  access(x);
  splay(x);
  evert(y);
  x->setCh(y, 1);
}
void cut(Splay *x, Splay *y){
//  evert(x);
  access(y);
  splay(y);
  y->push();
  y->ch[0] = y->ch[0]->f = nil;
}
int N, Q;
Splay *vt[MXN];
int ask(Splay *x, Splay *y){
  access(x);
  access(y);
  splay(x);
  int res = x->f->val;
  if (res == -1) res=x->val;
  return res;
}
int main(int argc, char** argv){
  scanf("%d%d", &N, &Q);
  for (int i=1; i<=N; i++)
    vt[i] = new (Splay::pmem++) Splay(i);
  while (Q--) {
    char cmd[105];
    int u, v;
    scanf("%s", cmd);
    if (cmd[1] == 'i') {
      scanf("%d%d", &u, &v);
      link(vt[v], vt[u]);
    } else if (cmd[0] == 'c') {
      scanf("%d", &v);
      cut(vt[1], vt[v]);
    } else {
      scanf("%d%d", &u, &v);
      int res=ask(vt[u], vt[v]);
      printf("%d\n", res);
    }
  }
}
```

7.3 Black Magic

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> set_t;
#include <ext/pb_ds/assoc_container.hpp>
typedef cc_hash_table<int,int> umap_t;
typedef priority_queue<int> heap;
#include<ext/rope>
using namespace __gnu_cxx;
int main(){
  // Insert some entries into s.
  set_t s; s.insert(12); s.insert(505);
  // The order of the keys should be: 12, 505.
  assert(*s.find_by_order(0) == 12);
  assert(*s.find_by_order(3) == 505);
  // The order of the keys should be: 12, 505.
  assert(s.order_of_key(12) == 0);
  assert(s.order_of_key(505) == 1);
  // Erase an entry.
  s.erase(12);
  // The order of the keys should be: 505.
  assert(*s.find_by_order(0) == 505);
  // The order of the keys should be: 505.
  assert(s.order_of_key(505) == 0);

  heap h1 , h2; h1.join( h2 );

  rope<char> r[ 2 ];
  r[ 1 ] = r[ 0 ]; // persistenet
  string t = "abc";
  r[ 1 ].insert( 0 , t.c_str() );
  r[ 1 ].erase( 1 , 1 );
  cout << r[ 1 ].substr( 0 , 2 );
}
```

8 Others

8.1 # of Intersection of segments

```
#include<bits/stdc++.h>
using namespace std;
#define F first
#define S second
#define PB push_back
#define IOS ios_base::sync_with_stdio(0); cin.tie(0);
#define SZ(x) ((int)((x).size()))
#define ALL(x) begin(x),end(x)
#define REP(i,x) for (int i=0; i<(x); i++)
#define REP1(i,a,b) for (int i=(a); i<=(b); i++)

typedef long long ll;
typedef pair<ll,ll> pll;

typedef pll Point;
const int MXN = 100005;

Point operator + (const Point &a, const Point &b) {
    return Point(a.F+b.F, a.S+b.S); }
Point operator - (const Point &a, const Point &b) {
    return Point(a.F-b.F, a.S-b.S); }
ll operator * (const Point &a, const Point &b) { return
    a.F*b.F + a.S*b.S; }
ll operator % (const Point &a, const Point &b) { return
    a.F*b.S - a.S*b.F; }

struct Segment {
  int v,id;
  Point p,q;
  Segment () {}
  Segment (int _v, int _id, Point _p, Point _q) :
    v(_v), id(_id), p(_p), q(_q) {}
};
bool operator < (const Segment &a, const Segment &b) {
  if (a.p == b.q) return false;
  if (a.q == b.p) return true;
  if (a.p == b.p) return (a.q-a.p) % (b.q-a.p) > 0;
  if (a.q == b.q) return (a.p-a.q) % (b.p-a.q) < 0;
  if (a.p.F == b.p.F) return a.p.S < b.p.S;
  if (a.q.F == b.q.F) return a.q.S < b.q.S;
  if (a.p.F < b.p.F) return (a.q-a.p) % (b.p-a.p) > 0;
  else return (b.q-b.p) % (a.p-b.p) < 0;
}
bool operator == (const Segment &a, const Segment &b) {
  return tie(a.v,a.id,a.p,a.q) == tie(b.v,b.id,b.p,b.q)
      ;
}
struct Triangle {
  Point pt[3];
}ip[MXN];

const int MEM = 350004;
struct Treap {
  static Treap nil, mem[MEM], *pmem;
  Treap *l, *r;
  int sum,presum,size;
  Segment seg;
  Treap () : l(&nil), r(&nil), sum(0), presum(0), size
      (0), seg() {}
  Treap (Segment _val) :
    l(&nil), r(&nil), sum(_val.v), presum(max(_val.v,0)
        ), size(1), seg(_val) {}
} Treap::nil, Treap::mem[MEM], *Treap::pmem = Treap::
    mem;

int size(const Treap *t) { return t->size; }
void pull(Treap *t) {
  if (!size(t)) return;
```

```
  t->size = size(t->l) + size(t->r) + 1;
  t->sum = t->l->sum + t->seg.v + t->r->sum;
  t->presum = max(t->l->presum, t->l->sum + t->seg.v);
  t->presum = max(t->presum, t->l->sum + t->seg.v + t->
      r->presum);
}
Treap* merge(Treap *a, Treap *b) {
  if (!size(a)) return b;
  if (!size(b)) return a;
  Treap *t;
  if (rand() % (size(a) + size(b)) < size(a)) {
    t = a;
    t->r = merge(a->r, b);
  } else {
    t = b;
    t->l = merge(a, b->l);
  }
  pull(t);
  return t;
}
void split(Treap *t, int k, Treap *&a, Treap *&b) {
  if (!size(t)) a = b = &Treap::nil;
  else if (size(t->l) + 1 <= k) {
    a = t;
    split(t->r, k - size(t->l) - 1, a->r, b);
    pull(a);
  } else {
    b = t;
    split(t->l, k, a, b->l);
    pull(b);
  }
}
int get_rank(Treap *t, Segment x) {
  if (!size(t)) return 0;
  if (x < t->seg) return get_rank(t->l, x);
  return get_rank(t->r,x) + size(t->l) + 1;
}
Treap* find_leftist(Treap *t) {
  while (size(t->l)) t = t->l;
  return t;
}
Treap* find_rightist(Treap *t) {
  while (size(t->r)) t = t->r;
  return t;
}

int N;
vector<int> allx;
vector<Segment> _seg[3*MXN];
#define seg(x) _seg[(x)+100000]

inline void add_seg(Segment s) {
  seg(s.p.F).PB(s);
  if (s.q.F != s.p.F) seg(s.q.F).PB(s);
}
void predo() {
  allx.clear();
  REP(i,N) REP(j,3) {
    seg(ip[i].pt[j].F).clear();
    allx.PB(ip[i].pt[j].F);
  }
  sort(ALL(allx));
  allx.resize(unique(ALL(allx))-begin(allx));
  REP(i,N) {
    sort(ip[i].pt, ip[i].pt+3);
    Point *pt = ip[i].pt;
    Segment seg1 = Segment(1,i,pt[0],pt[1]);
    Segment seg2 = Segment(1,i,pt[0],pt[2]);
    Segment seg3 = Segment(1,i,pt[1],pt[2]);
    if (seg2 < seg1) seg1.v = -1;
    else seg2.v = -1;
    seg3.v = seg1.v;
    add_seg(seg1);
    add_seg(seg2);
    add_seg(seg3);
  }
}
inline int sgn(ll x) { return x < 0 ? -1 : x > 0; }
bool interPnt(Point p1, Point p2, Point q1, Point q2){
  ll c1 = (p2-p1)%(q1-p1), c2 = (p2-p1)%(q2-p1);
  ll c3 = (q2-q1)%(p1-q1), c4 = (q2-q1)%(p2-q1);
  return sgn(c1) * sgn(c2) <= 0 and sgn(c3) * sgn(c4)
      <= 0;
}
bool check_error(Segment a, Segment b) {
  if (a.id == b.id) return false;
  return interPnt(a.p,a.q,b.p,b.q);
}
int solve() {
  Treap::pmem = Treap::mem;
  Treap *rt = &Treap::nil;
  int res = 0;
  for (auto i:allx) {
    for (auto l:seg(i)) {
      int k = get_rank(rt, l);
      Treap *t,*tl,*tm,*tr;
      split(rt,k,tl,tr);
      t = find_rightist(tl);
      if (size(t) and check_error(t->seg,l)) return -1;
      t = find_leftist(tr);
      if (size(t) and check_error(t->seg,l)) return -1;
      rt = merge(tl,tr);
      if (l.p.F == i and l.p.F != l.q.F) {
        k = get_rank(rt, l);
        split(rt,k,tl,tr);
        tm = new (Treap::pmem++) Treap(l);
        rt = merge(merge(tl,tm),tr);
      }
    }
    for (auto l:seg(i)) {
      if (l.q.F == i and l.p.F != l.q.F) {
        Treap *tl,*tm,*tr;
        int k = get_rank(rt, l);
        split(rt,k-1,tl,tm);
        split(tm,1,tm,tr);
        Treap *t1=find_rightist(tl),*t2=find_leftist(tr
            );
        if (size(t1) and size(t2) and check_error(t1->
            seg,t2->seg)) return -1;
        rt = merge(tl,tr);
      }
    }
    res = max(res, rt->presum);
  }
  res++;
  return res;
}
int main() {
  IOS;
  int cas = 0;
  while (cin >> N) {
    if (N == -1) break;
    REP(i,N) {
      REP(j,3) cin >> ip[i].pt[j].F >> ip[i].pt[j].S;
    }
    predo();
    int ans = solve();
    cas++;
    cout << "Case " << cas << ": ";
    if (ans == -1) cout << "ERROR\n";
    else cout << ans << " shades\n";
  }

  return 0;
}
```