# Team notebook

Convex Chull

December 21, 2022

## Contents

# 1 Advice

```
Pre-submit:
Are time limits close? If so, generate
    max cases. Is the memory usage fine?
    Could anything overflow? Make sure to
    submit the right file.

Wrong answer: Print your solution! Print
    debug output, as well. Are you
    clearing all datastructures between
    test cases? Can your algorithm handle
    the whole range of input?
Read the full problem statement again.
    Do you handle all corner cases
    correctly? Have you understood the
    problem correctly? Any uninitialized
    variables? Any overflows? Confusing N
    and M, i and j, etc.? Are you sure
    your algorithm works? What special
    cases have you not thought of?
Are you sure the STL functions you use
    work as you think? Add some
    assertions, maybe resubmit Create
    some testcases to run your algorithm
    on. Go through the algorithm for a
    simple case.
Go through this list again. Explain your
    algorithm to a team mate. Ask the
    team mate to look at your code. Go
    for a small walk, e.g. to the toilet.
    Is your output format correct?
    Rewrite your solution from the start
    or let a team mate do it.

Runtime error: Have you tested all
    corner cases locally? Any
    uninitialized variables? Are you
    reading or writing outside the range
    of any vector? Any assertions that
    might fail? Any possible division by
    0? (mod 0 for example). Any possible
    infinite recursion? Invalidated
    pointers or iterators? Are you using
    too much memory? Debug with resubmits.

Time limit exceeded: Do you have any
    possible infinite loops? What is the
    complexity of your algorithm? Are you
    copying a lot of unnecessary data?
    (References) How big is the input and
    output? (consider scanf) Avoid
    vector, map. (use
    arrays/unordered_map) What do your
    team mates think about your algorithm?

Memory limit exceeded: What is the max
    amount of memory your algorithm
    should need? Are you clearing
    alldatastructures between test cases?

Primes - 10001st prime is 1299721,
    100001st prime is 15485867 Large
    primes - 999999937, 1e9+7, 987646789,
    987101789; 78498 primes less than
    10^6 The number of divisors of n is
    at most around 100, for n<5e4, 500
    for n<=1e7, 2000 for n<1e10, 200,000
    for n<1e19 7! = 5040, 8! = 40320, 9!
    = 362880, 10! = 362880, 11! = 4.0e7,
    12! = 4.8e8, 15! = 1.3e12, 20! = 2e18

The number of divisors of n is at most
    around 100 for n < 5e4, 500 for n <
    1e7, 2000 for n < 1e10, 200 000 for n
    < 1e19.

Articulation points and bridges
    articulation point:- there exist
    child : dfslow[child] >= dfsnum[curr]
    bridge :- tree ed: dfslow[ch] >
    dfsnum[par];

A connected multigraph has an Euler path
    but not an Euler circuit if and only
    if it has exactly two vertices of odd
    degree

Binomial coefficients - base case ncn
    and nc0 = 1; recursion is nCk =
    (n-1)C(k-1)+(n-1)Ck

Catalan numbers - used in valid
    paranthesis expressions - formula is
    Cn = summation{i=0 to n-1}
    (CiCn-i-1); Another formula is Cn =
```

2nCn/(n+1). There are Cn binary trees of n nodes and Cn-1 rooted trees of n nodes

Derangements - D(n) = (n-1)(D(n-1)+D(n-2))

Burnsides Lemma - number of equivalence classes = (summation I(pi))/n : I(pi) are number of fixed points. Usual formula: [summation {i=0 to n-1} k^gcd(i,n)]/n

Stirling numbers - first kind - permutations of n elements with k disjoint cycles. s(n+1,k) = ns(n,k)+s(n,k-1). s(0,0) = 1, s(n,0) = 0 if n>0. Summation {k=0 to n} s(n,k) = n!

Stirling numbers - Second kind - partition n objects into k non empty subsets. S(n+1,k) = kS(n,k) + S(n,k-1). S(0,0) = 1, S(n,0) = 0 if n>0. S(n,k) = (summation{j=0 to k} [(-1)^(k-j)(kCj)j^n])/k!

Hermite identity - summation{k=0 to n-1} floor[(x+k)/n] = floor[nx]

Kirchoff matrix tree theorem - number of spanning trees in a graph is determinant of Laplacian Matrix with one row and column removed, where L = degree matrix - adjacency matrix

Expected value tricks:

1. Linearity of Expectation: E(X+Y) = E(X)+E(Y)
2. Contribution to the sum - If we want to find the sum over many ways/possibilities, we should consider every element (maybe a number, or a pair or an edge) and count how many times it will be added to the answer.
3. For independent events - E(XY) = E(X)E(Y)
4. Ordered pairs (Super interpretation of square) - The square of the size of a set is equal to the number of ordered pairs of elements in the set. So we iterate over pairs and for each we compute the contribution to the answer. Similarly, the k-th power is equal to the number of sequences (tuples) of length k.
5. Powers technique - If you want to maintain the sum of k-th powers, it might help to also maintain the sum of smaller powers. For example, if the sum of 0-th, 1-th and 2-nd powers is S0, S1 and S2, and we increase every element by x, the new sums are S0, S1+S0x and S2 + 2S1x + x^2S0.

---

# 2 Aho Corasick

---

```
struct AhoCorasick{
 enum {alpha=26,first='a'};
 struct Node{
  int back, next[alpha], start = -1, end
      = -1, nmatches = 0;
  Node(int
      v){memset(next,v,sizeof(next));}};
 vector<Node> N;
 vector<int> backp;
 inline void insert(string &s,int j){
  assert(!s.empty());
  int n=0;
  for(auto &c: s){
   int &m=N[n].next[c-first];
   if(m==-1){n=m=N.size();
       N.emplace_back(-1);}
   else n=m;
  }
  if(N[n].end==-1) N[n].start=j;
  backp.push_back(N[n].end);
  N[n].end=j;
  N[n].nmatches++;}
 void clear(){
  N.clear();
  backp.clear();}
 void create(vector<string>& pat){
  N.emplace_back(-1);
  for(int i=0;i<pat.size();++i)
      insert(pat[i],i);
  N[0].back=N.size();
  N.emplace_back(0);
  queue<int> q;
  for(q.push(0);!q.empty();q.pop()){
   int n=q.front(),prev=N[n].back;
   for(int i=0;i<alpha;++i){
```

```
    int
       &ed=N[n].next[i],y=N[prev].next[i];
    if(ed==-1) ed=y;
    else{
     N[ed].back=y;
     (N[ed].end==-1 ?
        N[ed].end:backp[N[ed].start])=N[y].end;
     N[ed].nmatches+=N[y].nmatches;
     q.push(ed);}}}}
 ll find(string word){
  int n=0;
  // vector<int> res;
  ll count=0;
  for(auto &c: word){
   n=N[n].next[c-first];
   // res.push_back(N[n].end);
   count+=N[n].nmatches;}
  return count;}};
struct AhoOnline{
 int sz=0;
 vector<string> v[25];
 AhoCorasick c[25];
 void add(string &p){
  int val=__builtin_ctz(~sz);
  auto &cur=v[val];
  for(int i=0;i<val;++i){
   for(auto &it: v[i]) cur.push_back(it);
   c[i].clear();
   v[i].clear();}
  cur.push_back(p);
  c[val].create(cur);
  ++sz;}
 ll query(string &p){
  ll ans=0;
  for(int i=0;i<25;++i){
   if((1<<i)&sz) ans+=c[i].find(p);
   if((1<<i)>=sz) break;}
  return ans;}} add,del;
```

# 3 Anti-DSU

```
int par[N], siz[N], op[N];
// DON'T TAKE 0 AS A NODE
int findset(int a) {
    if(par[a]==a)
    return a;
    return par[a]=findset(par[a]);}
void unionset(int a, int b) {
    if(a==0 || b==0)
    return;
    a=findset(a);
    b=findset(b);
    if(a==b)
    return;
    if(siz[a]>siz[b])
    swap(a, b);
    par[a]=b;
    siz[b]+=siz[a];
    unionset(op[a], op[b]);
    op[b]=max(op[b], op[a]);}
```

# 4 Auxiliary Tree

```
set<pair<int, int> > vertSet;
for(int i = 0; i < k; i++)
vertSet.insert({out[a[i]], a[i]});
vector<pair <int, pii> > compressedEdges;
while((int)vertSet.size() > 1)
{
    int u = vertSet.begin()->second;
    vertSet.erase({out[u], u});
    int v = vertSet.begin()->second;
    int lca2 = lca(u, v);
    compressedEdges.push_back({cal(u,
        lca2), {u, lca2}});
    vertSet.insert({out[lca2], lca2});
}
sort(all(compressedEdges));
```

# 5 Centroid Decomposition

```
struct centroid {
  vvi adj; int n;
  vi vis,par,sz;
  void init(int s){
    n=s; adj=vvi(n,vi());
    vis=vi(n,0); par=sz=vi(n);}
  void addEdge(int a,int b){
    adj[a].pb(b); adj[b].pb(a);}
  int findSize(int v,int p=-1){
    if(vis[v]) return 0;
    sz[v]=1;
    for(int x:adj[v]){
      if(x!=p) sz[v]+=findSize(x,v);}
    return sz[v];}
  int findCentroid(int v,int p,int n){
    for(int x:adj[v])
      if(x!=p && !vis[x] && sz[x]>n/2)
        return findCentroid(x,v,n);
```

```
    return v;}
  void initCentroid(int v=0,int p=-1){
    findSize(v);
    int c=findCentroid(v,-1,sz[v]);
    vis[c]=true; par[c] = p;
    for(int x:adj[c])
      if(!vis[x]) initCentroid(x,c);}
};
```

# 6 Convex Hull and Li Chao tree

```
// Li chao Tree (can be made persistent)
struct Line{
 ll m, c;
 Line(ll mm=0,ll cc=-3e18): m(mm),c(cc){}
 inline ll get(const int &x){return
     m*x+c;}
 inline ll operator [](const int
     &x){return m*x+c;} };
vector<Line> LN;
struct node{
 node *lt,*rt;
 int Ln;
 node(const int&l): Ln(l),lt(0),rt(0){};
 inline ll operator[](const int &x){
     return LN[Ln].get(x);}
 inline ll get(const int &x){return
     LN[Ln].get(x);}};
const static int LX=-(1e9+1),RX=1e9+1;
struct Dynamic_Hull{ /* Max hull */
 node *root=0;
 void add(int l,node* &it,int lx=LX,int
     rx=RX){
  if(it==0) it=new node(l);
  if(it->get(lx)>=LN[l].get(lx) and
     it->get(rx)>=LN[l].get(rx)) return;
  if(it->get(lx)<=LN[l].get(lx) and
     it->get(rx)<=LN[l].get(rx)){
   it->Ln=l;
   return;}
  int mid=(lx+rx)>>1;
  if(it->get(lx)<LN[l][lx])
     swap(it->Ln,l);
  if(it->get(mid)>=LN[l][mid]){
   add(l,it->rt,mid+1,rx);}
     else{
    swap(it->Ln,l);
    add(l,it->lt,lx,mid); }}
  inline void add(int
     ind){add(ind,root);}
  inline void add(int m,int
     c){LN.pb(Line(m,c));add(LN.size()-1,root);}
  ll get(int &x,node* &it,int lx=LX,int
     rx=RX){
    if(it==0) return -3e18; // Max hull
    ll ret=it->get(x);
    int mid=(lx+rx)>>1;
    if(x<=mid)
      ret=max(ret,get(x,it->lt,lx,mid));
    else
      ret=max(ret,get(x,it->rt,mid+1,rx));
    return ret;}
  inline ll get(int x){return
     get(x,root);}};
struct Hull{
struct line {
  ll m,c;
  ll eval(ll x){return m*x+c;}
  ld intersectX(line l){return
      (ld)(c-l.c)/(l.m-m);}
  line(ll m,ll c): m(m),c(c){}};
deque<line> dq;
v32 ints;
Hull(int n){ints.clear(); forn(i,n)
    ints.pb(i); dq.clear();}
// Dec order of slopes
void add(line cur){
  while(dq.size()>=2 &&
      cur.intersectX(dq[0])>=dq[0].intersectX(d
    dq.pop_front();
  dq.push_front(cur);}
void add(const ll &m,const ll
    &c){add(line(m,c));}
// query sorted dec.
// ll getval(ll x){
//   while(dq.size()>=2 &&
    dq.back().eval(x)<=dq[dq.size()-2].eval(x))
//     dq.pop_back();
//   return dq.back().eval(x);
// }
// arbitary query
ll getval(ll x,deque<line> &dq){
  auto cmp = [&dq](int idx,ll x){return
      dq[idx].intersectX(dq[idx+1])<x;};
  int idx =
      *lower_bound(ints.begin(),ints.begin()+
      dq.size()-1,x,cmp);
  return dq[idx].eval(x);}
ll get(const ll &x){return
    getval(x,dq);}};
```

# 7 DSU with Rollback

```
struct dsu{
 int sz;
   v32 par,rk;
   stack<int> st;
   void reset(int n){
       rk.assign(n,1);
       par.resize(n);
       iota(all(par),0);
       sz=n;
   }
   int getpar(int i){
       return (par[i]==i)?
           i:getpar(par[i]);
   }
   bool con(int i,int j){
       return getpar(i)==getpar(j);
   }
   bool join(int i,int j){
       i=getpar(i),j=getpar(j);
       if(i==j) return 0;
       --sz;
       if(rk[j]>rk[i]) swap(i,j);
       par[j]=i,rk[i]+=rk[j];
       st.push(j);
       return 1;
   }
   int moment(){
    return st.size();
   }
   void revert(int tm){
    while(st.size()>tm){
     auto tp=st.top();
     rk[par[tp]]-=rk[tp];
     par[tp]=tp;
     st.pop();
     ++sz;
    }
   }
} d;
```

# 8 Dinic

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long
        cap) : v(v), u(u), cap(cap) {}};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n),
        s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);}

    void add_edge(int v, int u, long
        long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;}

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap -
                    edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] !=
                    -1)
                    continue;
                level[edges[id].u] =
                    level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;}

    long long dfs(int v, long long
        pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid <
            (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u]
                || edges[id].cap -
```

```
            edges[id].flow < 1)
            continue;
        long long tr = dfs(u,
            min(pushed, edges[id].cap
            - edges[id].flow));
        if (tr == 0)
            continue;
        edges[id].flow += tr;
        edges[id ^ 1].flow -= tr;
        return tr;
    }
    return 0;}

long long flow() {
    long long f = 0;
    while (true) {
        fill(level.begin(),
            level.end(), -1);
        level[s] = 0;
        q.push(s);
        if (!bfs())
            break;
        fill(ptr.begin(), ptr.end(),
            0);
        while (long long pushed =
            dfs(s, flow_inf)) {
            f += pushed;
        }
    }
    return f;}};
```

# 9 Euler Path

```
procedure FindEulerPath(V)
  1. iterate through all the edges
     outgoing from vertex V;
       remove this edge from the graph,
       and call FindEulerPath from the
           second end of this edge;
  2. add vertex V to the answer.
```

# 10 Extended Euclidean GCD

```
int egcd(int a,int b, int* x, int* y){
    if(a==0){
        *x=0;*y=1;
        return b;}
    int x1,y1;
    int gcd=egcd(b%a,a,&x1,&y1);
    *x=y1-(b/a)*x1;
    *y=x1;
    return gcd;}
```

# 11 FFT

```
long double pi = acos(-1);
class FFT{
    public:
    static void
        reorder(vector<complex<long
        double>> &A){
        ll n = A.size();
        for (int i = 1, j = 0; i < n;
            i++) {
            int bit = n >> 1;
            for (; j & bit; bit >>= 1){
                j ^= bit;
            }
            j ^= bit;
            if (i < j){
                swap(A[i], A[j]);
            }
        }
    }
    static void fft(vector<complex<long
        double>> &A, bool invert = false){
        ll n = A.size();
        if(n==1) return;
        reorder(A);
        for(ll sz=2; sz<=n; sz*=2){
            long double angle =
                ((2*pi)/sz) * (1-2*invert);
            complex<long double>
                sz_root(cos(angle),
                sin(angle));
            for(ll i=0; i<n; i+=sz){
                complex<long double>
                    cur_w(1);
                rep(j, 0, sz/2){
                    complex<long double>
                        ff = A[i+j], ss =
                        A[i+j+sz/2]*cur_w;
                    A[i+j] = ff + ss;
                    A[i+j+sz/2] = ff - ss;
                    cur_w *= sz_root;
                }
            }
```

```
        }
        if(invert)
            for(auto &x: A)
                x/=n;
    }
    static vector <ll> multiply(vector
        <ll> &A, vector <ll> &B){
        vector<complex<long double>>
            dA(all(A)), dB(all(B));
        ll n = 1;
        while(n < A.size() + B.size())
        n *= 2;
        dA.resize(n);
        dB.resize(n);
        fft(dA);
        fft(dB);
        rep(i, 0, n)
            dA[i] *= dB[i];
        fft(dA, true);
        vector <ll> ans(n);
        rep(i, 0, n)
        ans[i] = round(dA[i].real());
        reverse(all(ans));
        while(ans.back() == 0)
            ans.pop_back();
        reverse(all(ans));
        return ans;
    }
};
```

# 12 FWHT

```
namespace fwht{
template<typename T>
void hadamard_xor(vector<T> &a){
 int n = a.size();
 for(int k = 1 ; k < n ; k <<= 1){
  for(int i = 0 ; i < n ; i += 2*k){
   for(int j = 0 ; j < k ; j++){
    T x = a[i + j];
    T y = a[i + j + k];
    a[i + j] = x + y;
    a[i + j + k] = x - y;}}}}
template<typename T>
void hadamard_or(vector<T> &a,bool
    inverse){
 int n = a.size();
 for(int k = 1 ; k < n ; k <<= 1){
  for(int i = 0 ; i < n ; i += 2*k){
   for(int j = 0 ; j < k ; j++){
    T x = a[i + j];
    T y = a[i + j + k];
    if(inverse){
     a[i + j] = x;
     a[i + j +k] = y - x;
    }
    else{
     a[i + j] = x;
     a[i + j + k] = x + y;}}}}}
template<typename T>
void hadamard_and(vector<T> &a,bool
    inverse){
 int n = a.size();
 for(int k = 1 ; k < n ; k <<= 1){
  for(int i = 0 ; i < n ; i += 2*k){
   for(int j = 0 ; j < k ; j++){
    T x = a[i + j];
    T y = a[i + j +k];
    if(inverse){
     a[i + j] = x - y;
     a[i + j + k] = y;
    }
    else{
     a[i + j] = x + y;
     a[i + j + k] = y; }}}}}
template<typename T>
vector<T> multiply(vector<T>
    a,vector<T> b){
 int eq = (b==a);
 int n = 1 ;
 while (n <
     (int)max(a.size(),b.size())){
  n <<= 1;
 }
 a.resize(n);
 b.resize(n);
 hadamard_xor(a);
 if (eq) b = a; else hadamard_xor(b);
 for(int i = 0 ; i < n ; i++){
  a[i]*=b[i];
 }
 hadamard_xor(a);
 T q = static_cast<T>(n);
 for(int i = 0 ; i < n ; i++){
  a[i]/=q;
 }
 return a;
}}
```

# 13 Fenwick 2D

```
//BIT<N, M, K> b; N x M x K
    (3-dimensional) BIT
//b.update(x, y, z, P); // add P to
    (x,y,z)
//b.query(x1, x2, y1, y2, z1, z2); //
    query between (x1, y1, z1) and (x2,
    y2, z2)
inline int lastbit(int x){
  return x&(-x);}
template <int N, int... Ns>
struct BIT<N, Ns...> {
  BIT<Ns...> bit[N + 1];
  template<typename... Args>
  void update(int pos, Args... args) {
    for (; pos <= N;
        bit[pos].update(args...), pos +=
        lastbit(pos));}
  template<typename... Args>
  int query(int l, int r, Args... args) {
    int ans = 0;
    for (; r >= 1; ans +=
        bit[r].query(args...), r -=
        lastbit(r));
    for (--l; l >= 1; ans -=
        bit[l].query(args...), l -=
        lastbit(l));
    return ans;}};
// Another implementation
struct FenwickTree2D {
    vector<vector<int>> bit;
    int n, m;
    // init(...) { ... }
    int sum(int x, int y) {
        int ret = 0;
        for (int i = x; i >= 0; i = (i &
            (i + 1)) - 1)
            for (int j = y; j >= 0; j =
                (j & (j + 1)) - 1)
                ret += bit[i][j];
        return ret;}
    void add(int x, int y, int delta) {
        for (int i = x; i < n; i = i | (i
            + 1))
            for (int j = y; j < m; j = j
                | (j + 1))
                bit[i][j] += delta;}};
```

## 14 Gaussian Elimination, Base 2

```
struct Gaussbase2{
 int numofbits=20;
 int rk=0;
 v32 Base;
 Gaussbase2() {clear();}
 void clear(){
  rk=0;
  Base.assign(numofbits,0);}
 Gaussbase2& operator = (Gaussbase2 &g){
  forn(i,numofbits) Base[i]=g.Base[i];
  rk=g.rk;}
 bool canbemade(int x){
  rforn(i,numofbits-1)
      x=min(x,x^Base[i]);
  return x==0;}
 void Add(int x){
  rforn(i,numofbits-1){
   if((x>>i)&1){
    if(!Base[i]){
     Base[i]=x;
     rk++;
     return;
    }else x^=Base[i];}}}
 int maxxor(){
  int ans=0;
  rforn(i,numofbits-1){
   if(ans < (ans^Base[i])) ans^=Base[i];}
  return ans;}};
```

## 15 Gaussian Elimination

```
int gauss (vector <vector<double> > a,
    vector<double> &ans){
    int n = (int) a.size();
    int m = (int) a[0].size()-1;

    vector<int> where(m,-1);
    for(int col=0, row=0;col<m && row<n;
        ++col){
        int sel = row;
        for(int i=row;i<n;++i){
            if(abs(a[i][col]) >
                abs(a[sel][col])){
                sel = i;}}
        if(abs(a[sel][col])<EPS) continue;
        for(int i=col; i<=m; ++i){
            swap(a[sel][i],a[row][i]);}
        where[col] = row;
        for(int i=0;i<n;++i){
```

```
        if(i!=row){
            double c =
                a[i][col]/a[row][col];
            for(int j=col;j<=m;++j){
                a[i][j] -=
                    a[row][j]*c;}}}
        ++row;}
    ans.assign(m,0);
    for(int i=0;i<m;++i){
        if(where[i]!=-1){
            ans[i] =
                a[where[i]][m]/a[where[i]][i];}}
    for(int i=0;i<n;++i){
        double sum=0;
        for(int j=0;j<m;++j){
            sum+=ans[j]*a[i][j];}
        if(abs(sum-a[i][m])>EPS)
            return 0;}
    for(int i=0;i<m;++i){
        if(where[i]==-1) return MOD;}
    return 1;}
```

# 16 Geometry

```
const int MAX_SIZE = 1000;
const double PI = 2.0*acos(0.0);
struct PT
{
 double x,y;
 double length() {return sqrt(x*x+y*y);}
 int normalize(){
 // normalize the vector to unit length;
    return -1 if the vector is 0
  double l = length();
  if(fabs(l)<EPS) return -1;
  x/=l; y/=l;
  return 0;}
 PT operator-(PT a){
  PT r;
  r.x=x-a.x; r.y=y-a.y;
  return r;}
 PT operator+(PT a){
  PT r;
  r.x=x+a.x; r.y=y+a.y;
  return r;}
 PT operator*(double sc){
  PT r;
  r.x=x*sc; r.y=y*sc;
  return r;}};

bool operator<(const PT& a,const PT& b){
 if(fabs(a.x-b.x)<EPS) return a.y<b.y;
 return a.x<b.x;}
double dist(PT& a, PT& b){
 // the distance between two points
 return sqrt((a.x-b.x)*(a.x-b.x) +
    (a.y-b.y)*(a.y-b.y));}
double dot(PT& a, PT& b){
 // the inner product of two vectors
 return(a.x*b.x+a.y*b.y);}
double cross(PT& a, PT& b){
 return(a.x*b.y-a.y*b.x);}

// ================================
// The Convex Hull
// ================================

int sideSign(PT& p1,PT& p2,PT& p3){
// which side is p3 to the line p1->p2?
   returns: 1 left, 0 on, -1 right
 double sg =
    (p1.x-p3.x)*(p2.y-p3.y)-(p1.y -
    p3.y)*(p2.x-p3.x);
 if(fabs(sg)<EPS) return 0;
 if(sg>0) return 1;
 return -1;}
bool better(PT& p1,PT& p2,PT& p3){
 // used by convec hull: from p3, if p1
    is better than p2
 double sg = (p1.y -
    p3.y)*(p2.x-p3.x)-(p1.x-p3.x)*(p2.y-p3.y);
 //watch range of the numbers
 if(fabs(sg)<EPS){
  if(dist(p3,p1)>dist(p3,p2))return true;
  else return false;
 }
 if(sg<0) return true;
 return false;}
void vex2(vector<PT> vin,vector<PT>&
   vout){
 // vin is not pass by reference, since
    we will rotate it
 vout.clear();
 int n=vin.size();
 sort(vin.begin(),vin.end());
 PT stk[MAX_SIZE];
 int pstk, i;
 // hopefully more than 2 points
 stk[0] = vin[0];
 stk[1] = vin[1];
 pstk = 2;
```

```
 for(i=2; i<n; i++){
  if(dist(vin[i], vin[i-1])<EPS)
      continue;
  while(pstk > 1 && better(vin[i],
      stk[pstk-1], stk[pstk-2]))
  pstk--;
  stk[pstk] = vin[i];
  pstk++;}
 for(i=0; i<pstk; i++)
    vout.push_back(stk[i]);
 // turn 180 degree
 for(i=0; i<n; i++){
  vin[i].y = -vin[i].y;
  vin[i].x = -vin[i].x;}
 sort(vin.begin(), vin.end());
 stk[0] = vin[0];
 stk[1] = vin[1];
 pstk = 2;
 for(i=2; i<n; i++){
  if(dist(vin[i], vin[i-1])<EPS)
      continue;
  while(pstk > 1 && better(vin[i],
      stk[pstk-1], stk[pstk-2]))
  pstk--;
  stk[pstk] = vin[i];
  pstk++;}
 for(i=1; i<pstk-1; i++){
  stk[i].x= -stk[i].x; // dont forget
      rotate 180 d back.
  stk[i].y= -stk[i].y;
  vout.push_back(stk[i]);}}

int isConvex(vector<PT>& v){
 // test whether a simple polygon is
    convex
 // return 0 if not convex, 1 if
    strictly convex,
 // 2 if convex but there are points
    unnecesary
 // this function does not work if the
    polycon is self intersecting
 // in that case, compute the convex
    hull of v, and see if both have the
    same area
 int i,j,k;
 int c1=0; int c2=0; int c0=0;
 int n=v.size();
 for(i=0;i<n;i++){
  j=(i+1)%n;
  k=(j+1)%n;
  int s=sideSign(v[i], v[j], v[k]);
  if(s==0) c0++;
  if(s>0) c1++;
  if(s<0) c2++;
 }
 if(c1 && c2) return 0;
 if(c0) return 2;
 return 1;}
// ==============================
// Areas
// ==============================
double trap(PT a, PT b){
 // Used in various area functions
 return (0.5*(b.x - a.x)*(b.y + a.y));}
double area(vector<PT> &vin){
 // Area of a simple polygon, not
    neccessary convex
 int n = vin.size();
 double ret = 0.0;
 for(int i = 0; i < n; i++) ret +=
    trap(vin[i], vin[(i+1)%n]);
 return fabs(ret);}
double peri(vector<PT> &vin){
// Perimeter of a simple polygon, not
   neccessary convex
 int n = vin.size();
 double ret = 0.0;
 for(int i = 0; i < n; i++) ret +=
    dist(vin[i], vin[(i+1)%n]);
 return ret;}

double triarea(PT a, PT b, PT c){
 return
    fabs(trap(a,b)+trap(b,c)+trap(c,a));}

double height(PT a, PT b, PT c){
 // height from a to the line bc
 double s3 = dist(c, b);
 double ar=triarea(a,b,c);
 return(2.0*ar/s3);}

// ==============================
// Points and Lines
// ==============================
int intersection( PT p1, PT p2, PT p3,
   PT p4, PT &r ) {
 // two lines given by p1->p2, p3->p4 r
    is the intersection point
 // return -1 if two lines are parallel
 double d = (p4.y - p3.y)*(p2.x-p1.x) -
    (p4.x - p3.x)*(p2.y - p1.y);
 if( fabs( d ) < EPS ) return -1;
 // might need to do something special!!!
 double ua, ub;
```

```
 ua = (p4.x - p3.x)*(p1.y-p3.y) -
     (p4.y-p3.y)*(p1.x-p3.x);
 ua /= d;
 // ub = (p2.x - p1.x)*(p1.y-p3.y) -
     (p2.y-p1.y)*(p1.x-p3.x);
 //ub /= d;
 r = p1 + (p2-p1)*ua;
 return 0;}

void closestpt( PT p1, PT p2, PT p3, PT
    &r ){
 // the closest point on the line p1->p2
     to p3
 if( fabs( triarea( p1, p2, p3 ) ) < EPS
     ) { r = p3; return; }
 PT v = p2-p1;
 v.normalize();
 double pr; // inner product
 pr = (p3.y-p1.y)*v.y + (p3.x-p1.x)*v.x;
 r = p1+v*pr;}

int hcenter( PT p1, PT p2, PT p3, PT& r
    ){
 // point generated by altitudes
 if( triarea( p1, p2, p3 ) < EPS )
     return -1;
 PT a1, a2;
 closestpt( p2, p3, p1, a1 );
 closestpt( p1, p3, p2, a2 );
 intersection( p1, a1, p2, a2, r );
 return 0;}
int center( PT p1, PT p2, PT p3, PT& r ){
 // point generated by circumscribed
     circle
 if( triarea( p1, p2, p3 ) < EPS )
     return -1;
 PT a1, a2, b1, b2;
 a1 = (p2+p3)*0.5;
 a2 = (p1+p3)*0.5;
 b1.x = a1.x - (p3.y-p2.y);
 b1.y = a1.y + (p3.x-p2.x);
 b2.x = a2.x - (p3.y-p1.y);
 b2.y = a2.y + (p3.x-p1.x);
 intersection( a1, b1, a2, b2, r );
 return 0;}

int bcenter( PT p1, PT p2, PT p3, PT& r
    ){
 // angle bisection
 if( triarea( p1, p2, p3 ) < EPS )
     return -1;
 double s1, s2, s3;
 s1 = dist( p2, p3 );
 s2 = dist( p1, p3 );
 s3 = dist( p1, p2 );
 double rt = s2/(s2+s3);
 PT a1,a2;
 a1 = p2*rt+p3*(1.0-rt);
 rt = s1/(s1+s3);
 a2 = p1*rt+p3*(1.0-rt);
 intersection( a1,p1, a2,p2, r );
 return 0;}

// ===============================
// Angles
// ===============================
double angle(PT& p1, PT& p2, PT& p3){
 // angle from p1->p2 to p1->p3, returns
     -PI to PI
 PT va = p2-p1;
 va.normalize();
 PT vb; vb.x=-va.y; vb.y=va.x;
 PT v = p3-p1;
 double x,y;
 x=dot(v, va);
 y=dot(v, vb);
 return(atan2(y,x));}

double angle(double a, double b, double
    c){
 // in a triangle with sides a,b,c, the
     angle between b and c
 // we do not check if a,b,c is a
     triangle here
 double cs=(b*b+c*c-a*a)/(2.0*b*c);
 return(acos(cs));}

void rotate(PT p0, PT p1, double a, PT&
    r){
 // rotate p1 around p0 clockwise, by
     angle a
 //  dont  pass by reference for p1, so
     r and p1 can be the same
 p1 = p1-p0;
 r.x = cos(a)*p1.x-sin(a)*p1.y;
 r.y = sin(a)*p1.x+cos(a)*p1.y;
 r = r+p0;}

void reflect(PT& p1, PT& p2, PT p3, PT&
    r){
 // p1->p2 line, reflect p3 to get r.
 if(dist(p1, p3)<EPS) {r=p3; return;}
 double a=angle(p1, p2, p3);
 r=p3;
```

```
 rotate(p1, r, -2.0*a, r);}

// ==============================
// points, lines, and circles
// ==============================

int pAndSeg(PT& p1, PT& p2, PT& p){
 // the relation of the point p and the
    segment p1->p2.
 // 1 if point is on the segment; 0 if
    not on the line; -1 if on the line
    but not on the segment
 double s=triarea(p, p1, p2);
 if(s>EPS) return(0);
 double sg=(p.x-p1.x)*(p.x-p2.x);
 if(sg>EPS) return(-1);
 sg=(p.y-p1.y)*(p.y-p2.y);
 if(sg>EPS) return(-1);
 return(1);}

int lineAndCircle(PT& oo, double r, PT&
   p1, PT& p2, PT& r1, PT& r2){
 // returns -1 if there is no
    intersection
 // returns 1 if there is only one
    intersection
 PT m;
 closestpt(p1,p2,oo,m);
 PT v = p2-p1;
 v.normalize();
 double r0=dist(oo, m);
 if(r0>r+EPS) return -1;
 if(fabs(r0-r)<EPS){
  r1=r2=m;
  return 1;}
```

```
 double dd = sqrt(r*r-r0*r0);
 r1 = m-v*dd; r2 = m+v*dd;
 return 0;}

int CAndC(PT o1, double r1, PT o2,
   double r2, PT &q1, PT& q2){

 // intersection of two circles
 // -1 if no intersection or infinite
    intersection
 // 1 if only one point

 double r=dist(o1,o2);
 if(r1<r2) { swap(o1,o2); swap(r1,r2); }
 if(r<EPS) return(-1);
 if(r>r1+r2+EPS) return(-1);
 if(r<r1-r2-EPS) return(-1);
 PT v = o2-o1; v.normalize();
 q1 = o1+v*r1;
 if(fabs(r-r1-r2)<EPS ||
    fabs(r+r2-r1)<EPS)
 { q2=q1; return(1); }
 double a=angle(r2, r, r1);
 q2=q1;
 rotate(o1, q1, a, q1);
 rotate(o1, q2, -a, q2);
 return 0;}

int pAndPoly(vector<PT> pv, PT p){
 // the relation of the point and the
    simple polygon
 // 1 if p is in pv; 0 outside; -1 on
    the polygon
 int i, j;
 int n=pv.size();
```

```
 pv.push_back(pv[0]);
 for(i=0;i<n;i++) if(pAndSeg(pv[i],
    pv[i+1], p)==1) return(-1);
 for(i=0;i<n;i++) pv[i] = pv[i]-p;
 p.x=p.y=0.0;
 double a, y;
 while(1){
  a=(double)rand()/10000.00;
  j=0;
  for(i=0;i<n;i++){
   rotate(p, pv[i], a, pv[i]);
   if(fabs(pv[i].x)<EPS) j=1;}
  if(j==0){
   pv[n]=pv[0];
   j=0;
   for(i=0;i<n;i++) if(pv[i].x*pv[i+1].x
      < -EPS){
   y=pv[i+1].y-pv[i+1].x*(pv[i].y-pv[i+1].y)/(p
    if(y>0) j++;}
   return(j%2);}}
 return 1;}

double maxdist(vector<PT> poly){
 //Rotating calliper method to find max
    distance in a convex polygon
 // If not convex, first run convex hull
    algo then use this function
 int n = poly.size();
 double res = 0;
 for(int i = 0, j = n<2?0:1; i<j;i++){
  for(;; j = (j+1)%n){
   res =
     max(res,dist(poly[i],poly[j])*dist(poly[
     poly[j]));
   PT dummy;
```

```
 dummy.x = 0, dummy.y = 0;
 if(sideSign(dummy,poly[(j+1)%n]-poly[j],poly[i+1]
    -poly[i]) >= 0) break;}
 }
 return res;
}

template <class T> inline int sgn(const
   T& x) { return (T(0) < x) - (x <
   T(0)); }

template <class F1, class F2>
int pointVsConvexPolygon(const
   Point<F1>& point, const Polygon<F2>&
   poly, int top) {
  if (point < poly[0] || point >
     poly[top]) return 1;
  auto orientation = ccw(point,
     poly[top], poly[0]);
  if (orientation == 0) {
    if (point == poly[0] || point ==
       poly[top]) return 0;
    return top == 1 || top + 1 ==
       poly.size() ? 0 : -1;
  } else if (orientation < 0) {
    auto itRight =
       lower_bound(begin(poly) + 1,
       begin(poly) + top, point);
    return sgn(ccw(itRight[0], point,
       itRight[-1]));
  } else {
    auto itLeft =
       upper_bound(poly.rbegin(),
       poly.rend() - top-1, point);
    return sgn(ccw(itLeft ==
       poly.rbegin() ? poly[0] :
       itLeft[-1], point, itLeft[0]));
  }
}

PT perp(PT p) {
 PT r; r.x = -p.y; r.y = p.x;
 return r;}

//Code for tangency between two circles
//if there are 2 tangents, it fills out
   with two pairs of points
//if there is 1 tangent, the circles are
   tangent to each other at some point
   P, out just contains P 4 times
//if there are 0 tangents, it does
   nothing
//if the circles are identical, it
   aborts.
//Set r2 = 0 to get tangency from a
   point to a circle
int tangents(pt o1, double r1, pt o2,
   double r2, bool inner, vector<
pair<pt,pt>> &out) {
if (inner) r2 = -r2;
pt d = o2-o1;
double dr = r1-r2, d2 = sq(d), h2 =
   d2-dr*dr;
if (d2 == 0 || h2 < 0) {assert(h2 != 0);
   return 0;}
for (double sign : {-1,1}) {
pt v = (d*dr + perp(d)*sqrt(h2)*sign)/d2;
out.push_back({o1 + v*r1, o2 + v*r2});
}
return 1 + (h2 > 0);
}
```

# 17 Giant Step Baby Step

```
// Giant Step - Baby Step for discrete
   log
// find x with a^x = b mod MOD
// Find one soln can be changed to find
   all
// O(root(MOD)*log(MOD)) can be reduced
   with unordered map or array
ll solve(ll a,ll b,ll MOD){
   int n=(int)sqrt(MOD+.0)+1;
   ll an=1,cur;
   forn(i,n) an=(an*a)%MOD;
   cur=an;
   vector<pair<ll,int> > vals;
   forsn(i,1,n+1){
      vals.pb(mp(cur,i));
      cur=(cur*an)%MOD;}
   cur=b;
   sort(all(vals));
   forn(i,n+1){
      auto in=lower_bound(all(vals),
         mp(cur,-1))-vals.begin();
      if(in!=vals.size() &&
         vals[in].fi==cur){
            ll ans=n*(ll)vals[in].se-i;
            if(ans<MOD) return ans;}
      cur=(cur*a)%MOD;}
   return -1;}
```

# 18 Heavy Light Decomposition

```
using node = array<int,CNT> ;
node comb(node a, node b) {
    node c;
    rep(i,0,CNT) c[i] = a[i] ^ b[i];
    return c;
}
// 0-indexed
template<class T> struct
    basic_segment_tree { // comb(ID,b) = b
    const T ID = {0};
    int n; vector<T> seg;
    void init(int _n) {
        n = _n; seg.assign(2*n,ID);
    }
    void pull(int p) {
        seg[p] = comb(seg[2*p], seg[2*p +
            1]);
    }
    void upd(int p, T val) { // update
        val at position p
        seg[p += n] = val;
        for (p /= 2; p; p /= 2) pull(p);
    }
    T query(int l, int r) { // query on
        interval [l, r]
        T ra = ID, rb = ID;
        for (l += n, r += n+1; l < r; l
            /= 2, r /= 2) {
            if (l&1) ra =
                comb(ra,seg[l++]);
            if (r&1) rb =
                comb(seg[--r],rb);
        }
        return comb(ra,rb);
    }
};
// 0-indexed
template<bool VALS_IN_EDGES> struct HLD {
    int N;
    int timer;
    vector<vector<int>> adj;
    vector<int> par, root, depth, sz,
        pos;
    vector<int> rpos; // rpos not used,
        but could be useful
    basic_segment_tree<node> tree; //
        segment tree
    void init(int _N){
        N = _N;
        adj.assign(N,{});
        par.assign(N,-1);
        root.assign(N,-1);
        depth.assign(N,-1);
        sz.assign(N,-1);
        pos.assign(N,-1);
        tree.init(N);
    }
    void ae(int x, int y) {
        adj[x].push_back(y),
            adj[y].push_back(x);
    }
    void dfs_sz(int x) {
        sz[x] = 1;
        for(auto& y : adj[x]) {
            par[y] = x; depth[y] =
                depth[x] + 1;
            adj[y].erase(find(adj[y].begin(),adj
                // remove parent from adj
                list
            dfs_sz(y);
            sz[x] += sz[y];
            if (sz[y] > sz[adj[x][0]])
                swap(y,adj[x][0]); //
                store the heavy child at
                first vertex
        }
    }
    void dfs_hld(int x) {
        pos[x] = timer++;
            rpos.push_back(x);
        for(auto& y : adj[x]) {
            root[y] = (y == adj[x][0] ?
                root[x] : y);
            dfs_hld(y);
        }
    }
    void gen(int R = 0) {
        par[R] = depth[R] = timer = 0;
        dfs_sz(R);
        root[R] = R;
        dfs_hld(R);
    }
    int lca(int x, int y) {
        for (; root[x] != root[y]; y =
            par[root[y]]){
            if (depth[root[x]] >
                depth[root[y]]) swap(x,y);
        }
```

```
    return depth[x] < depth[y] ? x :
        y;
}
int dist(int x, int y) { // # edges
    on path
     return depth[x] + depth[y] - 2 *
         depth[lca(x,y)];
}
void process_path(int x, int y, auto
    op) {
    for (; root[x] != root[y]; y =
        par[root[y]]) {
        if (depth[root[x]] >
            depth[root[y]]) swap(x,y);
        op(pos[root[y]],pos[y]);
    }
    if (depth[x] > depth[y])
        swap(x,y);
    op(pos[x]+VALS_IN_EDGES,pos[y]);
}
void modify_path(int x, int y, node
    v) {
    process_path(x,y,[this,&v](int l,
        int r) {
        assert(l == r);
        tree.upd(l,v);
    });
}
node query_path(int x, int y) {
    node res = {0};
    process_path(x,y,[this,&res](int
        l, int r) {
        res =
            comb(res,tree.query(l,r));
    });
        return res;
    }
     /*
      * this is for range update.
    void modify_subtree(int x, int v) {
        tree.upd(pos[x] + VALS_IN_EDGES,
            pos[x] + sz[x] - 1, v);
    }
    */
};
```

# 19 Hopcraft Karp

```
// Max matching
//1 indexed Hopcroft-Karp Matching in
    O(E sqrtV)
struct Hopcroft_Karp{
 static const int inf = 1e9;
 int n;
 vector<int> matchL, matchR, dist;
 vector<vector<int> > g;
 Hopcroft_Karp(int
     n):n(n),matchL(n+1),matchR(n+1),
     dist(n+1),g(n+1){}
 void addEdge(int u, int v){
  g[u].pb(v);}
 bool bfs(){
  queue<int> q;
  for(int u=1;u<=n;u++){
   if(!matchL[u]){
    dist[u]=0;
    q.push(u);
   }else dist[u]=inf;}
  dist[0]=inf;
  while(!q.empty()){
   int u=q.front();
   q.pop();
   for(auto v:g[u]){
    if(dist[matchR[v]] == inf){
     dist[matchR[v]] = dist[u] + 1;
     q.push(matchR[v]);}}}
  return (dist[0]!=inf);}
 bool dfs(int u){
  if(!u) return true;
  for(auto v:g[u]){
   if(dist[matchR[v]] == dist[u]+1
       &&dfs(matchR[v])){
    matchL[u]=v;
    matchR[v]=u;
    return true;}}
  dist[u]=inf;
  return false;}
 int max_matching(){
  int matching=0;
  while(bfs()){
   for(int u=1;u<=n;u++){
    if(!matchL[u])
     if(dfs(u)) matching++;}}
  return matching;}};
```

# 20 Hungarian Algorithm

```
#define v64 vector <ll>
#define sz(a) (int)a.size()
pair<ll, v64> hungarian(const
    vector<v64> &a) {
```

```
if (a.empty()) return {-1e17, {}};
int n = sz(a) + 1;
int m = sz(a[0]) + 1;
vi u(n), v(m), p(m), ans(n - 1);
rep(i,1,n) {
 p[0] = i;
 int j0 = 0; // add "dummy" worker 0
 vector<ll> dist(m, 1e17), pre(m, -1);
 vector<bool> done(m + 1);
 do { // dijkstra
  done[j0] = true;
  int i0 = p[j0], j1, delta = 1e17;
  rep(j,1,m) if (!done[j]) {
   auto cur = a[i0 - 1][j - 1] - u[i0]
       - v[j];
   if (cur < dist[j]) dist[j] = cur,
       pre[j] = j0;
   if (dist[j] < delta) delta =
       dist[j], j1 = j;
  }
  rep(j,0,m) {
   if (done[j]) u[p[j]] += delta, v[j]
       -= delta;
   else dist[j] -= delta;
  }
  j0 = j1;
 } while (p[j0]);
 while (j0) { // update alternating path
  int j1 = pre[j0];
  p[j0] = p[j1], j0 = j1;
 }
}
rep(j,1,m) if (p[j]) ans[p[j] - 1] = j
    - 1;
return {-v[0], ans}; // min cost
}
```

# 21 Int 128bit

```
std::ostream&
operator<<( std::ostream& dest,
    __int128_t value )
{
    std::ostream::sentry s( dest );
    if ( s ) {
        __uint128_t tmp = value < 0 ?
           -value : value;
        char buffer[ 128 ];
        char* d = std::end( buffer );
        do
        {
            -- d;
            *d = "0123456789"[ tmp % 10 ];
            tmp /= 10;
        } while ( tmp != 0 );
        if ( value < 0 ) {
            -- d;
            *d = '-';
        }
        int len = std::end( buffer ) - d;
        if ( dest.rdbuf()->sputn( d, len
           ) != len ) {
            dest.setstate(
                std::ios_base::badbit );
        }
    }
    return dest;
}
```

# 22 KMP Automaton

```
vector<int> prefix_function(string s){
        int n = (int)s.size();
        vector<int>pi(n);
        for(int i=1;i<n;i++){
                int j = pi[i-1];
                while(j>0&& s[i]!=s[j]) j
                    = pi[j-1];
                if(s[j]==s[i]) j++;
                pi[i] = j;
        }
        return pi;}
void compute_automaton(string s,
    vector<vector<int>>& aut) {
    s += '#';
    int n = s.size();
    vector<int> pi = prefix_function(s);
    aut.assign(n, vector<int>(26));
    for (int i = 0; i < n; i++) {
        for (int c = 0; c < 26; c++) {
            if (i > 0 && 'a' + c != s[i])
                aut[i][c] =
                    aut[pi[i-1]][c];
            else
                aut[i][c] = i + ('a' + c
                    == s[i]);
        }}}
```

# 23 Longest Increasing Subsequence

```
int lis(vector<int> const& a) {
    int n = a.size();
    const int INF = 1e9;
    vector<int> d(n+1, INF);
    d[0] = -INF;
    for (int i = 0; i < n; i++) {
        int j = upper_bound(d.begin(),
            d.end(), a[i]) - d.begin();
        if (d[j-1] < a[i] && a[i] < d[j])
            d[j] = a[i];}
    int ans = 0;
    for (int i = 0; i <= n; i++) {
        if (d[i] < INF)
            ans = i;}
    return ans;}
```

# 24 Lowest Common Ancestor

```
vv32 v;
v32 tin,tout,dist;
vv32 up;
int l;
void dfs(int i,int par,int lvl){
    tin[i]= ++t;
    dist[i]= lvl;
    up[i][0] = par;
    forsn(j,1,l+1) up[i][j]=
        up[up[i][j-1]][j-1];
    forstl(it,v[i]) if(it!=par)
        dfs(it,i,lvl+1);
    tout[i] = ++t;}
bool is_ancetor(int u, int v){
    return tin[u]<=tin[v] &&
        tout[u]>=tout[v];}
int lca(int u, int v){
    if (is_ancetor(u, v)) return u;
    if (is_ancetor(v, u)) return v;
    rforn(i,l) if(!is_ancetor(up[u][i],
        v)) u=up[u][i];
    return up[u][0];}
int get_dis(int u,int v){
    int lcauv=lca(u,v);
    return
        dist[u]+dist[v]-2*dist[lcauv];}
void preprocess(int root){
    tin.resize(n);
    tout.resize(n);
    dist.resize(n);
    t=0;
    l=ceil(log2((double)n));
    up.assign(n,v32(l+1));
    dfs(root,root,0);}
```

# 25 Lucas Theorem

```
//Lucas Theorem: Find (n Choose m) mod p
    for prime p and large n,m. in
    O(log(m*n))
// nCm mod p by lucas theorem for large
    n,m >=0
// p prime, require fact(factorial) &
    invfact(inverse factorial)
v32 fact,invfact;
ll lucas(ll n,ll m,int p){
 ll res=1;
 while(n || m) {
  ll a=n%p,b=m%p;
  if(a<b) return 0;
  res=((res*fact[a]%p)*(invfact[b]%p)%p)*
      (invfact[a-b]%p)%p;
  n/=p; m/=p;}
 return res;}
```

# 26 Manacher

```
Manacher
// Given a string s of length N, finds
    all palindromes as its substrings.
// p[0][i] = half length of longest even
    palindrome around pos i
// p[1][i] = longest odd at i (half
    rounded down i.e len 2*x+1).
//Time: O(N)
void manacher(const string& s){
int n=s.size();
v32 p[2]={v32(n+1),v32(n)};
forn(z,2) for(int i=0,l=0,r=0;i<n;++i){
int t=r-i+!z;
if(i<r) p[z][i]=min(t,p[z][l+t]);
int L=i-p[z][i],R=i+p[z][i]-!z;
while(L>=1 && R+1<n && s[L1]==s[R+1])
    p[z][i]++,L--,R++;
if(R>r) l=L,r=R;}}
```

# 27 Min Cost Max Flow

```
struct MinimumCostMaximumFlow {
  typedef int Index; typedef int Flow;
     typedef int Cost;
  static const Flow InfCapacity = inf;
  struct Edge {
    Index to; Index rev;
    Flow capacity; Cost cost;
  };
  vector<vector<Edge> > g;
  void init(Index n) { g.assign(n,
     vector<Edge>()); }
  void addEdge(Index i, Index j, Flow
     capacity = InfCapacity, Cost cost =
     Cost()) {
    Edge e, f; e.to = j, f.to = i;
       e.capacity = capacity, f.capacity
       = 0; e.cost = cost, f.cost =
       -cost;
    g[i].push_back(e); g[j].push_back(f);
    g[i].back().rev = (Index)g[j].size()
       - 1; g[j].back().rev =
       (Index)g[i].size() - 1;
  }
  void addB(Index i, Index j, Flow
     capacity = InfCapacity, Cost cost =
     Cost()) {
    addEdge(i, j, capacity, cost);
    addEdge(j, i, capacity, cost);
  }
  pair<Cost, Flow>
     minimumCostMaximumFlow(Index s,
     Index t, Flow f = InfCapacity, bool
     useSPFA = false) {
    ll n = g.size();
    vector<Cost> dist(n); vector<Index>
       prev(n); vector<Index>
       prevEdge(n);
    pair<Cost, Flow> total =
       make_pair(0, 0);
    vector<Cost> potential(n);
    while(f > 0) {
      fill(dist.begin(), dist.end(), INF);
      if(useSPFA || total.second == 0) {
        deque<Index> q;
        q.push_back(s); dist[s] = 0;
           vector<bool> inqueue(n);
        while(!q.empty()) {
          Index i = q.front();
             q.pop_front(); inqueue[i] =
             false;
          for(Index ei = 0; ei <
             (Index)g[i].size(); ei ++) {
            const Edge &e = g[i][ei];
               Index j = e.to; Cost d =
               dist[i] + e.cost;
            if(e.capacity > 0 && d <
               dist[j]) {
              if(!inqueue[j]) {
                inqueue[j] = true;
                q.push_back(j);
              }
              dist[j] = d; prev[j] = i;
                 prevEdge[j] = ei;
            }
          }
        }
      } else {
        vector<bool> vis(n);
        priority_queue<pair<Cost, Index>
           > q;
        q.push(make_pair(-0, s)); dist[s]
           = 0;
        while(!q.empty()) {
          Index i = q.top().second;
             q.pop();
          if(vis[i]) continue;
          vis[i] = true;
          for(Index ei = 0; ei <
             (Index)g[i].size(); ei ++) {
            const Edge &e = g[i][ei];
            if(e.capacity <= 0) continue;
            Index j = e.to; Cost d =
               dist[i] + e.cost +
               potential[i] -
               potential[j];
            if(dist[j] > d) {
              dist[j] = d; prev[j] = i;
                 prevEdge[j] = ei;
              q.push(make_pair(-d, j));
            }
          }
        }
      }
      if(dist[t] == INF) break;
      if(!useSPFA) for(Index i = 0; i <
         n; i ++) potential[i] +=
         dist[i];

      Flow d = f; Cost distt = 0;
      for(Index v = t; v != s; ) {
        Index u = prev[v]; const Edge &e
           = g[u][prevEdge[v]];
```

```
      d = min(d, e.capacity); distt +=
          e.cost; v = u;
    }
    f -= d; total.first += d * distt;
        total.second += d;
    for(Index v = t; v != s; v =
        prev[v]) {
      Edge &e = g[prev[v]][prevEdge[v]];
      e.capacity -= d;
          g[e.to][e.rev].capacity += d;
    }
  }
  return total;
 }
};
```

## 28 Nearest Pair of Points

```
vector<pt> t;

void rec(int l, int r) {
    if (r - l <= 3) {
        for (int i = l; i < r; ++i) {
            for (int j = i + 1; j < r;
                ++j) {
                upd_ans(a[i], a[j]);}}
        sort(a.begin() + l, a.begin() +
            r, cmp_y());
        return;}

    int m = (l + r) >> 1;
    int midx = a[m].x;
    rec(l, m);
    rec(m, r);
    merge(a.begin() + l, a.begin() + m,
        a.begin() + m, a.begin() + r,
        t.begin(), cmp_y());
    copy(t.begin(), t.begin() + r - l,
        a.begin() + l);

    int tsz = 0;
    for (int i = l; i < r; ++i) {
        if (abs(a[i].x - midx) < mindist)
            {
            for (int j = tsz - 1; j >= 0
                && a[i].y - t[j].y <
                mindist; --j)
                upd_ans(a[i], t[j]);
            t[tsz++] = a[i];}}}
// In main, call as:
t.resize(n);
sort(a.begin(), a.end(), cmp_x());
mindist = 1E20;
rec(0, n);
```

## 29 Number Theoretic Transform

```
const int mod=998244353;
// 998244353=1+7*17*2^23 : g=3
// 1004535809=1+479*2^21 : g=3
// 469762049=1+7*2^26 : g=3
// 7340033=1+7*2^20 : g=3
// For below change mult as overflow:
 // 10000093151233=1+3^3*5519*2^26 : g=5
 // 1000000523862017=1+10853*1373*2^26 :
     g=3
 //
     1000000000949747713=1+2^29*3*73*8505229
     : g=2
// For rest find primitive root using
    Shoup's generator algorithm
// root_pw: power of 2 >= maxn,
    Mod-1=k*root_pw => w = primitive^k
template<long long Mod,long long
    root_pw,long long primitive>
struct NTT{
 inline long long powm(long long x,long
     long pw){
  x%=Mod;
  if(abs(pw)>Mod-1) pw%=(Mod-1);
  if(pw<0) pw+=Mod-1;
  ll res=1;
  while(pw){
   if(pw&1LL) res=(res*x)%Mod;
   pw>>=1;
   x=(x*x)%Mod;}
  return res;}
 inline ll inv(ll x){
     return powm(x,Mod-2); }
 ll root,root_1;
 NTT(){
  root=powm(primitive,(Mod-1)/root_pw);
  root_1=inv(root);}
 void ntt(vector<long long> &a,bool
     invert){
  int n=a.size();
  for(long long i=1,j=0;i<n;i++){
   long long bit=n>>1;
   for(;j&bit;bit>>=1) j^=bit;
```

```
  j^=bit;
  if(i<j) swap(a[i],a[j]);}
 for(long long len=2;len<=n;len<<=1){
  long long wlen= invert ? root_1:root;
  for(long long i=len;i<root_pw;i<<=1)
      wlen=wlen*wlen%Mod;
  for(long long i=0;i<n;i+=len){
   long long w=1;
   for(long long j=0;j<len/2;j++){
    long long
        u=a[i+j],v=a[i+j+len/2]*w%Mod;
    a[i+j]= u+v<Mod ? u+v:u+v-Mod;
    a[i+j+len/2]= u-v>=0 ? u-v:u-v+Mod;
    w=w*wlen%Mod;}}}
 if(invert){
  ll n_1=inv(n);
  for(long long &x: a) x=x*n_1%Mod;}}
vector<long long> multiply(vector<long
    long> const& a,vector<ll> const& b){
 vector<long long>
     fa(a.begin(),a.end()),fb(b.begin(),b.end());
 int n=1;
 while(n<a.size()+b.size()) n<<=1;
 point(fa,1,n);
 point(fb,1,n);
 for(int i=0;i<n;++i)
     fa[i]=fa[i]*fb[i]%Mod;
 coef(fa);
 return fa;}
void point(vector<long long> &A,bool
    not_pow=1,int atleast=-1){
 if(not_pow){
  if(atleast==-1){
   atleast=1;
   while(atleast<A.size()) atleast<<=1;}
  A.resize(atleast,0);}
 ntt(A,0);}
void coef(vector<long long> &A,bool
    reduce=1){
 ntt(A,1);
 if(reduce) while(A.size() and
     A.back()==0) A.pop_back(); }
void point_power(vector<long long>
    &A,long long k){
 for(long long &x: A) x=powm(x,k);}
void coef_power(vector<long long>
    &A,int k){
 while(A.size() and A.back()==0)
     A.pop_back();
 int n=1;
 while(n<k*A.size()) n<<=1;
 point(A,1,n);
 point_power(A,k);
 coef(A);}
vector<long long> power(vector<long
    long> a,ll p){
 while(a.size() and a.back()==0)
     a.pop_back();
 vector<long long> res;
 res.pb(1);
 while(p){
  if(p&1) res=multiply(res,a);
  a=multiply(a,a);
  p/=2;}
 return res;}};
NTT<mod,1<<20,3> ntt;
```

## 30 Ordered Set

```
// Set/Map using Leftist Trees
// * To get a map, change {null_type to
    some value}.
#include <bits/extc++.h> /**
    keep-include */
using namespace __gnu_pbds;
template<class T>
using Tree = tree<T, null_type, less<T>,
    rb_tree_tag,
    tree_order_statistics_node_update>;
void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
  assert(t.order_of_key(10) == 1);
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2);} // assuming T < T2 or T >
      T2, merge t2 into t
```

## 31 Persistent Segment Tree

```
struct PST {
#define lc t[cur].l
#define rc t[cur].r
  struct node {
    int l = 0, r = 0, val = 0;
  } t[20 * N];
  int T = 0;
  int build(int b, int e) {
    int cur = ++T;
```

```
  if(b == e) return cur;
  int mid = b + e >> 1;
  lc = build(b, mid);
  rc = build(mid + 1, e);
  t[cur].val = t[lc].val + t[rc].val;
  return cur;
}
int upd(int pre, int b, int e, int i,
    int v) {
  int cur = ++T;
  t[cur] = t[pre];
  if(b == e) {
    t[cur].val += v;
    return cur;
  }
  int mid = b + e >> 1;
  if(i <= mid) {
    rc = t[pre].r;
    lc = upd(t[pre].l, b, mid, i, v);
  } else {
    lc = t[pre].l;
    rc = upd(t[pre].r, mid + 1, e, i,
        v);
  }
  t[cur].val = t[lc].val + t[rc].val;
  return cur;
}
int query(int pre, int cur, int b, int
    e, int k) {
  if(b == e) return b;
  int cnt = t[lc].val -
      t[t[pre].l].val;
  int mid = b + e >> 1;
  if(cnt >= k) return query(t[pre].l,
      lc, b, mid, k);
    else return query(t[pre].r, rc, mid
        + 1, e, k - cnt);
  }
} t;
```

# 32 Primitive Root

```
// Primitive root Exist for n=1,2,4,(odd
    prime power),2*(odd prime power)
// O(Ans.log(p).logp + sqrt(phi)) <=
    O((log p)^8 + root(p))
// Change phi when not prime
// Include powm (inverse)
ll phi_cal(ll n){
 ll result=n;
 for(ll i=2;i*i<=n;++i){
  if(n%i==0){
   while(n%i==0) n/=i;
   result-=result/i;}}
 if(n>1) result-=result/n;
 return result;}
ll generator(ll p){
 v64 fact;
 ll phi=p-1; // Call phi_cal if not prime
 ll n=phi;
 for(ll i=2;i*i<=n;++i){
  if(n%i==0){
   fact.push_back(i);
   while(n%i==0) n/=i;}}
 if(n>1) fact.push_back(n);
 for(ll res=2;res<=p;++res){
  bool ok=true;
  for(size_t i=0;i<fact.size() && ok;++i)
   ok&=(powm(res,phi/fact[i],p)!=1);
  if(ok) return res;}
 return -1;}
```

# 33 Segtree Lazy

```
void propogate(int node, int l, int r){
    if(l!=r){
        lazy[node*2]+=lazy[node];
        lazy[node*2+1]+=lazy[node];
    }
    st[node]+=lazy[node];
    lazy[node]=0;}
void build(int node, int l, int r){
    if(l==r){
        st[node]=ar[l];
            lazy[node]=0;
        return;}
    int mid=(l+r)/2;
    build(node*2, l, mid);
    build(node*2+1, mid+1, r);
    st[node]=min(st[node*2],
        st[node*2+1]);
    lazy[node]=0;
    return;}
void update(int node, int l, int r, int
    x, int y, int val){
    if(lazy[node]!=0) propogate(node, l,
        r);
    if(y<x||x>r||y<l) return;
    if(l>=x&&r<=y){
        st[node]+=val;
        if(l!=r){
```

```
            lazy[node*2]+=val;
            lazy[node*2+1]+=val;
        }
        return;}
    int mid=(l+r)/2;
    update(node*2, l, mid, x, y, val);
    update(node*2+1, mid+1, r, x, y,
        val);
    st[node]=min(st[node*2],
        st[node*2+1]);
    return;}
int query(int node, int l, int r, int x,
    int y){
    if(lazy[node]!=0) propogate(node, l,
        r);
    if(y<x||y<l||x>r) return INF;
    if(l>=x&&r<=y) return st[node];
    int mid=(l+r)/2;
    return min(query(node*2, l, mid, x,
        y), query(node*2+1, mid+1, r, x,
        y));
}
```

# 34 Suffix Array

```
vector<int> sort_cyclic_shifts(string
    const& s) {
    int n = s.size();
    const int alphabet = 256;
    vector<int> p(n), c(n),
        cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        rep(i,0,classes)
        cnt[i]=0;
        //fill(cnt.begin(), cnt.begin() +
            classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur =
                {c[p[i]], c[(p[i] + (1 <<
                h)) % n]};
            pair<int, int> prev =
                {c[p[i-1]], c[(p[i-1] + (1
                << h)) % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}
vector<int>
    suffix_array_construction(string s) {
    s += "$";
    vector<int> sorted_shifts =
        sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}
vector<int> lcp_construction(string
    const& s, vector<int> const& p) {
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++)
        rank[p[i]] = i;

    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
```

```
        while (i + k < n && j + k < n &&
            s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if (k)
            k--;
    }
    return lcp;
}
```

# 35 Template

```
#pragma GCC optimize ("-O2")
#pragma GCC optimize("Ofast")
// ~ #pragma GCC
    target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
// ~ #pragma GCC optimize("unroll-loops")
#include <bits/stdc++.h>
using namespace std;
#define fastio
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define forstl(i,v) for(auto &i: v)
#define forn(i,e) for(int i=0;i<e;++i)
#define forsn(i,s,e) for(int i=s;i<e;++i)
#define rforn(i,s) for(int i=s;i>=0;--i)
#define rforsn(i,s,e) for(int
    i=s;i>=e;--i)
#define getcurrtime() cerr<<"Time =
    "<<((double)clock()/CLOCKS_PER_SEC)<<endl
#define inputfile freopen("input.txt",
    "r", stdin)
#define outputfile freopen("output.txt",
    "w", stdout)
typedef pair<ll,ll> p64;
typedef pair<int,p32> p96;
typedef vector<ll> v64;
typedef vector<v64> vv64;
mt19937
    rng(chrono::steady_clock::now().time_since_epoch().count());
```

# 36 XOR-Basis

```
int basis[d]; // basis[i] keeps the mask
    of the vector whose f value is i
int sz; // Current size of the basis
void insertVector(int mask) {
 // 0 se d ke jagah d-1 se 0 kar lena
    agar smallest ka kaam ho
 for (int i = 0; i < d; i++) {
  if ((mask & 1 << i) == 0) continue; //
      continue if i != f(mask)

  if (!basis[i]) { // If there is no
      basis vector with the i'th bit set,
      then insert this vector into the
      basis
   basis[i] = mask;
   ++sz;

   return;
  }

  mask ^= basis[i]; // Otherwise
      subtract the basis vector from this
      vector
 }}
```

# 37 Z Algorithm

```
// Z Algorithm
// Z[i] is the length of the longest
    substring starting from S[i]
// which is also a prefix of S
// O(n)
void z_func(v32 &s,v32 &z){
        int L=0,R=0;
        int sz=s.size();
        z.assign(sz,0);
        forsn(i,1,sz){
                if(i>R){
                        L=R=i;
                        while(R<sz &&
                            s[R-L]==s[R])
                            R++;
                        z[i]=R-L; R--;
                }else{
                        int k=i-L;
                        if(z[k]<R-i+1)
                            z[i]=z[k];
                        else{
                                L=i;
                                while(R<sz
                                    &&
                                    s[R-L]==s[R])
                                    R++;
                                z[i]=R-L;
                                    R--;
                                    }}}}
```

# 38 Z Ideas

Gray codes Applications:

1. Gray code of n bits forms a Hamiltonian cycle on a hypercube, where each bit corresponds to one dimension.
2. Gray code can be used to solve the Towers of Hanoi problem. Let n denote number of disks. Start with Gray code of length n which consists of all zeroes (G(0)) and move between consecutive Gray codes (from G(i) to G(i+1)).

Let i-th bit of current Gray code represent n-th disk (the least significant bit corresponds to the smallest disk and the most significant bit to the biggest disk).Since exactly one bit changes on each step, we can treat changing i-th bit as moving i-th disk. Notice that there is exactly one move option for each disk (except the smallest one) on each step (except start and finish positions).

There are always two move options for the smallest disk but there is a strategy which will always lead to answer:

if n is odd then sequence of the smallest disk moves looks like ftrftr ... where f is the initial rod, t is the terminal rod and r is the remaining rod),

and if n is even: frtfrt ....

```
int gray (int n) {return n ^ (n >> 1);}
int rev_g (int g) {
  int n = 0;
  for (; g; g >>= 1) n ^= g;
  return n;}
```

Enumerating all submasks of a bitmask:

```
for (int s=m; ; s=(s-1)&m) {
 ... you can use s ...
 if (s==0) break;}
```

Divide and Conquer DP:

Some dynamic programming problems have a recurrence of this form:

dp(i,j)= minkj {dp( i1 ,k)+C(k,j)} where C(k,j) is some cost function.

Say 1<=i<=n and 1<=j<=m, and evaluating C takes O(1) time.

Straightforward evaluation of the above recurrence is O(nm2).

There are nm states, and m transitions for each state.

Let opt(i,j) be the value of k that minimizes the above expression.

If opt(i,j) opt (i,j+1) for all i,j, then we can apply

divide-and-conquer DP. This known as the monotonicity condition.

The optimal "splitting point" for a fixed i increases as j increases.

```
int m, n;
vector<long long> dp_before(n),
   dp_cur(n);
long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r]
   (inclusive)
void compute(int l, int r, int optl, int
   optr) {
   if (l > r) return;

   int mid = (l + r) >> 1;
   pair<long long, int> best =
      {LLONG_MAX, -1};

   for (int k = optl; k <= min(mid,
      optr); k++)
      best = min(best, {(k ?
         dp_before[k - 1] : 0) + C(k,
         mid), k});

   dp_cur[mid] = best.first;
   int opt = best.second;
   compute(l, mid - 1, optl, opt);
   compute(mid + 1, r, opt, optr);}

int solve() {
   for (int i = 0; i < n; i++)
      dp_before[i] = C(0, i);
   for (int i = 1; i < m; i++) {
      compute(0, n - 1, 0, n - 1);
      dp_before = dp_cur;
   }
   return dp_before[n - 1];}
```

Knuth Optimization:

```
dp[i][j] = mini < k < j {dp[i][k] + dp [k][j]} + C[i][j]
monotonicity : C[b][c] <= C[a][d]
quadrangle inequality: C[a][c]+C[b][d]
    <= C[a][d]+C[b][c]
```

Lyndon factorization: We can get the minimum cyclic shift.
Factorize the string as s = w1w2w3...wn

```
string min_cyclic_string(string s) {
    s += s;
    int n = s.size();
    int i = 0, ans = 0;
    while (i < n / 2) {
        ans = i;
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;}
        while (i <= k)
            i += j - k;}
    return s.substr(ans, n / 2);}
```

Rank of a matrix:

```
const double EPS = 1E-9;
int compute_rank(vector<vector<double>>
    A) {
    int n = A.size();
    int m = A[0].size();
    int rank = 0;
    vector<bool> row_selected(n, false);
    for (int i = 0; i < m; ++i) {
        int j;
        for (j = 0; j < n; ++j) {
            if (!row_selected[j] &&
                abs(A[j][i]) > EPS)
                break;}
        if (j != n) {
            ++rank;
            row_selected[j] = true;
            for (int p = i + 1; p < m;
                ++p)
                A[j][p] /= A[j][i];
            for (int k = 0; k < n; ++k) {
                if (k != j && abs(A[k][i])
                    > EPS) {
                    for (int p = i + 1; p
                        < m; ++p)
                        A[k][p] -= A[j][p]
                            * A[k][i];}}}}
    return rank;}
```

Determinant of a matrix:

```
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n,
    vector<double> (n));
double det = 1;
for (int i=0; i<n; ++i) {
    int k = i;
    for (int j=i+1; j<n; ++j)
        if (abs (a[j][i]) > abs (a[k][i]))
            k = j;
    if (abs (a[k][i]) < EPS) {
        det = 0;
        break;}
    swap (a[i], a[k]);
    if (i != k)
        det = -det;
    det *= a[i][i];
    for (int j=i+1; j<n; ++j)
        a[i][j] /= a[i][i];
    for (int j=0; j<n; ++j)
        if (j != i && abs (a[j][i]) > EPS)
            for (int k=i+1; k<n; ++k)
                a[j][k] -= a[i][k] *
                    a[j][i];}
cout << det;
```

Generating all k-subsets:

```
vector<int> ans;
void gen(int n, int k, int idx, bool
    rev) {
    if (k > n || k < 0) return;
    if (!n) {
        for (int i = 0; i < idx; ++i) {
            if (ans[i]) cout << i + 1;}
        cout << "\n";
        return;}
    ans[idx] = rev;
    gen(n - 1, k - rev, idx + 1, false);
    ans[idx] = !rev;
    gen(n - 1, k - !rev, idx + 1, true);}
void all_combinations(int n, int k) {
    ans.resize(n);gen(n, k, 0, false);}
```

Picks theorem:

Given a certain lattice polygon with non-zero area. We denote its area by S, the number of points with integer coordinates lying strictly inside the polygon by I and the number of points lying on polygon sides by B. Then, the Pick formula states: S=I + B/2 - 1 In particular, if the values of I and B for a polygon are given, the area can be calculated in O(1) without even knowing the vertices.

Strongly Connected component and Condensation Graph:

```
vector < vector<int> > g, gr;
vector<bool> used;
vector<int> order, component;
void dfs1 (int v) {
    used[v] = true;
    for (size_t i=0; i<g[v].size();
        ++i)
        if (!used[ g[v][i] ]) dfs1
            (g[v][i]);
    order.push_back (v);}
void dfs2 (int v) {
    used[v] = true;
    component.push_back (v);
    for (size_t i=0; i<gr[v].size();
        ++i)
        if (!used[ gr[v][i] ]) dfs2
            (gr[v][i]);}
int main() {
    int n;
    ... reading n ...
    for (;;) {
        int a, b;
        ... reading next edge (a,b)
            ...
        g[a].push_back (b);
        gr[b].push_back (a);
    }
    used.assign (n, false);
    for (int i=0; i<n; ++i)
        if (!used[i]) dfs1 (i);
    used.assign (n, false);
    for (int i=0; i<n; ++i) {
        int v = order[n-1-i];
        if (!used[v]) { dfs2 (v);
            ... printing next
                component ...
            component.clear();
        }}}
```

FFT Matrices:
XOR FFT: 1 1 / 1 -1, AND FFT: 0 1/ 1 1,
OR FFT: 1 1/ 1 0

Harmonic lemma:

```
for (int i = 1, la; i <= n; i = la + 1) {
            la = n / (n / i);
            v.pb(mp(n/i,la-i+1));}
    //n / x yields the same value for
        i <= x <= la.
```

Mobius inversion theory:

if f and g are multiplicative, then their dirichlet convolution, i.e sum_{d|x} f(d)g(x/d) is also multiplicative. eg. choose g = 1
Properties:
1. If g(n) = sum_{d|n}f(d), then f(n) = sum_{d|x}g(d)u(n/d).
2. sum_{d|n}u(d) = [n==1]

Standard question: Number of co-prime integers in range 1,n
Answer: f(n) = sum_{d = 1 to n} u(d)floor(n/d)^2

Euler totient: phi(totient fn) = u*n (dirichlet convolution)

a Nim position (n1, ,nk) is a second player win in misere Nim if and only if some ni>1 and n1 xor .. xor nk=0, or all ni<=1 and n1 xor .. xor nk=1.

Fibonacci Identities:

1. F_{n-1}F_{n+1} - F_{n}^2 = (-1)^n
2. F_{n+k} = F_{k}F_{n+1} + F_{k-1}F_{n}
3. Fn | Fm <=> n | m
4. GCD(F_m,F_n) = F_{gcd(m,n)}
5. F_{2k} = F_{k}(2F_{k+1}-F_{k}). F_{2k+1} = F^2_{k+1} +F^2_{k}
6. n>=phi(m) => x^n = x^(phi(m)+n%phi(m)) mod m

Counting labeled graphs:
The total number of labelled graphs is G_n = 2^{n(n-1)/2}

```
Number of connected labelled graphs is
    C_n = G_n - 1/n*(sum_{k = 1 to n-1}
    k.(nCk).C_{k}G_{n-k}
Number of labelled graphs with k
    components: D[n][k] = sum_{s = 1 to
    n} ((n-1)C(s-1))C_{s}D[n-s][k-1]

Sum of subsets DP:

F(x) = sum of all A(i) such that x&i = i.

//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
        dp[mask][-1] = A[mask]; //handle
            base case separately (leaf
            states)
        for(int i = 0;i < N; ++i){
                if(mask & (1<<i))
                        dp[mask][i] =
                            dp[mask][i-1] +
                            dp[mask^(1<<i)][i-1];
                else
                        dp[mask][i] =
                            dp[mask][i-1];}
        F[mask] = dp[mask][N-1];}

//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i) F[i] =
    A[i];
for(int i = 0;i < N; ++i) for(int mask =
    0; mask < (1<<N); ++mask){
        if(mask & (1<<i)) F[mask] +=
            F[mask^(1<<i)];}
```