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 Misc

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

#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;
```

1.3 python-related

```
from fractions import Fraction
from decimal import Decimal, getcontext
getcontext().prec = 250 # set precision

itwo = Decimal(0.5)
two = Decimal(2)

N = 200
def angle(cosT):
  """given cos(theta) in decimal return theta"""
  for i in range(N):
    cosT = ((cosT + 1) / two) ** itwo
  sinT = (1 - cosT * cosT) ** itwo
  return sinT * (2 ** N)
pi = angle(Decimal(-1))
```

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; // LL
  int n,match[MXN],vx[MXN],vy[MXN];
```

```
  int edge[MXN][MXN],lx[MXN],ly[MXN],slack[MXN];
  // ^^^^ LL
  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 addEdge(int x, int y, int w) // LL
  { 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(int x=0, int y=0, int z=0) : u(x), v(y), c(z){}
};
int V, E, root;
Edge edges[MAXE];
inline int newV(){ return ++ V; }
inline void addEdge(int u, int v, int c)
{ 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 addEdge(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 , 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 from l[i] to r[i] must 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 Relabel to Front

```
// O(N^3), 0-base
struct Edge{
  int from, to, cap, flow;
  Edge(int _from, int _to, int _cap, int _flow = 0):
    from(_from), to(_to), cap(_cap), flow(_flow) {}
};
struct PushRelabel{
  int n;
  vector<Edge> edges;
  vector<int> count, h, inQ, excess;
  vector<vector<int> > G;
  queue<int> Q;
  PushRelabel(int _n):
    n(_n), count(_n<<1), G(_n), h(_n), inQ(_n), excess(
        _n) {}
  void addEdge(int from, int to, int cap) {
    G[from].push_back(edges.size());
    edges.push_back(Edge(from, to, cap));
    G[to].push_back(edges.size());
    edges.push_back(Edge(to, from, 0));
  }
  void enQueue(int u) {
    if(!inQ[u] && excess[u] > 0) Q.push(u), inQ[u] =
        true;
  }
  void Push(int EdgeIdx) {
    Edge & e = edges[EdgeIdx];
    int toPush = min<int>(e.cap - e.flow, excess[e.from
        ]);
    if(toPush > 0 && h[e.from] > h[e.to]) {
      e.flow += toPush;
      excess[e.to] += toPush;
      excess[e.from] -= toPush;
      edges[EdgeIdx^1].flow -= toPush;
      enQueue(e.to);
    }
  }
  void Relabel(int u) {
    count[h[u]] -= 1; h[u] = 2*n-2;
    for (size_t i = 0; i < G[u].size(); ++i) {
      Edge & e = edges[G[u][i]];
      if(e.cap > e.flow) h[u] = min(h[u], h[e.to]);
    }
    count[++h[u]] += 1;
  }
  void gapRelabel(int height) {
    for (int u = 0; u < n; ++u) if(h[u] >= height && h[
        u] < n) {
      count[h[u]] -= 1;
      count[h[u] = n] += 1;
      enQueue(u);
    }
  }
```

```
  void Discharge(int u) {
    for (size_t i = 0; excess[u] > 0 && i < G[u].size()
        ; ++i)
      Push(G[u][i]);
    if(excess[u] > 0) {
      if(h[u] < n && count[h[u]] < 2) gapRelabel(h[u]);
      else Relabel(u);
    }
    else if(!Q.empty()) { // dequeue
      Q.pop();
      inQ[u] = false;
    }
  }
  int solve(int src, int snk) {
    h[src] = n; inQ[src] = inQ[snk] = true;
    count[0] = n - (count[n] = 1);
    for (size_t i = 0; i < G[src].size(); ++i) {
      excess[src] += edges[G[src][i]].cap;
      Push(G[src][i]);
    }
    while (!Q.empty())
      Discharge(Q.front());
    return excess[snk];
  }
};
```

2.10 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
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
   16   65536       65537      1    3
   20   1048576     7340033    7    3 */
// (must be 2^k)
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 , 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;
    }
  }
};
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 Poly operator

```
struct PolyOp {
#define FOR(i, c) for (int i = 0; i < (c); ++i)
  NTT<P, root, MAXN> ntt;
  static int nxt2k(int x) {
    int i = 1; for (; i < x; i <<= 1); return i;
  }
  void Mul(int n, LL a[], int m, LL b[], LL c[]) {
    static LL aa[MAXN], bb[MAXN];
    int N = nxt2k(n+m);
    copy(a, a+n, aa); fill(aa+n, aa+N, 0);
    copy(b, b+m, bb); fill(bb+m, bb+N, 0);
    ntt(N, aa); ntt(N, bb);
    FOR(i, N) c[i] = aa[i] * bb[i] % P;
    ntt(N, c, 1);
  }
  void Inv(int n, LL a[], LL b[]) {
    // ab = aa^-1 = 1 mod x^(n/2)
    // (b - a^-1)^2 = 0 mod x^n
    // bb - a^-2 + 2 ba^-1 = 0
    // bba - a^-1 + 2b = 0
    // bba + 2b = a^-1
    static LL tmp[MAXN];
    if (n == 1) {b[0] = ntt.inv(a[0], P); return;}
    Inv((n+1)/2, a, b);
    int N = nxt2k(n*2);
    copy(a, a+n, tmp);
    fill(tmp+n, tmp+N, 0);
    fill(b+n, b+N, 0);
    ntt(N, tmp); ntt(N, b);
    FOR(i, N) {
      LL t1 = (2 - b[i] * tmp[i]) % P;
      if (t1 < 0) t1 += P;
      b[i] = b[i] * t1 % P;
    }
    ntt(N, b, 1);
    fill(b+n, b+N, 0);
  }
  void Div(int n, LL a[], int m, LL b[], LL d[], LL r
      []) {
    // Ra = Rb * Rd mod x^(n-m+1)
    // Rd = Ra * Rb^-1 mod
    static LL aa[MAXN], bb[MAXN], ta[MAXN], tb[MAXN];
    if (n < m) {copy(a, a+n, r); fill(r+n, r+m, 0);
        return;}
    // d: n-1 - (m-1) = n-m (n-m+1 terms)
    copy(a, a+n, aa); copy(b, b+m, bb);
    reverse(aa, aa+n); reverse(bb, bb+m);
    Inv(n-m+1, bb, tb);
    Mul(n-m+1, ta, n-m+1, tb, d);
    fill(d+n-m+1, d+n, 0); reverse(d, d+n-m+1);
    // r: m-1 - 1 = m-2 (m-1 terms)
    Mul(m, b, n-m+1, d, ta);
    FOR(i, n) { r[i] = a[i] - ta[i]; if (r[i] < 0) r[i]
        += P; }
  }
  void dx(int n, LL a[], LL b[]) { REP(i, 1, n-1) b[i
      -1] = i * a[i] % P; }
  void Sx(int n, LL a[], LL b[]) {
    b[0] = 0;
    FOR(i, n) b[i+1] = a[i] * ntt.iv[i+1] % P;
  }
  void Ln(int n, LL a[], LL b[]) {
    // Integral a' a^-1 dx
    static LL a1[MAXN], a2[MAXN], b1[MAXN];
    int N = nxt2k(n*2);
    dx(n, a, a1); Inv(n, a, a2);
    Mul(n-1, a1, n, a2, b1);
    Sx(n+n-1-1, b1, b);
    fill(b+n, b+N, 0);
  }
  void Exp(int n, LL a[], LL b[]) {
    // Newton method to solve g(a(x)) = ln b(x) - a(x)
        = 0
    // b' = b - g(b(x)) / g'(b(x))
    // b' = b (1 - lnb + a)
    static LL lnb[MAXN], c[MAXN], tmp[MAXN];
    assert(a[0] == 0); // dont know exp(a[0]) mod P
    if (n == 1) {b[0] = 1; return;}
    Exp((n+1)/2, a, b);
    fill(b+(n+1)/2, b+n, 0);
    Ln(n, b, lnb);
    fill(c, c+n, 0); c[0] = 1;
    FOR(i, n) {
      c[i] += a[i] - lnb[i];
      if (c[i] < 0) c[i] += P;
      if (c[i] >= P) c[i] -= P;
    }
    Mul(n, b, n, c, tmp);
    copy(tmp, tmp+n, b);
  }
} polyop;
```

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.
ll 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=(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=magic number size
  // 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=magic[s]%(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

```
LL solve(LL x1, LL m1, LL x2, LL m2) {
  LL g = __gcd(m1, m2);
  if((x2 - x1) % g) return -1;// no sol
  m1 /= g; m2 /= g;
  pair<LL,LL> p = gcd(m1, m2);
  LL lcm = m1 * m2 * g;
  LL res = p.first * (x2 - x1) * m1 + x1;
  return (res % lcm + lcm) % lcm;
}
```

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 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)};
}
```

3.11 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;
}
```

3.12 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)));
        }
    }
  }
}
```

3.13 Romberg

```
// Estimates the definite integral of
// \int_a^b f(x) dx
template<class T>
double romberg( T& f, double a, double b, double eps=1e
    -8){
  vector<double>t; double h=b-a,last,curr; int k=1,i=1;
  t.push_back(h*(f(a)+f(b))/2);
  do{ last=t.back(); curr=0; double x=a+h/2;
    for(int j=0;j<k;j++) curr+=f(x), x+=h;
    curr=(t[0] + h*curr)/2; double k1=4.0/3.0,k2
        =1.0/3.0;
    for(int j=0;j<i;j++){ double temp=k1*curr-k2*t[j];
      t[j]=curr; curr=temp; k2/=4*k1-k2; k1=k2+1;
    } t.push_back(curr); k*=2; h/=2; i++;
  }while( fabs(last-curr) > eps);
  return t.back();
}
```

3.14 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;
}
```

3.15 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]);
}
```

3.16 Result

- Lucas' Theorem :
  For $n, m \in \mathbb{Z}^*$ and prime P, $C(m,n)$ mod $P = \Pi(C(m_i, n_i))$ where m_i is the i-th digit of m in base P.
- Stirling Numbers(permutation $|P| = n$ with k cycles):
  $S(n,k)$ = coefficient of x^k in $\Pi_{i=0}^{n-1}(x+i)$
- Stirling Numbers(Partition n elements into k non-empty set):
  $S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$
- 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=0; 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 Pt& o , const Pt& a , const Pt& b ){
  LL ret = ( a - o ) ^ ( b - o );
  return (ret > 0) - (ret < 0);
}
// p1 == p2 || q1 == q2 need to be handled
bool banana( const Pt& p1 , const Pt& p2 ,
             const Pt& q1 , const Pt& 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 circle and segment

```
bool Inter( const Pt& p1 , const Pt& p2 , Circle& cc ){
  Pt dp = p2 - p1;
  double a = dp * dp;
  double b = 2 * ( dp * ( p1 - cc.O ) );
  double c = cc.O * cc.O + p1 * p1 - 2 * ( cc.O * p1 )
      - cc.R * cc.R;
  double bb4ac = b * b - 4 * a * c;
  return !( fabs( a ) < eps or bb4ac < 0 );
}
```

4.5 Intersection of 2 circles

4.6 Circle cover

```
#define N 1021
struct CircleCover{
  int C; Circ 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( Circ& a , Circ& b , Pt& p1 , Pt& p2 ){
    Pt o1 = a.O , o2 = b.O;
    D r1 = a.R , r2 = b.R;
    if( norm( o1 - o2 ) > r1 + r2 ) return {};
    if( norm( o1 - o2 ) < max(r1, r2) - min(r1, r2) )
        return {};
    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 Teve {
    Pt p; D ang; int add;
    Teve() {}
    Teve(Pt _a, D _b, int _c):p(_a), ang(_b), add(_c){}
    bool operator<(const Teve &a)const
    {return ang < a.ang;}
  }eve[ N * 2 ];
  // strict: x = 0, otherwise x = -1
  bool disjuct( Circ& a, Circ &b, int x )
  {return sign( norm( a.O - b.O ) - a.R - b.R ) > x;}
  bool contain( Circ& a, Circ &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 ++] = Teve(bb, B, 1);
          eve[E ++] = Teve(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;
        }
      }
    }
  }
};
```

4.7 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;
}
};
```

4.8 Tangent line of two circles

```
vector<Line> go( const Cir& c1 , const Cir& c2 , int
    sign1 ){
  // sign1 = 1 for outer tang, -1 for inter tang
  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;
  double c = ( c1.R - sign1 * c2.R ) / d;
  if( c * c > 1 ) return ret;
  double h = sqrt( max( 0.0 , 1.0 - c * c ) );
  for( int sign2 = 1 ; sign2 >= -1 ; sign2 -= 2 ){
    Pt n = { v.X * c - sign2 * h * v.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;
}
```

4.9 KD Tree

```
const int MXN = 100005;
struct KDTree {
  struct Nd {
    int x,y,x1,y1,x2,y2;
    int id,f;
    Nd *L, *R;
  }tree[MXN];
  int n;
  Nd *root;
  LL dis2(int x1, int y1, int x2, int y2) {
    LL dx = x1-x2; LL dy = y1-y2;
    return dx*dx+dy*dy;
  }
  static bool cmpx(Nd& a, Nd& b){ return a.x<b.x; }
  static bool cmpy(Nd& a, Nd& 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);
  }
  Nd* 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(Nd* r, int x, int y, LL d2){
    LL 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(Nd* r, int x, int y, int &mID, LL &md2){
    if (!r || !touch(r, x, y, md2)) return;
    LL 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;
    LL d2 = 102938475612345678LL;
    nearest(root, x, y, id, d2);
    return id;
  }
}tree;
```

4.10 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);
  }
};
```

4.11 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 );
}
```

4.12 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;
```

4.13 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 Min dist on Cuboid

```
typedef LL T;
T r;
void turn(T i, T j, T x, T y, T z,
          T x0, T y0, T L, T W, T H) {
  if (z==0) { T R = x*x+y*y; if (R<r) r=R; return; }
  if(i>=0 && i< 2) turn(i+1, j, x0+L+z, y, x0+L-x,
                        x0+L, y0, H, W, L);
  if(j>=0 && j< 2) turn(i, j+1, x, y0+W+z, y0+W-y,
                        x0, y0+W, L, H, W);
  if(i<=0 && i>-2) turn(i-1, j, x0-z, y, x-x0,
                        x0-H, y0, H, W, L);
  if(j<=0 && j>-2) turn(i, j-1, x, y0-z, y-y0,
                        x0, y0-H, L, H, W);
}
T solve(T L, T W, T H,
        T x1, T y1, T z1, T x2, T y2, T z2){
  if( z1!=0 && z1!=H ){
    if( y1==0 || y1==W )
      swap(y1,z1), swap(y2,z2), swap(W,H);
  }else swap(x1,z1), swap(x2,z2), swap(L,H);
  if (z1==H) z1=0, z2=H-z2;
  r=INF; turn(0,0,x2-x1,y2-y1,z2,-x1,-y1,L,W,H);
  return r;
}
```

4.15 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 REP(i, s, e) for(int i = (s); i <= (e); i++)
#define REPD(i, s, e) for(int i = (s); i >= (e); i--)
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 prt[MAXN][LOG], head[MAXN];
  // head[ u ] : head of the chain contains u
  void dfssz(int u, int p){
    dep[u] = dep[p] + 1;
    prt[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){
    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 != prt[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);
    int diff = dep[b] - dep[a];
    REPD(k, LOG-1, 0) if(diff & (1<<k)){
      b = prt[b][k];
    }
    if(a == b) return a;
    REPD(k, LOG-1, 0) if(prt[a][k] != prt[b][k]){
      a = prt[a][k]; b = prt[b][k];
    }
    return prt[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)
      prt[i][k] = prt[prt[i][k-1]][k-1];
  }
  vector< PII > getPath( int u , int v ){
    vector< PII > res;
    while( tid[ u ] < tid[ head[ v ] ] ){
      res.push_back( PII(tid[ head[ v ] ] , tid[ v ]) )
          ;
      v = prt[ head[ v ] ][ 0 ];
    }
    res.push_back( PII( 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< PII >& path = tree.getPath( u , v )
     * for( PII 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

```
#define N 111
struct MaxClique{ // 0-base
  typedef bitset< N > Int;
  Int linkto[ N ] , v[ N ];
  int n;
  void init( int _n ){
    n = _n;
    for( int i = 0 ; i < n ; i ++ ){
      linkto[ i ].reset();
      v[ i ].reset();
    }
  }
  void addEdge( int a , int b ){
    v[ a ][ b ] = v[ b ][ a ] = 1;
  }
  int popcount(const Int& val)
  { return val.count(); }
  int lowbit(const Int& val)
  { return val._Find_first(); }
  int ans , stk[ N ];
  int id[ N ] , di[ N ] , deg[ N ];
  Int cans;
  void maxclique(int elem_num, Int candi){
    if(elem_num > ans){
      ans = elem_num;
      cans.reset();
      for( int i = 0 ; i < elem_num ; i ++ )
        cans[ id[ stk[ i ] ] ] = 1;
    }
    int potential = elem_num + popcount(candi);
    if(potential <= ans) return;
    int pivot = lowbit(candi);
    Int smaller_candi = candi & (~linkto[pivot]);
    while(smaller_candi.count() && potential > ans){
      int next = lowbit(smaller_candi);
      candi[next] = !candi[next];
      smaller_candi[ next ] = !smaller_candi[ next ];
      potential --;
      if(next == pivot || (smaller_candi & linkto[next
          ]).count() ){
        stk[elem_num] = next;
        maxclique(elem_num + 1, candi & linkto[next]);
      }
    }
  }
  int solve(){
    for( int i = 0 ; i < n ; i ++ ){
      id[ i ] = i;
      deg[ i ] = v[ i ].count();
    }
    sort( id , id + n , [&](int id1, int id2){
          return deg[id1] > deg[id2]; } );
    for( int i = 0 ; i < n ; i ++ )
      di[ id[ i ] ] = i;
    for( int i = 0 ; i < n ; i ++ )
      for( int j = 0 ; j < n ; j ++ )
        if( v[ i ][ j ] )
          linkto[ di[ i ] ][ di[ j ] ] = 1;
    Int cand; cand.reset();
    for( int i = 0 ; i < n ; i ++ )
      cand[ i ] = 1;
    ans = 1;
    cans.reset(); cans[ 0 ] = 1;
    maxclique(0, cand);
    return ans;
  }
} solver;
```

5.4 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 addEdge(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++;
      }
  }
};
```

5.5 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(); }
```

5.6 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;
```

5.7 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;
```

5.8 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;
```

5.9 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;
```

5.10 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 addEdge(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);
        }
      }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;
```

5.11 Min Mean Cycle

```
/* minimum mean cycle O(VE) */
struct MMC{
#define E 101010
#define V 1021
#define inf 1e9
#define eps 1e-6
  struct Edge { int v,u; double c; };
  int n, m, prv[V][V], prve[V][V], vst[V];
  Edge e[E];
  vector<int> edgeID, cycle, rho;
  double d[V][V];
  void init( int _n )
  { n = _n; m = 0; }
  // WARNING: TYPE matters
  void addEdge( int vi , int ui , double ci )
  { e[ m ++ ] = { vi , ui , ci }; }
  void bellman_ford() {
    for(int i=0; i<n; i++) d[0][i]=0;
    for(int i=0; i<n; i++) {
      fill(d[i+1], d[i+1]+n, inf);
      for(int j=0; j<m; j++) {
        int v = e[j].v, u = e[j].u;
        if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {
          d[i+1][u] = d[i][v]+e[j].c;
          prv[i+1][u] = v;
          prve[i+1][u] = j;
        }
      }
    }
  }
  double solve(){
    // returns inf if no cycle, mmc otherwise
    double mmc=inf;
    int st = -1;
    bellman_ford();
    for(int i=0; i<n; i++) {
      double avg=-inf;
      for(int k=0; k<n; k++) {
        if(d[n][i]<inf-eps) avg=max(avg,(d[n][i]-d[k][i
            ])/(n-k));
        else avg=max(avg,inf);
      }
      if (avg < mmc) tie(mmc, st) = tie(avg, i);
    }
    FZ(vst); edgeID.clear(); cycle.clear(); rho.clear()
        ;
    for (int i=n; !vst[st]; st=prv[i--][st]) {
      vst[st]++;
      edgeID.PB(prve[i][st]);
      rho.PB(st);
    }
    while (vst[st] != 2) {
      int v = rho.back(); rho.pop_back();
      cycle.PB(v);
      vst[v]++;
    }
    reverse(ALL(edgeID));
    edgeID.resize(SZ(cycle));
    return mmc;
  }
} mmc;
```

5.12 Directed Graph Min Cost Cycle

```
// works in O(N M)
#define INF 100000000000000LL
#define N 5010
#define M 200010
struct edge{
```

```
  int to; LL w;
  edge(int a=0, LL b=0): to(a), w(b){}
};
struct node{
  LL d; int u, next;
  node(LL a=0, int b=0, int c=0): d(a), u(b), next(c){}
}b[M];
struct DirectedGraphMinCycle{
  vector<edge> g[N], grev[N];
  LL dp[N][N], p[N], d[N], mu;
  bool inq[N];
  int n, bn, bsz, hd[N];
  void b_insert(LL d, int u){
    int i = d/mu;
    if(i >= bn) return;
    b[++bsz] = node(d, u, hd[i]);
    hd[i] = bsz;
  }
  void init( int _n ){
    n = _n;
    for( int i = 1 ; i <= n ; i ++ )
      g[ i ].clear();
  }
  void addEdge( int ai , int bi , LL ci )
  { g[ai].push_back(edge(bi,ci)); }
  LL solve(){
    fill(dp[0], dp[0]+n+1, 0);
    for(int i=1; i<=n; i++){
      fill(dp[i]+1, dp[i]+n+1, INF);
      for(int j=1; j<=n; j++) if(dp[i-1][j] < INF){
        for(int k=0; k<(int)g[j].size(); k++)
          dp[i][g[j][k].to] =min(dp[i][g[j][k].to],
                                 dp[i-1][j]+g[j][k].w);
      }
    }
    mu=INF; LL bunbo=1;
    for(int i=1; i<=n; i++) if(dp[n][i] < INF){
      LL a=-INF, b=1;
      for(int j=0; j<=n-1; j++) if(dp[j][i] < INF){
        if(a*(n-j) < b*(dp[n][i]-dp[j][i])){
          a = dp[n][i]-dp[j][i];
          b = n-j;
        }
      }
      if(mu*b > bunbo*a)
        mu = a, bunbo = b;
    }
    if(mu < 0) return -1; // negative cycle
    if(mu == INF) return INF; // no cycle
    if(mu == 0) return 0;
    for(int i=1; i<=n; i++)
      for(int j=0; j<(int)g[i].size(); j++)
      g[i][j].w *= bunbo;
    memset(p, 0, sizeof(p));
    queue<int> q;
    for(int i=1; i<=n; i++){
      q.push(i);
      inq[i] = true;
    }
    while(!q.empty()){
      int i=q.front(); q.pop(); inq[i]=false;
      for(int j=0; j<(int)g[i].size(); j++){
        if(p[g[i][j].to] > p[i]+g[i][j].w-mu){
          p[g[i][j].to] = p[i]+g[i][j].w-mu;
          if(!inq[g[i][j].to]){
            q.push(g[i][j].to);
            inq[g[i][j].to] = true;
          }
        }
      }
    }
    for(int i=1; i<=n; i++) grev[i].clear();
    for(int i=1; i<=n; i++)
      for(int j=0; j<(int)g[i].size(); j++){
        g[i][j].w += p[i]-p[g[i][j].to];
        grev[g[i][j].to].push_back(edge(i, g[i][j].w));
      }
    LL mldc = n*mu;
    for(int i=1; i<=n; i++){
      bn=mldc/mu, bsz=0;
      memset(hd, 0, sizeof(hd));
      fill(d+i+1, d+n+1, INF);
      b_insert(d[i]=0, i);
      for(int j=0; j<=bn-1; j++) for(int k=hd[j]; k; k=
          b[k].next){
        int u = b[k].u;
        LL du = b[k].d;
        if(du > d[u]) continue;
        for(int l=0; l<(int)g[u].size(); l++) if(g[u][l
            ].to > i){
          if(d[g[u][l].to] > du + g[u][l].w){
            d[g[u][l].to] = du + g[u][l].w;
            b_insert(d[g[u][l].to], g[u][l].to);
          }
        }
      }
      for(int j=0; j<(int)grev[i].size(); j++) if(grev[
          i][j].to > i)
        mldc=min(mldc,d[grev[i][j].to] + grev[i][j].w);
    }
    return mldc / bunbo;
  }
} graph;
```

5.13 K-th Shortest Path

```
// time: O(|E| \lg |E| + |V| \lg |V| + K)
// memory: O(|E| \lg |E| + |V|)
struct KSP{ // 1-base
  struct nd{
    int u, v, d;
    nd(int ui = 0, int vi = 0, int di = INF)
    { u = ui; v = vi; d = di; }
  };
  struct heap{
    nd* edge; int dep; heap* chd[4];
  };
  static int cmp(heap* a,heap* b)
  { return a->edge->d > b->edge->d; }
  struct node{
    int v; LL d; heap* H; nd* E;
    node(){}
    node(LL _d, int _v, nd* _E)
    { d =_d; v = _v; E = _E; }
    node(heap* _H, LL _d)
    { H = _H; d = _d; }
    friend bool operator<(node a, node b)
    { return a.d > b.d; }
  };
  int n, k, s, t, dst[ N ];
  nd *nxt[ N ];
  vector<nd*> g[ N ], rg[ N ];
  heap *nullNd, *head[ N ];
  void init( int _n , int _k , int _s , int _t ){
    n = _n; k = _k; s = _s; t = _t;
    for( int i = 1 ; i <= n ; i ++ ){
      g[ i ].clear(); rg[ i ].clear();
      nxt[ i ] = head[ i ] = NULL;
      dst[ i ] = -1;
    }
  }
  void addEdge( int ui , int vi , int di ){
    nd* e = new nd(ui, vi, di);
    g[ ui ].push_back( e );
    rg[ vi ].push_back( e );
  }
  queue<int> dfsQ;
  void dijkstra(){
    while(dfsQ.size()) dfsQ.pop();
    priority_queue<node> Q;
    Q.push(node(0, t, NULL));
    while (!Q.empty()){
      node p = Q.top(); Q.pop();
      if(dst[p.v] != -1) continue;
      dst[ p.v ] = p.d;
      nxt[ p.v ] = p.E;
      dfsQ.push( p.v );
      for(auto e: rg[ p.v ])
        Q.push(node(p.d + e->d, e->u, e));
    }
  }
  heap* merge(heap* curNd, heap* newNd){
```

```
    if(curNd == nullNd) return newNd;
    heap* root = new heap;
    memcpy(root, curNd, sizeof(heap));
    if(newNd->edge->d < curNd->edge->d){
      root->edge = newNd->edge;
      root->chd[2] = newNd->chd[2];
      root->chd[3] = newNd->chd[3];
      newNd->edge = curNd->edge;
      newNd->chd[2] = curNd->chd[2];
      newNd->chd[3] = curNd->chd[3];
    }
    if(root->chd[0]->dep < root->chd[1]->dep)
      root->chd[0] = merge(root->chd[0],newNd);
    else
      root->chd[1] = merge(root->chd[1],newNd);
    root->dep = max(root->chd[0]->dep, root->chd[1]->
        dep) + 1;
    return root;
  }
  vector<heap*> V;
  void build(){
    nullNd = new heap;
    nullNd->dep = 0;
    nullNd->edge = new nd;
    fill(nullNd->chd, nullNd->chd+4, nullNd);
    while(not dfsQ.empty()){
      int u = dfsQ.front(); dfsQ.pop();
      if(!nxt[ u ]) head[ u ] = nullNd;
      else head[ u ] = head[nxt[ u ]->v];
      V.clear();
      for( auto&& e : g[ u ] ){
        int v = e->v;
        if( dst[ v ] == -1 ) continue;
        e->d += dst[ v ] - dst[ u ];
        if( nxt[ u ] != e ){
          heap* p = new heap;
          fill(p->chd, p->chd+4, nullNd);
          p->dep = 1;
          p->edge = e;
          V.push_back(p);
        }
      }
      if(V.empty()) continue;
      make_heap(V.begin(), V.end(), cmp);
#define L(X) ((X<<1)+1)
#define R(X) ((X<<1)+2)
      for( size_t i = 0 ; i < V.size() ; i ++ ){
        if(L(i) < V.size()) V[i]->chd[2] = V[L(i)];
        else V[i]->chd[2]=nullNd;
        if(R(i) < V.size()) V[i]->chd[3] = V[R(i)];
        else V[i]->chd[3]=nullNd;
      }
      head[u] = merge(head[u], V.front());
    }
  }
  vector<LL> ans;
  void first_K(){
    ans.clear();
    priority_queue<node> Q;
    if( dst[ s ] == -1 ) return;
    ans.push_back( dst[ s ] );
    if( head[s] != nullNd )
      Q.push(node(head[s], dst[s]+head[s]->edge->d));
    for( int _ = 1 ; _ < k and not Q.empty() ; _ ++ ){
      node p = Q.top(), q; Q.pop();
      ans.push_back( p.d );
      if(head[ p.H->edge->v ] != nullNd){
        q.H = head[ p.H->edge->v ];
        q.d = p.d + q.H->edge->d;
        Q.push(q);
      }
      for( int i = 0 ; i < 4 ; i ++ )
        if( p.H->chd[ i ] != nullNd ){
          q.H = p.H->chd[ i ];
          q.d = p.d - p.H->edge->d + p.H->chd[ i ]->
              edge->d;
          Q.push( q );
        }
    }
  }
  void solve(){
    dijkstra();
    build();
    first_K();
  }
} solver;
```

6 String

6.1 PalTree

```
/*
 * sfail: compressed fail links with same diff
 * O(lgn): length of sfail link path
 */
const int MAXN = 1e6+10;
struct PalT{
  int tot,lst;
  int nxt[MAXN][26], len[MAXN];
  int fail[MAXN], diff[MAXN], sfail[MAXN];
  char* s;
  int newNode(int l, int _fail) {
    int res = ++tot;
    fill(nxt[res], nxt[res]+26, 0);
    len[res] = l, fail[res] = _fail;
    diff[res] = l - len[_fail];
    if (diff[res] == diff[_fail])
      sfail[res] = sfail[_fail];
    else
      sfail[res] = _fail;
    return res;
  }
  void push(int p) {
    int np = lst;
    int c = s[p]-'a';
    while (p-len[np]-1 < 0 || s[p] != s[p-len[np]-1])
      np = fail[np];
    if ((lst=nxt[np][c])) return;
    int nq_f = 0;
    if (len[np]+2 == 1) nq_f = 2;
    else {
      int tf = fail[np];
      while (p-len[tf]-1 < 0 || s[p] != s[p-len[tf]-1])
        tf = fail[tf];
      nq_f = nxt[tf][c];
    }
    int nq = newNode(len[np]+2, nq_f);
    nxt[np][c] = nq;
    lst=nq;
  }
  void init(char* _s){
    s = _s;
    tot = 0;
    newNode(-1, 1);
    newNode(0, 1);
    diff[2] = 0;
    lst = 2;
  }
} palt;
```

6.2 KMP

```
int failure[MAXN];
void KMP(string& t, string& p)
{
    if (p.size() > t.size()) return;
    for (int i=1, j=failure[0]=-1; i<p.size(); ++i)
    {
        while (j >= 0 && p[j+1] != p[i])
            j = failure[j];
        if (p[j+1] == p[i]) j++;
        failure[i] = j;
    }
    for (int i=0, j=-1; i<t.size(); ++i)
    {
        while (j >= 0 && p[j+1] != t[i])
            j = failure[j];
        if (p[j+1] == t[i]) j++;
```

```
        if (j == p.size()-1)
        {
            cout << i - p.size() + 1<<" ";
            j = failure[j];
        }
    }
}
```

6.3 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)
}
```

6.4 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;
```

6.5 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]);
        }
      }
    }
  }
};
```

6.6 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;
    }
  }
}
```

6.7 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;
```

6.8 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;
    }
  }
}
```

6.9 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);
}
```

6.10 Cyclic LCS

```
#define L 0
#define LU 1
#define U 2
const int mov[3][2]={0,-1, -1,-1, -1,0};
int al,bl;
char a[MAXL*2],b[MAXL*2]; // 0-indexed
int dp[MAXL*2][MAXL];
char pred[MAXL*2][MAXL];
inline int lcs_length(int r) {
  int i=r+al,j=bl,l=0;
  while(i>r) {
    char dir=pred[i][j];
    if(dir==LU) l++;
    i+=mov[dir][0];
    j+=mov[dir][1];
  }
  return l;
}
inline void reroot(int r) { // r = new base row
  int i=r,j=1;
  while(j<=bl&&pred[i][j]!=LU) j++;
  if(j>bl) return;
  pred[i][j]=L;
  while(i<2*al&&j<=bl) {
    if(pred[i+1][j]==U) {
      i++;
      pred[i][j]=L;
    } else if(j<bl&&pred[i+1][j+1]==LU) {
      i++;
      j++;
      pred[i][j]=L;
    } else {
      j++;
    }
  }
}
int cyclic_lcs() {
  // a, b, al, bl should be properly filled
```

```
  // note: a WILL be altered in process
  //       -- concatenated after itself
  char tmp[MAXL];
  if(al>bl) {
    swap(al,bl);
    strcpy(tmp,a);
    strcpy(a,b);
    strcpy(b,tmp);
  }
  strcpy(tmp,a);
  strcat(a,tmp);
  // basic lcs
  for(int i=0;i<=2*al;i++) {
    dp[i][0]=0;
    pred[i][0]=U;
  }
  for(int j=0;j<=bl;j++) {
    dp[0][j]=0;
    pred[0][j]=L;
  }
  for(int i=1;i<=2*al;i++) {
    for(int j=1;j<=bl;j++) {
      if(a[i-1]==b[j-1]) dp[i][j]=dp[i-1][j-1]+1;
      else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
      if(dp[i][j-1]==dp[i][j]) pred[i][j]=L;
      else if(a[i-1]==b[j-1]) pred[i][j]=LU;
      else pred[i][j]=U;
    }
  }
  // do cyclic lcs
  int clcs=0;
  for(int i=0;i<al;i++) {
    clcs=max(clcs,lcs_length(i));
    reroot(i+1);
  }
  // recover a
  a[al]='\0';
  return clcs;
}
```

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 (int _val=-1) : 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);
  }
}
int id(Splay *x) { return x - Splay::mem + 1; }
Splay* access(Splay *x){
  Splay *q = nil;
  for (;x!=nil;x=x->f){
    splay(x);
    x->setCh(q, 1);
```

```
    q = x;
  }
  return q;
}
void chroot(Splay *x){
  access(x);
  splay(x);
  x->rev ^= 1;
  x->push(); x->pull();
}
void link(Splay *x, Splay *y){
  access(x);
  splay(x);
  chroot(y);
  x->setCh(y, 1);
}
void cut_p(Splay *y) {
  access(y);
  splay(y);
  y->push();
  y->ch[0] = y->ch[0]->f = nil;
}
void cut(Splay *x, Splay *y){
  chroot(x);
  cut_p(y);
}
Splay* get_root(Splay *x) {
  access(x);
  splay(x);
  for(; x->ch[0] != nil; x = x->ch[0])
    x->push();
  splay(x);
  return x;
}
bool conn(Splay *x, Splay *y) {
  x = get_root(x);
  y = get_root(y);
  return x == y;
}
Splay* lca(Splay *x, Splay *y) {
  access(x);
  access(y);
  splay(x);
  if (x->f == nil) return x;
  else return x->f;
}
```

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 Find max tangent(x,y is increasing)

```
typedef long long LL;
const int MAXN = 100010;
struct Coord{
  LL x, y;
  Coord operator - (Coord ag) const{
    Coord res;
    res.x = x - ag.x;
    res.y = y - ag.y;
    return res;
  }
}sum[MAXN], pnt[MAXN], ans, calc;

inline bool cross(Coord a, Coord b, Coord c){
  return (c.y-a.y)*(c.x-b.x) > (c.x-a.x)*(c.y-b.y);
}

int main(){
  int n, l, np, st, ed, now;
  scanf("%d %d\n", &n, &l);
  sum[0].x = sum[0].y = np = st = ed = 0;
  for (int i = 1, v; i <= n; i++){
    scanf("%d", &v);
    sum[i].y = sum[i - 1].y + v;
    sum[i].x = i;
  }
  ans.x = now = 1;
  ans.y = -1;
  for (int i = 0; i <= n - l; i++){
    while (np > 1 &&
           cross(pnt[np - 2], pnt[np - 1], sum[i]))
      np--;
    if (np < now && np != 0) now = np;
    pnt[np++] = sum[i];
    while (now < np &&
           !cross(pnt[now - 1], pnt[now], sum[i + l]))
      now++;
    calc = sum[i + l] - pnt[now - 1];
    if (ans.y * calc.x < ans.x * calc.y){
      ans = calc;
      st = pnt[now - 1].x;
      ed = i + l;
    }
  }
  double res = (sum[ed].y - sum[st].y) /
               (sum[ed].x - sum[st].x);
  printf("%f\n", res);
  return 0;
}
```