Team Notebook

Indian Institute of Technology Bombay

December 7, 2019

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 Centroid Decomposition

```
vector<set<int> > g;
vector<int> par,sub;
int dfs(int u,int p){
 sub[u]=1;
 for(auto &it: g[u]) if(it!=p) sub[u]+=dfs(it,u);
 return sub[u];}
int find_centroid(int u,int p,int n){
 for(auto &it: g[u]){
  if(it!=p && sub[it]>n/2){
   return find_centroid(it,u,n);}}
 return u;}
void decompose(int u,int p=-1){
 int n=dfs(u,p);
 int centroid=find_centroid(u,p,n);
 if(p==-1) p=centroid;
 // Do stuff here for merges
 // Recurse
 par[centroid]=p;
 for(auto &it: g[centroid]){
  g[it].erase(centroid);
  decompose(it,centroid);}
 g[centroid].clear();}
void reset(int n){
 par.resize(n);
 sub.resize(n);
 g.assign(n,set<int>());}
```

4 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);}};
// const static int LX = -(1e9), RX = 1e9;
// struct Dynamic_Hull { /* Max hull */
//   struct Line{
//     ll m, c; // slope, intercept
//     Line(ll mm=0, ll cc=-1e18) { m = mm; c = cc;
     }
//     ll operator[](const int&x){ return m*x+c; }
//   };
//   struct node {
//     node *lt,*rt; Line Ln;
//     node(const Line &l){lt=rt=nullptr; Ln=l;}
//   };
//   node *root=nullptr;
//   void add(Line l,node*&it,int lx=LX,int rx=RX){
//     if(it==nullptr)it=new node(l);
//     if(it->Ln[lx]>=l[lx] and it->Ln[rx]>=l[rx])
    return;
//     if(it->Ln[lx]<=l[lx] and it->Ln[rx]<=l[rx])
    {it->Ln=l; return;}
//     int mid = (lx+rx)>>1;
//     if(it->Ln[lx] < l[lx]) swap(it->Ln,l);
//     if(it->Ln[mid] >= l[mid]) add(l,it->rt,mid
    +1,rx);
//     else { swap(it->Ln,l); add(l,it->lt,lx,mid);
      }
//   }
//   void add(const ll &m,const ll &c) { add(Line(m
    ,c),root); }
//   ll get(int &x,node*&it,int lx=LX,int rx=RX){
//     if(it==NULL) return -1e18; // Max hull
//     ll ret = it->Ln[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;
//   }
//   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(dq[1]))
      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);}};
```

5 Dynamic Connectivity

```
int u[LIM],v[LIM],e[LIM],q[LIM];
map<p32,int> ids;
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;
void solve(int l,int r,vp32 &ed){
 if(l>r) return;
 // dbg(ed,l,r,d.sz);
 int mid=(l+r)>>1;
 vp32 low;
 int tm=d.moment();
 forstl(it,ed){
  if(it.se<l or it.fi>r) continue;
  else if(it.fi<=l and it.se>=r) d.join(u[it.fi],v[
      it.fi]);
  else low.pb(it);
 }
 if(l==r){
  if(q[l]) cout<<(d.con(u[l],v[l])? "YES":"NO")<<ln
      ;
 }else{
  solve(l,mid,low);
  solve(mid+1,r,low);
 }
 d.revert(tm);
}
signed main(){
 fastio;
 cin>>n>>k;
 d.reset(n);
 string t;
 forn(i,k){
  cin>>t;
  cin>>x>>y; --x,--y;
  if(x>y) swap(x,y);
  u[i]=x,v[i]=y;
  if(t[0]=='c'){
    q[i]=1;
  }else{
   if(t[0]=='a'){
    ids[mp(x,y)]=i;
    e[i]=k-1;
   }else{
    e[ids[mp(x,y)]]=i;
    e[i]=-1;
   }
  }
 }
 vp32 ed;
 forn(i,k) if(!q[i] && e[i]!=-1) ed.pb({i,e[i]});
 solve(0,k-1,ed);
 return 0;
}
```

6 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.
```

7 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;}
```

8 Fast Fourier Transform

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) { //invert =
     1 for inverse FFT
    int 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]);}

    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 :
            1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }}}

    if (invert) {
        for (cd & x : a)
            x /= n;}}

vector<int> multiply(vector<int> const& a, vector<
    int> const& b) {
//function to multiply two polynomials
    vector<cd> fa(a.begin(), a.end()), fb(b.begin()
        , b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);
    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);
    vector<int> result(n);
```

```
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;}
```

9 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;}};
```

10 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;}};
```

11 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;}
```

12 General Weighted Matching

```
struct MaxMatchingEdmonds{
    // Assume General Unweighted Directed Graph
    // O(V^3) edmonds for maximum matching
    vv32 g;
    v32 match,p,base;
    vector<bool> blossom;
```

```
int n;
int lca(int a,int b){
    vector<bool> used(match.size(),0);
    while(1){
      a=base[a];
      used[a]=1;
      if(match[a]==-1) break;
      a=p[match[a]];}
    while(1){
      b=base[b];
      if(used[b]) return b;
      b=p[match[b]];}}
void markPath(int v,int b,int children) {
    for(;base[v]!=b;v=p[match[v]]){
      blossom[base[v]]=blossom[base[match[v
          ]]]=1;
      p[v]=children;
      children=match[v];}}
int findPath(int root) {
    vector<bool> used(n,0);
    p.assign(n,-1);
    base.assign(n,0);
    for(int i=0;i<n;++i) base[i] = i;
    used[root]=1;
    int qh=0;
    int qt=0;
    v32 q(n,0);
    q[qt++]=root;
    while(qh<qt){
      int v=q[qh++];
      for(int &to:g[v]){
        if(base[v]==base[to] || match[v]==to)
            continue;
        if(to==root || match[to]!=-1 && p[match[
            to]]!=-1){
          int curbase=lca(v,to);
          blossom.assign(n,0);
          markPath(v,curbase,to);
          markPath(to,curbase,v);
          for(int i=0;i<n;++i)
            if(blossom[base[i]]){
              base[i]=curbase;
              if(!used[i]){
                used[i]=1;
                q[qt++]=i;}}
          }else if(p[to]==-1){
            p[to]=v;
            if(match[to]==-1) return to;
            to=match[to];
            used[to]=1;
            q[qt++]=to;}}}
      return -1;}
  int maxMatching(vv32 &graph){
      n=graph.size();
      g=graph;
      match.assign(n,-1);
      p.assign(n,0);
      for(int i=0;i<n;++i){
        if(match[i]==-1){
          int v=findPath(i);
          while(v!=-1){
            int pv=p[v];
            int ppv=match[pv];
            match[v]=pv;
            match[pv]=v;
            v=ppv;}}}
      int matches=0;
      for(int i=0;i<n;++i) if(match[i]!=-1) ++
          matches;
      return matches/2;}};
```

13 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)/(pv[i
      ].x-pv[i+1].x);
   if(y>0) j++;}
  return(j%2);}}
return 1;}
```

14 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;}
```

15 Hashtable

```
struct hashtable{
 v64 hash1,hash2,inv1,inv2;
 ll MOD1=MOD,MOD2=MOD+2;
 ll pr1=31,pr2=37;
 void create(string &p){
  int len=p.size();
  hash1.resize(len);hash2.resize(len);
  inv1.resize(len);inv2.resize(len);
  ll p1=1,p2=1;
  int i=0;
  while(p[i]){
   hash1[i]= (i==0)? 0:hash1[i-1];
   hash2[i]= (i==0)? 0:hash2[i-1];
   hash1[i]= (hash1[i]+p[i]*p1)%MOD1;
   hash2[i]= (hash2[i]+p[i]*p2)%MOD2;
   p1=p1*pr1%MOD1;
   p2=p2*pr2%MOD2;
   i++;}
  ll iv1=inv(pr1,MOD1),iv2=inv(pr2,MOD2);
  inv1[0]=1,inv2[0]=1;
  forsn(i,1,len){
   inv1[i]=inv1[i-1]*iv1%MOD1;
   inv2[i]=inv2[i-1]*iv2%MOD2;}}
 p64 gethash(int l,int r){
  ll ans1=hash1[r-1];
  if(l!=0) ans1+=MOD1-hash1[l-1];
  ll ans2=hash2[r-1];
  if(l!=0) ans2+=MOD2-hash2[l-1];
  ans1=ans1*inv1[l]%MOD1;
  ans2=ans2*inv2[l]%MOD2;
  return mp(ans1,ans2);}};
```

16 Heavy Light Decomposition

```
struct SegTree{
 v32 T,lazy;
 int N,MX;
 void clear(int n,int mx){
  N=n,MX=mx;
  T.assign(4*N,0);
  lazy.assign(4*N,0);}
 void build(int a[],int v,int tl,int tr){
  if(tl==tr){
   T[v]=a[tl];}else{
   int tm=(tl+tr)>>1,lf=v<<1,rt=lf^1;;
   build(a,lf,tl,tm);
   build(a,rt,tm+1,tr);
   T[v]=min(T[lf],T[rt]);}}
 void push(int v){
  int lf=v<<1,rt=lf^1;
  T[lf]=(T[lf]+lazy[v]);
  lazy[lf]=(lazy[lf]+lazy[v]);
  T[rt]=(T[rt]+lazy[v]);
  lazy[rt]=(lazy[rt]+lazy[v]);
  lazy[v]=0;}
 void update(int v,int tl,int tr,int l,int r,int
     val){
  if(l>r or tl>r or tr<l) return;
  if(l<=tl && tr<=r){
   T[v]=T[v]+val;
   lazy[v]=(lazy[v]+val);}else{
   if(tl==tr) return;
   push(v);
   int tm=(tl+tr)>>1,lf=v<<1,rt=lf^1;;
   update(lf,tl,tm,l,r,val);
   update(rt,tm+1,tr,l,r,val);
   T[v]=max(T[lf],T[rt]);}}
 int query(int v,int tl,int tr,int l,int r){
  if(l>r) return MX;
  if(l<=tl && tr<=r) return T[v];
  push(v);
  int tm=(tl+tr)>>1,lf=v<<1,rt=lf^1;
  return max(query(lf,tl,tm,l,min(r,tm)),query(rt,
      tm+1,tr,max(l,tm+1),r));}
 int q(int l,int r){
  return query(1,0,N-1,l,r);}
 void u(int l,int r,int val){
  update(1,0,N-1,l,r,val);}
```

```
} st;
struct hld{
 int n,t;
 v32 sz,in,out,root,par,depth;
 vv32 g;
 SegTree tree;
 void dfs_sz(int v=0,int p=0){
  sz[v]=1;
  for(auto &u: g[v]){
   if(u==p) continue;
   dfs_sz(u,v);
   sz[v]+=sz[u];
   if(sz[u]>sz[g[v][0]]) swap(u, g[v][0]);}}
 void dfs_hld(int v=0,int p=0){
    in[v]=t++;
    par[v]=p;
    depth[v]=depth[p]+1;
    for(auto u: g[v]){
     if(u==p) continue;
       root[u]= (u==g[v][0] ? root[v]:u);
       dfs_hld(u,v);}
    out[v]=t;}
 void pre(vv32 &v){
  g=v;n=v.size();t=0;
  sz.assign(n,0);in.assign(n,0);out.assign(n,0);
  root.assign(n,0);par.assign(n,0);depth.assign(n
     ,0);
  depth[0]=-1;
  dfs_sz();dfs_hld();
  tree.clear(n,-MOD);}
 template <class BinaryOperation>
 void processPath(int u,int v,BinaryOperation op){
  for(;root[u]!=root[v];v=par[root[v]]){
   if(depth[root[u]] > depth[root[v]]) swap(u,v);
   op(in[root[v]],in[v]); }
  if(depth[u]>depth[v]) swap(u,v);
  op(in[u],in[v]);}
 void modifyPath(int u,int v,const int &value){
    processPath(u,v,[this,&value](int l,int r){
        tree.u(l,r,value);});} // [l,r]
   void modifySubtree(int u,const int &value){
    tree.u(in[u],out[u]-1,value);}
   int queryPath(int u,int v){
     int res=-MOD;
     auto add=[](int &a,const int &b){a=max(a,b);};
      processPath(u,v,[this,&res,&add](int l,int r){
          add(res,tree.q(l,r));});
      return res;}
    int querySubtree(int u){
     return tree.q(in[u],out[u]-1);}
};
```

17 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;}};
```

18 Hungarian Algorithm

```
struct Hungarian{
  //Important: cost matrix a[1..n][1..m]>=0,n<=m (
      works with negative costs)
  // O(V^3) Use p to find matching of 1..m
  vv64 a;
  v64 u,v;
  v32 p,way;
  int n,m;
  Hungarian(int n,int m): n(n),m(m),u(n+1,0),v(m
      +1,0),p(m+1,0),way(m+1,0),a(n+1,v64(m+1,0)){}
  void addEdge(int u,int v,ll val){
    a[u][v]=val;}
  ll solveAssignmentProblem(){
    for(int i=1;i<=n;++i){
      p[0]=i;
      int j0=0;
      v64 minv(m+1,2e17+10);
      vector<bool> used(m+1,0);
      do{
        used[j0] = true;
        int i0=p[j0];
        ll delta=2e17+10;
        int j1=0;
        for(int j=1;j<=m;++j){
          if(!used[j]){
            ll cur=a[i0][j]-u[i0]-v[j];
            if(cur<minv[j]){
              minv[j]=cur;
```

```
            way[j]=j0;}
          if(minv[j]<delta){
            delta=minv[j];
            j1=j;}}}
      for(int j=0;j<=m;++j){
        if(used[j]){
          u[p[j]]+=delta;
          v[j]-=delta;
        }else minv[j]-=delta;}
      j0=j1;
    }while(p[j0]!=0);
    do{
      int j1=way[j0];
      p[j0]=p[j1];
      j0=j1;
    }while(j0!=0);}
  return -v[0];}};
```

19 Interval Handling

```
map<int, int> active;
int ans = 0, n;

void init(){
 active[-1] = -1;
 active[2e9] = 2e9;
 active[1] = n;
 ans = n;}

void add(int L, int R){ //Always remove [L, R]
    before adding
 active[L]=R;
 ans+=R-L+1;}

void remove(int L, int R){
 int removed=0;
 auto it = active.lower_bound(L);
 it--;
 if(it->second>=L){
  active[L] = it->second;
  it->second = L-1;}
 it++;
 while(it->first <= R){
  if(it->second > R){
   removed+=R + 1 - it->first;
   active[R+1] = it->second;
  }
  else
   removed+= it->second - it->first + 1;
  auto it2=it;
  it++;
  active.erase(it2);}
 ans-=removed;}
```

20 Linear Sieve

```
int mu[LIM],is_com[LIM];
v32 pr;
void sieve(){
    mu[1]=1;
    forsn(i,2,LIM){
        if(!is_com[i]) pr.pb(i),mu[i]=-1;
        forstl(it,pr){
            if(it*i>=LIM) break;
            is_com[i*it]=1;
            if(i%it==0){
                mu[i*it]=0;
                break;
            }else{
                mu[i*it]=mu[i]*mu[it];}}}}
```

21 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;}
```

22 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);}
```

23 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;}
```

24 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;}}
```

25 Merge Sort Tree

```
// Merge sort Tree
const int MAXN=1e5+5;
v32 T[4*MAXN]; // nlogn memory
void build(int a[],int v,int tl,int tr){
 if(tl==tr){
  T[v]=v32(1,a[tl]);
 }else{
  int tm=(tl+tr)>>1;
  build(a,v<<1,tl,tm);
  build(a,(v<<1)^1,tm+1,tr);
  merge(all(T[v<<1]),all(T[(v<<1)^1]),back_inserter
      (T[v]));
  // built in combine in sorted order (2pointer)}}
// number of numbers <=x in [l,r]
int query(int v,int tl,int tr,int l,int r,int x){
 if(l>r) return 0;
 if(l<=tl && tr<=r){
  return upper_bound(all(T[v]),x)-T[v].begin();}
 int tm=(tl+tr)>>1;
 return query(v<<1,tl,tm,l,min(r,tm),x)+query((v
     <<1)^1,tm+1,tr,max(l,tm+1),r,x);}
// Number of distinct integers in [l,r]
int b[MAXN];
void convert(int a[],int n){ // b store next occ
    index
 m32 m; // Can be replaced by vv32 in small numbers
 rforn(i,n-1){
  auto it=m.find(a[i]);
  if(it==m.end()) b[i]=MOD;
  else b[i]=it->se;
  m[a[i]]=i;}
 build(b,1,0,n-1);}
inline int q(int l,int r){ // no. of val in [l,r]
    with nxt ind > r
 return (r-l+1)-query(1,0,n-1,l,r,r);}
```

26 Miller Rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;}
    return result;}
bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;}
    return true;};
bool MillerRabin(u64 n) { // returns true if n is
    prime, else returns false.
    if (n < 2)
        return false;
    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;}
    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23,
        29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;}
    return true;}
```

27 Min Cost Max Flow

```
// Mincost Maxflow : O(E^2)
// [Hell-Johnson MinCostMaxFlow using Dijstra with
    potential & Fibonnaci Heap]
// Negative cost cycles are not supported.
```

```
struct MCMF{
  struct Edge{
    int u,v,rind;
    FLOW cap,flow;
    COST cost;};
  int N;
  vector<COST> pot,dist;
  vector<vector<Edge> > v;
  vector<pair<int,int> > par;
  MCMF(int n): N(n),dist(n),v(n),par(n){}
  void AddEdge(int to,int from,int cap,int cost){
    if(to==from){
      assert(cost>=0);
      return;}
    int i1=v[to].size(),i2=v[from].size();
    v[to].push_back({to,from,i2,cap,0,cost});
    v[from].push_back({from,to,i1,0,0,-cost});}
  void setpi(int s){
    pot.assign(N,CINF);
    pot[s]=0;
    int ch=1,ite=N;
    COST cur,nw;
    while(ch-- && ite--){
      for(int i=0;i<N;++i){
        if(pot[i]!=CINF){
          cur=pot[i];
          for(auto &e: v[i]){
            if(e.cap>0 && (nw=cur+e.cost)<pot[e.v]){
              pot[e.v]=nw; ch=1;}}}}}
    assert(ite>=0);} // Else negative cycle
  bool path(int s,int t){
    fill(dist.begin(),dist.end(),CINF);
    dist[s]=0;
    __gnu_pbds::priority_queue<pair<COST,int> > pq;
    vector<decltype(pq)::point_iterator> its(N);
    pq.push({0,s});
    COST curr,val;
    int node,cnt;
    bool ok=0;
    while(!pq.empty()){
      tie(curr,node)=pq.top();
      pq.pop();
      curr=-curr;
      if(curr!=dist[node]) continue;
      curr+=pot[node];
      if(node==t) ok=1;
      cnt=0;
      for(auto &e: v[node]){
        if(e.cap>e.flow && (val=curr+e.cost-pot[e.v
            ])<dist[e.v]){
          dist[e.v]=val;
          par[e.v]=make_pair(node,cnt);
          if(its[e.v]==pq.end()) its[e.v]=pq.push
              ({-val,e.v});
            else pq.modify(its[e.v],{-val,e.v});}
        ++cnt;}}
    for(int i=0;i<N;++i){
      pot[i]=min(pot[i]+dist[i],FINF);}
    return ok;}
  pair<FLOW,COST> SolveMCMF(int s,int t,FLOW need=
      FINF,bool neg=0){
    FLOW tot=0,cflow=0; COST tcost=0;
    if(s==t) return {tot,tcost};
    if(!neg) pot.assign(N,0);
    else setpi(s);
    int cntr=0;
    while(path(s,t) && need>0){
      cflow=need;
      for(int node=t,u,ind;node!=s;node=u){
        u=par[node].first;
        ind=par[node].second;
        cflow=min(cflow,v[u][ind].cap-v[u][ind].
            flow);}
      tot+=cflow; need-=cflow;
      for(int node=t,u,ind,rind;node!=s;node=u){
        u=par[node].first;
        ind=par[node].second;
        rind=v[u][ind].rind;
        v[u][ind].flow+=cflow;
        v[node][rind].flow-=cflow;}}
    return {tot,tcost};}};
```

28 Mo's Algorithm

```
const int N = 2e5 + 5;
const int Q = 2e5 + 5;
const int M = 1e6 + 5;
const int SZ = sqrt(N) + 1;

struct data{
 int l, r, idx;}qr[Q];

int n, q, a[N];
int freq[M];
long long ans[Q];
long long cur = 0;

bool comp(struct data &d1, struct data &d2){
 int b1 = d1.l / SZ;
 int b2 = d2.l / SZ;
 if(b1 != b2)
  return b1 < b2;
 else
  return (b1 & 1) ? d1.r < d2.r : d1.r > d2.r;}

inline void add(int x){
 cur -= 1LL * freq[x] * freq[x] * x;
 freq[x]++;
 cur += 1LL * freq[x] * freq[x] * x;}

inline void remove(int x){
 cur -= 1LL * freq[x] * freq[x] * x;
 freq[x]--;
 cur += 1LL * freq[x] * freq[x] * x;}

void mo(){
 sort(qr + 1, qr + q + 1, comp);
 int l = 1, r = 0;
 cur = 0;
 for(int i=1;i<=q;i++){
  while(l < qr[i].l) remove(a[l++]);
  while(l > qr[i].l) add(a[--l]);
  while(r < qr[i].r) add(a[++r]);
  while(r > qr[i].r) remove(a[r--]);
  ans[qr[i].idx] = cur;}}
```

29 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);
```

30 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;
```

31 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
```

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 Push Relabel

```
//Push-Relabel Algorithm for Flows - Gap Heuristic
    , Complexity: O(V^3)
//To obtain the actual flow values, look at all
    edges with capacity > 0
//Zero capacity edges are residual edges
struct edge{
 int from, to, cap, flow, index;
 edge(int from, int to, int cap, int flow, int
     index):
  from(from), to(to), cap(cap), flow(flow), index(
      index) {}};
struct PushRelabel{
 int n;
 vector<vector<edge> > g;
 vector<long long> excess;
 vector<int> height,active,count;
 queue<int> Q;
 PushRelabel(int n): n(n),g(n),excess(n),height(n),
     active(n),count(2*n) {}
 void addEdge(int from, int to, int cap){
  g[from].push_back(edge(from,to,cap,0,g[to].size()
      ));
  if(from==to) g[from].back().index++;
  g[to].push_back(edge(to,from,0,0, g[from].size()
      -1));}
 void enqueue(int v){
  if(!active[v] && excess[v]>0){
   active[v]=true;
   Q.push(v);}}
 void push(edge &e){
  int amt=(int)min(excess[e.from],(long long)e.cap
      - e.flow);
  if(height[e.from]<=height[e.to] || amt==0) return
      ;
  e.flow += amt;
  g[e.to][e.index].flow -= amt;
  excess[e.to] += amt;
  excess[e.from] -= amt;
  enqueue(e.to);}
 void relabel(int v){
  count[height[v]]--;
  int d=2*n;
  for(auto &it:g[v]){
   if(it.cap-it.flow>0) d=min(d, height[it.to]+1);}
  height[v]=d;
  count[height[v]]++;
  enqueue(v);}
 void gap(int k){
  for(int v=0;v<n;v++){
   if(height[v]<k) continue;
   count[height[v]]--;
   height[v]=max(height[v], n+1);
   count[height[v]]++;
   enqueue(v);}}
 void discharge(int v){
  for(int i=0; excess[v]>0 && i<g[v].size(); i++)
      push(g[v][i]);
  if(excess[v]>0){
   if(count[height[v]]==1) gap(height[v]);
   else relabel(v);}}
 long long max_flow(int source, int dest){
  count[0] = n-1;
  count[n] = 1;
  height[source] = n;
  active[source] = active[dest] = 1;
  for(auto &it:g[source]){
   excess[source]+=it.cap;
   push(it);}
  while(!Q.empty()){
   int v=Q.front();
   Q.pop();
   active[v]=false;
   discharge(v);}
  long long max_flow=0;
```

```
  for(auto &e:g[source]) max_flow+=e.flow;
  return max_flow;}};
```

34 Simplex

```
// Two-phase simplex algorithm for solving linear
    programs of the form
//     maximize     c^T x
//     subject to  Ax <= b
//                  x >= 0
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution
    will be stored
// OUTPUT: value of the optimal solution (infinity
     if unbounded
//         above, nan if infeasible)
// To use this code, create an LPSolver object
     with A, b, and c as
// arguments. Then, call Solve(x).
typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;
const DOUBLE EPS = 1e-9;

struct LPSolver {
  int m, n;
  VI B, N;
  VVD D;

  LPSolver(const VVD &A, const VD &b, const VD &c)
      :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m +
        2, VD(n + 2)) {
    for (int i = 0; i < m; i++) for (int j = 0; j <
        n; j++) D[i][j] = A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n + i; D[i
        ][n] = -1; D[i][n + 1] = b[i]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j]
        = -c[j]; }
    N[n] = -1; D[m + 1][n] = 1;}

void Pivot(int r, int s) {
  for (int i = 0; i < m + 2; i++) if (i != r)
    for (int j = 0; j < n + 2; j++) if (j != s)
      D[i][j] -= D[r][j] * D[i][s] / D[r][s];
  for (int j = 0; j < n + 2; j++) if (j != s) D[r
      ][j] /= D[r][s];
  for (int i = 0; i < m + 2; i++) if (i != r) D[i
      ][s] /= -D[r][s];
  D[r][s] = 1.0 / D[r][s];
  swap(B[r], N[s]);}

bool Simplex(int phase) {
  int x = phase == 1 ? m + 1 : m;
  while (true) {
    int s = -1;
    for (int j = 0; j <= n; j++) {
      if (phase == 2 && N[j] == -1) continue;
      if (s == -1 || D[x][j] < D[x][s] || D[x][j]
           == D[x][s] && N[j] < N[s]) s = j;}
    if (D[x][s] > -EPS) return true;
    int r = -1;
    for (int i = 0; i < m; i++) {
      if (D[i][s] < EPS) continue;
      if (r == -1 || D[i][n + 1] / D[i][s] < D[r
          ][n + 1] / D[r][s] ||
        (D[i][n + 1] / D[i][s]) == (D[r][n + 1] /
             D[r][s]) && B[i] < B[r]) r = i;}
    if (r == -1) return false;
    Pivot(r, s);}}

DOUBLE Solve(VD &x) {
  int r = 0;
  for (int i = 1; i < m; i++) if (D[i][n + 1] < D
      [r][n + 1]) r = i;
  if (D[r][n + 1] < -EPS) {
    Pivot(r, n);
    if (!Simplex(1) || D[m + 1][n + 1] < -EPS)
      return -numeric_limits<DOUBLE>::infinity
      ();
    for (int i = 0; i < m; i++) if (B[i] == -1) {
      int s = -1;
      for (int j = 0; j <= n; j++)
        if (s == -1 || D[i][j] < D[i][s] || D[i][
            j] == D[i][s] && N[j] < N[s]) s = j;
        Pivot(i, s);}}
    if (!Simplex(2)) return numeric_limits<DOUBLE
        >::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i
        ]] = D[i][n + 1];
    return D[m][n + 1];}};

int main() {
  const int m = 4;
  const int n = 3;
  DOUBLE _A[m][n] = {
    { 6, -1, 0 },
    { -1, -5, 0 },
    { 1, 5, 1 },
    { -1, -5, -1 }};
  DOUBLE _b[m] = { 10, -4, 5, -5 };
  DOUBLE _c[n] = { 1, -1, 0 };
  VVD A(m);
  VD b(_b, _b + m);
  VD c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i
      ] + n);
  LPSolver solver(A, b, c);
  VD x;
  DOUBLE value = solver.Solve(x);
  cerr << "VALUE: " << value << endl; // VALUE:
      1.29032
  cerr << "SOLUTION:"; // SOLUTION: 1.74194
      0.451613 1
  for (size_t i = 0; i < x.size(); i++) cerr << " "
       << x[i];}
```

35 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;}
        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;}
```

36 Suffix Automaton

```
struct state {
    int len, link;
    map<char, int> next;
};
const int MAXLEN = 100000;
state st[MAXLEN * 2];
int sz, last;
void sa_init() {
    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;}
void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;}
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;}
            st[q].link = st[cur].link = clone;}}
    last = cur;}
```

37 Suffix Tree

```
string s; int n;
struct node {
    int l, r, par, link;
    map<char,int> next;

    node (int l=0, int r=0, int par=-1)
        : l(l), r(r), par(par), link(-1) {}
    int len() { return r - l; }
    int &get (char c) {
        if (!next.count(c)) next[c] = -1;
        return next[c];}};
node t[MAXN]; int sz;

struct state {
    int v, pos;
    state (int v, int pos) : v(v), pos(pos) {}};
state ptr (0, 0);

state go (state st, int l, int r) {
    while (l < r)
        if (st.pos == t[st.v].len()) {
            st = state (t[st.v].get( s[l] ), 0);
            if (st.v == -1) return st;
        }
        else {
            if (s[ t[st.v].l + st.pos ] != s[l])
                return state (-1, -1);
            if (r-l < t[st.v].len() - st.pos)
                return state (st.v, st.pos + r-l);
            l += t[st.v].len() - st.pos;
            st.pos = t[st.v].len();}
    return st;}

int split (state st) {
    if (st.pos == t[st.v].len())
        return st.v;
    if (st.pos == 0)
        return t[st.v].par;
    node v = t[st.v];
    int id = sz++;
    t[id] = node (v.l, v.l+st.pos, v.par);
    t[v.par].get( s[v.l] ) = id;
    t[id].get( s[v.l+st.pos] ) = st.v;
    t[st.v].par = id;
    t[st.v].l += st.pos;
    return id;}
```

```
int get_link (int v) {
    if (t[v].link != -1) return t[v].link;
    if (t[v].par == -1) return 0;
    int to = get_link (t[v].par);
    return t[v].link = split (go (state(to,t[to].
        len()), t[v].l + (t[v].par==0), t[v].r));}

void tree_extend (int pos) {
    for(;;) {
        state nptr = go (ptr, pos, pos+1);
        if (nptr.v != -1) {
            ptr = nptr;
            return;}

        int mid = split (ptr);
        int leaf = sz++;
        t[leaf] = node (pos, n, mid);
        t[mid].get( s[pos] ) = leaf;

        ptr.v = get_link (mid);
        ptr.pos = t[ptr.v].len();
        if (!mid) break;}}

void build_tree() {
    sz = 1;
    for (int i=0; i<n; ++i)
        tree_extend (i);}
```

38 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 pb push_back
#define mp make_pair
#define fi first
#define se second
#define all(x) x.begin(),x.end()
#define memreset(a) memset(a,0,sizeof(a))
#define testcase(t) int t;cin>>t;while(t--)
#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 bitcount(a) __builtin_popcount(a) // set
    bits (add ll)
#define ln '\n'
#define getcurrtime() cerr<<"Time = "<<((double)
    clock()/CLOCKS_PER_SEC)<<endl
#define dbgarr(v,s,e) cerr<<#v<<" = "; forsn(i,s,e
    ) cerr<<v[i]<<", "; cerr<<endl
#define inputfile freopen("input.txt", "r", stdin)
#define outputfile freopen("output.txt", "w",
    stdout)
#define dbg(args...) { string _s = #args; replace(
    _s.begin(), _s.end(), ',', ' '); \
stringstream _ss(_s); istream_iterator<string> _it
    (_ss); err(_it, args); }
void err(istream_iterator<string> it) { cerr<<endl
    ; }
template<typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args...
     args) {
 cerr << *it << " = " << a << "\t"; err(++it, args
     ...);
}
template<typename T1,typename T2>
ostream& operator <<(ostream& c,pair<T1,T2> &v){
 c<<"("<<v.fi<<","<<v.se<<")"; return c;
}
template <template <class...> class TT, class ...T
    >
ostream& operator<<(ostream& out,TT<T...>& c){
    out<<"{ ";
    forstl(x,c) out<<x<<" ";
    out<<"}"; return out;
}
typedef long long ll;
typedef unsigned long long ull;
typedef long double ld;
typedef pair<ll,ll> p64;
typedef pair<int,int> p32;
typedef pair<int,p32> p96;
typedef vector<ll> v64;
typedef vector<int> v32;
typedef vector<v32> vv32;
typedef vector<v64> vv64;
typedef vector<p32> vp32;
typedef vector<p64> vp64;
typedef vector<vp32> vvp32;
typedef map<int,int> m32;
const int LIM=1e5+5,MOD=1e9+7;
const ld EPS = 1e-9;
mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
```

39 Topological Sort

```
int n; // number of vertices
vector<int> adj[LIM], ans; // adjacency list of
    graph
vector<bool> visited;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);}
    ans.push_back(v);}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);}
    reverse(ans.begin(), ans.end());}
```

40 Treap

```
mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
struct rope{
    struct node{
        int val,sz,priority,lazy,rev,sum;
        node *l, *r, *par;
        node(): lazy(0),rev(0),val(0),sz(0),sum(0),
            r(NULL),l(NULL),par(NULL){}
        node(int _val): lazy(0),rev(0),val(_val),
            sum(_val),sz(1),r(NULL),l(NULL),par(
            NULL){
            priority = rng();}};
    typedef node* pnode;
    pnode root;
    void clear(){root=NULL;}
    rope(){clear();}
    int size(pnode p){
        return p ? p->sz : 0;}
    void update_size(pnode &p){
        if(p) p->sz=size(p->l)+size(p->r)+1;}
    void update_parent(pnode &p){
        if(!p) return;
        if(p->l) p->l->par=p;
        if(p->r) p->r->par=p;}
    void push(pnode &p){
        if(!p) return;
        p->sum+=size(p)*p->lazy;
        p->val+=p->lazy;
        if(p->rev) swap(p->l,p->r);
        if(p->l){
            p->l->lazy+=p->lazy;
            p->l->rev^=p->rev; }
        if(p->r){
            p->r->lazy+=p->lazy;
            p->r->rev^=p->rev;}
        p->lazy = 0;
        p->rev = 0;}
    void reset(pnode &t){
        if(t) t->sum=t->val; }
    void combine(pnode &t, pnode l, pnode r){
        if(!l){
            t = r;return;}
        if(!r){
            t = l;return;}
        t->sum = l->sum + r->sum;}
    void operation(pnode &t){
        if(!t) return;
        reset(t);
        push(t->l);
        push(t->r);
        combine(t, t->l, t);
        combine(t, t, t->r);}
    void split(pnode t, pnode &l, pnode &r, int k,
     int add = 0){
        if(t == NULL){
            l=r=NULL; return;}
        push(t);
        int idx = add + size(t->l);
        if(idx <= k) split(t->r, t->r, r, k, idx +
            1), l = t;
        else split(t->l, l, t->l, k, add), r = t;
        update_parent(t);
        update_size(t);
        operation(t);}
    void merge(pnode &t, pnode l, pnode r){
        push(l);
        push(r);
        if(!l){
            t=r;return;}
        if(!r){
            t=l;return;}
        if(l->priority > r->priority) merge(l->r, l
            ->r, r), t = l;
        else merge(r->l, l, r->l), t = r;
        update_parent(t);
        update_size(t);
        operation(t);}
    void insert(int pos, int val){
        if(root == NULL){
            pnode to_add = new node(val);
            root = to_add;
            return;}
        pnode l, r, mid;
        mid = new node(val);
        split(root, l, r, pos - 1);
        merge(l, l, mid);
        merge(root, l, r);}
    void erase(int qL, int qR){
        pnode l, r, mid;
        split(root, l, r, qL - 1);
        split(r, mid, r, qR - qL);
        merge(root, l, r);}
    int query(int qL, int qR){
        pnode l, r, mid;
        split(root, l, r, qL - 1);
        split(r, mid, r, qR - qL);
        int answer = mid->sum;
        merge(r, mid, r);
        merge(root, l, r);
        return answer;}
    void update(int qL, int qR, int val){
        pnode l, r, mid;
        split(root, l, r, qL - 1);
        split(r, mid, r, qR - qL);
        mid->lazy += val;
        merge(r, mid, r);
        merge(root, l, r);}
    void reverse(int qL, int qR){
        pnode l, r, mid;
        split(root, l, r, qL - 1);
        split(r, mid, r, qR - qL);
        mid->rev ^= 1;
        merge(r, mid, r);
        merge(root, l, r);}
    void cyclic_shift(int qL, int qR, int k){
        if(qL == qR) return;
        k %= (qR - qL + 1);
        pnode l, r, mid, fh, sh;
        split(root, l, r, qL - 1);
        split(r, mid, r, qR - qL);
        split(mid, fh, sh, (qR - qL + 1) - k - 1);
        merge(mid, sh, fh);
        merge(r, mid, r);
        merge(root, l, r);}};
rope r;
r.insert(i,x);
r.cyclic_shift(lf,rt,1);
r.reverse(lf,rt);
r.query(x,x) <<" ";
```

41 Tree Bridge

```
struct dsu{
    v32 par,rk;
    dsu(){}
    dsu(int n) {reset(n);}
    void reset(int n){
        rk.assign(n,0);
        par.resize(n);
        iota(all(par),0);}
    int getpar(int i){
        return (par[i]==i)? i:(par[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;
        if(rk[i]>rk[j]) par[j]=i;
        else{
            par[i]=j;
            if(rk[i]==rk[j]) rk[j]++;}
        return 1;}};
vector<vector<int> > getBridgeTree(vector<vector<
    int> > &v,int n){
 int in[n]={0},low[n],ctm=0;
 vector<pair<int,int> > bridge;
 dsu d(n);
 function<void(int,int)> dfs=[&](int u,int p){
  in[u]=low[u]=++ctm;
  for(auto &it: v[u]){
   if(it==p) continue;
   if(in[it]){
    if(low[it]<low[u]) low[u]=low[it];
   }else{
    dfs(it,u);
    if(low[it]<low[u]) low[u]=low[it];
    if(low[it]>in[u]) bridge.push_back({u,it});
    else d.join(u,it);}}};
 for(int i=0;i<n;++i) if(!in[i]) dfs(i,i);
 int par[n],id[n];
 for(int i=0;i<n;++i) par[i]=d.getpar(i),id[i]=-1;
 vector<vector<int> > g;
 for(auto &it: bridge){
  it.first=par[it.first],it.second=par[it.second];
  if(id[it.first]==-1){
   id[it.first]=g.size();
   g.push_back(vector<int>(0));}
  if(id[it.second]==-1){
   id[it.second]=g.size();
   g.push_back(vector<int>(0));}
  g[id[it.first]].push_back(id[it.second]);
  g[id[it.second]].push_back(id[it.first]);}
 return g;}
// dfs find all bridgeges and g contain bridge tree
     rest is diameter calculation
```

42 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--; }}}}
```

43 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;}

Sparse Table:
int st[MAXN][K];
for (int i = 0; i < N; i++) st[i][0] = array[i];
for (int j = 1; j <= K; j++)
   for (int i = 0; i + (1 << j) <= N; i++)
       st[i][j] = min(st[i][j-1], st[i + (1 << (j
           - 1))][j - 1]);
To get minimum of a range:
int j = log[R - L + 1];
```

```
int minimum = min(st[L][j], st[R - (1 << j) + 1][j
    ]);

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 n;
long long C(int i, int j);
vector<long long> dp_before(n), dp_cur(n);
// 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 = {INF, -1};
    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {dp_before[k] + 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);}

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);}
```

Simpsons formula for integration:

```
const int N = 1000 * 1000; // number of steps (
    already multiplied by 2)
double simpson_integration(double a, double b){
    double h = (b - a) / N;
    double s = f(a) + f(b); // a = x_0 and b = x_2n
    for (int i = 1; i <= N - 1; ++i) { // Refer to
        final Simpson's formula
        double x = a + h * i;
        s += f(x) * ((i & 1) ? 4 : 2);}
    s *= h / 3;
    return s;}
```

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

Ternary Search

```
double ternary_search(double l, double r) {
    double eps = 1e-9;          //set the error
        limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);    //evaluates the
            function at m1
        double f2 = f(m2);    //evaluates the
            function at m2
        if (f1 < f2) l = m1;
        else r = m2;}
    return f(l);}                //return the
        maximum of f(x) in [l, r]
```

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]
```

Steiner tree dp:

The idea is to build a dynamic programming DP[i][m], where i is which vertex you are at and m is a bitmask of which capitals you joined. You can preprocess the APSP (Floyd-Warshall, or many Dijkstras because of the small constants) and calculate DP[i][m] like this: DP[i][m]=min(DP[i][s]+DP[j][m-s]+dist[i][j]), with s being a submask of m. In the end the complexity is O(3^k*n^2).
To get O(3^k * n) complexity, you do 2 transitions: 1. O(3^k * n) transition using submasks and 2. O(2^k * n^2) transition, that is, O(n^2) transition for each mask.

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)];}
```

15 puzzle problem: existence of solution

```
int a[16];
for (int i=0; i<16; ++i)
    cin >> a[i];
int inv = 0;
for (int i=0; i<16; ++i)
    if (a[i])
        for (int j=0; j<i; ++j)
            if (a[j] > a[i])
                ++inv;
for (int i=0; i<16; ++i)
    if (a[i] == 0)
        inv += 1 + i / 4;
puts ((inv & 1) ? "No Solution" : "Solution Exists");
```

Largest repetition string: (s = x+x)

```
vector<int> z_function(string const& s) {
    int n = s.size();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; i++) {
        if (i <= r) z[i] = min(r-i+1, z[i-l]);
        while (i + z[i] < n && s[z[i]] == s[i+z[i]])
            z[i]++;
        if (i + z[i] - 1 > r) {
            l = i;
            r = i + z[i] - 1;}}
    return z;}
int get_z(vector<int> const& z, int i) {
    if (0 <= i && i < (int)z.size()) return z[i];
    else return 0;}
vector<pair<int, int>> repetitions;
void convert_to_repetitions(int shift, bool left,
    int cntr, int l, int k1, int k2) {
    for (int l1 = max(1, l - k2); l1 <= min(l, k1); l1++) {
        if (left && l1 == l) break;
        int l2 = l - l1;
        int pos = shift + (left ? cntr - l1 : cntr - l - l1 + 1);
        repetitions.emplace_back(pos, pos + 2*l - 1);}}
void find_repetitions(string s, int shift = 0) {
    int n = s.size();
    if (n == 1)
        return;
    int nu = n / 2;
    int nv = n - nu;
    string u = s.substr(0, nu);
    string v = s.substr(nu);
    string ru(u.rbegin(), u.rend());
    string rv(v.rbegin(), v.rend());
    find_repetitions(u, shift);
    find_repetitions(v, shift + nu);
    vector<int> z1 = z_function(ru);
    vector<int> z2 = z_function(v + '#' + u);
    vector<int> z3 = z_function(ru + '#' + rv);
    vector<int> z4 = z_function(v);
    for (int cntr = 0; cntr < n; cntr++) {
        int l, k1, k2;
        if (cntr < nu) {
            l = nu - cntr;
            k1 = get_z(z1, nu - cntr);
            k2 = get_z(z2, nv + 1 + cntr);
        } else {
            l = cntr - nu + 1;
            k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
            k2 = get_z(z4, (cntr - nu) + 1);}
        if (k1 + k2 >= l)
            convert_to_repetitions(shift, cntr < nu, cntr, l, k1, k2);}}
```

44 Z Techniques

```
techniques
Tech
Recursion
DIVIDE AND CONQUER
 Finding interesting points in N log N
GREEDY ALGORITHM
 Scheduling
 Max contiguous subvector sum
 Invariants
 Huffman encoding
GRAPH THEORY
 Dynamic graphs (extra book-keeping)
 Breadth first search
 Depth first search
 * Normal trees / DFS trees
 Dijkstras algorithm
```

```
 MST: Prims algorithm
 Bellman-Ford
 Konigs theorem and vertex cover
 Min-cost max flow
 Matrix tree theorem
 Maximal matching, general graphs
 Hopcroft-Karp
 Halls marriage theorem
 Floyd-Warshall
 Euler cycles
 Flow networks
 * Augmenting paths
 * Edmonds-Karp
 Bipartite matching
 Min. path cover
 Topological sorting
 Strongly connected components
 2-SAT
 Cut vertices, cut-edges and biconnected components
 Edge coloring
 * Trees
 Vertex coloring
 * Bipartite graphs (=> trees)
 * 3^n (special case of set cover)
 Diameter and centroid
 Kth shortest path
 Shortest cycle
DYNAMIC PROGRAMMING
 Knapsack
 Coin change
 Longest common subsequence
 Longest increasing subsequence
 Number of paths, shortest path in a dag
 Dynprog over intervals, subsets, probabilities,
     trees
 3^n set cover
 Divide and conquer
 Knuth optimization
 Convex hull optimizations
 RMQ (sparse table a.k.a 2^k-jumps)
 Bitonic cycle
 Log partitioning (loop over most restricted)
COMBINATORICS
 Computation of binomial coefficients
 Pigeon-hole principle
 Inclusion/exclusion
 Catalan number
 Picks theorem
NUMBER THEORY
 Integer parts
 Divisibility
 Euclidean algorithm
 Modular arithmetic
 * Modular multiplication
 * Modular inverses
 * Modular exponentiation by squaring
 Chinese remainder theorem
 Fermats little theorem
 Eulers theorem
 Phi function
 Frobenius number
 Quadratic reciprocity
 Pollard-Rho
 Miller-Rabin
 Hensel lifting
 Vieta root jumping
GAME THEORY
 Combinatorial games
 Game trees
 Mini-max
 Nim
 Games on graphs
 Games on graphs with loops
 Grundy numbers
 Bipartite games without repetition
 General games without repetition
 Alpha-beta pruning
PROBABILITY THEORY
 Optimization
 Binary search
 Ternary search
 Unimodality and convex functions
 Binary search on derivative
NUMERICAL METHODS
 Numeric integration
 Newtons method
 Root-finding with binary/ternary search
 Golden section search
MATRICES
 Gaussian elimination
 Exponentiation by squaring
SORTING
 Radix sort
GEOMETRY
 Coordinates and vectors
 * Cross product
 * Scalar product
 Convex hull
 Polygon cut
 Closest pair
 Coordinate-compression
 Quadtrees
 KD-trees
 All segment-segment intersection
SWEEPING
 Discretization (convert to events and sweep)
 Angle sweeping
 Line sweeping
 Discrete second derivatives
STRINGS
 Longest common substring
 Palindrome subsequences
 Knuth-Morris-Pratt
 Tries
 Rolling polynomial hashes
 Suffix array
 Suffix tree
 Aho-Corasick
 Manachers algorithm
 Letter position lists
COMBINATORIAL SEARCH
 Meet in the middle
 Brute-force with pruning
 Best-first (A*)
 Bidirectional search
 Iterative deepening DFS / A*
DATA STRUCTURES
 LCA (2^k-jumps in trees in general)
 Pull/push-technique on trees
 Heavy-light decomposition
 Centroid decomposition
 Lazy propagation
 Self-balancing trees
 Convex hull trick
 Monotone queues / monotone stacks / sliding queues
 Sliding queue using 2 stacks
 Persistent segment tree
```