Contents

Sublime Build

```
{
    "shell_cmd": "g++ -std=c++17 -o ${file_path}/${
        file_base_name} ${file} && ${file_path}/${
        file_base_name} < input.txt > output.txt",
    "working_dir": "${file_path}",
    "selector": "source.cpp"
}
```

Sublime Build Ubuntu

```
{
    "shell_cmd": "g++ -std=c++20 -DLOCAL $file_name
         -o $file_base_name &&timeout 5s ./
        $file_base_name<in.txt>out.txt",
    "working_dir": "$file_path",
    "selector": "source.cpp"
}
```

Stress-tester

```
#!/bin/bash
# Call as stresstester ITERATIONS [--count]

g++ gen.cpp -o gen
g++ sol.cpp -o sol
g++ brute.cpp -o brute

for i in $(seq 1 "$1") ; do
    echo "Attempt $i/$1"
    ./gen > in.txt
    ./sol < in.txt > out1.txt
    ./brute < in.txt > out2.txt
    diff -y out1.txt out2.txt > diff.txt
    if [ $? -ne 0 ] ; then
        echo "Differing Testcase Found:"; cat in.txt
        echo -e "\nOutputs:"; cat diff.txt
        break
    fi
done
```

1 All Macros

```
/*--- DEBUG TEMPLATE STARTS HERE ---*/
void show(int x) {cerr << x;}
void show(long long x) {cerr << x;}
void show(double x) {cerr << x;}
void show(char x) {cerr << '\'' << x << '\'';}
void show(const string &x) {cerr << '\"' << x << '
    \"';}
void show(bool x) {cerr << (x ? "true" : "false");}

template<typename T, typename V>
void show(pair<T, V> x) { cerr << '{'; show(x.first
    ); cerr << ", "; show(x.second); cerr << '}'; }
template<typename T>
void show(T x) {int f = 0; cerr << "{"; for (auto &
    i: x) cerr << (f++ ? ", " : ""), show(i); cerr
    << "}";}

void debug_out(string s) {
  s.clear();
  cerr << s << '\n';
}
template <typename T, typename... V>
void debug_out(string s, T t, V... v) {
  s.erase(remove(s.begin(), s.end(), ' '), s.end())
      ;
  cerr << "        "; // 8 spaces
  cerr << s.substr(0, s.find(','));
  s = s.substr(s.find(',') + 1);
  cerr << " = ";
  show(t);
  cerr << endl;
  if(sizeof...(v)) debug_out(s, v...);
}
#define dbg(x...) cerr << "LINE: " << __LINE__ <<
    endl; debug_out(#x, x); cerr << endl;
/*--- DEBUG TEMPLATE ENDS HERE ---*/
//#pragma GCC optimize("Ofast")
//#pragma GCC optimization ("O3")
//#pragma comment(linker, "/stack:200000000")
//#pragma GCC optimize("unroll-loops")
//#pragma GCC target("sse,sse2,sse3,ssse3,sse4,
    popcnt,abm,mmx,avx,tune=native")

#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
  //find_by_order(k) --> returns iterator to the
      kth largest element counting from 0
  //order_of_key(val) --> returns the number of
      items in a set that are strictly smaller than
       our item
template <typename DT>
using ordered_set = tree <DT, null_type, less<DT>,
    rb_tree_tag,tree_order_statistics_node_update>;
mt19937 rnd(chrono::steady_clock::now().
    time_since_epoch().count());

#ifdef LOCAL
#include "dbg.h"
#else
#define dbg(x...)
#endif

int main() {
  cin.tie(0) -> sync_with_stdio(0);
}
```

2 Data Structure

2.1 Sparse Table

```
ll spars[MAX][18];

void build(vector<ll>& a) //1-indexed
{
  int n = a.size();
  for(int i = 1; i <= n; i++) spars[i][0] = a[i-1];

  for(int p = 1; p <= 18; p++)
  {
    for(int i = 1; i+(1<<p) - 1 <= n; i++)
    {
      spars[i][p] = min(spars[i][p-1], spars[i
          +(1<<(p-1))][p-1]);
    }
  }
}

ll query(int l, int r)
{
  int p = 31 - __builtin_clz(r-l+1);
  return min(spars[l][p], spars[r-(1<<p)+1][p]);
}
```

2.2 BIT

```
template <typename T> class BIT
```

```
{
  public:
    int n; vector<T> tree;

    BIT(int size) // 1-indexed
    {
      n = size; tree.assign(n+1, 0);
    }

    BIT(const vector<T> &a) : BIT(a.size())
    {
      for(int i = 1; i <= n; i++) update(i, a[i-1])
          ;
    }

    T query(int i)
    {
      T ans = 0;
      for( ; i >= 1; i-= (i & -i)) ans+= tree[i];
      return ans;
    }

    T query(int l, int r)
    {
      return query(r) - query(l-1);
    }

    void update(int i, T delta)
    {
      for( ; i <= n; i+= (i & -i)) tree[i]+= delta;
    }
};
```

2.3 Lazy SegmentTree

```
ll tree[4*MAX], lazy[4*MAX]; // 1-indexed

void build(vector<ll>&a, int b = 0, int e = -1, int
     v=1)
{
  if(e == -1) e = a.size()-1;

  if(b == e)
  {
    tree[v] = a[b];
    return;
  }
  int mid = (b+e)/2;
  build(a, b, mid, 2*v);
  build(a, mid+1, e,2*v+1);
  tree[v] = tree[2*v] + tree[2*v+1];
}

ll query(int l, int r, int b, int e, int v=1, ll
    carry = 0)
{
  if(b > r || e < l) return 0;
  if(b >= l && e <= r) return tree[v]+carry*(e-b+1)
      ;

  int mid = (b+e)/2;
  ll lseg = query(l, r, b, mid, 2*v, carry+lazy[v])
      ;
  ll rseg = query(l, r, mid+1, e, 2*v+1, carry+lazy
      [v]);
  return lseg + rseg;
}

void update(int l, int r, ll val, int b, int e, int
     v = 1)
{
  if(b > r || e < l) return;
  if(b >= l && e <= r)
  {
    tree[v]+= (e-b+1)*val;
    lazy[v]+= val;
    return;
  }

  int mid = (b+e)/2;
  update(l, r, val, b, mid, 2*v);
  update(l, r, val, mid+1, e, 2*v+1);
  tree[v] = tree[2*v] + tree[2*v+1]+ (e-b+1)*lazy[v
      ];
}
```

2.4 Generic SegmentTree

```
template<typename ST, typename LZ>
class SegmentTree {
private:
  int n;
  ST *tree, identity;
  ST (*merge) (ST, ST);

  LZ *lazy, unmark;
  void (*mergeLazy)(int, int, LZ&, LZ);
  void (*applyLazy)(int, int, ST&, LZ);

  void build(vector<ST> &arr, int lo, int hi, int
      cur=1)
  {
    if(lo == hi)
    {
      tree[cur] = arr[lo-1];
      return;
    }
    int mid = (hi+lo)/2, left = 2*cur, right = 2*
        cur+1;
    build(arr, lo, mid, left);
    build(arr, mid+1, hi, right);
    tree[cur] = merge(tree[left], tree[right]);
  }

  void propagate(int lo, int hi, int cur)
  {
    applyLazy(lo, hi, tree[cur], lazy[cur]);
    if(lo < hi)
    {
      int mid = (lo+hi)/2, left = 2*cur, right = 2*
          cur+1;
      mergeLazy(lo, mid, lazy[left], lazy[cur]);
      mergeLazy(mid+1, hi, lazy[right], lazy[cur]);
    }
    lazy[cur] = unmark;
  }

  void update(int from, int upto, LZ delta, int lo,
       int hi, int cur=1)
  {
    if(lo>hi) return;

    propagate(lo, hi, cur);
    if(from > hi or upto < lo) return;
    if(from<= lo and upto >= hi)
    {
      mergeLazy(lo, hi, lazy[cur], delta);
      propagate(lo, hi, cur);
      return;
```

```
    }
    int mid = (lo+hi)/2, left = 2*cur, right = 2*
        cur+1;
    update(from, upto, delta, lo, mid, left);
    update(from, upto, delta, mid+1, hi, right);
    tree[cur] = merge(tree[left], tree[right]);
  }

  ST query(int from, int upto, int lo, int hi, int
      cur=1)
  {
    if(lo>hi) return identity;

    propagate(lo, hi, cur);
    if(from > hi or upto < lo) return identity;
    if(from<= lo and upto >= hi) return tree[cur];
    int mid = (lo+hi)/2, left = 2*cur, right = 2*
        cur+1;
    ST lseg = query(from, upto, lo, mid, left);
    ST rseg = query(from, upto, mid+1, hi, right);
    return merge(lseg, rseg);
  }

public:
  SegmentTree(
    vector<ST> arr, ST (*merge) (ST, ST), ST
        identity,
    void (*mergeLazy)(int, int, LZ&, LZ),
    void (*applyLazy)(int, int, ST&, LZ), LZ unmark
  ):
    n(arr.size()), merge(merge), identity(identity)
        ,
    mergeLazy(mergeLazy), applyLazy(applyLazy),
        unmark(unmark)
  {
    tree = new ST[n*4];
    lazy = new LZ[n*4];
    build(arr, 1, n);
    fill(lazy, lazy+n*4, unmark);
  }

  void update(int from, int upto, LZ delta)
  {
    update(from, upto, delta, 1, n);
  }
  ST query(int from, int upto)
  {
    return query(from, upto, 1, n);
  }

  ~SegmentTree()
  {
    delete[] tree;
    delete[] lazy;
  }
};


ll add(ll l, ll r) { return l+r;}
void mergeAdd(int lo, int hi, ll &cur, ll pending)
    { cur+= pending;}
void applyAdd(int lo, int hi, ll &cur, ll pending)
    { cur+= pending*(hi-lo+1);}

void solve(int tcase)
{
  vector<ll> a(n);
  SegmentTree<ll, ll> st(a, add, 0, mergeAdd,
      applyAdd, 0);
}
```

2.5 MO

```
struct node {
  LL l, r, idx;
};
bool cmp(const node &x, const node &y) {
  return x.r < y.r;
}
void add(LL x) {
  if(mp[x] % 2) curr++;
  mp[x]++;
}
void diminish(LL x) {
  if(mp[x] % 2 == 0) curr--;
  mp[x]--;
}
void solve()
{
  BLOCK_SIZE = sqrt(n) + 1;
  rep(i, 0, q-1) {
    LL x, y; cin >> x >> y;
    x--; y--;
    query[x / BLOCK_SIZE].pb({x, y, i});
    m = max(m, x / BLOCK_SIZE);
  }
  rep(i, 0, m) sort(all(query[i]), cmp);
  LL mo_left = 0, mo_right = -1;
  rep(i, 0, m) {
    for(auto [left, right, id] : query[i]) {
      while(mo_right < right) add(v[++mo_right]);
      while(mo_right > right) diminish(v[mo_right
          --]);
      while(mo_left < left) diminish(v[mo_left++]);
      while(mo_left > left) add(v[--mo_left]);
      answer[id] = curr;
    }
  }
  rep(i, 0, q-1) cout << answer[i] << endl;
}
```

2.6 MergeSort Tree

```
vector<LL> tree[5*MAXN];
LL A[N];
void build_tree(LL now , LL curLeft, LL curRight) {
          if(curLeft == curRight) {
                tree[now].push_back(A[curLeft]);
                return;
          }
          LL mid = (curLeft + curRight) / 2;
          build_tree(2 * now, curLeft, mid);
          build_tree(2 * now + 1, mid + 1 , curRight
              );
          tree[now] = merge(tree[2 * now] , tree[2 *
               now + 1]);
}
LL query(LL now, LL curLeft, LL curRight, LL l, LL
    r, LL k) {
        if(curRight < l || curLeft > r) return 0;
        if(curLeft >= l && curRight <= r)
                Return lower_bound(tree[now].begin()
                    , tree[now].end(), k) - tree[now
                    ].begin();
        LL mid = (curLeft + curRight) / 2;
        return query(2 * now, curLeft, mid, l, r, k)
             + query(2 * now + 1, mid + 1, curRight,
             l, r, k);
}
```

2.7 BIT2d

```
const int N = 1008;
int bit[N][N], n, m;
int a[N][N], q;
void update(int x, int y, int val) {
  for (; x < N; x += -x & x)
    for (int j = y; j < N; j += -j & j) bit[x][j]
        += val;
}
int get(int x, int y) {
  int ans = 0;
  for (; x; x -= x & -x)
    for (int j = y; j; j -= j & -j) ans += bit[x][j
        ];
  return ans;
}
int get(int x1, int y1, int x2, int y2) {
  return get(x2, y2) - get(x1 - 1, y2) - get(x2, y1
      - 1) + get(x1 - 1, y1 - 1);
}
```

2.8 SparseTable2d

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 505;
const int LOGN = 9;

// O(n^2 (logn)^2
// Supports Rectangular Query
int A[MAXN][MAXN];
int M[MAXN][MAXN][LOGN][LOGN];

void Build2DSparse(int N) {
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            M[i][j][0][0] = A[i][j];
        }
        for (int q = 1; (1 << q) <= N; q++) {
            int add = 1 << (q - 1);
            for (int j = 1; j + add <= N; j++) {
                M[i][j][0][q] = max(M[i][j][0][q -
                    1], M[i][j + add][0][q - 1]);
            }
        }
    }
    for (int p = 1; (1 << p) <= N; p++) {
        int add = 1 << (p - 1);
        for (int i = 1; i + add <= N; i++) {
            for (int q = 0; (1 << q) <= N; q++) {
                for (int j = 1; j <= N; j++) {
                    M[i][j][p][q] = max(M[i][j][p -
                        1][q], M[i + add][j][p - 1][q
                        ]);
                }
            }
        }
    }
}

// returns max of all A[i][j], where x1<=i<=x2 and
    y1<=j<=y2
int Query(int x1, int y1, int x2, int y2) {
    int kX = log2(x2 - x1 + 1);
    int kY = log2(y2 - y1 + 1);
    int addX = 1 << kX;
    int addY = 1 << kY;

    int ret1 = max(M[x1][y1][kX][kY], M[x1][y2 -
        addY + 1][kX][kY]);
    int ret2 = max(M[x2 - addX + 1][y1][kX][kY],
                   M[x2 - addX + 1][y2
                       - addY + 1][kX][
                       kY]);
    return max(ret1, ret2);
}
```

2.9 SegmentTree

```
#include <bits/stdc++.h>

using namespace std;
using ll = long long;
using pii = pair<ll, ll>;

const ll N=5e5+5, mod=998244353;

using treenode = ll;
using lazynode = pii;
#define fir(a) for(int i=0; i<a; i++)
treenode treeidn = 0;
lazynode lazyidn = {1, 0};

vector<ll> v(N);
vector<treenode> tree(4*N, treeidn);
vector<lazynode> lazy(4*N, lazyidn);

treenode merge(treenode &a, treenode &b){
  return ;//add merge function here
}
void lazyapply(treenode &to, ll l, ll r, lazynode &
    fr){
  //apply lazy to treenode here
}
void lazymerge(lazynode &to, lazynode &fr){
  //combine to lazy updates here
}

//-------dont touch: start
void build(ll id, ll l, ll r){
  if(l==r){
    tree[id]=v[l];
    return;
  }
  ll m=(l+r)/2;
  build(id*2+1, l, m);
  build(id*2+2, m+1, r);
  tree[id] = merge(tree[id*2+1], tree[id*2+2]);
}
void push(ll id, ll l, ll r){
  if(l-r){
    ll m=(l+r)/2;
    lazyapply(tree[2*id+1], l, m, lazy[id]);
    lazymerge(lazy[2*id+1], lazy[id]);

    lazyapply(tree[2*id+2], m+1, r, lazy[id]);
    lazymerge(lazy[2*id+2], lazy[id]);

    lazy[id]=lazyidn;
  }
}
treenode query(ll id, ll l, ll r, ll ql, ll qr){
  push(id, l, r);
  if(ql<=l && r<=qr) return tree[id];
  if(ql>r || qr<l) return treeidn;
```

```
  ll m=(l+r)/2;
  treenode tl=query(id*2+1, l, m, ql, qr);
  treenode tr=query(id*2+2, m+1, r, ql, qr);
  return merge(tl, tr);
}
void update(ll id, ll l, ll r, ll ul, ll ur,
    lazynode uv){
  push(id, l, r);
  if(ul<=l && r<=ur){
    lazyapply(tree[id], l, r, uv);
    lazymerge(lazy[id], uv);
    return;
  }
  if(ul>r || ur<l) return;

  ll m=(l+r)/2;
  update(id*2+1, l, m, ul, ur, uv);
  update(id*2+2, m+1, r, ul, ur, uv);
  tree[id]=merge(tree[id*2+1], tree[id*2+2]);
  return;
}
//--------dont touch: end

void solve(){
  ll n, q; cin>>n>>q;
  fir(n) cin>>v[i];

  build(0, 0, n-1);

  while(q--){
    ll t; cin>>t;
    if(t){
      ll l, r; cin>>l>>r;
      cout<<query(0, 0, n-1, l, r-1)<<"\n";
    }else{
      ll l, r, a, b; cin>>l>>r>>a>>b;
      update(0, 0, n-1, l, r-1, {a, b});
    }
  }
  return;
}
```

2.10 SQRT Decomp

```
// input data
int n;
vector<int> a (n);
// preprocessing
int len = (int) sqrt (n + .0) + 1; // size of the
    block and the number of blocks
vector<int> b (len);
for (int i=0; i<n; ++i)
        b[i / len] += a[i];
// answering the queries
for (;;) {
        int l, r;
    // read input data for the next query
        int sum = 0;
        for (int i=l; i<=r; )
                if (i % len == 0 && i + len - 1 <= r
                    ) {
                        // if the whole block
                            starting at i belongs to
                            [l, r]
                        sum += b[i / len];
                        i += len;
                }
                else {
                        sum += a[i];
                        ++i;
                }
}

int sum = 0;
int c_l = l / len, c_r = r / len;
if (c_l == c_r)
        for (int i=l; i<=r; ++i)
                sum += a[i];
else {
        for (int i=l, end=(c_l+1)*len-1; i<=end; ++i
            )
                sum += a[i];
        for (int i=c_l+1; i<=c_r-1; ++i)
                sum += b[i];
        for (int i=c_r*len; i<=r; ++i)
                sum += a[i];
}


void remove(idx); // TODO: remove value at idx from
     data structure
void add(idx);   // TODO: add value at idx from
    data structure
int get_answer(); // TODO: extract the current
    answer of the data structure
int block_size;
struct Query {
        int l, r, idx;
        bool operator<(Query other) const
        {
                return make_pair(l / block_size, r)
                    <
                            make_pair(other.l /
                                block_size, other.r)
                                ;
        }
};
vector<int> mo_s_algorithm(vector<Query> queries) {
        vector<int> answers(queries.size());
        sort(queries.begin(), queries.end());

        // TODO: initialize data structure

        int cur_l = 0;
        int cur_r = -1;
        // invariant: data structure will always
            reflect the range [cur_l, cur_r]
        for (Query q : queries) {
                while (cur_l > q.l) {
                        cur_l--;
                        add(cur_l);
                }
                while (cur_r < q.r) {
                        cur_r++;
                        add(cur_r);
                }
                while (cur_l < q.l) {
                        remove(cur_l);
                        cur_l++;
                }
                while (cur_r > q.r) {
                        remove(cur_r);
                        cur_r--;
                }
                answers[q.idx] = get_answer();
        }
        return answers;
}
```

3 Graph

3.1 DSU BySize

```
vector<int> parent, setSize;
void make_set(int v) {
  parent[v] = v;
  setSize[v] = 1;
}
int find_set(int v) {
  if (v == parent[v]) return v;
  return parent[v] = find_set(parent[v]);
}
void union_sets(int a, int b) {
  a = find_set(a);
  b = find_set(b);
if (a != b) {
    if (setSize[a] < setSize[b]) swap(a, b);
    parent[b] = a;
    setSize[a] += setSize[b];
  }
}
int main() {
  int n;
  cin >> n;
  parent.resize(n);
  setSize.resize(n);
  for (int i = 0; i < n; i++) make_set(i);
}
```

3.2 MST Kruskal

```
const ll sz = 1e5 + 7;
vector<ll> pr(sz);

ll find(ll x) {
  if (pr[x] == x) return x;
  return pr[x] = find(pr[x]);
}

void _union(ll x, ll y) {
  pr[find(y)] = find(x);
}

signed main() {
  ll n, m, i;
  cin >> n >> m;
  vector<tuple<ll, ll, ll>> edg(m);
  iota(pr.begin(), pr.begin() + n + 1, 0);
  for (auto &[w, u, v] : edg) cin >> u >> v >> w;
  sort(edg.begin(), edg.end());

  ll cost = 0;
  for (auto [w, u, v] : edg) {
    if (find(u) != find(v)) {
      _union(u, v);
      cost += w;
    }
  }
  for (i = 1; i < n; i++) {
    if (find(i) != find(i + 1)) {
      cout << "IMPOSSIBLE\n";
      return 0;
    }
  }
  cout << cost << "\n";
}
```

3.3 Dijkstra

```
using pll = pair<ll, ll>;
vector<pll> adj[MAX];
vector<ll> dist(MAX, INF);
vector<ll> par(MAX, -1);

void dijkstra(int src)
{
  dist[src] = 0;
  priority_queue<pll, vector<pll>, greater<pll>> pq
      ;
  pq.push({0, src});

  while(!pq.empty())
  {
    auto [d, u] = pq.top();
    pq.pop();

    if(d > dist[u]) continue;

    for(auto &[v, w]: adj[u])
    {
      if(dist[u]+w < dist[v])
      {
        dist[v] = dist[u]+w;
        par[v] = u;
        pq.push({dist[v], v});
      }
    }
  }
}
```

3.4 Bellman Ford

```
#define sz 100007
ll INF = 1e18;
vector<tuple<ll, ll, ll>> edg;
vector<ll> dis(sz, INF);

void bellman_ford(ll n) {
  ll i, brk;
  dis[1] = 0ll;
  for (i = 1; i <= n; i++) {
    brk = 0;
    for (auto [u, v, w] : edg) {
      if (dis[v] > dis[u] + w)
        dis[v] = dis[u] + w; // for directional
            graph
      else
        brk++;
    }
    if (brk == n)
      break; // optimization
  }
}

bellman_ford(n);
```

3.5 Floyd Warshall

```
vector<vector<ll>> w(sz, vector<ll>(sz, inf));

void floyd_warshall(ll n) {
  ll i, j, k;
  for (i = 1; i <= n; i++)
    w[i][i] = 0;
  for (k = 1; k <= n; k++) {
    for (i = 1; i <= n; i++) {
      for (j = 1; j <= n; j++) {
        w[j][i] = w[i][j] = min(w[i][j], w[i][k] + w
            [k][j]); // for bidirectional graph
      }
    }
  }
}
```

```
w[b][a] = w[a][b] = min(c, w[a][b]); // for
    bidirectional graph
floyd_warshall(n);
```

3.6 SCC

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long

const ll sz = 1e5 + 7;
vector<ll> adj[sz];
vector<ll> Radj[sz];
vector<bool> vis(sz);
vector<ll> ord;

void dfs1(ll cur) {
  vis[cur] = 1;
  for (auto nxt : adj[cur]) {
    if (!vis[nxt])
      dfs1(nxt);
  }
  ord.push_back(cur);
}

void dfs2(ll cur) {
  vis[cur] = 1;
  for (auto nxt : Radj[cur]) {
    if (!vis[nxt])
      dfs2(nxt);
  }
}

signed main() {
  // strongly connected component kosaraju's
      algorithm
  ll n, m, a, b, i;
  cin >> n >> m;
  for (i = 0; i < m; i++) {
    cin >> a >> b;
    adj[a].push_back(b);
    Radj[b].push_back(a);
  }
  for (i = 1; i <= n; i++) {
    if (!vis[i])
      dfs1(i);
  }
  reverse(ord.begin(), ord.end());
  for (i = 1; i <= n; i++)
    vis[i] = 0;
  vector<ll> scc;
  for (auto e : ord) {
    if (!vis[e]) {
      scc.push_back(e);
      dfs2(e);
    }
  }
  if (scc.size() == 1) {
    cout << "YES\n";
    return 0;
  }
  cout << "NO\n";

  for (i = 1; i <= n; i++)
    vis[i] = 0;
  dfs2(scc[0]);
  if (vis[scc[1]])
    cout << scc[0] << " " << scc[1] << "\n";
  else
    cout << scc[1] << " " << scc[0] << "\n";
  return 0;
}
```

3.7 LCA

```
LL n, l, timer;
vector<vector<LL>> adj;
vector<LL> tin, tout;
vector<vector<LL>> up;
void dfs(LL v, LL p) {
  tin[v] = ++timer;
  up[v][0] = p;
  for (LL i = 1; i <= l; ++i)
    up[v][i] = up[up[v][i-1]][i-1];
  for (LL u : adj[v]) {
    if (u != p) dfs(u, v);
  }
  tout[v] = ++timer;
}
bool is_ancestor(LL u, LL v) {
  return tin[u] <= tin[v] && tout[u] >= tout[v];
}
LL lca(LL u, LL v) {
  if (is_ancestor(u, v)) return u;
  if (is_ancestor(v, u)) return v;
  for (LL i = l; i >= 0; --i) {
    if (!is_ancestor(up[u][i], v)) u = up[u][i];
  }
  return up[u][0];
}
void preprocess(LL root) {
  tin.resize(n);
  tout.resize(n);
  timer = 0;
  l = ceil(log2(n));
  up.assign(n, vector<LL>(l + 1));
  dfs(root, root);
}
```

3.8 EulerTourTree

```
using ll = long long;
using vi = vector<ll>;
using grid = vector<vi>;

void et(grid &edg, ll at, ll pt, grid &tr, ll &id){
  tr[0][id]=at; //val[at];
  tr[1][at]=id++;

  for(ll to: edg[at]) if(to-pt){
    et(edg, to, at, tr, id);
  }
  tr[0][id]=at; //val[at];
  tr[2][at]=id++;
  return;
}

grid etour(grid &edg, ll rt){
  ll cn=edg.size(), id=1;
  grid tour={vi(2*cn, 0), vi(cn), vi(cn)};
  et(edg, rt, 0, tour, id);
  return tour;
}
```

3.9 BFS

```
ll bfs(grid &edg, ll sn){
  ll cn=edg.size(), lv=-1, cl=0, nl=1, at, ls;
```

```
  vi vst(cn+1, 0), prt(cn+1, -1);
  queue<ll> call;
  call.push(sn); vst[sn]++;
  while(!call.empty()){
    if(!cl){
      lv++; cl=nl; nl=0;
    }

    at=call.front();
    //if(at==en) return lv;
    call.pop(); cl--; ls=at;
    for(ll to:edg[at]){
      if(!vst[to]){

        prt[to]=at;
        call.push(to);
        vst[to]++;
        nl++;
      }
    }
  }
  return 0;
  //return ls; //for deepest.
}
```

4 String

4.1 Hashing

```
class HashedString {
private:
  static const long long M = 1e9 + 7;
  static const long long B = 256;
  static vector<long long> pow;
  vector<long long> p_hash;

public:
  HashedString(const string& s) : p_hash(s.size() +
      1) {
    while (pow.size() < s.size()) {
      pow.push_back((pow.back() * B) % M);
    }
    p_hash[0] = 0;
    for (int i = 0; i < s.size(); i++) {
      p_hash[i + 1] = ((p_hash[i] * B) % M + s[i])
          % M;
    }
  }

  long long getHash(int start, int end) {
    long long raw_val = (
      p_hash[end + 1] - (p_hash[start] * pow[end -
          start + 1])
    );
    return (raw_val % M + M) % M;
  }
};

vector<long long> HashedString::pow = {1};
```

4.2 Double hash

```
// define +, -, * for (PLL, LL) and (PLL, PLL), %
    for (PLL, PLL);
PLL base(1949313259, 1997293877);
PLL mod(2091573227, 2117566807);

PLL power(PLL a, LL p) {
  PLL ans = PLL(1, 1);
  for(; p; p >>= 1, a = a * a % mod) {
      if(p & 1) ans = ans * a % mod;
  }
  return ans;
}

PLL inverse(PLL a) { return power(a, (mod.ff - 1) *
    (mod.ss - 1) - 1); }
PLL inv_base = inverse(base);
PLL val;
vector<PLL> P;

void hash_init(int n) {
  P.resize(n + 1);
  P[0] = PLL(1, 1);
  for (int i = 1; i <= n; i++) P[i] = (P[i - 1] *
      base) % mod;
}
PLL append(PLL cur, char c) { return (cur * base +
    c) % mod; }
/// prepends c to string with size k
PLL prepend(PLL cur, int k, char c) { return (P[k]
    * c + cur) % mod; }
/// replaces the i-th (0-indexed) character from
    right from a to b;
PLL replace(PLL cur, int i, char a, char b) {
  cur = (cur + P[i] * (b - a)) % mod;
  return (cur + mod) % mod;
}
/// Erases c from the back of the string
PLL pop_back(PLL hash, char c) {
  return (((hash - c) * inv_base) % mod + mod) %
      mod;
}
/// Erases c from front of the string with size len
PLL pop_front(PLL hash, int len, char c) {
  return ((hash - P[len - 1] * c) % mod + mod) %
      mod;
}
/// concatenates two strings where length of the
    right is k
PLL concat(PLL left, PLL right, int k) { return (
    left * P[k] + right) % mod; }
/// Calculates hash of string with size len
    repeated cnt times
/// This is O(log n). For O(1), pre-calculate
    inverses
PLL repeat(PLL hash, int len, LL cnt) {
  PLL mul = (P[len * cnt] - 1) * inverse(P[len] -
      1);
  mul = (mul % mod + mod) % mod;
  PLL ret = (hash * mul) % mod;
  if (P[len].ff == 1) ret.ff = hash.ff * cnt;
  if (P[len].ss == 1) ret.ss = hash.ss * cnt;
  return ret;
}
LL get(PLL hash) { return ((hash.ff << 32) ^ hash.
    ss); }
struct hashlist {
  int len;
  vector<PLL> H, R;
  hashlist() {}
  hashlist(string& s) {
    len = (int)s.size();
    hash_init(len);
    H.resize(len + 1, PLL(0, 0)), R.resize(len + 2,
        PLL(0, 0));
    for (int i = 1; i <= len; i++) H[i] = append(H[
        i - 1], s[i - 1]);
    for (int i = len; i >= 1; i--) R[i] = append(R[
        i + 1], s[i - 1]);
  }
```

```
  /// 1-indexed
  PLL range_hash(int l, int r) {
    return ((H[r] - H[l - 1] * P[r - l + 1]) % mod
        + mod) % mod;
  }
  PLL reverse_hash(int l, int r) {
    return ((R[l] - R[r + 1] * P[r - l + 1]) % mod
        + mod) % mod;
  }
  PLL concat_range_hash(int l1, int r1, int l2, int
      r2) {
    return concat(range_hash(l1, r1), range_hash(l2
        , r2), r2 - l2 + 1);
  }
  PLL concat_reverse_hash(int l1, int r1, int l2,
      int r2) {
    return concat(reverse_hash(l2, r2),
        reverse_hash(l1, r1), r1 - l1 + 1);
  }
};
```

4.3 Aho Corasick

```
struct AC {
int N, P;
const int A = 26;
vector<vector<int>> next;
vector<int> link, out_link;
vector<vector<int>> out;
AC() : N(0), P(0) { node(); }
int node() {
  next.emplace_back(A, 0);
  link.emplace_back(0);
  out_link.emplace_back(0);
  out.emplace_back(0);
  return N++;
}
inline int get(char c) { return c - 'a'; }
int add_pattern(const string T) {
  int u = 0;
  for (auto c : T) {
    if (!next[u][get(c)]) next[u][get(c)] = node();
    u = next[u][get(c)];
  }
  out[u].push_back(P);
  return P++;
}
void compute() {
  queue<int> q;
  for (q.push(0); !q.empty();) {
    int u = q.front(); q.pop();
    for (int c = 0; c < A; ++c) {
      int v = next[u][c];
      if (!v) next[u][c] = next[link[u]][c];
      else {
        link[v] = u ? next[link[u]][c] : 0;
        out_link[v] = out[link[v]].empty() ?
            out_link[link[v]] : link[v];
        q.push(v);
      }
    }
  }
}
int advance(int u, char c) {
  while (u && !next[u][get(c)]) u = link[u];
  u = next[u][get(c)];
  return u;
}
void match(const string S) {
  int u = 0;
  for (auto c : S) {
    u = advance(u, c);
    for (int v = u; v; v = out_link[v]) {
      for (auto p : out[v]) cout << "match " << p
          << endl;
    }
  }
}
};
int main() {
  AC aho; int n; cin >> n;
  while (n--) {
    string s; cin >> s;
    aho.add_pattern(s);
  }
  aho.compute(); string text;
  cin >> text; aho.match(text);
  return 0;
}
```

4.4 KMP

```
vector<int> prefix_function(string s) {
  int n = (int)s.length();
  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[i] == s[j])
      j++;
    pi[i] = j;
  }

  return pi;
}
```

4.5 Manacher's

```
vector<int> d1(n);
// d[i] = number of palindromes taking s[i] as
    center
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 1 : min(d1[l + r - i], r - i +
      1);
  while (0 <= i - k && i + k < n && s[i - k] == s[i
       + k]) k++;
  d1[i] = k--;
  if (i + k > r) l = i - k, r = i + k;
}
vector<int> d2(n);
// d[i] = number of palindromes taking s[i-1] and s
    [i] as center
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 0 : min(d2[l + r - i + 1], r -
      i + 1);
  while (0 <= i - k - 1 && i + k < n && s[i - k -
      1] == s[i + k]) k++;
  d2[i] = k--;
  if (i + k > r) l = i - k - 1, r = i + k;
}
```

4.6 Suffix Match FFT

```
// Find occurrences of t in s where '?'s are
    automatically matched with any character
// res[i + m - 1] = sum_j=0 to m - 1 { s[i + j] * t
    [j] * (s[i + j] - t[j]) }
vector<int> string_matching(string &s, string &t) {
  int n = s.size(), m = t.size();
  vector<int> s1(n), s2(n), s3(n);
```

```
  for(int i = 0; i < n; i++)
    s1[i] = s[i] == '?' ? 0 : s[i] - 'a' + 1; //
        assign any non zero number for non '?'s
  for(int i = 0; i < n; i++)
    s2[i] = s1[i] * s1[i];
  for(int i = 0; i < n; i++)
    s3[i] = s1[i] * s2[i];

  vector<int> t1(m), t2(m), t3(m);
  for(int i = 0; i < m; i++)
    t1[i] = t[i] == '?' ? 0 : t[i] - 'a' + 1;
  for(int i = 0; i < m; i++)
    t2[i] = t1[i] * t1[i];
  for(int i = 0; i < m; i++)
    t3[i] = t1[i] * t2[i];

  reverse(t1.begin(), t1.end());
  reverse(t2.begin(), t2.end());
  reverse(t3.begin(), t3.end());

  vector<int> s1t3 = multiply(s1, t3);
  vector<int> s2t2 = multiply(s2, t2);
  vector<int> s3t1 = multiply(s3, t1);

  vector<int> res(n);
  for(int i = 0; i < n; i++)
    res[i] = s1t3[i] - s2t2[i] * 2 + s3t1[i];

  vector<int> oc;
  for(int i = m - 1; i < n; i++)
    if(res[i] == 0)
      oc.push_back(i - m + 1);

  return oc;
}
```

4.7 Suffix Array

```
vector<VI> c;
VI sort_cyclic_shifts(const string &s) {
  int n = s.size();
  const int alphabet = 256;
  VI p(n), cnt(alphabet, 0);

  c.clear();
  c.emplace_back();
  c[0].resize(n);

  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[0][p[0]] = 0;
  int classes = 1;

  for (int i = 1; i < n; i++) {
    if (s[p[i]] != s[p[i - 1]]) classes++;
    c[0][p[i]] = classes - 1;
  }

  VI pn(n), cn(n);
  cnt.resize(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.end(), 0);
    /// radix sort
    for (int i = 0; i < n; i++) cnt[c[h][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[h][
        pn[i]]]] = pn[i];

    cn[p[0]] = 0;
    classes = 1;

    for (int i = 1; i < n; i++) {
      PII cur = {c[h][p[i]], c[h][(p[i] + (1 << h))
          % n]};
      PII prev = {c[h][p[i - 1]], c[h][(p[i - 1] +
          (1 << h)) % n]};
      if (cur != prev) ++classes;
      cn[p[i]] = classes - 1;
    }
    c.push_back(cn);
  }
  return p;
}
VI suffix_array_construction(string s) {
  s += "!";
  VI sorted_shifts = sort_cyclic_shifts(s);
  sorted_shifts.erase(sorted_shifts.begin());
  return sorted_shifts;
}
/// LCP between the ith and jth (i != j) suffix of
    the STRING
int suffixLCP(int i, int j) {
  assert(i != j);
  int log_n = c.size() - 1;

  int ans = 0;
  for (int k = log_n; k >= 0; k--) {
    if (c[k][i] == c[k][j]) {
      ans += 1 << k;
      i += 1 << k;
      j += 1 << k;
    }
  }
  return ans;
}

VI lcp_construction(const string &s, const VI &sa)
    {
  int n = s.size();
  VI rank(n, 0);
  VI lcp(n - 1, 0);

  for (int i = 0; i < n; i++) rank[sa[i]] = i;

  for (int i = 0, k = 0; i < n; i++, k -= (k != 0))
      {
    if (rank[i] == n - 1) {
      k = 0;
      continue;
    }
    int j = sa[rank[i] + 1];
    while (i + k < n && j + k < n && s[i + k] == s[
        j + k]) k++;
    lcp[rank[i]] = k;
  }
  return lcp;
}
```

4.8 Trie

```
template<int sz>
struct Trie {
```

```
  Trie() : id(1) {
    memset(endMark, 0, sizeof endMark);
    for_each(all(trie), [](vector<int> &v) { v.
        assign(sz, 0); });
  }

  void insert(const string &s) {
    int cur = 0;
    for (auto c : s) {
      int val = c - 'a';
      if (!trie[cur][val])
        trie[cur][val] = id++;
      cur = trie[cur][val];
    }
    endMark[cur] = true;
  }

  bool search(const string &s) {
    int cur = 0;
    for (auto c : s) {
      int val = c - 'a';
      if (!trie[cur][val])
        return false;
      cur = trie[cur][val];
    }
    return endMark[cur];
  }

private:
  int id, endMark[100005];
  vector<int> trie[100005];
};
```

4.9 Z Algo

```
vector<int> calcz(string s) {
  int n = s.size();
  vector<int> z(n);
  int l = 0, r = 0;
  for (int i = 1; i < n; i++) {
    if (i > r) {
      l = r = i;
      while (r < n && s[r] == s[r - l]) 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 < n && s[r] == s[r - l]) r++;
        z[i] = r - l, r--;
      }
    }
  }
  return z;
}
```

5 DP

5.1 Bitmask

```
for(int mask= 0; mask < (1 << 4); mask++){
  ll sum_of_set = 0;
  for(int i = 0; (1ll << i) <= mask; i++) if(mask&
      (1ll << i)) sum_of_set += v[i];
  if(sum_of_set == S){
    cout << "Yes\n";
    flg = true;
    break;
  }
}
if(!flg) cout << "No\n";
```

5.2 LIS

```
vector<pair<ll, ll>> LIS(vector<ll> &v){
  ll n=v.size();
  vector<pair<ll, ll>> seq(n); //{size, last
      element}
  set<ll> s; //multiset for non_dcrs
  for(int i=0; i<n; ++i){
    auto it=s.lower_bound(v[i]);
    if(it==s.end()) s.insert(v[i]);
    else{
      s.erase(it);
      s.insert(v[i]);
    }
    seq[i]={s.size(), *(s.rbegin())};
  }
  return seq;
} //seq[i] = {size of LIS in v[0, i], largest
    element in that sequence}
```

5.3 Divide and Conquer DP

```
const int K = 805, N = 4005;
LL dp[2][N], _cost[N][N];
// 1-indexed for convenience
LL cost(int l, int r) {
  return _cost[r][r] - _cost[l - 1][r] - _cost[r][l
      - 1] + _cost[l - 1][l - 1] >> 1;
}
void compute(int cnt, int l, int r, int optl, int
    optr) {
  if (l > r) return;
  int mid = l + r >> 1;
  LL best = INT_MAX;
  int opt = -1;
  for (int i = optl; i <= min(mid, optr); i++) {
    LL cur = dp[cnt ^ 1][i - 1] + cost(i, mid);
    if (cur < best) best = cur, opt = i;
  }
  dp[cnt][mid] = best;
  compute(cnt, l, mid - 1, optl, opt);
  compute(cnt, mid + 1, r, opt, optr);
}
LL dnc_dp(int k, int n) {
  fill(dp[0] + 1, dp[0] + n + 1, INT_MAX);
  for (int cnt = 1; cnt <= k; cnt++) {
    compute(cnt & 1, 1, n, 1, n);
  }
  return dp[k & 1][n];
}
```

5.4 Knuth Optimization

```
const int N = 1005;
LL dp[N][N], a[N];
int opt[N][N];
LL cost(int i, int j) { return a[j + 1] - a[i]; }
LL knuth_optimization(int n) {
  for (int i = 0; i < n; i++) {
    dp[i][i] = 0;
    opt[i][i] = i;
  }
  for (int i = n - 2; i >= 0; i--) {
    for (int j = i + 1; j < n; j++) {
      LL mn = LLONG_MAX;
      LL c = cost(i, j);
      for (int k = opt[i][j - 1]; k <= min(j - 1,
          opt[i + 1][j]); k++) {
        if (mn > dp[i][k] + dp[k + 1][j] + c) {
          mn = dp[i][k] + dp[k + 1][j] + c;
          opt[i][j] = k;
```

```
      }
    }
    dp[i][j] = mn;
  }
 }
 return dp[0][n - 1];
}
```

6 Math

6.1 BigMod

```
ll bigmod(ll a, ll b, ll m) {
 if(b == 0) return 1;
 ll x = bigmod(a, b/2, m);
 x = (x * x) % m;
 if(b % 2) x = (x * a) % m;
 return x;
}
```

6.2 Combi

```
array<int, N + 1> fact, inv, inv_fact;
void init() {
 fact[0] = inv_fact[0] = 1;
 for (int i = 1; i <= N; i++) {
  inv[i] = i == 1 ? 1 : (LL)inv[i - mod % i] * (
     mod / i + 1) % mod;
  fact[i] = (LL)fact[i - 1] * i % mod;
  inv_fact[i] = (LL)inv_fact[i - 1] * inv[i] %
     mod;
 }
}
LL C(int n, int r) {
 return (r < 0 or r > n) ? 0 : (LL)fact[n] *
    inv_fact[r] % mod * inv_fact[n - r] % mod;
}
```

6.3 Sieve

```
const ll m = 10e6;
vector<ll> lp(m+1);
vector<ll> prime;
void ln_sieve() {
 for(ll i = 2; i <= m; i++){
  if(!lp[i]){
   lp[i] = i;
   prime.push_back(i);
  }
  for(ll j = 0; i * prime[j] <= m; j++){
   lp[i * prime[j]] = prime[j];
   if(prime[j] == lp[i]) break;
  }
 }
}
```

6.4 Linear Sieve

```
const int N = 1e7;
vector<int> primes;
int spf[N + 5], phi[N + 5], NOD[N + 5], cnt[N + 5],
    POW[N + 5];
bool prime[N + 5];
int SOD[N + 5];
void init() {
 fill(prime + 2, prime + N + 1, 1);
 SOD[1] = NOD[1] = phi[1] = spf[1] = 1;
 for (LL i = 2; i <= N; i++) {
  if (prime[i]) {
   primes.push_back(i), spf[i] = i;
   phi[i] = i - 1;
   NOD[i] = 2, cnt[i] = 1;
   SOD[i] = i + 1, POW[i] = i;
  }
  for (auto p : primes) {
   if (p * i > N or p > spf[i]) break;
   prime[p * i] = false, spf[p * i] = p;
   if (i % p == 0) {
    phi[p * i] = p * phi[i];
    NOD[p * i] = NOD[i] / (cnt[i] + 1) * (cnt[i]
       + 2),
        cnt[p * i] = cnt[i] + 1;
    SOD[p * i] = SOD[i] / SOD[POW[i]] * (SOD[POW
      [i]] + p * POW[i]),
        POW[p * i] = p * POW[i];
    break;
   } else {
    phi[p * i] = phi[p] * phi[i];
    NOD[p * i] = NOD[p] * NOD[i], cnt[p * i] =
      1;
    SOD[p * i] = SOD[p] * SOD[i], POW[p * i] = p
      ;
   }
  }
 }
}
```

6.5 Pollard Rho

```
LL mul(LL a, LL b, LL mod) {
  return (__int128)a * b % mod;
  // LL ans = a * b - mod * (LL) (1.L / mod * a *
     b);
  // return ans + mod * (ans < 0) - mod * (ans >=
     (LL) mod);
}
LL bigmod(LL num, LL pow, LL mod) {
  LL ans = 1;
  for (; pow > 0; pow >>= 1, num = mul(num, num,
     mod))
    if (pow & 1) ans = mul(ans, num, mod);
  return ans;
}
bool is_prime(LL n) {
  if (n < 2 or n % 6 % 4 != 1) return (n | 1) ==
     3;
  LL a[] = {2, 325, 9375, 28178, 450775, 9780504,
     1795265022};
  LL s = __builtin_ctzll(n - 1), d = n >> s;
  for (LL x : a) {
    LL p = bigmod(x % n, d, n), i = s;
    for (; p != 1 and p != n - 1 and x % n and i
        --; p = mul(p, p, n))
        ;
    if (p != n - 1 and i != s) return false;
  }
  return true;
}
LL get_factor(LL n) {
  auto f = [&](LL x) { return mul(x, x, n) + 1;
     };
  LL x = 0, y = 0, t = 0, prod = 2, i = 2, q;
  for (; t++ % 40 or gcd(prod, n) == 1; x = f(x),
     y = f(f(y))) {
    (x == y) ? x = i++, y = f(x) : 0;
    prod = (q = mul(prod, max(x, y) - min(x, y),
       n)) ? q : prod;
  }
  return gcd(prod, n);
}
map<LL, int> factorize(LL n) {
  map<LL, int> res;
  if (n < 2) return res;
```

```
    LL small_primes[] = {2, 3, 5, 7, 11, 13, 17,
        19, 23, 29, 31, 37, 41,
                                            43, 47,
                                                53,
                                                59,
                                                61,
                                                67,
                                                71,
                                                73,
                                                79,
                                                83,
                                                89,
                                                97};
    for (LL p : small_primes)
        for (; n % p == 0; n /= p, res[p]++)
            ;

    auto _factor = [&](LL n, auto &_factor) {
        if (n == 1) return;
        if (is_prime(n))
            res[n]++;
        else {
            LL x = get_factor(n);
            _factor(x, _factor);
            _factor(n / x, _factor);
        }
    };
    _factor(n, _factor);
    return res;
}
```

6.6 Chinese Remainder Theorem

```
// given a, b will find solutions for
// ax + by = 1
tuple<LL, LL, LL> EGCD(LL a, LL b) {
  if (b == 0)
    return {1, 0, a};
  else {
    auto [x, y, g] = EGCD(b, a % b);
    return {y, x - a / b * y, g};
  }
}
// given modulo equations, will apply CRT
PLL CRT(vector<PLL> &v) {
  LL V = 0, M = 1;
  for (auto &[v, m] : v) { // value % mod
    auto [x, y, g] = EGCD(M, m);
    if ((v - V) % g != 0) return {-1, 0};
    V += x * (v - V) / g % (m / g) * M, M *= m / g;
    V = (V % M + M) % M;
  }
  return make_pair(V, M);
}
```

6.7 Mobius Function

```
const int N = 1e6 + 5;
int mob[N];
void mobius() {
  memset(mob, -1, sizeof mob);
  mob[1] = 1;
  for (int i = 2; i < N; i++)
    if (mob[i]) {
      for (int j = i + i; j < N; j += i) mob[j] -=
          mob[i];
    }
}
```

6.8 FFT

```
using CD = complex<double>;
typedef long long LL;
const double PI = acos(-1.0L);

int N;
vector<int> perm;
vector<CD> wp[2];
void precalculate(int n) {
  assert((n & (n - 1)) == 0), N = n;
  perm = vector<int>(N, 0);
  for (int k = 1; k < N; k <<= 1) {
    for (int i = 0; i < k; i++) {
      perm[i] <<= 1;
      perm[i + k] = 1 + perm[i];
    }
  }
  wp[0] = wp[1] = vector<CD>(N);
  for (int i = 0; i < N; i++) {
    wp[0][i] = CD(cos(2 * PI * i / N), sin(2 * PI *
        i / N));
    wp[1][i] = CD(cos(2 * PI * i / N), -sin(2 * PI
        * i / N));
  }
}
void fft(vector<CD> &v, bool invert = false) {
  if (v.size() != perm.size()) precalculate(v.size
      ());
  for (int i = 0; i < N; i++)
    if (i < perm[i]) swap(v[i], v[perm[i]]);
  for (int len = 2; len <= N; len *= 2) {
    for (int i = 0, d = N / len; i < N; i += len) {
      for (int j = 0, idx = 0; j < len / 2; j++,
          idx += d) {
        CD x = v[i + j];
        CD y = wp[invert][idx] * v[i + j + len / 2];
        v[i + j] = x + y;
        v[i + j + len / 2] = x - y;
      }
    }
  }
  if (invert) {
    for (int i = 0; i < N; i++) v[i] /= N;
  }
}
void pairfft(vector<CD> &a, vector<CD> &b, bool
    invert = false) {
  int N = a.size();
  vector<CD> p(N);
  for (int i = 0; i < N; i++) p[i] = a[i] + b[i] *
      CD(0, 1);
  fft(p, invert);
  p.push_back(p[0]);
  for (int i = 0; i < N; i++) {
    if (invert) {
      a[i] = CD(p[i].real(), 0);
      b[i] = CD(p[i].imag(), 0);
    } else {
      a[i] = (p[i] + conj(p[N - i])) * CD(0.5, 0);
      b[i] = (p[i] - conj(p[N - i])) * CD(0, -0.5);
    }
  }
}
vector<LL> multiply(const vector<LL> &a, const
    vector<LL> &b) {
  int n = 1;
  while (n < a.size() + b.size()) n <<= 1;
  vector<CD> fa(a.begin(), a.end()), fb(b.begin(),
      b.end());
  fa.resize(n);
  fb.resize(n);
```

```
  //       fft(fa); fft(fb);
  pairfft(fa, fb);
  for (int i = 0; i < n; i++) fa[i] = fa[i] * fb[i
      ];
  fft(fa, true);
  vector<LL> ans(n);
  for (int i = 0; i < n; i++) ans[i] = round(fa[i].
      real());
  return ans;
}
const int M = 1e9 + 7, B = sqrt(M) + 1;
vector<LL> anyMod(const vector<LL> &a, const vector
    <LL> &b) {
  int n = 1;
  while (n < a.size() + b.size()) n <<= 1;
  vector<CD> al(n), ar(n), bl(n), br(n);
  for (int i = 0; i < a.size(); i++) al[i] = a[i] %
      M / B, ar[i] = a[i] % M % B;
  for (int i = 0; i < b.size(); i++) bl[i] = b[i] %
      M / B, br[i] = b[i] % M % B;
  pairfft(al, ar);
  pairfft(bl, br);
  //       fft(al); fft(ar); fft(bl); fft(br);
  for (int i = 0; i < n; i++) {
    CD ll = (al[i] * bl[i]), lr = (al[i] * br[i]);
    CD rl = (ar[i] * bl[i]), rr = (ar[i] * br[i]);
    al[i] = ll;
    ar[i] = lr;
    bl[i] = rl;
    br[i] = rr;
  }
  pairfft(al, ar, true);
  pairfft(bl, br, true);
  //       fft(al, true); fft(ar, true); fft(bl,
      true); fft(br, true);
  vector<LL> ans(n);
  for (int i = 0; i < n; i++) {
    LL right = round(br[i].real()), left = round(al
        [i].real());
    ;
    LL mid = round(round(bl[i].real()) + round(ar[i
        ].real()));
    ans[i] = ((left % M) * B * B + (mid % M) * B +
        right) % M;
  }
  return ans;
}
```

6.9 NTT

```
const LL N = 1 << 18;
const LL MOD = 786433;

vector<LL> P[N];
LL rev[N], w[N | 1], a[N], b[N], inv_n, g;
LL Pow(LL b, LL p) {
  LL ret = 1;
  while (p) {
    if (p & 1) ret = (ret * b) % MOD;
    b = (b * b) % MOD;
    p >>= 1;
  }
  return ret;
}
LL primitive_root(LL p) {
  vector<LL> factor;
  LL phi = p - 1, n = phi;
  for (LL i = 2; i * i <= n; i++) {
    if (n % i) continue;
    factor.emplace_back(i);
    while (n % i == 0) n /= i;
  }
  if (n > 1) factor.emplace_back(n);
  for (LL res = 2; res <= p; res++) {
    bool ok = true;
    for (LL i = 0; i < factor.size() && ok; i++)
      ok &= Pow(res, phi / factor[i]) != 1;
    if (ok) return res;
  }
  return -1;
}
void prepare(LL n) {
  LL sz = abs(31 - __builtin_clz(n));
  LL r = Pow(g, (MOD - 1) / n);
  inv_n = Pow(n, MOD - 2);
  w[0] = w[n] = 1;
  for (LL i = 1; i < n; i++) w[i] = (w[i - 1] * r)
      % MOD;
  for (LL i = 1; i < n; i++)
    rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (sz -
        1));
}
void NTT(LL *a, LL n, LL dir = 0) {
  for (LL i = 1; i < n - 1; i++)
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (LL m = 2; m <= n; m <<= 1) {
    for (LL i = 0; i < n; i += m) {
      for (LL j = 0; j < (m >> 1); j++) {
        LL &u = a[i + j], &v = a[i + j + (m >> 1)];
        LL t = v * w[dir ? n - n / m * j : n / m * j
            ] % MOD;
        v = u - t < 0 ? u - t + MOD : u - t;
        u = u + t >= MOD ? u + t - MOD : u + t;
      }
    }
  }
  if (dir)
    for (LL i = 0; i < n; i++) a[i] = (inv_n * a[i
        ]) % MOD;
}
vector<LL> mul(vector<LL> p, vector<LL> q) {
  LL n = p.size(), m = q.size();
  LL t = n + m - 1, sz = 1;
  while (sz < t) sz <<= 1;
  prepare(sz);

  for (LL i = 0; i < n; i++) a[i] = p[i];
  for (LL i = 0; i < m; i++) b[i] = q[i];
  for (LL i = n; i < sz; i++) a[i] = 0;
  for (LL i = m; i < sz; i++) b[i] = 0;

  NTT(a, sz);
  NTT(b, sz);
  for (LL i = 0; i < sz; i++) a[i] = (a[i] * b[i])
      % MOD;
  NTT(a, sz, 1);

  vector<LL> c(a, a + sz);
  while (c.size() && c.back() == 0) c.pop_back();
  return c;
}
```

6.10 ModInverse

```
//solves ax+by=gcd(a, b) i guess
int gcd(int a, int b, int& x, int& y) {
  x = 1, y = 0;
  int x1 = 0, y1 = 1, a1 = a, b1 = b;
  while (b1) {
```

```cpp
    int q = a1 / b1;
    tie(x, x1) = make_tuple(x1, x - q * x1);
    tie(y, y1) = make_tuple(y1, y - q * y1);
    tie(a1, b1) = make_tuple(b1, a1 - q * b1);
  }
  return a1;
}

//finds mod inverse?
int x, y;
int g = gcd(a, m, x, y);
if (g != 1) {
  cout << "No solution!";
}
else {
  x = (x % m + m) % m;
  cout << x << endl;
}
```

6.11 Diophantine

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

bool find_any_solution(int a, int b, int c, int &x0
    , int &y0, int &g) {
  g = gcd(abs(a), abs(b), x0, y0);
  if (c % g) {
    return false;
  }

  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}

void shift_solution(int & x, int & y, int a, int b,
     int cnt) {
  x += cnt * b;
  y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int
    minx, int maxx, int miny, int maxy) {
  int x, y, g;
  if (!find_any_solution(a, b, c, x, y, g))
    return 0;
  a /= g;
  b /= g;

  int sign_a = a > 0 ? +1 : -1;
  int sign_b = b > 0 ? +1 : -1;

  shift_solution(x, y, a, b, (minx - x) / b);
  if (x < minx)
    shift_solution(x, y, a, b, sign_b);
  if (x > maxx)
    return 0;
  int lx1 = x;

  shift_solution(x, y, a, b, (maxx - x) / b);
  if (x > maxx)
    shift_solution(x, y, a, b, -sign_b);
  int rx1 = x;

  shift_solution(x, y, a, b, -(miny - y) / a);
  if (y < miny)
    shift_solution(x, y, a, b, -sign_a);
  if (y > maxy)
    return 0;
  int lx2 = x;

  shift_solution(x, y, a, b, -(maxy - y) / a);
  if (y > maxy)
    shift_solution(x, y, a, b, sign_a);
  int rx2 = x;

  if (lx2 > rx2)
    swap(lx2, rx2);
  int lx = max(lx1, lx2);
  int rx = min(rx1, rx2);

  if (lx > rx)
    return 0;
  return (rx - lx) / abs(b) + 1;
}
```

7 Geometry

7.1 Point

```cpp
typedef double Tf;
typedef double Ti; /// use long long for exactness
const Tf PI = acos(-1), EPS = 1e-9;
int dcmp(Tf x) { return abs(x) < EPS ? 0 : (x < 0 ?
     -1 : 1); }

struct Point {
    Ti x, y;
    Point(Ti x = 0, Ti y = 0) : x(x), y(y) {}

    Point operator+(const Point& u) const { return
        Point(x + u.x, y + u.y); }
    Point operator-(const Point& u) const { return
        Point(x - u.x, y - u.y); }
    Point operator*(const LL u) const { return
        Point(x * u, y * u); }
    Point operator*(const Tf u) const { return
        Point(x * u, y * u); }
    Point operator/(const Tf u) const { return
        Point(x / u, y / u); }

    bool operator==(const Point& u) const {
        return dcmp(x - u.x) == 0 && dcmp(y - u.y)
            == 0;
    }
    bool operator!=(const Point& u) const { return
        !(*this == u); }
    bool operator<(const Point& u) const {
        return dcmp(x - u.x) < 0 || (dcmp(x - u.x)
            == 0 && dcmp(y - u.y) < 0);
    }
};
Ti dot(Point a, Point b) { return a.x * b.x + a.y *
     b.y; }
Ti cross(Point a, Point b) { return a.x * b.y - a.y
     * b.x; }
Tf length(Point a) { return sqrt(dot(a, a)); }
Ti sqLength(Point a) { return dot(a, a); }
```

```
Tf distance(Point a, Point b) { return length(a - b
    ); }
Tf angle(Point u) { return atan2(u.y, u.x); }

// returns angle between oa, ob in (-PI, PI]
Tf angleBetween(Point a, Point b) {
    Tf ans = angle(b) - angle(a);
    return ans <= -PI ? ans + 2 * PI : (ans > PI ?
        ans - 2 * PI : ans);
}
// Rotate a ccw by rad radians, Tf Ti same
Point rotate(Point a, Tf rad) {
    return Point(a.x * cos(rad) - a.y * sin(rad),
                a.x * sin(rad) + a.y *
                    cos(rad));
}
// rotate a ccw by angle th with cos(th) = co &&
    sin(th) = si, tf ti same
Point rotatePrecise(Point a, Tf co, Tf si) {
    return Point(a.x * co - a.y * si, a.y * co + a.
        x * si);
}
Point rotate90(Point a) { return Point(-a.y, a.x);
    }
// scales vector a by s such that length of a
    becomes s, Tf Ti same
Point scale(Point a, Tf s) { return a / length(a) *
     s; }
// returns an unit vector perpendicular to vector a
    , Tf Ti same
Point normal(Point a) {
    Tf l = length(a);
    return Point(-a.y / l, a.x / l);
}
// returns 1 if c is left of ab, 0 if on ab && -1
    if right of ab
int orient(Point a, Point b, Point c) { return dcmp
    (cross(b - a, c - a)); }
/// Use as sort(v.begin(), v.end(), polarComp(O,
    dir))
/// Polar comparator around O starting at direction
    dir
struct polarComp {
    Point O, dir;
    polarComp(Point O = Point(0, 0), Point dir =
        Point(1, 0)) : O(O), dir(dir) {}
    bool half(Point p) {
        return dcmp(cross(dir, p)) < 0 ||
                (dcmp(cross(dir, p)) == 0 &&
                    dcmp(dot(dir, p)) > 0);
    }
    bool operator()(Point p, Point q) {
        return make_tuple(half(p), 0) < make_tuple(
            half(q), cross(p, q));
    }
};
struct Segment {
    Point a, b;
    Segment(Point aa, Point bb) : a(aa), b(bb) {}
};
typedef Segment Line;
struct Circle {
    Point o;
    Tf r;
    Circle(Point o = Point(0, 0), Tf r = 0) : o(o),
         r(r) {}
    // returns true if point p is in || on the
        circle
    bool contains(Point p) { return dcmp(sqLength(p
         - o) - r * r) <= 0; }
    // returns a point on the circle rad radians
        away from +X CCW
    Point point(Tf rad) {
        static_assert(is_same<Tf, Ti>::value);
        return Point(o.x + cos(rad) * r, o.y + sin(
            rad) * r);
    }
    // area of a circular sector with central angle
         rad
    Tf area(Tf rad = PI + PI) { return rad * r * r
        / 2; }
    // area of the circular sector cut by a chord
        with central angle alpha
    Tf sector(Tf alpha) { return r * r * 0.5 * (
        alpha - sin(alpha)); }
};
```

7.2 Linear

```
// **** LINE LINE INTERSECTION START ****
// returns true if point p is on segment s
bool onSegment(Point p, Segment s) {
  return dcmp(cross(s.a - p, s.b - p)) == 0 && dcmp
      (dot(s.a - p, s.b - p)) <= 0;
}
// returns true if segment p && q touch or
    intersect
bool segmentsIntersect(Segment p, Segment q) {
  if (onSegment(p.a, q) || onSegment(p.b, q))
      return true;
  if (onSegment(q.a, p) || onSegment(q.b, p))
      return true;

  Ti c1 = cross(p.b - p.a, q.a - p.a);
  Ti c2 = cross(p.b - p.a, q.b - p.a);
  Ti c3 = cross(q.b - q.a, p.a - q.a);
  Ti c4 = cross(q.b - q.a, p.b - q.a);
  return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp
      (c4) < 0;
}
bool linesParallel(Line p, Line q) {
  return dcmp(cross(p.b - p.a, q.b - q.a)) == 0;
}
// lines are represented as a ray from a point: (
    point, vector)
// returns false if two lines (p, v) && (q, w) are
    parallel or collinear
// true otherwise, intersection point is stored at
    o via reference, Tf Ti Same
bool lineLineIntersection(Point p, Point v, Point q
    , Point w, Point& o) {
  if (dcmp(cross(v, w)) == 0) return false;
  Point u = p - q;
  o = p + v * (cross(w, u) / cross(v, w));
  return true;
}
// returns false if two lines p && q are parallel
    or collinear
// true otherwise, intersection point is stored at
    o via reference
bool lineLineIntersection(Line p, Line q, Point& o)
     {
  return lineLineIntersection(p.a, p.b - p.a, q.a,
      q.b - q.a, o);
}
// returns the distance from point a to line l
// **** LINE LINE INTERSECTION FINISH ****
Tf distancePointLine(Point p, Line l) {
```

```cpp
  return abs(cross(l.b - l.a, p - l.a) / length(l.b
      - l.a));
}
// returns the shortest distance from point a to
    segment s
Tf distancePointSegment(Point p, Segment s) {
  if (s.a == s.b) return length(p - s.a);
  Point v1 = s.b - s.a, v2 = p - s.a, v3 = p - s.b;
  if (dcmp(dot(v1, v2)) < 0)
    return length(v2);
  else if (dcmp(dot(v1, v3)) > 0)
    return length(v3);
  else
    return abs(cross(v1, v2) / length(v1));
}
// returns the shortest distance from segment p to
    segment q
Tf distanceSegmentSegment(Segment p, Segment q) {
  if (segmentsIntersect(p, q)) return 0;
  Tf ans = distancePointSegment(p.a, q);
  ans = min(ans, distancePointSegment(p.b, q));
  ans = min(ans, distancePointSegment(q.a, p));
  ans = min(ans, distancePointSegment(q.b, p));
  return ans;
}
// returns the projection of point p on line l, Tf
    Ti Same
Point projectPointLine(Point p, Line l) {
  Point v = l.b - l.a;
  return l.a + v * ((Tf)dot(v, p - l.a) / dot(v, v)
      );
}
```

7.3 Circular

```cpp
// Extremely inaccurate for finding near touches
// compute intersection of line l with circle c
// The intersections are given in order of the ray
    (l.a, l.b), Tf Ti same
vector<Point> circleLineIntersection(Circle c, Line
     l) {
    vector<Point> ret;
    Point b = l.b - l.a, a = l.a - c.o;
    Tf A = dot(b, b), B = dot(a, b);
    Tf C = dot(a, a) - c.r * c.r, D = B * B - A * C
       ;
    if (D < -EPS) return ret;
    ret.push_back(l.a + b * (-B - sqrt(D + EPS)) /
        A);
    if (D > EPS) ret.push_back(l.a + b * (-B + sqrt
        (D)) / A);
    return ret;
}
// signed area of intersection of circle(c.o, c.r)
    &&
// triangle(c.o, s.a, s.b) [cross(a-o, b-o)/2]
Tf circleTriangleIntersectionArea(Circle c, Segment
     s) {
    using Linear::distancePointSegment;
    Tf OA = length(c.o - s.a);
    Tf OB = length(c.o - s.b);
    // sector
    if (dcmp(distancePointSegment(c.o, s) - c.r) >=
        0)
       return angleBetween(s.a - c.o, s.b - c.o) *
           (c.r * c.r) / 2.0;
    // triangle
    if (dcmp(OA - c.r) <= 0 && dcmp(OB - c.r) <= 0)
       return cross(c.o - s.b, s.a - s.b) / 2.0;
    // three part: (A, a) (a, b) (b, B)
    vector<Point> Sect = circleLineIntersection(c,
        s);
    return circleTriangleIntersectionArea(c,
        Segment(s.a, Sect[0])) +
               circleTriangleIntersectionArea(c,
                  Segment(Sect[0], Sect[1])) +
               circleTriangleIntersectionArea(c,
                  Segment(Sect[1], s.b));
}
// area of intersecion of circle(c.o, c.r) &&
    simple polyson(p[])
Tf circlePolyIntersectionArea(Circle c, Polygon p)
    {
    Tf res = 0;
    int n = p.size();
    for (int i = 0; i < n; ++i)
       res += circleTriangleIntersectionArea(c,
           Segment(p[i], p[(i + 1) % n]));
    return abs(res);
}
// locates circle c2 relative to c1
// interior           (d < R - r)        ----> -2
// interior tangents (d = R - r)      ----> -1
// concentric      (d = 0)
// secants          (R - r < d < R + r) ----> 0
// exterior tangents (d = R + r)      ----> 1
// exterior          (d > R + r)        ----> 2
int circleCirclePosition(Circle c1, Circle c2) {
    Tf d = length(c1.o - c2.o);
    int in = dcmp(d - abs(c1.r - c2.r)), ex = dcmp(
        d - (c1.r + c2.r));
    return in < 0 ? -2 : in == 0 ? -1 : ex == 0 ? 1
        : ex > 0 ? 2 : 0;
}
// compute the intersection points between two
    circles c1 && c2, Tf Ti same
vector<Point> circleCircleIntersection(Circle c1,
    Circle c2) {
    vector<Point> ret;
    Tf d = length(c1.o - c2.o);
    if (dcmp(d) == 0) return ret;
    if (dcmp(c1.r + c2.r - d) < 0) return ret;
    if (dcmp(abs(c1.r - c2.r) - d) > 0) return ret;

    Point v = c2.o - c1.o;
    Tf co = (c1.r * c1.r + sqLength(v) - c2.r * c2.
        r) / (2 * c1.r * length(v));
    Tf si = sqrt(abs(1.0 - co * co));
    Point p1 = scale(rotatePrecise(v, co, -si), c1.
        r) + c1.o;
    Point p2 = scale(rotatePrecise(v, co, si), c1.r
        ) + c1.o;

    ret.push_back(p1);
    if (p1 != p2) ret.push_back(p2);
    return ret;
}
// intersection area between two circles c1, c2
Tf circleCircleIntersectionArea(Circle c1, Circle
    c2) {
    Point AB = c2.o - c1.o;
    Tf d = length(AB);
    if (d >= c1.r + c2.r) return 0;
    if (d + c1.r <= c2.r) return PI * c1.r * c1.r;
    if (d + c2.r <= c1.r) return PI * c2.r * c2.r;

    Tf alpha1 = acos((c1.r * c1.r + d * d - c2.r *
        c2.r) / (2.0 * c1.r * d));
```

```
    Tf alpha2 = acos((c2.r * c2.r + d * d - c1.r *
        c1.r) / (2.0 * c2.r * d));
    return c1.sector(2 * alpha1) + c2.sector(2 *
        alpha2);
}
// returns tangents from a point p to circle c, Tf
    Ti same
vector<Point> pointCircleTangents(Point p, Circle c
    ) {
    vector<Point> ret;
    Point u = c.o - p;
    Tf d = length(u);
    if (d < c.r)
        ;
    else if (dcmp(d - c.r) == 0) {
        ret = {rotate(u, PI / 2)};
    } else {
        Tf ang = asin(c.r / d);
        ret = {rotate(u, -ang), rotate(u, ang)};
    }
    return ret;
}
// returns the points on tangents that touches the
    circle, Tf Ti Same
vector<Point> pointCircleTangencyPoints(Point p,
    Circle c) {
    Point u = p - c.o;
    Tf d = length(u);
    if (d < c.r)
        return {};
    else if (dcmp(d - c.r) == 0)
        return {c.o + u};
    else {
        Tf ang = acos(c.r / d);
        u = u / length(u) * c.r;
        return {c.o + rotate(u, -ang), c.o + rotate(
            u, ang)};
    }
}
// for two circles c1 && c2, returns two list of
    points a && b
// such that a[i] is on c1 && b[i] is c2 && for
    every i
// Line(a[i], b[i]) is a tangent to both circles
// CAUTION: a[i] = b[i] in case they touch | -1 for
     c1 = c2
int circleCircleTangencyPoints(Circle c1, Circle c2
    , vector<Point> &a, vector<Point> &b) {
    a.clear(), b.clear();
    int cnt = 0;

    if (dcmp(c1.r - c2.r) < 0) {
        swap(c1, c2);
        swap(a, b);
    }

    Tf d2 = sqLength(c1.o - c2.o);
    Tf rdif = c1.r - c2.r, rsum = c1.r + c2.r;

    if (dcmp(d2 - rdif * rdif) < 0)
        return 0;
    if (dcmp(d2) == 0 && dcmp(c1.r - c2.r) == 0)
        return -1;

    Tf base = angle(c2.o - c1.o);

    if (dcmp(d2 - rdif * rdif) == 0) {
        a.push_back(c1.point(base));
        b.push_back(c2.point(base));
        cnt++;
        return cnt;
    }

    Tf ang = acos((c1.r - c2.r) / sqrt(d2));
    a.push_back(c1.point(base + ang));
    b.push_back(c2.point(base + ang));
    cnt++;
    a.push_back(c1.point(base - ang));
    b.push_back(c2.point(base - ang));
    cnt++;

    if (dcmp(d2 - rsum * rsum) == 0) {
        a.push_back(c1.point(base));
        b.push_back(c2.point(PI + base));
        cnt++;
    } else if (dcmp(d2 - rsum * rsum) > 0) {
        Tf ang = acos((c1.r + c2.r) / sqrt(
            d2));
        a.push_back(c1.point(base + ang));
        b.push_back(c2.point(PI + base + ang
            ));
        cnt++;
        a.push_back(c1.point(base - ang));
        b.push_back(c2.point(PI + base - ang
            ));
        cnt++;
    }
    return cnt;
}
```

7.4 Convex

```
/// minkowski sum of two polygons in O(n)
Polygon minkowskiSum(Polygon A, Polygon B) {
    int n = A.size(), m = B.size();
    rotate(A.begin(), min_element(A.begin(), A.end
        ()), A.end());
    rotate(B.begin(), min_element(B.begin(), B.end
        ()), B.end());

    A.push_back(A[0]);
    B.push_back(B[0]);
    for (int i = 0; i < n; i++) A[i] = A[i + 1] - A
        [i];
    for (int i = 0; i < m; i++) B[i] = B[i + 1] - B
        [i];

    Polygon C(n + m + 1);
    C[0] = A.back() + B.back();
    merge(A.begin(), A.end() - 1, B.begin(), B.end
        () - 1, C.begin() + 1,
            polarComp(Point(0, 0), Point(0, -1))
                );
    for (int i = 1; i < C.size(); i++) C[i] = C[i]
        + C[i - 1];
    C.pop_back();
    return C;
}
// finds the rectangle with minimum area enclosing
    a convex polygon and
// the rectangle with minimum perimeter enclosing a
     convex polygon
// Tf Ti Same
pair<Tf, Tf> rotatingCalipersBoundingBox(const
    Polygon &p) {
    using Linear::distancePointLine;
    int n = p.size();
    int l = 1, r = 1, j = 1;
    Tf area = 1e100;
```

```
    Tf perimeter = 1e100;
    for (int i = 0; i < n; i++) {
        Point v = (p[(i + 1) % n] - p[i]) / length(p
            [(i + 1) % n] - p[i]);
        while (dcmp(dot(v, p[r % n] - p[i]) - dot(v,
             p[(r + 1) % n] - p[i])) < 0)
            r++;
        while (j < r || dcmp(cross(v, p[j % n] - p[i
            ]) -
                                cross(v, p[(j + 1) % n] - p[i])) < 0)
            j++;
        while (l < j ||
                   dcmp(dot(v, p[l % n] - p[i]) -
                        dot(v, p[(l + 1) % n] - p[i
                        ])) > 0)
            l++;
        Tf w = dot(v, p[r % n] - p[i]) - dot(v, p[l
            % n] - p[i]);
        Tf h = distancePointLine(p[j % n], Line(p[i
            ], p[(i + 1) % n]));
        area = min(area, w * h);
        perimeter = min(perimeter, 2 * w + 2 * h);
    }
    return make_pair(area, perimeter);
}
// returns the left side of polygon u after cutting
     it by ray a->b
Polygon cutPolygon(Polygon u, Point a, Point b) {
    using Linear::lineLineIntersection;
    using Linear::onSegment;

    Polygon ret;
    int n = u.size();
    for (int i = 0; i < n; i++) {
        Point c = u[i], d = u[(i + 1) % n];
        if (dcmp(cross(b - a, c - a)) >= 0) ret.
            push_back(c);
        if (dcmp(cross(b - a, d - c)) != 0) {
            Point t;
            lineLineIntersection(a, b - a, c, d - c,
                 t);
            if (onSegment(t, Segment(c, d))) ret.
                push_back(t);
        }
    }
    return ret;
}
// returns true if point p is in or on triangle abc
bool pointInTriangle(Point a, Point b, Point c,
    Point p) {
    return dcmp(cross(b - a, p - a)) >= 0 && dcmp(
        cross(c - b, p - b)) >= 0 &&
                 dcmp(cross(a - c, p - c)) >= 0;
}
// pt must be in ccw order with no three collinear
    points
// returns inside = -1, on = 0, outside = 1
int pointInConvexPolygon(const Polygon &pt, Point p
    ) {
    int n = pt.size();
    assert(n >= 3);

    int lo = 1, hi = n - 1;
    while (hi - lo > 1) {
        int mid = (lo + hi) / 2;
        if (dcmp(cross(pt[mid] - pt[0], p - pt[0]))
             > 0)
            lo = mid;
        else
            hi = mid;
    }

    bool in = pointInTriangle(pt[0], pt[lo], pt[hi
        ], p);
    if (!in) return 1;

    if (dcmp(cross(pt[lo] - pt[lo - 1], p - pt[lo -
         1])) == 0) return 0;
    if (dcmp(cross(pt[hi] - pt[lo], p - pt[lo])) ==
         0) return 0;
    if (dcmp(cross(pt[hi] - pt[(hi + 1) % n], p -
        pt[(hi + 1) % n])) == 0)
        return 0;
    return -1;
}
// Extreme Point for a direction is the farthest
    point in that direction
// u is the direction for extremeness
int extremePoint(const Polygon &poly, Point u) {
    int n = (int)poly.size();
    int a = 0, b = n;
    while (b - a > 1) {
        int c = (a + b) / 2;
        if (dcmp(dot(poly[c] - poly[(c + 1) % n], u)
            ) >= 0 &&
                dcmp(dot(poly[c] - poly[(c - 1 + n)
                    % n], u)) >= 0) {
            return c;
        }

        bool a_up = dcmp(dot(poly[(a + 1) % n] -
            poly[a], u)) >= 0;
        bool c_up = dcmp(dot(poly[(c + 1) % n] -
            poly[c], u)) >= 0;
        bool a_above_c = dcmp(dot(poly[a] - poly[c],
             u)) > 0;

        if (a_up && !c_up)
            b = c;
        else if (!a_up && c_up)
            a = c;
        else if (a_up && c_up) {
            if (a_above_c)
                b = c;
            else
                a = c;
        } else {
            if (!a_above_c)
                b = c;
            else
                a = c;
        }
```

```
    }

    if (dcmp(dot(poly[a] - poly[(a + 1) % n], u)) >
        0 &&
          dcmp(dot(poly[a] - poly[(a - 1 + n) % n
             ], u)) > 0)
        return a;
    return b % n;
}
// For a convex polygon p and a line l, returns a
    list of segments
// of p that touch or intersect line l.
// the i'th segment is considered (p[i], p[(i + 1)
    modulo |p|])
// #1 If a segment is collinear with the line, only
     that is returned
// #2 Else if l goes through i'th point, the i'th
    segment is added
// Complexity: O(lg |p|)
vector<int> lineConvexPolyIntersection(const
    Polygon &p, Line l) {
    assert((int)p.size() >= 3);
    assert(l.a != l.b);

    int n = p.size();
    vector<int> ret;

    Point v = l.b - l.a;
    int lf = extremePoint(p, rotate90(v));
    int rt = extremePoint(p, rotate90(v) * Ti(-1));
    int olf = orient(l.a, l.b, p[lf]);
    int ort = orient(l.a, l.b, p[rt]);

    if (!olf || !ort) {
        int idx = (!olf ? lf : rt);
        if (orient(l.a, l.b, p[(idx - 1 + n) % n])
            == 0)
            ret.push_back((idx - 1 + n) % n);
        else
            ret.push_back(idx);
        return ret;
    }
    if (olf == ort) return ret;

    for (int i = 0; i < 2; ++i) {
        int lo = i ? rt : lf;
        int hi = i ? lf : rt;
        int olo = i ? ort : olf;

        while (true) {
            int gap = (hi - lo + n) % n;
            if (gap < 2) break;

            int mid = (lo + gap / 2) % n;
            int omid = orient(l.a, l.b, p[mid]);
            if (!omid) {
                lo = mid;
                break;
            }
            if (omid == olo)
                lo = mid;
            else
                hi = mid;
        }
        ret.push_back(lo);
    }
    return ret;
}
// Calculate [ACW, CW] tangent pair from an
    external point
constexpr int CW = -1, ACW = 1;
bool isGood(Point u, Point v, Point Q, int dir) {
    return orient(Q, u, v) != -dir;
}
Point better(Point u, Point v, Point Q, int dir) {
    return orient(Q, u, v) == dir ? u : v;
}
Point pointPolyTangent(const Polygon &pt, Point Q,
    int dir, int lo, int hi) {
    while (hi - lo > 1) {
        int mid = (lo + hi) / 2;
        bool pvs = isGood(pt[mid], pt[mid - 1], Q,
            dir);
        bool nxt = isGood(pt[mid], pt[mid + 1], Q,
            dir);

        if (pvs && nxt) return pt[mid];
        if (!(pvs || nxt)) {
            Point p1 = pointPolyTangent(pt, Q, dir,
                mid + 1, hi);
            Point p2 = pointPolyTangent(pt, Q, dir,
                lo, mid - 1);
            return better(p1, p2, Q, dir);
        }

        if (!pvs) {
            if (orient(Q, pt[mid], pt[lo]) == dir)
                hi = mid - 1;
            else if (better(pt[lo], pt[hi], Q, dir)
                 == pt[lo])
                hi = mid - 1;
            else
                lo = mid + 1;
        }
        if (!nxt) {
            if (orient(Q, pt[mid], pt[lo]) == dir)
                lo = mid + 1;
            else if (better(pt[lo], pt[hi], Q, dir)
                 == pt[lo])
                hi = mid - 1;
            else
                lo = mid + 1;
        }
    }

    Point ret = pt[lo];
    for (int i = lo + 1; i <= hi; i++) ret = better
        (ret, pt[i], Q, dir);
    return ret;
}
// [ACW, CW] Tangent
pair<Point, Point> pointPolyTangents(const Polygon
    &pt, Point Q) {
    int n = pt.size();
    Point acw_tan = pointPolyTangent(pt, Q, ACW, 0,
        n - 1);
    Point cw_tan = pointPolyTangent(pt, Q, CW, 0, n
        - 1);
    return make_pair(acw_tan, cw_tan);
}
```

7.5 Polygon

```
typedef vector<Point> Polygon;
// removes redundant colinear points
// polygon can't be all colinear points
Polygon RemoveCollinear(const Polygon &poly) {
    Polygon ret;
    int n = poly.size();
```

```
    for (int i = 0; i < n; i++) {
        Point a = poly[i];
        Point b = poly[(i + 1) % n];
        Point c = poly[(i + 2) % n];
        if (dcmp(cross(b - a, c - b)) != 0 && (ret.
            empty() || b != ret.back()))
            ret.push_back(b);
    }
    return ret;
}
// returns the signed area of polygon p of n
    vertices
Tf signedPolygonArea(const Polygon &p) {
    Tf ret = 0;
    for (int i = 0; i < (int)p.size() - 1; i++)
        ret += cross(p[i] - p[0], p[i + 1] - p[0]);
    return ret / 2;
}
// given a polygon p of n vertices, generates the
    convex hull in in CCW
// Tested on https://acm.timus.ru/problem.aspx?
    space=1&num=1185
// Caution: when all points are colinear AND
    removeRedundant == false
// output will be contain duplicate points (from
    upper hull) at back
Polygon convexHull(Polygon p, bool removeRedundant)
    {
    int check = removeRedundant ? 0 : -1;
    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end()), p.end());

    int n = p.size();
    Polygon ch(n + n);
    int m = 0; // preparing lower hull
    for (int i = 0; i < n; i++) {
        while (m > 1 &&
                    dcmp(cross(ch[m - 1] - ch[m -
                        2], p[i] - ch[m - 1])) <=
                        check)
            m--;
        ch[m++] = p[i];
    }
    int k = m; // preparing upper hull
    for (int i = n - 2; i >= 0; i--) {
        while (m > k &&
                    dcmp(cross(ch[m - 1] - ch[m -
                        2], p[i] - ch[m - 2])) <=
                        check)
            m--;
        ch[m++] = p[i];
    }
    if (n > 1) m--;
    ch.resize(m);
    return ch;
}
// returns inside = -1, on = 0, outside = 1
int pointInPolygon(const Polygon &p, Point o) {
    using Linear::onSegment;
    int wn = 0, n = p.size();
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        if (onSegment(o, Segment(p[i], p[j])) || o
            == p[i]) return 0;
        int k = dcmp(cross(p[j] - p[i], o - p[i]));
        int d1 = dcmp(p[i].y - o.y);
        int d2 = dcmp(p[j].y - o.y);
        if (k > 0 && d1 <= 0 && d2 > 0) wn++;
        if (k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    return wn ? -1 : 1;
}
// Given a simple polygon p, and a line l, returns
    (x, y)
// x = longest segment of l in p, y = total length
    of l in p.
pair<Tf, Tf> linePolygonIntersection(Line l, const
    Polygon &p) {
    using Linear::lineLineIntersection;
    int n = p.size();
    vector<pair<Tf, int>> ev;
    for (int i = 0; i < n; ++i) {
        Point a = p[i], b = p[(i + 1) % n], z = p[(i
             - 1 + n) % n];
        int ora = orient(l.a, l.b, a), orb = orient(
            l.a, l.b, b),
                orz = orient(l.a, l.b, z);
        if (!ora) {
            Tf d = dot(a - l.a, l.b - l.a);
            if (orz && orb) {
                if (orz != orb) ev.emplace_back(d,
                    0);
                // else // Point Touch
            } else if (orz)
                ev.emplace_back(d, orz);
            else if (orb)
                ev.emplace_back(d, orb);
        } else if (ora == -orb) {
            Point ins;
            lineLineIntersection(l, Line(a, b), ins)
                ;
            ev.emplace_back(dot(ins - l.a, l.b - l.a
                ), 0);
        }
    }
    sort(ev.begin(), ev.end());

    Tf ans = 0, len = 0, last = 0, tot = 0;
    bool active = false;
    int sign = 0;
    for (auto &qq : ev) {
        int tp = qq.second;
        Tf d = qq.first; /// current Segment is (
            last, d)
        if (sign) {     /// On Border
            len += d - last;
            tot += d - last;
            ans = max(ans, len);
            if (tp != sign) active = !active;
            sign = 0;
        } else {
            if (active) { /// Strictly Inside
                len += d - last;
                tot += d - last;
                ans = max(ans, len);
            }
            if (tp == 0)
                active = !active;
            else
                sign = tp;
        }
        last = d;
        if (!active) len = 0;
    }
    ans /= length(l.b - l.a);
    tot /= length(l.b - l.a);
    return {ans, tot};
}
```

7.6 Half Plane

```
using Linear::lineLineIntersection;
struct DirLine {
    Point p, v;
    Tf ang;
    DirLine() {}
    /// Directed line containing point P in the direction v
    DirLine(Point p, Point v) : p(p), v(v) { ang = atan2(v.y, v.x); }
    bool operator<(const DirLine& u) const { return ang < u.ang; }
};
// returns true if point p is on the ccw-left side of ray l
bool onLeft(DirLine l, Point p) { return dcmp(cross(l.v, p - l.p)) >= 0; }

// Given a set of directed lines returns a polygon such that
// the polygon is the intersection by halfplanes created by the
// left side of the directed lines. MAY CONTAIN DUPLICATE POINTS
int halfPlaneIntersection(vector<DirLine>& li, Polygon& poly) {
    int n = li.size();
    sort(li.begin(), li.end());

    int first, last;
    Point* p = new Point[n];
    DirLine* q = new DirLine[n];
    q[first = last = 0] = li[0];

    for (int i = 1; i < n; i++) {
        while (first < last && !onLeft(li[i], p[last - 1])) last--;
        while (first < last && !onLeft(li[i], p[first])) first++;
        q[++last] = li[i];

        if (dcmp(cross(q[last].v, q[last - 1].v)) == 0) {
            last--;
            if (onLeft(q[last], li[i].p)) q[last] = li[i];
        }

        if (first < last)
            lineLineIntersection(q[last - 1].p, q[last - 1].v, q[last].p, q[last].v, p[last - 1]);
    }

    while (first < last && !onLeft(q[first], p[last - 1])) last--;
    if (last - first <= 1) {
        delete[] p;
        delete[] q;
        poly.clear();
        return 0;
    }
    lineLineIntersection(q[last].p, q[last].v, q[first].p, q[first].v, p[last]);

    int m = 0;
    poly.resize(last - first + 1);
    for (int i = first; i <= last; i++) poly[m++] = p[i];
    delete[] p;
    delete[] q;
    return m;
}
```

8 Equations and Formulas

8.1 Catalan Numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} \quad C_0 = 1, C_1 = 1 \text{ and } C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

The number of ways to completely parenthesize n+1 factors.
The number of triangulations of a convex polygon with n+2 sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
The number of ways to connect the $2n$ points on a circle to form n disjoint i.e. non-intersecting chords.
The number of rooted full binary trees with n+1 leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
Number of permutations of $1,,n$ that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing sub-sequence. For $n = 3$, these permutations are 132, 213, 231, 312 and 321.

8.2 Stirling Numbers First Kind

The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).
$S(n,k)$ counts the number of permutations of n elements with k disjoint cycles.
$S(n,k) = (n-1)\cdot S(n-1,k) + S(n-1,k-1),\ where,\ S(0,0) = 1, S(n,0) = S(0,n) = 0 \ \sum_{k=0}^{n} S(n,k) = n!$

The unsigned Stirling numbers may also be defined algebraically, as the coefficient of the rising factorial:

$$x^{\bar{n}} = x(x+1)...(x+n-1) = \sum_{k=0}^{n} S(n,k)x^k$$

Lets $[n,k]$ be the stirling number of the first kind, then

$$\left[{n \atop n-k}\right] = \sum_{0\le i_1<i_2<i_k<n} i_1 i_2 .... i_k.$$

8.3 Stirling Numbers Second Kind

Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets.
$S(n,k) = k\cdot S(n-1,k) + S(n-1,k-1),\ where\ S(0,0) = 1, S(n,0) = S(0,n) = 0$ $S(n,2) = 2^{n-1}-1$ $S(n,k)\cdot k! =$ number of ways to color n nodes using colors from 1 to k such that each color is used at least once.
An r-associated Stirling number of the second kind is the number of ways to partition a set of n objects into k subsets, with each subset containing at least r elements. It is denoted by $S_r(n,k)$ and obeys the recurrence relation. $S_r(n+1,k) = kS_r(n,k) + \binom{n}{r-1}S_r(n-r+1,k-1)$
Denote the n objects to partition by the integers $1,2,.,n$. Define the reduced Stirling numbers of the second kind, denoted $S^d(n,k)$, to be the number of ways to partition the integers $1,2,.,n$ into k nonempty subsets such that all elements in each subset have pairwise distance at least d. That is, for any integers i and j in a given subset, it is required that $|i-j| \ge d$. It has been shown that these numbers satisfy, $S^d(n,k) = S(n-d+1,k-d+1), n \ge k \ge d$

8.4 Other Combinatorial Identities

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$$

$$\sum_{i=0}^{k}\binom{n+i}{i} = \sum_{i=0}^{k}\binom{n+i}{n} = \binom{n+k+1}{k}$$

$$n,r \in N, n > r, \sum_{i=r}^{n}\binom{i}{r} = \binom{n+1}{r+1}$$

If $P(n) = \sum_{k=0}^{n}\binom{n}{k}\cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n}(-1)^{n-k}\binom{n}{k}\cdot P(k)$$

If $P(n) = \sum_{k=0}^{n}(-1)^k\binom{n}{k}\cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n}(-1)^k\binom{n}{k}\cdot P(k)$$

8.5 Different Math Formulas

Picks Theorem : $A = i + b/2 - 1$
Deragements : $d(i) = (i-1)\times(d(i-1)+d(i-2))$

$$\frac{n}{ab} - \left\{\frac{b'n}{a}\right\} - \left\{\frac{a'n}{b}\right\} + 1$$

8.6 GCD and LCM

if m is any integer, then $\gcd(a+m\cdot b, b) = \gcd(a,b)$
The gcd is a multiplicative function in the following sense: if a_1 and a_2 are relatively prime, then $\gcd(a_1\cdot a_2, b) = \gcd(a_1,b)\cdot\gcd(a_2,b)$.
$\gcd(a, \operatorname{lcm}(b,c)) = \operatorname{lcm}(\gcd(a,b), \gcd(a,c))$.
$\operatorname{lcm}(a, \gcd(b,c)) = \gcd(\operatorname{lcm}(a,b), \operatorname{lcm}(a,c))$.
For non-negative integers a and b, where a and b are not both zero, $\gcd(n^a-1, n^b-1) = n^{\gcd(a,b)}-1$

$$\gcd(a,b) = \sum_{k|a \text{ and } k|b}\phi(k)$$

$$\sum_{i=1}^{n}[\gcd(i,n)=k] = \phi\left(\frac{n}{k}\right)$$

$$\sum_{k=1}^{n}\gcd(k,n) = \sum_{d|n} d\cdot\phi\left(\frac{n}{d}\right)$$

$$\sum_{k=1}^{n}x^{\gcd(k,n)} = \sum_{d|n}x^d\cdot\phi\left(\frac{n}{d}\right)$$

$$\sum_{k=1}^{n}\frac{1}{\gcd(k,n)} = \sum_{d|n}\frac{1}{d}\cdot\phi\left(\frac{n}{d}\right) = \frac{1}{n}\sum_{d|n}d\cdot\phi(d)$$

$$\sum_{k=1}^{n}\frac{k}{\gcd(k,n)} = \frac{n}{2}\cdot\sum_{d|n}\frac{1}{d}\cdot\phi\left(\frac{n}{d}\right) = \frac{n}{2}\cdot\frac{1}{n}\cdot\sum_{d|n}d\cdot\phi(d)$$

$$\sum_{k=1}^{n}\frac{n}{\gcd(k,n)} = 2*\sum_{k=1}^{n}\frac{k}{\gcd(k,n)} - 1, \text{ for } n > 1$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}[\gcd(i,j)=1] = \sum_{d=1}^{n}\mu(d)\lfloor\frac{n}{d}\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\gcd(i,j) = \sum_{d=1}^{n}\phi(d)\lfloor\frac{n}{d}\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}i\cdot j[\gcd(i,j)=1] = \sum_{i=1}^{n}\phi(i)i^2$$

$$F(n) = \sum_{i=1}^{n}\sum_{j=1}^{n}\operatorname{lcm}(i,j) = \sum_{l=1}^{n}\left(\frac{\left(1+\lfloor\frac{n}{l}\rfloor\right)\left(\lfloor\frac{n}{l}\rfloor\right)}{2}\right)^2\sum_{d|l}\mu(d)ld$$

8.7 Geometry

Cone: $V = \frac{1}{3}\pi r^2 h$, $A = \pi r(r+\sqrt{h^2+r^2})$
Pyramid: $V = \frac{1}{3}\times$ base $\times$ height, $A =$ base area $+ \frac{1}{2}\times$ perimeter $\times$ slant height
Triangular Prism: $V = \frac{1}{2}\times$ base $\times$ height $\times$ depth, $A =$ base $\times$ height $+ 3\times\left(\frac{1}{2}\times\text{side}\times\text{perimeter}\right)$
Torus: $V = 2\pi^2 Rr^2$, $A = 4\pi^2 Rr$
Ellipsoid: $V = \frac{4}{3}\pi abc$, $A = 4\pi\left(\frac{(ab)^{1.6}+(bc)^{1.6}+(ca)^{1.6}}{3}\right)^{1/1.6}$