Serso Library

Table of Contents

DP

Convex Hull

CHT_maxline.cpp

```
/*
 * Description: Container where you can add lines of the form kx+m, and query minimum values at points x.
 * Can be applied on:
   - monotonic slopes, monotonic queries
   - monotonic slopes, random queries

  Here's monotonic can be (ASCE , DESC)
 */
const ll INF = LLONG_MAX;
const ll NEG_INF = LLONG_MIN;

struct mxCHT {  // Convex Hull Trick for Maximum
    deque<long long> A; // Slopes
    deque<long long> B; // Intercepts

    // Modified 'bad' function with reversed inequality for Maximum CHT
    bool bad(int l1, int l2, int l3) {
        return (B[l3] - B[l1]) * (long double)(A[l1] - A[l2])
                >= (B[l2] - B[l1]) * (long double)(A[l1] - A[l3]);  // Changed <= to >=
    }

    // Adds a line with slope `a` and intercept `b` when slopes are in ascending order
    void addASC(long long a, long long b) {
        A.push_back(a);
        B.push_back(b);
        // Remove the second last line if it's unnecessary
        while (A.size() >= 3 && bad(A.size() - 3, A.size() - 2, A.size() - 1)) {
            A.erase(A.end() - 2);
            B.erase(B.end() - 2);
        }
    }

    // Adds a line with slope `a` and intercept `b` when slopes are in descending order
    void addDESC(long long a, long long b) {
        A.push_front(a);
        B.push_front(b);
        // Remove the second line if it's unnecessary
        while (A.size() >= 3 && bad(0, 1, 2)) {
            A.erase(A.begin() + 1);
            B.erase(B.begin() + 1);
        }
    }
```

```

// Evaluates the line at index `l` for a given `x`
long long f(int l, long long x) {
    return A[l] * x + B[l];
}

// Queries the convex hull for maximum value at `x` when x-values are increasing
long long queryASC(long long x) {
    while (A.size() >= 2 && f(0, x) < f(1, x)) {  // Changed '>' to '<'
        // Remove the front line if it's not optimal
        A.pop_front();
        B.pop_front();
    }
    if (A.empty())
        return NEG_INF;  // Changed from LLONG_MAX to LLONG_MIN
    return f(0, x);
}

// Queries the convex hull for maximum value at `x` when x-values are decreasing
long long queryDESC(long long x) {
    while (A.size() >= 2 && f(A.size() - 1, x) < f(A.size() - 2, x)) {  // Changed '>' to '<'
        // Remove the back line if it's not optimal
        A.pop_back();
        B.pop_back();
    }
    if (A.empty())
        return NEG_INF;  // Changed from LLONG_MAX to LLONG_MIN
    return f(A.size() - 1, x);
}

// Optional: Query for random x-values using binary search
long long query(long long x) {
    int lo = 0, hi = A.size() - 1;
    long long res = NEG_INF;  // Changed from LLONG_MAX to LLONG_MIN
    while (lo <= hi) {
        int mid = (lo + hi) / 2;
        long long val = f(mid, x);
        res = max(res, val);  // Changed from min to max
        if (mid + 1 < A.size() && f(mid + 1, x) > val) {  // Changed '<=' to '>'
            lo = mid + 1;
        }
        else if (mid - 1 >= 0 && f(mid - 1, x) > val) {  // Changed '<=' to '>'
            hi = mid - 1;
        }
```

```
            else {
                break;
            }
        }
        return res;
    }
```

CHT_minline.cpp

```
/*
 * Description: Container where you can add lines of the form kx+m, and query minimum values at points x.
 * Can be applied on:
    - monotonic slopes, monotonic queries
    - monotonic slopes, random queries

  Here's monotonic can be (ASCE , DESC)
 */
struct mnCHT {  // Convex Hull Trick for Minimum
    deque<long long> A; // Slopes
    deque<long long> B; // Intercepts


    bool bad(int l1, int l2, int l3) {
        return (B[l3] - B[l1]) * (long double) (A[l1] - A[l2])
                <= (B[l2] - B[l1]) * (long double) (A[l1] - A[l3]);
    }

    // Adds a line with slope `a` and intercept `b` when slopes are in ascending order
    void addASC(long long a, long long b) {
        A.push_back(a);
        B.push_back(b);
        // Remove the second last line if it's unnecessary
        while (A.size() >= 3 && bad(A.size() - 3, A.size() - 2, A.size() - 1)) {
            A.erase(A.end() - 2);
            B.erase(B.end() - 2);
        }
    }

    // Adds a line with slope `a` and intercept `b` when slopes are in descending order
    void addDESC(long long a, long long b) {
        A.push_front(a);
    B.push_front(b);
    // Remove the second line if it's unnecessary
    while (A.size() >= 3 && bad(0, 1, 2)) {
      A.erase(A.begin() + 1);
      B.erase(B.begin() + 1);
    }
  }

  // Evaluates the line at index `l` for a given `x`
  long long f(int l, long long x) {
    return A[l] * x + B[l];
  }

  // Queries the convex hull for minimum value at `x` when x-values are increasing
  long long queryASC(long long x) {
    while (A.size() >= 2 && f(0, x) > f(1, x)) {
      // Remove the front line if it's not optimal
      A.pop_front();
      B.pop_front();
    }
    if (A.empty())
      return LLONG_MAX;  // Return maximum value if no lines are left
    return f(0, x);
  }
  // Queries the convex hull for minimum value at `x` when x-values are decreasing
  long long queryDESC(long long x) {
    while (A.size() >= 2 && f(A.size() - 1, x) > f(A.size() - 2, x)) {
      // Remove the back line if it's not optimal
      A.pop_back();
      B.pop_back();
    }
    if (A.empty())
      return LLONG_MAX;  // Return maximum value if no lines are left
    return f(A.size() - 1, x);
  }

  // Optional: Query for random x-values using binary search
  long long query(long long x) {
    int lo = 0, hi = A.size() - 1;
    long long res = LLONG_MAX;
    while (lo <= hi) {
      int mid = (lo + hi) / 2;
      long long val = f(mid, x);
      res = min(res, val);
      if (mid + 1 < A.size() && f(mid + 1, x) <= val) {
        lo = mid + 1;
      } else if (mid - 1 >= 0 && f(mid - 1, x) <= val) {
        hi = mid - 1;
      } else {
        break;
      }
    }
    return res;
  }
```

LineContainerMax_double.cpp

```
struct Line {
  mutable double k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(double x) const { return p < x; }
};

struct LineContainerMax : multiset<Line, less<>> {
  static constexpr double inf = numeric_limits<double>::infinity();
  static constexpr double EPS = 1e-9;

  double div(double a, double b) {
    return a / b;
  }

  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (abs(x->k - y->k) < EPS) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }

  void add(double k, double m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }

```

```cpp
  double query(double x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

LineContainerMin_double.cpp

```cpp
struct Line {
  mutable double k, m, p;
  bool operator<(const Line& o) const { return k > o.k; }
  bool operator<(double x) const { return p < x; }
};

struct LineContainerMin : multiset<Line, less<>> {
  static constexpr  double inf = numeric_limits<double>::infinity();
  static constexpr double EPS = 1e-9;
  double div(double a, double b) {
    return a / b;
  }

  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (abs(x->k - y->k) < EPS) x->p = x->m < y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }

  void add(double k, double m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }

  double query(double x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

LineContainer_minLine.cpp

```
/*
 * Description: Container where you can add lines of the form kx+m, and query minimum values at points x.
 * Can be applied on:
   - monotonic slopes, monotonic queries
   - monotonic slopes, random queries
   - random slopes, random queries
 */
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k > o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainerMin : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m < y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

Linecontainer_maxLine.cpp

```cpp
/*
 * Description: Container where you can add lines of the form kx+m, and query maximum values at points x.
 * Can be applied on:
   - monotonic slopes, monotonic queries
   - monotonic slopes, random queries
   - random slopes, random queries
 */

struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

struct LineContainerMax : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

Divide and Conquer

DP_dnc.cpp

```cpp
const int N = 5002;
int n,k;
ll dp[2][N] , p[N] , c[N][N];
int cost(int l , int r){ return; } // cost function
void dnc(int ind , int l , int r , int optl , int optr){
  if(l > r) return;
  pair<ll,int> bst{LLONG_MAX,-1};
  int m = (l+r)/2;
  for(int opt = optl; opt <= optr; ++opt){
    /*
      In case 0-base
      bst = min(bst , { (opt ? dp[ind^1][opt-1] : 0) + c(opt,m) , opt });
    */
    bst = min(bst , {dp[ind^1][opt-1] + c(opt,m) , opt} );
  }
  dp[ind][m] = bst.f;
  int opt = bst.s;
  dnc(ind,l,m-1,optl,opt);
  dnc(ind,m+1,r,opt,optr);
}
void run(){
  for(int i = 1; i <= n;++i) dp[0][i] = c[1][i];
  for(int x = 1; x < k; ++x){
    dnc(x&1,1,n,1,n);
  }
}
```

Knapsack

knapsack_with_bitset.cpp

```cpp
/*
 optimized Knapsack problem work if
 n <= 2e5
 a1 + a2 + ... + an <= 1e6
 Time complexity: O(n * sqrt(n))
*/
```

```cpp
template<int len = 1>
int solve(const vector<int> &v , int all){
  if(v.empty()) return 0;
  if(len < all)
    return solve<min(2 * len , N)>(v, all);
  map<int,int> frq;
  int Max = 0;
  for(auto &i : v) frq[i]++ , Max = max(Max , i);
  // specific for a problem, think of something that can optimize the code.
  vector<int> cur;
  for(auto &[x , f] : frq){
    int need = 1;
    while(f){
      need = min(need , f);
      f -= need;
      cur.emplace_back(need * x);
      need *= 2;
    }
  }
  bitset<len> dp;
  dp[0] = 1;
  for(auto &i : cur) dp |= (dp << i);
  // specific for a problem.
}
```

optimized_knapsack.cpp

```cpp
/*
 Problem: For every possible sum , find the minimum number of elements that can form this sum.
 Time complexity: O(n sqrt(n))
*/
map<int,int> frq; // store the frq of every element in the array.
int dp[n + 1];
memset(dp , '?' , sizeof dp);
dp[0] = 0;
for(auto &[sz , f] : frq){
  int cost = 1;
  while(f){
    cost = min(cost , f);
    f -= cost;
    for(int sum = n; sum >= cost * sz; sum--)
```

```
            dp[sum] = min(dp[sum] , dp[sum - cost * sz] + cost);
        cost *= 2;
    }
}
```

Knuth's

DP_knuths_cpAlgo.cpp

```
int solve() {
    int N;
    ... // read N and input
    int dp[N][N], opt[N][N];

    auto C = [&](int i, int j) {
        ... // Implement cost function C.
    };

    for (int i = 0; i < N; i++) {
        opt[i][i] = i;
        ... // Initialize dp[i][i] according to the problem
    }

    for (int i = N-2; i >= 0; i--) {
        for (int j = i+1; j < N; j++) {
            int mn = INT_MAX;
            int cost = C(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] + cost) {
                    opt[i][j] = k;
                    mn = dp[i][k] + dp[k+1][j] + cost;
                }
            }
            dp[i][j] = mn;
        }
    }

    return dp[0][N-1];
}
```

Sum of subset

SOS_DP.cpp

```cpp
const int N = 20;
int dp[1<<N];
int comb(int x , int y){
    // depend on the problem
    return x+y;
}
void mask_to_supermask(){
    // Combing every msk to all its supermask
    for(int x = 0; x < N; ++x){
        for(int msk = 0; msk < (1<<N); msk++){
            if(msk>>x&1) dp[msk] = comb(dp[msk] , dp[msk^(1<<x)]);
        }
    }
}
void mask_to_submask(){
    //Combing every msk to all its submask
    for(int x = 0; x < N; ++x){
        for(int msk = (1<<N)-1 ; msk >= 0; msk--){
            if(msk>>x&1) dp[msk^(1<<x)] = comb(dp[msk^(1<<x)] , dp[msk]);
        }
    }
}
void supermask_to_mask(){
    // Combing every supermask to mask
    for(int x = 0; x < N; ++x){
        for(int msk = (1<<N) - 1; msk >= 0; --msk){
            if(!(msk>>x&1)) dp[msk] = comb(dp[msk] , dp[msk|(1<<x)]);
        }
    }
}
void submask_to_mask(){
    // combing every submask to mask
    for(int x = 0; x < N; ++x){
        for(int msk = 0; msk < N; ++msk){
            if(!(msk>>x&1)) dp[msk | (1<<x)] = comb(dp[msk | (1<<x)] , dp[ms]);
        }
    }
}
```

Zero matrix

largest_zero_submatrix.cpp

```cpp
/*
    You are given a matrix with n rows and m columns.
    Find the largest submatrix consisting of only zeros (a submatrix is a rectangular area of the matrix).
    Elements of the matrix will be a[i][j], where i = 0...n - 1, j = 0... m - 1.
    For simplicity, we will consider all non-zero elements equal to 1.

    Time Complexity: O(n x m)
    Space Complexity: O(m)
*/
int zero_matrix(vector<vector<int>> a) {
    int n = a.size();
    int m = a[0].size();

    int ans = 0;
    vector<int> d(m, -1), d1(m), d2(m);
    stack<int> st;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (a[i][j] == 1)
                d[j] = i;
        }

        for (int j = 0; j < m; ++j) {
            while (!st.empty() && d[st.top()] <= d[j])
                st.pop();
            d1[j] = st.empty() ? -1 : st.top();
            st.push(j);
        }
        while (!st.empty())
            st.pop();

        for (int j = m - 1; j >= 0; --j) {
            while (!st.empty() && d[st.top()] <= d[j])
                st.pop();
            d2[j] = st.empty() ? m : st.top();
            st.push(j);
        }
        while (!st.empty())
            st.pop();

        for (int j = 0; j < m; ++j)
            ans = max(ans, (i - d[j]) * (d2[j] - d1[j] - 1));
    }
    return ans;
}
```

Data Structure

2D Range Queries

2D_prefixsum.cpp

```
const int N = 500, M = 500;
int pref[N][M]; // 1-base
int n , m;
int qry(int from_x , int from_y , int to_x , int to_y){
    return  pref[to_x][to_y] - pref[from_x - 1][to_y] - pref[to_x][from_y - 1] + pref[from_x - 1][from_y - 1];
}
void upd(int from_x , int from_y, int to_x , int to_y, int val){
    pref[from_x][from_y] += val;
    pref[to_x + 1][from_y] -= val;
    pref[from_x][to_y + 1] -= val;
    pref[to_x + 1][to_y + 1] += val;
}
void build(){
    for(int i = 1; i <= n; ++i) for(int j = 1; j <= m; ++j){
        pref[i][j] += pref[i - 1][j] + pref[i][j - 1] - pref[i - 1][j - 1];
    }
}
void init(){
    for(int i = 1; i <= n; ++i) for(int j = 1; j <= m; ++j){
        pref[i][j] = 0;
    }
}
```

2d_sparsetable.cpp

```
/*
    2D sparse table
    - Memory allocation: O(nlog(n) x mlog(m))
```

```
  - range [l..r] , r: inclusive
  - Base: 0-index

  Function description:
    1. build(n , m , g): Build 2D sparse table of g[n][m]
    2. qry(lx , rx , ly , ry): find (min,max,sum,produc) of rectangle [lx...rx] x [ly...ry]
      Time complexity: O(1)
*/
#define vi vector<int>
#define vi2 vector<vi>
#define vi3 vector<vi2>
#define vi4 vector<vi3>
#define rep(i , st , ed) for(int i = st; i < ed; i++)
struct sparse_2d{
  vi4 f;
  void init(int n , int m , vi2 &g){
    int lgN = 31 - __builtin_clz(n + 1);
    int lgM = 31 - __builtin_clz(m + 1);
    f = vi4(n , vi3(m , vi2(lgN , vi(lgM))));
    rep(i , 0 , n) rep(j , 0 , m) f[i][j][0][0] = g[i][j];

    rep(k1 , 0 , lgN) rep(k2 , 0 , lgM) if(k1 || k2){
      rep(i , 0 , n - (1 << k1) + 1) rep(j , 0 , m - (1 << k2) + 1){
        if (k1 > 0) {
          f[i][j][k1][k2] = comb(f[i][j][k1 - 1][k2], f[i + (1 << (k1 - 1))][j][k1 - 1][k2]);
        }
        if (k2 > 0) {
          f[i][j][k1][k2] = comb(f[i][j][k1][k2 - 1], f[i][j + (1 << (k2 - 1))][k1][k2 - 1]);
        }
        if (k1 > 0 && k2 > 0) {
          f[i][j][k1][k2] = comb(comb(
              f[i][j][k1 - 1][k2 - 1],
              f[i + (1 << (k1 - 1))][j][k1 - 1][k2 - 1]), comb(
              f[i][j + (1 << (k2 - 1))][k1 - 1][k2 - 1],
              f[i + (1 << (k1 - 1))][j + (1 << (k2 - 1))][k1 - 1][k2 - 1]
              )
          );
        }
      }
    }
  }
  int comb(int a, int b){
    return __gcd(a , b); // custom operator
  }
  int qry(int l, int d, int r, int u) {
    int k1 = 31 - __builtin_clz(r - l + 1);
    int k2 = 31 - __builtin_clz(u - d + 1);
    return comb(comb(
                f[l][d][k1][k2],
                f[r - (1 << k1) + 1][d][k1][k2]),
            comb(
                f[l][u - (1 << k2) + 1][k1][k2],
                f[r - (1 << k1) + 1][u - (1 << k2) + 1][k1][k2])
    );
  }
}
```

3D_prefixsum.cpp

```
const int N = 500, M = 500 , K = 500;
int P[N][M][K]; // 1-base
int n , m , k;
int qry(int from_x , int from_y , int from_z , int to_x , int to_y , int to_z){
  int result = P[to_x][to_y][to_z]
          - P[from_x-1][to_y][to_z]
          - P[to_x][from_y-1][to_z]
          - P[to_x][to_y][from_z-1]
          + P[from_x-1][from_y-1][to_z]
          + P[from_x-1][to_y][from_z-1]
          + P[to_x][from_y-1][from_z-1]
          - P[from_x-1][from_y-1][from_z-1];
}
void upd(int from_x , int from_y, int from_z, int to_x , int to_y, int to_z , int val){
  P[from_x][from_y][from_z] += val;
  P[to_x + 1][from_y][from_z] -= val;
  P[from_x][to_y + 1][from_z] -= val;
  P[to_x][from_y][to_z + 1] -= val;
  P[to_x + 1][to_y + 1][from_z] += val;
  P[to_x + 1][from_y][to_z + 1] += val;
  P[from_x][to_y + 1][to_z + 1] += val;
  P[to_x + 1][to_y + 1][to_z + 1] -= val;
}
void build(vector<vector<vector<int>>> &A){
  for (int x = 1; x <= N; ++x) {
    for (int y = 1; y <= N; ++y) {
      for (int z = 1; z <= N; ++z) {
        P[x][y][z] = A[x][y][z]
```

```
                          + P[x-1][y][z]
                          + P[x][y-1][z]
                          + P[x][y][z-1]
                          - P[x-1][y-1][z]
                          - P[x-1][y][z-1]
                          - P[x][y-1][z-1]
                          + P[x-1][y-1][z-1];
            }
        }
    }
}
void init(){
    for(int i = 1; i <= n; ++i) for(int j = 1; j <= m; ++j) for(int k = 1; k <= K; ++k){
        P[i][j][k] = 0;
    }
}
```

BIT2D.cpp

```
/*
    --------------BIT2D-----------------
    - Memory allocation: O(n x m)
    - range [l..r] , r: inclusive
    - Base: 0-index

    Function description:
    1. init(n , m): initial a BIT with 2D grid[n][m]
       Time complexity: O(n x 4)
    2. upd(x , y , val): add to a cell(x ,y) value "val"
       Time complexity: O( log(n) x log(m) ) , here log is small constanst
    2. qry(lx , rx , ly , ry): find (sum,xor) of rectangle [lx...rx] x [ly...ry]
       Time complexity: O(log(n) x log(m))

*/
template<class T = int>
struct BIT2D {
private:
    vector<vector<T>> tree;
    int n , m;
    T sum(int x, int y) {
        T ret = 0;
        for (int i = x + 1; i; i -= (i & (-i))) {
            for (int j = y + 1; j; j -= (j & (-j))) {
```

```cpp
                ret += tree[i - 1][j - 1];
            }
        }
        return ret;
    }
public:
    void init(int n , int m) {
        this->n = n; this->m = m;
        tree.assign(n , vector<T>(m , 0));
    }
    void upd(int x, int y, T val) {
        for (int i = x + 1; i <= n; i += (i & (-i))) {
            for (int j = y + 1; j <= m; j += (j & (-j))) {
                tree[i - 1][j - 1] += val;
            }
        }
    }
    T qry(int sx, int sy, int ex, int ey) {
        return sum(ex, ey) - sum(ex, sy - 1) - sum(sx - 1, ey) + sum(sx - 1, sy - 1);
    }
    T qry(int x , int y){ return sum(x , y , x , y); }
};
```

BIT2D_nlogn2.cpp

```cpp
// O(N(logN)^2)
template<class T = int>
struct Bit2D {
    vector<T> ord;
    vector<vector<T>> fw, coord;

    // pts needs all points that will be used in the upd
    // if range upds remember to build with {x1, y1}, {x1, y2 + 1}, {x2 + 1, y1}, {x2 + 1, y2 + 1}
    Bit2D(vector<pair<T, T>> pts) {
        sort(pts.begin(), pts.end());
        for (auto a : pts)
            if (ord.empty() || a.first != ord.back())
                ord.push_back(a.first);
        fw.resize(ord.size() + 1);
        coord.resize(fw.size());

        for (auto& a : pts)
```

```
            swap(a.first, a.second);
        sort(pts.begin(), pts.end());
        for (auto& a : pts) {
            swap(a.first, a.second);
            for (int on = std::upper_bound(ord.begin(), ord.end(), a.first) - ord.begin(); on < fw.size(); on += on & -on)
                if (coord[on].empty() || coord[on].back() != a.second)
                    coord[on].push_back(a.second);
        }

        for (int i = 0; i < fw.size(); i++)
            fw[i].assign(coord[i].size() + 1, 0);
    }
    T merge(T a , T b){
        return a + b;
    }
    // point upd
    void upd(T x, T y, T v) {
        for (int xx = upper_bound(ord.begin(), ord.end(), x) - ord.begin(); xx < fw.size(); xx += xx & -xx)
            for (int yy = upper_bound(coord[xx].begin(), coord[xx].end(), y) - coord[xx].begin(); yy < fw[xx].size(); yy += yy & -yy)
                fw[xx][yy] = merge(fw[xx][yy], v);
    }

    // point qry
    T qry(T x, T y) {
        T ans = 0;
        for (int xx = upper_bound(ord.begin(), ord.end(), x) - ord.begin(); xx > 0; xx -= xx & -xx)
            for (int yy = upper_bound(coord[xx].begin(), coord[xx].end(), y) - coord[xx].begin(); yy > 0; yy -= yy & -yy)
                ans = merge(ans, fw[xx][yy]);
        return ans;
    }

    // range qry
    T qry(T x1, T y1, T x2, T y2) {
        return qry(x2, y2) - qry(x2, y1 - 1) - qry(x1 - 1, y2) + qry(x1 - 1, y1 - 1);
    }

    // range upd
    void upd(T x1, T y1, T x2, T y2, T v) {
        upd(x1, y1, v);
        upd(x1, y2 + 1, -v);
        upd(x2 + 1, y1, -v);
        upd(x2 + 1, y2 + 1, v);
    }
```

segTree_2d.cpp

```
/*
    2D segment Tree
    - Memory allocation: O(4n x 4m)
    - range [l..r] , r: inclusive
    - Base: 0-index

    Function description:
        1. upd(r , c , val): update cell (r,c) with value val
            Time complexity: O(log(n) x log(m))
        2. qry(lx , rx , ly , ry): find (min,max,sum,produc) of rectangle [lx...rx] x [ly...ry]
            Time complexity: O(log(n) x log(m))
        3. init(n , m): initial the segment tree
        4. build(g): build the segment tree with grid[n][m]
            Time complexity: O(n log(n) x m log(m))
*/
struct Node{
    int v;
    Node(){ v = 0;}
    Node(int x){ this->v = x; }
    Node operator +(const Node &other) const{
        Node res;
        res.v = __gcd(v , other.v); // custom operator
        return res;
    }
};
struct segTree_2d{
private:
    vector<vector<Node>> t;
    int n , m;
    void build_y(int vx, int lx, vector<vector<int>> &g ,int rx, int vy, int ly, int ry) {
        if (ly == ry) {
            if (lx == rx){
                t[vx][vy] = Node(g[lx][ly]);
            }
            else{
```

```
            t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
        }
    } else {
        int my = (ly + ry) / 2;
        build_y(vx, lx, g, rx, vy*2, ly, my);
        build_y(vx, lx, g, rx, vy*2+1, my+1, ry);
        t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
    }
}
void build_x(vector<vector<int>> &g , int vx, int lx, int rx) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        build_x(g , vx*2, lx, mx);
        build_x(g , vx*2+1, mx+1, rx);
    }
    build_y(vx, lx, g, rx, 1, 0, m-1);
}
Node qry_y(int vx, int vy, int tly, int try_, int ly, int ry) {
    if (ly > ry)
        return Node();
    if (ly == tly && try_ == ry)
        return Node(t[vx][vy]);
    int tmy = (tly + try_) / 2;
    return qry_y(vx, vy*2, tly, tmy, ly, min(ry, tmy)) +
           qry_y(vx, vy*2+1, tmy+1, try_, max(ly, tmy+1), ry);
}

Node qry_x(int vx, int tlx, int trx, int lx, int rx, int ly, int ry) {
    if (lx > rx)
        return Node();
    if (lx == tlx && trx == rx)
        return qry_y(vx, 1, 0, m-1, ly, ry);
    int tmx = (tlx + trx) / 2;
    return qry_x(vx*2, tlx, tmx, lx, min(rx, tmx), ly, ry) +
           qry_x(vx*2+1, tmx+1, trx, max(lx, tmx+1), rx, ly, ry);
}
void upd_y(int vx, int lx, int rx, int vy, int ly, int ry, int x, int y, int nval) {
    if (ly == ry) {
        if (lx == rx)
            t[vx][vy] = nval;
        else
            t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
    } else {
        int my = (ly + ry) / 2;
```

```
      if (y <= my)
        upd_y(vx, lx, rx, vy*2, ly, my, x, y, nval);
      else
        upd_y(vx, lx, rx, vy*2+1, my+1, ry, x, y, nval);
      t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
    }
  }
  void upd_x(int vx, int lx, int rx, int x, int y, int nval) {
    if (lx != rx) {
      int mx = (lx + rx) / 2;
      if (x <= mx)
        upd_x(vx*2, lx, mx, x, y, nval);
      else
        upd_x(vx*2+1, mx+1, rx, x, y, nval);
    }
    upd_y(vx, lx, rx, 1, 0, m-1, x, y, nval);
  }
public:
  void init(int n, int m){
    this->n = n;
    this->m = m;
    int r = 1 , c = 1;
    while(r < n) r *= 2;
    while(c < m) c *= 2;
    t = vector<vector<Node>>(2 * r , vector<Node>(2 * c));
  }
  void build(vector<vector<int>> &g){
    build_x(g , 1 , 0 , n - 1);
  }
  int qry(int x , int y , int xx , int yy){
    return qry_x(1 , 0 , n - 1 , x , xx , y , yy).v;
  }
  void upd(int r , int c , int val){
    upd_x(1 , 0 , n - 1 , r , c , val);
  }
```

segment_sparse_table_2d.cpp

```
/*
  2D segment sparse table
  - Memory allocation: O(4n x mlog(m))
  - range [l..r] , r: inclusive
  - Base: 0-index

  Function description:
    1. build(n , m , n): build the segment tree with grid[n][m]
      Time complexity: O(4n x mlog(m))
    2. qry(lx , rx , ly , ry): find (min,max,sum,produc) of rectangle [lx...rx] x [ly...ry]
      Time complexity: O(log(n))
*/
template<class T = int>
struct sparse_2d {
  int n, m, LOG , N;
  T DEFAULT = 0;
  vector<vector<vector<T>>> tree;

  void build(int _n, int _m , vector<vector<T>> &a) {
    n = _n;
    m = _m;
    LOG = 31 - __builtin_clz(m) + 1;

    N = 1;
    while(N < n) N *= 2; // N must be power of 2
    tree = vector<vector<vector<T>>>(2 * N + 1, vector<vector<T>>(LOG, vector<T>(m + 1)));

    for (int x = 0; x < n; x++)
    {
      for (int y = 0; y < m; y++)
        tree[N + x][0][y] = a[x][y];
      for (int k = 1; k < LOG; k++)
        for (int y = 0; y + (1 << k) <= m; y++)
          tree[N + x][k][y] = comb(tree[N + x][k - 1][y], tree[N + x][k - 1][y + (1 << (k - 1))]);
    }
    for (int v = N - 1; v > 0; v--)
      for (int k = 0; k < LOG; k++)
        for (int y = 0; y + (1 << k) <= m; y++)
          tree[v][k][y] = comb(tree[2 * v][k][y], tree[2 * v + 1][k][y]);
    return;
  }

  T comb(T a, T b) {
    return __gcd(a, b); // Change the custom operator
  }

  T qry(int v, int a, int b, int x1, int x2, int y1, int y2, int k) {
    if (x1 <= a && b <= x2){
      return comb(tree[v][k][y1], tree[v][k][y2]);
    }
    if (x1 >= b || a >= x2) return 0;
    int mid = (a + b) / 2;
    return comb(qry(2 * v, a, mid, x1, x2, y1, y2, k), qry(2 * v + 1, mid, b, x1, x2, y1, y2, k));
  }

  T qry(int lx, int ly, int rx, int ry) {
    int k = 31 - __builtin_clz(ry - ly + 1);
    return qry(1, 0, N, lx, rx + 1, ly, ry - (1 << k) + 1, k);
  }
```

BIT

BIT.cpp

```
/*
    --------------BIT-----------------
    Use: 1 - add value to element in the array
         2 - sum of range in the array
    Base: 0-index
    Time complexity : add , sum >> O(log(n))

*/
template<class T = int>
struct BIT{
    vector<T> tree;
    int n;
    void init(int n){
        this->n = n;
        tree.assign(n , 0);
    }
    void add(int pos , T val){
        for(pos++; pos <= n ; pos += (pos & (-pos)))
            tree[pos - 1] += val;
    }
    T sum(int pos){
        T ret = 0;
        for(pos++; pos ; pos -= (pos & (-pos)))
            ret += tree[pos - 1];
        return ret;
    }
    T qry(int l , int r){ return sum(r) - sum(l - 1); }
    T qry(int i){ return sum(i , i); }
};
```

BIT_LAZY_MODIFICATION.cpp

```
/*
    BIT with LAZY MODIFICATION
    use : So far we have presented BIT as a structure which is entirely allocated in memory during the initialization.
        An advantage of this approach is that accessing tree[idx] requires a constant time. On the other hand, we might need to access only tree[idx] for a couple of different values of idx, e.g. log n different values, while we allocate much larger memory. This is especially aparent in the cases when we work with multidimensional BIT.
*/
// 1-base
const int inf = 1e9 + 9;
map<int,long long> tree; // LAZY MODIFICATION
struct BIT {
    void add(int x, int val) {
        for(; x < inf; x += (x & -x))  tree[x] += val;
    }
    long long sum(int x){
        long long res = 0;
        for(;x > 0; x -= (x & -x)) res += tree[x];
        return res;
    }
    long long sum(int l , int r){ return sum(r) - sum(l - 1); }
};
```

BIT_Ranges.cpp

```
const int N = 2e5 + 9;
class BITrange {
private:
    long long m[N], c[N];
    void add(int pos, long long mVal, long long cVal) {
        for (++pos; pos <= N; pos += (pos & (-pos))) {
            m[pos - 1] += mVal;
            c[pos - 1] += cVal;
        }
    }
    long long get(int pos) {
        long long ret = 0;
        int x = pos;
        for (++pos; pos; pos -= (pos & (-pos))) {
            ret += m[pos - 1] * x + c[pos - 1];
        }
        return ret;
    }

public:
    void init(int n) {
        memset(m, 0, n * sizeof(m[0]));
        memset(c, 0, n * sizeof(m[0]));
    }
    void addRange(int st, int en, long long val) {
        if(st > en) return;
        add(st, val, -val * (st - 1));
        add(en + 1, -val, val * en);
    }
    void addIndex(int i, ll val){ addRange(i,i,val); }
    ll qry(int l , int r){ return get(r) - get(l - 1); }
    ll qry(int i){ return get(i) - get(i - 1); }
};
```

BIT_with_coordinate_compression.cpp

```
template<class T = int>
struct BIT{
    vector<T> tree , v;
    int n;
    void init(vector<T> vv){
        this->v = vv;
        sort(all(v));
        v.erase(unique(v.begin() , v.end()) , v.end());
        this->n = v.size();
        tree.assign(n , 0);
    }
    void add(T idx , T val){
        int pos = lower_bound(v.begin(), v.end() , idx) - v.begin();
        for(pos++; pos <= n ; pos += (pos & (-pos)))
```

```
            tree[pos - 1] += val;
    }
    T sum(int pos){
        pos = lower_bound(v.begin(), v.end() , pos) - v.begin();
        T ret = 0;
        for(pos++; pos ; pos -= (pos & (-pos)))
            ret += tree[pos - 1];
        return ret;
    }
    T qry(int l, int r){ return sum(r) - sum(l - 1); }
    T get_idx(int i){ return qry(i , i); }
};
```

Multiset_fenwick.cpp

```
const int N = 1e5 + 9;
struct BIT{
    vector<int> tree;
    int n;
    void init(int n){
        this->n = n;
        tree.assign(n , 0);
    }
    void add(int pos , int val){
        for(pos++; pos <= n ; pos += (pos & (-pos)))
            tree[pos - 1] += val;
    }
    int sum(int pos){
        int ret = 0;
        for(pos++; pos ; pos -= (pos & (-pos)))
            ret += tree[pos - 1];
        return ret;
    }
    int sum(int l , int r){ return sum(r) - sum(l - 1); }
    int getidx(int i){ return sum(i , i); }
};
struct MultiSet{
    BIT B;
    MultiSet(){ B.init(N); }
    void insert(int val){ B.add(val,1); }
    void erase(int val){ B.add(val,-1); }
    int count(int val){ return B.sum(val - 1 , val); }
```

```cpp
    int size(){ return B.sum(N-1); }
    int order_of_key(int key){
        int l = 0 , r = N - 1, mid;
        while(l < r){
            mid = l + (r - l) / 2;
            if(B.sum(mid) > key) r = mid;
            else l = mid + 1;
        }
        return l;
    }
}s;
```

mex_using_BIT.cpp

```cpp
// It 's faster than set
struct BIT {
    vector<int> tree;
    int n;
    void init(int n) {
        this->n = 1 << (__lg(n) + !!(n & (n - 1)));
        tree.assign(this->n + 1, 0);
    }
    void insert(int x, int v = 1) {
        for (++x; x <= n; x += (x) & (-x)) {
            tree[x - 1] += v;
        }
    }
    void erase(int x, int v = -1) {
        for (++x; x <= n; x += (x) & (-x)) {
            tree[x - 1] += v;
        }
    }
    int search(int v = 0) { // O(log(n))
        int p = 0, idx = 0;
        for (int sz = n >> 1; sz; sz >>= 1) {
            if (tree[p + sz - 1] <= v) {
                p += sz;
            }
        }
        return p;
    }
}
```

```

// More function can be useful
struct Mex{
  BIT missing;
  vector<int> frq;
  int n;
  void init(int n){
    missing.init(n + 2);
    frq.resize(n + 2);
    this->n = n + 2;
  }
  int get(int i){ return missing.search(); } // O(log(n))
  void add(int v){ // O(log(n))
    if(v > n) return;
    if(++frq[v] == 1) missing.erase(v);
  }
  void rm(int v){ // O(log(n))
    if(v > n) return;
    if(--frq[v] == 0) missing.insert(v);
  }
};
```

DSU

DSU.cpp

```
struct DSU{
 vector<int> par , size;
 DSU(int n){
  par.resize(n); size.resize(n , 1);
  for(int i = 0; i < n; i++) par[i] = i;
 }
 int get(int a){return par[a] = (par[a] == a) ? a : get(par[a]);}
 void Union(int a , int b){
  a = get(a); b = get(b);
  if(a == b) return;
  if(size[a] > size[b]) swap(a , b);
  size[b] += size[a];
  par[a] = b;
 }
 int same_Group(int a , int b){return get(a) == get(b);}
};
```

DynamicConnectivity_offline.cpp

```cpp
int ans;

struct Query {
    int u, v;   // Vertices to connect
    int l, r;   // Time interval [l,r] when edge exists (in case large time use compression)
};

struct Elem {
    int u;      // Parent vertex
    int v;      // Child vertex being attached
    int szU;    // Original size of set u
    int cnt;    // Previous count of disjoint sets
};

struct DSURollback {
    int cnt;
    int currentTime = 0;  // initialized
    stack<Elem> st;
    vector<int> sz, par;
    vector<vector<pair<int, int>>> g;

    DSURollback(int n) : cnt(n), currentTime(0) {  // initialized in constructor
        int seg = 1;
        while (seg < n) seg *= 2;
        g.resize(2 * seg + 5);
        par.resize(n + 1);
        sz.resize(n + 1, 1);
        iota(par.begin(), par.end(), 0);
    }

    int findSet(int u) { return par[u] == u ? u : findSet(par[u]); }

    void update(int u, int v) {
        st.push({u, v, sz[u], cnt});
        cnt--, par[v] = u;
        sz[u] += sz[v];
        ans = max(ans, currentTime);  // using tracked time
    }

    void unionSet(int u, int v) {
        u = findSet(u), v = findSet(v);
        if (u != v) {
            if (sz[u] < sz[v]) swap(u, v);
            update(u, v);
        }
    }

    void rollback(int x) {
        while (st.size() > x) {
            auto e = st.top(); st.pop();
            cnt = e.cnt;
            sz[e.u] = e.szU;
            par[e.v] = e.v;
        }
    }

    void traverse(int x, int lX, int rX, const int &l, const int &r, const int &u, const int &v) {
        if (rX < l || lX > r) return;
        if (lX >= l && rX <= r) { g[x].emplace_back(u, v); return; }
        int m = (lX + rX) / 2;
        traverse(x * 2, lX, m, l, r, u, v);
        traverse(x * 2 + 1, m + 1, rX, l, r, u, v);
    }

    void solve(int x, int l, int r) {
        int cur = st.size();
        currentTime = l;  // updating current time
        for (auto &i : g[x]) unionSet(i.first, i.second);
        if (l == r) { rollback(cur); return; }
        int m = (l + r) / 2;
        solve(x * 2, l, m);
        solve(x * 2 + 1, m + 1, r);
        rollback(cur);
    }

    void build(vector<Query> &queries) {
        for (auto &q : queries) traverse(1, 0, cnt - 1, q.l, q.r, q.u, q.v);
        solve(1, 0, cnt - 1);
    }
};
```

RollbackUF.cpp

```cpp
struct RollbackUF {
  vector<int> e;          // Parent/size array (-ve size, +ve parent)
  vector<pair<int,int>> st;   // History for rollback (index, prev value)

  RollbackUF(int n) : e(n, -1) {}

  int size(int x) {        // Returns size of set containing x
    return -e[find(x)];
  }

  int find(int x) {        // Gets root/representative of x's set
    return e[x] < 0 ? x : find(e[x]);
  }

  int time() {             // Current operation count
    return st.size();
  }

  void rollback(int t) {     // Undo operations after time t
    for(int i = time(); i > t; i--) {
      e[st[i-1].first] = st[i-1].second;
    }
    st.resize(t);
  }

  bool join(int a, int b) {  // Union sets of a and b, return true if joined
    a = find(a);
    b = find(b);
    if(a == b) return false;

    if(e[a] > e[b]) swap(a, b);  // a will be root

    st.push_back({a, e[a]});     // Save state for rollback
    st.push_back({b, e[b]});
    e[a] += e[b];             // Update size
    e[b] = a;                // Set parent
    return true;
  }
};
```

DSU on tree

sackDFS_keepBigChild.cpp

```
/*
DSU on tree by keeping largest child
Let:
  - st[u] dfs starting time of vertex u,
  - ft[u] be it's finishing time and
  - ver[time] is the vertex which it's starting time is equal to time.
  - sz[u] is the size of subtree of node

Pros: You can keep segment tree for each node with using extra memory
*/

const int N = 1e5; // TODO: change it to maximum possible N
int dfs_time = 0;
int st[N] , ft[N] , big[N] , ver[N] , sz[N];
vector<int> adj[N];
void preDFS(int u , int p){
  st[u] = dfs_time++;
  ver[ st[u] ] = u;
  sz[u] = 1, big[u] = -1;
  for(auto v : adj[u]) if(v != p){
    preDFS(v,u);
    sz[u] += sz[v];
    if(big[u] == -1 || sz[v] > sz[ big[u] ]) big[u] = v;
  }
  ft[u] = dfs_time;
}
void sackDFS(int u, int p, bool keep){
  int bigChild = big[u];
  if(bigChild != -1) sackDFS(bigChild, u, 1);  // bigChild marked as big and not cleared from cnt
  for(auto v : adj[u]){
    if(v != p && v != bigChild){
      for(int p = st[v]; p < ft[v]; p++){
        // TODO: Add your information about ver[p]

      }


    }


  }
  // TODO: Add your information about u


```

```
    // TODO: All information about the subtree of  u  is kept, and you can now query it.


    if(keep == 0){
        for(int p = st[u]; p < ft[u]; p++){
            // TODO: Remove the added information about ver[p]

        }
    }

}
// Calling -> sackDFS(root, -1 , 0)
```

small_to_large_trick.cpp

```
/*
    Classic problems: https://cses.fi/problemset/task/1139
    You are given a rooted tree consisting of n nodes.
    The nodes are numbered 1,2,..,n, and node 1 is the root. Each node has a color.
    Your task is to determine for each node the number of distinct colors in the subtree of the node.
    Time complexity: O(n log n)
    Space comlexity: O(n log n)
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
#define all(v) v.begin() , v.end()
#ifndef ONLINE_JUDGE
#define debug(x) cerr << #x << ": " << x << '\n';
#else
#define debug(x)
#endif
const int N = 2e5 + 9;
vector<int> adj[N];
map<int,int> mp[N];
int  ans[N], color[N];
int dfs(int u, int par){
    int p = u;
    mp[p][color[u]]++;
    for(auto &v : adj[u]) if(v != par){
        int x = dfs(v , u);
        if(mp[x].size() > mp[p].size()) swap(x,p);
        for(auto &[c,frq]: mp[x]) mp[p][c] += frq;
    }
    ans[u] = (int) mp[p].size();
    return p;
}
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    int n; cin >> n;
    for(int i = 0; i < n; ++i) cin >> color[i];
    for(int i = 0; i < n - 1; ++i){
        int u , v; cin >> u >> v;
        --u; --v;
        adj[u].emplace_back(v);
        adj[v].emplace_back(u);
    }
    dfs(0,0);
    for(int i = 0; i < n; ++i) cout << ans[i] << " ";
}
```

Divide and conquer

offline_static_rmq_dc.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define f first
#define s second
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    int n  , q; cin >> n >> q;
    int a[n];
    for(int i = 0; i < n; ++i) cin >> a[i];
    int lt[n] , rg[n] , ans[q];
    vector<array<int,3>> Queries;
    for(int i = 0; i < q; ++i){
        int l , r; cin >> l >> r;
        --l; --r;
        Queries.push_back({l , r , i});
    }
    auto comb = [&](int x , int y){ return min(x , y); };
    function<void(int,int,vector<array<int,3>>)> solve = [&]( int l , int r , vector<array<int,3>> Queries){
        if(Queries.empty()) return; // just optimization
        if(l == r){
            for(auto &[L , R , ind] : Queries) ans[ind] = a[l];
            return;
        }
        int m = (l + r) / 2;
        lt[m] = a[m];  rg[m + 1] = a[m + 1];
        for(int i = m - 1; i >= l; --i) lt[i] = comb(a[i] , lt[i + 1]);
        for(int i = m + 2; i <= r; ++i) rg[i] = comb(a[i] , rg[i - 1]);
        vector<array<int,3>> tmp[2];

        for(auto &[L , R , ind] : Queries){
            if(L <= m && m < R){
                ans[ind] = comb(lt[L] , rg[R]);
            }else{
                tmp[L > m].push_back({L , R , ind});
            }
        }
        solve(l , m , tmp[0]); solve(m + 1 , r , tmp[1]);
    };
    solve(0 , n - 1 , Queries);
    for(int i = 0; i < q; ++i){
        cout << ans[i] << '\n';
    }
```

onlineRangeQueryD&C_benq.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first

```

```
#define s second
#define all(v) v.begin() , v.end()
#ifndef ONLINE_JUDGE
#define debug(x) cerr << #x << ": " << x << '\n';
#else
#define debug(x)
#endif
template<class T, int SZ> struct RangeQuery {
  int n;
  T stor[SZ][32-__builtin_clz(SZ)], id = 1;
  vector<T> a;
  T comb (T a, T b) {
    return mul(a,b);  // associative operation
  }
  void fill(int l, int r, int ind) {
    if (ind < 0) return;
    int m = (l+r)/2;
    T prod = id; for(int i = l; i < m; ++i) stor[i][ind] = prod = comb(a[i],prod);
    prod = id; for(int i = m; i < r; ++i) stor[i][ind] = prod = comb(prod,a[i]);
    fill(l,m,ind-1); fill(m,r,ind-1);
  }
  void init() {
    n = 1; while ((1<<n) < int(a.size())) ++n;
    a.resize(1 << n);
    fill(0,(1<<n),n-1);
  }
  T query(int l, int r) {
    if (l == r) return a[l];
    int t = 31-__builtin_clz(r^l);
    return comb(stor[l][t],stor[r][t]);
  }
};

int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
  #ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
  #endif

}
```

online_static_rmq_dc.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 2e5 + 9 , lg = 18;
int n , q , a[N];
int prec[lg][N] , msk[N];
int comb(int x , int y){ return min(x , y); }; // associative operator (eg. sum, prod, xor,or, and)
void dc(int l, int r, int lvl){
  if(l == r) return;
  int m = (l + r) / 2;
  prec[lvl][m] = a[m];
  prec[lvl][m + 1] = a[m + 1];
  for(int i = m - 1; i >= l; --i) prec[lvl][i] = comb(a[i] , prec[lvl][i + 1]);
  for(int i = m + 2; i <= r; ++i) prec[lvl][i] = comb(a[i] , prec[lvl][i - 1]);
  for(int i = m + 1; i <= r; ++i) msk[i] |= (1 << lvl);
  dc(l , m , lvl + 1); dc(m + 1 , r , lvl + 1);
};
int qry(int l , int r){
  int k = __builtin_ctz(msk[l] ^ msk[r]);
  return comb(prec[k][l] , prec[k][r]);
};
int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
  #ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
  #endif
  cin >> n >> q;
  for(int i = 0; i < n; ++i) cin >> a[i];
  dc(0 , n , 0);
  for(int i = 0; i < q; ++i){
    int l , r; cin >> l >> r;
    --l; --r;
    if(l == r) cout << a[l] << '\n';
    else cout << qry(l , r) << '\n';
  }
}
```

Kd-tree

kd-tree 3d.cpp

```cpp
struct Point {
    long double x, y, z;
    Point(long double x , long double y , long double z):x(x) , y(y) , z(z){}
    bool operator<(const Point& p) const {
        return x < p.x || (x == p.x && (y < p.y || (y == p.y && z < p.z)));
    }
};

class KDTree {
private:
    struct Node {
        Point point;
        Node* left;
        Node* right;
        Node(Point p) : point(p), left(nullptr), right(nullptr) {}
    };
    Node* root;

    Node* build(vector<Point>& points, int depth) {
        if (points.empty()) return nullptr;
        int axis = depth % 3;
        sort(points.begin(), points.end(), [axis](Point a, Point b) {
            if (axis == 0) return a.x < b.x;
            if (axis == 1) return a.y < b.y;
            return a.z < b.z;
        });
        int mid = points.size() / 2;
        Node* node = new Node(points[mid]);
        vector<Point> leftPoints(points.begin(), points.begin() + mid);
        vector<Point> rightPoints(points.begin() + mid + 1, points.end());
        node->left = build(leftPoints, depth + 1);
        node->right = build(rightPoints, depth + 1);
        return node;
    }

    long double dist(Point p1, Point p2) {
        long double res = (p1.x - p2.x) * (p1.x - p2.x);
        res += (p1.y - p2.y) * (p1.y - p2.y);
        res += (p1.z - p2.z) * (p1.z - p2.z);
        return sqrtl(res);
    }

    void nearest(Node* node, Point target, int depth, long double& bestDist) {
        if (!node) return;
        long double currentDist = dist(node->point, target);
        if (currentDist < bestDist && currentDist > 0) bestDist = currentDist;  // Added currentDist > 0 check

        int axis = depth % 3;
        Node* next = (axis == 0 ? target.x < node->point.x :
              (axis == 1 ? target.y < node->point.y : target.z < node->point.z)) ? node->left : node->right;
        Node* other = next == node->left ? node->right : node->left;

        nearest(next, target, depth + 1, bestDist);

        long double axisDist = (axis == 0 ? abs(target.x - node->point.x) :
                   (axis == 1 ? abs(target.y - node->point.y) : abs(target.z - node->point.z)));
        if (axisDist < bestDist) {
            nearest(other, target, depth + 1, bestDist);
        }
    }

    void findMinDistanceHelper(Node* node, long double& minDist) {
        if (!node) return;

        // Find nearest point to current node's point
        long double currentMin = numeric_limits<long double>::max();
        nearest(root, node->point, 0, currentMin);
        minDist = min(minDist, currentMin);

        // Recurse on children
        findMinDistanceHelper(node->left, minDist);
        findMinDistanceHelper(node->right, minDist);
    }

public:
    KDTree(vector<Point>& points) {
        root = build(points, 0);
    }

    long double nearest(Point target) {
        long double bestDist = numeric_limits<long double>::max();
```

```
        nearest(root, target, 0, bestDist);
        return bestDist;
    }

    long double findMinDistance() {
        if (!root || (!root->left && !root->right)) return numeric_limits<long double>::max();
        long double minDist = numeric_limits<long double>::max();
        findMinDistanceHelper(root, minDist);
        return minDist;
    }
```

kd-tree kactl.cpp

```
/*
Quick Tips for KD-tree in Contest:
1. Randomize input points to avoid O(n) search on sorted data
2. If TLE on findMinDistance(), consider using simpler n^2 brute force
3. For nearest neighbor, if getting WA/TLE, check if input points are unique
4. Watch out for integer overflow in distance calculations
5. Consider using Manhattan distance if precision is an issue
*/

template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
    T dist2() const { return x*x + y*y; } // Replace if u use manhattan distance {return abs(x) + abs(y); }
    friend ostream& operator<<(ostream& os, P p) {
        return os << "(" << p.x << "," << p.y << ")"; }
};

typedef long long ll;
typedef Point<ll> P;
const ll INF = numeric_limits<ll>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
  P pt;
  ll x0 = INF, x1 = -INF, y0 = INF, y1 = -INF;
  Node *first = 0, *second = 0;

  ll distance(const P& p) {
    ll x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    ll y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      sort(vp.begin(), vp.end(), x1 - x0 >= y1 - y0 ? on_x : on_y);
      int half = vp.size()/2;
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
    }
  }
};

class KDTree {
private:
  Node* root;

  // O(log n) average, O(n) worst: Helper for nearest neighbor search
  pair<ll, P> search(Node *node, const P& p) {
    if (!node->first) {
      return make_pair((p - node->pt).dist2(), node->pt);
    }
    Node *f = node->first, *s = node->second;
    ll bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
    auto best = search(f, p);
    if (bsec < best.first)
      best = min(best, search(s, p));
    return best;
  }

  void findMinDistanceHelper(Node* node, long double& minDist) {
    if (!node || (!node->first && !node->second)) return;

    if (node->first && node->second) {
      minDist = min(minDist, (long double)(node->first->pt - node->second->pt).dist2());
    }

    findMinDistanceHelper(node->first, minDist);
    findMinDistanceHelper(node->second, minDist);
  }

public:
  KDTree(const vector<P>& vp) : root(new Node({vp.begin(), vp.end()})) {}

  // O(log n) average: Finds nearest neighbor to query point
  pair<ll, P> nearest(const P& p) {
    return search(root, p);
  }

  long double findMinDistance() {
    if (!root) return numeric_limits<long double>::max();
    long double minDist = numeric_limits<long double>::max();
    findMinDistanceHelper(root, minDist);
    return sqrt(minDist);
  }
}
```

LCA

LCA.cpp

```
const int N = 1e5 + 10 , LOG = 20;
vector<int> adj[N];
struct lca{
  int n;
  vector<int> depth , parent[LOG];
  void init(int n , int root = 0){
    this->n = n;
    depth.resize(n);
    for(int i = 0; i < LOG; ++i) parent[i].resize(n);
    dfs(root , root , 0);
  }
```

```cpp
    void dfs(int u, int p, int d) {
        depth[u] = d;
        parent[0][u] = p;
        for (int i = 1; i < LOG; ++i) {
            parent[i][u] = parent[i - 1][parent[i - 1][u]];
        }
        for (int v : adj[u]) {
            if (v == p) continue;
            dfs(v, u, d + 1);
        }
    }
    int kth_ancestor(int u, int k) {
        for (int i = 0; i < LOG; ++i) {
            if ((1 << i) & k) {
                u = parent[i][u];
            }
        }
        return u;
    }
    int LCA(int u, int v) {
        if (depth[u] > depth[v]) swap(u, v);
        int k = depth[v] - depth[u];
        v = kth_ancestor(v, depth[v] - depth[u]);
        if (u == v) return u;

        for (int i = LOG - 1; ~i; --i) {
            if (parent[i][u] != parent[i][v]) {
                u = parent[i][u];
                v = parent[i][v];
            }
        }
        assert(parent[0][u] == parent[0][v]);
        return parent[0][u];
    }
};
```

LCA_in_DSU.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
```

```
#define f first
#define s second
const int LOG = 20;
struct DSU
{
 vector<int> par , size  , W[LOG] , up[LOG] , dep;
 ll MST = 0;
 int n;
 DSU(int n){
  this->n = n;
  par.resize(n); size.resize(n , 1); dep.resize(n);
  rep(i , 0 , n) par[i] = i;
  rep(i , 0 , LOG) up[i].resize(n);
 }
 pair<int,int> get(int a){ // {par , depth}
  if(par[a] == a) return {a , 0};
  auto p = get(par[a]); p.s++;
  return p;
 } // no path compression
 void Union(int a, int b , int w){
  a = get(a).f; b = get(b).f;
  if(a == b) return; // same component
  if(size[a] > size[b]) swap(a , b);
  size[b] += size[a];
  par[a] = b;
  MST += w;
 }
 void Build(){
  rep(i , 0 , n){
    up[0][i] = par[i];
    dep[i] = get(i).s;
  }
  rep(x , 1 , LOG) rep(u , 0 , n){
    up[x][u] = up[x - 1][ up[x - 1][u] ];
  }
 }
 ll lca(int u , int v, int w){
  int mx = 0;
  if(dep[u] < dep[v]) swap(u , v);
  int k = dep[u] - dep[v];
  for(int i = LOG - 1; i >= 0; i--) if((k >> i) & 1){
    mx = max(mx , W[i][u]);
    u = up[i][u];
  }
```

```
    if(u == v) return ;
    for(int i = LOG - 1; i >= 0; --i) if(up[i][u] != up[i][v]){
        mx = max({mx , W[i][u] , W[i][v]});
        u = up[i][u];
        v = up[i][v];
    }
    assert(up[0][u] == up[0][v]);
    return ;
  }
};
struct edge{ int u , v , w; };
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    int n , m; cin >> n >> m;
    vector<edge> e(m); // {w , u , v}
    rep(i , 0 , m){
        cin >> e[i].u >> e[i].v >> e[i].w;
        --e[i].u; --e[i].v;
    }
    int idx[m];
    iota(idx , idx + m , 0);
    sort(idx , idx + m , [&](int x , int y){
        return e[x].w < e[y].w;
    });
    // Building MST
    DSU d(n);
    for(int i = 0; i < m; ++i){
        int l = idx[i];
        d.Union(e[l].u , e[l].v , e[l].w);
    }
    d.Build();

}
```

lca_O(1).cpp

```
// Problem Link: https://cses.fi/problemset/task/1688
#include <bits/stdc++.h>
```

```
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
#define all(v) v.begin() , v.end()
#ifndef ONLINE_JUDGE
#define debug(x) cerr << #x << ": " << x << '\n';
#else
#define debug(x)
#endif
const int N = 2e5 + 9;
vector<int> adj[N];
vector<int> euler_tour; // in time
int in[4 * N] , Timer;
void dfs(int u = 0 , int p = 0){
    in[u] = Timer++;
    euler_tour.emplace_back(u);
    for(auto &v : adj[u]) if(v != p){
        dfs(v , u);
        euler_tour.emplace_back(u);
        Timer++;
    }
}
struct SparseTable {
    vector<int> log;
    vector<vector<pair<ll, int>>> spt;

    void init(int n) {
        log.assign(n + 1, 0);
        for (int i = 2; i <= n; i++) {
            log[i] = 1 + log[i / 2];
        }
        int k = log[n] + 1;
        spt = vector<vector<pair<ll, int>>>(k, vector<pair<ll, int>>(n));
        for (int i = 0; i < n; i++) {
            spt[0][i] = { in[euler_tour[i]] , euler_tour[i] };
        }
        for (int j = 1; 1 << j <= n; j++) {
            for (int i = 0; i + (1 << j) - 1 < n; i++) {
                spt[j][i] = merge(spt[j - 1][i], spt[j - 1][i + (1 << (j - 1))]);
            }
        }
    }
    pair<ll, int> merge(pair<ll, int> &x, pair<ll, int> &y) {
        if(x.f < y.f) return x;
        return y;
    }
    pair<ll, int> query(int i, int j) {
        int len = j - i + 1;
        int k = log[len];
        return merge(spt[k][i], spt[k][j - (1 << k) + 1]);
    }
    int lca(int i , int j){
        if(in[i] > in[j]) swap(i , j);
        return query(in[i] , in[j]).s;
    }
};
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    int n , q; cin >> n >> q;
    for(int i = 1; i < n; ++i){
        int p; cin >> p;
        --p;
        adj[p].emplace_back(i);
    }
    dfs();
    SparseTable spt;
    spt.init(Timer);
    while(q--){
        int u , v; cin >> u >> v;
        --u; --v;
        cout << spt.lca(u , v) + 1 << '\n';
    }
```

Minimum spanning tree

MST_Kruskal.cpp

```
struct DSU
{
 vector<int> par , size;
 DSU(int n){
  par.resize(n); size.resize(n , 1);
  rep(i , 0 , n) par[i] = i;
 }
 int get(int a){return par[a] = (par[a] == a) ? a : get(par[a]);}
 void Union(int a, int b){
  a = get(a); b = get(b);
  if(a == b) return; // In the same Group
  if(size[a] > size[b]) swap(a , b);
  size[b] += size[a];
  par[a] = b;
 }
};
void sol(){
 int n , m; cin >> n >> m;
 vector<array<int,3>> e(m); // {w , u , v}
 rep(i , 0 , m) cin >> e[i][1] >> e[i][2] >> e[i][0];
 sort(e.begin() , e.end());
 ll ans = 0;
 DSU d(n);
 for(auto &[w , u , v] : e){
  --u; --v;
  if(d.get(u) == d.get(v)) continue;
  d.Union(u , v);
  ans += w;
 }
 cout << ans;
}
```

MST_Prim's.cpp

```
int n , m; cin >> n >> m;
vector<pair<int,int>> adj[n]; // {v , w}
for(int i = 0;i < m; ++i){
  int u , v , w; cin >> u >> v >> w;
  --u; --v;
  adj[u].emplace_back(v , w);
  adj[v].emplace_back(u , w);
}
priority_queue<pair<int,int>> q; // {-w , u}
vector<int> vis(n);
```

```
q.push({0 , 0}); // start from any node
ll cost = 0;
while(q.size()){
   auto [d , u] = q.top(); q.pop();
   if(vis[u]) continue;
   d *= -1; vis[u] = 1;
   cost += d;
   for(auto &[v , w] : adj[u]) if(vis[v] == 0){
      q.push({-w , v});
   }
}
cout << cost;
```

Montomic stack and Queue

NxtorPrv_MinorMax.cpp

```
vector<int> getNxtMin(vector<int> &arr) {
   stack<int> st;
   vector<int> res(arr.size(), arr.size());
   for (int i = 0; i < arr.size(); i++) {
      while (!st.empty() && arr[st.top()] > arr[i]) {
         res[st.top()] = i;
         st.pop();
      }
      st.push(i);
   }
   return res;
}

vector<int> getPrevMin(vector<int> &arr) {
   stack<int> st;
   vector<int> res(arr.size(), -1);
   for (int i = arr.size() - 1; i >= 0; i--) {
      while (!st.empty() && arr[st.top()] >= arr[i]) {
         res[st.top()] = i;
         st.pop();
      }
      st.push(i);
   }
   return res;
}
```

```cpp

vector<int> getNxtMax(vector<int> &arr) {
    stack<int> st;
    vector<int> res(arr.size(), arr.size());
    for (int i = 0; i < arr.size(); i++) {
        while (!st.empty() && arr[st.top()] < arr[i]) {
            res[st.top()] = i;
            st.pop();
        }
        st.push(i);
    }
    return res;
}

vector<int> getPrevMax(vector<int> &arr) {
    stack<int> st;
    vector<int> res(arr.size(), -1);
    for (int i = arr.size() - 1; i >= 0; i--) {
        while (!st.empty() && arr[st.top()] <= arr[i]) {
            res[st.top()] = i;
            st.pop();
        }
        st.push(i);
    }
    return res;
}
```

monotonicQueue.cpp

```cpp
/**
* Monotonic queue to keep track of the minimum and the maximum
* elements so far in the queue in amortized time of O(1).
*/
template<class T>
class monotonic_queue {
    queue<T> qu;
    deque<T> mx, mn;
    public:
    void push(T v) {
        qu.push(v);
        while (mx.size() && mx.back() < v) mx.pop_back();
        mx.push_back(v);
        while (mn.size() && mn.back() > v) mn.pop_back();
        mn.push_back(v);
    }
    void pop() {
        if (mx.front() == qu.front()) mx.pop_front();
        if (mn.front() == qu.front()) mn.pop_front();
        qu.pop();
    }
    T front() const {
        return qu.front();
    }
    T max() const {
        return mx.front();
    }
    T min() const {
        return mn.front();
    }
    size_t size() const {
        return qu.size();
    }
};
```

Offline Range Queries

Arithmetic_progression_prefix_sum.cpp

```
const int N = 1e6 + 5;
ll d[N] , P[N];
void inc(int l , int r , int k , int b){
    // add for each x : [l , r] , v_x += k * (x - l) + b
    if(l > r) return;
    d[l + 1] += k;
    d[r + 1] -= k;

    P[l] += b;
    P[r + 1] -= b + 1LL * k * (r - l);
}
void build(){
    for(int i = 1; i < N; ++i){
        d[i] += d[i - 1];
        P[i] += P[i - 1] + d[i];
    }
}

\\ ---------------------------------------------
const ixt N = 1e6 + 5;
ll d[N] , P[N];
void inc(int l , int r , int k , int b){
   // add for each x : [l , r] , v_x += k * (x - l + 1) + b
   if(l > r) return;
   d[l] += k;
   d[r + 1] -= k;

   P[l] += b;
   P[r + 1] -= b + 1LL * k * (r - l + 1);
}
void build(){
   for(int i = 1; i < N; ++i){
      d[i] += d[i - 1];
      P[i] += P[i - 1] + d[i];
   }
}
```

Ordered set

ordered_set.cpp

```
// only in less_equal lower_bound work such as upper_bound;
// by *find_by_order given index --> val ;
// by order_of_key given  value --> index ;
// erase and insert in log(n) ;
// used norm policy data strc when no dublcate the same element;

#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including tree_order_statistics_node_update
using namespace __gnu_pbds;
template<class T> using ordered_set = tree<T, null_type , less_equal<T> , rb_tree_tag , tree_order_statistics_node_update> ;
struct ordered__set{

   ordered_set< ll > se ;
   void erase( ll val ){
      if( se.size() == 0 || *se.find_by_order( se.size() - 1 ) < val || *se.lower_bound( val - 1 ) != val  ) return ;
```

```cpp
        se.erase( se.lower_bound( --val ) ) ;
    }
    int lower_bound( ll val ){ // log --> return index ;
        if( se.size() == 0 || *se.find_by_order( se.size() - 1 ) < val  ) return -1;
        return se.order_of_key( *se.lower_bound( --val ) ) ;
    }
    int upper_bound( ll val ){ return  lower_bound( val + 1ll ) ; }
    void insert( ll val ){ se.insert( val ); }
    ll operator[](int idx) { return *se.find_by_order( idx ) ;}
    int size( ){ return se.size(); }
    void clr(  ){ se.clear() ; }
};
```

Persistent Segment tree

PersistentSegmentTree_counter.cpp

```cpp
const int N = 1e6+9;
int n , m , id;
struct Node{
    int l , r , s;
}tree[20*N];
int newLeaf(int v){
    tree[id] = Node{-1,-1,v};
    return id++;
}
int newPar(int l , int r){
    tree[id] = Node{l,r,0};
    tree[id].s += tree[l].s;
    tree[id].s += tree[r].s;
    return id++;
}
int build(int l = 0 , int r = m){
    if(r - l == 1) return newLeaf(0);
    int mid = (l+r)/2;
    return newPar( build(l,mid) , build(mid,r) );
}
int copy(int i , int v , int x , int l = 0 , int r = m){
    if(r - l == 1) return newLeaf(tree[x].s + v);
    int mid = (l+r)/2;
    if(i < mid) return newPar( copy(i,v , tree[x].l , l , mid) , tree[x].r);
    return newPar( tree[x].l , copy(i,v, tree[x].r , mid , r) );
}
int qry(int lx , int rx , int prv , int cur , int l = 0 , int r = m){
    if(l >= lx && r <= rx) return tree[cur].s - tree[prv].s;
    if(r <= lx || l >= rx) return 0;
    int mid = (l+r)/2;
    return qry(lx,rx, tree[prv].l , tree[cur].l ,l,mid) + qry(lx,rx, tree[prv].r , tree[cur].r,mid,r);
}
```

PersistentSegmentTree_pointer.cpp

```
struct Node{
    Node* l , *r;
    int s;
    Node(int s): s(s) , l(NULL) , r(NULL){};
    Node(Node *l , Node *r): l(l) , r(r){
        this->s = 0;
        if(l != NULL) this->s += l->s;
        if(r != NULL) this->s += r->s;
    }
};
Node* newLeaf(int v){ return new Node{v}; }
Node* newPar(Node *l , Node *r){ return new Node(l,r); }
Node* build(int l = 0 , int r = m){
    if(r - l == 1) return newLeaf(0);
    int mid = (l+r)/2;
    return newPar( build(l,mid) , build(mid,r) );
}
Node* copy(int i , int v , Node* x , int l = 0 , int r = m){
    if(r-l== 1) return newLeaf(x->s + v);
    int mid = (r+l) / 2;
    if(i < mid) return newPar( copy(i,v, x->l , l , mid) , x->r );
    return newPar( x->l , copy(i,v , x->r , mid , r) );
}
int qry(int lx , int rx , Node* prv , Node* cur , int l = 0 , int r = m){
    if(l >= lx && r <= rx) return cur->s - prv->s;
    if(r <= lx || l >= rx) return 0;
    int mid = (l+r)/2;
    return qry(lx,rx,prv->l , cur->l,l,mid) + qry(lx,rx,prv->r , cur->r,mid,r);
}
```

PresistentSegmentTree_serso.cpp

```cpp
const int N = 1e6+9;
int n , m , id;
struct Node{
   int l , r , s;
}tree[20*N];
int getL(int x){ return ~x ? tree[x].l : -1; }
int getR(int x){ return ~x ? tree[x].r : -1; }
int getS(int x){ return ~x ? tree[x].s : 0; }

int newLeaf(int v){
   tree[id] = Node{-1,-1,v};
   return id++;
}
int newPar(int l , int r){
   tree[id] = Node{l,r, getS(l) + getS(r)};
   return id++;
}
int copy(int i , int v , int x , int l = 0 , int r = m){
   if(r - l == 1) return newLeaf( getS(x) + v);
   int mid = (l+r)/2;
   if(i < mid) return newPar( copy(i,v , getL(x) , l , mid) , getR(x));
   return newPar( getL(x) , copy(i,v, getR(x) , mid , r) );
}
int qry(int lx , int rx , int prv , int cur , int l = 0 , int r = m){
   if(l >= lx && r <= rx) return getS(cur) - getS(prv);
   if(r <= lx || l >= rx) return 0;
   int mid = (l+r)/2;
   return qry(lx,rx, getL(prv) , getL(cur) ,l,mid) + qry(lx,rx, getR(prv) , getR(cur) , mid , r);
}
```

SQRT and Mo's

MO-Algorithm.cpp

```cpp
const int N = 1e6 + 9, BLOCK_SIZE = 460;
void add(int idx){

}
void remove(int idx){

}
struct Query{
  int l , r , id;
  bool operator <(const Query &other) const{
    int n1 = l / BLOCK_SIZE , n2 = other.l / BLOCK_SIZE;
    if(n1 != n2) return n1 < n2;
    return n1 % 2 ? r > other.r : r < other.r;
  }
};
void Mo(vector<Query> &query){
  sort(all(query));
  int mo_l = 0 , mo_r = -1;
  for(auto &q : query){
    while(mo_l < q.l) remove(mo_l++);
    while(mo_l > q.l) add(--mo_l);
    while(mo_r > q.r) remove(mo_r--);
    while(mo_r < q.r) add(++mo_r);
    // calculate answer of Query
  }
}
```

Segment tree

SparseSegmentTree.cpp

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

class SparseSegtree {
 private:
  struct Node {
    int freq = 0;
    int lazy = 0;
    int left = -1;
    int right = -1;
  };
  vector<Node> tree;
  const int n;
  int timer = 0;

  int comb(int a, int b) { return a + b; }

```

```
    void apply(int cur, int len, int val) {
        if (val == 1) {
            tree[cur].lazy = val;
            tree[cur].freq = len * val;
        }
    }

    void push_down(int cur, int l, int r) {
        if (tree[cur].left == -1) {
            tree[cur].left = ++timer;
            tree.push_back(Node());
        }
        if (tree[cur].right == -1) {
            tree[cur].right = ++timer;
            tree.push_back(Node());
        }
        int m = (l + r) / 2;
        apply(tree[cur].left, m - l + 1, tree[cur].lazy);
        apply(tree[cur].right, r - m, tree[cur].lazy);
        tree[cur].lazy = 0;
    }

    void range_set(int cur, int l, int r, int ql, int qr, int val) {
        if (qr < l || ql > r) { return; }
        if (ql <= l && r <= qr) {
            apply(cur, r - l + 1, val);
        } else {
            push_down(cur, l, r);
            int m = (l + r) / 2;
            range_set(tree[cur].left, l, m, ql, qr, val);
            range_set(tree[cur].right, m + 1, r, ql, qr, val);
            tree[cur].freq =
                comb(tree[tree[cur].left].freq, tree[tree[cur].right].freq);
        }
    }

    int range_sum(int cur, int l, int r, int ql, int qr) {
        if (qr < l || ql > r) { return 0; }
        if (ql <= l && r <= qr) { return tree[cur].freq; }
        push_down(cur, l, r);
        int m = (l + r) / 2;
        return comb(range_sum(tree[cur].left, l, m, ql, qr),
                    range_sum(tree[cur].right, m + 1, r, ql, qr));
    }
```

```

 public:
  SparseSegtree(int n, int q = 0) : n(n) {
    if (q > 0) { tree.reserve(2 * q * __lg(n)); }
    tree.push_back(Node());
  }

  void range_set(int ql, int qr, int val) { range_set(0, 0, n - 1, ql, qr, val); }

  int range_sum(int ql, int qr) { return range_sum(0, 0, n - 1, ql, qr); }
};

int main() {
  int query_num;
  cin >> query_num;
  const int RANGE_SIZE = 1e9;
  SparseSegtree st(RANGE_SIZE + 1, query_num);

  int c = 0;
  for (int i = 0; i < query_num; i++) {
    int type, x, y;
    cin >> type >> x >> y;
    if (type == 1) {
      c = st.range_sum(x + c, y + c);
      cout << c << '\n';
    } else if (type == 2) {
      st.range_set(x + c, y + c, 1);
    }
  }
}
```

segmentTree_iterative.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
#define all(v) v.begin() , v.end()
#ifndef ONLINE_JUDGE
#define debug(x) cerr << #x << ": " << x << '\n';
#else
```

```cpp
#define debug(x)
#endif
const int N = 2e5 + 9;
int seg[4 * N];
void update(int k, int x) {
  k += N;
  seg[k] = x; // update node with value
  k >>= 1;
  while (k > 0) {
    seg[k] = max(seg[2*k], seg[2*k+1]);
    k >>= 1;
  }
}
int merge(int a, int b){
  // write code here
  return 0; // return value
}
int query(int a, int b) {
  a += N, b += N;
  int s = 0;
  while (a <= b) {
    if (a & 1) {
      s = merge(s, seg[a]);
      a++;
    }
    if (~b & 1) {
      s = merge(s, seg[b]);
      b--;
    }
    a >>= 1, b >>= 1;
  }
  return s;
}
int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
  #ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
  #endif

}
```

segtreeBeat.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

const int MAXN = 200001;  // 1-based

int N;
ll A[MAXN];

struct Node {
    ll sum;   // Sum tag
    ll max1;  // Max value
    ll max2;  // Second Max value
    ll maxc;  // Max value count
    ll min1;  // Min value
    ll min2;  // Second Min value
    ll minc;  // Min value count
    ll lazy;  // Lazy tag
} T[MAXN * 4];

void merge(int t) {
    // sum
    T[t].sum = T[t << 1].sum + T[t << 1 | 1].sum;

    // max
    if (T[t << 1].max1 == T[t << 1 | 1].max1) {
        T[t].max1 = T[t << 1].max1;
        T[t].max2 = max(T[t << 1].max2, T[t << 1 | 1].max2);
        T[t].maxc = T[t << 1].maxc + T[t << 1 | 1].maxc;
    } else {
        if (T[t << 1].max1 > T[t << 1 | 1].max1) {
            T[t].max1 = T[t << 1].max1;
            T[t].max2 = max(T[t << 1].max2, T[t << 1 | 1].max1);
            T[t].maxc = T[t << 1].maxc;
        } else {
            T[t].max1 = T[t << 1 | 1].max1;
            T[t].max2 = max(T[t << 1].max1, T[t << 1 | 1].max2);
            T[t].maxc = T[t << 1 | 1].maxc;
        }
    }

    // min
    if (T[t << 1].min1 == T[t << 1 | 1].min1) {
        T[t].min1 = T[t << 1].min1;
        T[t].min2 = min(T[t << 1].min2, T[t << 1 | 1].min2);
        T[t].minc = T[t << 1].minc + T[t << 1 | 1].minc;
    } else {
        if (T[t << 1].min1 < T[t << 1 | 1].min1) {
            T[t].min1 = T[t << 1].min1;
            T[t].min2 = min(T[t << 1].min2, T[t << 1 | 1].min1);
            T[t].minc = T[t << 1].minc;
        } else {
            T[t].min1 = T[t << 1 | 1].min1;
            T[t].min2 = min(T[t << 1].min1, T[t << 1 | 1].min2);
            T[t].minc = T[t << 1 | 1].minc;
        }
    }
}

void push_add(int t, int tl, int tr, ll v) {
    if (v == 0) { return; }
    T[t].sum += (tr - tl + 1) * v;
    T[t].max1 += v;
    if (T[t].max2 != -llINF) { T[t].max2 += v; }
    T[t].min1 += v;
    if (T[t].min2 != llINF) { T[t].min2 += v; }
    T[t].lazy += v;
}

// corresponds to a chmin update
void push_max(int t, ll v, bool l) {
    if (v >= T[t].max1) { return; }
    T[t].sum -= T[t].max1 * T[t].maxc;
    T[t].max1 = v;
    T[t].sum += T[t].max1 * T[t].maxc;
    if (l) {
        T[t].min1 = T[t].max1;
    } else {
        if (v <= T[t].min1) {
            T[t].min1 = v;
        } else if (v < T[t].min2) {
            T[t].min2 = v;
        }
    }
}
```

```

// corresponds to a chmax update
void push_min(int t, ll v, bool l) {
  if (v <= T[t].min1) { return; }
  T[t].sum -= T[t].min1 * T[t].minc;
  T[t].min1 = v;
  T[t].sum += T[t].min1 * T[t].minc;
  if (l) {
    T[t].max1 = T[t].min1;
  } else {
    if (v >= T[t].max1) {
      T[t].max1 = v;
    } else if (v > T[t].max2) {
      T[t].max2 = v;
    }
  }
}

void pushdown(int t, int tl, int tr) {
  if (tl == tr) return;
  // sum
  int tm = (tl + tr) >> 1;
  push_add(t << 1, tl, tm, T[t].lazy);
  push_add(t << 1 | 1, tm + 1, tr, T[t].lazy);
  T[t].lazy = 0;

  // max
  push_max(t << 1, T[t].max1, tl == tm);
  push_max(t << 1 | 1, T[t].max1, tm + 1 == tr);

  // min
  push_min(t << 1, T[t].min1, tl == tm);
  push_min(t << 1 | 1, T[t].min1, tm + 1 == tr);
}

void build(int t = 1, int tl = 0, int tr = N - 1) {
  T[t].lazy = 0;
  if (tl == tr) {
    T[t].sum = T[t].max1 = T[t].min1 = A[tl];
    T[t].maxc = T[t].minc = 1;
    T[t].max2 = -llINF;
    T[t].min2 = llINF;
    return;
```

```cpp
    }

    int tm = (tl + tr) >> 1;
    build(t << 1, tl, tm);
    build(t << 1 | 1, tm + 1, tr);
    merge(t);
}

void update_add(int l, int r, ll v, int t = 1, int tl = 0, int tr = N - 1) {
    if (r < tl || tr < l) { return; }
    if (l <= tl && tr <= r) {
        push_add(t, tl, tr, v);
        return;
    }
    pushdown(t, tl, tr);

    int tm = (tl + tr) >> 1;
    update_add(l, r, v, t << 1, tl, tm);
    update_add(l, r, v, t << 1 | 1, tm + 1, tr);
    merge(t);
}

void update_chmin(int l, int r, ll v, int t = 1, int tl = 0, int tr = N - 1) {
    if (r < tl || tr < l || v >= T[t].max1) { return; }
    if (l <= tl && tr <= r && v > T[t].max2) {
        push_max(t, v, tl == tr);
        return;
    }
    pushdown(t, tl, tr);

    int tm = (tl + tr) >> 1;
    update_chmin(l, r, v, t << 1, tl, tm);
    update_chmin(l, r, v, t << 1 | 1, tm + 1, tr);
    merge(t);
}

void update_chmax(int l, int r, ll v, int t = 1, int tl = 0, int tr = N - 1) {
    if (r < tl || tr < l || v <= T[t].min1) { return; }
    if (l <= tl && tr <= r && v < T[t].min2) {
        push_min(t, v, tl == tr);
        return;
    }
    pushdown(t, tl, tr);
```

```

  int tm = (tl + tr) >> 1;
  update_chmax(l, r, v, t << 1, tl, tm);
  update_chmax(l, r, v, t << 1 | 1, tm + 1, tr);
  merge(t);
}

ll query_sum(int l, int r, int t = 1, int tl = 0, int tr = N - 1) {
  if (r < tl || tr < l) { return 0; }
  if (l <= tl && tr <= r) { return T[t].sum; }
  pushdown(t, tl, tr);

  int tm = (tl + tr) >> 1;
  return query_sum(l, r, t << 1, tl, tm) +
      query_sum(l, r, t << 1 | 1, tm + 1, tr);
}

int main() {
  int Q;

  cin >> N >> Q;
  for (int i = 0; i < N; i++) { cin >> A[i]; }
  build();
  for (int q = 0; q < Q; q++) {
    int t;
    cin >> t;
    if (t == 0) {
      int l, r;
      ll x;
      cin >> l >> r >> x;
      update_chmin(l, r - 1, x);
    } else if (t == 1) {
      int l, r;
      ll x;
      cin >> l >> r >> x;
      update_chmax(l, r - 1, x);
    } else if (t == 2) {
      int l, r;
      ll x;
      cin >> l >> r >> x;
      update_add(l, r - 1, x);
    } else if (t == 3) {
      int l, r;
```

segtreeBeat_forMod.cpp

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

const int MAXN = 100001;

int N, Q;
long long tsum[MAXN * 4], tmax[MAXN * 4];

void update_mod(int l, int r, long long v, int t = 1, int tl = 1, int tr = N) {
    if (r < tl || tr < l || tmax[t] < v) {
        return;
    } else if (tl == tr) {
        int val = tmax[t] % v;
        tsum[t] = tmax[t] = val;
        return;
    }

    int tm = (tl + tr) / 2;
    update_mod(l, r, v, t * 2, tl, tm);
    update_mod(l, r, v, t * 2 + 1, tm + 1, tr);
    tsum[t] = tsum[t * 2] + tsum[t * 2 + 1];
    tmax[t] = max(tmax[t * 2], tmax[t * 2 + 1]);
}

void update_set(int i, long long v, int t = 1, int tl = 1, int tr = N) {
    if (tl == tr) {
        tsum[t] = tmax[t] = v;
        return;
    }

    int tm = (tl + tr) / 2;
    if (i <= tm) {
        update_set(i, v, t * 2, tl, tm);
    } else {
        update_set(i, v, t * 2 + 1, tm + 1, tr);
    }
    tsum[t] = tsum[t * 2] + tsum[t * 2 + 1];
    tmax[t] = max(tmax[t * 2], tmax[t * 2 + 1]);
}

long long query(int l, int r, int t = 1, int tl = 1, int tr = N) {
  if (r < tl || tr < l) {
    return 0;
  } else if (l <= tl && tr <= r) {
    return tsum[t];
  }

  int tm = (tl + tr) / 2;
  return query(l, r, t * 2, tl, tm) + query(l, r, t * 2 + 1, tm + 1, tr);
}

int main() {
  cin >> N >> Q;
  for (int i = 1; i <= N; i++) {
    long long a;
    cin >> a;
    update_set(i, a);
  }
  for (int q = 0; q < Q; q++) {
    int t;
    cin >> t;
    if (t == 1) {
      int l, r;
      cin >> l >> r;
      cout << query(l, r) << '\n';
    } else if (t == 2) {
      int l, r;
      long long x;
      cin >> l >> r >> x;
      update_mod(l, r, x);
    } else if (t == 3) {
      int i;
      long long x;
      cin >> i >> x;
      update_set(i, x);
    }
  }
}
```

Sparse table

getIdxSparseTable.cpp

```cpp
struct SparseTable {
  vector<ll> A;
  vector<int> log;
  vector<vector<pair<ll, int>>> spt;

  void init(vector<ll> &a) {
    int n = a.size();
    A = a;
    log.assign(n + 1, 0);
    for (int i = 2; i <= n; i++) {
      log[i] = 1 + log[i / 2];
    }
    int k = log[n] + 1;
    spt = vector<vector<pair<ll, int>>>(k, vector<pair<ll, int>>(n));
    for (int i = 0; i < n; i++) {
      spt[0][i] = { A[i], i };
    }
    for (int j = 1; 1 << j <= n; j++) {
      for (int i = 0; i + (1 << j) - 1 < n; i++) {
        spt[j][i] = merge(spt[j - 1][i], spt[j - 1][i + (1 << (j - 1))]);
      }
    }
  }

  pair<ll, int> merge(pair<ll, int> &x, pair<ll, int> &y) {
    // choose x or y
  }

  pair<ll, int> query(int i, int j) {
    int len = j - i + 1;
    int k = log[len];
    return merge(spt[k][i], spt[k][j - (1 << k) + 1]);
  }
};
```

sparse_segmentTree.cpp

```cpp
#include <bits/stdc++.h>
#pragma GCC optimize("O3")
#define FOR(i, x, y) for (int i = x; i < y; i++)
```

```cpp
#define MOD 1000000007
typedef long long ll;
using namespace std;

struct Node {
    int sum, lazy, tl, tr, l, r;
    Node() : sum(0), lazy(0), l(-1), r(-1) {}
};

const int MAXN = 123456;
Node segtree[64 * MAXN];
int cnt = 2;

void push_lazy(int node) {
    if (segtree[node].lazy) {
        segtree[node].sum = segtree[node].tr - segtree[node].tl + 1;
        int mid = (segtree[node].tl + segtree[node].tr) / 2;
        if (segtree[node].l == -1) {
            segtree[node].l = cnt++;
            segtree[segtree[node].l].tl = segtree[node].tl;
            segtree[segtree[node].l].tr = mid;
        }
        if (segtree[node].r == -1) {
            segtree[node].r = cnt++;
            segtree[segtree[node].r].tl = mid + 1;
            segtree[segtree[node].r].tr = segtree[node].tr;
        }
        segtree[segtree[node].l].lazy = segtree[segtree[node].r].lazy = 1;
        segtree[node].lazy = 0;
    }
}

void update(int node, int l, int r) {
    push_lazy(node);
    if (l == segtree[node].tl && r == segtree[node].tr) {
        segtree[node].lazy = 1;
        push_lazy(node);
    } else {
        int mid = (segtree[node].tl + segtree[node].tr) / 2;
        if (segtree[node].l == -1) {
            segtree[node].l = cnt++;
            segtree[segtree[node].l].tl = segtree[node].tl;
            segtree[segtree[node].l].tr = mid;
        }
        if (segtree[node].r == -1) {
            segtree[node].r = cnt++;
            segtree[segtree[node].r].tl = mid + 1;
            segtree[segtree[node].r].tr = segtree[node].tr;
        }

        if (l > mid) update(segtree[node].r, l, r);
        else if (r <= mid) update(segtree[node].l, l, r);
        else {
            update(segtree[node].l, l, mid);
            update(segtree[node].r, mid + 1, r);
        }

        push_lazy(segtree[node].l);
        push_lazy(segtree[node].r);
        segtree[node].sum =
            segtree[segtree[node].l].sum + segtree[segtree[node].r].sum;
    }
}

int query(int node, int l, int r) {
    push_lazy(node);
    if (l == segtree[node].tl && r == segtree[node].tr)
        return segtree[node].sum;
    else {
        int mid = (segtree[node].tl + segtree[node].tr) / 2;
        if (segtree[node].l == -1) {
            segtree[node].l = cnt++;
            segtree[segtree[node].l].tl = segtree[node].tl;
            segtree[segtree[node].l].tr = mid;
        }
        if (segtree[node].r == -1) {
            segtree[node].r = cnt++;
            segtree[segtree[node].r].tl = mid + 1;
            segtree[segtree[node].r].tr = segtree[node].tr;
        }

        if (l > mid) return query(segtree[node].r, l, r);
        else if (r <= mid) return query(segtree[node].l, l, r);
        else
            return query(segtree[node].l, l, mid) +
                   query(segtree[node].r, mid + 1, r);
    }
```

```cpp
}

int main() {
    iostream::sync_with_stdio(false);
    cin.tie(0);
    int m;
    cin >> m;

    segtree[1].sum = 0;
    segtree[1].lazy = 0;
    segtree[1].tl = 1;
    segtree[1].tr = 1e9;

    int c = 0;
    FOR(_, 0, m) {
        int d, x, y;
        cin >> d >> x >> y;
        if (d == 1) {
            c = query(1, x + c, y + c);
            cout << c << '\n';
        } else update(1, x + c, y + c);
    }
```

sparse_table.cpp

```cpp
template<class T>
struct Sparetable{
    vector<vector<T>> v;
    int n , LOG;
    void init(vector<T> &a){
        this-> n = (int) a.size();
        this->LOG = 0;
        int size = 1;
        while(size <= n) size *= 2 , LOG++;
        v.assign(n , vector<T>(LOG));
        for (int i = 0; i < n; i++)v[i][0] = a[i];
        for (int j = 1; (1 << j) <= n; j++){
            for (int i = 0; (i + (1 << j) - 1) < n; i++){
                v[i][j] = merge(v[i][j - 1] , v[i + (1 << (j - 1))][j - 1]);
            }
        }
    }
    T merge(T a , T b){
        return min(a , b); // change the operation
    }
    T qry(int l, int r){
        int len = r - l + 1;
        int j = 31 - __builtin_clz(len);
        T res = merge(v[l][j] , v[r - (1 << j) + 1][j]);
        return res; // determine what you want to return
    }
};
```

Treap

Treap Builtin.cpp

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

/// Importing policy_based_data_structure:
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
/// Importing ends here.

struct vals{ /// struct for declaring struct type pb_ds:
    int num;
    int typ;
    vals(int a, int b){
        num = a;
        typ = b;
    }
    bool operator <(const vals& other) const {
        return num > other.num;
    }
};

/// Supports all the operations of a set including two additional features:
/// 1. find_by_order(k) # Returns an iterator pointing to the k-th smallest element (zero based).
/// 2. order_of_key(k) # Returns the number of elements strictly smaller than k.

```

```cpp
int main(){
    /// typedef original name to pb_ds for simplicity:
    typedef tree <int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update> pb_ds;
    /// Ordered Set
    pb_ds treap;

    treap.insert(2); /// Insert an element
    treap.insert(3);

    cout << treap.order_of_key(5) << endl; /// Returns number of elements smaller than k

    pb_ds::iterator it = treap.find_by_order(0); /// Returns an iterator pointing to the k-th smallest element
    cout << *it << endl; /// Print the element

    /// Iterate though the elements (similar to a set)
    for(pb_ds::iterator it = treap.begin(); it != treap.end(); it++){
        cout << *it << endl;
    }

    treap.erase(2); /// Erase an element
    treap.clear(); /// Clear the treap

    /// Struct type pb_ds. Ordering depends on the operator overloading inside the struct.
    typedef tree <vals, null_type, less<vals>, rb_tree_tag, tree_order_statistics_node_update> pb_ds_st;

    /// Ordered Multiset. Notice the less_equal<int> parameter.
    typedef tree<int, null_type, less_equal<int>, rb_tree_tag, tree_order_statistics_node_update> ordered_multiset;

    ordered_multiset tmset;
    tmset.insert(2);
    tmset.insert(2);
    /// Erasing is very tricky in Ordered Multiset
    tmset.erase(tmset.find_by_order(tmset.order_of_key(2))); /// Erase 2 (Just one of the 2's get erased)

    /// Pair <int,int> type Ordered Set
    typedef tree<pair<int, int>, null_type, less_equal<pair<int, int>>, rb_tree_tag, tree_order_statistics_node_update> ordered_pair;

```

Treap Implicit.cpp

```cpp
/// Implicit Treap Implementation
/// Can handle most of the operations we do in segment tree
/// Ex: Range update, range query
/// Additionally also handles insert or erase at any position, reverse a range
/// Call clear() to clear the treap, then use each function carefully following the comments
/// If there is propagation, uncomment propagate() inside the functions
/// Note 1: All the operations are zero based
/// Note 2: Remember to propagate if you try to access the treap nodes outside split and merge

struct node{
    int size, prior;
    int sum, prop, key, mnn;
    bool rev;
    struct node *l, *r;
    node() { }
    node(int v) {
        key = v;
        prior = rand();
        size = 1;
        l = r = NULL;
        sum = prop = 0;
        mnn = inf;
    }
    node (int key, int prior) : key(key), prior(prior), l(NULL), r(NULL) { }
};

typedef node* pnode;

struct Treap{
    pnode t;
    Treap(){}

    /// Returns size of treap
    int size(){
        return sz(t);
    }

```

```
int sz(pnode t){
    return t ? t->size:0;
}

int sum(pnode t){
    return t ? t->sum:0;
}

int mnn(pnode t){
    return t ? t->mnn:inf;
}

void upd_node(pnode t){
    if(t){
        t->size = sz(t->l) + 1 + sz(t->r);
        //t->sum = sum(t->l) + t->key + sum(t->r); /// If you need sum
        t->mnn = min(t->key, min(mnn(t->l), mnn(t->r)));
    }
}

/// Works like segment tree propagation
void propagate(pnode t){
    if(!t) return;
    if(t->prop>0){ /// Propagate range addition
        if(t->l){
            t->l->key += t->prop;
            t->l->sum += sz(t->l)*t->prop;
            t->l->prop += t->prop;
            t->l->mnn += t->prop;
        }
        if(t->r){
            t->r->key += t->prop;
            t->r->sum += sz(t->r)*t->prop;
            t->r->prop += t->prop;
            t->r->mnn += t->prop;
        }
        t->prop = 0;
    }
    if(t->rev){ /// Propagate range reverse
        swap(t->l, t->r);
        if(t->l) t->l->rev ^= true;
        if(t->r) t->r->rev ^= true;
        t->rev = false;
    }
}

/// Split t into l and r such that all elements in l is < key and
/// all elements in r is >= than key
void split(pnode t, pnode &l, pnode &r, int key, int add = 0){
    if(!t){
        l = r = NULL;
        return;
    }
    propagate(t);
    int cur_key = add + sz(t->l);
    if(cur_key < key)
        split(t->r, t->r, r, key, add + 1 + sz(t->l)), l = t;
    else
        split(t->l, l, t->l, key, add), r = t;
    upd_node(t);
}

/// Merge l and r into t, where all elements in l
/// is less than all elements in r
void merge(pnode &t, pnode l, pnode r){
    propagate(l);
    propagate(r);
    if(!l || !r) t = l ? l:r;
    else if(l->prior > r->prior) merge(l->r, l->r, r), t = l;
    else merge(r->l, l, r->l), t = r;
    upd_node(t);
}

void insert(pnode &t, int pos, pnode it){
    pnode l, r, tmp;
    split(t, l, r, pos);
    merge(tmp, l, it);
    merge(t, tmp, r);
    upd_node(t);
}

void insertEnd(pnode &t, pnode it){
    pnode l, r, tmp;
    merge(t, t, it);
    upd_node(t);
}

```

```
void erase(pnode &t, int key){
    pnode t1, t2, nt1, nt2;
    split(t, t1, t2, key+1);
    split(t1, nt1, nt2, key);
    merge(t, nt1, t2);
    upd_node(t);
    free(nt2);
}

int get(pnode &t, int key, int add = 0){
    if(!t) return 0;
    propagate(t);
    int cur_key = add + sz(t->l);
    if(cur_key == key){
        return t->key;
    }else{
        if(cur_key < key) return get(t->r, key, add + 1 + sz(t->l));
        else return get(t->l, key, add);
    }
    upd_node(t);
}

void print(pnode t){
    if(!t) return;
    propagate(t);
    print(t->l);
    cerr << t->key << " ";
    print(t->r);
}

void nullify(pnode t){
    if(t == NULL) return;
    nullify(t->l); nullify(t->r);
    delete t;
    t->l = NULL; t->r = NULL; t = NULL;
    free(t);
}

/// Insert val at position p in the treap
void insert(int p, int val){
    pnode it = new node(val);
    insert(t, p, it);
}
```

```

    /// Insert val at the end of the treap
    void insertEnd(int val){
        pnode it = new node(val);
        insertEnd(t, it);
    }

    /// Erase the element at p from the treap
    void erase(int p){
        erase(t, p);
    }

    /// Returns the value at position p
    int get(int p){
        return get(t, p);
    }

    /// Print all the elements in treap in sorted order
    void print(){
        cerr<<"\nPRINT TREAP: ";
        print(t);
        cerr<<" \n";
    }

    /// Clear the treap
    void clear(){
        nullify(t);
        t = NULL;
    }

    /// Get the minimum in range u to v
    int getRangeMin(int u, int v){
        pnode tv, tvn, tu, tuv;
        split(t, tv, tvn, v+1);
        split(tv, tu, tuv, u);

        int res = min(tuv->key, min(mnn(tuv->l), mnn(tuv->r)));
        merge(tv, tu, tuv);
        merge(t, tv, tvn);
        return res;
    }

    /// Get the sum of range u to v
```

```
int getRangeSum(int u, int v){
    pnode tv, tvn, tu, tuv;
    split(t, tv, tvn, v+1);
    split(tv, tu, tuv, u);

    int res = tuv->sum;
    merge(tv, tu, tuv);
    merge(t, tv, tvn);
    return res;
}

/// Rotate(right) the range from u to v k times
void updateRangeRotate(int u, int v, int k){
    pnode tv, tvn, tu, tuv;
    split(t, tv, tvn, v+1);
    split(tv, tu, tuv, u);

    int len = v - u + 1;
    k %= len;

    pnode tuv1, tuv2;
    split(tuv, tuv1, tuv2, len-k);

    merge(tuv, tuv2, tuv1);
    merge(tv, tu, tuv);
    merge(t, tv, tvn);
}

/// Reverse the range from u to v
void updateRangeReverse(int u, int v){
    pnode tv, tvn, tu, tuv;
    split(t, tv, tvn, v+1);
    split(tv, tu, tuv, u);

    tuv->rev ^= true;

    merge(tv, tu, tuv);
    merge(t, tv, tvn);
}

/// Add val to each node in range u to v
void updateRangeAdd(int u, int v, int val){
        pnode tv, tvn;
        split(t, tv, tvn, v+1);

        pnode tu, tuv;
        split(tv, tu, tuv, u);

        tuv->key += val;
        tuv->sum += sz(tuv)*val;
```

Treap.cpp

```
/// Treap which support multiple entry, works like a multiset
/// If you want to use like a set then just erase the element before insert
/// All the functions are similar to the built in treap
/// Every function works in log(N) except unite()

struct node{
    int key, prior, size;
    struct node *l, *r;
    node() { }
    node(int v) {
        key = v;
        prior = rand();
        size = 1;
        l = r = NULL;
    }
    node (int key, int prior) : key(key), prior(prior), l(NULL), r(NULL) { }
};

typedef node* pnode;

struct Treap{
    pnode t;
    Treap(){}

    /// Returns size of treap
    int size(){
        return sz(t);
    }

    int sz(pnode t){
```

```
    return t ? t->size:0;
}

void upd_sz(pnode t){
    if(t) t->size = sz(t->l) + 1 + sz(t->r);
}

/// Split t into l and r such that all elements in l
/// is less than key and all elements in r is greater than key
void split(pnode t, pnode &l, pnode &r, int key){
    if(!t) l = r = NULL;
    else if(t->key < key) split(t->r, t->r, r, key), l = t;
    else split(t->l, l, t->l, key), r = t;
    upd_sz(t);
}

/// Merge l and r into t, where all elements in l
/// is less than all elements in r
void merge(pnode &t, pnode l, pnode r){
    if(!l || !r) t = l ? l:r;
    else if(l->prior > r->prior) merge(l->r, l->r, r), t = l;
    else merge(r->l, l, r->l), t = r;
    upd_sz(t);
}

/// Unite two different treap l and r into a new treap
/// Complexity O(N)
pnode unite (pnode l, pnode r) {
    if (!l || !r)  return l ? l : r;
    if (l->prior < r->prior)  swap (l, r);
    pnode lt, rt;
    split (r, lt, rt, l->key);
    l->l = unite (l->l, lt);
    l->r = unite (l->r, rt);
    return l;
}

void insert(pnode &t, pnode it){
    if(!t) t = it;
    else if(it->prior > t->prior) split(t, it->l, it->r, it->key), t = it;
    else insert(t->key < it->key ? t->r:t->l, it);
    upd_sz(t);
}

```

```
    void erase(pnode &t, int key){
        if(!t) return;
        else if(t->key == key){
            pnode temp = t; merge(t, t->l, t->r); free(temp);
        }else{
            erase(t->key < key ? t->r:t->l,key);
        }
        upd_sz(t);
    }

    void init(pnode &t, int c){
        t->prior = rand(); t->size = 1;  t->l = t->r = NULL;
        t->key = c;
    }

    void print(pnode t){
        if(!t) return;
        print(t->l);
        cerr << t->key << " " << endl;
        print(t->r);
    }

    int getKth(pnode temp, int par, int k){
        if(temp == NULL) return 0;
        int currSize = par + sz(temp->l)  + 1;
        if(currSize == k) return temp->key;

        else if(currSize <= k) return getKth(temp->r, currSize, k);
        else return getKth(temp->l, par, k);
    }

    int orderOf(pnode temp, int k){
        int x = 0;
        if(temp == NULL) return 0;
        if(temp->key < k) return sz(temp->l) + 1 + orderOf(temp->r, k);
        else return orderOf(temp->l, k);
    }

    void nullify(pnode t){
        if(t == NULL) return;
        nullify(t->l); nullify(t->r);
        delete t;
        t->l = NULL; t->r = NULL; t = NULL;
        free(t);
    }

    /// Insert k in the treap
    void insert(int k){
        pnode it = new node(k);
        insert(t, it);
    }

    /// Erase k for the treap
    void erase(int k){
        erase(t, k);
    }

    /// Returns the k'th smallest element in treap(0 based)
    int find_by_order(int k){
        return getKth(t, 0, k+1);
    }

    /// Returns number of elements less than k
    int order_of_key(int k){
        return orderOf(t, k);
    }

    /// Print all the elements in treap in sorted order
    void print(){
        cerr<<"\nPRINT TREAP: ";
        print(t);
        cerr<<" \n\n";
    }

    /// Clear the treap
    void clear(){
        nullify(t);
        t = NULL;
    }
```

Xor Basis

combining_two_xor_basis.cpp

```cpp
struct Basis{
  int basis[LOG];
  Basis(){ memset(basis ,0, sizeof basis); }
  void insert(int x){
    for(int i = LOG - 1; i >= 0; --i){
      if(!(x >> i & 1)) continue;
      if(basis[i] == 0){
        basis[i] = x;
        return;
      }
      x ^= basis[i];
    }
  }
  void insert(Basis &other){
    for(int i = LOG - 1; i >= 0; --i) if(other.basis[i]){
      insert(other.basis[i]);
    }
  }
  int max_xor(){
    int x = 0;
    for(int i = LOG - 1; i >= 0; --i){
      if((x >> i & 1) || basis[i] == 0) continue;
      x ^= basis[i];
    }
    return x;
  }
  void reset(){ memset(basis ,0, sizeof basis); }
}
```

minOrMaxXorPathFrom1ToN.cpp

```cpp
/*
Given an undirected connected graph with non-negative integer edge weights and node numbers from 1
 to N, find a path from node 1 to node N such that the XOR of the weights of the edges along the path is maximized.
A path can pass through certain nodes or edges repeatedly. This means you are allowed to revisit the same node more than once, and if an edge is traversed multiple times, its weight must be included in the XOR each time it is used.
The following M lines each contain three integers u, v, and w — representing an edge between nodes u and v with a weight w.
*/
```

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define f first
#define s second
const int N = 1e5 , LOG = 30;
int Basis[N];
void insert(int x){
    for(int i = LOG - 1; i >= 0; --i){
        if(!(x >> i & 1)) continue;
        if(Basis[i] == 0){
            Basis[i] = x;
            return;
        }
        x ^= Basis[i];
    }
}
int min_xor(int x){
    for(int i = LOG - 1; i >= 0; --i){
        if( (x ^ Basis[i]) < x) x ^= Basis[i];
    }
    return x;
}

int a[N];
bool vis[N];
vector<pair<int,int>> adj[N];
void dfs(int u , int p , int xor_sum){
    if(vis[u]){
        insert(xor_sum ^ a[u]);
        return;
    }
    vis[u] = 1;
    a[u] = xor_sum;
    for(auto &[v , w] : adj[u]) if(v != p){
        dfs(v , u , xor_sum ^ w);
    }
}
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    int n , m; cin >> n >> m;
    for(int i = 0; i < m; ++i){
        int u , v , w; cin >> u >> v >> w;
        --u; --v;
        adj[u].emplace_back(v , w);
        adj[v].emplace_back(u , w);
    }
    dfs(0,-1,0);
    cout << min_xor(a[n - 1]); // u can replace it with max xor
```

xor_basis_application.cpp

```cpp
const int LOG = 30; // log(max(a))
int Basis[LOG] , sz;
void insert(int x){
    /*
        Insert vector into basis
    */
    for(int i = LOG - 1; i >= 0; --i){
        if(!(x >> i & 1)) continue;
        if(Basis[i] == 0){
            Basis[i] = x;
            ++sz;
            return;
        }
        x ^= Basis[i];
    }
}
int max_xor(){
    /*
        Return the maximum xor_sum over all different subsequence
    */
    int x = 0;
    for(int i = LOG - 1; i >= 0; --i){
        if(x >> i & 1) continue;
        x ^= Basis[i];
    }
    return x;
}
bool check(int x){
    /*
        Check if there is a subsequence that xor_sum = x
    */
    for(int i = 0; i < LOG; ++i){
        if(!((x >> i) & 1)) continue;
        if(Basis[i] == 0) return false;
        x ^= Basis[i];
    }
    return true;
}
int k_th(int k){
    /*
        Finding the k-th smallest xor_sum of all different subsequence xor_sum
    */
    int low = 1 << sz;
    int x = 0;
    for(int i = LOG - 1; i >= 0; --i){
        if(!Basis[i]) continue;
        low /= 2;
        if( (!(x >> i & 1) && low < k) || ((x >> i & 1) && low >= k) ){
            x ^= Basis[i];
        }
        if(low < k) k -= low;
    }
    return x;
}
int count(int x){
     if(!check(x)) return false;
     return pow(2 , n - sz); // where n is the total size of array and sz is the size of basis
}
```

xor_basis_lexicographically_largest.cpp

```cpp
const int LOG = 30;
struct Basis{
    int basis[LOG];
    int lt[LOG];
    Basis(){
        memset(basis,0,sizeof basis);
        memset(lt,-1,sizeof lt);
    }
    void insert(int x , int ind){
        for(int i = LOG - 1; i >= 0; --i){
            if(!(x >> i & 1)) continue;
            if(lt[i] == -1){
                lt[i] = ind;
                basis[i] = x;
                return;
            }
            if(lt[i] < ind){
                swap(lt[i] , ind);
                swap(basis[i] , x);
            }
            x ^= basis[i];
        }
    }
    int max_xor(int ind){
        int x = 0;
        for(int i = LOG - 1; i >= 0; --i){
            if((x >> i & 1) || (lt[i] < ind) ) continue;
            x ^= basis[i];
        }
        return x;
    }
    void reset(){ memset(lt , -1,sizeof lt); memset(basis , 0 , sizeof basis); }
}
```

Graph

01_BFS.cpp

```
/*
    ########### 0-1 BFS ###########
    Optimzed algrothim from dikjstra can used when weight is (0 , 1)
    Time complexity: O(n + m)
*/
// 0-1 BFS
deque<int> q; //{x , y}
vector<int>> dis(n , INT32_MAX);
q.push_front();
dis[] = 0;
while(!q.empty()){
// if new weight incearse by 1 >> push_back in deque
// if new weight still the same >> push_front in deque
}
```

BFS.cpp

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

```
typedef long long ll;
#define s second
#define f first
using namespace std;
int main(){
 /*
  ------ BFS Algrothim --------
  - use to Find Shortest path from Single Sourse to other vertices
  - can used if weight of edge == 1
  - Find answer in O(n + m)
 */
 int n , m; cin >> n >> m; // n : number of vertices , m : number of edges
 vector<int> adj[n];
 for(int i = 0 ; i < m ; i++){
  int u , v; cin >> u >> v;
  adj[u].emplace_back(v);
  adj[v].emplace_back(u);
 }
 int Start; cin >> Start; // vertice that you start from
 Start--;
 queue<int> q;
 vector<int> dis(n , -1);
 vector<int> par(n , -1);
 dis[Start] = 0; q.push(Start);
 while(q.size()){
  int u = q.front(); q.pop();
  for(auto &v : adj[u]){
   if(!~dis[v]){
    dis[v] = dis[u] + 1;
    q.push(v);
    par[v] = u;
   }
  }
 }
 // Find the path
 vector<int> path;
 function<void(int)> gen = [&](int i){
  path.emplace_back(i);
  if(~par[i]) gen(par[i]);
 };
 reverse(path.begin() , path.end());
}
```

Bridges.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
const int N = 1e5 + 9;
vector<vector<pair<int,int>>> adj;
vector<vector<int>> BridgeTree;
vector<int> lowLink , dfn , isBridge , comp;
int  ndfn , comp_num , total;
void tarjan(int u , int par){
 dfn[u] = lowLink[u] = ndfn++;
 for(auto &[v , id] : adj[u]){
  if(dfn[v] == -1){
   tarjan(v , u);
   lowLink[u] = min(lowLink[u] , lowLink[v]);
    if(lowLink[v] == dfn[v]){
      isBridge[id] = true;
      total++;
    }
  }else if(v != par){
   lowLink[u] = min(lowLink[u] , dfn[v]);
  }
 }
}
void Find_component(int u , int par){
  comp[u] = comp_num;
  for(auto &[v , id] : adj[u]) if(comp[v] == -1 && isBridge[id] == 0)
    Find_component(v , u);
}
pair<int, int> diameter(int u, int par = -1)
{
  int diam = 0;
  int mxHeights[3] = {-1, -1, -1};   // keep 2 highest trees
  for(auto &v : BridgeTree[u]) if(v != par)
  {
    auto p = diameter(v , u);
    diam = max(diam, p.f);
    mxHeights[0] = p.s+1;
    sort(mxHeights, mxHeights+3);
    }
    for(int i = 0; i < 3; i++)if(mxHeights[i] == -1)
        mxHeights[i] = 0;
    diam = max(diam, mxHeights[1] + mxHeights[2]);
    return {diam, mxHeights[2]};
}
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    int q; cin >> q;
    while(q--){
        int n, m; cin >> n >> m;
        // reset
        dfn.assign(n , -1);
        comp.assign(n , -1);
        lowLink.assign(n , 0);
        isBridge.assign(m , 0);
        ndfn = comp_num = total = 0;
        adj.assign(n , vector<pair<int,int>>());

        for(int i = 0; i < m; i++){
            int u , v; cin >> u >> v;
            --u; --v;
            adj[u].emplace_back(v,  i);
            adj[v].emplace_back(u , i);
        }
        // Finding Bridges using Tarjan algo.
        tarjan(0 , 0);
        // dfs to group all the maximal components together, so that we can shrink it to one node
        for(int i = 0; i < n; i++) if(comp[i] == -1){
            Find_component(i , i);
            comp_num++;
        }
        // shrinking all the maximal components to one node
        BridgeTree.assign(comp_num , vector<int>());
        for(int u = 0; u < n; u++) for(auto &[v , id] : adj[u]) if(isBridge[id]){
            BridgeTree[comp[u]].emplace_back(comp[v]);
        }
        // Finding the diameter of the Bridgestree
        int d = diameter(0 , 0).f;
    }
}
```

Hierholzer.cpp

```
/*
  # Hierholzer's Algorithm for directed graph

  Euler circuit is a path that traverses every edge of a graph, and the path ends on the starting vertex
  Problem: Given a directed Eulerian graph, print an Euler circuit

  restrictions:
  A directed graph has an eulerian cycle if following conditions are true
    1. All vertices with nonzero degree belong to a single strongly connected component.
    2. In degree is equal to the out degree for every vertex.

  Idea:
  Choose any starting vertex v, and follow a trail of edges from that vertex until returning to v. It is not possible to get stuck at any vertex other than v,
  because indegree and outdegree of every vertex must be same, when the trail enters another vertex w there must be an unused edge leaving w.
  The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.
  As long as there exists a vertex u that belongs to the current tour, but that has adjacent edges not part of the tour,
  start another trail from u, following unused edges until returning to u, and join the tour formed in this way to the previous tour.


  Time complexity : O(V+E)
  Space complexity : O(V+E)
*/


// Don't forget to check if the graph has an euler circuite or not.
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
```

```
#define all(v) v.begin() , v.end()
#ifndef ONLINE_JUDGE
#define debug(x) cerr << #x << ": " << x << '\n';
#else
#define debug(x)
#endif
const int N = 1e5 + 9;
vector<int> adj[N];
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    int n , m; cin >> n >> m;
    for(int i = 0; i < m; ++i){
        int u , v; cin >> u >> v;
        --u; --v;
        adj[u].emplace_back(v);
    }
    vector<int> edge_count(n);
    for (int i= 0; i< n; i++) edge_count[i] = adj[i].size();
    stack<int> cur_path;  // Maintain a stack to keep vertices
    vector<int> circuit;  // vector to store final circuit
    cur_path.push(0);  // start from any vertex
    int cur_v = 0; // curent vertex
    while (!cur_path.empty()){
        if (edge_count[cur_v]){  // If there's remaining edge
            cur_path.push(cur_v); // Push the vertex
            int next_v = adj[cur_v].back(); // Find the next vertex using an edge
            edge_count[cur_v]--; // and remove that edge
            adj[cur_v].pop_back();
            cur_v = next_v; // Move to next vertex
        }

        else{ // back-track to find remaining circuit
            circuit.push_back(cur_v);
            // Back-tracking
            cur_v = cur_path.top();
            cur_path.pop();
        }
    }

```

```
  // we've got the circuit, now print it in reverse
  reverse(circuit.begin() , circuit.end());
  for(auto &i : circuit) cout << i + 1 << ' ';
```

SCC_Floyed.cpp

```
#include <bits/stdc++.h>
typedef long long ll;
#define s second
#define f first
#define rep(i , st , ed) for(int i = st ; i < ed ; i++)
const int N = 500;
using namespace std;
void burn(){
ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
#ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
 #endif
}
//\\//\\//\\//\\//\\//\\//\\//\\//\\//
int reach[N][N] , dag[N][N];
vector<int> comp , sz;
void SCC(int n){
 comp.clear(); comp.resize(n , -1);
 sz.clear();
 // Floyed
 rep(k , 0 , n) rep(i , 0 , n) rep(j , 0 , n)
   reach[i][j] |= (reach[i][k] && reach[k][j]); // Warshall Transitive closure
 int cnt = 0;
 // detect SCC
 rep(i , 0 , n){
  if(comp[i] == -1){
   comp[i] = cnt++;
   rep(j , 0 , n)
    if(reach[i][j] && reach[j][i]) comp[j] = comp[i];
  }
 }
 sz.resize(cnt);
 rep(i , 0 , n) sz[comp[i]]++;
 // Create Dag
 rep(i , 0 , n) rep(j , 0 , n)
  if(reach[i][j]) dag[comp[i]][comp[j]] = 1;
}
int main(){
 burn();
 int n , m; cin >> n >> m;
 for(int i = 0 ; i < m ; i++){
  int u , v; cin >> u >> v;
  --u; --v;
  reach[u][v] = 1;
 }
 SCC(n);
}
```

SCC_kosaraju.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
#define all(v) v.begin() , v.end()
const int N = 2e5 + 9;
vector<int> adj[N], adj_rev[N];
bool used[N];
vector<int> order, component;

void dfs1(int v) {
    used[v] = true;
    for (auto u : adj[v])
        if (!used[u])
            dfs1(u);
    order.push_back(v);
}
void dfs2(int v) {
    used[v] = true;
    component.push_back(v);
    for (auto u : adj_rev[v])
        if (!used[u])   dfs2(u);
}

```

```
int main() {
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    int n,m; cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int a, b; cin >> a >> b;
        --a; --b;
        adj[a].push_back(b);
        adj_rev[b].push_back(a);
    }
    for(int i = 0; i < n; ++i) used[i] = false;
    for (int i = 0; i < n; i++) if (!used[i]){
        dfs1(i);
    }
    for(int i = 0; i < n; ++i) used[i] = false;
    reverse(order.begin(), order.end());
    for (auto v : order)if (!used[v]) {
        dfs2 (v);
        component.clear();
    }
}
```

SCC_tarjan.cpp

```
#include <bits/stdc++.h>
typedef long long ll;
#define s second
#define f first
#define rep(i , st , ed) for(int i = st ; i < ed ; i++)
using namespace std;
void burn(){
ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
#ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
  #endif
}
const int N = 1e5;
```

```
//\\//\\//\\//\\//\\//\\//\\//\\//\\//
vector<vector<int>> adj , dag , comps;
int comp[N] , inStack[N] , lowLink[N] , dfn[N] , deg[N];
stack<int> st;
int ndfn;
void tarjan(int u){
 dfn[u] = lowLink[u] = ndfn++;
 inStack[u] = true;
 st.push(u);
 for(auto &v : adj[u]){
  if(dfn[v] == -1){
   tarjan(v);
   lowLink[u] = min(lowLink[u] , lowLink[v]);
  }else if(inStack[v]){
   lowLink[u] = min(lowLink[u] , dfn[v]);
  }
 }
 if(dfn[u] == lowLink[u]){
  // head of component
  int x = -1;
  comps.emplace_back(vector<int>());
  while(x != u){
   x = st.top(); st.pop(); inStack[x] = 0;
   comps.back().emplace_back(x);
   comp[x] = comps.size() - 1;
  }
 }
}
void genDag(){
 dag.resize(comps.size());
 for(int u = 0 ; u < adj.size() ; u++){
  for(auto &v :adj[u]){
   if(comp[u] != comp[v]){
    dag[comp[u]].emplace_back(comp[v]);
    deg[comp[v]]++;
   }
  }
 }
}
void SCC(int n){
 ndfn = 0;
 comps.clear();
 rep(i , 0 , n){
   dfn[i] = -1;
   lowLink[i] = inStack[i] = deg[i] = 0;
 }
 for(int i = 0 ; i < n ; i++)
  if(dfn[i] == -1) tarjan(i);
 genDag();
}
int main(){
 burn();
 int n , m; cin >> n >> m;
 adj.resize(n);
 for(int i = 0 ; i < m ; i++){
  int u , v; cin >> u >> v;
  --u; --v;
  adj[u].emplace_back(v);
 }
 SCC(n);
```

artPoints.cpp

```
#include <bits/stdc++.h>
typedef long long ll;
#define s second
#define f first
#define rep(i , st , ed) for(int i = st ; i < ed ; i++)
using namespace std;
void burn(){
ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
#ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
 #endif
}
const int N = 1e5;
//\/\//\/\//\/\//\/\//\/\//\/\//\/\//
vector<vector<int>> adj;
int lowLink[N] , dfn[N];
set<int>artpoints;
int ndfn;
bool root = false;
void tarjan(int u, int par){
 dfn[u] = lowLink[u] = ndfn++;
```

```
  for(auto &v : adj[u]){
    if(dfn[v] == -1){
      tarjan(v, u);
      lowLink[u] = min(lowLink[u] , lowLink[v]);
      if (lowLink[v] >= dfn[u]){
        if (dfn[u] == 0 && root == false)
            root = true;
        else artpoints.emplace(u);
      }
    }else if(v != par){
      lowLink[u] = min(lowLink[u] , dfn[v]);
    }
  }
}
int main(){
  burn();
  int n , m; cin >> n >> m;
  for (int i = 0; i < n; ++i)
    dfn[i] = -1;
  adj.resize(n);
  for(int i = 0 ; i < m ; i++){
    int u , v; cin >> u >> v;
    --u; --v;
    adj[u].emplace_back(v);
    adj[v].emplace_back(u);
  }
  tarjan(0, -1);
  // ALL articulation points are stored in **artspoints** set
}
```

bellman_ford.cpp

```
struct Edge {
    int a, b, cost;
};

int n, m;
vector<Edge> edges;
const int INF = 1000000000;

void solve()
{
    vector<int> d(n);
    vector<int> p(n, -1);
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (Edge e : edges) {
            if (d[e.a] + e.cost < d[e.b]) {
                d[e.b] = d[e.a] + e.cost;
                p[e.b] = e.a;
                x = e.b;
            }
        }
    }

    if (x == -1) {
        cout << "No negative cycle found.";
    } else {
        for (int i = 0; i < n; ++i)
            x = p[x];

        vector<int> cycle;
        for (int v = x;; v = p[v]) {
            cycle.push_back(v);
            if (v == x && cycle.size() > 1)
                break;
        }
        reverse(cycle.begin(), cycle.end());

        cout << "Negative cycle: ";
        for (int v : cycle)
            cout << v << ' ';
        cout << endl;
    }
}
```

biConnected.cpp

```
#include <bits/stdc++.h>
typedef long long ll;
#define s second
#define f first
#define rep(i , st , ed) for(int i = st ; i < ed ; i++)
```

```
using namespace std;
void burn(){
ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
#ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
 #endif
}
const int N = 1e5;
//\\//\\//\\//\\//\\//\\//\\//\\//\\//
vector<vector<int>> adj;
int lowLink[N] , dfn[N];
stack<pair<int,int>> comps;
vector<vector<pair<int, int>>>bi_connected;
int ndfn;
pair<int,int> edge;
bool root = false;
void tarjan(int u, int par){
 dfn[u] = lowLink[u] = ndfn++;
 for(auto &v : adj[u]){
  if (v != par && dfn[v] < dfn[u])
    comps.push(make_pair(u, v));

  if(dfn[v] == -1){
   tarjan(v, u);
   lowLink[u] = min(lowLink[u] , lowLink[v]);
   if (lowLink[v] >= dfn[u]){
    bi_connected.emplace_back(vector<pair<int,int>>());
    do{
      edge = comps.top();
      comps.pop();
      bi_connected.back().emplace_back(edge);

    }while(edge.first != u || edge.second != v);
    reverse(bi_connected.back().begin(), bi_connected.back().end());
   }
  }else if(v != par){
   lowLink[u] = min(lowLink[u] , dfn[v]);
  }
 }
}
int main(){
 //burn();
```

```cpp
  int n , m; cin >> n >> m;
  for (int i = 0; i < n; ++i)
    dfn[i] = -1;
  adj.resize(n);
  for(int i = 0 ; i < m ; i++){
    int u , v; cin >> u >> v;
    --u; --v;
    adj[u].emplace_back(v);
    adj[v].emplace_back(u);
  }
  tarjan(0, -1);
  // bi_connected vector stores all the edges in each biconnected component
  // bi_connected.size() is the number of biconnected componenets
}
```

dijkstra.cpp

```cpp
#include <bits/stdc++.h>
typedef long long ll;
#define s second
#define f first
using namespace std;
int main(){
  /*
    ------ Dijkstra Algrothim --------
    - use to Find Shortest path from Single Sourse to other vertices
    - can used if weight of edge >= 0
    - Find answer in O(nlog(n))
  */
  int n , m; cin >> n >> m; // n : number of vertices , m : number of edges
  vector<pair<int,ll>> adj[n]; // {v , w}
  for(int i = 0 ; i < m ; i++){
    int u , v; ll w; cin >> u >> v >> w;
    adj[u].emplace_back(v , w);
    adj[v].emplace_back(u , w);
  }
  int Start; cin >> Start; // vertice that you start from
  Start--;
  priority_queue<pair<ll,int>> q; // {dis , u}
  vector<ll> dis(n , 1e15);
  q.push({0 , Start});
  dis[Start] = 0;
  while(q.size()){
```

```
  auto [d , u] = q.top().s; q.pop();
  if(-d != dis[u]) continue;
  vis[u] = true;
  for(auto &[v , w] : adj[u]){
   if(dis[v] > dis[u] + w){ // relaxing
    dis[v] = dis[u] + w;
    q.push({-dis[v] , v});
   }
  }
 }
}
```

dijkstra_sparse_graph.cpp

```
/*
O(n^2 + m)
*/
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

void dijkstra(int s, vector<int> & d, vector<int> & p) {
  int n = adj.size();
  d.assign(n, INF);
  p.assign(n, -1);
  vector<bool> u(n, false);

  d[s] = 0;
  for (int i = 0; i < n; i++) {
    int v = -1;
    for (int j = 0; j < n; j++) {
      if (!u[j] && (v == -1 || d[j] < d[v]))
        v = j;
    }

    if (d[v] == INF)
      break;

    u[v] = true;
    for (auto edge : adj[v]) {
      int to = edge.first;
      int len = edge.second;

```

```
            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                p[to] = v;
            }
        }
    }
}
```

euler_ciruite_undirectedgraph.cpp

```
/*
    Eulerian Circuit is an Eulerian Path that starts and ends on the same vertex.

    Restrictions:
    An undirected graph has Eulerian cycle if following two conditions are true.
        1. All vertices with non-zero degree are connected. We don’t care about vertices with zero degree
           because they don’t belong to Eulerian Cycle or Path (we only consider all edges).
        2. All vertices have even degree.

*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
const int N = 1e5 + 9 , M = 2e5 + 9;
vector<pair<int,int>> adj[N];
int vis[M];
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    int n, m; cin >> n >> m;
    for(int i = 0; i < m; ++i){
        int u,v; cin >> u >> v;
        --u; --v;
        adj[u].emplace_back(v,i);
        adj[v].emplace_back(u,i);
    }
    for(int i = 0; i < n; ++i)if ((int) adj[i].size() & 1){
        cout << "IMPOSSIBLE"; // Handling manual
        return 0;
    }
    // there could be more than one euler circuit if the graph aren't connected ans we will find one of them;
    stack<int> st;
    for(int i = 0; i < n; ++i) if(adj[i].size()){
        st.push(i);
        break;
    }
    // if st.empty() --> no.edges = 0
    vector<int> path;
    while(!st.empty()){
        int v = st.top();
        int f=0;
        while(!adj[v].empty()) {
            auto [u,i] =  adj[v].back();
            adj[v].pop_back();
            if (!vis[i]) {
                st.push(u);
                vis[i]=1;
                f=1;
                break;
            }
        }
        if (!f){
            path.emplace_back(v);
            st.pop();
        }
    }
    for (auto &i: path) cout << i + 1 << " ";
}
```

euler_path_directed.cpp

```
/*
  Finding euler path in directed graph
  Time complexity: O(N + M)
  Space complexity: O(N + M)
```

```
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
const int N = 1e5 + 9; // no.of node
vector<int> adj[N];
vector<int> path; // euler path
void dfs(int s){
  while((int) adj[s].size()){
    int u = adj[s].back();
    adj[s].pop_back();
    dfs(u);
  }
  path.emplace_back(s);
}
int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
  #ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
  #endif
  int n,m; cin >> n >> m;
  int in[n] = {}, out[n] = {};
  for(int i = 0; i < m; ++i){
    int x,y; cin >> x >> y;
    --x; --y;
    adj[x].emplace_back(y);
    in[y]++, out[x]++;
  }
  int a=0,b=0,c=0,s1=0,s2=0;
  for(int i = 0; i < n; ++i){
    if (in[i]==out[i]) c++;
    if (in[i]-out[i]==1){ b++; s2=i; }
    if (in[i]-out[i]==-1){ a++; s1=i; }
  }
  if (s1 != 0 || s2 != n - 1){
    cout << "IMPOSSIBLE";
    return 0;
  }
  if (!(c==n-2 && a==1 && b == 1)){
      cout << "IMPOSSIBLE";
    return 0;
  }
  dfs(0);
  if (path.size() != m + 1){
    cout << "IMPOSSIBLE";
    return 0;
  }
  reverse(path.begin(), path.end());
  for (auto &i: path) cout << i + 1 << " ";
```

euler_path_undirected.cpp

```
/*
    Finding Euler path in undirected graph
    Time complexity: O(M)
    Space complexity: O(M + N)

    Standard problem: https://cses.fi/problemset/task/1691
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
const int N = 1e5 + 9 , M = 2e5 + 9;
vector<pair<int,int>> adj[N];
int vis[M];
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    int n, m; cin >> n >> m;
    for(int i = 0; i < m; ++i){
        int u,v; cin >> u >> v;
        --u; --v;
        adj[u].emplace_back(v,i);
        adj[v].emplace_back(u,i);
    }
    for(int i = 0; i < n; ++i)if ((int) adj[i].size() & 1){
        cout << "IMPOSSIBLE"; // Handling manual
        return 0;
    }
    stack<int> st;
    st.push(0);
    vector<int> path;
    while(!st.empty()){
        int v = st.top();
        int f=0;
        while(!adj[v].empty()) {
            auto [u,i] =  adj[v].back();
            adj[v].pop_back();
            if (!vis[i]) {
                st.push(u);
                vis[i]=1;
                f=1;
                break;
            }
        }
        if (!f){
            path.emplace_back(v);
            st.pop();
        }
    }
    if ((int) path.size() != m + 1){
        cout << "IMPOSSIBLE"; // Handling manual
        return 0;
    }
    for (auto &i: path) cout << i + 1<< " ";
}
```

floyd.cpp

```
int n , m; cin >> n >> m;
vector<vector<ll>> adj(n , vector<ll>(n , OOLL));
for(int i = 0; i < n; ++i) adj[i][i] = 0;
for(int i = 0; i < m; ++i){
int u , v; ll w; cin >> u >> v >> w;
--u; --v;
adj[u][v] = min(adj[u][v] , w);
adj[v][u] = min(adj[v][u] , w);
}
// Floyd
rep(k , 0, n) rep(i , 0 , n) rep(j , 0, n){
adj[i][j] = min(adj[i][j] , adj[i][k] + adj[k][j]);
}
```

kth_shortest_path.cpp

```
/*
  Finding the First k's shortest path in
  O(m * k) such that m : no. of edges
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
  #ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
  #endif
  int n , m, k; cin >> n >> m >> k;
  vector<pair<int,ll>> adj[n]; // {v , w}
  for(int i = 0; i < m; ++i){
    int u , v;  ll w; cin >> u >> v >> w;
    --u; --v;
    adj[u].emplace_back(v , w);
  }
  priority_queue<pair<ll,int>> q; // {-w , u}
  vector<ll> cnt(n) , ans;
  int start = 0 , end = n - 1;
  q.push({0 , start});
  while(q.size() && cnt[end] < k){
    auto [d , u] = q.top(); q.pop();
    d *= -1;
    cnt[u]++;
    if(u == end) ans.emplace_back(d);
```

```cpp
        if(cnt[u] <= k){
            for(auto &[v,  w] : adj[u]) if(cnt[v] < k)
                q.push({-(w + d) , v});
        }
    }
    sort(ans.begin() , ans.end());
    for(auto &w : ans) cout << w << " ";
}
```

Handbook material

Imgs

Masking

Generate_next_lexicographical_K-combination.cpp

```cpp
bool next_combination(vector<int>& a, int n) {
    int k = (int)a.size();
    for (int i = k - 1; i >= 0; i--) {
        if (a[i] < n - k + i + 1) {
            a[i]++;
            for (int j = i + 1; j < k; j++)
                a[j] = a[j - 1] + 1;
            return true;
        }
    }
    return false;
}
```

Generating_all_submasking_of_k_ones.cpp

```cpp
/*
    Gosper’s Hack (Bankers sequence)
    Time Complexity: nCk
*/
void f(int mask){
    // proccess the current mask with k 1's
}
void GospersHack(int n,int k){
    int sets=(1ll<<k)-1;
    int limit=(1ll<<n);
    while(sets<limit){
        f(sets);
        int c= sets & -sets;
        int r= sets + c;
        sets=( ( (r^sets) >>2 ) /c ) | r;
    }
}
```

gen_all_possible_submasking.cpp

```cpp
/*
    ----> Generating All possible submasks <-----
    - Iterating through all masks with their submasks. Complexity O(3^n)
*/
// m : mask , s : submask
for (int m=0; m<(1<<n); ++m)
    for (int s=m; s; s=(s-1)&m)
```

Math

Counting

BurnsideLemma.cpp

```cpp
/// Burnside Lemma Notes:

/**
Problem 1: Consider a circular stripe of N cells and we are given M colors.
In how many ways we can color the stripe. 2 ways are same if we can make one from other using rotation.

Here, X is a set of all colored stripes (it has M^N elements),
G is the group of its rotations (it has N elements: rotation by 0 cells, by 1 cell...by (N-1) cells),
An orbit is exactly the set of all stripes that can be obtained from each other using rotations,
So the number of orbits will be the number of distinct stripes up to a rotation.

```

```
Now let's apply the lemma, and find the number of stripes that are fixed by the rotation by K cells.
If a stripe becomes itself after rotating by K cells, then its 1st cell must have the same color as its (1+K modulo N)'th cell,
which is in turn the same as its (1+2K modulo N)'th cell...until we get back to the 1st cell again when (P*K % N)=0.

This will happen when P = N/gcd(K,N), and thus we get N/gcd(K,N) cells that must all be of the same color.
Analogously,the same amount of cells must be of the same color starting with cell 2, (2+K modulo N) etc.

Thus, all cells are separated into gcd(K,N) groups, with each group being of one color, and that yields us M^gcd(K,N) choices.
And by Burnside's lemma, the answer to the original problem is sum(M^gcd(K,N))/N, for K from 0 to N-1
**/

/**
Problem 2: You have 4 red, 4 white, and 4 blue identical dinner plates.
In how many different ways can you set a square table with one plate on each side?
2 ways are same if we can make one from other using rotation.

We have four possible rotations(clockwise) 0,90,180 and 270 degrees.
Let's A0 = rotation by 0, A1 = rotation by 90...A3 = rotation by 270 degree
So we have cyclic group of 4 elements (possible rotations)

Let's S = RWBR is a valid arrangement where R is on north, W on East, B on south and R on west
So A1(S) = RRWB(rotation by 90 degree)

Now using Burnside lemma let's find  how many arrangements are fixed under various rotations.

For A0 we rotate S by 0 degree. So there are 3^4 fixed points.

For A1 we rotate S by 90 degree. If S and A1(S) will have to be same then,
north-east must have same color, east-south must have same color,
same for south-west and west-north. Which means all side must have same color.
So there are 3 fixed points for A1.

A3 is same as A1, because rotation by 270 degree does same as rotation by 90 degree.
So there are 3 fixed points for A3 too.

```

```
For A2 we rotate S by 180 degree. If S and A2(S) will have to be same then,
north-south must have same color and east-west must have same color.
So we have 3*3 fixed points for A2.

So there are total (3^4 + 3 + 3 + 3*3) = 96 fixed points.
And by Burnside's lemma, the answer to the original problem is 96/4 = 24.
*/
```

FFT.cpp

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;


// FFT
namespace FFT {
    using DD = double;
    const DD PI = acosl(-1);

    struct Comp {
        DD real, imag;
        Comp(DD real = 0, DD imag = 0) : real(real), imag(imag) {}
        friend inline ostream& operator << (ostream &s, const Comp &c) {
            return s << '<' << c.real << ',' << c.imag << '>';
        }
        inline Comp operator + (const Comp &c) {
            return {real + c.real, imag + c.imag};
        }
        inline Comp operator - (const Comp &c) {
            return {real - c.real, imag - c.imag};
        }
        inline Comp operator * (const Comp &c) {
            return {real * c.real - imag * c.imag,
                    real * c.imag + imag * c.real};
        }
        inline Comp operator * (DD a) {
            return {real * a, imag * a};
        }
        inline Comp operator / (DD a) {
            return {real / a, imag / a};
        }
    };

    // FFT
    void trans(vector<Comp> &v, bool inv = false) {
      int n = (int)v.size();
      for (int i = 0, j = 1; j < n-1; j++) {
        for (int k = n>>1; k > (i ^= k); k >>= 1);
        if (i > j) swap(v[i], v[j]);
      }
      for (int t = 2; t <= n; t <<= 1) {
        DD ang = acosl(-1.0) * 2 / t;
        if (inv) ang = -ang;
        for (int i = 0; i < n; i += t) {
          for (int j = 0; j < t/2; ++j) {
            Comp w = {cos(ang * j), sin(ang * j)};
            int j1 = i + j, j2 = i + j + t/2;
            Comp c1 = v[j1], c2 = v[j2] * w;
            v[j1] = c1 + c2;
            v[j2] = c1 - c2;
          }
        }
      }
      if (inv) for (int i = 0; i < n; ++i) v[i] = v[i]/n;
    }

    // A * B
    vector<long long> mult(const vector<long long> &A,
                const vector<long long> &B) {
      int size_a = 1; while (size_a < A.size()) size_a <<= 1;
      int size_b = 1; while (size_b < B.size()) size_b <<= 1;
      int size_fft = max(size_a, size_b) << 1;

      vector<Comp> cA(size_fft), cB(size_fft), cC(size_fft);
      for (int i = 0; i < A.size(); ++i) cA[i] = {(DD)A[i], 0};
      for (int i = 0; i < B.size(); ++i) cB[i] = {(DD)B[i], 0};

      trans(cA); trans(cB);
      for (int i = 0; i < size_fft; ++i) cC[i] = cA[i] * cB[i];
      trans(cC, true);

      vector<long long> res((int)A.size() + (int)B.size() - 1);
      for (int i = 0; i < res.size(); ++i) {
        res[i] = (long long)(cC[i].real + 0.5);
      }
```

```cpp
        return res;
    }
};



//-----------------------------//
// Examples
//-----------------------------//

int main() {
    int N;
    while (cin >> N) {
        vector<long long> a(N), b(N);
        for (int i = 0; i < N; ++i) cin >> a[i] >> b[i];
        auto res = FFT::mult(a, b);
        cout << 0 << endl;
        for (int i = 0; i < N*2-1; ++i) cout << res[i] << endl;
    }
}
```

JosephusTheorem.cpp

```cpp
/// Given a group of n men arranged in a circle under the edict that every k'th man
/// will be executed going around the circle until only one remains.
/// Find out who will be the final survivor.

int josephus(int n, int k){
    if (n == 1) return 1;
    return (josephus(n - 1, k) + k-1) % n + 1;
}

/// log base solution when k is 2
int josephus(int n){
    int p = 1;
    while (p <= n) p *= 2;
    return (2*n) - p + 1;
}
```

Matrix_Exponential.cpp

```cpp
struct Mat {
    ll mat[3][3];
    ll row, col;

    Mat(ll _r, ll _c) : row(_r), col(_c) {
        memset(mat , 0 , sizeof mat);

    }

    Mat operator *(const Mat& b) const {
        Mat Product(row, b.col);
        for(int i = 0; i < row; ++i) {
            for(int k = 0; k < col; ++k) {
                if(mat[i][k] != 0) {
                    for(int j = 0; j < b.col; ++j) {
                        Product.mat[i][j] += mat[i][k] * b.mat[k][j] % mod;
                    }
                }

            }
            for(int j = 0; j < b.col; ++j) {
                Product.mat[i][j] %= mod;
            }
        }
        return Product;
    }
};

Mat power(Mat a, ll b) {
    Mat res(a.row, a.col);
    for(int i = 0; i < res.row; ++i) res.mat[i][i] = 1;
    while(b > 0) {
        if(b & 1) {
            res = res * a;
        }
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

Mint.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define rrep(i,a,b) for(int i=a;i>=b;i--)

#define f first
#define s second
template<int MOD> struct ModInt {
  static const int Mod = MOD; unsigned x; ModInt() : x(0) { }
  ModInt(signed sig) { x = sig < 0 ? sig % MOD + MOD : sig % MOD; }
  ModInt(signed long long sig) { x = sig < 0 ? sig % MOD + MOD : sig % MOD; }
  int get() const { return (int)x; }
  ModInt &operator+=(ModInt that) { if ((x += that.x) >= MOD) x -= MOD; return *this; }
  ModInt &operator-=(ModInt that) { if ((x += MOD - that.x) >= MOD) x -= MOD; return *this; }
  ModInt &operator*=(ModInt that) { x = (unsigned long long)x * that.x % MOD; return *this; }
  ModInt &operator/=(ModInt that) { return *this *= that.inverse(); }
  ModInt operator+(ModInt that) const { return ModInt(*this) += that; }
  ModInt operator-(ModInt that) const { return ModInt(*this) -= that; }
  ModInt operator*(ModInt that) const { return ModInt(*this) *= that; }
  ModInt operator/(ModInt that) const { return ModInt(*this) /= that; }
  ModInt inverse() const { long long a = x, b = MOD, u = 1, v = 0;
    while (b) { long long t = a / b; a -= t * b; std::swap(a, b); u -= t * v; std::swap(u, v); }
    return ModInt(u); }
  bool operator==(ModInt that) const { return x == that.x; }
  bool operator!=(ModInt that) const { return x != that.x; }
  ModInt operator-() const { ModInt t; t.x = x == 0 ? 0 : Mod - x; return t; }
};
template<int MOD> ostream& operator<<(ostream& st, const ModInt<MOD> a) { st << a.get(); return st; };
template<int MOD> ModInt<MOD> operator^(ModInt<MOD> a, unsigned long long k) {
  ModInt<MOD> r = 1; while (k) { if (k & 1) r *= a; a *= a; k >>= 1; } return r; }
template<typename T, int FAC_MAX> struct Comb { vector<T> fac, ifac;
  Comb(){fac.resize(FAC_MAX,1);ifac.resize(FAC_MAX,1);rep(i,1,FAC_MAX)fac[i]=fac[i-1]*i;
    ifac[FAC_MAX-1]=T(1)/fac[FAC_MAX-1];rrep(i,FAC_MAX-2,1)ifac[i]=ifac[i+1]*T(i+1);}
  T aPb(int a, int b) { if (b < 0 || a < b) return T(0); return fac[a] * ifac[a - b]; }
  T aCb(int a, int b) { if (b < 0 || a < b) return T(0); return fac[a] * ifac[a - b] * ifac[b]; }
  T nHk(int n, int k) { if (n == 0 && k == 0) return T(1); if (n <= 0 || k < 0) return 0;
    return aCb(n + k - 1, k); } // nHk = (n+k-1)Ck : n is separator
  T pairCombination(int n) {if(n%2==1)return T(0);return fac[n]*ifac[n/2]/(T(2)^(n/2));}
  // combination of paris for n
```

```cpp
};
typedef ModInt<998244353> mint;
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    const int N = 1e5;
    Comb<mint , N> com;
}
```

NTT_prime_mod.cpp

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


// modint
template<int MOD> struct Fp {
    // inner value
    long long val;

    // constructor
    constexpr Fp() : val(0) { }
    constexpr Fp(long long v) : val(v % MOD) {
        if (val < 0) val += MOD;
    }
    constexpr long long get() const { return val; }
    constexpr int get_mod() const { return MOD; }

    // arithmetic operators
    constexpr Fp operator + () const { return Fp(*this); }
    constexpr Fp operator - () const { return Fp(0) - Fp(*this); }
    constexpr Fp operator + (const Fp &r) const { return Fp(*this) += r; }
    constexpr Fp operator - (const Fp &r) const { return Fp(*this) -= r; }
    constexpr Fp operator * (const Fp &r) const { return Fp(*this) *= r; }
    constexpr Fp operator / (const Fp &r) const { return Fp(*this) /= r; }
    constexpr Fp& operator += (const Fp &r) {
        val += r.val;
        if (val >= MOD) val -= MOD;
        return *this;
    }
    constexpr Fp& operator -= (const Fp &r) {
        val -= r.val;
        if (val < 0) val += MOD;
        return *this;
    }
    constexpr Fp& operator *= (const Fp &r) {
        val = val * r.val % MOD;
        return *this;
    }
    constexpr Fp& operator /= (const Fp &r) {
        long long a = r.val, b = MOD, u = 1, v = 0;
        while (b) {
            long long t = a / b;
            a -= t * b, swap(a, b);
            u -= t * v, swap(u, v);
        }
        val = val * u % MOD;
        if (val < 0) val += MOD;
        return *this;
    }
    constexpr Fp pow(long long n) const {
        Fp res(1), mul(*this);
        while (n > 0) {
            if (n & 1) res *= mul;
            mul *= mul;
            n >>= 1;
        }
        return res;
    }
    constexpr Fp inv() const {
        Fp res(1), div(*this);
        return res / div;
    }

    // other operators
    constexpr bool operator == (const Fp &r) const {
        return this->val == r.val;
    }
    constexpr bool operator != (const Fp &r) const {
        return this->val != r.val;
    }
    constexpr Fp& operator ++ () {
        ++val;
        if (val >= MOD) val -= MOD;
        return *this;
    }
    constexpr Fp& operator -- () {
        if (val == 0) val += MOD;
        --val;
        return *this;
    }
    constexpr Fp operator ++ (int) const {
        Fp res = *this;
        ++*this;
        return res;
    }
    constexpr Fp operator -- (int) const {
        Fp res = *this;
        --*this;
        return res;
    }
    friend constexpr istream& operator >> (istream &is, Fp<MOD> &x) {
        is >> x.val;
        x.val %= MOD;
        if (x.val < 0) x.val += MOD;
        return is;
    }
    friend constexpr ostream& operator << (ostream &os, const Fp<MOD> &x) {
        return os << x.val;
    }
    friend constexpr Fp<MOD> pow(const Fp<MOD> &r, long long n) {
        return r.pow(n);
    }
    friend constexpr Fp<MOD> inv(const Fp<MOD> &r) {
        return r.inv();
    }
};

namespace NTT {
    long long modpow(long long a, long long n, int mod) {
        long long res = 1;
        while (n > 0) {
            if (n & 1) res = res * a % mod;
            a = a * a % mod;
            n >>= 1;
        }
    return res;
}

long long modinv(long long a, int mod) {
    long long b = mod, u = 1, v = 0;
    while (b) {
        long long t = a / b;
        a -= t * b, swap(a, b);
        u -= t * v, swap(u, v);
    }
    u %= mod;
    if (u < 0) u += mod;
    return u;
}

int calc_primitive_root(int mod) {
    if (mod == 2) return 1;
    if (mod == 167772161) return 3;
    if (mod == 469762049) return 3;
    if (mod == 754974721) return 11;
    if (mod == 998244353) return 3;
    int divs[20] = {};
    divs[0] = 2;
    int cnt = 1;
    long long x = (mod - 1) / 2;
    while (x % 2 == 0) x /= 2;
    for (long long i = 3; i * i <= x; i += 2) {
        if (x % i == 0) {
            divs[cnt++] = i;
            while (x % i == 0) x /= i;
        }
    }
    if (x > 1) divs[cnt++] = x;
    for (int g = 2;; g++) {
        bool ok = true;
        for (int i = 0; i < cnt; i++) {
            if (modpow(g, (mod - 1) / divs[i], mod) == 1) {
                ok = false;
                break;
            }
        }
        if (ok) return g;
    }
```

```
}

int get_fft_size(int N, int M) {
    int size_a = 1, size_b = 1;
    while (size_a < N) size_a <<= 1;
    while (size_b < M) size_b <<= 1;
    return max(size_a, size_b) << 1;
}

// number-theoretic transform
template<class mint> void trans(vector<mint> &v, bool inv = false) {
    if (v.empty()) return;
    int N = (int)v.size();
    int MOD = v[0].get_mod();
    int PR = calc_primitive_root(MOD);
    static bool first = true;
    static vector<long long> vbw(30), vibw(30);
    if (first) {
        first = false;
        for (int k = 0; k < 30; ++k) {
            vbw[k] = modpow(PR, (MOD - 1) >> (k + 1), MOD);
            vibw[k] = modinv(vbw[k], MOD);
        }
    }
    for (int i = 0, j = 1; j < N - 1; j++) {
        for (int k = N >> 1; k > (i ^= k); k >>= 1);
        if (i > j) swap(v[i], v[j]);
    }
    for (int k = 0, t = 2; t <= N; ++k, t <<= 1) {
        long long bw = vbw[k];
        if (inv) bw = vibw[k];
        for (int i = 0; i < N; i += t) {
            mint w = 1;
            for (int j = 0; j < t/2; ++j) {
                int j1 = i + j, j2 = i + j + t/2;
                mint c1 = v[j1], c2 = v[j2] * w;
                v[j1] = c1 + c2;
                v[j2] = c1 - c2;
                w *= bw;
            }
        }
    }
    if (inv) {
            long long invN = modinv(N, MOD);
            for (int i = 0; i < N; ++i) v[i] = v[i] * invN;
        }
    }

    // for garner
    static constexpr int MOD0 = 754974721;
    static constexpr int MOD1 = 167772161;
    static constexpr int MOD2 = 469762049;
    using mint0 = Fp<MOD0>;
    using mint1 = Fp<MOD1>;
    using mint2 = Fp<MOD2>;
    static const mint1 imod0 = 95869806; // modinv(MOD0, MOD1);
    static const mint2 imod1 = 104391568; // modinv(MOD1, MOD2);
    static const mint2 imod01 = 187290749; // imod1 / MOD0;

    // small case (T = mint, long long)
    template<class T> vector<T> naive_mul(const vector<T> &A, const vector<T> &B) {
        if (A.empty() || B.empty()) return {};
        int N = (int)A.size(), M = (int)B.size();
        vector<T> res(N + M - 1);
        for (int i = 0; i < N; ++i)
            for (int j = 0; j < M; ++j)
                res[i + j] += A[i] * B[j];
        return res;
    }

    // mul by convolution
    template<class mint> vector<mint> mul(const vector<mint> &A, const vector<mint> &B) {
        if (A.empty() || B.empty()) return {};
        int N = (int)A.size(), M = (int)B.size();
        if (min(N, M) < 30) return naive_mul(A, B);
        int MOD = A[0].get_mod();
        int size_fft = get_fft_size(N, M);
        if (MOD == 998244353) {
            vector<mint> a(size_fft), b(size_fft), c(size_fft);
            for (int i = 0; i < N; ++i) a[i] = A[i];
            for (int i = 0; i < M; ++i) b[i] = B[i];
            trans(a), trans(b);
            vector<mint> res(size_fft);
            for (int i = 0; i < size_fft; ++i) res[i] = a[i] * b[i];
```

```cpp
            trans(res, true);
            res.resize(N + M - 1);
            return res;
        }
        vector<mint0> a0(size_fft, 0), b0(size_fft, 0), c0(size_fft, 0);
        vector<mint1> a1(size_fft, 0), b1(size_fft, 0), c1(size_fft, 0);
        vector<mint2> a2(size_fft, 0), b2(size_fft, 0), c2(size_fft, 0);
        for (int i = 0; i < N; ++i)
            a0[i] = A[i].val, a1[i] = A[i].val, a2[i] = A[i].val;
        for (int i = 0; i < M; ++i)
            b0[i] = B[i].val, b1[i] = B[i].val, b2[i] = B[i].val;
        trans(a0), trans(a1), trans(a2), trans(b0), trans(b1), trans(b2);
        for (int i = 0; i < size_fft; ++i) {
            c0[i] = a0[i] * b0[i];
            c1[i] = a1[i] * b1[i];
            c2[i] = a2[i] * b2[i];
        }
        trans(c0, true), trans(c1, true), trans(c2, true);
        mint mod0 = MOD0, mod01 = mod0 * MOD1;
        vector<mint> res(N + M - 1);
        for (int i = 0; i < N + M - 1; ++i) {
            int y0 = c0[i].val;
            int y1 = (imod0 * (c1[i] - y0)).val;
            int y2 = (imod01 * (c2[i] - y0) - imod1 * y1).val;
            res[i] = mod01 * y2 + mod0 * y1 + y0;
        }
        return res;
    }
};

// Binomial coefficient
template<class T> struct BiCoef {
    vector<T> fact_, inv_, finv_;
    constexpr BiCoef() {}
    constexpr BiCoef(int n) noexcept : fact_(n, 1), inv_(n, 1), finv_(n, 1) {
        init(n);
    }
    constexpr void init(int n) noexcept {
        fact_.assign(n, 1), inv_.assign(n, 1), finv_.assign(n, 1);
        int MOD = fact_[0].getmod();
        for(int i = 2; i < n; i++){
            fact_[i] = fact_[i-1] * i;
            inv_[i] = -inv_[MOD%i] * (MOD/i);
            finv_[i] = finv_[i-1] * inv_[i];
        }
    }
    constexpr T com(int n, int k) const noexcept {
        if (n < k || n < 0 || k < 0) return 0;
        return fact_[n] * finv_[k] * finv_[n-k];
    }
    constexpr T fact(int n) const noexcept {
        if (n < 0) return 0;
        return fact_[n];
    }
    constexpr T inv(int n) const noexcept {
        if (n < 0) return 0;
        return inv_[n];
    }
    constexpr T finv(int n) const noexcept {
        if (n < 0) return 0;
        return finv_[n];
    }
};



//------------------------------//
// Examples
//------------------------------//

const int MOD = 998244353;
using mint = Fp<MOD>;

int main() {
    int N;
    cin >> N;
    map<int,long long> ma;
    for (int i = 0; i < N*2; ++i) {
        int h;
        cin >> h;
        ma[h]++;
    }
    BiCoef<mint> bc(N*2+1);
    priority_queue<pair<int,vector<mint>>, vector<pair<int,vector<mint>>>, greater<pair<int,vector<mint>>>> que;
    for (auto it : ma) {
        int n = it.second;
        vector<mint> pol(n/2+1, 1);
        for (int i = 0; i <= n/2; ++i) {
            pol[i] = bc.fact(n) * bc.finv(n - i*2) * bc.finv(i) / modpow(mint(2), i);
        }
        que.push({pol.size(), pol});
    }
    while (que.size() >= 2) {
        auto f = que.top().second; que.pop();
        auto g = que.top().second; que.pop();
        auto h = NTT::mul(f, g);
```

NTT_random_mod.cpp

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


// modint
template<int MOD> struct Fp {
    // inner value
    long long val;

    // constructor
    constexpr Fp() : val(0) { }
    constexpr Fp(long long v) : val(v % MOD) {
        if (val < 0) val += MOD;
    }
    constexpr long long get() const { return val; }
    constexpr int get_mod() const { return MOD; }

    // arithmetic operators
    constexpr Fp operator + () const { return Fp(*this); }
    constexpr Fp operator - () const { return Fp(0) - Fp(*this); }
    constexpr Fp operator + (const Fp &r) const { return Fp(*this) += r; }
    constexpr Fp operator - (const Fp &r) const { return Fp(*this) -= r; }
    constexpr Fp operator * (const Fp &r) const { return Fp(*this) *= r; }
    constexpr Fp operator / (const Fp &r) const { return Fp(*this) /= r; }
    constexpr Fp& operator += (const Fp &r) {
        val += r.val;
        if (val >= MOD) val -= MOD;
        return *this;
    }
    constexpr Fp& operator -= (const Fp &r) {
        val -= r.val;
        if (val < 0) val += MOD;
        return *this;
    }
    constexpr Fp& operator *= (const Fp &r) {
        val = val * r.val % MOD;
        return *this;
    }
    constexpr Fp& operator /= (const Fp &r) {
        long long a = r.val, b = MOD, u = 1, v = 0;
        while (b) {
            long long t = a / b;
            a -= t * b, swap(a, b);
            u -= t * v, swap(u, v);
        }
        val = val * u % MOD;
        if (val < 0) val += MOD;
        return *this;
    }
    constexpr Fp pow(long long n) const {
        Fp res(1), mul(*this);
        while (n > 0) {
            if (n & 1) res *= mul;
            mul *= mul;
            n >>= 1;
        }
        return res;
    }
    constexpr Fp inv() const {
        Fp res(1), div(*this);
        return res / div;
    }

    // other operators
    constexpr bool operator == (const Fp &r) const {
        return this->val == r.val;
    }
    constexpr bool operator != (const Fp &r) const {
        return this->val != r.val;
    }
    constexpr Fp& operator ++ () {
        ++val;
        if (val >= MOD) val -= MOD;
        return *this;
    }
    constexpr Fp& operator -- () {
        if (val == 0) val += MOD;
        --val;
        return *this;
    }
    constexpr Fp operator ++ (int) const {
        Fp res = *this;
        ++*this;
        return res;
    }
    constexpr Fp operator -- (int) const {
        Fp res = *this;
        --*this;
        return res;
    }
    friend constexpr istream& operator >> (istream &is, Fp<MOD> &x) {
        is >> x.val;
        x.val %= MOD;
        if (x.val < 0) x.val += MOD;
        return is;
    }
    friend constexpr ostream& operator << (ostream &os, const Fp<MOD> &x) {
        return os << x.val;
    }
    friend constexpr Fp<MOD> pow(const Fp<MOD> &r, long long n) {
        return r.pow(n);
    }
    friend constexpr Fp<MOD> inv(const Fp<MOD> &r) {
        return r.inv();
    }
};

namespace NTT {
    long long modpow(long long a, long long n, int mod) {
        long long res = 1;
    while (n > 0) {
        if (n & 1) res = res * a % mod;
        a = a * a % mod;
        n >>= 1;
    }
    return res;
}

long long modinv(long long a, int mod) {
    long long b = mod, u = 1, v = 0;
    while (b) {
        long long t = a / b;
        a -= t * b, swap(a, b);
        u -= t * v, swap(u, v);
    }
    u %= mod;
    if (u < 0) u += mod;
    return u;
}

int calc_primitive_root(int mod) {
    if (mod == 2) return 1;
    if (mod == 167772161) return 3;
    if (mod == 469762049) return 3;
    if (mod == 754974721) return 11;
    if (mod == 998244353) return 3;
    int divs[20] = {};
    divs[0] = 2;
    int cnt = 1;
    long long x = (mod - 1) / 2;
    while (x % 2 == 0) x /= 2;
    for (long long i = 3; i * i <= x; i += 2) {
        if (x % i == 0) {
            divs[cnt++] = i;
            while (x % i == 0) x /= i;
        }
    }
    if (x > 1) divs[cnt++] = x;
    for (int g = 2;; g++) {
        bool ok = true;
        for (int i = 0; i < cnt; i++) {
            if (modpow(g, (mod - 1) / divs[i], mod) == 1) {
                ok = false;
                break;
            }
        }
        if (ok) return g;
    }
}

int get_fft_size(int N, int M) {
    int size_a = 1, size_b = 1;
    while (size_a < N) size_a <<= 1;
    while (size_b < M) size_b <<= 1;
    return max(size_a, size_b) << 1;
}

// number-theoretic transform
template<class mint> void trans(vector<mint> &v, bool inv = false) {
    if (v.empty()) return;
    int N = (int)v.size();
    int MOD = v[0].get_mod();
    int PR = calc_primitive_root(MOD);
    static bool first = true;
    static vector<long long> vbw(30), vibw(30);
    if (first) {
        first = false;
        for (int k = 0; k < 30; ++k) {
            vbw[k] = modpow(PR, (MOD - 1) >> (k + 1), MOD);
            vibw[k] = modinv(vbw[k], MOD);
        }
    }
    for (int i = 0, j = 1; j < N - 1; j++) {
        for (int k = N >> 1; k > (i ^= k); k >>= 1);
        if (i > j) swap(v[i], v[j]);
    }
    for (int k = 0, t = 2; t <= N; ++k, t <<= 1) {
        long long bw = vbw[k];
        if (inv) bw = vibw[k];
        for (int i = 0; i < N; i += t) {
            mint w = 1;
            for (int j = 0; j < t/2; ++j) {
                int j1 = i + j, j2 = i + j + t/2;
                mint c1 = v[j1], c2 = v[j2] * w;
                v[j1] = c1 + c2;
                v[j2] = c1 - c2;
                w *= bw;
            }
        }
    }
    if (inv) {
        long long invN = modinv(N, MOD);
        for (int i = 0; i < N; ++i) v[i] = v[i] * invN;
    }
}

// for garner
static constexpr int MOD0 = 754974721;
static constexpr int MOD1 = 167772161;
static constexpr int MOD2 = 469762049;
using mint0 = Fp<MOD0>;
using mint1 = Fp<MOD1>;
using mint2 = Fp<MOD2>;
static const mint1 imod0 = 95869806; // modinv(MOD0, MOD1);
static const mint2 imod1 = 104391568; // modinv(MOD1, MOD2);
static const mint2 imod01 = 187290749; // imod1 / MOD0;

// small case (T = mint, long long)
template<class T> vector<T> naive_mul(const vector<T> &A, const vector<T> &B) {
    if (A.empty() || B.empty()) return {};
    int N = (int)A.size(), M = (int)B.size();
    vector<T> res(N + M - 1);
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < M; ++j)
            res[i + j] += A[i] * B[j];
    return res;
}

// mul by convolution
template<class mint> vector<mint> mul(const vector<mint> &A, const vector<mint> &B) {
    if (A.empty() || B.empty()) return {};
    int N = (int)A.size(), M = (int)B.size();
    if (min(N, M) < 30) return naive_mul(A, B);
    int MOD = A[0].get_mod();
    int size_fft = get_fft_size(N, M);
    if (MOD == 998244353) {
        vector<mint> a(size_fft), b(size_fft), c(size_fft);
```

```
            for (int i = 0; i < N; ++i) a[i] = A[i];
            for (int i = 0; i < M; ++i) b[i] = B[i];
            trans(a), trans(b);
            vector<mint> res(size_fft);
            for (int i = 0; i < size_fft; ++i) res[i] = a[i] * b[i];
            trans(res, true);
            res.resize(N + M - 1);
            return res;
        }
        vector<mint0> a0(size_fft, 0), b0(size_fft, 0), c0(size_fft, 0);
        vector<mint1> a1(size_fft, 0), b1(size_fft, 0), c1(size_fft, 0);
        vector<mint2> a2(size_fft, 0), b2(size_fft, 0), c2(size_fft, 0);
        for (int i = 0; i < N; ++i)
            a0[i] = A[i].val, a1[i] = A[i].val, a2[i] = A[i].val;
        for (int i = 0; i < M; ++i)
            b0[i] = B[i].val, b1[i] = B[i].val, b2[i] = B[i].val;
        trans(a0), trans(a1), trans(a2), trans(b0), trans(b1), trans(b2);
        for (int i = 0; i < size_fft; ++i) {
            c0[i] = a0[i] * b0[i];
            c1[i] = a1[i] * b1[i];
            c2[i] = a2[i] * b2[i];
        }
        trans(c0, true), trans(c1, true), trans(c2, true);
        mint mod0 = MOD0, mod01 = mod0 * MOD1;
        vector<mint> res(N + M - 1);
        for (int i = 0; i < N + M - 1; ++i) {
            int y0 = c0[i].val;
            int y1 = (imod0 * (c1[i] - y0)).val;
            int y2 = (imod01 * (c2[i] - y0) - imod1 * y1).val;
            res[i] = mod01 * y2 + mod0 * y1 + y0;
        }
        return res;
    }
};



//------------------------------//
// Examples
//------------------------------//

void Yosupo_Convolution_mod_1000000007() {
    const int MOD = 1000000007;
    using mint = Fp<MOD>;

    int N, M;
    cin >> N >> M;
    vector<mint> a(N), b(M);
    for (int i = 0; i < N; ++i) cin >> a[i];
    for (int i = 0; i < M; ++i) cin >> b[i];
```

check_point_inside_polygon.cpp

```
#include <bits/stdc++.h>
typedef long long ll;
typedef __int128 lll;
typedef long double ld;
typedef __float128 lld;
using namespace std;

ld pi = acos(-1);
ld epsilon = 1e-9;

struct vec2 {
    ld x, y;
    vec2() {this->x = 0; this->y = 0;}
    vec2(ld x, ld y) {this->x = x; this->y = y;}
};

vec2 add(vec2 a, vec2 b){
    vec2 ret;
    ret.x = a.x + b.x;
    ret.y = a.y + b.y;
    return ret;
}

vec2 sub(vec2 a, vec2 b) {
    vec2 ret;
    ret.x = a.x - b.x;
    ret.y = a.y - b.y;
    return ret;
}

```

```
ld cross(vec2 a, vec2 b) {
    return a.x * b.y - a.y * b.x;
}

ld dot(vec2 a, vec2 b) {
    return a.x * b.x + a.y * b.y;
}

ld length(vec2 a) {
    return sqrt(a.x * a.x + a.y * a.y);
}

ld lerp(ld t0, ld t1, ld x0, ld x1, ld t) {
    ld slope = (x1 - x0) / (t1 - t0);
    return x0 + slope * (t - t0);
}

vec2 mul(vec2 a, ld s) {
    a.x *= s;
    a.y *= s;
    return a;
}

vec2 normalize(vec2 a){
    ld len = length(a);
    vec2 ret;
    ret.x = a.x / len;
    ret.y = a.y / len;
    return ret;
}

//angle from the +x axis in range (-pi, pi)
ld polar_angle(vec2 a) {
    return atan2(a.y, a.x);
}

//project a onto b
vec2 project(vec2 a, vec2 b) {
    b = normalize(b);
    ld proj_mag = dot(a, b);
    return mul(b, proj_mag);
}

vec2 rotateCCW(vec2 a, ld theta) {
    vec2 ret(0, 0);
    ret.x = a.x * cos(theta) - a.y * sin(theta);
    ret.y = a.x * sin(theta) + a.y * cos(theta);
    return ret;
}

//returns the coefficients s and t, where p1 + v1 * s = p2 + v2 * t
vector<ld> lineLineIntersect(vec2 p1, vec2 v1, vec2 p2, vec2 v2) {
    if(cross(v1, v2) == 0){
        return {};
    }
    ld s = cross(sub(p2, p1), v2) / cross(v1, v2);
    ld t = cross(sub(p1, p2), v1) / cross(v2, v1);
    return {s, t};
}

ld tri_area(vec2 t1, vec2 t2, vec2 t3) {
    vec2 v1 = sub(t1, t2);
    vec2 v2 = sub(t2, t3);
    return abs(cross(v1, v2) / 2.0);
}

//returns the distance along the ray from ray_a to the nearest point on the circle.
ld rayCircleIntersect(vec2 ray_a, vec2 ray_b, vec2 center, ld radius) {
    vec2 ray_dir = normalize(sub(ray_b, ray_a));
    vec2 to_center = sub(center, ray_a);
    vec2 center_proj = add(ray_a, mul(ray_dir, dot(ray_dir, to_center)));
    ld center_proj_len = length(sub(center, center_proj));
    //radius^2 = center_proj_len^2 + int_depth^2
    //int_depth = sqrt(radius^2 - center_proj_len^2)
    ld int_depth = sqrt(radius * radius - center_proj_len * center_proj_len);
    return dot(ray_dir, to_center) - int_depth;
}

//sector area of circle
ld sector_area(ld theta, ld radius) {
    return radius * radius * pi * ((theta) / (2.0 * pi));
}

ld chord_area(ld theta, ld radius) {
    ld sector = sector_area(theta, radius);
    ld tri_area = radius * radius * cos(theta) * sin(theta);
    return sector - tri_area;
```

```
}

//dist = distance from center
ld chord_area_dist(ld dist, ld radius) {
    ld theta = acos(dist / radius);
    return chord_area(theta, radius);
}

//length of chord
ld chord_area_length(ld length, ld radius) {
    ld theta = asin((length / 2.0) / radius);
    return chord_area(theta, radius);
}

//given a point inside and outside a circle, find the point along the line that intersects the circle.
vec2 find_circle_intersect(vec2 in, vec2 out, vec2 c_center, ld c_radius) {
    //just binary search :D
    //i think we can reduce this to some sort of quadratic.
    ld low = 0;
    ld high = 1;
    ld len = length(sub(in, out));
    vec2 norm = normalize(sub(out, in));
    while(abs(high - low) > epsilon) {
        ld mid = (high + low) / 2.0;
        vec2 mid_pt = add(in, mul(norm, len * mid));
        ld mid_dist = length(sub(mid_pt, c_center));
        if(mid_dist < c_radius) {
            low = mid;
        }
        else {
            high = mid;
        }
    }
    return add(in, mul(norm, len * low));
}

//returns the area of the polygon.
//winding direction doesn't matter
//polygon can be self intersecting i think...
ld polygon_area(vector<vec2>& poly) {
    ld area = 0;
    for(int i = 0; i < poly.size(); i++){
        vec2 v0 = poly[i];
        vec2 v1 = poly[(i + 1) % poly.size()];
        area += cross(v0, v1);
    }
    return abs(area / 2.0);
}

//assuming that the density of the polygon is uniform, the centroid is the center of mass.
//winding direction matters...
vec2 polygon_centroid(vector<vec2>& poly) {
    vec2 c = vec2();
    for(int i = 0; i < poly.size(); i++){
        vec2 v0 = poly[i];
        vec2 v1 = poly[(i + 1) % poly.size()];
        ld p = cross(v0, v1);
        c.x += (v0.x + v1.x) * p;
        c.y += (v0.y + v1.y) * p;
    }
    ld area = polygon_area(poly);
    c.x /= (6.0 * area);
    c.y /= (6.0 * area);
    return c;
}

//i believe this gives in CCW order, have to verify though.
vector<vec2> convex_hull(vector<vec2> a, bool include_collinear = false) {
    function<int(vec2, vec2, vec2)> orientation = [](vec2 a, vec2 b, vec2 c) -> int {
        ld v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
        if (v < 0) return -1; // clockwise
        if (v > 0) return +1; // counter-clockwise
        return 0;
    };

    function<bool(vec2, vec2, vec2)> collinear = [&orientation](vec2 a, vec2 b, vec2 c) -> bool {
        return orientation(a, b, c) == 0;
    };

    function<bool(vec2, vec2, vec2, bool)> cw = [&orientation](vec2 a, vec2 b, vec2 c, bool include_collinear) -> bool {
        int o = orientation(a, b, c);
        return o < 0 || (include_collinear && o == 0);
    };

    vec2 p0 = *min_element(a.begin(), a.end(), [](vec2 a, vec2 b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0, &orientation](const vec2& a, const vec2& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size()-1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }

    vector<vec2> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i], include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }

    //make sure there are no duplicate vertices
    vector<vec2> ans(0);
    for(int i = 0; i < st.size(); i++){
        vec2 v0 = st[i];
        vec2 v1 = st[(i + 1) % st.size()];
        if(v0.x == v1.x && v0.y == v1.y) {
            continue;
        }
        ans.push_back(st[i]);
    }

    return ans;
}

//checks if the area of the triangle is the same as the three triangle areas formed by drawing lines from pt to the vertices.
//i don't think triangle winding order matters
```

```
bool point_inside_triangle(vec2 pt, vec2 t0, vec2 t1, vec2 t2) {
    ld a1 = abs(cross(sub(t1, t0), sub(t2, t0)));
    ld a2 = abs(cross(sub(t0, pt), sub(t1, pt))) + abs(cross(sub(t1, pt), sub(t2, pt))) + abs(cross(sub(t2, pt), sub(t0, pt)));
    return abs(a1 - a2) < epsilon;
}

//runs in O(n * log(n)) time.
//has to do O(n * log(n)) preprocessing, but after preprocessing can answer queries online in O(log(n))
vector<bool> points_inside_convex_hull(vector<vec2>& pts, vector<vec2>& hull) {
    vector<bool> ans(pts.size(), false);

    //edge case
    if(hull.size() <= 2){
        return ans;
    }

    //find point of hull that has minimum x coordinate
    //if multiple elements have same x, then minimum y.
    int pivot_ind = 0;
    for(int i = 1; i < hull.size(); i++){
        if(hull[i].x < hull[pivot_ind].x || (hull[i].x == hull[pivot_ind].x && hull[i].y < hull[pivot_ind].y)) {
            pivot_ind = i;
        }
    }

    //sort all the remaining elements according to polar angle to the pivot
    vector<vec2> h_pts(0);
    vec2 pivot = hull[pivot_ind];
    for(int i = 0; i < hull.size(); i++){
        if(i != pivot_ind) {
            h_pts.push_back(hull[i]);
        }
    }
    sort(h_pts.begin(), h_pts.end(), [&pivot](vec2& a, vec2& b) -> bool {
        return polar_angle(sub(a, pivot)) < polar_angle(sub(b, pivot));
    });

    //for each point we want to check, compute it's polar angle, then binary search for the sector that should contain it
    for(int i = 0; i < pts.size(); i++){
        vec2 pt = pts[i];
        ld pt_ang = polar_angle(sub(pt, pivot));
        int low = 0;
        int high = h_pts.size() - 2;
        int tri_ind = low;
        while(low <= high) {
            int mid = low + (high - low) / 2;
            if(polar_angle(sub(h_pts[mid], pivot)) <= pt_ang) {
                tri_ind = max(tri_ind, mid);
                low = mid + 1;
            }
            else {
                high = mid - 1;
            }
        }
        ans[i] = point_inside_triangle(pt, pivot, h_pts[tri_ind], h_pts[tri_ind + 1]);
    }

    return ans;
}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m;
    cin >> n >> m;
    vector<vec2> topping(n), crust(m);
    for(int i = 0; i < n; i++){
        cin >> topping[i].x >> topping[i].y;
    }
    for(int i = 0; i < m; i++){
        cin >> crust[i].x >> crust[i].y;
    }
    crust = convex_hull(crust);
```

convolution_ntt_optimized.cpp

```
onst int mod = (119 << 23) + 1;
const int N = 1e6 + 1;
```

```
int fac[N], inv_fac[N];

void pre(){
    fac[0] = fac[1] = 1;
    inv_fac[0] = inv_fac[1] = 1;
    for (int i = 2; i < N; ++i) {
        inv_fac[i] = mod - 1ll * mod / i * inv_fac[mod % i] % mod;
    }
    for (int i = 2; i < N; ++i) {
        fac[i] = 1ll * i * fac[i - 1] % mod;
        inv_fac[i] = 1ll * inv_fac[i] * inv_fac[i - 1] % mod;
    }
}

ll fast_power(ll a, ll b){
    ll res = 1;
    while (b){
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

ll mod_inv(ll a){
    return fast_power(a, mod - 2);
}

const int root = fast_power(3, 119);
const int root_inv = mod_inv(root);
const int root_pw = 23;

int reverse(int num, int lg_n){
    int res = 0;
    for (int i = 0; i < lg_n; ++i) {
        if (num & (1 << i)) res |= 1 << (lg_n - 1 - i);
    }
    return res;
}

vector<vector<int>> roots, roots_inv;
vector<int> rev;

void pre(int n){
    int lg_n = __lg(n);
    rev.resize(n);
    roots.resize(lg_n);
    roots_inv.resize(lg_n);
    for (int i = 0; i < n; ++i) {
        rev[i] = reverse(i, lg_n);
    }

    int wlen = root, wlen_inv = root_inv;
    for (int i = 0; i < root_pw - lg_n; ++i) {
        wlen = 1ll * wlen * wlen % mod;
        wlen_inv = 1ll * wlen_inv * wlen_inv % mod;
    }

    for (int l = lg_n - 1; l >= 0 ; --l) {
        int len = 1 << (l + 1);
        int w = 1, w_inv = 1;
        roots[l].resize(len / 2);
        roots_inv[l].resize(len / 2);
        for (int i = 0; i < len / 2; ++i) {
            roots[l][i] = w;
            roots_inv[l][i] = w_inv;
            w = 1ll * w * wlen % mod;
            w_inv = 1ll * w_inv * wlen_inv % mod;
        }
        wlen = 1ll * wlen * wlen % mod;
        wlen_inv = 1ll * wlen_inv * wlen_inv % mod;
    }
}

void ntt(vector<int>& a, bool invert){
    int n = a.size();
    for (int i = 1; i < n; ++i) {
        if (i < rev[i]) swap(a[i], a[rev[i]]);
    }
    int mx = __lg(n);
    for (int l = 0; l < mx; ++l) {
        int len = 1 << (l + 1), shift = 1 << l;
        for (int i = 0; i < n; i += len) {
            for (int j = 0; j < len / 2; ++j) {
                int w = invert ? roots_inv[l][j] : roots[l][j];
                int u = a[i + j], v = 1ll * a[i + j + shift] * w % mod;
                a[i + j] = (u + v) % mod;
                a[i + j + shift] = (u - v + mod) % mod;
            }
        }
    }

    if (invert){
        int n_1 = mod_inv(n);
        for(int &x: a) x = 1ll * x * n_1 % mod;
    }
}

vector<int> convolve(int n, int m , vector<int> a){
    int size = n * m + 1;
    int _n = 1;
    while (_n < size) _n <<= 1;
    a.resize(_n);
    pre(_n);
    ntt(a, false);
    for (int i = 0; i < _n; ++i) a[i] = fast_power(a[i], n);
    ntt(a, true);
    a.resize(size);
```

convolution_using_NTT_kactl.cpp

```
/*
  Convolution two polynomial in O(n log n) instead of O(n^2)
*/

#define vi vector<int>
#define rep(x,l,r) for(int x = l; x < r; ++x)
#define sz(x) (size(x))
const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
ll modpow(ll b, ll e) {
    ll ans = 1;
    for (; e; b = b * b % mod, e /= 2)
        if (e & 1) ans = ans * b % mod;
    return ans;
}
typedef vector<ll> vl;
```

```
void ntt(vl &a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vl rt(2, 1);
  for (static int k = 2, s = 2; k < n; k *= 2, s++) {
    rt.resize(n);
    ll z[] = {1, modpow(root, mod >> s)};
    rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
      ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
      a[i + j + k] = ai - z + (z > ai ? mod : 0);
      ai += (ai + z >= mod ? z - mod : z);
    }
}
vl conv(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
    n = 1 << B;
  int inv = modpow(n, mod - 2);
  vl L(a), R(b), out(n);
  L.resize(n), R.resize(n);
  ntt(L), ntt(R);
  rep(i,0,n)
    out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
  ntt(out);
  return {out.begin(), out.begin() + s};
}
```

derangement.cpp

```
#include <bits/stdc++.h>
typedef long long ll;
#define s second
#define f first
const int N = 1e6;
using namespace std;
ll D[N]
int main(){
  /*
   ----------- Derangements----------------
   derangement is a permutation of n elements with no fixed point
   there is no i such that p_i == i
   recursive Formula:
   1.
    D(n)=n*D(n−1)+(−1)^n
   2.
    D(n)=(n−1)(D(n−1)+D(n−2))
   Find more information : https://brilliant.org/wiki/derangements/
  */
  D[2] = 1; // Base-case
  for(int i = 3 ; i < N ; i++){
    D[i] = i * D[i - 1] + pow(-1 , i);
  }
}
```

evaluating_polynomial_at_n_points.cpp

```
//evalating a polynomial at a number of points under a mod, the length of the polynomial is a power of 2
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;

int n;
ll m, q, a[1 << 20], q_pow[1 << 20];

vector<ll> dft(int k = 1, int idx = 0) {
  if (k == n) return {a[idx]};
  else {
    vector<ll> even = dft(k * 2, idx);
    vector<ll> odd = dft(k * 2, idx | k);

    int mid = n / k / 2;
    vector<ll> ans;
    for (int i = 0; i < 2 * mid; i++)
      ans.push_back((even[i % mid] + q_pow[k * i] * odd[i % mid] % m) %
             m);
    return ans;
  }
}
```

```

int main() {
    cin.tie(0)->sync_with_stdio(0);
    cin >> n >> m >> q;
    for (int i = 0; i < n; i++) cin >> a[i];

    q_pow[0] = 1;
    for (int i = 1; i < n; i++) q_pow[i] = q * q_pow[i - 1] % m;

    vector<ll> ans = dft();
    ll tot = 0;
    for (ll i : ans) tot = (tot + i) % m;
    cout << tot << '\n';
    for (int i = 1; i < n; i++) cout << ans[i] << ' ';
    cout << ans[0];
    return 0;
}
```

lucas.cpp

```
const int N = 2e6 + 100;
const int mod = 1e6 + 3;
ll fact[N];
ll inv[N]; //mod inverse for i
ll invfact[N]; //mod inverse for i!
void init() {
    fact[0] = inv[1] = fact[1] = invfact[0] = invfact[1] = 1;
    for (long long i = 2; i < N; i++) {
        fact[i] = (fact[i - 1] * i) % mod;
        inv[i] = mod - (inv[mod % i] * (mod / i) % mod);
        invfact[i] = (inv[i] * invfact[i - 1]) % mod;
    }
}
ll nCr(int n, int r) {
    if(r > n || n < 0 || r < 0) return 0; // manual handling
    return (((fact[n] * invfact[r]) % mod) * invfact[n - r]) % mod;
}
int lucas(int n , int r){
    if(r == 0) return 1;
    int res = 1;
    while(r){
        res = 1LL * res * nCr(n % mod , r % mod) % mod;
        n /= mod; r/= mod;
    }
    return res;
}
```

nCr.cpp

```
ll bpow(ll n, ll x) { return !x ? 1 : bpow(n * n % mod, x >> 1) * (x & 1 ? n : 1) % mod; }
ll inv(ll b) { return bpow(b, mod - 2); }
ll fact[N], factinv[N];
void init(){for (int i = 0; i < N; ++i) fact[i] = i ? fact[i - 1] * i % mod : 1, factinv[i] =
inv(fact[i]);}
ll C(ll n, ll k) { return fact[n] * factinv[n - k] % mod * factinv[k] % mod; }
```

nCr_DP.cpp

```
const int N = 61;
ll nCr[N][N];
void gen(){
    nCr[0][0] = 1;
    for (int i = 0; i < N; ++i) {
        nCr[i][0] = nCr[i][i] = 1;
        for (int j = 1; j < i; ++j) {
            nCr[i][j] = (nCr[i-1][j] + nCr[i-1][j-1]);
        }
    }
}
```

nCr_O(r).cpp

```
const int N = 5000;
const int mod = 1e9 + 7;
ll fact[N];
ll inv[N]; //mod inverse for i
ll invfact[N]; //mod inverse for i!
void init() {
    fact[0] = inv[1] = fact[1] = invfact[0] = invfact[1] = 1;
    for (long long i = 2; i < N; i++) {
        fact[i] = (fact[i - 1] * i) % mod;
        inv[i] = mod - (inv[mod % i] * (mod / i) % mod);
        invfact[i] = (inv[i] * invfact[i - 1]) % mod;
    }
}

ll nCr(int n, int r) {
    if (r > n) {
        return 0;
    }
    if(r < 0) return 1;
    ll res = 1;
    for (int i = 0; i < r; ++i) {
        res = res * (n - i) % mod;
    }
    res = res * invfact[r] % mod;
    return res;
}
```

nCr_mod2.cpp

```
int C(int n, int k) {
    while (n > 0 || k > 0) {
        if ((k & 1) > (n & 1)) {
            return 0;
        }
        n >>= 1;
        k >>= 1;
    }
    return 1;
}
```

ncr_O(n).cpp

```
const int N = 5e6 + 100;
const int mod = 1e9 + 7;
ll fact[N];
ll inv[N]; //mod inverse for i
ll invfact[N]; //mod inverse for i!
void init() {
    fact[0] = inv[1] = fact[1] = invfact[0] = invfact[1] = 1;
    for (long long i = 2; i < N; i++) {
```

```
    fact[i] = (fact[i - 1] * i) % mod;
    inv[i] = mod - (inv[mod % i] * (mod / i) % mod);
    invfact[i] = (inv[i] * invfact[i - 1]) % mod;
  }
}

ll nCr(int n, int r) {
  if(r > n || n < 0 || r < 0) return 0; // manual handling
  return (((fact[n] * invfact[r]) % mod) * invfact[n - r]) %
      mod;
}
```

ntt_optimized.cpp

```
ll fast_power(ll a, ll b){
  ll res = 1;
  while (b){
    if (b & 1) res = res * a % mod;
    a = a * a % mod;
    b >>= 1;
  }
  return res;
}

ll mod_inv(ll a){
  return fast_power(a, mod - 2);
}

const int root = fast_power(3, 119);
const int root_inv = mod_inv(root);
const int root_pw = 23;

int reverse(int num, int lg_n){
  int res = 0;
  for (int i = 0; i < lg_n; ++i) {
    if (num & (1 << i)) res |= 1 << (lg_n - 1 - i);
  }
  return res;
}

vector<vector<int>> roots, roots_inv;
vector<int> rev;
```

```

void pre(int n){
    int lg_n = __lg(n);
    rev.resize(n);
    roots.resize(lg_n);
    roots_inv.resize(lg_n);
    for (int i = 0; i < n; ++i) {
        rev[i] = reverse(i, lg_n);
    }

    int wlen = root, wlen_inv = root_inv;
    for (int i = 0; i < root_pw - lg_n; ++i) {
        wlen = 1ll * wlen * wlen % mod;
        wlen_inv = 1ll * wlen_inv * wlen_inv % mod;
    }

    for (int l = lg_n - 1; l >= 0 ; --l) {
        int len = 1 << (l + 1);
        int w = 1, w_inv = 1;
        roots[l].resize(len / 2);
        roots_inv[l].resize(len / 2);
        for (int i = 0; i < len / 2; ++i) {
            roots[l][i] = w;
            roots_inv[l][i] = w_inv;
            w = 1ll * w * wlen % mod;
            w_inv = 1ll * w_inv * wlen_inv % mod;
        }
        wlen = 1ll * wlen * wlen % mod;
        wlen_inv = 1ll * wlen_inv * wlen_inv % mod;
    }
}

void ntt(vector<int>& a, bool invert){
    int n = a.size();
    for (int i = 1; i < n; ++i) {
        if (i < rev[i]) swap(a[i], a[rev[i]]);
    }
    int mx = __lg(n);
    for (int l = 0; l < mx; ++l) {
        int len = 1 << (l + 1), shift = 1 << l;
        for (int i = 0; i < n; i += len) {
            for (int j = 0; j < len / 2; ++j) {
                int w = invert ? roots_inv[l][j] : roots[l][j];
                int u = a[i + j], v = 1ll * a[i + j + shift] * w % mod;
```

```cpp
                a[i + j] = (u + v) % mod;
                a[i + j + shift] = (u - v + mod) % mod;
            }
        }
    }

    if (invert){
        int n_1 = mod_inv(n);
        for(int &x: a) x = 1ll * x * n_1 % mod;
    }
}

vector<int> convolve(int n, int m , vector<int> a){
    int size = n * m + 1;
    int _n = 1;
    while (_n < size) _n <<= 1;
    a.resize(_n);
    pre(_n);
    ntt(a, false);
    for (int i = 0; i < _n; ++i) a[i] = fast_power(a[i], n);
    ntt(a, true);
    a.resize(size);
    return a;
}
```

simpson_integration.cpp

```cpp
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;
}
```

startbars.cpp

```cpp
const int mod = 1e9 + 7;
const int N = 5e6 + 9;
ll fac[N], inv[N];
void preprocess() {
    for (int i = 0; i < N; i++) {
        if (i < 2) {
            fac[i] = inv[i] = 1;
        } else {
            fac[i] = 1ll * i * fac[i - 1] % mod;
            inv[i] = mod - 1ll * mod / i * inv[mod % i] % mod;
        }
    }
    for (int i = 2; i < N; i++) {
        inv[i] = 1ll * inv[i] * inv[i - 1] % mod;
    }
}
ll ncr(ll n, ll r) {
    if(r > n || n < 0 || r < 0) return 0; // manual handling
    return 1ll * fac[n] * (1ll * inv[r] * inv[n - r] % mod) % mod;
}
ll starsBars(ll stars, ll boxes) {
    if (boxes == 0) return stars == 0;
    return ncr(stars - 1 , boxes - 1) ;
}
```

stirling.cpp

```cpp
/*
    #Stirling numbe
    1. S(r, n), represents the number of ways that we can arrange r objects around indistinguishable circles of length n,
    and every circle n must have at least one object around it.
    2. S(n,k) as the different ways to cut n different elements into k undifferentiated non-empty subsets. For example, S(5,3) denotes to:25

    S[i][j] = S[i-1][j-1] + s[i-1][j] *( i-1)
    Time Complexity : O(r * n)
    Auxiliary Space : O(r * n)
*/
int stirling_number(int n,int k){
    if(k==0)return n==0;
    if(n==0)return 0;
    return stirling_number(n-1,k-1)+(n-1)* stirling_number(n-1,k);

}
```

sum_xor_range.cpp

```
ll range_xor(ll l, ll r) {
    ll res = 0;
    if (l&1) res ^= l++;
    if (!(r&1)) res ^= r--;
    if ((r-l+1)/2&1) res ^= 1;
    return res;
}
```

Game Theory

Grundy 01.cpp

```
/// Include My Code Template
#include <bits/stdc++.h>
using namespace std;

/**
    Initially there are n piles.
    A pile is formed by some cells.
    Alice starts the game and they alternate turns.
    In each tern a player can pick any pile and divide it into two unequal piles.
    If a player cannot do so, he/she loses the game.
**/

#define Size 100005

int N;
int A[105];
int DP[10005];

int call(int cur){
    if(cur <= 2) return 0;
    if(DP[cur] != -1) return DP[cur];
    vector<int> grundy;
    for(int d1 = 1;d1<=cur/2;d1++){
```

```cpp
        int d2 = cur-d1;
        if(d1 != d2){
            grundy.pb(call(d1) ^ call(d2));
        }
    }
    make_unique(grundy);
    if(grundy[0] != 0) return DP[cur] = 0;
    int f = 0;
    for(int i = 1;i<grundy.size();i++){
        if(grundy[i] == f) continue;
        f++;
        if(grundy[i] != f) return DP[cur] = f;
    }
    return DP[cur] = grundy[grundy.size()-1] + 1;
}

int main(){
    int nCase;
    sf("%d",&nCase);
    mems(DP,-1);
    for(int cs = 1;cs<=nCase;cs++){
        sf("%d",&N);
        int res = 0;
        for(int i = 0;i<N;i++){
            sf("%d",&A[i]);
            res = res ^ call(A[i]);
        }
        if(res == 0) pf("Case %d: Bob\n",cs);
        else pf("Case %d: Alice\n",cs);
    }
    return 0;
}
```

Grundy 02.cpp

```cpp
/// Include My Code Template
#include <bits/stdc++.h>
using namespace std;

/**
    Initially there are n piles.
    A pile is formed by some stones.
    Alice starts the game and they alternate turns.
    In each tern a player can pick any pile and remove some stones.
    At least 1 and at most half of stones on that pile.
    If a player cannot do so, he/she loses the game.
**/

#define Size 100005

int N;
int A[1005];
int DP[10005];

int call(int cur){
    if(cur%2 == 0) return cur/2;
    while(cur%2 != 0) cur/=2;
    return cur/2;
}

int main(){
    int nCase;
    sf("%d",&nCase);
    mems(DP,-1);
    for(int cs = 1;cs<=nCase;cs++){
        sf("%d",&N);
        int res = 0;
        for(int i = 0;i<N;i++){
            sf("%d",&A[i]);
            res = res ^ call(A[i]);
        }
        if(res == 0) pf("Case %d: Bob\n",cs);
        else pf("Case %d: Alice\n",cs);
    }
    return 0;
}
```

Grundy String Game.cpp

```
/// Include My Code Template
#include <bits/stdc++.h>
using namespace std;

/**
```

```
  Given a string of dot and X.
  XXX is winning position.
  A player can place an X at any dot position.
**/


#define Size 100005

int N;
int DP[255];
vector<int> res;
string s;

int call(int cur){
  if(cur <= 0) return DP[cur] = 0;
  if(DP[cur] != -1) return DP[cur];
  vector<int> grundy;

  for(int p = 0;p<=cur/2;p++){
    int d1 = p - 2,d2 = cur - p - 2 - 1;
    grundy.push_back(call(d1) ^ call(d2));
  }

  make_unique(grundy);
  if(grundy.size() == 0 || grundy[0] != 0) return DP[cur] = 0;
  int f = 0;
  for(int i = 1;i<grundy.size();i++){
    if(grundy[i] == f) continue;
    f++;
    if(grundy[i] != f) return DP[cur] = f;
  }
  return DP[cur] = grundy[grundy.size()-1] + 1;
}

bool isPossible(int pos){
  string ss = s;
  if(ss[pos] == 'X') return false;
  ss[pos] = 'X';
  for(int i = 1;i<N-1;i++){
    if(ss[i] == 'X'){
      if(ss[i-1] == 'X' && ss[i+1] == 'X') return true;
    }
  }
  for(int i = 1;i<N-1;i++){
        if(ss[i] == 'X'){
            if(ss[i-1] == 'X' || ss[i+1] == 'X') return false;
        }else if(ss[i] == '.'){
            if(ss[i-1] == 'X' && ss[i+1] == 'X') return false;
        }
    }
    int i = 0,lastX = -1;
    int xorr = 0;
    while(i<N){
        if(ss[i] == 'X'){
            int cnt = i - lastX - 2 - 1;
            xorr = xorr ^ call(cnt);
            lastX = i;
            i++;
            break;
        }
        i++;
    }
    while(i<N){
        if(ss[i] == 'X'){
            int cnt = i - lastX - 2 - 2 - 1;
            xorr = xorr ^ call(cnt);
            lastX = i;
        }
        i++;
    }
    int cnt = i - lastX - 2 - 1;
    xorr = xorr ^ call(cnt);
    if(xorr == 0) return true;
    return false;
}

int main(){
    fast_cin;
    mems(DP,-1);
    int nCase;
    cin >> nCase;
    for(int cs = 1;cs<=nCase;cs++){
        cin >> s;
        N = s.length();
        res.clear();
        for(int i = 0;i<N;i++){
            if(isPossible(i) == true){
                res.pb(i+1);
            }
        }
        pf("Case %d:",cs);
        int Sz = res.size();
        for(int i = 0;i<Sz;i++){
            pf(" %d",res[i]);
        }
        if(Sz == 0) pf(" 0\n");
        else pf("\n");
    }
```

Grundy_Knights_Move_In_Matrix.cpp

```
/// Include My Code Template
#include <bits/stdc++.h>
using namespace std;

/**
    Given a matrix and some knights with their positions.
    Some possible moves are also given.
    A player can move any knight from it's cell using any given possible move.
    Last player who gives move win.
**/


#define Size 100005

int N;
int DP[1055][1055];
int dx[] = {1, -1, -1, -2, -3, -2};
int dy[] = {-2, -3, -2, -1, -1, 1};

bool isValid(int R,int C){
    if(R<0 || C<0 || R>=1055 || C>=1055) return false;
    return true;
}

int call(int R,int C){
    //pf("Cur R: %d , C: %d\n",R,C);
    if(R == 0 && C == 0) return 0;
    if(DP[R][C] != -1) return DP[R][C];
    vector<int> grundy;
    for(int i = 0; i < 6; i++) {
        int nR = R + dx[i];
        int nC = C + dy[i];
        if(isValid(nR,nC) == false) continue;
        grundy.pb(call(nR,nC));
    }
    make_unique(grundy);
    if(grundy.size() == 0) return DP[R][C] = 0;
    if(grundy[0] != 0) return DP[R][C] = 0;
    int f = 0;
    for(int i = 1;i<grundy.size();i++){
        if(grundy[i] == f) continue;
        f++;
        if(grundy[i] != f) return DP[R][C] = f;
    }
    return DP[R][C] = grundy[grundy.size()-1] + 1;
}

int main(){
    int nCase,x,y;
    sf("%d",&nCase);
    mems(DP,-1);
    for(int cs = 1;cs<=nCase;cs++){
        sf("%d",&N);
        int res = 0;
        for(int i = 0;i<N;i++){
            sf("%d %d",&x,&y);
            res = res ^ call(x,y);
        }
        if(res == 0) pf("Case %d: Bob\n",cs);
        else pf("Case %d: Alice\n",cs);
    }
    return 0;
}
```

Matrix Nim.cpp

```
/// Include My Code Template
#include <bits/stdc++.h>
using namespace std;
```

```cpp

/**
    Each cell of matrix is a pile.
    In each move a player can move stone to down or right cell.
    Who gives last move win.
**/


#define INF 99999999


int main() {
    int nCase,r,c,a;
    sf("%d", &nCase);
    for (int cs = 1; cs <= nCase; cs++) {
        scanf("%d %d", &r, &c);
        int xsum = 0;
        for(int i = 1;i<=r;i++){
            for(int j = 1;j<=c;j++){
                sf("%d",&a);
                if((r+c)%2 != (i+j)%2) xsum ^= a;
            }
        }
        if(xsum != 0) printf("Case %d: win\n", cs);
        else printf("Case %d: lose\n", cs);
    }
    return 0;
}
```

Misere Nim.cpp

```cpp
/**
    In misere nim who gives last move win.
    When all the pile have 1 stone then xor sum doesn't work (where you need to handle as a special case)
**/

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define f first
#define s second
```

```cpp
int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);

  int tc; cin >> tc;
  while(tc--){
    int n; cin >> n;
    int xor_sum = 0 , ones = 0;
    for(int i = 0; i < n; ++i){
      int x; cin >> x;
      ones += (x==1);
      xor_sum ^= x;
    }
    if(ones == n) cout << (n&1 ? "Second\n" : "First\n");
    else cout << (xor_sum ? "First\n" : "Second\n");
  }
}
```

Nim 1.cpp

```cpp
/// Include My Code Template
#include <bits/stdc++.h>
using namespace std;

/**
  Given N piles a player can remove any number of stone from a pile to it's left.
**/

#define INF 99999999

int main() {
  int nCase,n,a,b;
  sf("%d", &nCase);
  for (int cs = 1; cs <= nCase; cs++) {
    scanf("%d", &n);
    int xsum = 0;
    for(int i = 0; i < n; i++) {
      scanf("%d", &a);
      xsum ^= (a);
    }
    if(xsum != 0) printf("Case %d: Alice\n", cs);
    else printf("Case %d: Bob\n", cs);
  }
```

```cpp
    return 0;
}
```

Staircase Nim.cpp

```cpp
/// Include My Code Template
#include <bits/stdc++.h>
using namespace std;

/**
    Given a tree and coins in each vertex.
    A player can move any no of coins to it's parent vertex.
    Last player move wins.
    In this problem for query a parent can be changed in the tree.
**/


#define Size 100005

int n, cnt, q, uu, vv;
int st[100005], en[100005], color[100005], parent[100005];
int whiteXor[100005], blackXor[100005], c[100005];
vector<int> Graph[100005];

void dfs(int pos, int par, int colr) {
    cnt++;
    st[pos] = cnt;
    color[pos] = colr;
    parent[pos] = par;

    if(colr == 0) whiteXor[pos] = c[pos];
    else blackXor[pos] = c[pos];

    for (int i=0; i<Graph[pos].size(); i++) {
        int j = Graph[pos][i];
        if (j == par) continue;
        dfs(j,pos,colr^1);
        blackXor[pos] ^= blackXor[j];
        whiteXor[pos] ^= whiteXor[j];
    }
    en[pos] = cnt;
    return;
}

int main () {
    scanf("%d", &n);
    for (int i=1; i<=n; i++) {
        scanf("%d", &c[i]);
    }
    for (int i=2; i<=n; i++) {
        scanf("%d %d", &uu, &vv);
        Graph[uu].pb(vv);
        Graph[vv].pb(uu);
    }
    cnt = 0;
    dfs(1,1,1);
    scanf("%d", &q);
    while (q--) {
        scanf("%d %d", &uu, &vv);
        if ( st[uu] <= st[vv] && en[uu] >= en[vv] ) {
            printf("INVALID\n");
        } else {
            int cww = whiteXor[1], cbb = blackXor[1];
            if (color[parent[uu]] == color[vv]) {
                if (cww > 0) {
                    printf("YES\n");
                } else {
                    printf("NO\n");
                }
            } else {
                cww ^= whiteXor[uu];
                cww ^= blackXor[uu];
                if (cww > 0) {
                    printf("YES\n");
                } else {
                    printf("NO\n");
                }
            }
        }
    }
    return 0;
}
```

General

__int128.cpp

```cpp
// Define int128 types and I/O operators
typedef __int128 int128;
typedef unsigned __int128 uint128;

// Helper functions for int128 I/O
ostream& operator << (ostream &os, int128 num) {
    string str;
    if(num == 0) return os << "0";
    bool neg = false;
    if(num < 0) {
        neg = true;
        num = -num;
    }
    while(num) {
        str.push_back('0' + num % 10);
        num /= 10;
    }
    if(neg) str.push_back('-');
    reverse(str.begin(), str.end());
    return os << str;
}

istream& operator >> (istream &is, int128 &num) {
    string str;
    is >> str;
    num = 0;
    bool neg = false;
    for(char c : str) {
        if(c == '-') neg = true;
        else num = num * 10 + (c - '0');
    }
    if(neg) num = -num;
    return is;
}
```

comp_double.cpp

```cpp
#define ld long double
const ld EPS = 1e-6;
int dcmp(const ld &a, const ld &b) {
  // Double compare
  if (fabs(a - b) < EPS)
      return 0;

  return (a > b ? 1 : -1);
}
```

convert_double_int.cpp

```
int EPS = 10000 , Precision = 4;
long long read(){
    // 1.2345 --> 12345
    string s;cin >> s;
    int x=s.find('.');
    if(x==-1) return stoll(s+"0000");
    string one=s.substr(0,x);
    string two=s.substr(x+1);
    while(two.size()< Precision) two+="0";
    return stoll(one+two);
}
```

floor_ceil.cpp

```
ll ceil(ll a, ll b){
    if(b < 0) a *= -1, b *= -1;
    if(a < 0) return a / b;
    else return (a + b - 1) / b;
}

ll floor(ll a, ll b){
    if(b < 0) a *= -1, b *= -1;
    if(a > 0) return a / b;
    else return (a - b + 1) / b;
}
```

fraction.cpp

```
long long gcd (long long a, long long b) {
  while (b) {
      a %= b;
      swap(a, b);
  }
  return a;
}
struct frac{
  // fraction n / d
  long long n ,d;
  // constructor
  frac(){n = 0; d = 1;}
  frac(ll n , ll d): n(n) , d(d){simplify();};
  bool operator < (const frac &other) const{
    return n * other.d < d * other.n;
  }
  frac operator + (const frac &f) const{
    frac ans {n * f.d + f.n * d , d * f.d};
    ans.simplify();
    return ans;
  }
  frac operator - (const frac &f) const{
    frac ans {n * f.d - f.n * d , d * f.d};
    ans.simplify();
    return ans;
  }
  void Abs(){n = abs(n); d = abs(d);}
  void simplify(){
    long long g = gcd(abs(n) , abs(d));
    n /= g; d /= g;
  }
};
```

generate_n_digits_after_point.cpp

```
/*
  ========== having a / b , generate n digits after point.=============
  e.g : 1 / 3 = 0.333333333333333333333333333333
*/
ll a , b; cin >> a >> b;
vector<int> v(n);
for(int i = 0; i < n; i++){
  v[i] = a / b;
  a = (a%b) * 10;
}
```

manual_multiply.cpp

```
#include <bits/stdc++.h>
typedef long long ll;
#define s second
#define f first
#define rep(i , st , ed) for(int i = st ; i < ed ; i++)
using namespace std;
void burn(){
ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
#ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
 #endif
}
//\//\//\/\//\//\//\//\//\//

int main(){
 burn();
 string s1 , s2; cin>> s1 >> s2;
 reverse(s1.begin(), s1.end());
 reverse(s2.begin(), s2.end());
 int MXN = 2 * max(s1.size() , s2.size())+ 9;
 int a[MXN];
 memset(a, 0, sizeof(a));
 for (int i=0; i<s1.length(); i++) {
  for (int j=0; j<s2.length(); j++) {
   a[i + j] += (s1[i]-'0') * (s2[j] - '0');
  }
 }
 for (int i=0; i<MXN - 1; i++) {
  a[i + 1] += a[i] / 10;
  a[i] %= 10;
 }
 int i = MXN - 1;
 while (i > 0 && a[i] == 0) i--;
 for (; i>=0; i--) cout<<a[i];
 cout<<endl;
}
```

random.cpp

```cpp
mt19937 random_seed(time(0));
long long rnd(long long l , long long r){
    uniform_int_distribution<long long> dist(l, r);
    return dist(random_seed);
}
```

Geometry

3dGeometry.cpp

```cpp
struct Point {
    double x, y, z;
};
long double eps = 1.2e-6;
Point cross(Point a, Point b, Point c) {
    Point result;
    result.x = (b.y - a.y) * (c.z - a.z) - (b.z - a.z) * (c.y - a.y);
    result.y = (b.z - a.z) * (c.x - a.x) - (b.x - a.x) * (c.z - a.z);
    result.z = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
    return result;
}

double dot(Point a, Point b) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}

double TetrahedronVolume(Point a, Point b, Point c, Point d) {
    Point crossProduct = cross(a, b, c);
    Point vectorAD = {d.x - a.x, d.y - a.y, d.z - a.z};
    return dot(crossProduct, vectorAD) / 6.0;
}

bool intersect(Point a, Point b, Point c, Point p, Point q) {
    if (TetrahedronVolume(p, a, b, c) * TetrahedronVolume(q, a, b, c) < 0 &&
        TetrahedronVolume(p, q, a, b) * TetrahedronVolume(p, q, b, c) > 0 &&
        TetrahedronVolume(p, q, b, c) * TetrahedronVolume(p, q, c, a) > 0)
    {
        return true;
    }
    return false;
}
```

Distinct_line_detecting.cpp

```
//ay + bx = c
void fix(int &a, int &b, int &c){
  int g = __gcd(__gcd(a,b),c);
  a/=g,b/=g,c/=g;
  if(a < 0 || (a == 0 && b < 0))
  {
    a*=-1, b*=-1, c*=-1;
  }
}
array<int,3> Line(int dx, int dy, Point P)
{
  int a = dx;
  int b = -dy;
  int c = P.y() * a + P.x() * b;
  fix(a,b,c);
  return {a,b,c};
}
```

Geometry.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
#define all(v) v.begin() , v.end()
#ifndef ONLINE_JUDGE
#define debug(x) cerr << #x << ": " << x << '\n';
#else
#define debug(x)
#endif
#define ld long double
const ld EPS = 1e-6;
#define vec(a,b) ((b) - (a))
template<typename T = double> struct Point {
  typedef Point P;
```

```
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
    P operator*(T d) const { return P(x*d, y*d); }
    P operator/(T d) const { return P(x/d, y/d); }
    T dot(P p) const { return x*p.x + y*p.y; }
    T cross(P p) const { return x*p.y - y*p.x; }
    T cross(P a, P b) const { return (a-*this).cross(b-*this); }
    T dist2() const { return x*x + y*y; }
    double dist() const { return sqrt((double)dist2()); }
    // angle to x-axis in interval [-pi, pi]
    double angle() const { return atan2(y, x); }
    P unit() const { return *this/dist(); } // makes dist()=1
    P perp() const { return P(-y, x); } // rotates +90 degrees
    P normal() const { return perp().unit(); }
    // returns point rotated 'a' radians ccw around the origin
    P rotate(double a) const {
        return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
    friend ostream& operator<<(ostream& os, P p) {
        return os << "(" << p.x << "," << p.y << ")";
    }
    friend istream &operator>>(istream &os, P &p) {
        return os >> p.x >> p.y;
    }

    // Project point onto line through a and b (assuming a != b).
    P projectOnLine(const P &a, const P &b) const {
        P ab = a.getVector(b);
        P ac = a.getVector(*this);
        return a + ab * ac.dot(ab) / a.dist2(b);
    }

    // Project point c onto line segment through a and b (assuming a != b).
    P projectOnSegment(const P &a, const P &b) const {
        P &c = *this;
        P ab = a.getVector(b);
        P ac = a.getVector(c);

        long double r = dot(ac, ab), d = a.dist2(b);
        if (r < 0) return a;
        if (r > d) return b;

        return a + ab * r / d;
    }

    P reflectAroundLine(const P &a, const P &b) const {
        return projectOnLine(a, b) * 2 - (*this);
    }

};
int dcmp(const ld &a, const ld &b) {
    // Double compare
    if (fabs(a - b) < EPS)
        return 0;

    return (a > b ? 1 : -1);
}
// length of vector
double length(const Point<> a){
    return sqrt(a.x * a.x + a.y * a.y);
}
// cross product
double cross(const Point<> &a, const Point<> &b){
    return a.x * b.y - b.x * a.y;
}
// Check if there is intersect two lines or not and return the intersection point
bool intersect(const Point<> &a, const Point<> &b,
        const Point<> &p, const Point<> &q,  Point<> &ret) {
    //handle degenerate cases (2 parallel lines, 2 identical lines,   line is 1 point)
    double d1 = cross(p - a, b - a);
    double d2 = cross(q - a, b - a);
    ret = (q * d1  - p * d2) / (d1 - d2);
    if(fabs(d1 - d2) > EPS) return 1;
    return 0;
}

// dot product
double dot(const Point<> a, const Point<> b){ return a.x * b.x + a.y * b.y; }

// Point On Line
bool pointOnLine(const Point<>& a, const Point<>& b, const Point<>& p) {
    // determine the point "p" is in the line or not
    return fabs(cross(vec(a,b),vec(a,p))) < EPS;
}
```

```
// Is Point On Ray
bool pointOnRay(const Point<>& a, const Point<>& b, const Point<>& p) {
  //IMP NOTE: a,b,p must be collinear
  return dot(vec(a,p), vec(a,b)) > -EPS;
}
// Point On Segment
bool pointOnSegment(const Point<>& a, const Point<>& b, const Point<>& p) {
    //el satr da momken y3mel precision error
  if(!pointOnLine(a,b,p)) return 0;
  return pointOnRay(a, b, p) && pointOnRay(b, a, p);
}
//Point Line Dist
double pointLineDist(const Point<>& a, const Point<>& b, const Point<>& p) {
 // shortest distance between line and point
 return fabs(cross(vec(a,b),vec(a,p)) / length(vec(a,b)));
}
// Point Segment Dist
double pointSegmentDist(const Point<> &a, const Point<> &b,const Point<> &p){
   // shortest distance between segment and point
    if (dot(vec(a,b),vec(a,p)) < EPS)
        return length(vec(a,p));
    if (dot(vec(b,a),vec(b,p)) < EPS)
        return length(vec(b,p));
    return pointLineDist(a, b, p);
}
// Count the number of Lattice Point in segment
int segmentLatticePointsCount(int x1, int y1, int x2, int y2) {
    return abs(__gcd(x1 - x2, y1 - y2)) + 1;
}

template<class P> bool onSegment(P s, P e, P p) {
  // check if point (p) on line (s , e)
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}

template<class P> vector<P> segInter(P a, P b, P c, P d) {
  // The intersection between two segment, return the index section point
  auto oa = c.cross(d, a), ob = c.cross(d, b),
      oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
```

```
    if (onSegment(c, d, a)) s.insert(a);
    if (onSegment(c, d, b)) s.insert(b);
    if (onSegment(a, b, c)) s.insert(c);
    if (onSegment(a, b, d)) s.insert(d);
    return {s.begin(), s.end()};
}
vector<vector<Point<int>>> createLine(int x, int y, int d){
    //
    vector<vector<Point<int>>> ret;
    ret.push_back({Point<int>(x+d, y), Point<int>(x, y+d)});
    ret.push_back({Point<int>(x-d, y), Point<int>(x, y+d)});
    ret.push_back({Point<int>(x-d, y), Point<int>(x, y-d)});
    ret.push_back({Point<int>(x+d, y), Point<int>(x, y-d)});
    return ret;
}

template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
    // the intersection of line and circule
    P ab = b - a, p = a + ab * (c - a).dot(ab) / ab.dist2();
    double s = a.cross(b, c), h2 = r * r - s * s / ab.dist2();
    if (h2 < 0) return {};
    if (h2 == 0) return {p};
    P h = ab.unit() * sqrt(h2);
    return {p - h, p + h};
}
// Cosine Rule
//get angle opposite to side a
double cosRule(double a, double b, double c) {
        // Handle denom = 0
        double res = (b * b + c * c - a * a) / (2 * b * c);
        if (res > 1)
                res = 1;
        if (res < -1)
                res = -1;
        return acos(res);
}

Point<> normalize(const Point<> p){ return ((p) / length(p)); }
Point<> polar(const Point<> &r, double t){
    return Point{r.x * cos(t)  - r.y * sin(t), r.x * sin(t) + r.y * cos(t)};
}
// Circle Circle Intersection
```

```cpp
int circleCircleIntersection(const Point<> &c1, const double &r1, const Point<> &c2, const double &r2, Point<> &res1, Point<> &res2) {
    if (c1 == c2 && fabs(r1 - r2) < EPS) {
        res1 = res2 = c1;
        return fabs(r1) < EPS ? 1 : INT32_MAX;
    }
    double len = length(vec(c1,c2));
    if (fabs(len - (r1 + r2)) < EPS || fabs(fabs(r1 - r2) - len) < EPS) {
        Point<> d, c;
        double r;
        if (r1 > r2)
            d = vec(c1,c2), c = c1, r = r1;
        else
            d = vec(c2,c1), c = c2, r = r2;
        res1 = res2 = normalize(d) * r + c;
        return 1; // intersect in one point
    }
    if (len > r1 + r2 || len < fabs(r1 - r2))
        return 0; // intersect on two points
    double a = cosRule(r2, r1, len);
    Point<> c1c2 = normalize(vec(c1,c2)) * r1;
    res1 = polar(c1c2,  a) + c1;
    res2 = polar(c1c2, -a) + c1;
    return 2; // intersect in one point
}
// Circle From 3 Points
bool circle3(const Point<> &p1, const Point<> &p2, const Point<> &p3,
    Point<>& cen, double& r) {
    Point<> m1 = (p1 + p2) / 2;
    Point<> m2 = (p2 + p3) / 2;
    Point<> perp1 = vec(p1, p2);
    perp1 = perp1.perp();
    Point<> perp2 = vec(p2, p3);
    perp2 = perp2.perp();
    bool res = intersect(m1, m1 + perp1, m2, m2 + perp2, cen);
    r = length(vec(cen,p1));
    return res;
}
double lengthSqr(const Point<> a){ return a.x * a.x + a.y * a.y; }
// check Point according to circle  (in boundary, inside , outside)
int circlePoint(const Point<>  &cen, const double &r, const Point<>  &p) {
    double lensqr = lengthSqr(vec(cen,p));
    if (fabs(lensqr - r * r) < EPS)
        return 1; // In the Boundary
    if (lensqr < r * r)
        return -1; // In the circle
    return 0; // Out the circle
}
// Maximum  triangle inside a circle
double maxAreaTriangleInsideCircle(double a){
  return 3 * sqrt((double) 3) * a * a / 4;
}
// find the Slope
pair<int,int> slope(pair<int,int> u, pair<int,int> v)
{
  int dy = v.s-u.s;
  int dx = v.f-u.f;
  if(dx == 0) return {0,0};
  if(dy == 0) return{0,1};
  int sgn = (dy < 0) ^ (dx < 0);
  if(sgn) sgn = -1; else sgn = 1;
  return {sgn*abs(dx)/(abs(__gcd(dy,dx))),abs(dy)/(abs(__gcd(dy,dx)))};
}

int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
```

Point.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
using ld = long double;
using ll = long long;
using pii = pair<int,int>;
using Point = complex<ld>;
const ld EPS = 1e-9;
#define X real()
#define Y imag()

// Core geometry operations
ld cross(const Point& a, const Point& b) { return imag(conj(a) * b); }
ld dot(const Point& a, const Point& b) { return real(conj(a) * b); }
ld dist2(const Point& a, const Point& b) { return norm(a - b); }
ld dist(const Point& a, const Point& b) { return abs(a - b); }
ld angle(const Point& a, const Point& b) { return atan2(cross(a,b), dot(a,b)); }

// Orientation check
int ccw(const Point& a, const Point& b, const Point& c) {
    ld cr = cross(b-a, c-a);
    if(cr > EPS) return +1;   // CCW
    if(cr < -EPS) return -1;  // CW
    if(dot(b-a, c-a) < 0) return +2;  // c--a--b
    if(norm(b-a) < norm(c-a)) return -2; // a--b--c on line
    return 0;  // Colinear and overlapping
}

// Integer slope
pii slope(const pii& a, const pii& b) {
    int dy = b.second - a.second, dx = b.first - a.first;
    if(dx == 0) return {0,0};  // Vertical
    if(dy == 0) return {0,1};  // Horizontal
    int g = __gcd(abs(dy), abs(dx));
    dy /= g; dx /= g;
    if(dx < 0) dy = -dy, dx = -dx;
    return {dy,dx};
}

// Point operations
Point rotate(const Point& p, const Point& c, ld theta) {
    return (p-c) * polar((ld)1.0, theta) + c;
}

Point proj(const Point& p, const Point& a, const Point& b) {
    Point ab = b-a;
    return a + ab * dot(p-a, ab) / norm(ab);
}

Point refl(const Point& p, const Point& a, const Point& b) {
    Point prj = proj(p,a,b);
    return prj * (ld)2.0 - p;
}

// Input/Output
istream& operator>>(istream& is, Point& p) {
    ld x,y; is >> x >> y; p = Point(x,y);
    return is;
}
```

```

ostream& operator<<(ostream& os, const Point& p) {
    return os << p.X << ' ' << p.Y;
}

void solve() {
    // Your solution here
    Point a, b;
    cin >> a >> b;
    cout << dist(a,b) << '\n';
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int t = 1;
    // cin >> t;  // Uncomment for multiple test cases
    while(t--) solve();

    return 0;
```

Triangle.cpp

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

typedef long double ld;
typedef complex<ld> Point;

inline ld cross(const Point& a, const Point& b) { return a.real() * b.imag() - a.imag() * b.real(); }
inline ld lengthSqr(const Point& p) { return norm(p); }
inline ld triangleAreaBH(ld base, ld height) { return base * height / 2.0L; }
inline ld triangleArea2SidesAngle(ld a, ld b, ld angle) { return fabs(a * b * sin(angle)) / 2.0L; }
inline ld triangleArea2AnglesSide(ld ang1, ld ang2, ld side) {
    return fabs(side * side * sin(ang1) * sin(ang2) / (2.0L * sin(ang1 + ang2)));
}
inline ld triangleArea3Sides(ld a, ld b, ld c) {
    ld s = (a + b + c) / 2.0L;
    return sqrt(s * (s - a) * (s - b) * (s - c));
```

```cpp
}
inline ld cosRule(ld a, ld b, ld c) {
    ld denom = 2.0L * b * c;
    if (fabs(denom) < 1e-9) return 0.0L;
    return acos(min(max((b * b + c * c - a * a) / denom, -1.0L), 1.0L));
}

int main() { return 0; }
```

areaOfreactangles.cpp

```cpp
/*
Given n rectange calculate the area of total rectange (take in care that rectangle can intersect)

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

using namespace std;
typedef long long ll;
typedef array<int,4> Operation;
const int maxN = 1e5;
const int SZ = 9e6;

int N, lo[SZ], hi[SZ];
ll area, delta[SZ], score[SZ];
Operation op[2*maxN];

int len(int i){
    return hi[i]-lo[i]+1;
}

void pull(int i){
    if(lo[i] == hi[i])  score[i] = (delta[i] > 0 ? 1 : 0);
    else                score[i] = (delta[i] > 0 ? len(i) : score[2*i] + score[2*i+1]);
}

void build(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r)  return;
    int m = l+(r-l)/2;
    build(2*i, l, m);
    build(2*i+1, m+1, r);
}

void increment(int i, int l, int r, ll val){
    if(l > hi[i] || r < lo[i])  return;
    if(l <= lo[i] && hi[i] <= r){
        delta[i] += val;
        pull(i);
        return;
    }
    increment(2*i, l, r, val);
    increment(2*i+1, l, r, val);
    pull(i);
}

ll query(){
    return score[1];
}

int main(){
    scanf("%d", &N);
    for(int i = 0, a, b, c, d; i < N; i++){
        scanf("%d %d %d %d", &a, &b, &c, &d);
        op[2*i] = {1, b, a+1, c};
        op[2*i+1] = {-1, d, a+1, c};
    }
    sort(op, op+2*N, [](Operation A, Operation B){
       return (A[1] == B[1] ? A[0] < B[0] : A[1] < B[1]);
    });

    build(1, -1e6-5, 1e6+5);
    int lst = -1e6;
    for(int i = 0; i < 2*N; i++){
        int t = op[i][0], y = op[i][1], x1 = op[i][2], x2 = op[i][3];
        area += (y-lst) * query();
        increment(1, x1, x2, t);
        lst = y;
    }

    printf("%lld\n", area);
}
```

circle.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
typedef complex<ld> Point;
const ld EPS = 1e-9;

// Cross product of OA and OB vectors
ld cross(const Point& O, const Point& A, const Point& B) { return imag((A - O) * conj(B - O)); }

// Dot product of vectors A and B
ld dot(const Point& A, const Point& B) { return real(A * conj(B)); }

// Circle-Line Intersection: returns intersection points
vector<Point> circleLine(const Point& c, ld r, const Point& a, const Point& b){
    Point ab = b - a; ld ab2 = norm(ab);
    ld t = dot(ab, c - a) / ab2;
    Point p = a + ab * t; ld s2 = r*r - norm(c - p);
    if(s2 < -EPS) return {};
    if(abs(s2) < EPS) return {p};
    ld s = sqrt(s2); Point h = ab / abs(ab) * s;
    return {p - h, p + h};
}

// Cosine Rule: angle opposite to side a
ld cosRule(ld a, ld b, ld c){
    ld res = (b*b + c*c - a*a)/(2*b*c);
    return acos(max(-1.0L, min(1.0L, res)));
}

// Circle-Circle Intersection: returns number of intersections and sets res1, res2
int circleCircle(const Point& c1, ld r1, const Point& c2, ld r2, Point& res1, Point& res2){
    Point d = c2 - c1; ld dist = abs(d);
    if(dist < EPS && abs(r1 - r2) < EPS) return INT32_MAX; // Infinite intersections
    if(dist > r1 + r2 + EPS || dist < abs(r1 - r2) - EPS) return 0; // No intersection
    ld a = (r1*r1 - r2*r2 + dist*dist)/(2*dist);
    ld h2 = r1*r1 - a*a;
    if(h2 < -EPS) return 0;
    ld h = h2 < EPS ? 0 : sqrt(h2);
    Point p = c1 + d * (a / dist);
    if(h == 0){ res1 = p; return 1; }
    Point offset = d * (h / dist) * Point(0,1);
    res1 = p + offset; res2 = p - offset;
    return 2;
}

// Circle from Three Points: returns true if successful
bool circle3(const Point& p1, const Point& p2, const Point& p3, Point& cen, ld& r){
    ld a = cross(p1, p2, p3);
    if(abs(a) < EPS) return false; // Collinear points
    cen = ((norm(p2) - norm(p1)) * Point(0,1) - (norm(p3) - norm(p1)) * Point(1,0)) / (2*a) + p1;
    r = abs(cen - p1);
    return true;
}

// Point Position Relative to Circle: -1 inside, 0 outside, 1 on boundary
int circlePoint(const Point& cen, ld r, const Point& p){
    ld dist2 = norm(p - cen), r2 = r*r;
    if(abs(dist2 - r2) < EPS) return 1;
    return (dist2 < r2) ? -1 : 0;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

}
```

convex_hull.cpp

```cpp
#define double long double
struct Point {
    double x, y;
};

int orientation(Point a, Point b, Point c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}

bool cw(Point a, Point b, Point c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
```

```
bool collinear(Point a, Point b, Point c) { return orientation(a, b, c) == 0; }

void convex_hull(vector<Point>& a, bool include_collinear = false) {
    Point p0 = *min_element(a.begin(), a.end(), [](Point a, Point b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0](const Point& a, const Point& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size()-1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }

    vector<Point> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i], include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }

    a = st;
}
double area(Point a, Point b, Point c)
{
    return 0.5*abs(a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y));
}
```

dynamic_convex_hull.cpp

```
typedef long double ld;
struct Line {
    ll m, b;
    mutable function<const Line *()> succ;

    bool operator<(const Line &other) const {
        return m < other.m;
```

```
  }

  bool operator<(const ll &x) const {
    const Line *s = succ();
    if (!s)
      return 0;
    return b - s->b < (s->m - m) * x;
  }
};
// will maintain upper hull for maximum
struct HullDynamic : public multiset<Line, less<>> {
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end())
        return 0;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if (z == end())
      return y->m == x->m && y->b <= x->b;
    return (x->b - y->b) * (z->m - y->m) >= (y->b - z->b) * (y->m - x->m);
  }

  void insert_line(ll m, ll b) { // log(n)
    m *= -1;
    b *= -1;
    auto y = insert({m, b});
    y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
    if (bad(y)) {
      erase(y);
      return;
    }
    while (next(y) != end() && bad(next(y)))
      erase(next(y));
    while (y != begin() && bad(prev(y)))
      erase(prev(y));
  }

  ll query(ll x) { // log(n)
    if(size() == 0) return 5e18; // degerated case
    auto l = *lower_bound(x);
    return -(l.m * x + l.b);
  }
```

```
}
```

getClosestPair.cpp

```
#define type double
#define MapIterator map<type, multiset<type> >::iterator
#define SetIterator multiset<type>::iterator

const int SIZE = 10000; //Maximum number of points
type x[SIZE], y[SIZE]; //Coordinates of points
int N; //Number of points
double INF = INT_MAX;
double getClosestPair() {
    map<type, multiset<type> > points;
    for (int i = 0; i < N; i++)
        points[x[i]].insert(y[i]);
    double d = INF;
    for (MapIterator xitr1 = points.begin(); xitr1 != points.end(); xitr1++){
    for (SetIterator yitr1 = (*xitr1).second.begin(); yitr1!= (*xitr1).second.end(); yitr1++) {
    type x1 = (*xitr1).first, y1 = *yitr1;
    MapIterator xitr3 = points.upper_bound(x1 + d);
    for (MapIterator xitr2 = xitr1; xitr2 != xitr3; xitr2++)
    {
        type x2 = (*xitr2).first;
        SetIterator yitr2 = (*xitr2).second.lower_bound(y1 - d);
        SetIterator yitr3 = (*xitr2).second.upper_bound(y1 + d);
        for (SetIterator yitr4 = yitr2; yitr4 != yitr3; yitr4++) {
            if (xitr1 == xitr2 && yitr1 == yitr4)
                continue; //same point      type y2 = *yitr4;
            d = min(d, hypot(x1 - x2, y1 - y2));
        }
    }
    }
    }
    return d;
}
```

line.cpp

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

```
typedef long double ld;
typedef complex<ld> Point;
const ld EPS = 1e-9;
#define X real()
#define Y imag()

// Geometry Helpers
ld cross(const Point& a, const Point& b) { return imag(conj(a) * b); }
ld dot(const Point& a, const Point& b) { return real(conj(a) * b); }
ld length_sq(const Point& a) { return norm(a); }
ld length_(const Point& a) { return abs(a); }

// Point on Line
bool pointOnLine(const Point& a, const Point& b, const Point& p) {
    return abs(cross(b - a, p - a)) < EPS;
}

// Point on Ray
bool pointOnRay(const Point& a, const Point& b, const Point& p) {
    return dot(p - a, b - a) > -EPS;
}

// Point on Segment
bool pointOnSegment(const Point& a, const Point& b, const Point& p) {
    return pointOnLine(a, b, p) && pointOnRay(a, b, p) && pointOnRay(b, a, p);
}

// Distance from Point to Line
ld pointLineDist(const Point& a, const Point& b, const Point& p) {
    return abs(cross(b - a, p - a)) / length_(b - a);
}

// Distance from Point to Segment
ld pointSegmentDist(const Point& a, const Point& b, const Point& p){
    if(dot(b - a, p - a) < EPS) return length_(p - a);
    if(dot(a - b, p - b) < EPS) return length_(p - b);
    return pointLineDist(a, b, p);
}

// Line Intersection
bool intersectLines(const Point& a, const Point& b, const Point& p, const Point& q, Point& r) {
    ld d1 = cross(p - a, b - a);
    ld d2 = cross(q - a, b - a);
    if(abs(d1 - d2) < EPS) return false;
    r = (q * d1 - p * d2) / (d1 - d2);
    return true;
}
vector<Point> segInter(const Point& a, const Point& b, const Point& c, const Point& d) {
    Point r;
    if(intersectLines(a, b, c, d, r)) {
        // Check if r is on both segments
        if(pointOnSegment(a, b, r) && pointOnSegment(c, d, r))
            return {r};
    }
    auto on = [&](const Point& s, const Point& e, const Point& p) -> bool {
        return pointOnSegment(s, e, p);
    };
    vector<Point> res;
    if(on(a, b, c)) res.emplace_back(c);
    if(on(a, b, d)) res.emplace_back(d);
    if(on(c, d, a)) res.emplace_back(a);
    if(on(c, d, b)) res.emplace_back(b);
    sort(res.begin(), res.end(), [&](const Point& x, const Point& y) -> bool {
        return real(x) < real(y) || (abs(real(x) - real(y)) < EPS && imag(x) < imag(y));
    });
    res.erase(unique(res.begin(), res.end(), [&](const Point& x, const Point& y) -> bool {
        return abs(x - y) < EPS;
    }), res.end());
    return res;
}

// Count Lattice Points on Segment
int segmentLatticePointsCount(int x1, int y1, int x2, int y2) {
    return abs(__gcd(x1 - x2, y1 - y2)) + 1;
}

// Create Lines (e.g., for cross shapes)
vector<pair<Point, Point>> createLines(int x, int y, int d){
    return {
        {Point(x + d, y), Point(x, y + d)},
        {Point(x - d, y), Point(x, y + d)},
        {Point(x - d, y), Point(x, y - d)},
        {Point(x + d, y), Point(x, y - d)}
    };
}
istream& operator>>(istream& is, Point& p) {
    ld x,y; is >> x >> y; p = Point(x,y);
    return is;
}

ostream& operator<<(ostream& os, const Point& p) {
    return os << p.X << ' ' << p.Y;
}
// Example Usage
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n = 4;
    vector<Point> v(n);
    for(auto &i : v) cin >> i;
    Point q;
    intersectLines(v[0], v[1], v[2], v[3], q);
```

minimum_enclosing_circle.cpp

```
const double EPS = 1e-9;

#define EQ(a, b) (fabs((a) - (b)) <= EPS)
#define LE(a, b) ((a) <= (b) + EPS)

typedef std::pair<double, double> point;
#define x first
#define y second

double sqnorm(const point &a) { return a.x*a.x + a.y*a.y; }
double norm(const point &a) { return sqrt(sqnorm(a)); }

struct circle {
  double h, k, r;

  circle() : h(0), k(0), r(0) {}
  circle(double h, double k, double r) : h(h), k(k), r(fabs(r)) {}

  // Circle with the line segment ab as a diameter.
  circle(const point &a, const point &b) {
    h = (a.x + b.x)/2.0;
    k = (a.y + b.y)/2.0;
    r = norm(point(a.x - h, a.y - k));
  }

  // Circumcircle of three points.
  circle(const point &a, const point &b, const point &c) {
    double an = sqnorm(point(b.x - c.x, b.y - c.y));
    double bn = sqnorm(point(a.x - c.x, a.y - c.y));
    double cn = sqnorm(point(a.x - b.x, a.y - b.y));
    double wa = an*(bn + cn - an);
    double wb = bn*(an + cn - bn);
    double wc = cn*(an + bn - cn);
    double w = wa + wb + wc;
    if (EQ(w, 0)) {
      throw std::runtime_error("No circumcircle from collinear points.");
    }
    h = (wa*a.x + wb*b.x + wc*c.x)/w;
    k = (wa*a.y + wb*b.y + wc*c.y)/w;
    r = norm(point(a.x - h, a.y - k));
  }

  bool contains(const point &p) const {
    return LE(sqnorm(point(p.x - h, p.y - k)), r*r);
  }
};

template<class It>
circle minimum_enclosing_circle(It lo, It hi) {
  if (lo == hi) {
    return circle(0, 0, 0);
  }
  if (lo + 1 == hi) {
    return circle(lo->x, lo->y, 0);
  }
  std::random_shuffle(lo, hi);
  circle res(*lo, *(lo + 1));
  for (It i = lo + 2; i != hi; ++i) {
    if (res.contains(*i)) {
      continue;
    }
    res = circle(*lo, *i);
    for (It j = lo + 1; j != i; ++j) {
      if (res.contains(*j)) {
        continue;
```

```
   }
   res = circle(*i, *j);
   for (It k = lo; k != j; ++k) {
    if (!res.contains(*k)) {
     res = circle(*i, *j, *k);
    }
   }
  }
 }
 return res;
}
```

polygon.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
typedef complex<ld> Point;
#define X real()
#define Y imag()
const ld EPS = 1e-9;

// Cross product
ld cross(const Point& a, const Point& b) {
  return imag(conj(a) * b);
}
ld cross(const Point& O, const Point& A, const Point& B) {
  return imag((A - O) * conj(B - O));
}

// Polygon area (absolute)
//cross function is the needed function
ld polygonArea(const vector<Point>& p) {
  ld res = 0;
  for(int i = 0; i < p.size(); i++) res += cross(p[i], p[(i+1)%p.size()]);
  return abs(res) / 2.0;
}

// Polygon centroid
//cross function is needed here
Point polygonCentroid(const vector<Point>& p) {
  Point c(0,0); ld A = 0;
  for(int i = 0; i < p.size(); i++) {
        int j = (i+1) % p.size();
        ld cross_prod = cross(p[i], p[j]);
        c += (p[i] + p[j]) * cross_prod;
        A += cross_prod;
    }
    return c / (3.0 * A);
}

// Line intersection, returns true if intersecting
//cross function is needed here
bool intersect(const Point& a, const Point& b, const Point& p, const Point& q, Point& r) {
    ld d1 = cross(p - a, b - a);
    ld d2 = cross(q - a, b - a);
    if(abs(d1 - d2) < EPS) return false;
    r = (q * d1 - p * d2) / (d1 - d2);
    return true;
}

// Sutherland–Hodgman polygon clipping
//cross function is needed here as well as the intersect function
void sortCounterClockwise(vector<Point>& p) {
    Point c = polygonCentroid(p);
    sort(p.begin(), p.end(), [&](const Point& a, const Point& b) -> bool {
        ld angle_a = arg(a - c);
        ld angle_b = arg(b - c);
        return angle_a < angle_b;
    });
}
void polygonCut(const vector<Point>& subject, const Point& a, const Point& b, vector<Point>& res) {
    res.clear();
    for(int i = 0; i < subject.size(); i++) {
        int j = (i+1) % subject.size();
        ld cross1 = cross(b - a, subject[i] - a);
        ld cross2 = cross(b - a, subject[j] - a);
        bool in1 = cross1 > EPS, in2 = cross2 > EPS;
        if(in1) res.push_back(subject[i]);
        if(in1 != in2) { Point r; if(intersect(a, b, subject[i], subject[j], r)) res.push_back(r); }
    }
}
//for identifying the number of lattice points in a polygon
int picksTheorem(int a, int b) {
    // a: area of polygon, b: no. lattice Points in the boundaries
    return a - b / 2 + 1;
```

```
}
int picksTheorem(vector<Point>& p, bool b = 0) {
  // Point sorted in counter clock-wise;
  ld area = 0;
  int bound = 0;
  int sz = (int) p.size();
  for(int i = 0; i < sz; i++) {
    int j = (i + 1) % sz;
    area += cross(p[i], p[j]);
    Point v = p[j] - p[i];
    bound += abs(__gcd((int)round(real(v)),(int)round(imag(v))));
  }
  area /= 2;
  area = fabs(area);
  return round(area - bound / 2 + 1) + b * bound;
}

// Convex polygon intersection
//polygoncut(cross, intersect) function is needed
void convexIntersect(const vector<Point>& p, const vector<Point>& q, vector<Point>& res) {
    res = q;
    for(int i = 0; i < p.size(); i++) {
        int j = (i+1) % p.size();
        vector<Point> temp;
        polygonCut(res, p[i], p[j], temp);
        res = temp; if(res.empty()) break;
    }
}
// Cross product of vectors OA and OB (returns z-component)


vector<Point> convex_hull(vector<Point> pts, bool include_collinear = false) {
    int n = pts.size(), k = 0;
    if(n <= 1) return pts;
    sort(pts.begin(), pts.end(), [&](const Point& a, const Point& b) -> bool {
        return (real(a) < real(b)) || (abs(real(a) - real(b)) < EPS && imag(a) < imag(b));
    });

    vector<Point> hull(2 * n);
    for(int i = 0; i < n; ++i) {
        while(k >= 2 && (include_collinear ? cross(hull[k-2], hull[k-1], pts[i]) < 0
                         : cross(hull[k-2], hull[k-1], pts[i]) <= 0))
        k--;
      hull[k++] = pts[i];
    }
    for(int i = n-2, t = k+1; i >=0; --i){
      while(k >= t && (include_collinear ? cross(hull[k-2], hull[k-1], pts[i]) < 0
                                  : cross(hull[k-2], hull[k-1], pts[i]) <= 0))
        k--;
      hull[k++] = pts[i];
    }
    hull.resize(k-1);
    return hull;
}
istream& operator>>(istream& is, Point& p) {
    ld x,y; is >> x >> y; p = Point(x,y);
    return is;
}

ostream& operator<<(ostream& os, const Point& p) {
    return os << p.X << ' ' << p.Y;
}
// Example usage
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n; cin >> n;
    vector<Point> poly1(n); for(auto &p : poly1) cin >> p;
    sortCounterClockwise(poly1);
    poly1 = convex_hull(poly1,1);
    cout << poly1.size() << '\n';
```

sweep_line.cpp

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

int bit[2000005];

void update(int i, int x) {
    for (; i < 2000005; i += i & (-i)) bit[i] += x;
}
```

```cpp
int query(int i) {
  int sum = 0;
  for (; i > 0; i -= i & (-i)) sum += bit[i];
  return sum;
}

int n;
vector<array<int, 4>> v;

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

  cin >> n;
  for (int i = 0, x1, y1, x2, y2; i < n; ++i) {
    cin >> x1 >> y1 >> x2 >> y2;
    if (y1 == y2) v.push_back({y1, 2, x1, x2});
    else {
      v.push_back({y1, 1, x1, 1});
      v.push_back({y2, 3, x1, 1});
    }
  }
  sort(begin(v), end(v));

  long long ans = 0;
  for (auto x : v) {
    x[2] += 1000001, x[3] += 1000001;
    if (x[1] == 1) update(x[2], 1);
    else if (x[1] == 2) ans += query(x[3]) - query(x[2] - 1);
    else update(x[2], -1);
  }
  cout << ans << '\n';
}
```

Number theory

CRT-Offline.cpp

```cpp
/// Chinese Reminder Theorem
/// Returns the smallest number x such that,
/// x % num[i] = rem[i] for each i
/// Numbers in num[] are pairwise co prime

```

```cpp
LL num[Size];
LL rem[Size];

LL CRT(int N){ /// N is size of num/rem
    LL prod = 1;
    for (int i = 0;i<N;i++){
        prod *= num[i];
    }
    LL result = 0;
    for (int i = 0;i<N;i++){
        LL pp = prod / num[i];
        result += rem[i] * modInv(pp, num[i]) * pp;
    }
    return (result % prod);
}
```

CRT-Online.cpp

```cpp
/// A is the list of reminders
/// M is the list of mod values
/// Doesn't work when the mod values aren't pairwise co-prime
vector<LL> A, M;

LL CRT(vector<LL> &A, vector<LL> &M){
    myint a1 = A[0], m1 = M[0];
    FOR(i,1,SZ(A)-1){
        LL a2 = A[i], m2 = M[i];
        LL p, q;
        ext_gcd(m1, m2, &p, &q);
        LL mod = m1*m2;
        LL x = ((((a1*m2)%mod)*q)%mod + (((a2*m1)%mod)*p)%mod)%mod;
        a1 = x;
        m1 = mod;
        if(a1<0) a1 += mod;
    }
    if(a1<0) a1 += m1;
    return a1;
}
```

CRT-OnlineNonCoPrimeModuli.cpp

```cpp
// #define __int128 LL /// Change here if __int128 is not supported

/**
    A CRT solver which works even when moduli are not pairwise coprime
    1. Call clear()
    2. Add equations using addEquation() method
    3. Call solve() to get {x, N} pair, where x is the unique solution modulo N.
    Assumptions: LCM of all mods will fit into long long.
*/

class ChineseRemainderTheorem {
    typedef long long vlong;
    typedef pair<vlong,vlong> pll;

    /** CRT Equations stored as pairs of vector. See addEqation()*/
    vector<pll> equations;

public:
    void clear() {
        equations.clear();
    }

    /** Add equation of the form x = r (mod m)*/
    void addEquation( vlong r, vlong m ) {
        equations.push_back({r, m});
    }
    pll solve() {
        if (equations.size() == 0) return {-1,-1}; /// No equations to solve

        vlong a1 = equations[0].first;
        vlong m1 = equations[0].second;
        a1 %= m1;
        /** Initially x = a_0 (mod m_0)*/

        /** Merge the solution with remaining equations */
        for ( int i = 1; i < equations.size(); i++ ) {
            vlong a2 = equations[i].first;
            vlong m2 = equations[i].second;

            vlong g = __gcd(m1, m2);
            if ( a1 % g != a2 % g ) return {-1,-1}; /// Conflict in equations

            /** Merge the two equations*/
            vlong p, q;
            ext_gcd(m1/g, m2/g, &p, &q);

            vlong mod = m1 / g * m2;
            vlong x = ( (__int128)a1 * (m2/g) % mod *q % mod + (__int128)a2 * (m1/g) % mod * p % mod ) % mod;

            /** Merged equation*/
            a1 = x;
            if ( a1 < 0 ) a1 += mod;
            m1 = mod;
        }
        return {a1, m1};
    }
}
```

Euler Phi.cpp

```
#define Max 1000000
int phi[Max];

void euler_phi(){
    phi[1] = 1;
    for(int i = 2;i<Max;i++){
        if(!phi[i]){
            phi[i] = i-1;
            for(int j = (i<<1);j<Max;j+=i){
                if(!phi[j]){
                    phi[j] = j;
                }
                phi[j] = phi[j]/i*(i-1);
            }
        }
    }
}
```

Linear_Diophantine_Equation.cpp

```
/*
  ## Linear Diophantine Equation
  A Linear Diophantine Equation (in two variables) is an equation of the general form:
   a.x + b.y = c
  where a,b are given intergs, and x,y are unknown integers.

  In this code, we consider several classical problems on these equations:

  1. finding one solution
  2. finding all solutions
  3. finding the number of solutions and the solutions themselves in a given interval
  4.finding a solution with minimum value of x + y$
*/

// ## Finding one solution
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;
}

// ## Find number of solution when x: [minx, maxx] , y: [miny,maxy]
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;
}
/*
```

```
  ## finding the number of solutions and the solutions themselves in a given interval
  Once we have lx and rx.
  Just need to iterate through
  x = lx + k * (b / g) for all k >= 0 until x = rx
  and find the corresponding y values using the equation a.x + b.y = c
*/

/*
  ## Find the solution with minimum value of x + y
  Here x and yalso need to be given some restriction, otherwise, the answer may become negative infinity.
  1. Find any solution (x , y) for the equations.

  minimum value = x + y + k * (b - a) / g

  if(a < b) select smallest possible value of k
  if(a > b) select the largest possible value of k
  if(a == b) all solution will have the same sum x + y.
```

binary_gcd.cpp

```
int gcd(int a, int b) {
  if (!a || !b)
    return a | b;
  unsigned shift = __builtin_ctz(a | b);
  a >>= __builtin_ctz(a);
  do {
    b >>= __builtin_ctz(b);
    if (a > b)
      swap(a, b);
    b -= a;
  } while (b);
  return a << shift;
}
```

discrete_log.cpp

```
int powmod(int a, int b, int m) {
  int res = 1;
  while (b > 0) {
```

```cpp
        if (b & 1) {
            res = (res * 1ll * a) % m;
        }
        a = (a * 1ll * a) % m;
        b >>= 1;
    }
    return res;
}

int solve(int a, int b, int m) {
    a %= m, b %= m;
    int n = sqrt(m) + 1;
    map<int, int> vals;
    for (int p = 1; p <= n; ++p)
        vals[powmod(a, p * n, m)] = p;
    for (int q = 0; q <= n; ++q) {
        int cur = (powmod(a, q, m) * 1ll * b) % m;
        if (vals.count(cur)) {
            int ans = vals[cur] * n - q;
            return ans;
        }
    }
    return -1;
}
```

extended_euclidean.cpp

```cpp
// a * x + b * y = gcd(a , b)
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;
}
```

gcd.cpp

```cpp
int gcd (int a, int b) {
  while (b) {
    a %= b;
    swap(a, b);
  }
  return a;
}
```

gcd_negative_integer.cpp

```cpp
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) {
    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;
}
```

mobius.cpp

```cpp
/*
 mobius Function (m)
 if i has a squared factor : m(i) = 0
 else m(i) = (-1)^r , r : number of distinct prime the i has
*/
const int N = 1e5;
int mobius[N] , sieve[N];
void gen_mobius(){
 for(int i = 1; i < N; i++) {mobius[i] = sieve[i] = 1;}
 sieve[1] = 0;
 for(long long i = 2; i < N; i++){
  if(sieve[i]){
   for(long long j = i; j < N; j += i){
    sieve[j] = 0;
    mobius[j] = (j % (i * i) == 0) ? 0 : -mobius[j];
   }
  }
 }
}
```

modularMultiplicativeInverse.cpp

```
/*
  ## Extended Euclidean algorithms
  Modular multiplicative inverse when M and A are coprime or gcd(A, M) = 1:

  Time Complexity: O(log M)
  Auxiliary Space: O(1)
*/
ll gcd(ll a, ll b, ll& x, ll& y) {
  x = 1, y = 0;
  ll x1 = 0, y1 = 1, a1 = a, b1 = b;
  while (b1) {
    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;
}
ll inv(ll A , ll M){
  // modular inverse of A mod M
  ll x , y;
  ll g = gcd(A,M,x ,y);
  if(g != 1){ // Inverse doesn't exist
    exit(3);
  }
  ll res = (x % M + M) % M;
  return res;
}

//------------------------------------------------------------------------------
/*
 ## Fermat’s little theorem
 Modular multiplicative inverse when M is prime:
 Time Complexity: O(log M)
 Auxiliary Space: O(1)
```

```
*/
ll bpw(ll a , ll b, ll mod){
  ll res = 1;
  a %= mod; // avoid overflow from a * a
  while(b){
    if(b % 2) res = (res * a) % mod;
    b /= 2;
    a = (a * a) % mod;
  }
}
ll inv(ll N, ll M){
  return bpw(N , M - 2);
}
```

optimized_sieve_LPF.cpp

```
vector<int> prime, lpf;
void Sieve(int n) {
    prime.clear();
    lpf.assign(n + 1, 0);

    lpf[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (lpf[i] == 0) {
            lpf[i] = i;
            prime.push_back(i);
        }
        for (int j = 0; j < (int) prime.size() && i * prime[j] <= n; j++) {
            lpf[i * prime[j]] = prime[j];
            if (prime[j] == lpf[i]) break;
        }
    }
}
```

Notes

Searching

counting_sort.cpp

```
/*
  --Counting Sorting--
  use:- It's an algrothim to sort the array using bookkeeping array aka(frequency array)
  Time Complexity:- O(max(a_1 , a_2 ,.. , n));
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
const int N = 1e5 + 9;
int fre[N];
int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
  int n; cin >> n;
  int a[n] , b[n];
  for(int i = 0; i < n; ++i) cin >> a[i] , fre[a[i]]++;
  // counting-sorting
  int idx = 0;
  for(int i = 0; i < N; ++i) for(int j = 0; j < fre[i]; j++)
    b[idx++] = i;
  // b >> contains array a after sorting
}
```

double_ternary_search.cpp

```
long double l = 0 , r = 1e9 , mid;
int cnt = 400;
while(cnt--){
 double g = l + (r - l) / 3,
    h = r - (r - l) / 3;
 if(f(g) < f(h)) r = h; // get minumum value
 else l = g;
}
```

merge_sort.cpp

```
/*
  --Merge sort--
  Use: sorting the array
```

```
    Time Complexity: O(n log n)
    Space Complexity: O(n)
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
const int N = 1e5 + 1;
int a[N] , v1[N] , v2[N];
void mergeSort(int l , int r){
    int mid = (l + r) / 2;
    int n = mid - l , m = r - mid;
    for(int i = l; i < mid; ++i) v1[i - l] = a[i];
    for(int i = mid; i < r; ++i) v2[i - mid] = a[i];
    int i = l , idx1 = 0 , idx2 = 0;
    while(idx1 < n && idx2 < m){
        if(v1[idx1] <= v2[idx2]) a[i++] = v1[idx1++];
        else a[i++] = v2[idx2++];
    }
    while(idx1 < n) a[i++] = v1[idx1++];
    while(idx2 < m) a[i++] = v2[idx2++];
}
void merge(int l , int r){
    if(r - l == 1) return; // interval of length 1
    int mid = (l + r) / 2; // [l , mid[ , [mid , r[
    merge(l , mid);
    merge(mid , r);
    mergeSort(l , r);
}
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    int n; cin >> n;
    for(int i = 0; i < n; ++i) cin >> a[i];
    merge(0 , n);
    for(int i = 0; i < n; ++i) cout << a[i] << " "; // array after sorted
}
```

minNumberOfSwapsToSortTwoBinaryString.cpp

```cpp
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including tree_order_statistics_node_update
using namespace __gnu_pbds;
template<class T> using ordered_set = tree<T, null_type , less_equal<T> , rb_tree_tag , tree_order_statistics_node_update> ;
int solve(string &s1 , string &s2){
  if(s1.size() != s2.size() || count(s1.begin() , s1.end() , '1') != count(s2.begin() , s2.end() , '1')){
    return INT32_MAX; // Two strings can't be equal
  }
  ordered_set<int> pos[2];
  for(int i = 0; i < s1.size(); ++i){
    pos[s1[i] == '1'].insert(i);
  }
  ll ans = 0;
  for(auto &ch : s2){
    int f = (ch == '1');
    assert(pos[f].size());
    int i = *pos[f].find_by_order(0);
    int ope = pos[f ^ 1].order_of_key(i);
    ans += ope;
    pos[f].erase( pos[f].lower_bound(i - 1) );
  }
  return ans;
}
```

patient_sort.cpp

```cpp
void LIS(vector<int> &v , vector<int> &ans){
 vector<int> lis(v.size());
 int cnt = 0;
 for (int i = 0; i < v.size(); ++i) {
  int pos = lower_bound(lis.begin() , lis.begin() + cnt , v[i]) - lis.begin();
  lis[pos] = v[i];
  if(pos == cnt) cnt++;
  ans[i] = pos + 1;
 }
}
```

radix_sort.cpp

```
vector<int> radix_sort(vector<int> v){
  int n = v.size();
  const int MAX = 16;
  ll p10 = 1;
  for(int i = 0; i < MAX; ++i){
    vector<int> f(10) , tmp(n);
    for(auto &x : v) f[x / p10 % 10]++;
    for(int i = 1; i < 10; ++i) f[i] += f[i - 1];
    for(int i = n - 1; i >= 0; --i){
      tmp[ --f[v[i] / p10 % 10]  ] = v[i];
    }

    swap(v , tmp);
    p10 *= 10;

  }
  return v;
}
```

Snippet

clock.cpp

```
// Give the running time of code in ms.
ll get_time(){ return 1000 * clock() / CLOCKS_PER_SEC; }
```

fastCode.cpp

```
ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
```

gcc_optimize.cpp

```
// reduce the runnig time
#pragma GCC optimize("Ofast")
#pragma GCC target("avx2,bmi,bmi2,popcnt,lzcnt")
#pragma GCC optimize("Ofast,unroll-loops")
```

```
#pragma GCC target("avx2")
// runtime errors with overflow
#pragma GCC optimize("trapv")



#pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,fast-math")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
#pragma GCC target("avx,avx2,fma")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,fma")
```

Standard Problems

Geometry

check_point_in_convex.cpp

```
/*
  Standard Problem: https://codeforces.com/gym/104968/problem/H
  Given a convex polygon
  Given a n point check if this point inside a polygon or not in O(log n)
  by pre-proccessing O(n)
*/
#include <bits/stdc++.h>
using namespace std;
#define double long double
#define int long long
#define vec(a,b) {b.x-a.x,b.y-a.y}
long long const N  = 1e5;
struct Point { int x, y; };
double const eps = 1e-7;
int orientation(Point a, Point b, Point c) {
   double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
   if (v < 0) return -1; // clockwise
   if (v > 0) return +1; // counter-clockwise
   return 0;
}


bool cw(Point a, Point b, Point c, bool include_collinear) {
   int o = orientation(a, b, c);
   return o < 0 || (include_collinear && o == 0);
```

```
}
bool collinear(Point a, Point b, Point c) { return orientation(a, b, c) == 0; }

void convex_hull(vector<Point>& a, bool include_collinear = false) {
  Point p0 = *min_element(a.begin(), a.end(), [](Point a, Point b) {
    return make_pair(a.y, a.x) < make_pair(b.y, b.x);
  });
  sort(a.begin(), a.end(), [&p0](const Point& a, const Point& b) {
    int o = orientation(p0, a, b);
    if (o == 0)
      return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
        < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
    return o < 0;
  });
  if (include_collinear) {
    int i = (int)a.size()-1;
    while (i >= 0 && collinear(p0, a[i], a.back())) i--;
    reverse(a.begin()+i+1, a.end());
  }

  vector<Point> st;
  for (int i = 0; i < (int)a.size(); i++) {
    while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i], include_collinear))
      st.pop_back();
    st.push_back(a[i]);
  }

  a = st;
}
double cross(Point a, Point b){ return a.x*b.y - b.x*a.y; }
int cross(Point a, Point b, Point c){
  Point d = vec(b,a), e = vec(c,a);
  return cross(d,e);
}
double dot(Point a, Point b){
  return a.x*b.x + a.y*b.y;
}
bool pointOnLine(Point a, Point b, Point p){
  return fabs(cross(vec(a,b),vec(a,p))) < eps;
}
bool pointOnRay(Point a, Point b, Point p){
  return dot(vec(a,p),vec(a,b)) > -eps;
}
bool pointOnsegment(Point a, Point b, Point p){
    if(!pointOnLine(a,b,p)) return 0;
    return pointOnRay(a,b,p) && pointOnRay(b,a,p);
}
vector<Point> v(N);
void prepare(int n) {
    int pos = 0;
    for (int i = 0; i < n; i++) {
        if (make_pair(v[i].x, v[i].y) < make_pair(v[pos].x, v[pos].y))
            pos = i;
    }
    rotate(v.begin(), v.begin() + pos, v.end());
}
void print(Point p){
    cerr << p.x << " " << p.y << "\n";
}
bool fun(Point p,int n)
{
    int idx = 1;
    if(pointOnsegment(v[0],v[n-1],p)) return 1;
    if(cross(v[0],v[n-1],p) < 0) return 0;
    if(cross(v[0],v[1],p) > 0) return 0;
    int l = 1, r= n - 2;
    int ans = -1;
    while(r >= l)
    {
        int mid = l + (r-l)/2;
        if(cross(v[0],v[mid],p) <= 0) l = mid+1, ans = mid;
        else r=mid-1;
    }
    if(ans == -1) return 0;
    return (cross(v[ans],v[(ans+1)%n],p) <= 0);
}
double area(Point a, Point b, Point c)
{
    return 0.5*abs(a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y));
}
void solve()
{
    int n,m; cin >> n >> m;
    vector<Point> vans(n);
    for(int i=0; i < n; ++i)
    {
        cin >> vans[i].x >> vans[i].y;
```

```cpp
    }
    for(int i=0; i < m; ++i) cin >> v[i].x >> v[i].y;
    convex_hull(v);
    prepare(v.size());
    m = v.size();
    int ans = 0;
    for(int i=0; i < n; ++i) ans += fun(vans[i],m);
    cout << ans << "\n";
}
int32_t main()
{
    int t=1; //cin >> t;
    while(t--)
    {
        solve();
```

count_points_in_circle.cpp

```cpp
/*
Problem link: https://atcoder.jp/contests/abc191/tasks/abc191_d
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
int EPS = 10000 , Precision = 4;
long long read(){
    // 1.2345 --> 12345
    string s;
    cin >> s;
    int x=s.find('.');
    if(x==-1) return stoll(s+"0000");
    string one=s.substr(0,x);
    string two=s.substr(x+1);
    while(two.size()< Precision) two+="0";
    return stoll(one+two);
}
ll ceil(ll a, ll b){
    if(b < 0) a *= -1, b *= -1;
    if(a < 0) return a / b;
    else return (a + b - 1) / b;
}

ll floor(ll a, ll b){
    if(b < 0) a *= -1, b *= -1;
    if(a > 0) return a / b;
    else return (a - b + 1) / b;
}
ll BS(ll x , ll z){
    // find maximum y : y ^ 2 <= z ^ 2 - x ^ 2
    ll l = 0 , r = z , mid;
    while(l < r){
        mid = l + (r - l + 1) / 2;
        if(mid * mid <= z * z - x * x) l = mid;
        else r = mid - 1;
    }
    return l;
}
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    ll x = read(), y = read() , r = read();
    ll l_x = floor(x - r , EPS) * EPS;
    ll r_x = ceil(x + r , EPS) * EPS;
    ll cnt = 0;
    for(ll i = l_x; i <= r_x; i += EPS){
        if(abs(i - x) > r) continue;
        ll len = BS(abs(x - i) , r);
        ll l_y = ceil(y - len , EPS);
        ll r_y = floor(y + len , EPS);\
        if(r_y >= l_y) cnt += r_y - l_y + 1;
    }
    cout << cnt;
}
```

point_in-shape.cpp

```cpp
/*
    Problem Link: https://codeforces.com/gym/104447/problem/H
    You are given a polygon of n, vertices and q queries.

    Each query consists of a point (x,y)
    and you have to check if it is inside (including the borders) or outside the polygon.

    The points of the polygon are given in clockwise order with the property that either xi=xi−1
    or yi=yi−1
    (but not both), indicating that the edges are parallel to either the x- or y-axis, Also no two edges intersect
    (endpoints are not considered into the intersections).

    The first line of the input contains an integer n 4≤n≤1e5),representing the number of vertices in the polygon.

    The next n lines each contain two integers xi and yi (0≤xi,yi≤106), representing the x- and y-coordinates, respectively, of the i
    -th vertex of the polygon in clockwise order.
    The next q lines each contain two integers x and y (0≤x,y≤1e6), representing the x- and y-coordinates,
    respectively, of a point to be checked whether it is inside or outside the polygon.
*/
#include<iostream>
#include <bits/stdc++.h>

#define ll long long
#define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
const int N = 1e6 + 5, mod = 1e9 + 7, M = 17, inf = 2e9, sq = 632;
vector<vector<int> > x[N], y[N], queries[N];

bool inside(vector<vector<int> > &v, int val) {
    int low = 0, high = v.size() - 1;
    while (low <= high) {
        int mid = low + high >> 1;
        if (v[mid][1] < val) {
            low = mid + 1;
        } else if (v[mid][0] > val) {
            high = mid - 1;
        } else {
            return true;
        }
    }
    return false;
}

int bit[N];

void add(int idx, int val) {
    for (; idx < N; idx += idx & -idx) {
        bit[idx] += val;
    }
}

void add_range(int l, int r) {
    if (l > r)return;
    add(l, 1);
    add(r + 1, -1);
}

int query(int idx) {
    int ans = 0;
    for (; idx; idx -= idx & -idx) {
        ans += bit[idx];
    }
    return ans;
}

map<int, int> freq[2];

void doWork() {
    int n;
    cin >> n;
    vector<pair<int, int> > v(n);
    for (int i = 0; i < n; i++) {
        cin >> v[i].first >> v[i].second;
        v[i].second++;
    }
    for (int i = 0; i < n; i++) {
        if (v[i].first == v[(i + 1) % n].first) {
            int y1 = v[i].second, y2 = v[(i + 1) % n].second;
            x[v[i].first].push_back({min(y1, y2), max(y1, y2), y1 < y2});
        } else {
            int x1 = v[i].first, x2 = v[(i + 1) % n].first;
            y[v[i].second].push_back({min(x1, x2), max(x1, x2)});
        }
    }
    for (int i = 0; i < N; i++) {
        sort(x[i].begin(), x[i].end());
        sort(y[i].begin(), y[i].end());
    }
    int q;
    cin >> q;
    vector<int> ans(q, 0);
    for (int i = 0; i < q; i++) {
        int a, b;
        cin >> a >> b;
        b++;
        if (inside(x[a], b) || inside(y[b], a)) {
            ans[i] = true;
        } else {
            queries[a].push_back({b, i});
        }
    }
    for (int i = 0; i < N; i++) {
        for (auto j: x[i]) {
            add_range(j[0] + 1, j[1] - 1);
            freq[j[2]][j[0]]++;
            freq[j[2]][j[1]]++;
        }
        for (auto j: queries[i]) {
            int cnt = query(j[0]);
            int cntUP = freq[1][j[0]];
            int cntDown = freq[0][j[0]];
            cnt += (max(cntDown, cntUP) - min(cntDown, cntUP)) / 2;
            ans[j[1]] = cnt % 2;
        }
    }
    for (auto i: ans) {
        cout << (i ? "YES\n" : "NO\n");
    }
}


int main() {
    IO
    int t = 1;
  //  cin >> t;
```

```
    for (int i = 1; i <= t; i++) {
        //    cout << "Case #" << i << ": ";
```

Implementation

coloring_matrix_with_largest_area.cpp

```
/*
        Problem Link:- https://codeforces.com/gym/104874/problem/K
    Given a matrix n x m, each cell (i, j) can be
    1. g(i,j) = '.' -> empty cell
    2. g(i , j) = 'A' king , "There is exactly one letter 'A'"
    3. g(i , j) = child represented as charachter between 'z' -> 'z' , it's proved that letters are distinct.
    Output the same matrix, replacing each character '.' with the lowercase letter,
    corresponding to the owner of the  containing this cell
    "where e area of the king that belongs to his favorite child is as large as possible."
    Test Cases:
    Input:
        6 8
        ......X.
        .F......
        ...A....
        ........
        .....P..
        ..L.....
    Output:
        xxxxxxXx
        fFaaaaaa
        ffaAaaaa
        ffaaaaaa
        lllllPpp
        llLllppp
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
const int N = 1001;
```

```
int n , m , curI , curJ;
pair<int, int> L, R; // Diamtions of Maxiumum area
bool islower(char ch){ return ch >= 'a' && ch <= 'z';   }
bool validX(int x) { return x >= 0 && x < n; }
bool validY(int y) { return y >= 0 && y < m; }
void solveRange(int x1, int y1, int x2, int y2, vector<vector<char>> &g) {
  if (!validX(x1) || !validX(x2) || !validY(y1) || !validY(y2))
    return;
  for (int i = x1; i <= x2; i++) {
    int last = y1 - 1;
    char lastC = ' ';
    for (int j = y1; j <= y2; j++) {
      if (g[i][j] != '.') {
        char cur = g[i][j] - 'A' + 'a';
        for (int temp = j - 1; temp > last; temp--) {
          g[i][temp] = cur;
        }
        last = j;
        lastC = cur;
      }
    }
    if (lastC != ' ') {
      for (int j = y2; j > last; j--) {
        g[i][j] = lastC;
      }
    } else {
      if (i - 1 >= x1 && g[i - 1][y1] != '.') {
        for (int j = y1; j <= y2; j++) {
          g[i][j] = tolower(g[i - 1][j]);
        }
      }
    }
  }
  for (int i = x2; i >= x1; i--) {
    if (g[i][y1] == '.') {
      assert(i != x2);
      for (int j = y1; j <= y2; j++) {
        g[i][j] = tolower(g[i + 1][j]);
      }
    }
  }
}
void paint(int x1, int y1, int x2, int y2, char c , vector<vector<char>> &g) {
  for (int i = x1; i <= x2; i++) {
        for (int j = y1; j <= y2; j++) {
            g[i][j] = c;
        }
    }
}
void zero_matrix(vector<vector<char>> a) {
    n = a.size();
    m = a[0].size();
    vector<int> d(m, -1), d1(m), d2(m);
    stack<int> st;
    int mx = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (a[i][j] != '.') {
                d[j] = i;
            }
        }
        for (int j = 0; j < m; j++) {
            while (!st.empty() && d[st.top()] <= d[j]) {
                st.pop();
            }
            d1[j] = st.empty() ? -1 : st.top();
            st.push(j);
        }
        while (!st.empty()) {
            st.pop();
        }
        for (int j = m - 1; j >= 0; j--) {
            while (!st.empty() && d[st.top()] <= d[j]) {
                st.pop();
            }
            d2[j] = st.empty() ? m : st.top();
            st.push(j);
        }
        while (!st.empty()) {
            st.pop();
        }
        for (int j = 0; j < m; j++) {
            pair<int, int> tempL = {d[j] + 1, d1[j] + 1};
            pair<int, int> tempR = {i, d2[j] - 1};
            int area = (tempR.first - tempL.first + 1) * (tempR.second - tempL.second + 1);
            if (curI >= tempL.first && curI <= tempR.first && curJ >= tempL.second && curJ <=
tempR.second &&
        area >= mx) {
        L = tempL;
        R = tempR;
        mx = area;
      }
    }
  }
}

int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
  #ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
  #endif
  cin >> n >> m;
  vector<vector<char>> g(n , vector<char>(m));
  int st_x, st_y;
  rep(i , 0 , n) rep(j ,0,m){
   cin >> g[i][j];
   if(g[i][j] == 'A') curI = i, curJ = j , g[i][j] = '.';
  }

  g[curI][curJ] = '.';
  zero_matrix(g);
  paint(L.first, L.second, R.first, R.second, char('a'), g);
  g[curI][curJ] = char('A');


  int x1 = L.first, y1 = L.second, x2 = R.first, y2 = R.second;
  solveRange(0, 0, x1 - 1, m - 1,g);
  solveRange(x1, 0, x2, y1 - 1,g);
  solveRange(x2 + 1, 0, n - 1, m - 1,g);
  solveRange(x1, y2 + 1, x2, m - 1,g);

  rep(i , 0 , n){
    rep(j , 0 , m){
      cout << g[i][j];
    }
    cout << '\n';
  }
```

divide_grid_into_k_connctedcompented_01.cpp

```cpp
/*
Problem Statement:
Given a rectangular grid of size n × m, create a map of '0' and '1' terrain types where:
1. Only two terrain types are allowed ('0' and '1')
2. There must be exactly k connected components
  - Connected components are adjacent cells of same value
  - Cells are adjacent if they share a side

Input: Three integers n, m, k where:
- 1 ≤ n ≤ 1000 (rows)
- 1 ≤ m ≤ 1000 (columns)
- 1 ≤ k ≤ n × m (desired components)

Output:
- "YES" + grid solution if possible
- "NO" if impossible
*/
void solve() {
    int n, m, k;
    cin >> n >> m >> k;
    int a[n][m];

    // Case 1: One-dimensional grid (single row or column)
    if (n == 1 || m == 1) {
        cout << "YES\n";
        int num = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                cout << num;
                if (k > 1) {
                    num = 1 - num;
                    k--;
                }
            }
            cout << '\n';
        }
        return;
    }

    // Case 2: Impossible case - when components would equal n*m-1
    if (k == n * m - 1) {
        cout << "NO\n";
        return;
    }

    // Case 3: All other cases
    cout << "YES\n";
    int num = 0;
    int flag = 0;

    // Adjust k if (k+1) is divisible by m
    if ((k + 1) % m == 0) {
        k--;
        flag = 1;
    }

    // Fill the grid with alternating values until k components are created
    int i, j;
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            a[i][j] = num;
            num = 1 - num;
            k--;
            if (k <= 0) break;
        }
        if (k <= 0) break;
        if (m % 2 == 0) num = 1 - num;
    }

    // Fill remaining cells in the current row
    if (i == 0) {
        for (int x = j + 1; x < m; x++) {
            a[i][x] = 1 - num;
        }
    } else if (j + 1 < m) {
        a[i][j + 1] = num;
    }

    // Fill remaining rows by copying values from above
    for (int x = 0; x < m; x++) {
        int st;
        if (x <= j + 1 || i == 0) {
            st = i;
        } else {
            st = i - 1;
        }
        for (int y = st + 1; y < n; y++) {
            a[y][x] = a[st][x];
        }
    }

    // Handle special case when (k+1) was divisible by m
    if (flag) {
        a[n - 1][m - 1] = 1 - a[n - 1][m - 2];
    }

    // Print the final grid
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout << a[i][j];
        }
        cout << '\n';
    }
```

Number_Theory

count_no_subsequec_with_lcm.cpp

```
/*
 Problem Link: https://atcoder.jp/contests/abc349/tasks/abc349_f
 Problem description:
  You are given a sequence of positive integers
  A=(A1, A2,.. , An) of length  N and a positive integer  M. Find the number, modulo 998244353, of non-empty and not necessarily contiguous subsequences of
  A such that the least common multiple (LCM) of the elements in the subsequence is
  M. Two subsequences are distinguished if they are taken from different positions in the sequence, even if they coincide as sequences. Also, the LCM of a sequence with a single element is that element itself.

  Constraints
  1 <= N <= 2e5
  1 <= M <= 1e16
  1 <= Ai <= 1e16

  My sol:
   Time complexity: O( (2^x) ^ 3) --> can be decreased to O(3^x) using submasking method
```

```
    such that x: no. of primes in M
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
const int mod = 998244353;
ll bpw(ll a , ll b){
  ll res = 1;
  while(b){
    if(b % 2) res = res * a % mod;
    b /= 2;
    a = a * a % mod;
  }
  return res;
}
int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
  #ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
  #endif
  ll n , m; cin >> n >> m;
  vector<ll> dis;
  ll M = m;
  for(ll i = 2; i * i <= m; ++i) if(m % i == 0){
    ll d = 1;
    while(m % i == 0){ m /= i; d *= i; }
    dis.emplace_back(d);
  }
  if(m > 1) dis.emplace_back(m);
  m = (int) dis.size();
  int cnt[1 << m]{} , one = 0;
  for(int i = 0; i < n; ++i){
    ll x; cin >> x;
    if(__gcd(x , M) != x) continue;
    ll mask = 0;
    for(int j = 0; j < m; ++j) if(x % dis[j] == 0)
      mask |= (1 << j);
    one += (x == 1);
    cnt[mask]++;
  }
  if(M == 1){ cout << ( bpw(2 , one) - 1 + mod ) % mod; return 0; }
  vector<ll> dp(1 << m);
  dp[0] = bpw(2 , cnt[0]);
  for(int i = 1; i < (1 << m); ++i){
    ll c = bpw(2 , cnt[i]) - 1;
    // Previous mask
    for(int j = (1 << m) - 1; j >= 0; --j){
      dp[i | j] += dp[j] * c;
      dp[i | j] %= mod;
    }
  }
  cout << dp[(1 << m) - 1];
```

count_subsequece_gcd_equal_1.cpp

```
// problem link: https://codeforces.com/contest/803/problem/F
#include <bits/stdc++.h>
typedef long long ll;
#define s second
#define f first
#define add(a , b) a = (a + b + mod) % mod
#define rep(i , st , ed) for(int i = st ; i < ed ; i++)
using namespace std;
void burn(){
ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
#ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
 #endif
}
//\//\//\/\//\//\//\//\//\//
const int N = 100009 , mod = 1e9 + 7;
ll mobius[N] , sieve[N] , g[N] , fre[N];
void gen_mobius(){
 for(int i = 1; i < N; i++) {mobius[i] = sieve[i] = 1;}
 sieve[1] = 0;
 for(long long i = 2; i < N; i++){
  if(sieve[i]){
   for(long long j = i; j < N; j += i){
    sieve[j] = 0;
```

```
        mobius[j] = (j % (i * i) == 0) ? 0 : -mobius[j];
      }
    }
  }
}
ll bin_exp(ll a , ll b){
  ll ret = 1;
  while(b){
    if(b % 2) ret = (ret * a) % mod;
    a = (a * a) % mod;
    b /= 2;
  }
  return ret;
}
int main(){
  burn();
  ll n; cin >> n;
  rep(i , 0 , n){
    int x; cin >> x;
    fre[x]++;
  }
  gen_mobius();
  for(int i = 2; i < N; i++){
    for(int j = i; j < N; j += i){
      g[i] += fre[j];
    }
  }
  ll ans = (bin_exp(2 , n) - 1 + mod) % mod;
  // count number of subsequence the gcd > 1
  for(int i = 2; i < N; i++){
    add(ans, mobius[i] * ((bin_exp(2 , g[i]) - 1 + mod) % mod) % mod);
  }
  cout << ans;
}
```

Tree

No_of_paths_insides_paths.cpp

```
/*
Problem link: https://codeforces.com/group/Rilx5irOux/contest/564406/problem/H
  Problem Statement:
```

```
- Given n towns connected by n-1 roads forming a tree
- m trade agreements between towns
- Each agreement affects all towns on shortest path between si and ti
- Towns u,v can trade if shortest path between them covered by an agreement
- Must count valid trading pairs where u<v

Input:
- n, m (2≤n,m≤105)
- n-1 lines: xi,yi road connections
- m lines: si,ti trade agreement paths

Output:
- Number of valid trading pairs
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define f first
#define s second
const int N = 1e5 + 9; // TODO: change it to maximum possible N
const int LOG = 17;
int dfs_time = 0, cur_pos = 0;
int st[N] , ft[N] , big[N] , ver[N] , sz[N];
int head[N], par[N], dep[N] , up[N][LOG];
int n , m , bst[N];
ll ans;
vector<int> adj[N] , paths[N];

// Lazy segment tree
struct node{
  int mn , cnt;
  node(int x){ mn = x; cnt = 1;}
  node(){};
  node operator + (const node other) const{
    node res;
    res.mn = min(mn , other.mn);
    if(mn < other.mn) res.cnt = cnt;
    else if(other.mn < mn) res.cnt = other.cnt;
    else res.cnt = cnt + other.cnt;
    return res;
  }

} tree[4*N];
int lazy[4*N];
```

```
void push_down(int x , int par){
    lazy[x] += lazy[par];
    tree[x].mn += lazy[par];
}
void propogate(int x , int l , int r){
    if(r - l == 1) return;
    push_down(2*x+1,x);
    push_down(2*x+2,x);
    lazy[x] = 0;
}
void build(int x = 0 , int l = 0 , int r = n){
    if(r - l == 1){ tree[x] = node(0); return; }
    int mid = (l+r)/2;
    build(2*x+1,l,mid);
    build(2*x+2,mid,r);
    tree[x] = tree[2*x+1] + tree[2*x+2];
}
void upd(int lx , int rx , int v, int x = 0 , int l = 0 , int r = n){
    propogate(x,l,r);
    if(l >= lx && r <= rx){
        tree[x].mn += v;
        lazy[x] = v;
        return;
    }
    if(r <= lx || l >= rx) return;
    int mid = (l+r)/2;
    upd(lx,rx,v,2*x+1,l,mid);
    upd(lx,rx,v,2*x+2,mid,r);
    tree[x] = tree[2*x+1] + tree[2*x+2];
}
node qry(int lx , int rx , int x = 0 , int l = 0 , int r = n){
    propogate(x,l,r);
    if(l >= lx && r <= rx) return tree[x];
    if(r <= lx || l >= rx) return node(2e9);
    int mid = (l+r)/2;
    return qry(lx,rx,2*x+1,l,mid) + qry(lx,rx,2*x+2,mid,r);
}
int countZero(int l , int r){
    if(l > r) return 0;
    auto ans = qry(l,r+1);
    if(ans.mn) return 0;
    // cerr << "\n" << "Min: " << ans.mn <<  " cnt: " << ans.cnt  << " sz: " << r - l + 1 << '\n';
    return ans.cnt;
```

```
}

// HLD
void preDFS(int u = 0, int p = 0){
    sz[u] = 1, big[u] = -1;

    // Build LCA
    dep[u] = dep[p] + 1;
    up[u][0] = p;
    par[u] = p;
    for(int x = 1; x < LOG; ++x){
        up[u][x] = up[ up[u][x-1] ][x-1];
    }

    for(auto v : adj[u]) if(v != p){
        preDFS(v,u);
        sz[u] += sz[v];
        if(big[u] == -1 || sz[v] > sz[ big[u] ]) big[u] = v;
    }
}

void decomposition(int u = 0, int h = 0){
    head[u] = h;
    st[u] = dfs_time++;
    // cerr << u + 1 << ' ';
    ver[ st[u] ] = u;

    if(~big[u])
        decomposition(big[u], h);
    for(auto &v : adj[u]) if(v != par[u] && v != big[u])
        decomposition(v,v);
    ft[u] = dfs_time;
}
void upd_path(int u , int v , int val){
    int ans = 0;
    for(; head[u] != head[v]; v = par[head[v]]){
        if(dep[head[u]] > dep[head[v]]) swap(u,v);
        // proccess interval: [ st[head[v]] , st[v]  ]
        upd(st[head[v]] , st[v] + 1, val);

    }
    if(dep[u] > dep[v]) swap(u,v);
    // proccess interval: [ st[u] , st[v] ]
    upd(st[u] , st[v] + 1, val);
}

// DSU on tree
void sackDFS(int u, int p, bool keep){
    int bigChild = big[u];

     // run a dfs on small childs
    for(auto v : adj[u]){
        if(v != p && v != bigChild) sackDFS(v , u, 0);
    }

    if(bigChild != -1) sackDFS(bigChild, u, 1);  // bigChild marked as big and not cleared from cnt
    for(auto v : adj[u]){
        if(v != p && v != bigChild){
            for(int p = st[v]; p < ft[v]; p++){
                // Add your information about ver[p]
                int x = ver[p];
                // cerr << "+ " << x + 1 << '\n';
                for(auto &y : paths[x]) upd_path(x,y,+1);
            }

        }

    }
    // Add your information about u
    // cerr << "+ " << u + 1 << '\n';
    for(auto &y : paths[u]) upd_path(u,y,+1);

    // All information about the subtree of  u  is kept, and you can now query it.
    // cerr << "u: " << u + 1 << " , ans: " << (n - ft[u]) - countZero(ft[u] , n-1) << '\n';
    // cerr << "After: " << n - ft[u] << ' ' << " zero: " << countZero(ft[u] , n - 1) << '\n';
    ans += (n - ft[u]) - countZero(ft[u] , n-1);
    ans += dep[u] - bst[u];

    if(keep == 0){
        for(int p = st[u]; p < ft[u]; p++){
            // Remove the added information about ver[p]
            int x = ver[p];
            // cerr << "- " << x+1 << '\n';
            for(auto &y : paths[x]) upd_path(x,y,-1);
        }
    }

}

// Compute LCA
int getLCA(int x , int y){
    if(dep[x] < dep[y]) swap(x,y);
    int k = dep[x] - dep[y];
    for(int i = 0; i < LOG; ++i) if((k>>i) & 1) x = up[x][i];
    if(x == y) return x;

    for(int i = LOG - 1; i >= 0; --i) if(up[x][i] != up[y][i]){
        x = up[x][i];
        y = up[y][i];
    }

    assert(up[x][0] == up[y][0]);
    return up[x][0];
}

void _dfs(int u , int p){
    for(auto &v : adj[u]) if(v != p){
        _dfs(v,u);
        bst[u] = min(bst[u] , bst[v]);
    }
}
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    cin >> n >> m;
    for(int i = 1; i < n; ++i){
        int u,v; cin >> u >> v;
        --u; --v;
        adj[u].emplace_back(v);
        adj[v].emplace_back(u);
    }

    preDFS(0,0);
```

```cpp
    decomposition(0,0);
    build();

    for(int i = 0; i < n; ++i) bst[i] = dep[i];

    for(int i = 0; i < m; ++i){
        int x,y; cin >> x >> y;
        --x; --y;
        if(st[x] > st[y]) swap(x,y);
        int lca = getLCA(x,y);
        bst[x] = min(bst[x] , dep[lca]);
        bst[y] = min(bst[y] , dep[lca]);
        if(lca == x) continue;
        paths[x].emplace_back(y);
```

String

Aho corasick

AhoCorasick.cpp

```cpp
struct AhoCorasick {
    // Modify these values based on input alphabet
    // alpha: size of alphabet (26 for lowercase letters)
    // first: first character of alphabet ('a' for lowercase letters)
    enum { alpha = 26, first = 'a' };

    struct Node {
        int nxt[alpha];     // Next state transition for each character
        int suflink;        // Suffix link points to longest proper suffix
        int start = -1;     // Start index of pattern in original array
        int end = -1;       // Index in backup of longest matched suffix pattern
        int nmatches = 0;   // Count of matched strings ending at this node

        Node(int v) {
            memset(nxt, v, sizeof nxt);
        }
    };

    vector<Node> v;         // Stores all nodes of the trie
```

```
  vector<int> backup;   // Stores pattern indices with longest matching suffixes
             // Returns -1 if no match exists
             // Note: All patterns must be distinct when using backup

  // Inserts a pattern into the automaton
  // Time: O(|s|) where |s| is pattern length
  void insert(string &s, int id) {
    assert(s.size());   // Empty patterns not allowed
    int node = 0;
    for(auto &c : s) {
      int &m = v[node].nxt[c - first];
      if(m == -1) {
        node = m = v.size();
        v.emplace_back(-1);
      }
      else node = m;
    }
    if(v[node].end == -1) v[node].start = id;
    backup.emplace_back(v[node].end);
    v[node].end = id;
    v[node].nmatches++;
  }

  // Builds Aho-Corasick automaton from patterns
  // Time: O(26N) where N = sum of all pattern lengths
  // - Creates suffix links
  // - Allows duplicate patterns
  // - For large alphabets, split symbols into chunks with sentinel bits
  AhoCorasick(vector<string> &pat): v(1, -1) {
    for(int i = 0; i < pat.size(); ++i)
      insert(pat[i], i);

    v[0].suflink = v.size();  // Dummy node as suffix link of root
    v.emplace_back(0);

    queue<int> q;
    q.push(0);        // BFS from root to build suffix links

    while(q.size()) {
      int node = q.front();
      q.pop();
      int prv = v[node].suflink;

      for(int i = 0; i < alpha; ++i) {
```

```
                int &x = v[node].nxt[i], y = v[prv].nxt[i];
                if(x == -1) x = y;
                else {
                    v[x].suflink = y;
                    (v[x].end == -1 ? v[x].end : backup[v[x].start]) = v[y].end;
                    v[x].nmatches += v[y].nmatches;
                    q.push(x);
                }
            }
        }
    }

    // Returns index of longest word ending at each position, or -1 if none
    // Time: O(|word|) where |word| is text length
    vector<int> find(string &word) {
        int node = 0;
        vector<int> res;
        for(auto &c : word) {
            node = v[node].nxt[c - first];
            res.push_back(v[node].end);
        }
        return res;
    }

    // Finds all patterns starting at each position (shortest first)
    // Time: O(NM) where N = text length, M = number of matches
    // Can find up to N√N matches if no duplicate patterns
    vector<vector<int>> findAll(vector<string> &pat, string word) {
        vector<int> r = find(word);
        vector<vector<int>> res(word.size());
        for(int i = 0; i < word.size(); ++i) {
            int ind = r[i];
            while(ind != -1) {
                res[i - pat[ind].size() + 1].push_back(ind);
                ind = backup[ind];
            }
        }
        return res;
    }
}
```

Basic strings Algo

KMP.cpp

```cpp
void KMP(string str, string pat)
{
  int n = str.length();
  int m = pat.length();
  vector<int> longestPrefix = fail_fun(pat);

  for(int i = 0, k = 0; i < n; i++) {
    // as long as we can't add one more character in k, get best next prefix
    while (k > 0 && pat[k] != str[i])
      k = longestPrefix[k - 1];

    // if we match character in the pattern, move in pattern
    if (pat[k] == str[i])
      k++;

    // if we matched, print it and let's find one more matching
    if (k == m) {
      cout<<i - m + 1<<"\n";
      k = longestPrefix[k - 1];    // fail to next best suffix
    }
  }
}

vector<int> fail_fun(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;
}
```

Z_algorithm.cpp

```cpp
// Z-Function: Returns array z where z[i] is length of longest common prefix
// between s[0..n-1] and s[i..n-1]
// Time: O(n), Memory: O(n)
vector<int> zFunction(string s) {
  int n = s.size();
  vector<int> z(n);
  int left = 0, right = 0;
  for(int i = 1; i < n; i++) {
    if(i <= right) {
      z[i] = min(right - i + 1, z[i - left]);
    }
    while(i + z[i] < n && s[z[i]] == s[z[i] + i]) {
      z[i]++;
    }
    if(i + z[i] - 1 > right) {
      left = i;
      right = i + z[i] - 1;
    }
  }
  return z;
}
```

failer_fun.cpp

```
vector<int> fail_fun(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;
}
```

menacher.cpp

```
// Helper function to transform string for Manacher's algorithm
string transform(string &s) {
    string t;
    for(auto &val : s) {
        t += '#';
        t += val;
    }
    t += '#';
    return t;
}

// Main Manacher's algorithm implementation
vector<int> build(string &s) {
    string t = transform(s);
    int n = t.size();
    vector<int> p(n, 0);
    int l = 0, r = 0;

    for(int i = 0; i < n; i++) {
        p[i] = (i < r) ? min(r - i, p[l + r - i]) : 1;
        while(i - p[i] >= 0 && i + p[i] < n && t[i + p[i]] == t[i - p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            r = i + p[i];
            l = i - p[i];
        }
    }
    return p;
}

// Check if substring [l,r] is palindrome using p array
bool IsPalindrome(int l, int r, vector<int>& p) {
    int center = (l + r) / 2;
    bool odd = (l % 2 == r % 2);
    int newCenter = 2 * center + !odd + 1;
    return (r - l + 1) <= (p[newCenter] - 1);
}

// Returns {start_index, end_index} of longest palindrome substring using p array
pair<int,int> LongestPalindromeSubStr(vector<int>& p) {
    int maxLength = 0;
    int start = 0, end = 0;
    int n = p.size() / 2;  // Original string length

    for(int i = 0; i < n; i++) {
        // Even length palindromes
        int newCenter = 2 * i + 2;
        int len = p[newCenter] - 1;
        if(len > maxLength) {
            maxLength = len;
            start = i - len/2 + 1;
            end = i + len/2;
        }

        // Odd length palindromes
        newCenter = 2 * i + 1;
        len = p[newCenter] - 1;
        if(len > maxLength) {
            maxLength = len;
            start = i - len/2;
            end = i + len/2;
        }
    }

    return {start, end};
```

Hashing

Double_Hash_as_int.cpp

```
const int N = 2e5 + 9;
const int mod[] = { (int)1e9 + 7, 998244353 };
int o;  // Which mod wher I current use.
mt19937 random_seed(time(0));
int rnd(int l , int r){
    uniform_int_distribution<int> dist(l, r);
    return dist(random_seed);
}
ll bpw(ll a , ll b ){
    a %= mod[o];
    ll res = 1;
    while(b){
        if(b % 2) res = (res * a) % mod[o];
        a = (a * a) % mod[o];
        b >>= 1;
    }
    return res;
```

```
}
struct Mint{
    ll x;
    Mint(ll x = 0){  this->x = x % mod[o]; }
    Mint operator +(const Mint &other) const{ return (x + other.x) % mod[o]; }
    Mint operator -(const Mint &other) const{ return (x - other.x + mod[o]) % mod[o]; }
    Mint operator *(const Mint &other) const{ return (x * other.x) % mod[o]; }
    Mint operator /(const Mint &other) const{ return ( x * bpw(other.x , mod[o] - 2) ) % mod[o]; }
    bool operator ==(const Mint &other) const{ return x == other.x; }
    bool operator !=(const Mint &other) const{ return x != other.x; }
};

Mint p[2] , pw[2][N];
void init(){
    for(o = 0; o < 2; o++){
        p[o] = Mint( rnd(31 , 39) );
        pw[o][0] = Mint(1);
        for(int i = 1; i < N; ++i){
            pw[o][i] = pw[o][i - 1] * p[o];
        }
    }
}
struct Hash{
    Mint pref[2] , suff[2];
    int len;
    Hash(char ch = '?'){
        if(ch == '?'){
            pref[0] = pref[1] = suff[0] = suff[1] = Mint(0);
            len = 0;
            return;
        }
        len = 1;
        for(o = 0;o < 2; ++o){
            pref[o] = suff[o] = Mint(ch - 'a' + 1);
        }
    }
    Hash operator +(const Hash &other) const{
        Hash res;
        res.len = len + other.len;
        for(o = 0;o < 2; ++o){
            res.pref[o] = pref[o] + other.pref[o] * pw[o][len];
            res.suff[o] = other.suff[o] + suff[o] * pw[o][other.len];
        }
```

```cpp
        return res;
    }
    void rev(){
        for(o = 0; o < 2; ++o) swap(pref[o] , suff[o]);
    }
    bool is_palindrome() const{
        for(o = 0; o < 2; ++o){
            if(pref[o] != suff[o]) return false;
        }
        return true;
    }
};
Hash excludePrefix(Hash s1 , Hash s2){
    Hash res;
    res.len = s1.len - s2.len;
    for(o = 0; o < 2; ++o){
        res.pref[o] = s1.pref[o] - s2.pref[o];
        res.pref[o] = res.pref[o] / pw[o][s2.len];

        res.suff[o] = s1.suff[o] - s2.suff[o] * pw[o][s1.len - s2.len];
    }
    return res;
}
Hash excludeSuffix(Hash s1 , Hash s2){
    Hash res;
    res.len = s1.len - s2.len;
    for(o = 0; o < 2; ++o){
        res.pref[o] = s1.pref[o] - s2.pref[o] * pw[o][s1.len - s2.len];

        res.suff[o] = s1.suff[o] - s2.suff[o];
        res.suff[o] = res.suff[o] / pw[o][s2.len];
    }
    return res;
```

custom_unorderd_map.cpp

```cpp
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

// to initalize
unordered_map<long long, int, custom_hash> mp;
```

double_hashing.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
const int N = 1e6 + 9;
const int mod[] = { (int)1e9 + 7, 998244353 };
ll p[2] , pw[2][N];
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int gen(int l, int r) { return rng() % (r - l + 1) + l; }
ll get(ll x , ll y){ return (x + 1) * 1e9 + y; } // return Double hashing = get(hash[0] , hash[1])
void init(){
    for(int o = 0; o < 2; o++){
        p[o] = gen(31 , 39); // Generate Base randomly
        pw[o][0] = 1;
        for(int i = 1; i < N; ++i)
            pw[o][i] = pw[o][i - 1] * p[o] % mod[o];
    }
}
vector<ll> gen_prefix(string &s){
    int n = (int) s.size();
    vector<ll> ans(n);
    ll pre[2][n];
    pre[0][0] = pre[1][0] = (s[0] - 'a' + 1);
    for(int i = 1; i < n; ++i) {
        for(int o = 0; o < 2; o++){
            pre[o][i] = (pre[o][i - 1] + (s[i] - 'a' + 1) * pw[o][i]) % mod[o];
        }
    }
    for(int i = 0; i < n; ++i)
        ans[i] = get(pre[i][0] , pre[i][1]);
    return ans;
}
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif
    init();
}
```

hashing_grid.cpp

```
const ll MOD1 = (1LL<<61) - 1, MOD2 = (1LL<<49) - 1, BASE1 = 999983, BASE2 = 99991;
ll hash_matrix[2010][2010];
int64_t MUL(uint64_t a, uint64_t b , uint64_t HashMod) {
    uint64_t l1 = (uint32_t) a, h1 = a >> 32, l2 = (uint32_t) b, h2 = b >> 32;
    uint64_t l = l1 * l2, m = l1 * h2 + l2 * h1, h = h1 * h2;
    uint64_t ret = (l & HashMod) + (l >> 61) + (h << 3) + (m >> 29) + (m << 35 >> 3) + 1;
    ret = (ret & HashMod) + (ret >> 61);
    ret = (ret & HashMod) + (ret >> 61);
    return (int64_t) ret - 1;
}
ll compute(int lx , int rx , int ly , int ry,vector<vector<char>> &pattern) {
    for (int i = lx; i <= rx; i++) {
        ll rowHash = 0;
        for (int j = ly; j <= ry; j++) rowHash = (MUL(rowHash, BASE1, MOD1) + (pattern[i][j] - 'a' + 1)) % MOD1;
        hash_matrix[i][0] = rowHash;
    }
    ll finaleHash = 0;
    for (int j = lx; j <= rx; j++) {
        finaleHash = (MUL(finaleHash, BASE2, MOD2) + hash_matrix[j][0]) % MOD2;
    }
    return finaleHash;
}
```

k-hash.cpp

```cpp
const int mods[] = {1000000007, 1000000009, 1000000021, 1000000033, 1000000087, 1000000093, 1000000097, 1000000103, 1000000123, 1000000181}, b = 10, K = 6;
array<int, K> gethash(string &s) {
   array<int, K> ans, pw;
   for (int k = 0; k < K; k++) {
      ans[k] = 0;
      pw[k] = 1;
   }
   for (auto i : s) {
      for (int k = 0; k < K; k++) {
         ans[k] += 1ll * pw[k] * (i - '0') % mods[k];
         ans[k] %= mods[k];
         pw[k] = (lll) pw[k] * b % mods[k];
      }
   }
   return ans;
}
// probability of collision = 1 / 10^{9 * 6} = 1 / 10^{54}
```

string_hash.cpp

```cpp
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

ll rand(ll l, ll r) {
   return uniform_int_distribution<ll>(l, r)(rng);
}

const int mod1 = 1e9 + 7, mod2 = 1e9 + 9;
const int p1 = rand(1e7, 1e8), p2 = rand(1e7, 1e8);
const int N = 1e6 + 1;
vector<ll> pow1(N + 1), pow2(N + 1);

void init() {
   pow1[0] = pow2[0] = 1;
   for (int i = 1; i <= N; i++) {
      pow1[i] = p1 * pow1[i - 1] % mod1;
      pow2[i] = p2 * pow2[i - 1] % mod2;
   }
}
```

```

struct stringHashing {
    vector<ll> preHash1, preHash2, sufHash1, sufHash2;

    void init(string &s) {
        preHash1 = hash(s, preHash1, pow1, mod1);
        preHash2 = hash(s, preHash2, pow2, mod2);
        sufHash1 = hash(s, sufHash1, pow1, mod1, true);
        sufHash2 = hash(s, sufHash2, pow2, mod2, true);
    }

    vector<ll> hash(string &s, vector<ll> &h, vector<ll> &p, int m, bool rev = false) {
        int n = s.size();
        h.resize(n);
        int st = 0, en = n, delta = 1;
        if (rev)
            st = n - 1, en = -1, delta = -1;
        int i = 0;
        while (st != en) {
            h[st] = (s[st] - 'a' + 1) * p[i] % m;
            if (i != 0) {
                h[st] += h[st - delta];
                if (h[st] >= m)
                    h[st] -= m;
            }
            st += delta;
            i++;
        }
        return h;
    }

    ll query(int l, int r) {
        ll h1 = preHash1[r], h2 = preHash2[r];
        h1 -= (l == 0 ? 0 : preHash1[l - 1]);
        h2 -= (l == 0 ? 0 : preHash2[l - 1]);
        if (h1 < 0)h1 += mod1;
        if (h2 < 0)h2 += mod2;
        h1 = h1 * pow1[N - l] % mod1;
        h2 = h2 * pow2[N - l] % mod2;
        return h1 * h2;
    }

    ll revQuery(int l, int r) {
        int n = sufHash1.size();
```

```cpp
    ll h1 = sufHash1[l], h2 = sufHash2[l];
    h1 -= (r == (n - 1) ? 0 : sufHash1[r + 1]);
    h2 -= (r == (n - 1) ? 0 : sufHash2[r + 1]);
    if (h1 < 0)h1 += mod1;
    if (h2 < 0)h2 += mod2;
    h1 = h1 * pow1[N - n + r + 1] % mod1;
    h2 = h2 * pow2[N - n + r + 1] % mod2;
    return h1 * h2;
  }
```

Lyndon

Duval.cpp

```cpp
// Duval Algorithm: Lyndon Factorization
// Returns lexicographically decreasing Lyndon words
// Time: O(n), Space: O(n)
vector<string> duval(string const& s) {
  int n = s.size();
  int i = 0;
  vector<string> fact;

  while(i < n) {
    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) {
      fact.push_back(s.substr(i, j - k));
      i += j - k;
    }
  }
  return fact;
}

// For index only version (more efficient):
vector<int> duvalIdx(string const& s) {
  int n = s.size();
  int i = 0;
  vector<int> starts;  // Starting positions

  while(i < n) {
    starts.push_back(i);  // Add start of current factor
    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 starts;
}

// Usage example:
/*
string s = "abcab";
auto factors = duval(s);
// factors = ["abc", "ab"]

auto idx = duvalIdx(s);
// idx = [0, 3] meaning factors start at positions 0 and 3
*/
```

min_cyclic_string.cpp

```
// Minimum cyclic shift of string using Duval algorithm
// Returns number of positions to shift right to get lexicographically minimum string
// Example: "cba" -> 2 (shift right 2 to get "abc")
// Time: O(n), Space: O(1)
int min_cyclic_shift(string s) {
  s += s;
  int n = s.size();
  int i = 0, shift = 0;

  while(i < n/2) {
    shift = i;
    int j = i + 1, k = i;
    while(j < n && s[k] <= s[j]) {
      if(s[k] < s[j]) k = i;
      else k++;
      j++;
    }
```

```cpp
        while(i <= k) i += j - k;
    }

    return shift;
}

// Usage:
/*
string s = "cba"
int shift = min_cyclic_shift(s); // returns 2
rotate(s.begin(), s.begin() + shift, s.end()); // gets "abc"

s = "aaaa"
shift = min_cyclic_shift(s); // returns 0 (already minimum)
*/
```

suffixArray

Kasai.cpp

```cpp
/*
Build the LCP array using kasai algorithm in O(n) time
s: string , p: suffix array

lcp[i]: longest comment prefix of suffix[i] , suffix[i + 1]
*/
vector<int> Kasai(string const& s, vector<int> const& p) {
    int n = s.size();
    vector<int> rank(n, 0);
    for(int i = 0; i < n; i++) rank[p[i]] = i;
    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if(rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k]) k++;
        lcp[rank[i]] = k;
        if(k) k--;
    }
```

```
  return lcp;
}
```

RMQ_suffixarray.cpp

```
// Builds RMQ table for suffix array comparisons
// Returns equivalence classes for each power of 2
// Time: O(nlogn), Memory: O(nlogn)
vector<vector<int>> buildRMQSuffixArray(string s) {
  const int ALPHA = 256;
  s += '$';
  int n = s.size();
  vector<int> p(n), c(n), cnt(max(n, ALPHA), 0);
  vector<vector<int>> ans;
  for(int i = 0; i < n; i++) cnt[s[i]]++;
  for(int i = 1; i < ALPHA; i++) cnt[i] += cnt[i-1];
  for(int i = 0; i < n; i++) p[--cnt[s[i]]] = i;

  int classes = 1;
  for(int i = 1; i < n; i++) {
    if(s[p[i]] != s[p[i-1]]) classes++;
    c[p[i]] = classes - 1;
  }
  ans.emplace_back(c);
  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++) {
      int pos1 = p[i] + (1 << h);
      int pos2 = p[i-1] + (1 << h);
      // Replace modulo with comparison and subtraction
      if(pos1 >= n) pos1 -= n;
      if(pos2 >= n) pos2 -= n;

      pair<int,int> cur = {c[p[i]], c[pos1]};
      pair<int,int> prev = {c[p[i-1]], c[pos2]};
      if(cur != prev) ++classes;
      cn[p[i]] = classes - 1;
    }
    c.swap(cn);
    ans.emplace_back(c);
  }
  p.erase(p.begin());
  return ans;
}

// Get LCP of suffixes starting at i,j
// Time: O(logn)
int LCP(int i, int j, vector<vector<int>> &c) {
  int ans = 0, n = c[0].size(), lg_n = c.size();
  for(int k = lg_n - 1; k >= 0; k--) {
    if(c[k][i % n] == c[k][j % n]) {
      ans += 1 << k;
      i += 1 << k;
      j += 1 << k;
    }
  }
  return ans;
}

// Compare same length suffixes starting at i,j
// Returns: -1 if i<j, 0 if equal, 1 if i>j
// Time: O(1)
int compare(int i, int j, int l, vector<vector<int>> &c) {
  int n = c[0].size();
  int k = 31 - __builtin_clz(l);
  pair<int,int> a = {c[k][i], c[k][(i+l-(1<<k))%n]};
  pair<int,int> b = {c[k][j], c[k][(j+l-(1<<k))%n]};
  return a == b ? 0 : a < b ? -1 : 1;
}

// Compare substrings s[l1..r1] vs s[l2..r2]
// Returns: -1 if s1<s2, 0 if equal, 1 if s1>s2
// Time: O(logn)
int compare(int l1, int r1, int l2, int r2, string &s, vector<vector<int>> &c) {
```

```cpp
    int lcp = LCP(l1, l2, c);
    int len1 = r1 - l1 + 1, len2 = r2 - l2 + 1;
    lcp = min({lcp, len1, len2});
    if(lcp == len1 && lcp == len2) return 0;
    if(lcp == len1) return -1;
    if(lcp == len2) return 1;
    return s[l1 + lcp] < s[l2 + lcp] ? -1 : 1;
}
```

build_suffixArray_slow_O(nlognlogn).cpp

```cpp
vector<int> buildSuffixArray(string const & s){
    int n = s.length();
    vector<int> sa(n + 1) , rank(n + 1);
    for(int i = 0; i <= n; ++i) {
        sa[i] = i;
        rank[i] = i < n ? s[i] : -1;
    }
    auto rankf = [&](int i) { return i <= n ? rank[i] : -1; };
    vector<int> nxt(n + 1);
    for (int k = 1; k <= n ; k <<= 1) {
        auto cmp = [&](int i, int j) { return make_pair(rank[i], rankf(i + k)) < make_pair(rank[j], rankf(j + k)); };
        sort(sa.begin(), sa.end(), cmp);
        nxt[sa[0]] = 0;
        for(int i = 1; i <= n; ++i){
            nxt[sa[i]] = nxt[sa[i - 1]] + (cmp(sa[i - 1], sa[i]) ? 1 : 0);
        }
        rank.swap(nxt);
    }
    sa.erase(sa.begin());
    return sa;
}
```

build_suffixarray_veryfast.cpp

```cpp
vector<int> buildSuffixArray(string s) {
    int n = s.size(); s += "$";
    vector<int> p(n+1), c(n+1), c1(n+1), cnt(max(256, n+1)), p1(n+1);
    for(int i = 0; i <= n; i++) p[i] = i, c[i] = s[i];

    for(int k = 0; (1 << k) <= n; k++) {
        int len = 1 << k;

        fill(cnt.begin(), cnt.end(), 0);
        for(int i = 0; i <= n; i++) cnt[c[min(n, p[i] + len)]]++;
        for(int i = 1; i < cnt.size(); i++) cnt[i] += cnt[i-1];
        for(int i = n; i >= 0; i--) p1[--cnt[c[min(n, p[i] + len)]]] = p[i];

        fill(cnt.begin(), cnt.end(), 0);
        for(int i = 0; i <= n; i++) cnt[c[p1[i]]]++;
        for(int i = 1; i < cnt.size(); i++) cnt[i] += cnt[i-1];
        for(int i = n; i >= 0; i--) p[--cnt[c[p1[i]]]] = p1[i];

        c1[p[0]] = 0;
        for(int i = 1; i <= n; i++)
            c1[p[i]] = c1[p[i-1]] + (c[p[i]] != c[p[i-1]] || c[min(n,p[i]+len)] != c[min(n,p[i-1]+len)]);
        c.swap(c1);
        if(c[p[n]] == n) break;
    }

    vector<int> res;
    for(int i = 1; i <= n; i++) res.push_back(p[i]);
    return res;
}
```

problems

count_pattern.cpp

```
// Problem link: https://cses.fi/problemset/task/2103/
pair<int,int> findPattern(const string &text, const string& pat, vector<int> &sa) {
    /*
        Count the no. of occurence of pattern "pat" into "text" into O(2log(n))
        sa: Suffix order
    */
    int n = sa.size();
    auto compare = [&](int pos) {
        return text.compare(sa[pos], pat.length(), pat);
    };

    // Find leftmost occurrence
    int left = 0, right = n-1;
    while(left < right) {
        int mid = left + (right - left) / 2;
        if(compare(mid) >= 0) right = mid;
        else left = mid + 1;
    }
    if(compare(left) != 0) return {-1, -1};
    int start = left;

    // Find rightmost occurrence
    right = n-1;
    while(left < right) {
        int mid = left + (right - left + 1) / 2;
        if(compare(mid) <= 0) left = mid;
        else right = mid - 1;
    }

    return {start, left};
}

\\-----------------------------------------------
  auto find_string = [&](const string &pat)->pair<int,int> {
    int st = 0, ed = n;
    auto cmp = [&](int a, int b) {
        if (a == -1)
            return pat[i] < text[b + i];
        return text[a + i] < pat[i];
    };
    for (int i = 0; i < pat.size() && st < ed; i++) {
        st = lower_bound(p.begin() + st, p.begin() + ed, -1, cmp)
             - p.begin();
        ed = upper_bound(p.begin() + st, p.begin() + ed, -1, cmp)
             - p.begin();
    }
    if(st >= ed) return {-1,-1};
    return {st, ed - 1};
};
```

distinct_substring.cpp

```
int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    string s; cin >> s;
    int n = s.size();
```

```
  auto p = buildSuffixArray(s);
  auto lcp = Kasai(s , p);
  ll distinct = 1LL*n*(n+1)/2 - accumulate(lcp.begin(),lcp.end() , 0LL);
  cout << distinct << '\n';
}
```

kth_distinct_substring.cpp

```
/*
Problem link: https://cses.fi/problemset/task/2108/
You are given a string of length n.
If all of its distinct substrings are ordered lexicographically, what is the kth smallest of them?
*/
int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
  string s; cin >> s;
  int n = s.size();
  auto p = buildSuffixArray(s);
  auto lcp = Kasai(s,p);
  ll k; cin >> k;
  if(k <= n - p[0]){ cout << s.substr(p[0], k); return 0; }
  k -= n - p[0];
  for(int i = 0; i < lcp.size(); ++i){
    int len = n - p[i+1] - lcp[i];
    if(k <= len){ cout << s.substr(p[i+1],k+lcp[i]); return 0; }
    k -= len;
  }
}
```

kth_smallest_string.cpp

```
/*
Problem link: https://cses.fi/problemset/task/2109/
You are given a string of length n.
If all of its substrings (not necessarily distinct) are ordered lexicographically, what is the kth smallest of them?
*/
string s; cin >> s;
  int n = s.size();
  auto p = buildSuffixArray(s);
```

```
    auto lcp = Kasai(s,p);

    auto k_th = [&](ll k)->pair<int,int>{ // {index , length}
        if(k <= n - p[0]) return {0, k};
        k -= n - p[0];
        for(int i = 0; i < lcp.size(); ++i){
            int len = n - p[i+1] - lcp[i];
            if(k <= len) return {i+1 , k + lcp[i]} ;
            k -= len;
        }
        assert(false);
    };

    ll k; cin >> k;
    ll lo = 1 , hi = 1LL * n*(n+1) / 2 - accumulate(lcp.begin(), lcp.end(), 0LL) , md;

    while(lo < hi){
        md = lo + (hi - lo) / 2;

        auto [i, len] = k_th(md);

        ll sum = len;
        for(int j = 0; j < i; ++j) sum += n - p[j];
        int x = len;
        for(int j = i+1; j < n; ++j){
            x = min(x , lcp[j-1]);
            sum += x;
        }

        if(sum >= k) hi = md;
        else lo = md + 1;

    }

    auto [i,len] = k_th(lo);
    cout << s.substr(p[i] , len);
```

no_of_distinct_string_of_length_x.cpp

```
/*
Problem link: https://cses.fi/problemset/task/2110/
You are given a string of length n.
```

```
For every integer between 1, n you need to print the number of distinct substrings of that length.

*/
string s; cin >> s;
int n = s.size();
auto p = buildSuffixArray(s);
auto lcp = Kasai(s,p);

ll cnt[n+2]{};

auto upd = [&](int l , int r, int v){
if(l > r)return;
   cnt[l] += v; cnt[r+1] -= v;
};

for(int i = 0; i < lcp.size(); ++i)upd(1,lcp[i] , -1);

for(int i = 1; i <= n; ++i) cnt[i] += cnt[i-1];
for(int i = 1; i <= n; ++i) cnt[i] += n - i + 1;
for(int i = 1; i <= n; ++i) cout << cnt[i] << ' ';
```

no_of_substring_at_least_f.cpp

```
/*
Given a string S and several frequencies Fi.
For each Fi output the number of substrings of S (the characters of substring should be contiguous)
that occur at least Fi times in S. Note, that we consider two substrings distinct
if they have distinct length, or they have distinct starting indices.
problem link: https://www.codechef.com/problems/ANUSAR


*/
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

vector<int> buildSuffixArray(string s) {
   const int ALPHA = 256;
   s += '$';
   int n = s.size();
   vector<int> p(n), c(n);
```

```
    {
        // Optimize first phase with counting sort
        vector<int> cnt(ALPHA);
        for(auto ch : s) cnt[ch]++;
        for(int i = 1; i < ALPHA; i++) cnt[i] += cnt[i-1];
        for(int i = n-1; i >= 0; 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) {
        int len = 1 << h;
        for(int i = 0; i < n; i++) {
            pn[i] = p[i] - len;
            if(pn[i] < 0) pn[i] += n;
        }

        // Optimize counting sort for each phase
        vector<int> cnt(n);
        for(int i = 0; i < n; i++) cnt[c[pn[i]]]++;
        for(int i = 1; i < n; 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;
        int classes = 1;
        for(int i = 1; i < n; i++) {
            pair<int,int> cur = {c[p[i]], c[(p[i] + len) % n]};
            pair<int,int> prev = {c[p[i-1]], c[(p[i-1] + len) % n]};
            if(cur != prev) ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    p.erase(p.begin());
    return p;
}

vector<int> buildLCP(const string& s, const vector<int>& p) {
    int n = s.size();
    vector<int> rank(n), lcp(n-1);
    for(int i = 0; i < n; i++) rank[p[i]] = i;

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

// Optimized stack-based nearest smaller element
pair<vector<int>, vector<int>> getMinBounds(const vector<int>& arr) {
    int n = arr.size();
    vector<int> left(n), right(n);
    vector<int> st;
    st.reserve(n);

    // Get left bounds
    for(int i = 0; i < n; i++) {
        while(!st.empty() && arr[st.back()] > arr[i]) st.pop_back();
        left[i] = st.empty() ? -1 : st.back();
        st.push_back(i);
    }

    // Clear stack for right bounds
    st.clear();

    // Get right bounds
    for(int i = n-1; i >= 0; i--) {
        while(!st.empty() && arr[st.back()] >= arr[i]) st.pop_back();
        right[i] = st.empty() ? n : st.back();
        st.push_back(i);
    }

    return {left, right};
}
```

```

void solve() {
    string s;
    cin >> s;
    int n = s.size();

    auto p = buildSuffixArray(s);
    auto lcp = buildLCP(s, p);
    auto [L, R] = getMinBounds(lcp);

    // Pre-allocate vectors to avoid resizing
    vector<ll> frq(n+9);
    vector<vector<int>> group(n+1);
    for(int i = 0; i < n+1; i++) group[i].reserve(n/2);

    // Group LCP values
    for(int i = 0; i < (int)lcp.size(); ++i) {
        group[lcp[i]].push_back(i);
    }

    // Calculate frequencies
    for(int len = n; len > 0; --len) {
        for(size_t j = 0; j < group[len].size();) {
            int x = group[len][j];
            int l = L[x], r = R[x];
            int f = r - l;
            int minh = len;

            if(l >= 0) minh = min(minh, len - lcp[l]);
            if(r < (int)lcp.size()) minh = min(minh, len - lcp[r]);

            frq[f] += 1LL * f * minh;

            // Skip processed indices
            while(j < group[len].size() && group[len][j] <= r) ++j;
        }
    }

    // Calculate cumulative frequencies
    for(int i = n-1; i > 1; --i) frq[i] += frq[i+1];
    frq[1] = 1LL * n * (n + 1) / 2;

    // Process queries
    int q;
    cin >> q;
    while(q--) {
        int x;
        cin >> x;
        cout << (x > s.size() ? 0 : frq[x]) << '\n';
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int tc;
    cin >> tc;
    while(tc--) solve();
```

trie

Binary_trie.cpp

```
const int N = 2e5 * 30 + 9;
int nxt[N][2], isEnd[N], cntNode, frq[N];
void add(int x){
    int node = 0;
    for(int i = 30; i >= 0; --i){
        int cur = (x >> i) & 1;
        if(nxt[node][cur] == 0)
            nxt[node][cur] = ++cntNode;
        node = nxt[node][cur];
        frq[node]++;
    }
    isEnd[node] = true;
}
void erase(int x){
    int node = 0;
    for(int i = 30; i >= 0; --i){
        int cur = (x >> i) & 1;
        if(nxt[node][cur] == 0) return; // this number "x" doesn't exist, (handle it manual)
        node = nxt[node][cur];
        frq[node]--;
    }
    if(frq[node] == 0) isEnd[node] = false;
}
```

trie.cpp

```
const int N = 1e6 + 9;
int nxt[N][27] , cntNode, isEnd[N], frq[N], n;
int get(char ch){ return  ch - 'a'; }
void add(string &s){
    int node = 0;
    for(auto &ch : s){
        if(nxt[node][get(ch)] == 0)
            nxt[node][get(ch)] = ++cntNode;
        node = nxt[node][get(ch)];
        frq[node]++;
    }
    isEnd[node] = true;
}
void remove(string &s){
    int node = 0;
    for(auto &ch : s){
        if(nxt[node][get(ch)] == 0) return; // this word "s" doesn't exist, (handle it manual)
        node = nxt[node][get(ch)];
        frq[node]--;
    }
    if(frq[node] == 0) isEnd[node] = false;
}
```

Trees

CentroidDecomposition.cpp

```
const int N = 1e5 + 9;
vector<int> adj[N];
struct CentroidDecomposition{
    vector<int> removal , sz;
    CentroidDecomposition(int n){
        removal.assign(n , 0);
        sz.assign(n , 0);
        build(0, -1);
```

```cpp
    }
    void build(int u , int p){
        int n = dfs(u , p);
        int centriod = getCentriod(u , p , n);
        removal[centriod] = true;
        // depend on the problem

        for(auto &v : adj[centriod]) if(!removal[v])
            build(v , centriod);
    }
    int dfs(int u , int p){
        sz[u] = 1;
        for(auto &v : adj[u]) if(v != p && !removal[v])
            sz[u] += dfs(v , u);
        return sz[u];
    }
    int getCentriod(int u , int p , int n){
        for(auto &v : adj[u]) if(v != p && !removal[v]){
            if(sz[v] * 2 > n)
                return getCentriod(v , u , n);
        }
        return u;
    }

};
```

HLD.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
#define all(v) v.begin() , v.end()
#ifndef ONLINE_JUDGE
#define debug(x) cerr << #x << ": " << x << '\n';
#else
#define debug(x)
#endif
const int N = 2e5 + 9;
vector<int> adj[N];
```

```
int heavy[N], head[N], par[N], pos[N], dep[N],  cur_pos;
int dfs(int u = 0){
    heavy[u] = -1; // node is leaf
    int size = 1 , max_size = 0;
    for(auto &v : adj[u]) if(v != par[u]){
        par[v] = u;
        dep[v] = dep[u] + 1;
        int cur = dfs(v);
        if(cur > max_size){
            heavy[u] = v;
            max_size = cur;
        }
        size += cur;
    }
    return size;
}
void decomposition(int u = 0, int h = 0){
    head[u] = h;
    pos[u] = cur_pos++;
    if(~heavy[u])
        decomposition(heavy[u], h);
    for(auto &v : adj[u]) if(v != par[u] && v != heavy[u])
        decomposition(v,v);
}
int path(int u , int v){
    int ans = 0;
    for(; head[u] != head[v]; v = par[head[v]]){
        if(dep[head[u]] > dep[head[v]]) swap(u,v);
        // proccess interval: [ pos[head[v]] , pos[v]  ]

    }
    if(dep[u] > dep[v]) swap(u,v);
    // proccess interval: [ pos[u] , pos[v] ]

    return ans;
}


int main(){
    ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
    #ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
```

```cpp
    #endif
    int n,q; cin >> n >> q;
    vector<int> a(n);
    for(int i = 0; i < n; ++i) cin >> a[i];
    for(int i = 0; i < n - 1;++i){
        int u , v; cin >> u >> v;
        --u; --v;
        adj[u].emplace_back(v);
        adj[v].emplace_back(u);
    }
    dfs(); decomposition();
}
```

HLD_edges.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 2e5 + 9;

vector<pair<int,int>> adj[N];
int heavy[N], head[N], par[N], pos[N], dep[N], sz[N], cur_pos, n;
vector<int> values;

int dfs(int u = 0) {
    heavy[u] = -1;
    sz[u] = 1;
    int max_size = 0;
    for(auto &[v, w] : adj[u]) if(v != par[u]) {
        par[v] = u;
        dep[v] = dep[u] + 1;
        int cur = dfs(v);
        if(cur > max_size) {
            heavy[u] = v;
            max_size = cur;
        }
        sz[u] += cur;
    }
    return sz[u];
}

void decomposition(int u = 0, int h = 0) {
    head[u] = h;
    pos[u] = cur_pos++;
    if(~heavy[u])
        decomposition(heavy[u], h);
    for(auto &[v, w] : adj[u]) if(v != par[u] && v != heavy[u])
        decomposition(v, v);
}

void buildWeightArray() {
    values.resize(n);
    for(int u = 0; u < n; u++) {
        for(auto [v, w] : adj[u]) {
            if(dep[u] < dep[v]) {
                values[pos[v]] = w;
            }
        }
    }
}

int queryPath(int u, int v) {
    int ans = -1e9;
    for(; head[u] != head[v]; v = par[head[v]]) {
        if(dep[head[u]] > dep[head[v]]) swap(u,v);
        // process range [pos[head[v]], pos[v]]
    }
    if(dep[u] > dep[v]) swap(u,v);
    if(u != v) {
        // process range [pos[u] + 1, pos[v]]
    }
    return ans;
}

// Range to query: [pos[u] + 1, pos[u] + sz[u] - 1]
int querySubtree(int u) {
    // process range [pos[u] + 1, pos[u] + sz[u] - 1]
    return 0;  // replace with actual query
}

int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    int m, q; cin >> n >> m >> q;

    for(int i = 0; i < m; i++) {
        int u, v, w; cin >> u >> v >> w;
        --u; --v;
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});
    }

    dfs();
    decomposition();
    buildWeightArray();
```

Tree_center_using_bfs.cpp

```
/*
  Finding Tree Center and Tree diameter in Time O(n).
  Algo :
    1- Starting BFS from any node.
    2- Find the farthest node (Start) from it.
    3- Starting BFS from (Start).
    5- Find the farthest node (End) from (Start)
    6- The path from (Start) and (End) is one possible diameter for the tree.
*/
int n; cin >> n;
  for(int i = 0; i < n - 1; ++i){
    int u , v; cin >> u >> v;
    --u; --v;
    adj[u].emplace_back(v);
    adj[v].emplace_back(u);
  }
  auto bfs = [&](int st){
    vector<int> dis(n , -1);
    queue<int> q;
    q.push(st); dis[st] = 0;
      while(q.size()){
        int u = q.front(); q.pop();
        for(auto &v : adj[u]) if(dis[v] == -1){
          dis[v] = dis[u] + 1; pre[v] = u;
          q.push(v);
        }
      }
      return max_element(dis.begin() , dis.end()) - dis.begin();
  };
  int Start = bfs(0) , End = bfs(Start);
  vector<int> v;
  for(int i = End;; i = pre[i]){
    v.emplace_back(i);
    if(i == Start) break;
  }
  int center = v[v.size() / 2];
```

Tree_flatten.cpp

```
const int N = 1e5 + 9;
int l[N] , r[N] , timer;
vector<int> adj[N];
void dfs(int u , int par){
  l[u] = ++timer;
  for(auto &v : adj[u]) if(v != par)
    dfs(v , u);
  r[u] = timer;
  // For each subTree u : [ l[u] , r[u] ]
}
```

bridgestree.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i , st , ed) for(int i = st; i < ed; i++)
#define f first
#define s second
const int N = 3e5 + 9;
vector<pair<int,int>> adj[N] , edges;
vector<int> BridgeTree[N];
int lowLink[N] , dfn[N] , comp[N] , ndfn , comp_num;
bool isBridge[N] , vis[N];
void tarjan(int u , int par){
 dfn[u] = lowLink[u] = ndfn++;
 for(auto &[v , id] : adj[u]){
  if(dfn[v] == -1){
   tarjan(v , u);
   lowLink[u] = min(lowLink[u] , lowLink[v]);
    if(lowLink[v] == dfn[v]){
      int uu = u , vv = v;
      if(uu > vv) swap(uu , vv);
      isBridge[id] = true;
    }
  }else if(v != par){
   lowLink[u] = min(lowLink[u] , dfn[v]);
  }
 }
}
void Find_component(int u , int par){
  vis[u] = true;
  comp[u] = comp_num;
  for(auto &[v , id] : adj[u]) if(vis[v] == 0 && isBridge[id] == 0)
    Find_component(v , u);
}
pair<int, int> diameter(int u, int par = -1)
{
  int diam = 0;
  int mxHeights[3] = {-1, -1, -1};   // keep 2 highest trees
  for(auto &v : BridgeTree[u]) if(v != par)
  {
    auto p = diameter(v , u);
    diam = max(diam, p.f);
    mxHeights[0] = p.s+1;
    sort(mxHeights, mxHeights+3);
  }
  for(int i = 0; i < 3; i++)if(mxHeights[i] == -1)
    mxHeights[i] = 0;
  diam = max(diam, mxHeights[1] + mxHeights[2]);
  return {diam, mxHeights[2]};
}
int main(){
  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
  #ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
  #endif
  int n, m; cin >> n >> m;
  for(int i = 0; i < m; i++){
    int u , v; cin >> u >> v;
    --u; --v;
    adj[u].emplace_back(v,  i);
    adj[v].emplace_back(u , i);
    edges.emplace_back(u , v);
  }
  // Finding Bridges using Tarjan algo.
  for(int i = 0;i < n; i++){ dfn[i] = -1; lowLink[i] = 0; }
  ndfn = 0;
  tarjan(0 , 0);
  // dfs to group all the maximal components together, so that we can shrink it to one node
  for(int i = 0; i < n; i++) if(vis[i] == 0){
    Find_component(i , i);
    comp_num++;
  }
  // shrinking all the maximal components to one node
  for(int i = 0; i < m; i++) {
    if(isBridge[i]) {
      BridgeTree[comp[edges[i].f]].emplace_back(comp[edges[i].s]);
      BridgeTree[comp[edges[i].s]].emplace_back(comp[edges[i].f]);
    }
  }
  // Finding the diameter of the Bridgestree
  int d = diameter(0 , 0).f;
  cout << d;
}
```

centroid.cpp

```
/*
 Simply Centroid is a node if we delete it. It makes some subtrees where every subtree size must be less than sz/2 { sz is the size of the current tree T.}
*/
vector<int> adj[N] , centriod, sz(N);
int n;
void dfs(int u , int par){
 sz[u] = 1;
 bool is_Centriod = true;
 for(auto &v : adj[u]){
  if(v != par){
   dfs(v , u);
   sz[u] += sz[v];
   if(sz[v] * 2 > n) is_Centriod = false;
  }
 }
 // check above tree
 if((n - sz[u]) * 2 > n) is_Centriod = false;
 if(is_Centriod) centriod.emplace_back(u);
}
/*
```

```
  in main
  dfs(0 , -1)
*/
```

isophrisim.cpp

```
#include <bits/stdc++.h>
typedef long long ll;
#define s second
#define f first
#define rep(i , st , ed) for(int i = st ; i < ed ; i++)
using namespace std;
const int N = 4000;
void burn(){
ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
#ifndef ONLINE_JUDGE
   freopen("in.txt", "r", stdin);
   freopen("out.txt", "w", stdout);
   freopen("error.txt", "w", stderr);
  #endif
}
//\//\//\/\//\//\//\//\//\//
int n;
vector<int> adj[N];
string get_subcan(int u , vector<vector<string>> &subcan){
  sort(subcan[u].begin() , subcan[u].end());
  string ans = "(";
  for(auto &s : subcan[u]) ans += s;
  ans += ")";
  return ans;
}
void tree_isophrisim(){
  vector<int> deg(n);
  queue<int> leaf;
  vector<vector<string>> subcan(n);
  rep(i , 0 , n){
    deg[i] = adj[i].size();
    if(deg[i] <= 1) leaf.push(i);
  }
  int rem = n;
  while(rem > 2){
    int sz = leaf.size();
    while(sz--){
      rem--;
      int u = leaf.front(); leaf.pop();
      string temp = get_subcan(u ,  subcan);
      for(auto &v : adj[u]){
        subcan[v].emplace_back(temp);
        deg[v]--;
        if(deg[v] == 1) leaf.push(v);
      }
    }
  }
  vector<string> ans; // contain all possible canonical
  int c1 = leaf.front(); leaf.pop();
  if(leaf.empty()){
    // tree has one center
    ans.emplace_back(get_subcan(c1 , subcan));
  }else{
    // tree has 2 center
    int c2 = leaf.front();
    string temp1 = get_subcan(c1 , subcan),
           temp2 = get_subcan(c2 , subcan);
    subcan[c1].push_back(temp2);
    subcan[c2].push_back(temp1);
    ans.emplace_back(get_subcan(c1 , subcan));
    ans.emplace_back(get_subcan(c2 , subcan));
  }
}
int main(){
  burn();
}
```

tree_center.cpp

```
#include <bits/stdc++.h>
typedef long long ll;
#define s second
#define f first
#define rep(i , st , ed) for(int i = st ; i < ed ; i++)
using namespace std;
const int N = 4000;
void burn(){
ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
#ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
 #endif
}
//\//\//\/\//\//\//\//\//\//\//
int n;
vector<int> adj[N];
void tree_center(){
 vector<int> deg(n);
 queue<int> leaf;
 rep(i , 0 , n){
  deg[i] = adj[i].size();
  if(deg[i] <= 1) leaf.push(i);
 }
 int rem = n;
 while(rem > 2){
  int sz = leaf.size();
  while(sz--){
   rem--;
   int u = leaf.front(); leaf.pop();
   for(auto &v : adj[u]){
    deg[v]--;
    if(deg[v] == 1) leaf.push(v);
   }
  }
 }
 int c1 = leaf.front(); leaf.pop();
 int c2 = (leaf.size()) ? leaf.front() : -1;
}
int main(){
 burn();
}
```

tree_diameter_dfs.cpp

```
pair<int, int> diameter(int u, int par = -1)
{
  int diam = 0;
  int mxHeights[3] = {-1, -1, -1};   // keep 2 highest trees

```

```

    for(auto &v : adj[u]) if(v != par)
    {
        pair<int, int> p = diameter(v, u);
        diam = max(diam, p.first);

        // Keep only the 2 maximum children
        mxHeights[0] = p.second+1;
        sort(mxHeights, mxHeights+3);
    }

    for(int i = 0; i < 3; ++i)if(mxHeights[i] == -1)
        mxHeights[i] = 0;

    diam = max(diam, mxHeights[1] + mxHeights[2]);

    return make_pair(diam, mxHeights[2]);
}
```