CODE LIBRARY

MD. SHAHADAT HOSSAIN SHAHIN

UNIVERSITY OF DHAKA

Contents

1 Template

```cpp
#include <bits/stdc++.h>
//#include <ext/pb_ds/assoc_container.hpp>
//#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
//using namespace __gnu_pbds;

typedef long long ll;
typedef unsigned long long ull;
typedef long double ld;
typedef pair <int,int> pii;
typedef pair <ll,ll> pll;

#define si(a)           scanf("%d",&a)
#define sii(a,b)        scanf("%d %d",&a,&b)
#define siii(a,b,c)     scanf("%d %d %d",&a,&b,&c)

#define sl(a)           scanf("%lld",&a)
#define sll(a,b)        scanf("%lld %lld",&a,&b)
#define slll(a,b,c)     scanf("%lld %lld %lld",&a,&b,&c)

#define un(x)           x.erase(unique(all(x)), x.end())
#define xx              first
#define yy              second
#define pb              push_back
#define mp              make_pair
#define all(v)          v.begin(),v.end()
#define D(x)            cerr << #x " = " << x << '\n'
#define DBG             cerr << "Hi!" << '\n'

#define CLR(a)          memset(a,0,sizeof(a))
#define SET(a)          memset(a,-1,sizeof(a))

#define PI              acos(-1.0)

//inline int setBit(int n,int pos) { return n = n | (1 << pos); }
//inline int resetBit(int n,int pos) { return n = n & ~(1 << pos); }
//inline bool checkBit(int n,int pos) { return (bool)(n & (1 << pos)); }
//inline int countBit(ll n) { return __builtin_popcountll(n); }


//int fx[] = {+0, +0, +1, -1, -1, +1, -1, +1};
//int fy[] = {-1, +1, +0, +0, +1, +1, -1, -1}; //Four & Eight Direction

/*
    ********************************************************************************
    */

//const int MAX = 500010;
```

```cpp
//const int INF = 0x3f3f3f3f;
//const double inf = 1.0/0.0;
//const int MOD = 1000000007;
//inline int add(int a,int b) { return (a + 0LL + b) % MOD; }
//inline int mul(int a,int b) { return (a * 1LL * b) % MOD; }


int main() {
//    freopen("in.txt","r",stdin);
//    freopen("out.txt","w",stdout);

//    ios_base::sync_with_stdio(false);
//    cin.tie(0);

    return 0;
}
```

2 DP

2.1 Convex Hull Trick

```cpp
#include <bits/stdc++.h>
//#include <ext/pb_ds/assoc_container.hpp>
//#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
//using namespace __gnu_pbds;

typedef long long ll;
typedef unsigned long long ull;
typedef long double ld;
typedef pair <int,int> PII;
typedef pair <long long,long long> PLL;

#define si(a)           scanf("%d",&a)
#define sii(a,b)        scanf("%d %d",&a,&b)
#define siii(a,b,c)     scanf("%d %d %d",&a,&b,&c)

#define sl(a)           scanf("%lld",&a)
#define sll(a,b)        scanf("%lld %lld",&a,&b)
#define slll(a,b,c)     scanf("%lld %lld %lld",&a,&b,&c)

#define un(x)           x.erase(unique(all(x)), x.end())
#define xx              first
#define yy              second
#define pb              push_back
#define mp              make_pair
#define all(v)          v.begin(),v.end()
#define D(x)            cerr << #x " = " << x << '\n'
#define DBG             cerr << "Hi!" << '\n'

#define CLR(a)          memset(a,0,sizeof(a))
```

```
32 #define SET(a)          memset(a,-1,sizeof(a))
33
34 #define PI              acos(-1.0)
35
36 int setBit(int n,int pos) { return n = n | (1 << pos); }
37 int resetBit(int n,int pos) { return n = n & ~(1 << pos); }
38 bool checkBit(ll n,ll pos) { return (bool)(n & (1LL << pos)); }
39
40 //int fx[] = {+0, +0, +1, -1, -1, +1, -1, +1};
41 //int fy[] = {-1, +1, +0, +0, +1, +1, -1, -1}; //Four & Eight Direction
42
43 /*
    ********************************************************************************
    */
44
45 const int MAX = 200010;
46 const int INF = 2000000000000000;
47 //const int MOD = 1000000007;
48
49
50 /**
51     *   Lines should be added non-increasing order of m for minimizing
52         Non-decreasing order of m for maximizing
53     *   Intersection point of two lines (m1,c1) , (m2,c2) is
54         x = (c2-c1)/(m1-m2)
55 **/
56
57 ll M[MAX] , C[MAX];
58
59 struct CHT {
60     int len , cur;
61     void init() {
62         len = 0 ,cur = 0;
63     }
64
65     /// returns true if line[len-1] is unnecessary when we add line(nm,nc)
66     inline bool isBad(ll nm,ll nc) {
67         return ( (C[len-1]-C[len-2])/(double)(M[len-2]-M[len-1]) >= (nc-C[len
            -2])/(double)(M[len-2]-nm) );
68         //return ( (C[len-1]-C[len-2])*(M[len-2]-nm) >= (M[len-2]-M[len-1])*(
            nc-C[len-2]) );
69     }
70
71     inline void addLine(ll nm,ll nc) {
72         if(len == 0) M[len] = nm , C[len] = nc , ++len;
73         else if( M[len-1] == nm ) {
74             if(C[len-1] <= nc) return; /// <= to minimize, >= to maximize
75             else C[len-1] = nc;
76         }
77         else {
78             while(len >= 2 && isBad(nm,nc)) --len;
```

```
            M[len] = nm , C[len] = nc , ++len;
        }
    }

    inline ll getY(int id , ll x) {
        return ( M[id]*x + C[id] );
    }

    inline ll sortedQuery( ll x ) {
        if(cur >= len ) cur = len-1;
        while( cur < len-1 && getY(cur+1,x) >= getY(cur,x) ) cur++; /// <= to minimize, >= to maximize
        return getY(cur,x);
    }

    inline ll TS( ll x ) {
        int low = 0, high = len-1 , mid ;
        while( high - low > 1 ) {
            mid = low + high >> 1;
            if(getY(mid,x) < getY(mid+1,x)) low = mid + 1; /// > to minimize , < to maximize
            else high = mid;
        }
        return max(getY(low,x),getY(high,x)); /// adjust min/max
    }
};

int main() {
    CHT cht;
    cht.init();
    /// add line or make a query
    return 0;
}
```

2.2 Longest Increasing Subsequence (n log n)

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 100010;
int ara[MAX], b[MAX], f[MAX];
int main() {
    int n;
    cin >> n;
    int answer = 0;
    for (int i=1; i<=n; i++) {
        cin >> ara[i];
        f[i] = lower_bound(b+1, b+answer+1, ara[i]) - b;
        answer = max(answer, f[i]);
        b[f[i]] = ara[i];
    }
    printf("%d\n", answer);

```

```
    vector<int> S;

    int required = answer;
    for (int i=n; i>=1; i--)
        if (f[i]==required) {
            S.push_back(ara[i]);
            required--;
        }

    /// then print T with reversed order
    int i = S.size();
    while (i--) printf("%d ", S[i]);
    printf("\n");
}
```

2.3 Longest Increasing Subsequence Length (n log n)

```
/***
    The size of the vector after each iteration denotes the size
    of the LIS of the sub array starting at 1 and ending at i
***/

int ara[MAX];
vector <int> v;
int max_lis = 0;
for(i=1;i<=n;i++){
    x = lower_bound(all(v),ara[i])-v.begin();
    if(x==0){
        if(v.size()==0) v.pb(ara[i]);
        else v[0] = ara[i];
    }
    else if(x==v.size()) v.pb(ara[i]);
    else if(ara[i]<v[x]) v[x] = ara[i];
    max_lis = max(max_lis,(int)v.size());
}
cout << "The size of the lis is : " << max_lis << endl;
```

2.4 Number of Arrays Having Non Equal Consecutive Elements

```
/***
    * Number of arrays of size n having the first element as 1,
      the last element x and all the other elements between
      [1,k] and no two consecutive elements are equal

    * If k changes, preprocess has to be called

    * Preprocess Complexity -> O(n)
    * Query Complexity -> O(1)

    * If no bound on the first and last element,
      then there are k * (k-1)^(n-1) arrays
***/
```

```

int dp[MAX];
void preprocess(int n, int k) {
    dp[0] = 0; dp[1] = 1;
    for (int i=2;i<n;i++) {
        dp[i] = ( (k - 2) * 1LL * dp[i - 1] ) % MOD;
        dp[i] += ( (k - 1) * 1LL * dp[i - 2] ) % MOD;
        if(dp[i] >= MOD) dp[i] -= MOD;
    }
}

int howMany(int n,int k,int x) {
    if(x == 1) return ( (k - 1) * 1LL * dp[n - 2] ) % MOD;
    else return dp[n-1];
}
```

2.5 SOS DP

```
/// What SOS DP actually finds out
    for(int mask = 0;mask < (1<<N); ++mask)
        for(int i = 0;i < (1<<N); ++i)
            if( (mask & i) == i ) { /// i is a submask of mask
                F[mask] += A[i];
            }
/// O(4^n)


/// Suboptimal approach
/// iterating over all the masks
    for (int mask = 0; mask < (1<<n); mask++){
        F[mask] = A[0];
        /// iterating over all the submasks of the mask
        for(int submask = mask; submask > 0; submask = (submask-1) & mask) {
            F[mask] += A[i];
        }
    }
/// O(3^n)

/***
    S(mask,i) denotes those submasks of mask which differ
    from mask only in the least significant i+1 bits ( 0, 1, 2, ...., i --> 0
       based indexing)
    Example : S(1011010,3) = {1011010,1010010,1011000,1010000}


    if(bit i is on)
        S(mask,i) = S(mask,i-1) U S(mask ^ (1<<i),i-1)
    else
        S(mask,i) = S(mask,i-1)


    Let Sum(mask,i) denote the sum of the all A[x] where x element of S(mask,i
```

```
        )
    So, Sum(mask,N-1) will contain SOS DP result for a particular mask

    Recurrence :
    if(i'th bit is on)
        Sum(mask,i) = Sum(mask,i-1) + Sum(mask ^ (1<<i),i-1)
    else
        Sum(mask,i) = Sum(mask,i-1)
***/

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

///memory optimized version
    for(int i = 0; i<(1<<N); ++i) F[i] = A[i];
    for(int i = 0;i < N; ++i)
        for(int mask = 0; mask < (1<<N); ++mask){
            if(mask & (1<<i))
                F[mask] += F[mask^(1<<i)];
        }
/// O( N * 2^N )

/// How many pairs in ara[] such that (ara[i] & ara[j]) = 0
/// N --> Max number of bits of any array element
const int N = 20;
int inv = (1<<N) - 1;
int F[(1<<N) + 10];
int ara[MAX];

/// ara is 0 based
long long howManyZeroPairs(int n,int ara[]) {
    CLR(F);
    for(int i=0;i<n;i++) F[ara[i]]++;
    for(int i = 0;i < N; ++i)
        for(int mask = 0; mask < (1<<N); ++mask){
            if(mask & (1<<i))
                F[mask] += F[mask^(1<<i)];
        }

    long long ans = 0;
    for(int i=0;i<n;i++) ans += F[ara[i] ^ inv];
    return ans;
```

```
}


/// To get
    for(int mask = 0;mask < (1<<N); ++mask)
        for(int i = 0;i < (1<<N); ++i)
            if( (mask & i) == mask ) { /// i is a supermask of mask
                F[mask] += A[i];
            }
/// The code is the following
    for(int i = 0; i<(1<<N); ++i) F[i] = A[i];
    for(int i = 0;i < N; ++i)
        for(int mask = (1<<N)-1; mask >= 0; --mask){
            if (!(mask & (1<<i)))
                F[mask] += F[mask | (1<<i)];
        }


/// Number of subsequences of ara[0:n-1] such that
/// sub[0] & sub[2] & ... & sub[k-1] = 0
const int N = 20;
int inv = (1<<N) - 1;
int F[(1<<N) + 10];
int ara[MAX];
int p2[MAX]; /// p2[i] = 2^i
///0 based array
int howManyZeroSubSequences(int n,int ara[]) {
    CLR(F);
    for(int i=0;i<n;i++) F[ara[i]]++;
    for(int i = 0;i < N; ++i)
        for(int mask = (1<<N)-1; mask >= 0; --mask){
            if (!(mask & (1<<i)))
                F[mask] += F[mask | (1<<i)];
        }
    int ans = 0;
    for(int mask=0;mask<(1<<N);mask++) {
        if(countBit(mask) & 1) ans -= p2[F[mask]];
        else ans += p2[F[mask]];
        /// p2[F[mask]] is the count of subsets that will have the mask on at
            least
        if(ans<0) ans += MOD;
        if(ans>=MOD) ans -= MOD;
    }
    return ans;
}
```

2.6 Ways to reach cell(n,m) given some blocked cells

```
Given a grid of size N x M, where K given cells are blocked.
Find number of ways to reach (N, M) from (1, 1) if you can move right or down.
N, M <= 1e5 K <= 1e3

```

```
Explanation:
First a basic formula, number of ways to reach (x2, y2) from (x1, y1) if x2 >= x1 and y2 >= y1:
F(x1, y1, x2, y2) = (x+y)!/(x!y!) where n! denotes n factorial.

Now, an interesting observation is that if I block a cell at (i, j) all cells with their
respective coordinates greater than or equal to i and j will be affected by it.

Let's say our set S = {all blocked cells + cell(N, M)}.
I now sort S on increasing basis of x coordinate and then increasing on y.
Also I maintin an array ans where ans[i] denotes number of ways to reach
cell at index i in sorted(S). Intially ans[i] = F(1, 1, S[i].x, S[i].y).

Now, I traverse the sorted(S) in increasing order and updating the
number of ways for all the cells that it affects.

for i=0 to S.size()-2:
    for j=i+1 to S.size()-1:
        if S[j].x<S[i].x or S[j].y<S[i].y:  //cell j not affected
            continue
        //ans[i] stores current number of ways to reach that cell
        //now all paths from cell (1,1) to cell j are blocked
        //so we subtract (number of ways to reach i * number of paths from i to j)
        ans[j] -= ans[i]*F(S[i].x, S[i].y, S[j].x, S[j].y)

print ans[S.size()-1]
```

3 Data Structures

3.1 BIT

```
/***
    * 1 based
    * Initially the tree array is set to zero
    * Point Update(Adding v to index p)
    * query(id) returns sum of the range [1,id]
    * range_query(i,j) returns sum of range [i,j]

    * tree[idx] = sum of range [a,b]
      b = idx
      a = ( idx - (idx & -idx) ) + 1

***/

int tree[MAX]; /// size >= n

/// n --> size of the array
/// v --> value to be added to index idx
void update(int n,int id,int v){
```

```
    while( id<=n ) tree[id] += v , id += id & (-id);
}

/// returns sum of range[1,id]
int query(int id){
    int sum = 0;
    while(id > 0) sum += tree[id], id -= id & (-id);
    return sum;
}

/// returns sum of range[l,r]
int range_query(int l,int r){
    if(l>r) return 0;
    return query(r)-query(l-1);
}

#define LOGN 20

/// returns the lowest index id such that sum[1,id]>= v
/// if returned id > n, there is no such index then
/// The code is not tested
int tree_search(int n, int v) {

    int sum = 0, pos = 0;

    for(int i=LOGN; i>=0; i--) {
        if(pos + (1 << i) <= n and sum + tree[pos + (1 << i)] < v) {
            sum += tree[pos + (1 << i)];
            pos += (1 << i);
        }
    }
    return pos + 1;
}
```

3.2 Heavy Light Decomposition

```
/***
    * Code of Lightoj-1348 : Aladdin and the Return Journey
    * 1 based arrays and node indexing
    * construct ed
    * ara[nd] contains value on node nd
    * call HLDConstruct()
    * number of nodes n is global
    * Clear ed after the testcase
    * Per Query complexity O(n logn logn)
***/


int n;
int head[MAX];
int it , base[MAX] , pos[MAX] ;
int sub[MAX];
```

```
int ara[MAX];
vector <int> ed[MAX];

int L[MAX], P[MAX];

void dfs(int s,int par,int lev) {
    P[s] = par, L[s] = lev, sub[s] = 0;
    int sum = 1;
    for(auto &x : ed[s]) {
        if( x == par ) continue;
        dfs(x,s,lev+1);
        sum += sub[x];
        if(sub[x] > sub[ed[s][0]]  ) swap(x,ed[s][0]);
    }
    sub[s] = sum;
}

struct node{
    int sum;
} tree[4*MAX];

node Merge(node a,node b){
    node ret;
    ret.sum = a.sum + b.sum;
    return ret;
}

void build(int n,int st,int ed){
    if(st==ed){ tree[n].sum = base[st]; return; }
    int mid = (st+ed)/2;
    build(2*n,st,mid);
    build(2*n+1,mid+1,ed);
    tree[n] = Merge(tree[2*n],tree[2*n+1]);
}

void update(int n,int st,int ed,int id,int v){
    if(id>ed || id<st) return;
    if(st==ed && ed==id){ tree[n].sum = base[st] = v; return; }
    int mid = (st+ed)/2;
    update(2*n,st,mid,id,v);
    update(2*n+1,mid+1,ed,id,v);
    tree[n] = Merge(tree[2*n],tree[2*n+1]);
}

node query(int n,int st,int ed,int i,int j){
    if(st>=i && ed<=j) return tree[n];
    int mid = (st+ed)/2;
    if(mid<i) return query(2*n+1,mid+1,ed,i,j);
    else if(mid>=j) return query(2*n,st,mid,i,j);
    else return Merge(query(2*n,st,mid,i,j),query(2*n+1,mid+1,ed,i,j));
}
```

```
68
69  void HLD( int s , int hd ) {
70      pos[s] = ++it;
71      base[it] = ara[s];
72      head[s] = hd;
73      for(auto x : ed[s]) {
74          if(x == P[s]) continue;
75          HLD( x ,( x == ed[s][0] ? head[s] : x) );
76      }
77  }
78
79  void HLDConstruct(){
80      it = 0;
81      dfs(1,-1,0);
82      HLD(1,1);
83      build(1,1,n);
84  }
85
86  inline int LCA(int u,int v) {
87      while(head[u] != head[v]){
88          if(L[head[u]] < L[head[v]]) v = P[head[v]];
89          else u = P[head[u]];
90      }
91      if(L[u]<L[v]) return u;
92      else return v;
93  }
94
95  /// path from u to v ( v is an ancestor of u )
96  int call(int u,int v) {
97      int ret = 0,a,b,h;
98      while(true){
99          a = pos[v];
100         if(head[u] != head[v]) h = head[u], a = pos[h];
101         b = pos[u];
102         ret += query(1,1,n,a,b).sum;
103         if(head[u] == head[v]) return ret;
104         u = P[h];
105     }
106 }
107
108 /// returns the result of the path from node u to node v
109 int getResult(int u,int v){
110     int lca = LCA(u,v);
111     return call(u,lca) + call(v,lca) - base[pos[lca]];
112 }
113
114 /// changes the value of node nd to v
115 void updateNode(int nd, int v){
116     nd = pos[nd];
117     update(1,1,n,nd,v);
118 }
```

3.3 Maximum Sum Subarray Merging

```
tree[nd].maxsum = max(max(tree[2*nd].maxsum, tree[2*nd+1].maxsum),
                      tree[2*nd].suffixsum + tree[2*nd+1].prefixsum);

tree[nd].prefixsum = max(tree[2*nd].prefixsum,
                         tree[2*nd].sum + tree[2*nd+1].prefixsum);
tree[nd].suffixsum = max(tree[2*nd+1].suffixsum,
                         tree[2*nd+1].sum + tree[2*nd].suffixsum);

tree[nd].sum = tree[2*nd].sum + tree[2*nd+1].sum;
```

3.4 Mo's Algorithm

```
/// Complexity = nb * N + bs * Q
/// Better to keep input array 0 based

int bs;///block size
int ara[MAXN] , cnt[MAXV] , res[MAXQ];
int ans;

struct data{
    int l,r,id,bn;
    inline data() {}
    inline data(int _l, int _r, int _id){
        l = _l , r = _r , id = _id;
        bn = l / bs;
    }

    inline bool operator < (const data& other) const{
        if (bn != other.bn) return (bn < other.bn);
        return ((bn & 1) ? (r < other.r) : (r > other.r));
    }

} query[MAXQ];

void Add(int id){
    cnt[ara[id]]++;
    ///update ans
}

void Remove(int id){
    cnt[ara[id]]--;
    ///update ans
}

void Mo(int q){
    sort( query , query + q );
    int L = 0, R = 0,l,r;
    ans = 0;
    Add(0);
    for(int i=0; i<q; i++) {
```

```
        l = query[i].l;
        r = query[i].r;

        while(L>l) Add(--L);
        while(R<r) Add(++R);

        while(L<l) Remove(L++);
        while(R>r) Remove(R--);

        res[query[i].id] = ans;
    }
}
```

3.5 Monotonous Set

```
/***
    insert function inserts a pair(x,y) into the structure

    query(v) returns the maximum value y such that x <= v and
    pair(x,y) is present in the current structure
***/

struct MonotonousSet{
    set < pii > S;
    void insert(pii p){
        S.insert(p);
        auto it = S.find(p);
        if(it != S.begin()){
            auto tmp = it;
            --tmp;
            if(tmp->yy >= it->yy){
                S.erase(it);
                return;
            }
        }
        ++it;
        while(it!=S.end() && it->yy<=p.yy){
            S.erase(it);
            it = S.find(p);
            ++it;
        }
    }
    int query(int v){
        if(S.empty()) return 0;
        auto it = S.upper_bound({v,INF});
        if(it==S.begin()) return 0;
        return (--it)->second;
    }
    void clear() { S.clear(); }
};
```

3.6 PBDS

```cpp
/*** Policy Based Data Structures ***/
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including tree_order_statistics_node_update
using namespace std;
using namespace __gnu_pbds;
/// we can replace int with other data types
/// If the data type is user defined, we need to define less operator for that type
typedef tree<
    int ,
    null_type ,
    less < int > , // "less_equal<int>," for multiset
    rb_tree_tag,
    tree_order_statistics_node_update > ordered_set;
/// ordered_set has become a data type, OS is an ordered_set
ordered_set OS;
/***
    * this ordered_set is a set basically
    * ordered_set declared as above can supports all the set operations
      like insert() , erase() , find() , lower_bound() , upper_bound()

    * Ordered set supports two extra functions
        OS.find_by_order(x)
                returns the iterator to the k'th largest element starting count from 0
        OS.order_of_key(x)
                returns number of items in the set strictly smaller than x
*/

int main(){
    OS.insert(1);
    OS.insert(2);
    OS.insert(4);
    OS.insert(8);
    OS.insert(16);
    cout << ( *OS.find_by_order(0) ) << endl; /// 1
    cout << ( *OS.find_by_order(2) ) << endl; /// 4
    cout << ( *OS.find_by_order(4) ) << endl; /// 16
    cout << ( end(OS) == OS.find_by_order(5) ) <<endl; /// true
    cout << OS.order_of_key(-5) << endl;  /// 0
    cout << OS.order_of_key(3) << endl;   /// 2
    cout << OS.order_of_key(400) << endl; /// 5
    return 0;
}
```

3.7 Rope

```cpp
/***
    Problem : Given a string of length n, you will be given q queries
              Each query will contain two indexes L and R (L>=R)
```

```
4               You have to move the segment [L,R] to the beginning of the
                    string
5               *** All the indexes are zero based
6   */
7
8   #include <bits/stdc++.h>
9   #include <ext/rope>
10
11  using namespace std;
12  using namespace __gnu_cxx;
13
14  rope <char> R; ///use as usual STL container
15
16  string initial_string;
17
18  int main() {
19      int n,q;
20      cin >> n >> q;
21      cin >> initial_string;
22      for(int i=0;i<n;i++) R.push_back(initial_string[i]);
23
24      int l, r;
25      for(int i = 0; i < q; ++i) {
26          cin >> l >> r;
27          rope <char> cur = R.substr(l, r - l + 1);
28          R.erase(l, r - l + 1);
29          R.insert(R.mutable_begin(), cur);
30      }
31      for(rope <char>::iterator it = R.mutable_begin(); it != R.mutable_end();
            ++it)
32          cout << *it;
33      cout << "\n";
34      return 0;
35  }
36
37  /***
38      R.push_back(x) inserts character x at the end of rope R
39
40      R.insert(pos,nr) inserts rope nr into R at position pos
41      (the first character of nr will be in position pos)
42
43      R.erase(pos,cnt) deletes segment [pos , pos+cnt-1] from R
44
45      R.substr(pos,cnt) = segment [pos, pos+cnt-1]
46  ***/
```

3.8 STL

```
1   /*** Vector ***/
2
3       vector <int> V;
4       V.assign(n,0); /// resizes to n and makes every element 0
```

```
5
6       vector < vector <int> > V;
7       //int n = number of rows, m = number of columns;
8       M.resize(n, vector<int>(m)); // not tested
9
10      /// idx contains the index of the leftmost element in the vector which is
            greater than val
11      int idx = upper_bound( V.begin() , V.end() , val ) - V.begin();
12
13      /// idx contains the index of the leftmost element in the vector which is
            not less than val
14      int idx = lower_bound( V.begin() , V.end() , val ) - V.begin();
15
16      /// idx = V.size() if no such element in both cases
17
18      /***
19          V.begin() and V.end() are iterators
20          To get the value in a range [L,R) :
21          V.begin() should be replaced by iterator to L
22          V.end() should be replaced by iterator to R
23
24          Iterator to the elemnet at index i = ( v.begin() + i )
25      ***/
26
27      /// returns true if val is in the vector, false otherwise
28      binary_search ( V.begin() , V.end() , val )
29
30      ///merging two sorted vectors V1 and V2 to vector V
31      V.resize( V1.size() + V2.size() );
32      merge(V1.begin(),V1.end(),V2.begin(),V2.end(),V.begin());
33
34
35
36
37  /*** Priority Queue ***/
38      /// to keep the elements in ascending order
39      priority_queue < int , vector < int > , greater <int> > Q;
40
41
42
43  /*** Set ***/
44      set <int> S;
45      S.find(x) returns the iterator to the element x ( returns S.end() if x is
            not in S)
46      S.lower_bound(x) returns the iterator to the first element >= x
47      S.upper_bound(x) returns the iterator to the first element > x
48      In both cases if there is no such element, the functions returns
49      the iterator to the end of the set
50
51
52
```

```
/*** Multiset ***/

    multiset <int> S;
    S.erase(x) deletes all the occurrences of x from the set
    S.erase(it) deletes the element pointed by it
    S.erase(it1,it2) deletes the elements of range [it1,it2)
    S.find(x) returns a iterator to one of the occurrences of x ( returns X.
       end() if not present)
    S.count(x) returns the number of occurrences of x in S
    upper_bound and lower_bound is same as normal set


/*** Bitset ***/
    /// A bitset of size S
    bitset < S > B;
    /// A bitset of size S initialized with bits of 10("1010")
    bitset < S > B(10); ///...00001010
    /// A bitset of size S initialized with bits of 10("1010")
    bitset < S > B(string("1010")) ///...00001010

    bitset < S > B[MAX] /// array of bitsets each having size S
    /// to access the j'th element of the i't bitset B[i][j] is to be used

    B.set(); /// makes all the bits 1 (if no parameter given)
    B.reset(); /// makes all the bits 0 (if no parameter given)
    B.flip(i); /// flips all the bits (if no parameter given)
    B.any(); /// returns true if some bits are set
    B.count(); /// how many ones

    and, or, xor , right shift, left shift operations are also allowed in
       bitsets

    bitset < S > B1,B2,B;
    Example
    B = B1 ^ B2;
```

3.9 Suffix Tree

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

using namespace std;

const int MAX = 100010;
const int MAXC = 256;

struct SuffixTreeNode {
    struct SuffixTreeNode *children[MAXC];
    struct SuffixTreeNode *suffixLink;
    // (start,end) of node X contains the info of the edge between X
    // and it's parent P
    int start;
    int *end;
```

```
    int suffixIndex; // for leaf nodes
};

typedef struct SuffixTreeNode Node;

char text[MAX]; //Input string
Node *root = NULL; //Pointer to root node

Node *lastNewNode = NULL;
Node *activeNode = NULL;

int activeEdge = -1;
int activeLength = 0;

int remainingSuffixCount = 0;
int leafEnd = -1;
int *rootEnd = NULL;
int *splitEnd = NULL;
int size = -1; //Length of input string

Node *newNode(int start, int *end) {
    Node *node =(Node*) malloc(sizeof(Node));
    for (int i = 0; i < MAXC; i++) node->children[i] = NULL;

    /*For root node, suffixLink will be set to NULL
    For internal nodes, suffixLink will be set to root
    by default in current extension and may change in
    next extension*/
    node->suffixLink = root;
    node->start = start;
    node->end = end;

    /*suffixIndex will be set to -1 by default and
    actual suffix index will be set later for leaves
    at the end of all phases*/
    node->suffixIndex = -1;
    return node;
}

int edgeLength(Node *n) {
    return *(n->end) - (n->start) + 1;
}

int walkDown(Node *currNode) {
    if (activeLength >= edgeLength(currNode)) {
        activeEdge += edgeLength(currNode);
        activeLength -= edgeLength(currNode);
        activeNode = currNode;
        return 1;
    }
    return 0;
```

```
}

void extendSuffixTree(int pos) {
    leafEnd = pos;
    remainingSuffixCount++;
    lastNewNode = NULL;

    //Add all suffixes (yet to be added) one by one in tree
    while(remainingSuffixCount > 0) {
        if (activeLength == 0) activeEdge = pos; //APCFALZ

        if (activeNode->children[text[activeEdge]] == NULL) {
            activeNode->children[text[activeEdge]] = newNode(pos, &leafEnd);
            if (lastNewNode != NULL) {
                lastNewNode->suffixLink = activeNode;
                lastNewNode = NULL;
            }
        }
        else {
            Node *next = activeNode->children[text[activeEdge]];
            if (walkDown(next)) {//Do walkdown
                //Start from next node (the new activeNode)
                continue;
            }
            if (text[next->start + activeLength] == text[pos]) {
                if(lastNewNode != NULL && activeNode != root) {
                    lastNewNode->suffixLink = activeNode;
                    lastNewNode = NULL;
                }
                activeLength++;
                break;
            }
            splitEnd = (int*) malloc(sizeof(int));
            *splitEnd = next->start + activeLength - 1;
            Node *split = newNode(next->start, splitEnd);
            activeNode->children[text[activeEdge]] = split;
            split->children[text[pos]] = newNode(pos, &leafEnd);
            next->start += activeLength;
            split->children[text[next->start]] = next;

            if (lastNewNode != NULL) lastNewNode->suffixLink = split;

            lastNewNode = split;
        }

        remainingSuffixCount--;
        if (activeNode == root && activeLength > 0) {//APCFER2C1
            activeLength--;
            activeEdge = pos - remainingSuffixCount + 1;
        }
        else if (activeNode != root) //APCFER2C2
```

```
117             activeNode = activeNode->suffixLink;
118     }
119 }
120
121 void print(int i, int j){
122     for (int k=i; k<=j; k++) printf("%c", text[k]);
123 }
124
125 //Print the suffix tree as well along with setting suffix index
126 //So tree will be printed in DFS manner
127 //Each edge along with it's suffix index will be printed
128 void setSuffixIndexByDFS(Node *n, int labelHeight){
129     if (n == NULL) return;
130     if (n->start != -1) { //A non-root node
131         //Print the label on edge from parent to current node
132         print(n->start, *(n->end));
133     }
134     int leaf = 1;
135     for (int i = 0; i < MAXC; i++) {
136         if (n->children[i] != NULL) {
137             if (leaf == 1 && n->start != -1) printf(" [%d]\n", n->suffixIndex)
                ;
138             //Current node is not a leaf as it has outgoing edges from it.
139             leaf = 0;
140             setSuffixIndexByDFS(n->children[i], labelHeight + edgeLength(n->
                children[i]));
141         }
142     }
143     if (leaf == 1) {
144         n->suffixIndex = size - labelHeight;
145         printf(" [%d]\n", n->suffixIndex);
146     }
147 }
148
149 void freeSuffixTreeByPostOrder(Node *n) {
150     if (n == NULL) return;
151     for (int i = 0; i < MAXC; i++) {
152         if (n->children[i] != NULL) freeSuffixTreeByPostOrder(n->children[i]);
153     }
154     if (n->suffixIndex == -1) free(n->end);
155     free(n);
156 }
157
158 /*Build the suffix tree and print the edge labels along with
159 suffixIndex. suffixIndex for leaf edges will be >= 0 and
160 for non-leaf edges will be -1*/
161 void buildSuffixTree() {
162     size = strlen(text);
163     int i;
164     rootEnd = (int*) malloc(sizeof(int));
165     *rootEnd = - 1;
```

```
166
167     /*Root is a special node with start and end indices as -1,
168     as it has no parent from where an edge comes to root*/
169     root = newNode(-1, rootEnd);
170
171     activeNode = root; //First activeNode will be root
172     for (i=0; i<size; i++)
173         extendSuffixTree(i);
174     int labelHeight = 0;
175     setSuffixIndexByDFS(root, labelHeight);
176
177     //Free the dynamically allocated memory
178     freeSuffixTreeByPostOrder(root);
179 }
180
181 int main() {
182     strcpy(text, "abcabxabcd$"); buildSuffixTree();
183     return 0;
184 }
```

3.10 Segment Tree

3.10.1 How Many Non Zero Elements in the array

```
1  /***
2      Given an array consisting of all zeroes
3      Update -> Add some value to all the elements of a range
4                   (no element ever gets negative value)
5      Query -> How many non zero element in a range
6  ***/
7
8  int tree[MAX];  /// how many non zero elements in this segment
9  int lazy[MAX];  /// how many times a node is fully updated
10
11 void lazyUpdate(int n,int st,int ed){
12     if(st!=ed){
13         if(lazy[n]) tree[n] = ed-st+1;
14         else tree[n] = tree[2*n]+tree[2*n+1];
15     }
16     else{
17         if(lazy[n]) tree[n] = ed-st+1;
18         else tree[n] = 0;
19     }
20 }
21
22 void build(int n,int st,int ed){
23     lazy[n] = tree[n] = 0;
24     if(st==ed) return;
25     int mid = (st+ed)/2;
26     build(2*n,st,mid);
27     build(2*n+1,mid+1,ed);
28 }
```

```
29
30 void update(int n,int st,int ed,int i,int j,int v){
31     if(st>j || ed<i) return;
32     if(st>=i && ed<=j){
33         lazy[n] += v;
34         lazyUpdate(n,st,ed);
35         return;
36     }
37     int mid = (st+ed)/2;
38     update(2*n,st,mid,i,j,v);
39     update(2*n+1,mid+1,ed,i,j,v);
40     lazyUpdate(n,st,ed);
41 }
42
43 int query(int n,int st,int ed,int i,int j){
44     if(st>=i && ed<=j) return tree[n];
45     int mid = (st+ed)/2;
46     if(mid<i) return query(2*n+1,mid+1,ed,i,j);
47     else if(mid>=j) return query(2*n,st,mid,i,j);
48     else return query(2*n,st,mid,i,j) + query(2*n+1,mid+1,ed,i,j);
49 }
```

3.10.2 Implicit Segment Tree (Point Update, Range Query)

```
1  struct node{
2      int sum;
3      node *left,*right;
4      node(){}
5      node(int value){
6          sum = value;
7          left = right = NULL;
8      }
9  };
10
11 void update(node *cur,int st,int ed,int id,int v)
12 {
13     if(id<st || id>ed) return;
14     if(id==st && id==ed){
15         cur->sum = v;
16         return;
17     }
18     int mid = (st+ed)/2;
19     if(cur->left==NULL) cur->left = new node(0);
20     if(cur->right==NULL) cur->right = new node(0);
21     update(cur->left,st,mid,id,v);
22     update(cur->right,mid+1,ed,id,v);
23     cur->sum = cur->left->sum + cur->right->sum;
24 }
25
26 int query(node *cur,int st,int ed,int i,int j)
27 {
28     if(st>=i && ed<=j) return cur->sum;
```

```
    int mid = (st+ed)/2;
    if(cur->left==NULL) cur->left = new node(0);
    if(cur->right==NULL) cur->right = new node(0);
    if(mid<i) return query(cur->right,mid+1,ed,i,j);
    else if(mid>=j) return query(cur->left,st,mid,i,j);
    else return query(cur->right,mid+1,ed,i,j)+query(cur->left,st,mid,i,j);;
}

int main()
{
    int n = 1000000000;
    node *root = new node(0);
    update(root,1,n,5,1);
    update(root,1,n,3,1);
    cout << query(root,1,n,1,5) << endl;
    return 0;
}
```

3.10.3 Implicit Segment Tree (Range Update, Range Query)

```
struct node{
    int sum,lazy;
    node *left,*right;
    node(){}
    node(int value){
        sum = value;
        lazy = 0;
        left = right = NULL;
    }
};

void lazyUpdate(node *cur,int st,int ed)
{
    if(cur->lazy!=0){
        cur->sum += ((ed-st+1)*cur->lazy);
        if(st!=ed){
            if(cur->left==NULL) cur->left = new node(0);
            if(cur->right==NULL) cur->right = new node(0);
            cur->left->lazy += cur->lazy;
            cur->right->lazy += cur->lazy;
        }
        cur->lazy = 0;
    }
}

void update(node *cur,int st,int ed,int i,int j,int v){
    lazyUpdate(cur,st,ed);
    if(st>j || ed<i) return;
    if(st>=i && ed<=j){
        cur->lazy += v;
        lazyUpdate(cur,st,ed);
        return;
```

```
    }
    int mid = (st+ed)/2;
    if(cur->left==NULL) cur->left = new node(0);
    if(cur->right==NULL) cur->right = new node(0);
    update(cur->left,st,mid,i,j,v);
    update(cur->right,mid+1,ed,i,j,v);
    cur->sum = cur->left->sum + cur->right->sum;
}

int query(node *cur,int st,int ed,int i,int j){
    lazyUpdate(cur,st,ed);
    if(st>=i && ed<=j) return cur->sum;
    int mid = (st+ed)/2;
    if(cur->left==NULL) cur->left = new node(0);
    if(cur->right==NULL) cur->right = new node(0);
    if(mid<i) return query(cur->right,mid+1,ed,i,j);
    else if(mid>=j) return query(cur->left,st,mid,i,j);
    else return query(cur->right,mid+1,ed,i,j)+query(cur->left,st,mid,i,j);;
}

int main()
{
    int n = 1000000000;
    node *root = new node(0);
    update(root,1,n,1,5,1);
    update(root,1,n,4,10,1);
    update(root,1,n,9,14,1);
    cout << query(root,1,n,1,20) << endl;
    return 0;
}
```

3.10.4 Persistent Segment Tree (Point Update, Range Query)

```
/** Persistent Segment Tree using static Array
    Point Update , Range Sum
    Initialize ncnt to 0 in every test case **/

const int MAX = 100010;

int ncnt = 0;

struct node {
    int sum;
    int left,right;
    node() {}
    node(int val) {
        sum = val;
        left = right = -1;
    }
} tree[ ? ];

/// input araay
```

```
int ara[MAX];
/// root nodes for all versions
int version[MAX];

void build(int n,int st,int ed) {
    if (st==ed) {
        tree[n] = node(ara[st]);
        return;
    }

    int mid = (st+ed) / 2;

    tree[n].left = ++ncnt;
    tree[n].right = ++ncnt;

    build(tree[n].left, st, mid);
    build(tree[n].right, mid+1, ed);

    tree[n].sum = tree[tree[n].left].sum + tree[tree[n].right].sum;
}

void update(int prev,int cur,int st,int ed,int id, int val)
{
    if (id > ed or id < st) return;
    if (st == ed) {
        tree[cur] = node(val);
        return;
    }
    int mid = (st+ed) / 2;
    if (id <= mid) {
        tree[cur].right = tree[prev].right;
        tree[cur].left = ++ncnt;
        update(tree[prev].left,tree[cur].left, st, mid, id, val);
    }
    else {
        tree[cur].left = tree[prev].left;
        tree[cur].right = ++ncnt;
        update(tree[prev].right, tree[cur].right, mid+1, ed, id, val);
    }
    tree[cur].sum = tree[tree[cur].left].sum + tree[tree[cur].right].sum;
}

int query(int n,int st,int ed,int i,int j){
    if(st>=i && ed<=j) return tree[n].sum;
    int mid = (st+ed)/2;
    if(mid<i) return query(tree[n].right,mid+1,ed,i,j);
    else if(mid>=j) return query(tree[n].left,st,mid,i,j);
    else return query(tree[n].left,st,mid,i,j) + query(tree[n].right,mid+1,ed,
       i,j);
}

```

```
int main() {
    int n,q,l,r,k;

    sii(n,q);

    version[0] = ++ncnt;
    build(version[0],1,n);

    version[1] = ++ncnt;
    update(version[0],version[1],1,n,id,val);

    query(version[0],1,n,id,id);
    query(version[1],1,n,id,id);

    return 0;
}
```

3.10.5 Segment Tree (Point Update, Range Query)

```
int ara[MAX];

struct node{
    int sum;
}tree[4*MAX];

node Merge(node a,node b){
    node ret;
    ret.sum = a.sum+b.sum;
    return ret;
}

void build(int n,int st,int ed){
    if(st==ed){
        tree[n].sum = ara[st];
        return;
    }
    int mid = (st+ed)/2;
    build(2*n,st,mid);
    build(2*n+1,mid+1,ed);
    tree[n] = Merge(tree[2*n],tree[2*n+1]);
}

void update(int n,int st,int ed,int id,int v){
    if(id>ed || id<st) return;
    if(st==ed && ed==id){
        tree[n].sum = v;
        return;
    }
    int mid = (st+ed)/2;
    update(2*n,st,mid,id,v);
    update(2*n+1,mid+1,ed,id,v);
    tree[n] = Merge(tree[2*n],tree[2*n+1]);
```

```
}

node query(int n,int st,int ed,int i,int j){
    if(st>=i && ed<=j) return tree[n];
    int mid = (st+ed)/2;
    if(mid<i) return query(2*n+1,mid+1,ed,i,j);
    else if(mid>=j) return query(2*n,st,mid,i,j);
    else return Merge(query(2*n,st,mid,i,j),query(2*n+1,mid+1,ed,i,j));
}
```

3.10.6 Segment Tree (Range Update, Range Query)

```
int ara[MAX];

struct node{
    int sum;
}tree[4*MAX];

int lazy[4*MAX];

node Merge(node a,node b){
    node ret;
    ret.sum = a.sum+b.sum;
    return ret;
}

void lazyUpdate(int n,int st,int ed){
    if(lazy[n]!=0){
        tree[n].sum += ((ed-st+1)*lazy[n]);
        if(st!=ed){
            lazy[2*n] += lazy[n];
            lazy[2*n+1] += lazy[n];
        }
        lazy[n] = 0;
    }
}

void build(int n,int st,int ed){
    lazy[n] = 0;
    if(st==ed){
        tree[n].sum = ara[st];
        return;
    }
    int mid = (st+ed)/2;
    build(2*n,st,mid);
    build(2*n+1,mid+1,ed);
    tree[n] = Merge(tree[2*n],tree[2*n+1]);
}
void update(int n,int st,int ed,int i,int j,int v){
    lazyUpdate(n,st,ed);
    if(st>j || ed<i) return;
    if(st>=i && ed<=j){
```

```
        lazy[n] += v;
        lazyUpdate(n,st,ed);
        return;
    }
    int mid = (st+ed)/2;
    update(2*n,st,mid,i,j,v);
    update(2*n+1,mid+1,ed,i,j,v);
    tree[n] = Merge(tree[2*n],tree[2*n+1]);
}

node query(int n,int st,int ed,int i,int j){
    lazyUpdate(n,st,ed);
    if(st>=i && ed<=j) return tree[n];
    int mid = (st+ed)/2;
    if(mid<i) return query(2*n+1,mid+1,ed,i,j);
    else if(mid>=j) return query(2*n,st,mid,i,j);
    else return Merge(query(2*n,st,mid,i,j),query(2*n+1,mid+1,ed,i,j));
}
```

3.11 Treap

3.11.1 Implicit Treap

```
/**
    Treap as Interval Tree(1 based) With Insert and Remove Operation at any
       position
    The key(BST Value) is not explicitly stored and determined in the runtime.
    That's why called implicit treap
**/


typedef struct node{
    int prior,sz;
    int val; ///value stored in the array
    int sum; ///whatever info you want to maintain in segment tree for each
       node
    int lazy; ///whatever lazy update you want to do
    struct node *l,*r,*p;
} node;

typedef node* pnode;
pnode Treap;
inline int getSize(pnode t){ return t?t->sz:0; }
inline int get_sum(pnode t){ return t?t->sum:0; }

inline void lazyUpdate(pnode t){
    if(!t || !t->lazy)return;
    t->val += t->lazy;
    t->sum += t->lazy*getSize(t);
    if(t->l) t->l->lazy += t->lazy;
    if(t->r) t->r->lazy += t->lazy;
    t->lazy=0;
```

```
}

/// operation of segment tree and size,parent update
inline void operation(pnode t) {
    if(!t)return;
    lazyUpdate(t->l); lazyUpdate(t->r); ///imp:propagate lazy before combining
        t->l,t->r;
    t->sz=getSize(t->l)+1+getSize(t->r);
    t->sum = get_sum(t->l) + t->val + get_sum(t->r); /// updateing sum
    if(t->l) t->l->p = t;
    if(t->r) t->r->p = t;
}

/// The subarray[1:pos] is saved in node l, the rest in r
/// add --> Number of nodes that are not in t's subtree and has index less
   that t
void split(pnode t,pnode &l,pnode &r,int pos,int add=0){
    if(!t) return void( l = r = NULL) ;
    lazyUpdate(t);
    int curr_pos = add + getSize(t->l)+1;
    if(curr_pos<=pos) split(t->r,t->r,r,pos,curr_pos),l=t;
    else split(t->l,l,t->l,pos,add),r=t;
    operation(t);
}

void merge(pnode &t,pnode l,pnode r){
    lazyUpdate(l); lazyUpdate(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 ;
    operation(t);
}

pnode newNode(int val){
    pnode ret = (pnode)malloc(sizeof(node));
    ret->prior = rand();
    ret->sz = 1;
    ret->val = ret->sum = val;
    ret->lazy = 0;
    ret->p = ret->l = ret->r = NULL;
    return ret;
}


///changes the value of the node at position id to val
inline void point_update(pnode &t,int id,int val) {
    int sz = getSize(t->l);
    if( sz == (id-1) ) {
        t->val = val;
        pnode cur = t;
        while(cur!=NULL) operation(cur), cur = cur->p;
```

```
    }
    else if(sz < (id-1) ) point_update(t->r,id - sz - 1,val);
    else point_update(t->l,id,val);
}

/***
    * changes the value of the node at position id to val
    * Slower
    * Parent er track na rakhle use kora lagte pare
    void point_update(pnode &t,int id,int val){
        pnode L,mid,R;
        split(t,L,mid,id-1);
        split(mid,t,R,1);
        t->val = val;
        merge(mid,L,t);
        merge(t,mid,R);
    }
***/

/// deletes the node at position id
void Remove(pnode &t,int id){
    pnode L,mid,R,X;
    split(t,L,mid,id-1);
    split(mid,X,R,1);
    delete X;
    merge(t,L,R);
}

/// inserts a node at position id having array value = val
void Insert(pnode &t,int id,int val){
    pnode L,R,mid;
    pnode it = newNode(val);
    split(t,L,R,id-1);
    merge(mid,L,it);
    merge(t,mid,R);
}

/// add val to all the nodes [i:j]
void range_update(pnode t,int i,int j,int val){
    pnode L,M,R;
    split(t,L,M,i-1);
    split(M,t,R,j-i+1);
    t->lazy += val;
    merge(M,L,t);
    merge(t,M,R);
}

/// range query [i:j]
int range_query(pnode t,int i,int j){
    pnode L,M,R;
    split(t,L,M,i-1);
```

```
    split(M,t,R,j-i+1);
    int ans = t->sum;
    merge(M,L,t);
    merge(t,M,R);
    return ans;
}

/// Freeing memory after each test case
void Delete(pnode &t){
    if(!t) return;
    if(t->l) Delete(t->l);
    if(t->r) Delete(t->r);
    delete(t);
    t = NULL;
}

int ara[10];

int main(){
    ///creating a treap to use it as an interval tree of ara (1 based)
    int n = 10;
    for(int i=1; i<=n; i++){
        merge(Treap,Treap,newNode(ara[i]));
    }
    Delete(Treap);  /// Deleting when work done
    return 0;
}



/// Maximum contiguous sum merging
void operation(pnode t){
    if(!t)return;
    t->sum = get_sum(t->l) + t->val + get_sum(t->r);
    t->res = max( max(get_res(t->l), get_res(t->r)), max(0, get_rsum(t->l)) +
        t->val + max(0, get_lsum(t->r)));
    t->lsum = max(max(0,get_lsum(t->r)) + t->val + get_sum(t->l),get_lsum(t->l
        ));
    t->rsum = max(get_sum(t->r) + t->val + max(0,get_rsum(t->l)),get_rsum(t->r
        ));
}
```

3.11.2 Treap

```
/***
    * Treap is cartesian tree
    * Every node has two values(A BST value and a heap value)
    * The tree is built in such a way that if the tree is a BST WRT the BST
       values
      and also a heap WRT to heap values
    * Heap values are chosen randomly and thus the tree has height logn (
       approximate)
```

```
7
8       * If there are multiple nodes having same key, make them unique somehow
9         For example,Convert them to pair from integer
10
11  ***/
12
13  struct node{
14      int prior; /// Heap value generated randomly
15      int key; /// BST value
16      int sz; /// Subtree Size(including this node)
17      int sum; /// This bst maintains the sum of it's child nodes
18      struct node *l,*r,*p;
19  };
20
21  typedef node* pnode;
22
23  pnode Treap;
24
25  inline int getSize(pnode t) { return t?t->sz:0; }
26  inline int get_sum(pnode t) { return t?t->sum:0; }
27
28  inline void update(pnode t){
29      if(!t) return;
30      if(t->l) t->l->p = t;
31      if(t->r) t->r->p = t;
32      t->sz = getSize(t->l) + 1 + getSize(t->r);
33      t->sum = get_sum(t->l) + t->key + get_sum(t->r);
34  }
35
36  inline pnode newNode(int key){
37      pnode ret = (pnode)malloc(sizeof(node));
38      ret->sum = ret->key = key;
39      ret->sz = 1;
40      ret->prior = rand();
41      ret->p = ret->l = ret->r = NULL;
42      return ret;
43  }
44
45  /// l will contain the nodes having BST value <= key, rest will go to r
46  void split(pnode t,pnode &l,pnode &r,int key){
47      if(!t) l = r = NULL;
48      else if(t->key<=key) split(t->r,t->r,r,key) , l = t ;
49      else split(t->l,l,t->l,key) , r = t ;
50      update(t);
51  }
52
53  /// lowest value of r has to be > that largest value of l
54  void merge(pnode &t,pnode l,pnode r){
55      if(!l || !r) t = l ? l : r;
56      else if(l->prior > r->prior) merge(l->r,l->r,r), t = l ;
57      else merge(r->l,l,r->l), t = r ;
```

```
    update(t);
}

/// inserting a new node into BST
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 if(t->key<=it->key) insert(t->r,it);
    else insert(t->l,it);
    update(t);
}

/// Removing a node having BST value = key
void remove(pnode &t,int key){
    if(!t)return;
    else if( t->key == key ){
        pnode temp=t;
        merge(t,t->l,t->r);
        free(temp);
    }
    else if(t->key<key) remove(t->r,key);
    else remove(t->l,key);
    update(t);
}

/// Deleting the treap, freeing memory
void Delete(pnode &t){
    if(!t) return;
    if(t->l) Delete(t->l);
    if(t->r) Delete(t->r);
    delete(t);
    t = NULL;
}
```

3.12 Wavelet Tree

3.12.1 Wavelet Tree Extended

```
/// Actual algo pari na, Code copied
#include <bits/stdc++.h>
using namespace std;
#define pb  push_back
const int MAX = 1e5 + 10;
const int LIM = 1e6;
/***
    LIM is maximum possible value of any array element
    If array elements are not between 0 and 1e6, do array compression
    to make them so. Sum query will not work then.
    The code will have to be update to make the sum query work

    After construction, ara will be changed. Keep a copy if needed later.
***/
```

```
15
16  int a[MAX];
17  struct wavelet_tree {
18      int lo, hi;
19      wavelet_tree *l, *r;
20      vector <int> b;
21      vector <int> c; /// c holds the prefix sum of elements for sum query
22
23      ///array elements are in range [x,y]
24      ///array indices are [from, to)
25      wavelet_tree(int *from, int *to, int x, int y) {
26          lo = x, hi = y;
27          if( from >= to) return;
28          if( hi == lo ) {
29              b.reserve(to-from+1); b.pb(0);
30              c.reserve(to-from+1); c.pb(0);
31              for(auto it = from; it != to; it++) {
32                  b.pb(b.back() + 1);
33                  c.pb(c.back()+*it);
34              }
35              return;
36          }
37          int mid = (lo+hi)/2;
38          auto f = [mid](int x) { return x <= mid; };
39
40          b.reserve(to-from+1); b.pb(0);
41          c.reserve(to-from+1); c.pb(0);
42          for(auto it = from; it != to; it++) {
43              b.pb(b.back() + f(*it));
44              c.pb(c.back() + *it);
45          }
46          //see how lambda function is used here
47          auto pivot = stable_partition(from, to, f);
48          l = new wavelet_tree(from, pivot, lo, mid);
49          r = new wavelet_tree(pivot, to, mid+1, hi);
50      }
51
52      /// k'th smallest element in subarray [l,r]
53      int kth(int l, int r, int k) {
54          if(l > r) return 0;
55          if(lo == hi) return lo;
56          int inLeft = b[r] - b[l-1];
57          int lb = b[l-1]; /// amt of nos in first (l-1) nos that go in left
58          int rb = b[r]; /// amt of nos in first (r) nos that go in left
59          if(k <= inLeft) return this->l->kth(lb+1, rb, k);
60          return this->r->kth(l-lb, r-rb, k-inLeft);
61      }
62
63      /// number of elements <= k in subarray [l,r]
64      int LTE(int l, int r, int k) {
65          if(l > r or k < lo) return 0;
```

```
        if(hi <= k) return r - l + 1;
        int lb = b[l-1], rb = b[r];
        return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb, k);
    }

    /// number of occurrences of k in subarray [l,r]
    int count(int l, int r, int k) {
        if(l > r or k < lo or k > hi) return 0;
        if(lo == hi) return r - l + 1;
        int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
        if(k <= mid) return this->l->count(lb+1, rb, k);
        return this->r->count(l-lb, r-rb, k);
    }

    /// sum of the elements <= k in subarray [l,r]
    int sumk(int l, int r, int k) {
        if(l > r or k < lo) return 0;
        if(hi <= k) return c[r] - c[l-1];
        int lb = b[l-1], rb = b[r];
        return this->l->sumk(lb+1, rb, k) + this->r->sumk(l-lb, r-rb, k);
    }

    /// no need to call explicitly
    ~wavelet_tree() {
        delete l;
        delete r;
    }
};

int main() {
    int n; cin >> n;
    for(int i=1; i<=n; i++) cin >> a[i];
    wavelet_tree *Tree = new wavelet_tree(a+1, a+n+1, 1, LIM);
    return 0;
}
```

3.12.2 Wavelet Tree

```
/// Actual algo pari na, Code copied
#include <bits/stdc++.h>
using namespace std;
#define pb  push_back
const int MAX = 1e5 + 10;
const int LIM = 1e6;
/***
    LIM is maximum possible value of any array element
    If array elements are not between 0 and 1e6, do array compression
    to make them so.
    After construction, ara will be changed. Keep a copy if needed later.
***/

int ara[MAX];
```

```
15 struct wavelet_tree {
16     int lo, hi;
17     wavelet_tree *l, *r;
18     vector <int> b;
19 
20     ///array elements are in range [x,y]
21     ///array indices are [from, to)
22     wavelet_tree(int *from, int *to, int x, int y){
23         lo = x, hi = y;
24         if(lo == hi or from >= to) return;
25         int mid = (lo+hi)/2;
26         auto f = [mid](int x){ return x <= mid; };
27         b.reserve(to-from+1);
28         b.pb(0);
29         for(auto it = from; it != to; it++)
30             b.pb(b.back() + f(*it));
31         //see how lambda function is used here
32         auto pivot = stable_partition(from, to, f);
33         l = new wavelet_tree(from, pivot, lo, mid);
34         r = new wavelet_tree(pivot, to, mid+1, hi);
35     }
36 
37     /// k'th smallest element in subarray [l,r]
38     int kth(int l, int r, int k) {
39         if(l > r) return 0;
40         if(lo == hi) return lo;
41         int inLeft = b[r] - b[l-1];
42         int lb = b[l-1]; /// amt of nos in first (l-1) nos that go in left
43         int rb = b[r]; /// amt of nos in first (r) nos that go in left
44         if(k <= inLeft) return this->l->kth(lb+1, rb, k);
45         return this->r->kth(l-lb, r-rb, k-inLeft);
46     }
47 
48     /// number of elements <= k in subarray [l,r]
49     int LTE(int l, int r, int k) {
50         if(l > r or k < lo) return 0;
51         if(hi <= k) return r - l + 1;
52         int lb = b[l-1], rb = b[r];
53         return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb, k);
54     }
55 
56     /// number of occurrences of k in subarray [l,r]
57     int count(int l, int r, int k) {
58         if(l > r or k < lo or k > hi) return 0;
59         if(lo == hi) return r - l + 1;
60         int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
61         if(k <= mid) return this->l->count(lb+1, rb, k);
62         return this->r->count(l-lb, r-rb, k);
63     }
64 
65     /// no need to call explicitly
```

```
    ~wavelet_tree() {
        delete l;
        delete r;
    }
};

int main() {
    int n; cin >> n;
    for(int i=1; i<=n; i++) cin >> ara[i];
    wavelet_tree *Tree = new wavelet_tree(ara+1, ara+n+1, 1, LIM);
    return 0;
}
```

4 Graph

4.1 2-SAT

```
/***
    * 1 based index for variables
    * F = (a op b) and (c op d) and ..... (y op z)
      a, b, c ... are the variables
      sat::satisfy() returns true if there is some assignment(True/False)
      for all the variables that make F = True
    * init() at the start of every case
***/

namespace sat{
    const int MAX = 200010; /// number of variables * 2
    bool vis[MAX];
    vector <int> ed[MAX], rev[MAX];
    int n, m, ptr, dfs_t[MAX], ord[MAX], par[MAX];

    inline int inv(int x){
        return ((x) <= n ? (x + n) : (x - n));
    }

    /// Call init once
    void init(int vars){
        n = vars, m = vars << 1;
        for (int i = 1; i <= m; i++){
            ed[i].clear();
            rev[i].clear();
        }
    }

    /// Adding implication, if a then b ( a --> b )
    inline void add(int a, int b){
        ed[a].push_back(b);
        rev[b].push_back(a);
    }


```

```
36      /// (a or b) is true --> OR(a,b)
37      /// (\ACa or b) is true --> OR(inv(a),b)
38      /// (a or \ACb) is true --> OR(a,inv(b))
39      /// (\ACa or \ACb) is true --> OR(inv(a),inv(b))
40      inline void OR(int a, int b){
41          add(inv(a), b);
42          add(inv(b), a);
43      }
44
45      /// same rule as or
46      inline void AND(int a, int b){
47          add(a, b);
48          add(b, a);
49      }
50
51      /// same rule as or
52      void XOR(int a,int b){
53          add(inv(b), a);
54          add(a, inv(b));
55          add(inv(a), b);
56          add(b, inv(a));
57      }
58
59      /// same rule as or
60      inline void XNOR(int a, int b){
61          add(a,b);
62          add(b,a);
63          add(inv(a), inv(b));
64          add(inv(b), inv(a));
65      }
66
67      /// (x <= n) means forcing variable x to be true
68      /// (x = n + y) means forcing variable y to be false
69      inline void force_true(int x){
70          add(inv(x), x);
71      }
72
73      inline void topsort(int s){
74          vis[s] = true;
75          for(int x : rev[s]) if(!vis[x]) topsort(x);
76          dfs_t[s] = ++ptr;
77      }
78
79      inline void dfs(int s, int p){
80          par[s] = p;
81          vis[s] = true;
82          for(int x : ed[s]) if (!vis[x]) dfs(x, p);
83      }
84
85      void build(){
86          CLR(vis);
```

```
        ptr = 0;
        for(int i=m;i>=1;i--) {
            if (!vis[i]) topsort(i);
            ord[dfs_t[i]] = i;
        }
        CLR(vis);
        for (int i = m; i >= 1; i--){
            int x = ord[i];
            if (!vis[x]) dfs(x, x);
        }
    }

    /// Returns true if the system is 2-satisfiable and returns the solution (
       vars set to true) in vector res
    bool satisfy(vector <int>& res){
        build();
        CLR(vis);

        for (int i = 1; i <= m; i++){
            int x = ord[i];
            if (par[x] == par[inv(x)]) return false;
            if (!vis[par[x]]){
                vis[par[x]] = true;
                vis[par[inv(x)]] = false;
            }
        }
        res.clear();
        for (int i = 1; i <= n; i++){
            if (vis[par[i]]) res.push_back(i);
        }
        return true;
    }
}
```

4.2 Articulation Point

```
vector <int> edges[MAX];
bool vis[MAX] , isArt[MAX];
int st[MAX] , low[MAX] , Time = 0 , n;

void findArt(int s,int par){
    int i,x,child = 0;
    vis[s] = 1;
    Time++;
    st[s] = low[s] = Time;
    for(i=0;i< edges[s].size();i++){
        x = edges[s][i];
        if(!vis[x]){
            child++;
            findArt(x,s);
            low[s] = min(low[s],low[x]);
            if(par!=-1 && low[x]>=st[s]) isArt[s] = 1;
```

```
        }
        else{
            if(par!=x) low[s] = min(low[s],st[x]);
        }
    }
    if(par==-1 && child>1) isArt[s] = 1;
}

void processArticulation(){
    Time = 0;
    for(int i=1;i=<n;i++) if(!vis[i]) findArt(i,-1);
}
```

4.3 Bellman Ford

```
int dis[MAX];
struct data{
    int u,v,c;
} edge[MAX];

bool bellmanFord(int n,int e,int s){
    int i,j;
    for(i=1;i<=n;i++) dis[i] = INF;
    dis[s] = 0;
    for(j=1; j<=n-1; j++){
        for(i=1; i<=e; i++){
            if(dis[edge[i].u]!=INF && dis[edge[i].u]+edge[i].c<dis[edge[i].v])
                dis[edge[i].v] = dis[edge[i].u]+edge[i].c;
        }
    }
    bool negativeCycle = false;
    for(i=1; i<=e; i++){
        if(dis[edge[i].u]!=INF && dis[edge[i].u]+edge[i].c<dis[edge[i].v]){
            negativeCycle = true;
        }
    }
    return negativeCycle;
}
```

4.4 Biconnected Component

```
/***
    1 based indexing

    A graph is biconnected if every node is reachable from every other node
       even after
    removing a single node.
    Algorithm of checking Biconnectivity :
        (1) The graph is connected.
        (2) There is no articulation point in the graph.

    In the following code
```

```
    bcc_counter --> Total number of biconnected components
    bcc[i] keeps the list of nodes in the i'th BCC

    edges  should be cleared per test case

    call to prcoessBCC(n = total number of nodes) will construct bcc

 ***/

const int MAX = ?; /// maximum number of nodes

vector <int> edges[MAX];
bool vis[MAX], isArt[MAX];
int Time;
int low[MAX],st[MAX];
vector <int> bcc[MAX];
int bcc_counter;
stack <int> S;

void popBCC(int s,int x) {
    isArt[s] = 1;
    bcc[bcc_counter].pb(s);
    while(true) {
        bcc[bcc_counter].pb(S.top());
        if(S.top()==x) {
            S.pop();
            break;
        }
        S.pop();
    }
    bcc_counter++;
}

void findBCC(int s,int par) {
    S.push(s);
    int i,x,child = 0;
    vis[s] = 1;
    Time++;
    st[s] = low[s] = Time;
    for(i=0; i< edges[s].size(); i++) {
        x = edges[s][i];
        if(!vis[x]) {
            child++;
            findBCC(x,s);
            low[s] = min(low[s],low[x]);
            if(par!=-1 && low[x]>=st[s]) popBCC(s,x);
            else if(par==-1) if(child>1) popBCC(s,x);
        }
        else if(par!=x) low[s] = min(low[s],st[x]);
    }
    if(par==-1 && child>1) isArt[s] = 1;
```

```
}


/// Finds biconnected components for nodes from 1 to n
void processBCC(int n) {
    for(i=1;i<=MAX;i++)
        bcc[i].clear();

    CLR(vis); CLR(isArt);

    bcc_counter = 1;

    for(int i=1; i<=n; i++) {
        if(!vis[i]) {
            Time = 0;
            findBCC(i,-1);
            bool lala = false;
            while(!S.empty()) {
                lala = true;
                bcc[bcc_counter].push_back(S.top());
                S.pop();
            }
            if(lala) bcc_counter++;
        }
    }
    bcc_counter--;
}
```

4.5 Bridge Tree

```
/***
    1 based indexing

    call to processBridge(node,edges) generates bridge tree
    and the edge list of that is brTree

    Clear ed , isBridge , brTree per test case
***/

const int MAXN = ?;
const int MAXE = ?;

struct edges {
    int u,v;
} ara[MAXE];

vector <int> ed[MAXN]; /// actual graph
vector <int> isBridge[MAXN]; /// if the edge is a bridge, the entry will be 1
vector <int> brTree[MAXN]; /// edges of the bridge tree

bool vis[MAXN];
int st[MAXN], low[MAXN], Time = 0;
```

```
int cnum; /// number of nodes in bridge tree
int comp[MAXN];

void findBridge(int s,int par) {
    int i,x,child = 0,j;
    vis[s] = 1;
    Time++;
    st[s] = low[s] = Time;
    for(i=0; i<ed[s].size(); i++) {
        x = ed[s][i];
        if(!vis[x]) {
            child++;
            findBridge(x,s);
            low[s] = min(low[s],low[x]);
            if(low[x] > st[s]) {
                isBridge[s][i] = 1;
                j = lower_bound(ed[x].begin(),ed[x].end(),s)-ed[x].begin();
                isBridge[x][j] = 1;
            }
        }
        else if(par!=x)
            low[s] = min(low[s],st[x]);
    }
}

void dfs(int s) {
    int i,x;
    vis[s] = 1;
    comp[s] = cnum;
    for(i=0; i<ed[s].size(); i++) {
        if(!isBridge[s][i]) {
            x = ed[s][i];
            if(!vis[x]) dfs(x);
        }
    }
}

void processBridge(int n,int m) {
    CLR(vis);
    Time = 0;
    for(int i=1; i<=n; i++) if(!vis[i]) findBridge(i,-1);

    cnum = 0;
    CLR(vis);
    for(int i=1; i<=n; i++) {
        if(!vis[i]) {
            cnum++;
            dfs(i);
        }
    }

```

```
    n = cnum; ///number of nodes in the bridge tree

    for(int i=1; i<=m; i++) {
        if(comp[ara[i].u] != comp[ara[i].v]) {
            brTree[comp[ara[i].u]].pb(comp[ara[i].v]);
            brTree[comp[ara[i].v]].pb(comp[ara[i].u]);
        }
    }
}


int main() {
    int n,m,u,v;
    scanf("%d %d",&n,&m);
    for(int i=1; i<=m; i++) {
        sii(u,v);

        ed[u].pb(v);
        ed[v].pb(u);

        isBridge[u].pb(0);
        isBridge[v].pb(0);

        ara[i].u = u;
        ara[i].v = v;
    }
    for(int i=1; i<=n; i++) sort(all(ed[i]));
    processBridge(n,m);
    return 0;
}
```

4.6 Bridge

```
vector <int> ed[MAX];
vector <PII> res;
bool vis[MAX];
int st[MAX] , low[MAX] , Time = 0;

void findBridge(int s,int par){
    int i,x;
    vis[s] = 1;
    Time++;
    st[s] = low[s] = Time;
    for(int x : ed[s]){
        if(!vis[x]){
            findBridge(x,s);
            low[s] = min(low[s],low[x]);
            if(low[x]>st[s]) res.pb(mp(s,x));
        }
        else{
            if(par!=x) low[s] = min(low[s],st[x]);
```

```
        }
    }
}

void processBridge(int n){
    Time = 0;
    for(int i=1;i<=n;i++) if(!vis[i]) findBridge(i,-1);
}
```

4.7 Centroid Decomposition Offline Example

```
/***
    https://codeforces.com/contest/1156/problem/D

    You are given a tree consisting of n vertices and nâ 1 edges.
    A number is written on each edge, each number is either 0 or 1.
    Let's call an ordered pair of vertices (x,y) (xâ  y) valid if,
    while traversing the simple path from x to y,
    we never go through a 0-edge after going through a 1-edge.
    Your task is to calculate the number of valid pairs in the tree.
***/

const int MAX = 200010;

vector <pii> ed[MAX];
bool isC[MAX];
int sub[MAX];

void calc(int s,int p) {
    sub[s] = 1;
    for(pii x : ed[s]) {
        if(x.xx == p or isC[x.xx]) continue;
        calc(x.xx,s);
        sub[s] += sub[x.xx];
    }
}

int nn;

int getC(int s,int p) {
    for(pii x : ed[s]) {
        if(!isC[x.xx] and x.xx!=p and sub[x.xx]>(nn/2)) return getC(x.xx,s);
    }
    return s;
}


ll cnt0, cnt1, cnt01;
ll ans = 0;

void process(int s,int p,int zp,int op,int fg) {
    if(fg) ans++;
```

```
    if(zp and op) ans += cnt1;
    else if(zp) ans += cnt01 + cnt1 + cnt0;
    else if(op) ans += cnt1;
    else assert(false);

    for(pii x : ed[s]) {
        if(x.xx == p or isC[x.xx]) continue;
        if(zp and x.yy==1) continue;
        process(x.xx,s,zp or (x.yy==0),op or (x.yy==1),fg);
    }
}

void add(int s,int p,int zp,int op,int fg) {

    if(fg) ans++;

    if(zp and op) cnt01++;
    else if(zp) cnt0++;
    else if(op) cnt1++;
    else assert(false);

    for(pii x : ed[s]) {
        if(x.xx == p or isC[x.xx]) continue;
        if(op and x.yy==0) continue;
        add(x.xx,s,zp or (x.yy==0),op or (x.yy==1),fg);
    }
}

void decompose(int s,int p,int lev) {
    calc(s,p);
    nn = sub[s];
    int c = getC(s,p);

    cnt1 = cnt01 = cnt0 = 0;
    for(pii x : ed[c]) {
        if(isC[x.xx]) continue;
        process(x.xx,c,x.yy==0,x.yy==1,1);
        add(x.xx,c,x.yy==0,x.yy==1,1);
    }

    cnt1 = cnt01 = cnt0 = 0;
    reverse(all(ed[c]));
    for(pii x : ed[c]) {
        if(isC[x.xx]) continue;
        process(x.xx,c,x.yy==0,x.yy==1,0);
        add(x.xx,c,x.yy==0,x.yy==1,0);
    }
    isC[c] = true;
    for(pii x : ed[c]) {
        if(!isC[x.xx]) decompose(x.xx,c,lev+1);
    }
```

```
}

int main() {
//    freopen("in.txt","r",stdin);
//    freopen("out.txt","w",stdout);

    int n,a,b,c;
    si(n);
    for(int i=1;i<n;i++) {
        siii(a,b,c);
        ed[a].pb({b,c});
        ed[b].pb({a,c});
    }
    decompose(1,-1,0);
    cout << ans << endl;
    return 0;
}
```

4.8 Centroid Decomposition

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

using namespace std;

const int MAX = 100010;
const int INF = 2e9;

vector <int> ed[MAX]; /// adjacency list of the input tree
bool isCentroid[MAX]; /// if the node is already a centroid of some part
int sub[MAX], cpar[MAX], clevel[MAX];
int dis[20][MAX]; /// dis[i][j] = distance of node j from the root of the i'th
    level of decomposition

void calcSubTree(int s,int p) {
    sub[s] = 1;
    for(int x : ed[s]) {
        if(x == p or isCentroid[x]) continue;
        calcSubTree(x,s);
        sub[s] += sub[x];
    }
}

int nn; /// number of nodes in the part

int getCentroid(int s,int p) {
    for(int x : ed[s]) {
        if(!isCentroid[x] && x!=p && sub[x]>(nn/2)) return getCentroid(x,s);
    }
    return s;
}

void setDis(int s, int from, int p, int lev) {
```

```
32      dis[from][s] = lev;
33      for(int x : ed[s]) {
34          if(x == p or isCentroid[x] )  continue;
35          setDis(x, from, s, lev+1);
36      }
37  }
38
39  ///complexity --> O(nlog(n))
40  void decompose(int s,int p,int lev) {
41      calcSubTree(s,p);
42      nn = sub[s];
43      int c = getCentroid(s,p);
44      setDis(c,lev,p,0);
45
46      isCentroid[c] = true;
47      cpar[c] = p;
48      clevel[c] = lev;
49
50      for(int x : ed[c]) {
51          if(!isCentroid[x]) decompose(x,c,lev+1);
52      }
53  }
54
55  int ans[MAX];
56
57  inline void update(int v) {
58      int u = v;
59      while(u!=-1) {
60          ans[u] = min(ans[u], dis[clevel[u]][v]);
61          u = cpar[u];
62      }
63  }
64
65  inline int query(int v) {
66      int ret = INF;
67      int u = v;
68      while(u != -1) {
69          ret = min(ret, dis[clevel[u]][v]+ans[u]);
70          u = cpar[u];
71      }
72      return ret;
73  }
74  int main() {
75      decompose(1,-1,0);
76      for(int i=1; i<=n; i++) ans[i] = INF;
77      update(v);
78      query(v));
79      return 0;
80  }
```

4.9 Dijkstra

```
vector <int> ed[MAX],co[MAX];
int dis[MAX];
bool vis[MAX];

struct node{
    int city,cost;
};

bool operator < (node a,node b){return a.cost>b.cost;}

void dijkstra(int s,int n)
{
    CLR(vis);
    int i,x,u,v,c;
    node a,b;
    for(i=1;i<=n;i++) dis[i] = INF;
    dis[s] = 0;
    a = {s,0};
    priority_queue <node> q;
    q.push(a);
    while(!q.empty()){
        a = q.top();
        q.pop();
        u = a.city;
        if(!vis[u]){
            vis[u] = true;
            for(i=0;i<ed[u].size();i++){
                v = ed[u][i];
                c = co[u][i];
                if(dis[v]>dis[u]+c){
                    dis[v] = dis[u]+c;
                    b = {v,dis[v]};
                    q.push(b);
                }
            }
        }
    }
}
```

4.10 Directed Minimum Spanning Tree

```
/***
    Jaan Vai's code
    Finds cost of forming DMST
    Runs under (V^2)*log(V) where V is the number of nodes
    0 based indexing
    MM is the maximum number of nodes
    Put all the outgoing edges from u in E[u]
    Just call Find_DMST(root, number of nodes) and it will return the total
        cost of forming DMST
    if it returns inf, then initial graph was disconnected
***/
```

```
11 const int MM = ?
12 const int inf = ?
13
14 struct edge {
15     int v, w;
16     edge() {}
17     edge( int vv, int ww ) { v = vv, w = ww; }
18     bool operator < ( const edge &b ) const { return w < b.w; }
19 };
20
21 vector <edge> E[MM], inc[MM];
22 int DirectedMST( int n, int root, vector <edge> inc[MM] ) {
23     int pr[MM];
24     inc[root].clear();
25
26     /// if any node is not reachable from root, then no mst can be found
27     for( int i = 0; i < n; i++ ) {
28         sort( inc[i].begin(), inc[i].end() );
29         pr[i] = i;
30     }
31     bool cycle = true;
32     while( cycle ) {
33         cycle = false;
34         int vis[MM] = {0}, W[MM];
35         vis[root] = -1;
36         for( int i = 0, t = 1; i < n; i++, t++ ) {
37             int u = pr[i], v;
38             if( vis[u] ) continue;
39             for( v = u; !vis[v]; v = pr[inc[v][0].v] ) vis[v] = t;
40             if( vis[v] != t ) continue;
41             cycle = true;
42             int sum = 0, super = v;
43             for( ; vis[v] == t; v = pr[inc[v][0].v] ) {
44                 vis[v]++;
45                 sum += inc[v][0].w;
46             }
47             for( int j = 0; j < n; j++ ) W[j] = INT_MAX;
48             for( ; vis[v] == t + 1; v = pr[inc[v][0].v] ) {
49                 vis[v]--;
50                 for( int j = 1; j < inc[v].size(); j++ ) {
51                     int w = inc[v][j].w + sum - inc[v][0].w;
52                     W[ inc[v][j].v ] = min( W[ inc[v][j].v ], w );
53                 }
54                 pr[v] = super;
55             }
56             inc[super].clear();
57             for( int j = 0; j < n; j++ ) if( pr[j] != pr[ pr[j] ] ) pr[j] = pr
                   [ pr[j] ];
58             for( int j = 0; j < n; j++ ) if( W[j] < INT_MAX && pr[j] != super
                   ) inc[super].push_back( edge( j, W[j] ) );
59             sort( inc[super].begin(), inc[super].end() );
```

```
        }
    }
    int sum = 0;
    for( int i = 0; i < n; i++ ) if( i != root && pr[i] == i ) sum += inc[i
        ][0].w;
    return sum;
}

int Find_DMST(int root, int n) {
    bool visited[MM] = {0};
    queue <int> Q;
    for( int i = 0; i < n; i++ ) inc[i].clear();
    for( int i = 0; i < n; i++ ) for( int j = 0; j < E[i].size(); j++ ) {
        int v = E[i][j].v, w = E[i][j].w;
        inc[v].push_back( edge( i, w ) );
    }
    visited[root] = true;
    Q.push(root);
    while( !Q.empty() ) {
        int u = Q.front(); Q.pop();
        for( int i = 0; i < E[u].size(); i++ ) {
            int v = E[u][i].v;
            if( !visited[v] ) {
                visited[v] = true;
                Q.push(v);
            }
        }
    }
    /// The given graph is disconnected. So forming any MST is not possible.
    for( int i = 0; i < n; i++ ) if( !visited[i] ) return inf;
    return DirectedMST( n, root, inc );
}
```

4.11 Disjoint Set Union

```
/// Time complexity = 5*number of operations
struct DisjointSet{
    int *root,*rnk,n;
    DisjointSet(){}
    DisjointSet(int sz){
        root = new int[sz+1];
        rnk = new int[sz+1];
        n = sz;
    }
    ~DisjointSet(){
        delete[] root;
        delete[] rnk;
    }
    void init(){
        for(int i=1;i<=n;i++){
            root[i] = i;
            rnk[i] = 0;
```

```
        }
    }
    int findRoot(int u){
        if(u!=root[u]) root[u] = findRoot(root[u]);
        return root[u];
    }
    void Merge(int u,int v){
        int ru = findRoot(u); int rv = findRoot(v);
        if(rnk[ru]>rnk[rv]) root[rv] = ru;
        else root[ru] = rv;
        if(rnk[ru]==rnk[rv]) rnk[rv]++;
    }
};

int main(){
    DisjointSet *S;
    S = new DisjointSet(n);
    S->init();
    int ru = S->findRoot(u);
    S->Merge(u,v);
    delete S;

    /// or

    DisjointSet S(n);
    S.init();
    int ru = S.findRoot(u);
    S.Merge(u,v);
    return 0;
}
```

4.12 Dominator Tree

```
/***

    * A node u will be ancestor of node v in the dominator tree
      if all the the paths from source to node v contain node u
    * If a problem asks for edge disjoint paths, for every edge,
      take a new node w and  turn the  edge (u --> v) to (u --> w --> v)
      and find node disjoint path now.

    * 1 based directed graph input
    * g is the edge list of the graph you want to build dominator tree of
    * tree is the edge list of the dominator tree
      to get that we have to call the build() function
    * init() function must be called before the start of every test case

    * Only the nodes which are reachable from source will be in the dominator
       tree
      KEEP THAT IN MIND
***/

```

```
19
20  const int MAX = 200010;
21
22  vector<int> g[MAX+5],tree[MAX+5],rg[MAX+5],bucket[MAX+5];
23
24  int sdom[MAX+5],par[MAX+5],dom[MAX+5],dsu[MAX+5],label[MAX+5];
25  int arr[MAX+5],rev[MAX+5], Time ,n, source;
26
27  void init(int _n, int _source){
28      Time = 0;
29      n = _n;
30      source = _source;
31      for(int i = 1; i<=n; i++) {
32          g[i].clear(), rg[i].clear(), tree[i].clear(), bucket[i].clear();
33          arr[i] = sdom[i] = par[i] = dom[i] = dsu[i] = label[i] = rev[i] = 0;
34      }
35  }
36
37  void dfs(int u) {
38      Time++ ;
39      arr[u] = Time;
40      rev[Time] = u;
41      label[Time] = Time;
42      sdom[Time] = Time;
43      dsu[Time] = Time;
44      int i,w;
45      for(i=0; i<g[u].size(); i++) {
46          w = g[u][i];
47          if(!arr[w]) {
48              dfs(w);
49              par[arr[w]] = arr[u];
50          }
51          rg[arr[w]].push_back(arr[u]);
52      }
53  }
54
55  inline int Find(int u,int x = 0) {
56      if(u == dsu[u]) return x ? -1 : u;
57      int v = Find(dsu[u],x+1);
58      if(v<0) return u;
59      if(sdom[label[dsu[u]]] < sdom[label[u]])
60          label[u] = label[dsu[u]];
61      dsu[u] = v;
62      return x ? v : label[u];
63  }
64
65  ///Add an edge u-->v
66  inline void Union(int u,int v){
67      dsu[v]=u;
68  }
69
```

```
void build(){
    dfs(source);
    for(int i=n; i>=1; i--) {
        for(int j=0; j<rg[i].size(); j++)
            sdom[i] = min(sdom[i],sdom[Find(rg[i][j])]);
        if(i>1)bucket[sdom[i]].push_back(i);
        for(int j=0; j<bucket[i].size(); j++) {
            int w = bucket[i][j],v = Find(w);
            if(sdom[v]==sdom[w]) dom[w]=sdom[w];
            else dom[w] = v;
        }
        if(i>1) Union(par[i],i);
    }

    for(int i=2; i<=n; i++) {
        if(dom[i]!=sdom[i])dom[i]=dom[dom[i]];
        /// comment the following line out if you don't want bidirectional
           edges in dominator tree
        tree[rev[i]].push_back(rev[dom[i]]);
        tree[rev[dom[i]]].push_back(rev[i]);
    }
}
```

4.13 Dynamic Connectivity

```
/***
    Having N isolated nodes at first, there will be 3 types of queries :
    Type 1 : connect node u and node v by an edge
    Type 2 : remove the edge between node u and node v
    Type 3 : Is node u and node v in the same connected component ?
    M queries in total. No invalid removal.

    Divide the queries into sqrt(M) blocks.

    While processing block b, there will be 3 types of edges :

    Type 1 : Exists at the starting of the block and won't be removed in this
       block.
    Type 2 : Does not exist in the start of the block and won't be added
             in this block
    Type 3 : Will be added/removed in this block.

    We can ignore Type 2 edges.
    Before starting to process a block,we will build a graph G with the Type 1
        edges.
    If there are X components, we build a new graph G' with X nodes. We remove
        the
    nodes which won't be affected at all by the current block.
    Now we will start processing the block :

    We do nothing for Type1 and Type2 query.
    For type 3 query, we add the edges in G' which were added in the block and
```

```
    hasn't been removed yet and run a dfs to answer the query.

    Complexity : O( (N+M) * sqrt(N+M) )

    https://www.spoj.com/problems/DYNACON1/
    https://www.spoj.com/problems/DYNACON2/


***/
```

4.14 Floyd Warshal

```
int dis[MAX][MAX],P[MAX][MAX];
void warshall(int n){
    int i,j,k;
    for(i=0; i<n; i++)
        for(j=0; j<n; j++){
            if(dis[i][j]!=INF) P[i][j] = i;
            else P[i][j] = -1;
        }
    for(k=0; k<n; k++){
        for(i=0; i<n; i++){
            for(j=0; j<n; j++){
                if(dis[i][k]!=INF && dis[k][j]!=INF && dis[i][k]+dis[k][j]<=
                   dis[i][j]){
                    dis[i][j] = dis[i][k]+dis[k][j]; P[i][j] = k;
                }
            }
        }
    }
}
void printPath(int s,int d)
{
    if(P[s][d]==-1) puts("No Path!");
    else if(P[s][d]==s) printf("%d\n",s);
    else{
        printPath(s,P[s][d]);
        printPath(P[s][d],d);
    }
}

/*** Print d when the function returns ***/
```

4.15 MST (Kruskal)

```
struct edge{
    int u,v,c;
}ara[MAX];

bool cmp(edge a,edge b) { return a.c<b.c;}
int par[MAX];

```

```
int findParent(int u){
    if(par[u]==u) return u;
    else return par[u] = findParent(par[u]);
}
int kruskal(int n,int m){
    sort(ara+1,ara+m+1,cmp);
    int i,mst;
    mst = 0;
    for(i=1;i<=n;i++) par[i] = i;
    for(i=1;i<=m;i++){
        edge x = ara[i];
        par[x.u] = findParent(x.u);
        par[x.v] = findParent(x.v);
        if(par[x.u]!=par[x.v]){
            par[par[x.u]] = par[x.v];
            mst += x.c;
        }
    }
    return mst;
}
```

4.16 Strongly Connected Component

```
/***
    1 based indexing
    Step 1: Topsort All the nodes
    Step 2: Run DFS from the unvisited nodes in topsorted order.
            This will mark the component related to the node.
***/

vector <int> edges[MAX],trans[MAX];
int compNum[MAX];
bool vis[MAX];
int cnum;
stack <int> topSortedNodes;

void topSort(int s){
    int i,x;
    vis[s] = 1;
    for(i=0; i<edges[s].size(); i++) {
        x = edges[s][i];
        if(!vis[x]) topSort(x);
    }
    topSortedNodes.push(s);
}

void markComponent(int s)
{
    int i,x;
    vis[s] = 1;
    compNum[s] = cnum;
    for(i=0; i<trans[s].size(); i++) {
```

```
        x = trans[s][i];
        if(!vis[x]) markComponent(x);
    }
}

// finds the SCC for nodes from 1 to n
void SCC(int n) {
    int i,x;
    CLR(vis);
    for(int i=1; i<=n; i++)
        if(!vis[i]) topSort(i);

    cnum = 0;
    CLR(vis);

    while(!topSortedNodes.empty()) {
        x = topSortedNodes.top();
        topSortedNodes.pop();
        if(!vis[x]) {
            cnum++;
            markComponent(x);
        }
    }
}
```

4.17 DSU On Tree

4.17.1 DSU On Tree Using Map

```
/// n log n log n
map<int, int> *cnt[maxn];
void dfs(int v, int p){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p){
           dfs(u, v);
           if(sz[u] > mx)
               mx = sz[u], bigChild = u;
       }
    if(bigChild != -1)
        cnt[v] = cnt[bigChild];
    else
        cnt[v] = new map<int, int> ();
    (*cnt[v])[ col[v] ] ++;
    for(auto u : g[v])
       if(u != p && u != bigChild){
           for(auto x : *cnt[u])
               (*cnt[v])[x.first] += x.second;
       }
    ///now (*cnt[v])[c] is the number of vertices in subtree of vertex v that
       has color c.
    /// You can answer the queries easily.
```

```
}
```

4.17.2 DSU On Tree Using Vector

```
/// n log n
vector<int> *vec[maxn];
int cnt[maxn];
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
           mx = sz[u], bigChild = u;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           dfs(u, v, 0);
    if(bigChild != -1)
        dfs(bigChild, v, 1), vec[v] = vec[bigChild];
    else
        vec[v] = new vector<int> ();
    vec[v]->push_back(v);
    cnt[ col[v] ]++;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           for(auto x : *vec[u]){
               cnt[ col[x] ]++;
               vec[v] -> push_back(x);
           }
    /// now (*cnt[v])[c] is the number of vertices in subtree of vertex v that
        has color c. You can answer the queries easily.
    /// note that in this step *vec[v] contains all of the subtree of vertex v
       .
    if(keep == 0)
        for(auto u : *vec[v])
            cnt[ col[u] ]--;
}
```

4.17.3 DSU On Tree

```
/***
    Problem :   Given a tree, every vertex has color. Query is how many
                vertices in subtree of vertex v are colored with color c

    A call to dfs(root,-1,0) will process the answer for every query offline
    Only G needs to be cleared per case

    ********************************************************************************

    * If add() is ever called with v = -1, the whole sack becomes empty
                *
    * when the execution of add() ends. So, to maintain, any kind of min/max,
          *
    * if min/max is update, we don't need to keep track of the previous one.
           *
```

```
12      * If any value is deleted, the min/max will become +/- INF eventually.
                *
13      ********************************************************************************

14
15  ***/
16
17  const int MAX = 1e5 + 10; /// maximum number of nodes
18
19  vector <int> G[MAX]; /// adjacency list of the tree
20  int sub[MAX]; /// subtree size of a node
21  int color[MAX]; /// color of a node
22  int freq[MAX];
23  int n;
24
25  void calcSubSize(int s,int p) {
26      sub[s] = 1;
27      for(int x : G[s]) {
28          if(x==p) continue;
29          calcSubSize(x,s);
30          sub[s] += sub[x];
31      }
32  }
33
34  void add(int s,int p,int v,int bigchild = -1) {
35      freq[color[s]] += v;
36      for(int x : G[s]) {
37          if(x==p || x==bigchild) continue;
38          add(x,s,v);
39      }
40  }
41
42  void dfs(int s,int p,bool keep) {
43      int bigChild = -1;
44      for(int x : G[s]) {
45          if(x==p) continue;
46          if(bigChild==-1 || sub[bigChild] < sub[x] ) bigChild = x;
47      }
48
49      for(int x : G[s]) {
50          if(x==p || x==bigChild) continue;
51          dfs(x,s,0);
52      }
53
54      if(bigChild!=-1) dfs(bigChild,s,1);
55
56      add(s,p,1,bigChild);
57
58      /// freq[c] now contains the number of nodes in
59      /// the subtree of 'node' that have color  c
60      /// Save the answer for the queries here
```

```
    if(keep==0)
        add(s,p,-1);
}

int main() {
    input color
    construct G

    calcSubSize(root,-1);
    dfs(root,-1,0);
    return 0;
}
```

4.18 Euler Path

4.18.1 Euler Path (Directed Graph)

```
/***
    * 1 based graph input
    * Fill the edge list ed
    * Call findEuler()
***/


const int MAX = ?;

vector <int> ed[MAX+5], sltn;

int inDeg[MAX+5], outDeg[MAX+5];
bool vis[MAX+5];

void dfs(int nd) {
    vis[nd] = true; /// used to check the connectivity of the graph
    while(ed[nd].size()) {
        int v = ed[nd].back();
        ed[nd].pop_back();
        dfs(v);
    }
    sltn.pb(nd);
}

/// returns 0 if no Euler path or circuit exists
/// returns 1 if a Euler trail exists
/// returns 2 if a Euler circuit exists
int findEuler (int n) {
    int src , snk , ret = 1;
    bool found_src = false, found_snk = false;

    CLR(inDeg); CLR(outDeg);

    for(int u = 1; u <= n; u++) {
```

```
35          for(int i = 0; i<ed[u].size(); i++) {
36              int v = ed[u][i];
37              outDeg[u]++;
38              inDeg[v]++;
39          }
40      }
41
42      int diff;
43      for(int i = 1; i<=n; i++) {
44          diff = outDeg[i] - inDeg[i];
45
46          if(diff == 1) {
47              if(found_src) return 0;
48              found_src = true;
49              src = i;
50          }
51
52          else if (diff == -1) {
53              if(found_snk) return 0;
54              found_snk = true;
55              snk = i;
56          }
57
58          else if(diff != 0) return 0;
59      }
60
61      if(!found_src) {
62          /// there actually exists a euler cycle. So you need to pick a random
                node with non-zero degrees.
63          ret = 2;
64          for(int i = 1 ; i <= n ; i++) {
65              if( outDeg[i] ) {
66                  found_src = true;
67                  src = i;
68                  break;
69              }
70          }
71      }
72
73      if(!found_src) return ret; /// every node has out-degree 0
74
75      CLR(vis);
76      sltn.clear();
77      dfs(src);
78      for(int i = 1; i<=n; i++) {
79          /// the underlying graph is not even weakly connected.
80          if(outDeg[i] && !vis[i]) return 0;
81      }
82
83      /// printing path
84      for(int i = (int)sltn.size()-1; i>=0; i--) printf("%d ",sltn[i]);
```

```
    puts("");

    return ret;
}
```

4.18.2 Euler Path (Undirected Graph)

```
/***
    * 1 based graph input
    * Fill the edge list ed
    * Call findEuler()
***/

const int MAX = ?;

vector <int> ed[MAX+5], sltn;

int deg[MAX+5];
bool vis[MAX+5];

void dfs(int nd) {
    vis[nd] = true; /// used to check the connectivity of the graph
    while(ed[nd].size()) {
        int v = ed[nd].back();
        ed[nd].pop_back();
        dfs(v);
    }
    sltn.pb(nd);
}

/// returns 0 if no Euler path or circuit exists
/// returns 1 if a Euler trail exists
/// returns 2 if a Euler circuit exists
int findEuler (int n) {
    int src , snk , ret = 1;
    bool found_src = false, found_snk = false;

    CLR(deg);

    for(int u = 1; u <= n; u++) {
        for(int i = 0; i<ed[u].size(); i++) {
            int v = ed[u][i];
            deg[u]++;
            deg[v]++;
        }
    }

    for(int i = 1; i<=n; i++) {
        if( deg[i]&1 ){
            if( !found_src ) {
                found_src = true;
                src = i;
```

```
46              }
47              else if( !found_snk ) {
48                  found_snk = true;
49                  snk = i;
50              }
51              else return 0; /// more than two nodes with odd degree
52          }
53      }
54
55      if(!found_src) {
56          /// there actually exists a euler cycle. So you need to pick a random
                node with non-zero degree.
57          ret = 2;
58          for(int i = 1 ; i <= n ; i++) {
59              if( deg[i] ) {
60                  found_src = true;
61                  src = i;
62                  break;
63              }
64          }
65      }
66
67      if(!found_src) return ret; /// every node has degree 0
68
69      CLR(vis);
70      sltn.clear();
71      dfs(src);
72      for(int i = 1; i <= n ; i++) {
73          /// the underlying graph is not even weakly connected.
74          if(deg[i] && !vis[i]) return 0;
75      }
76
77      /// printing path
78      for(int i = (int)sltn.size()-1; i>=0; i--) printf("%d ",sltn[i]);
79      puts("");
80
81      return ret;
82  }
```

4.19 Flow and Matching

4.19.1 BPM (Kuhn)

```
1   /***
2       * call init at the start of every test case
3       * matchL[x] = y means node x of left side is matched to node y of right
            side
4       * matchR[y] = x means node y of right side is matched to node x of left
            side
5
6       * y is in G[x] if there is an edge between node x and node y
7         Node x is in the left and node y is in the right side
```

```
8
9       * worst case complexity V*E
10  ***/
11
12  namespace bpm{
13      const int L = 105;
14      const int R = 105;
15
16      vector <int> G[L];
17      int matchR[R], matchL[L], vis[L], it;
18
19      /// n = number of nodes in the left side
20      void init(int n) {
21          SET(matchL), SET(matchR), CLR(vis);
22          it = 1;
23          for(int i=1;i<=n;i++) G[i].clear();
24      }
25
26      inline void addEdge(int u,int v) { G[u].pb(v); }
27
28      bool dfs(int s) {
29          vis[s] = it;
30          for(auto x : G[s]) {
31              if( matchR[x] == -1 or (vis[matchR[x]] != it and dfs(matchR[x])) )
                    {
32                  matchL[s] = x; matchR[x] = s;
33                  return true;
34              }
35          }
36          return false;
37      }
38
39      int solve() {
40          int cnt = 0;
41          for(int i=1;i<=n;i++) {
42              if(dfs(i)) cnt++, it++;
43          }
44          return cnt;
45      }
46  }
```

4.19.2 MCMF (Dijkstra + Potentials)

```
1   /***
2       * 1 based node indexing
3       * call init at the start of every test case
4       * Sparse graph, amount of flow is low
5   ***/
6
7   namespace mcmf {
8       using T = int;
9       const T INF = ?; /// 0x3f3f3f3f or 0x3f3f3f3f3f3f3f3fLL
```

```
10      const int MAX = ?; /// maximum number of nodes
11
12      int n, src, snk;
13      bool vis[MAX];
14      int par[MAX], pos[MAX];
15      T pot[MAX], dis[MAX], mCap[MAX];
16      priority_queue < pair <T, int> > q;
17
18      struct Edge {
19          int to, rev_pos;
20          T cap, cost, flow;
21      };
22      vector <Edge> ed[MAX];
23
24      void init(int _n,int _src,int _snk) {
25          n = _n, src = _src, snk = _snk;
26          for(int i=1;i<=n;i++) ed[i].clear();
27      }
28
29      void addEdge(int u,int v,T cap,T cost){
30          Edge a = {v,ed[v].size(),cap,cost,0};
31          Edge b = {u,ed[u].size(),0,-cost,0};
32          ed[u].pb(a);
33          ed[v].pb(b);
34      }
35
36
37      T BellmanDP(int u) {
38          if (vis[u]) return pot[u];
39          if (u == src) {
40              pot[src] = 0; return 0;
41          }
42          vis[u] = true;
43          pot[u] = INF;
44          for (Edge e : ed[u]){
45              Edge r = ed[e.to][e.rev_pos];
46              if( r.flow < r.cap )
47                  pot[u] = min(pot[u], BellmanDP(e.to) + r.cost);
48          }
49
50          return pot[u];
51      }
52
53
54      // Dijkstra
55      bool augment() {
56          memset(vis, 0, (n + 1) * sizeof(bool));
57          for (int i=1;i<=n;i++) dis[i] = mCap[i] = INF;
58          dis[src] = 0;
59          q.push({0, src});
60
```

```
        int u, v;
        while (!q.empty()) {
            u = q.top().yy;
            q.pop();
            if (vis[u]) continue;
            vis[u] = true;

            int ptr = 0;
            for(Edge e : ed[u]) {
                v = e.to;
                T cost = e.cost + pot[u] - pot[v];
                if (e.flow < e.cap && dis[u] + cost < dis[v]) {
                    dis[v] = dis[u] + cost;
                    par[v] = u;
                    pos[v] = ptr;
                    mCap[v] = min(mCap[u],e.cap-e.flow);
                    q.push(make_pair(-dis[v], v));
                }
                ++ptr;
            }
        }
        for (int i=1;i<=n;i++) dis[i] += (pot[i] - pot[src]);
        return vis[snk];
    }

    // JohnsonDinic
    pair <T, T> solve() {
        memset(pot, 0, (n+1)*sizeof(T));
        memset(vis, 0, (n+1)*sizeof(bool));
        BellmanDP(snk);
        int u, v;
        T F = 0, C = 0, f;
        while( augment() ) {
            u = snk;
            f = mCap[snk];
            while (u != src) {
                v = par[u];
                ed[v][pos[u]].flow += mCap[snk]; /// edge of v-->u increases
                ed[u][ed[v][pos[u]].rev_pos].flow -= mCap[snk];
                u = v;
            }
            F += f;
            C += f * dis[snk];
            memcpy(pot, dis, (n + 1) * sizeof(T));
        }
        return mp(F,C);
    }
}
```

4.19.3 MCMF (spfa)

```
/***
```

```
    * 1 BASED NODE INDEXING
    * call init at the start of every test case

    * Complexity --> E*Flow (A lot less actually, not sure)

    * Maximizes the flow first, then minimizes the cost

    * The algorithm finds a path with minimum cost to send one unit of flow
      and sends flow over the path as much as possible. Then tries to find
      another path in the residual graph.

    * SPFA Technique :
        The basic idea of SPFA is the same as Bellman Ford algorithm in that
           each
        vertex is used as a candidate to relax its adjacent vertices. The
           improvement
        over the latter is that instead of trying all vertices blindly, SPFA
           maintains
        a queue of candidate vertices and adds a vertex to the queue only if
           that vertex
        is relaxed. This process repeats until no more vertex can be relaxed.
        This doesn't work if there is a negative cycle in the graph
***/

namespace mcmf {
    using T = int;
    const T INF = ?; /// 0x3f3f3f3f or 0x3f3f3f3f3f3f3f3fLL
    const int MAX = ?; /// maximum number of nodes


    int  n , src , snk;
    T dis[MAX], mCap[MAX];
    int par[MAX], pos[MAX];
    bool vis[MAX];

    struct Edge{
        int to, rev_pos;
        T cap, cost, flow;
    };

    vector <Edge> ed[MAX];

    void init(int _n,int _src,int _snk) {
        n = _n , src = _src , snk = _snk;
        for(int i=1;i<=n;i++) ed[i].clear();
    }

    void addEdge(int u,int v,int cap,int cost){
        Edge a = {v,ed[v].size(),cap,cost,0};
        Edge b = {u,ed[u].size(),0,-cost,0};
        ed[u].pb(a);
```

```
49          ed[v].pb(b);
50      }
51
52      inline bool SPFA(){
53          CLR(vis);
54          for(int i=1; i<=n; i++) mCap[i] = dis[i] = INF;
55          queue <int> q;
56          dis[src] = 0;
57          vis[src] = true; /// src is in the queue now
58          q.push(src);
59
60          while(!q.empty()){
61              int u = q.front();
62              q.pop();
63              vis[u] = false; /// u is not in the queue now
64              for(int i=0; i<ed[u].size(); i++) {
65                  Edge &e = ed[u][i];
66                  int v = e.to;
67                  if(e.cap>e.flow && dis[v]>dis[u]+e.cost){
68                      dis[v] = dis[u] + e.cost;
69                      par[v] = u;
70                      pos[v] = i;
71                      mCap[v] = min(mCap[u],e.cap-e.flow);
72                      if(!vis[v]) {
73                          vis[v] = true;
74                          q.push(v);
75                      }
76                  }
77              }
78          }
79          return (dis[snk]!=INF);
80      }
81
82      inline pair <T,T> solve() {
83          T F = 0, C = 0, f;
84          int u, v;
85          while(SPFA()){
86              u = snk;
87              f = mCap[u];
88              F += f;
89              while(u!=src){
90                  v = par[u];
91                  ed[v][pos[u]].flow += f; /// edge of v-->u increases
92                  ed[u][ed[v][pos[u]].rev_pos].flow -= f;
93                  u = v;
94              }
95              C += dis[snk] * f;
96          }
97          return mp(F,C);
98      }
99  }
```

4.19.4 MCMF (zkw)

```
/***
    * 1 based node indexing
    * call init at the start of every test case
    * works well on dense graphs
***/

namespace mcmf{
    using T = long long;
    const T INF = 200000000000000LL; /// 0x3f3f3f3f or 0x3f3f3f3f3f3f3f3fLL
    const int MAX = 4410; /// maximum number of nodes

    int n, src, snk, net[MAX], cur[MAX];
    bool vis[MAX];
    T F, C;
    T dis[MAX];

    struct Edge{
        int to;
        T cap, cost, nxt;
    };

    vector <Edge> ed;

    void init(int _n,int _src,int _snk){
        n = _n, src = _src, snk = _snk;
        memset(net,-1,(n+1) * sizeof(int));
        ed.clear();
    }

    void addEdge(int u, int v, T cap, T cost){
        ed.pb({v, cap, cost, net[u]});
        net[u] = ed.size() - 1;
        ed.pb({u, 0, -cost, net[v]});
        net[v] = ed.size() - 1;
    }

    bool modell(){
        int v;
        T mn = INF;

        for(int i=1;i<=n;i++){
            if(!vis[i]) continue;
            for(int j=net[i]; (j!=-1) and  (v=ed[j].to) ; j = ed[j].nxt){
                if(ed[j].cap){
                    if( !vis[v] and  mn > ( dis[v] - dis[i] + ed[j].cost ) ) {
                        mn = dis[v] - dis[i] + ed[j].cost;
                    }
                }
            }
```

```
50          }
51          if(mn==INF) return false;
52
53          for(int i=1; i<=n; i++){
54              if(vis[i]){
55                  cur[i] = net[i], vis[i] = false, dis[i] += mn;
56              }
57          }
58
59          return true;
60      }
61
62      T augment(int u, T flow){
63          if(u == snk){
64              C += dis[src]*flow;
65              F += flow;
66              return flow;
67          }
68          vis[u] = true;
69          for(int j=cur[u], v; (j!=-1 and (v=ed[j].to)); j=ed[j].nxt){
70              if(!ed[j].cap) continue;
71              if(vis[v] or (dis[v]+ed[j].cost)!=dis[u]){
72                  continue;
73              }
74              T delta = augment(v, min(flow, ed[j].cap));
75              if(delta){
76                  ed[j].cap -= delta;
77                  ed[j^1].cap += delta;
78                  cur[u]=j;
79                  return delta;
80              }
81          }
82          return 0;
83      }
84
85      queue <int> q;
86      void spfa(){
87          int u, v;
88
89          for(int i=1; i<=n; i++) vis[i] = false , dis[i]=INF;
90
91          dis[src]=0;
92          q.push(src);
93          vis[src] = true;
94          while(!q.empty()){
95              u = q.front(), q.pop();
96              vis[u]=false;
97              for(int i=net[u]; (i!=-1 && (v=ed[i].to)); i=ed[i].nxt){
98                  if( !ed[i].cap or dis[v] <= (dis[u]+ed[i].cost) ) continue;
99                  dis[v] = dis[u] + ed[i].cost;
100
```

```
                if(!vis[v]){
                    vis[v]=true, q.push(v);
                }
            }
        }
        for(int i=1; i<=n; i++) dis[i] = dis[snk] - dis[i];
    }

    pair <T,T> solve(){
        spfa();
        C = F = 0;
        memset(vis,0,(n+1) * sizeof(bool));
        memcpy(cur, net, (n + 1) * sizeof(int));
        do{
            while(augment(src, INF)) memset(vis,0,(n+1) * sizeof(bool));
        } while(modell());

        return {F, C};
    }
}
```

4.19.5 Maximum Flow (Dinic)

```
/***
    * 1 based indexing (preferred ...)
    * call init every test case
***/

namespace dinic {
    using T = int;
    const T INF = 0x3f3f3f3f;
    const int MAXN = 5010;

    int n, src, snk, work[MAXN];
    T dist[MAXN];

    struct Edge{
        int to, rev_pos;
        T c, f;
    };
    vector <Edge> ed[MAXN];

    void init(int _n, int _src, int _snk) {
        n = _n, src = _src, snk = _snk;
        for(int i=0;i<=n;i++) ed[i].clear();
    }

    inline void addEdge(int u, int v, T c)  {
        Edge a = {v,ed[v].size(),c,0};
        Edge b = {u,ed[u].size(),0,0};
        ed[u].pb(a);
        ed[v].pb(b);
```

```
    }

    bool dinic_bfs(){
        SET(dist);
        dist[src] = 0;
        queue <int> q;
        q.push(src);
        while(!q.empty()){
            int u = q.front();
            q.pop();
            for(Edge &e : ed[u]){
                if(dist[e.to]==-1 && e.f<e.c){
                    dist[e.to] = dist[u]+1;
                    q.push(e.to);
                }
            }
        }
        return (dist[snk]>=0);
    }

    T dinic_dfs(int u, T fl){
        if (u == snk) return fl;
        for (; work[u] < ed[u].size(); work[u]++){
            Edge &e = ed[u][work[u]];
            if (e.c <= e.f) continue;
            int v = e.to;
            if (dist[v] == dist[u] + 1){
                T df = dinic_dfs(v, min(fl, e.c - e.f));
                if (df > 0){
                    e.f += df;
                    ed[v][e.rev_pos].f -= df;
                    return df;
                }
            }
        }
        return 0;
    }
    T solve() {
        T ret = 0;
        while (dinic_bfs()){
            CLR(work);
            while (T delta = dinic_dfs(src, INF)) ret += delta;
        }
        return ret;
    }
}
```

4.19.6 Maximum Flow (Edmonds Carp)

```
/***
    * Edmonds Carp Algorithm
    * Finds Max Flow using ford fulkerson method
```

```
    * Finds path from source to sink using bfs
    * Complexity V*E*E
***/

vector <int> ed[MAX];
int cap[MAX][MAX];
int par[MAX]; ///keeps track of the parent in a path from s to d
int mCap[MAX]; ///mCap[i] keeps track edge that have minimum cost on the
    shortest path from s to i

bool getPath(int s,int d,int n){
    for(int i=0; i<=n; i++) mCap[i] = INF;
    SET(par);
    queue <int> q;
    q.push(s);
    while(!q.empty()){
        int u = q.front();
        q.pop();
        for(int i=0; i<ed[u].size(); i++){
            if(cap[u][ed[u][i]]!=0 && par[ed[u][i]]==-1){
                par[ed[u][i]] = u;
                mCap[ed[u][i]] = min(mCap[u],cap[u][ed[u][i]]);
                if(ed[u][i]==d) return true;
                q.push(ed[u][i]);
            }
        }
    }
    return false;
}

int getFlow(int s,int d,int n){
    int F = 0;
    while(getPath(s,d,n)){
        int f = mCap[d];
        F += f;
        int u = d;
        while(u!=s){
            int v = par[u];
            cap[u][v] += f;
            cap[v][u] -= f;
            u = v;
        }
    }
    return F;
}

int main(){
    int maxFlow = getFlow(s,d,n);
    return 0;
}
```

4.19.7 Weighted Matching (Hungarian Algorithm)

```
/***
    * Given a n by m matrix, a call to hungarian() returns
      minimum/maximum cost of matching
    * Complexity O(n^3), takes around 1s when n =  1000
***/

#define MAXIMIZE -1
#define MINIMIZE +1

namespace wm{
    using T = int;
    const T INF = ?;/// 0x3f3f3f3f or 0x3f3f3f3f3f3f3f3fLL
    const int MAX = ?;

    bool vis[MAX];
    int P[MAX], way[MAX], match[MAX];
    T U[MAX], V[MAX], minv[MAX], ara[MAX][MAX];

    /// n = number of row and m = number of columns in 1 based, flag =
       MAXIMIZE or MINIMIZE
    /// match[i] contains the column to which row i is matched
    T hungarian(int n, int m, T mat[MAX][MAX], int flag){
        CLR(U), CLR(V), CLR(P), CLR(ara), CLR(way);

        for (int i = 1; i <= n; i++){
            for (int j = 1; j <= m; j++){
                ara[i][j] = flag * mat[i][j];
            }
        }
        if (n > m) m = n;

        int a, b, d;
        T r, w;
        for (int i = 1; i <= n; i++){
            P[0] = i, b = 0;
            for (int j = 0; j <= m; j++) minv[j] = INF, vis[j] = false;

            do{
                vis[b] = true;
                a = P[b], d = 0, w = INF;

                for (int j = 1; j <= m; j++){
                    if (!vis[j]){
                        r = ara[a][j] - U[a] - V[j];
                        if (r < minv[j]) minv[j] = r, way[j] = b;
                        if (minv[j] < w) w = minv[j], d = j;
                    }
                }

```

```
                for (int j = 0; j <= m; j++){
                    if (vis[j]) U[P[j]] += w, V[j] -= w;
                    else minv[j] -= w;
                }
                b = d;
            } while (P[b] != 0);

            do{
                d = way[b];
                P[b] = P[d], b = d;
            } while (b != 0);
        }
        for (int j = 1; j <= m; j++) match[P[j]] = j;

        return (flag == MINIMIZE) ? -V[0] : V[0];
    }
}
```

4.20 Lowest Common Ancestor

4.20.1 Lowest Common Ancestor

```
/// 1 based indexing , n = number of nodes
const int MAX = 100010;

int lg;
int L[MAX]; /// Depth of a node
int P[MAX][20]; /// P[i][j] denotes (2^j)th parent of node i

vector <int> ed[MAX];

void dfs(int s,int par,int lev){
    int i,x;
    L[s] = lev;
    for(i=0; i<ed[s].size(); i++){
        x = ed[s][i];
        if(x!=par){
            P[x][0] = s;
            dfs(x,s,lev+1);
        }
    }
}

void lca_build(int n,int root){
    SET(P);

    dfs(root,-1,0);

    lg = (log(n)/log(2.0))+2;

    int i,j;
    for(j=1; (1<<j)<=n; j++)
```

```
        for(i=1; i<=n; i++)
            if(P[i][j-1]!=-1) P[i][j] = P[P[i][j-1]][j-1];
}

inline int lca_query(int x,int y){
    if(L[x]<L[y]) swap(x,y);
    int i,j;
    for(i=lg; i>=0; i--)
        if(L[x] - (1<<i) >= L[y]) x = P[x][i];

    if(x==y) return x;
    for(i=lg; i>=0; i--) {
        if(P[x][i]!=-1 && P[x][i]!=P[y][i]) {
            x = P[x][i];
            y = P[y][i];
        }
    }
    return P[x][0];
}
```

4.20.2 Query On a Path of a Tree

```
/***
    For sum query on the path from node u to node v,
    We need keep the nodes in an array in dfs order.

    ara[i] = val[x] if i is the starting time of node x
    ara[i] = -1*val[x] if i is the ending time of node x

    let\92s suppose p = lca(u,v).
    * Way 1
        The ranges [ st[p] , st[u] ] and [ st[p] , st[v] ] will give the
            answer.
        Here st[p] occurs twice. Needs to be handled.

    * Way 2
        if u == p : [ st[u] , st[v] ]
        else [ en[u] , st[v] ]. Here st[p] is not counted, needs to be handled
            .
***/
```

5 Math

5.1 Binomial Coefficient

```
const int MOD = 1000000007;
int inv[MAX],fact[MAX];

void precal(int N) {
    fact[0] = 1;
    for(int i=1;i<=N;i++)  fact[i] = ( (long long)fact[i-1]*i )  % MOD;
    inv[N] = bigMod(fact[N],MOD-2,MOD);
```

```
    for (int i = N - 1 ; i >= 0; i--)
        inv[i] = ( (long long)inv[i + 1]*(i + 1) ) % MOD;
}

/// returns nCr
int bin(int n,int r) {
    if(n<r) return 0;
    ll ret = fact[n];
    ret *= inv[r] , ret %= MOD;
    ret *= inv[n-r] , ret %= MOD;
    return ret;
}
```

5.2 Catalan Numbers

```
/***
    C_n =   C(2*n,n) - C(2*n,n+1)
        =   (1/(n+1)) * C(2*n,n)

    Here, C(n,r) denotes n Combination r

    7 * * * * * * * *
    6 * * * * * * * *
    5 * * * * * * * *
    4 * * * * * * * *
    3 * * * * * * * *
    2 * * * * * * * *
    1 * * * * * * * *
    0 * * * * * * * *
      0 1 2 3 4 5 6 7 (x-->)

    C_n =   number of paths from point (0,0) to point (n,n) in a n*n
            grid using only U and R moves where the path doesn't contain any
            point (x,y) where x<y

    Proof Using Reflection Technique

    Path Type 1 : Ways to go from (0,0) to (n,n)      = C(2*n,n)
    Path Type 2 : Ways to go from (0,0) to (n-1,n+1)  = C(2*n,n+1)

    There is a one-one mapping between the Type 2 paths and the invalid Type 1
        paths
    Invalid means that path violates the condition at least once

    So, Number of valid paths   =   C(2*n,n) - C(2*n,n+1)
                                =   C_n
***/
```

5.3 Discrete Logarithm (Shank's Algorithm)

```
/// returns (a^b) % m
ll bigMod(ll a,ll b,ll m){
```

```
    ll ret = 1LL;
    a %= m;
    while (b){
        if (b & 1LL) ret = (ret * a) % m;
        a = (a * a) % m;
        b >>= 1LL;
    }
    return ret;
}

PLL extEuclid(ll a,ll b) {
    if(b==0LL) return make_pair(1LL,0LL);
    PLL ret,got;
    got = extEuclid(b,a%b);
    ret = make_pair(got.yy,got.xx-(a/b)*got.yy);
    return ret;
}


/// returns modular invers of a with respect to m
/// inverse exists if and only if a and m are co-prime
ll modularInverse(ll a, ll m){
    ll x, y, inv;
    PLL sol = extEuclid(a,m);
    inv = (sol.xx + m) % m;
    return inv;
}

/***
    * returns smallest x such that (g^x) % p = h, -1 if none exists
    * p must be a PRIME ( gcd(g,p)=1 should be enough :/ )
    * function returns x, the discrete log of h with respect to g modulo p


    * g^x = h (mod p)
    * g^(mq+r) = h (mod p)
    * g^mq * g^r = h mod(p)
    * g^r = h * ((g^-1)^m)^q (mod p)

    * we will precompute all possible (g^r % p) and store the values in a map
      (value-->r)
    * for every q from 0 to m, we will find corresponding r in a map

***/
ll discrete_log(ll g, ll h, ll p){
    if (h >= p) return -1LL;
    if ( ( g % p) == 0LL ){
        /// return -1 if strictly positive integer solution is required
        if ( h == 1LL ) return 0;
        else return -1;
    }
```

```
53
54      unordered_map <ll, ll> mp;
55      ll i, q, r, m = ceil(sqrt(p));
56      ll d = 1LL, inv = bigMod(modularInverse(g, p), m, p);
57
58      for (r = 0; r <= m; r++){
59          if (mp.find(d)!=mp.end()) mp[d] = r ;
60          d *= g;
61          if (d >= p) d %= p;
62      }
63
64      d = h;
65      for (q = 0; q <= m; q++){
66          if(mp.find(d)!=mp.end()) {
67              r = mp[d];
68              return (m * q) + r;
69          }
70          d *= inv;
71          if (d >= p) d %= p;
72      }
73      return -1LL;
74  }
```

5.4 Enumeration of Partitions

```
1   public class Partitions {
2       public static boolean nextPartition(List<Integer> p) {
3           int n = p.size();
4           if (n <= 1)
5               return false;
6           int s = p.remove(n - 1) - 1;
7           int i = n - 2;
8           while (i > 0 && p.get(i).equals(p.get(i - 1))) {
9               s += p.remove(i);
10              --i;
11          }
12          p.set(i, p.get(i) + 1);
13          while (s-- > 0) {
14              p.add(1);
15          }
16          return true;
17      }
18
19      public static List<Integer> partitionByNumber(int n, long number) {
20          List<Integer> p = new ArrayList<>();
21          for (int x = n; x > 0; ) {
22              int j = 1;
23              while (true) {
24                  long cnt = partitionFunction(x)[x][j];
25                  if (number < cnt)
26                      break;
27                  number -= cnt;
```

```
                ++j;
            }
            p.add(j);
            x -= j;
        }
        return p;
    }

    public static long numberByPartition(List<Integer> p) {
        long res = 0;
        int sum = 0;
        for (int x : p) {
            sum += x;
        }
        for (int cur : p) {
            for (int j = 0; j < cur; j++) {
                res += partitionFunction(sum)[sum][j];
            }
            sum -= cur;
        }
        return res;
    }

    public static void generateIncreasingPartitions(int[] p, int left, int
       last, int pos) {
        if (left == 0) {
            for (int i = 0; i < pos; i++)
                System.out.print(p[i] + " ");
            System.out.println();
            return;
        }
        for (p[pos] = last + 1; p[pos] <= left; p[pos]++)
            generateIncreasingPartitions(p, left - p[pos], p[pos], pos + 1);
    }

    public static long countPartitions(int n) {
        long[] p = new long[n + 1];
        p[0] = 1;
        for (int i = 1; i <= n; i++) {
            for (int j = i; j <= n; j++) {
                p[j] += p[j - i];
            }
        }
        return p[n];
    }
    public static long[][] partitionFunction(int n) {
        long[][] p = new long[n + 1][n + 1];
        p[0][0] = 1;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                p[i][j] = p[i - 1][j - 1] + p[i - j][j];
```

```
            }
        }
        return p;
    }

    public static long[][] partitionFunction2(int n) {
        long[][] p = new long[n + 1][n + 1];
        p[0][0] = 1;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                for (int k = 0; k <= j; k++) {
                    p[i][j] += p[i - j][k];
                }
            }
        }
        return p;
    }

    // Usage example
    public static void main(String[] args) {
        System.out.println(7 == countPartitions(5));
        System.out.println(627 == countPartitions(20));
        System.out.println(5604 == countPartitions(30));
        System.out.println(204226 == countPartitions(50));
        System.out.println(190569292 == countPartitions(100));

        List<Integer> p = new ArrayList<>();
        Collections.addAll(p, 1, 1, 1, 1, 1);
        do {
            System.out.println(p);
        }
        while (nextPartition(p));

        int[] p1 = new int[8];
        generateIncreasingPartitions(p1, p1.length, 0, 0);

        List<Integer> list = partitionByNumber(5, 6);
        System.out.println(list);

        System.out.println(numberByPartition(list));
    }
}
```

5.5 Enumeration of Permutations

```
import java.util.*;

public class Permutations
{

    public static boolean nextPermutation(int[] p)
    {
```

```
        for (int a = p.length - 2; a >= 0; --a)
            if (p[a] < p[a + 1])
                for (int b = p.length - 1; ; --b)
                    if (p[b] > p[a])
                    {
                        int t = p[a];
                        p[a] = p[b];
                        p[b] = t;
                        for (++a, b = p.length - 1; a < b; ++a, --b)
                        {
                            t = p[a];
                            p[a] = p[b];
                            p[b] = t;
                        }
                        return true;
                    }
        return false;
    }

    public static int[] permutationByNumber(int n, long number)
    {
        long[] fact = new long[n];
        fact[0] = 1;
        for (int i = 1; i < n; i++)
        {
            fact[i] = i * fact[i - 1];
        }
        int[] p = new int[n];
        int[] free = new int[n];
        for (int i = 0; i < n; i++)
        {
            free[i] = i;
        }
        for (int i = 0; i < n; i++)
        {
            int pos = (int) (number / fact[n - 1 - i]);
            p[i] = free[pos];
            System.arraycopy(free, pos + 1, free, pos, n - 1 - pos);
            number %= fact[n - 1 - i];
        }
        return p;
    }

    public static long numberByPermutation(int[] p)
    {
        int n = p.length;
        long[] fact = new long[n];
        fact[0] = 1;
        for (int i = 1; i < n; i++)
        {
            fact[i] = i * fact[i - 1];
```

```
59          }
60          long res = 0;
61          for (int i = 0; i < n; i++)
62          {
63              int a = p[i];
64              for (int j = 0; j < i; j++)
65              {
66                  if (p[j] < p[i])
67                  {
68                      --a;
69                  }
70              }
71              res += a * fact[n - 1 - i];
72          }
73          return res;
74      }
75
76      public static void generatePermutations(int[] p, int depth)
77      {
78          int n = p.length;
79          if (depth == n)
80          {
81              System.out.println(Arrays.toString(p));
82              return;
83          }
84          for (int i = 0; i < n; i++)
85          {
86              if (p[i] == 0)
87              {
88                  p[i] = depth;
89                  generatePermutations(p, depth + 1);
90                  p[i] = 0;
91              }
92          }
93      }
94      public static long nextPermutation(long x) {
95          long s = x & -x;
96          long r = x + s;
97          long ones = x ^ r;
98          ones = (ones >> 2) / s;
99          return r | ones;
100     }
101
102     public static List<List<Integer>> decomposeIntoCycles(int[] p) {
103         int n = p.length;
104         boolean[] vis = new boolean[n];
105         List<List<Integer>> res = new ArrayList<>();
106         for (int i = 0; i < n; i++)
107         {
108             if (vis[i])
109                 continue;
```

```
            int j = i;
            List<Integer> cur = new ArrayList<>();
            do
            {
                cur.add(j);
                vis[j] = true;
                j = p[j];
            }
            while (j != i);
            res.add(cur);
        }
        return res;
    }

    // Usage example
    public static void main(String[] args) {
        // print all permutations method 1
        generatePermutations(new int[2], 1);

        // print all permutations method 2
        int[] p = {0, 1, 2};
        int cnt = 0;
        do
        {
            System.out.println(Arrays.toString(p));
            if (!Arrays.equals(p, permutationByNumber(p.length,
                numberByPermutation(p))) ||
                    cnt != numberByPermutation(permutationByNumber(p.length,
                        cnt)))
                throw new RuntimeException();
            ++cnt;
        }
        while (nextPermutation(p));

        System.out.println(5 == numberByPermutation(p));
        System.out.println(Arrays.equals(new int[] {1, 0, 2},
            permutationByNumber(3, 2)));

        System.out.println(0b1101 == nextPermutation(0b1011));
        System.out.println(decomposeIntoCycles(new int[] {0, 2, 1, 3}));
    }
}
```

5.6 Extended Euclid

```
/***
    c = gcd(a,b);

    ax + by = c;
    (bq + r)x + by = c;
    bqx + rx + by = c;
    b(qx + y) + rx = c;
```

```
    bx' + ry' = c;       [r = a % b]

    We get,
    x' = qx + y;
    y' = x

    So,
    y = x' - qx;
    y = x' - qy';        [y' = x]
    and
    x = y'

    If c is not the gcd then,

    actual x = x * (c/gcd)
    actual y = y * (c/gcd)
    But if gcd doesn't divide c, there is no solution.
***/


/// returns (x,y) for ax + by = gcd(a,b)
/// keep in mind that if a or b or both are negative,  gcd(a,b) will be
   negative

PLL extEuclid(ll a,ll b)
{
    if(b==0LL) return mp(1LL,0LL);
    PLL ret,got;
    got = extEuclid(b,a%b);
    ret = mp(got.yy,got.xx-(a/b)*got.yy);
    return ret;
}

/***
    From one solution (x0,y0), we can obtain all the solutions of the given
       equation.
    Let g = gcd(a,b) and let x0,y0 be integers which satisfy the following:
    a*x0+b*y0 = c
    Now, we should see that adding b/g to x0 and at the same time subtracting
       a/g
    from y0 will not break the equality:

      a*(x0 + b/g) + b*(y0 - a/g)
    = a*x0 + b*y0 + (a*b)/g - (b*a)/g
    = c
    Obviously, this process can be repeated again, so all the numbers of the
       form:

    x = x0 + k * (b/g)
    y = y0 - k * (a/g)
    are solutions of the given Diophantine equation.
```

```
55
56      In the solution returned by extEuclid :
57      |x| and |y| is minimized
58      |x| <= b/2g
59      |y| <= a/2g
60      Because we get a new x after every b/g amount of jump
61      and we get a new y after every a/g amount of jump
62
63      Solution with minimum (x+y):
64      x + y = x0 + y0 + k*(b/g - a/g)
65      x + y = x0 + y0 + k*((b-a)/g)
66
67      If b>a, we need to find the k with the minimum value
68      else we need to find the k with the maximum value
69  ***/
70
71  /// Iterative Implementation
72  PLL extEuclid(ll a,ll b){
73      ll s = 1,t = 0,st = 0,tt = 1;
74      while(b) {
75          s = s - (a/b)*st;
76          swap(s,st);
77          t = t - (a/b)*tt;
78          swap(t,tt);
79          a = a % b;
80          swap(a,b);
81      }
82      return mp(s,t);
83  }
84  /// returns number of solutions for the equation ax + by = c
85  /// where minx <= x <= maxx and miny <= y <= maxy
86  ll numberOfSolutions(ll a,ll b,ll c,ll minx,ll maxx,ll miny,ll maxy)
87  {
88      if(a==0 && b==0){
89          if(c!=0) return 0;
90          else return (maxx-minx+1)*(maxy-miny+1); /// all possible (x,y) within
                the ranges can be a solution
91      }
92
93      ll gcd = __gcd(a,b);
94      if(c%gcd!=0) return 0;/// no solution , gcd(a,b) doesn't divide c
95
96      /// If b==0, x will be fixed, any y in the range can form a pair with that
            x
97      if(b==0){
98          c /= a;
99          if(c>=minx && c<=maxx) return maxy-miny+1;
100         else return 0;
101     }
102
103     /// If a==0, x will be fixed, any x in the range can form a pair with that
```

```
        y
    if(a==0){
        c /= b;
        if(c>=miny && c<=maxy) return maxx-minx+1;
        else return 0;
    }

    /// gives a particular solution to the equation ax + by = gcd(a,b) {gcd(a,
       b) can be negative also}
    PLL sol = extEuclid(a,b);

    a /= gcd;
    b /= gcd;
    c /= gcd;

    ll x,y;
    x = sol.xx*c;
    y = sol.yy*c;

    ll lx,ly,rx,ry;

    /// lx -> minimum value of k such that sol.xx + k * (b/g) is in range[minx
       ,maxx]
    /// rx -> maximum value of k such that sol.xx + k * (b/g) is in range[minx
       ,maxx]
    if(x<minx) lx = ceil( (minx-x) / (double)abs(b) );
    else lx = -floor( (x-minx) / (double)abs(b) );

    if(x<maxx) rx = floor((maxx-x) / (double)abs(b) );
    else rx = -ceil((x-maxx) / (double)abs(b) );

    /// Doing this I because I ignored sign of b before passing to getCeil/
       getFloor
    if(b<0){
        lx *= -1;
        rx *= -1;
        swap(lx,rx);
    }
    if(lx>rx) return 0;

    /// ly -> minimum value of k such that sol.yy - k * (a/g) is in range[miny
       ,maxy]
    /// ry -> maximum value of k such that sol.yy - k * (a/g) is in range[miny
       ,maxy]
    if(y<miny) ly = ceil( (miny-y) / (double)abs(a) );
    else ly = -floor( (y-miny) / (double)abs(a) );

    if(y<maxy) ry = floor( (maxy-y) / (double)abs(a) );
    else ry = -ceil( (y-maxy) / (double)abs(a) );

    /// Doing this because I ignored sign of a before passing to getCeil/
```

```
        getFloor
    if(a<0){
        ly *= -1;
        ry *= -1;
        swap(ly,ry);
    }
    if(ly>ry) return 0;

    ly *= -1;
    ry *= -1;
    swap(ly,ry);

    /// getting the intersection between (x range) and (y range) of k
    ll li = max(lx,ly);
    ll ri = min(rx,ry);

    return max( ri - li + 1 , 0LL );
}
```

5.7 Highly Composite Numbers

```
/***

    Number of highly composite numbers less than 1000000000000000000 is 156

    number                  divisors    factorization
    1                       1
    2                       2           2
    4                       3           2^2
    6                       4                 2*3
    12                      6           2^2*3
    24                      8           2^3*3
    36                      9           2^2*3^2
    48                      10          2^4*3
    60                      12          2^2*3*5
    120                     16          2^3*3*5
    180                     18          2^2*3^2*5
    240                     20          2^4*3*5
    360                     24          2^3*3^2*5
    720                     30          2^4*3^2*5
    840                     32          2^3*3*5*7
    1260                    36          2^2*3^2*5*7
    1680                    40          2^4*3*5*7
    2520                    48          2^3*3^2*5*7
    5040                    60          2^4*3^2*5*7
    7560                    64          2^3*3^3*5*7
    10080                   72          2^5*3^2*5*7
    15120                   80          2^4*3^3*5*7
    20160                   84          2^6*3^2*5*7
    25200                   90          2^4*3^2*5^2*7
    27720                   96          2^3*3^2*5*7*11
    45360                   100         2^4*3^4*5*7
```

50400	108	2^5*3^2*5^2*7
55440	120	2^4*3^2*5*7*11
83160	128	2^3*3^3*5*7*11
110880	144	2^5*3^2*5*7*11
166320	160	2^4*3^3*5*7*11
221760	168	2^6*3^2*5*7*11
277200	180	2^4*3^2*5^2*7*11
332640	192	2^5*3^3*5*7*11
498960	200	2^4*3^4*5*7*11
554400	216	2^5*3^2*5^2*7*11
665280	224	2^6*3^3*5*7*11
720720	240	2^4*3^2*5*7*11*13
1081080	256	2^3*3^3*5*7*11*13
1441440	288	2^5*3^2*5*7*11*13
2162160	320	2^4*3^3*5*7*11*13
2882880	336	2^6*3^2*5*7*11*13
3603600	360	2^4*3^2*5^2*7*11*13
4324320	384	2^5*3^3*5*7*11*13
6486480	400	2^4*3^4*5*7*11*13
7207200	432	2^5*3^2*5^2*7*11*13
8648640	448	2^6*3^3*5*7*11*13
10810800	480	2^4*3^3*5^2*7*11*13
14414400	504	2^6*3^2*5^2*7*11*13
17297280	512	2^7*3^3*5*7*11*13
21621600	576	2^5*3^3*5^2*7*11*13
32432400	600	2^4*3^4*5^2*7*11*13
36756720	640	2^4*3^3*5*7*11*13*17
43243200	672	2^6*3^3*5^2*7*11*13
61261200	720	2^4*3^2*5^2*7*11*13*17
73513440	768	2^5*3^3*5*7*11*13*17
110270160	800	2^4*3^4*5*7*11*13*17
122522400	864	2^5*3^2*5^2*7*11*13*17
147026880	896	2^6*3^3*5*7*11*13*17
183783600	960	2^4*3^3*5^2*7*11*13*17
245044800	1008	2^6*3^2*5^2*7*11*13*17
294053760	1024	2^7*3^3*5*7*11*13*17
367567200	1152	2^5*3^3*5^2*7*11*13*17
551350800	1200	2^4*3^4*5^2*7*11*13*17
698377680	1280	2^4*3^3*5*7*11*13*17*19
735134400	1344	2^6*3^3*5^2*7*11*13*17
1102701600	1440	2^5*3^4*5^2*7*11*13*17
1396755360	1536	2^5*3^3*5*7*11*13*17*19
2095133040	1600	2^4*3^4*5*7*11*13*17*19
2205403200	1680	2^6*3^4*5^2*7*11*13*17
2327925600	1728	2^5*3^2*5^2*7*11*13*17*19
2793510720	1792	2^6*3^3*5*7*11*13*17*19
3491888400	1920	2^4*3^3*5^2*7*11*13*17*19
4655851200	2016	2^6*3^2*5^2*7*11*13*17*19
5587021440	2048	2^7*3^3*5*7*11*13*17*19
6983776800	2304	2^5*3^3*5^2*7*11*13*17*19
10475665200	2400	2^4*3^4*5^2*7*11*13*17*19

83	13967553600	2688	2^6*3^3*5^2*7*11*13*17*19
84	20951330400	2880	2^5*3^4*5^2*7*11*13*17*19
85	27935107200	3072	2^7*3^3*5^2*7*11*13*17*19
86	41902660800	3360	2^6*3^4*5^2*7*11*13*17*19
87	48886437600	3456	2^5*3^3*5^2*7^2*11*13*17*19
88	64250746560	3584	2^6*3^3*5*7*11*13*17*19*23
89	73329656400	3600	2^4*3^4*5^2*7^2*11*13*17*19
90	80313433200	3840	2^4*3^3*5^2*7*11*13*17*19*23
91	97772875200	4032	2^6*3^3*5^2*7^2*11*13*17*19
92	128501493120	4096	2^7*3^3*5*7*11*13*17*19*23
93	146659312800	4320	2^5*3^4*5^2*7^2*11*13*17*19
94	160626866400	4608	2^5*3^3*5^2*7*11*13*17*19*23
95	240940299600	4800	2^4*3^4*5^2*7*11*13*17*19*23
96	293318625600	5040	2^6*3^4*5^2*7^2*11*13*17*19
97	321253732800	5376	2^6*3^3*5^2*7*11*13*17*19*23
98	481880599200	5760	2^5*3^4*5^2*7*11*13*17*19*23
99	642507465600	6144	2^7*3^3*5^2*7*11*13*17*19*23
100	963761198400	6720	2^6*3^4*5^2*7*11*13*17*19*23
101	1124388064800	6912	2^5*3^3*5^2*7^2*11*13*17*19*23
102	1606268664000	7168	2^6*3^3*5^3*7*11*13*17*19*23
103	1686582097200	7200	2^4*3^4*5^2*7^2*11*13*17*19*23
104	1927522396800	7680	2^7*3^4*5^2*7*11*13*17*19*23
105	2248776129600	8064	2^6*3^3*5^2*7^2*11*13*17*19*23
106	3212537328000	8192	2^7*3^3*5^3*7*11*13*17*19*23
107	3373164194400	8640	2^5*3^4*5^2*7^2*11*13*17*19*23
108	4497552259200	9216	2^7*3^3*5^2*7^2*11*13*17*19*23
109	6746328388800	10080	2^6*3^4*5^2*7^2*11*13*17*19*23
110	8995104518400	10368	2^8*3^3*5^2*7^2*11*13*17*19*23
111	9316358251200	10752	2^6*3^3*5^2*7*11*13*17*19*23*29
112	13492656777600	11520	2^7*3^4*5^2*7^2*11*13*17*19*23
113	18632716502400	12288	2^7*3^3*5^2*7*11*13*17*19*23*29
114	26985313555200	12960	2^8*3^4*5^2*7^2*11*13*17*19*23
115	27949074753600	13440	2^6*3^4*5^2*7*11*13*17*19*23*29
116	32607253879200	13824	2^5*3^3*5^2*7^2*11*13*17*19*23*29
117	46581791256000	14336	2^6*3^3*5^3*7*11*13*17*19*23*29
118	48910880818800	14400	2^4*3^4*5^2*7^2*11*13*17*19*23*29
119	55898149507200	15360	2^7*3^4*5^2*7*11*13*17*19*23*29
120	65214507758400	16128	2^6*3^3*5^2*7^2*11*13*17*19*23*29
121	93163582512000	16384	2^7*3^3*5^3*7*11*13*17*19*23*29
122	97821761637600	17280	2^5*3^4*5^2*7^2*11*13*17*19*23*29
123	130429015516800	18432	2^7*3^3*5^2*7^2*11*13*17*19*23*29
124	195643523275200	20160	2^6*3^4*5^2*7^2*11*13*17*19*23*29
125	260858031033600	20736	2^8*3^3*5^2*7^2*11*13*17*19*23*29
126	288807105787200	21504	2^6*3^3*5^2*7*11*13*17*19*23*29*31
127	391287046550400	23040	2^7*3^4*5^2*7^2*11*13*17*19*23*29
128	577614211574400	24576	2^7*3^3*5^2*7*11*13*17*19*23*29*31
129	782574093100800	25920	2^8*3^4*5^2*7^2*11*13*17*19*23*29
130	866421317361600	26880	2^6*3^4*5^2*7*11*13*17*19*23*29*31
131	1010824870255200	27648	2^5*3^3*5^2*7^2*11*13*17*19*23*29*31
132	1444035528936000	28672	2^6*3^3*5^3*7*11*13*17*19*23*29*31
133	1516237305382800	28800	2^4*3^4*5^2*7^2*11*13*17*19*23*29*31

```
    1732842634723200          30720      2^7*3^4*5^2*7*11*13*17*19*23*29*31
    2021649740510400          32256      2^6*3^3*5^2*7^2*11*13*17*19*23*29*31
    2888071057872000          32768      2^7*3^3*5^3*7*11*13*17*19*23*29*31
    3032474610765600          34560      2^5*3^4*5^2*7^2*11*13*17*19*23*29*31
    4043299481020800          36864      2^7*3^3*5^2*7^2*11*13*17*19*23*29*31
    6064949221531200          40320      2^6*3^4*5^2*7^2*11*13*17*19*23*29*31
    8086598962041600          41472      2^8*3^3*5^2*7^2*11*13*17*19*23*29*31
    10108248702552000         43008      2^6*3^3*5^3*7^2*11*13*17*19*23*29*31
    12129898443062400         46080      2^7*3^4*5^2*7^2*11*13*17*19*23*29*31
    18194847664593600         48384      2^6*3^5*5^2*7^2*11*13*17*19*23*29*31
    20216497405104000         49152      2^7*3^3*5^3*7^2*11*13*17*19*23*29*31
    24259796886124800         51840      2^8*3^4*5^2*7^2*11*13*17*19*23*29*31
    30324746107656000         53760      2^6*3^4*5^3*7^2*11*13*17*19*23*29*31
    36389695329187200         55296      2^7*3^5*5^2*7^2*11*13*17*19*23*29*31
    48519593772249600         57600      2^9*3^4*5^2*7^2*11*13*17*19*23*29*31
    60649492215312000         61440      2^7*3^4*5^3*7^2*11*13*17*19*23*29*31
    72779390658374400         62208      2^8*3^5*5^2*7^2*11*13*17*19*23*29*31
    74801040398884800         64512
        2^6*3^3*5^2*7^2*11*13*17*19*23*29*31*37
    106858629141264000        65536      2^7*3^3*5^3*7*11*13*17*19*23*29*31*37
    112201560598327200        69120
        2^5*3^4*5^2*7^2*11*13*17*19*23*29*31*37
    149602080797769600        73728
        2^7*3^3*5^2*7^2*11*13*17*19*23*29*31*37
    224403121196654400        80640
        2^6*3^4*5^2*7^2*11*13*17*19*23*29*31*37
    299204161595539200        82944
        2^8*3^3*5^2*7^2*11*13*17*19*23*29*31*37
    374005201994424000        86016
        2^6*3^3*5^3*7^2*11*13*17*19*23*29*31*37
    448806242393308800        92160
        2^7*3^4*5^2*7^2*11*13*17*19*23*29*31*37
    673209363589963200        96768
        2^6*3^5*5^2*7^2*11*13*17*19*23*29*31*37
    748010403988848000        98304
        2^7*3^3*5^3*7^2*11*13*17*19*23*29*31*37
    897612484786617600        103680
        2^8*3^4*5^2*7^2*11*13*17*19*23*29*31*37
***/
```

5.8 Josephus Problem

```
/***
            1
        8       2
    7               3
        6       4
            5

    N person ar standing in a circular fashion above. Person 1
    moves first. Starting from person 1, everyone kills the alive
    person next to him.
```

```
    SO, they will be killed in the following order
    2, 4, 6, 8, 3, 7, 5 and person 1 will servive.

    For any N, N can be written as
    N = 2^a + L where (L>=0 and a is as large as possible)

    The the surviver J(N) = 2*L + 1
    Because if N is a power of two, surviver J(N) = 1.

    J(2*N) = 2*J(N) + 1  [ for N >= 1 ]

**/
```

5.9 Mobius Function

```
/***
    mu[1] = 1, mu[n] = 0 if n has a squared prime factor,
    mu[n] = 1 if n is square-free with even number of prime factors
    mu[n] = -1 if n is square-free with odd number of prime factors

    *** sum of mu[d] where d | n is 0 ( For n=1, sum is 1 )***
***/

int mu[MAX] = {0};

void Mobius(int N){
    int i, j;
    mu[1] = 1;
    for (i = 1; i <= N; i++){
        if (mu[i]){
            for (j = i + i; j <= N; j += i){
                mu[j] -= mu[i];
            }
        }
    }
}
```

5.10 Order, Primitive Root, Discrete Log

```
/***
    If (a,n) = 1 :
        ord(a wrt n) = x such that,
        a^x = 1 (mod n) and x is the smallest number

        x will always divide phi(n)

    *** Primitive Root
    If n is a prime and x = (p - 1),
    then a is a primitive root of n.

    There, phi(phi(n)) primitive roots modulo n
    If a is primitive root and
```

```
14
15      e1,e2,... are all the numbers less than and coprime to phi(phi(n)),
16
17      then, all the primitive roots are
18      a^e1 (mod n)
19      a^e2 (mod n)
20      a^e3 (mod n)
21      ..........
22
23
24      *** Discrete Log
25      If n is a prime and a is a primitive root and 1 <= b <= n-1 :
26
27      b = a ^ x (mod n)
28
29      Then x is the discrete log of b for base (a,n)
30
31
32  ***/
```

5.11 Partition Numbers

```
1   /***
2       5   =   5
3           =   4 + 1
4           =   3 + 2
5           =   3 + 1 + 1
6           =   2 + 2 + 1
7           =   2 + 1 + 1 + 1
8           =   1 + 1 + 1 + 1 + 1
9
10      So , P(5) = 7
11      Recurrence, O(n*n)
12      P(n,m)  =   Number of ways to partition n where the maximum size of a
13                  part can be m ( a+b and b+a is considered as same patition)
14
15
16      Another way( O(n*sqrt(n)) ) :
17
18      k = 1,-1,2,-2,3,-3,4,-4,.....
19      if(n<0) P(n) = 0
20      if(n==0) P(n) = 1
21      P(n) = 0
22      for(all k)
23          g_k = (k*(3k-1))/2;
24          mul = -1^(k-1)
25          P(n) += mul * P(n-g_k)
26
27      g_k here is the k'th generalized pentagonal number
28  ***/
```

5.12 Permutation of size n with k inversions

```
/***
    Problem: https://www.hackerrank.com/contests/101hack43/challenges/k-
       inversion-permutations/problem

    Suppose, we have a permutation of size x with y inversions.
    We can get a new permutation of size (x+1) from that by
    inserting (x+1) somewhere in the previous permutation. The new
    permutation will have (y+y) inversions where (z <= x).

    Thus, every permutation of size n can having k inversions has one
    to one mapping to the solutions of the following equation.

    x_1 + x_2 + ... + x_n = k,  0 <= x_i <= (i-1)

    This equation can be solved by the help of inclusion exclusion.

    e_i --> (x_i >= i)

    ans = 0;
    for(s = 0; s <= k ; s++)
        ans += C(n+k-1-s, n-1) * F(s)

    F[s] = 0
    for(c = 0;; c++)
        F[s] += (-1)^c * G(s, c)

    ************************************************************************
    *   G(s, c) = number of ways to sum up to s using exactly c distinct   *
    *             integers of range [1,n]                                  *
    *           = G(s-c,c) + call(s-c,c-1) - call(s-(n+1),c-1)             *
    ************************************************************************

    As, the value of c can be approximately sqrt(k), the complexity
    of the whole solution is --> O(k * sqrt(k))
***/

#include <bits/stdc++.h>
using namespace std;
#define SET(a) memset(a,-1,sizeof(a))
const int MAX = 200010;
const int MOD = 1000000007;

int N;
int G[MAX][450];
inline int call(int i,int j) {
    if(i<0 or j<0) return 0;
    if(i==0) return (j==0);
    if(j==0) return (i==0);
    if(G[i][j]!=-1) return G[i][j];
    int ret = 0;
    ret = ( call(i-j,j) + call(i-j,j-1) ) % MOD;
```

```
    ret += (MOD - call(i-(N+1),j-1));
    if(ret>=MOD) ret -= MOD;
    return G[i][j] = ret;
}

int fact[MAX], inv[MAX], F[MAX];

void pre() {
    fact[0] = 1;
    inv[200000] = 750007460;
    for(int i=1;i<=2e5;i++) fact[i] = ( fact[i-1] * 1LL * i ) % MOD;
    for(int i=(2e5)-1;i>=0;i--) inv[i] = (inv[i+1] * 1LL * (i+1)) % MOD;
}

inline int C(int n,int r) {
    if(n<r) return 0;
    int ret = fact[n];
    ret = (ret * 1LL * inv[r]) % MOD;
    ret = (ret * 1LL * inv[n-r]) % MOD;
    return ret;
}

int main() {
//    freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);

    pre(); SET(G);
    int n,k;
    cin >> n >> k;
    N = n;

    for(int s=1;s<=k;s++) {
        for(int e=1;;e++) {
            if( (e * (e+1)) > s+s ) break;
            if(e & 1) {
                F[s] += (MOD - call(s,e));
                if(F[s]>=MOD) F[s] -= MOD;
            }
            else F[s] = (F[s] + call(s,e)) % MOD;
        }
    }
    int ans = C(n+k-1,n-1);
    for(int i=1;i<=k;i++) {
        ans += (F[i] * 1LL * C(n+k-1-i,n-1)) % MOD;
        ans %= MOD;
    }
    cout << ans << endl;
    return 0;
}
```

5.13 Stirling Numbers

```
/***
    **  Stirling Number of The First Kind :
        Number of ways you can decompose a set of size n
        into k disjoint cycles.

        S(n,k)  =   Number of ways you can decompose a set of size n
                    into k disjoint cycles.
                =   (n-1) * S(n-1,k) + S(n-1,k-1)

        S(n,1)  =   (n-1)!
        S(n,2)  =   C(n,2)
        S(n,n)  =   1

        k>= 1       2       3       4       5       6       7
        1   1
        2   1       1
        3   2       3       1
        4   6       11      6       1
        5   24      50      35      10      1
       (n)

        P(x,n) = x(x-1)(x-2) ... (x-(n-1))
        S(n,k) is the absolute value of the coefficient of x^k in P(x,n)

        P(x,1)  =   +x
        P(x,2)  =   -x      +x^2
        P(x,3)  =   +2x     -3x^2       +x^3
        P(x,4)  =   -6x     +11x^2      -6x^3   +x^4

        So, to know s(n,k) we just need to know a coefficient in a polynomial
        This can be done in O(n * logn * logn) using divide and conquer
           technique

        F(x) = x(x - 1)(x - 2)   ... (x - (n-1))
        Q(x) = x(x - 1)(x - 2)   ... (x - n/2)
        R(x) = (x - ((n/2) + 1)) ... (x - (n-1))

        F(x) = Q(x) * R(x)
        Q(x)*R(x) takes n*logn time
        Q(x) and R(x) can be determined in a recursive manner using
        the same technique as determining F(x)



    **  Stirling Number of The Second Kind :
        S(n,k)  =   Number of ways to partition a set of n objects
                    into k non-empty subsets
                =   k * S(n-1,k) + S(n-1,k-1)

        Example :

```

```
        Number of ways to color a 1*n grid using k colors such
        That each color is used at least once
        =   k! * S(n,k)


        k * logn way :
        S(n,k) = (1/k!)*sum
        sum = 0
        for(j from 0 to k)
            sum += ( (-1)^(k-j) ) * C(k,j) * j^n


***/
```

5.14 Summation of Floor Function Series

```
Subject: Formula for the sum of [ ] (the sign means floor)
Hello!

Is there any formula for the sum [p/q] + [2p/q] + [3p/q] + ... +
[np/q] (p, q, n are natural numbers)?

For example, [3/7] + [2*3/7] + [3*3/7] + [4*3/7] + [5*3/7]
             =  0   +    0    +    1    +    1    +    2
             = 4

I can't find a correct formula for any natural p and q.  I suspect
that it does not exist.  Maybe there is an algorithm for calculating
such sums?

It can be proved that if n = q - 1 then

[p/q] + [2p/q] + [3p/q] + ... + [(q-1)p/q] = (p-1)(q-1)/2.


Date: 01/14/2009 at 14:12:28
From: Doctor Vogler
Subject: Re: Formula for the sum of [ ]

Hi Ivan,

Thanks for writing to Dr. Math.  That's a very interesting question.
I think that your suspicion is correct that there is not a closed-form
formula for the general sum.  Fortunately, however, there is an
efficient algorithm for computing the sum.

By way of notation, I will write your sum as

  sum(k=1, n, [kp/q]).

Your formula for n=q-1 can be generalized to reduce n by any multiple
```

of q, as in

  sum(k=1, n, [kp/q]) =
      pt(n+1) - t(pqt + p + q - 1)/2 + sum(k=1, n-tq, [kp/q])

You'll notice that when t=1 and n=q-1 (so that the last sum is zero),
you get your formula. Actually, your formula and this formula both
depend on p and q having no factors in common (so that p/q is a
fraction in reduced form).

Of course, it's also easy to reduce p by an integer multiple of q, and
then we get

  sum(k=1, n, [kp/q]) = tn(n+1)/2 + sum(k=1, n, [k(p-qt)/q]).

But when n and p are both smaller than q, then it's harder to figure
out what to do. But here is one idea: We can count the number of
times that [kp/q] = m for each number m. It turns out that [kp/q] = m
when

  m <= kp/q < m+1

or

  mq/p <= k < (m+1)q/p

so that if both of those numbers are smaller than n, then this happens for

  {(m+1)q/p} - {mq/p}

different values of k, where I write {x} to mean x rounded *up* to the
nearest integer (which is also sometimes called the ceiling function),
in parallel to your notation [x] to mean x rounded *down* to the
nearest integer (which is also sometimes called the floor function).
(It turns out that these satisfy {x} = -[-x] for all x. Also, if x is
an integer, then {x} = [x] = x, but if x is not an integer, then {x} =
[x] + 1.) The last (biggest) m that will appear is m = [np/q], which
happens for

  n+1 - {mq/p}

different values of k. So that means that

  sum(k=1, n, [kp/q]) =
       m(n + 1 - {mq/p}) + sum(k=1, m-1, k({(k+1)q/p} - {kq/p})).

By looking at how this sum nearly telescopes, we can rewrite it as

  sum(k=1, n, [kp/q]) = m(n + 1) - sum(k=1, m, {kq/p}).

If we additionally assume that mq/p is never an integer, then this is

```
the same as

  sum(k=1, n, [kp/q]) = mn - sum(k=1, m, [kq/p]).

Now you might ask whether we have actually made any progress.  We've
changed one sum that we don't know how to evaluate efficiently for
another sum that looks exactly the same.  But here's the clincher:
The new sum changed around some numbers, allowing us to repeat all
three formulas and continue to make progress.  So when we put it all
together, we get the following very efficient algorithm:

  Input: Positive integers n, p, q
  Output: s = sum(k=1, n, [kp/q])

  Algorithm:
    t = GCD(p, q)
    p = p/t
    q = q/t
    s = 0
    z = 1
    while (q > 0) and (n > 0)
      (point A)
      t = [p/q]
      s = s + ztn(n+1)/2
      p = p - qt
      (point B)
      t = [n/q]
      s = s + zpt(n+1) - zt(pqt + p + q - 1)/2
      n = n - qt
      (point C)
      t = [np/q]
      s = s + ztn
      n = t
      swap p and q (e.g. t = p, p = q, q = t)
      z = -z

It can be proven that this algorithm will finish in polynomial time,
which basically means that a computer could use this to evaluate sums
very quickly even if the input numbers are hundreds or thousands of
digits long.  In fact, this algorithm is comparable to the Euclidean
Algorithm for computing the GCD of two numbers.

The reason the algorithm works is that we initially force GCD(p, q) =
1, so that our formula for reducing n by a multiple of q will work.
Then subtracting qt from p and swapping p and q do not change this
fact.  Then at points A, B, and C, the sum we are looking for is

  s + z*sum(k=1, n, [kp/q]).

Going from A to B, we use the formula for reducing p by a multiple of
q.  Going from B to C, we use the formula for reducing n by a multiple
```

```
of q.  Then going from C to A, we use the last formula that I
demonstrated.  The reason that kq/p is never an integer at that point
is that kq/p is never an integer at point C (for k <= t) is that when
we went from B to C, we caused that n < q, and then

  k <= t <= np/q < qp/q = p

and since k is less than p, p cannot be a factor of k.  And then since
q and p have no factors in common, p cannot be a factor of kq, so kq/p
is not an integer, and we can use the formula with [] rather than the
one with {}.
```

5.15 Geometry

5.15.1 2D Point Line Segment

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

using namespace std;

const double PI = acos(-1.0);
const double EPS = 1e-12;

/***
    u . v = |u|*|v|*cos(theta)
          = u.x*v.x + u.y*v.y
          = How much parallel they are
          = Dot product does not change if one vector move perpendicular to
             the other

    u x v = |u|*|v|*sin(theta)
          = u.x*v.y - v.x*u.y
          = How much perpendicular they are
          = Cross product does not change if one vector move parallel to the
             other


    dot(a-b,a-b) returns squared distance between pt a and pt b

***/

struct pt {
    double x, y;
    pt() {}
    pt(double x, double y) : x(x) , y(y) {}

    pt operator + (const pt &p) const { return pt( x+p.x , y+p.y ); }
    pt operator - (const pt &p) const { return pt( x-p.x , y-p.y ); }
    pt operator * (double c) const { return pt( x*c , y*c ); }
    pt operator / (double c) const { return pt( x/c , y/c ); }

    bool operator == (const pt &p) const { return ( fabs( x - p.x ) < EPS &&
```

```
        fabs( y - p.y ) < EPS ); }
    bool operator != (const pt &p) const { return !(pt(x,y) == p); }
};
ostream& operator << (ostream& os, pt p) {
    return os << "("<< p.x << "," << p.y << ")";
}

// u.v = |u|*|v|*cos(theta)
inline double dot(pt u, pt v) { return u.x*v.x + u.y*v.y; }

// a x b = |a|*|b|*sin(theta)
inline double cross(pt u, pt v) {return u.x*v.y - u.y*v.x;}

// returns |u|
inline double norm(pt u) { return sqrt(dot(u,u)); }

// returns angle between two vectors
inline double angle(pt u,pt v) {
    double cosTheta = dot(u,v)/norm(u)/norm(v);
    return acos(max(-1.0, min(1.0, cosTheta))); // keeping cosTheta in [-1,1]
}

// returns ang radian rotated version of vector u
// ccw rotation if angle is positive else cw rotation
inline pt rotate(pt u,double ang) {
    return pt( u.x*cos(ang) - u.y*sin(ang) , u.x*sin(ang) + u.y*cos(ang) );
}

// returns a vector perpendicular to v
inline pt perp(pt u) { return pt( -u.y , u.x ); }

// returns 2*area of triangle
inline double triArea2(pt a,pt b,pt c) { return cross(b-a,c-a); }

// compare function for angular sort around point P0
inline bool comp(pt P0,pt a, pt b) {
    double d = triArea2(P0, a, b);
    if(d < 0) return false;
    if(d == 0 && dot(P0-a, P0-a) > dot(P0-b, P0-b) ) return false;
    return true;
}

/***
    if line equation is, ax + by = c
    then,
        v --> direction vector of the line (b,-a)
        c --> v cross p
        p --> Any point(vector) on the line


    side(p) = ( v cross p) - c )
```

```
            = triArea2(origin,v,p)
    if side(p) is,
        positive --> p is above the line
        zero     --> p is on the line
        negative --> p is below the line
***/

struct line {
    pt v;
    double c;

    line(pt v, double c) : v(v), c(c) {}

    // From equation ax + by = c
    line(double a, double b, double c) : v({b,-a}), c(c) {}

    // From points p and q
    line(pt p, pt q) : v(q-p), c(cross(v,p)) {}

    // |v| * dist
    // dist --> distance of p from the line
    double side(pt p) { return cross(v,p)-c; }

    // better to using sqDist than dist
    double dist(pt p) { return abs(side(p)) / norm(v); }
    double sqDist(pt p) {return side(p)*side(p) / dot(v,v);}

    // perpendicular line through point p
    // 90deg ccw rotated line
    line perpThrough(pt p) { return {p, p + perp(v)}; }

    // translates a line by vector t(dx,dy)
    // every point (x,y) of previous line is translated to (x+dx,y+dy)
    line translate(pt t) {return {v, c + cross(v,t)};}

    // for every point
    // distance between previous position and current position is dist
    line shiftLeft(double dist) { return {v, c + dist*norm(v)}; }

    // projection of point p on the line
    pt projection(pt p) { return p - perp(v)*side(p)/dot(v,v); }

    // reflection of point p wrt the line
    pt reflection(pt p) { return p - perp(v)*side(p)*2.0/dot(v,v); }
};

inline bool lineLineIntersection(line l1, line l2, pt &out) {
    double d = cross(l1.v, l2.v);
    if (d == 0) return false;
    out = (l2.v*l1.c - l1.v*l2.c) / d;
    return true;
```

```
}


// interior = true for interior bisector
// interior = false for exterior bisector
inline line bisector(line l1, line l2, bool interior) {
    assert(cross(l1.v, l2.v) != 0); // l1 and l2 cannot be parallel!
    double sign = interior ? 1 : -1;
    return {l2.v/norm(l2.v) + (l1.v * sign)/norm(l1.v),
            l2.c/norm(l2.v) + (l1.c * sign)/norm(l1.v)};
}


/*** Segment ***/


/// C --> A circle which have diameter ab
/// returns true if point p is inside C or on the border of C
inline bool inDisk(pt a, pt b, pt p) { return dot(a-p, b-p) <= 0; }


/// returns true if point p is on the segment
inline bool onSegment(pt a, pt b, pt p) {
    return triArea2(a,b,p) == 0 && inDisk(a,b,p);
}

inline bool segSegIntersection(pt a,pt b,pt c,pt d,pt &out) {
    if(onSegment(a,b,c)) return out = c, true;
    if(onSegment(a,b,d)) return out = d, true;
    if(onSegment(c,d,a)) return out = a, true;
    if(onSegment(c,d,b)) return out = b, true;

    double oa = triArea2(c,d,a);
    double ob = triArea2(c,d,b);
    double oc = triArea2(a,b,c);
    double od = triArea2(a,b,d);

    if (oa*ob < 0 && oc*od < 0) {
        out = (a*ob - b*oa) / (ob-oa);
        return true;
    }
    return false;
}

// returns distance between segment ab and point p
inline double segPointDist(pt a,pt b,pt p) {
    if( norm(a-b) == 0 ) {
        line l(a,b);
        pt pr = l.projection(p);
        if(onSegment(a,b,p)) return l.dist(p);
    }
```

```
    return min(norm(a-p),norm(b-p));
}


// returns distance between segment ab and segment cd
inline double segSegDist(pt a, pt b, pt c, pt d) {
    double oa = triArea2(c,d,a);
    double ob = triArea2(c,d,b);
    double oc = triArea2(a,b,c);
    double od = triArea2(a,b,d);
    if (oa*ob < 0 && oc*od < 0) return 0; // proper intersection

    // If the segments don't intersect, the result will be minimum of these
       four
    return min({segPointDist(a,b,c), segPointDist(a,b,d),
                segPointDist(c,d,a), segPointDist(c,d,b)});
}


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

5.15.2 Circle Line Intersection

```
struct Point {
    double x, y;
    Point(double px, double py) {
        x = px;
        y = py;
    }
    Point sub(Point p2) {
        return Point(x - p2.x, y - p2.y);
    }
    Point add(Point p2) {
        return Point(x + p2.x, y + p2.y);
    }
    double distance(Point p2) {
        return sqrt((x - p2.x)*(x - p2.x) + (y - p2.y)*(y - p2.y));
    }
    Point normal() {
        double length = sqrt(x*x + y*y);
        return Point(x/length, y/length);
    }
    Point scale(double s) {
        return Point(x*s, y*s);
    }
};

struct line   // Creates a line with equation ax + by + c = 0
{
```

```
27      double a, b, c;
28      line() {}
29      line( Point p1,Point p2 ) {
30          a = p1.y - p2.y;
31          b = p2.x - p1.x;
32          c = p1.x * p2.y - p2.x * p1.y;
33      }
34  };
35
36  inline bool eq(double a, double b) {
37      return fabs( a - b ) < eps;
38  }
39  struct Circle {
40      double x, y, r, left,right;
41      Circle () {}
42      Circle(double cx, double cy, double cr) {
43          x = cx;
44          y = cy;
45          r = cr;
46          left = x - r;
47          right = x + r;
48      }
49      pair<Point, Point> intersections(Circle c) {
50          Point P0(x, y);
51          Point P1(c.x, c.y);
52          double d, a, h;
53          d = P0.distance(P1);
54          a = (r*r - c.r*c.r + d*d)/(2*d);
55          h = sqrt(r*r - a*a);
56          Point P2 = P1.sub(P0).scale(a/d).add(P0);
57          double x3, y3, x4, y4;
58          x3 = P2.x + h*(P1.y - P0.y)/d;
59          y3 = P2.y - h*(P1.x - P0.x)/d;
60          x4 = P2.x - h*(P1.y - P0.y)/d;
61          y4 = P2.y + h*(P1.x - P0.x)/d;
62
63          return pair<Point, Point>(Point(x3, y3), Point(x4, y4));
64      }
65  };
66  inline double Distance( Point a, Point b ) {
67      return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y )
            );
68  }
69
70  inline double Distance( Point P, line L ) {
71      return fabs( L.a * P.x + L.b * P.y + L.c ) / sqrt( L.a * L.a + L.b * L.b )
           ;
72  }
73  bool intersection(Circle C,line L,Point &p1,Point &p2) {
74      if( Distance( {C.x,C.y}, L ) > C.r + eps ) return false;
75      double a, b, c, d, x = C.x, y = C.y;
```

```
    d = C.r*C.r - x*x - y*y;
    if( eq( L.a, 0) ) {
        p1.y = p2.y = -L.c / L.b;
        a = 1;
        b = 2 * x;
        c = p1.y * p1.y - 2 * p1.y * y - d;
        d = b * b - 4 * a * c;
        d = sqrt( fabs (d) );
        p1.x = ( b + d ) / ( 2 * a );
        p2.x = ( b - d ) / ( 2 * a );
    }
    else {
        a = L.a *L.a + L.b * L.b;
        b = 2 * ( L.a * L.a * y - L.b * L.c - L.a * L.b * x);
        c = L.c * L.c + 2 * L.a * L.c * x - L.a * L.a * d;
        d = b * b - 4 * a * c;
        d = sqrt( fabs(d) );
        p1.y = ( b + d ) / ( 2 * a );
        p2.y = ( b - d ) / ( 2 * a );
        p1.x = ( -L.b * p1.y -L.c ) / L.a;
        p2.x = ( -L.b * p2.y -L.c ) / L.a;
    }
    return true;
}
```

5.15.3 Circle Operations

```
import java.util.*;

public class CircleOperations {
    static final double EPS = 1e-10;
    public static double fastHypot(double x, double y) {
        return Math.sqrt(x * x + y * y);
    }
    public static class Point {
        public double x, y;
        public Point(double x, double y) {
            this.x = x;
            this.y = y;
        }
    }
    public static class Circle {
        public double r, x, y;
        public Circle(double x, double y, double r) {
            this.x = x;
            this.y = y;
            this.r = r;
        }
        public boolean contains(Point p) {
            return fastHypot(p.x - x, p.y - y) < r + EPS;
        }
    }
```

```
26
27      public static class Line {
28          double a, b, c;
29          public Line(double a, double b, double c) {
30              this.a = a;
31              this.b = b;
32              this.c = c;
33          }
34          public Line(Point p1, Point p2) {
35              a = +(p1.y - p2.y);
36              b = -(p1.x - p2.x);
37              c = p1.x * p2.y - p2.x * p1.y;
38          }
39      }
40      // geometric solution
41      public static Point[] circleLineIntersection(Circle circle, Line line) {
42          double a = line.a;
43          double b = line.b;
44          double c = line.c + circle.x * a + circle.y * b;
45          double r = circle.r;
46          double aabb = a * a + b * b;
47          double d = c * c / aabb - r * r;
48          if (d > EPS)
49              return new Point[0];
50          double x0 = -a * c / aabb;
51          double y0 = -b * c / aabb;
52          if (d > -EPS)
53              return new Point[] {new Point(x0 + circle.x, y0 + circle.y)};
54          d /= -aabb;
55          double k = Math.sqrt(d < 0 ? 0 : d);
56          return new Point[] {
57              new Point(x0 + k * b + circle.x, y0 - k * a + circle.y),
58              new Point(x0 - k * b + circle.x, y0 + k * a + circle.y)
59          };
60      }
61
62      // algebraic solution
63      public static Point[] circleLineIntersection2(Circle circle, Line line) {
64          return Math.abs(line.a) >= Math.abs(line.b)
65                  ? intersection(line.a, line.b, line.c, circle.x, circle.y,
                        circle.r, false)
66                  : intersection(line.b, line.a, line.c, circle.y, circle.x,
                        circle.r, true);
67      }
68
69      static Point[] intersection(double a, double b, double c, double CX,
            double CY, double R, boolean swap) {
70          // ax+by+c=0
71          // (by+c+aCX)^2+(ay-aCY)^2=(aR)^2
72          double A = a * a + b * b;
73          double B = 2.0 * b * (c + a * CX) - 2.0 * a * a * CY;
```

```
74          double C = (c + a * CX) * (c + a * CX) + a * a * (CY * CY - R * R);
75          double d = B * B - 4 * A * C;
76          if (d < -EPS)
77              return new Point[0];
78          d = Math.sqrt(d < 0 ? 0 : d);
79          double y1 = (-B + d) / (2 * A);
80          double x1 = (-c - b * y1) / a;
81          double y2 = (-B - d) / (2 * A);
82          double x2 = (-c - b * y2) / a;
83  return swap ? d > EPS ? new Point[] {new Point(y1, x1), new Point(y2, x2)} :
84          new Point[] {new Point(y1, x1)}
85  :
86  d > EPS ? new Point[] {new Point(x1, y1), new Point(x2, y2)} :
87          new Point[] {new Point(x1, y1)};
88      }
89
90      public static Point[] circleCircleIntersection(Circle c1, Circle c2) {
91          if (fastHypot(c1.x - c2.x, c1.y - c2.y) < EPS) {
92              if (Math.abs(c1.r - c2.r) < EPS)
93                  return null; // infinity intersection points
94              return new Point[0];
95          }
96          double dx = c2.x - c1.x;
97          double dy = c2.y - c1.y;
98          double A = -2 * dx;
99          double B = -2 * dy;
100         double C = dx * dx + dy * dy + c1.r * c1.r - c2.r * c2.r;
101         Point[] res = circleLineIntersection(new Circle(0, 0, c1.r), new Line(
                A, B, C));
102         for (Point point : res) {
103             point.x += c1.x;
104             point.y += c1.y;
105         }
106         return res;
107     }
108     public static double circleCircleIntersectionArea(Circle c1, Circle c2) {
109         double r = Math.min(c1.r, c2.r);
110         double R = Math.max(c1.r, c2.r);
111         double d = fastHypot(c1.x - c2.x, c1.y - c2.y);
112         if (d < R - r + EPS)
113             return Math.PI * r * r;
114         if (d > R + r - EPS)
115             return 0;
116         double area = r * r * Math.acos((d * d + r * r - R * R) / 2 / d / r) +
                R * R
117                         * Math.acos((d * d + R * R - r * r) / 2 / d / R) - 0.5
118                         * Math.sqrt((-d + r + R) * (d + r - R) * (d - r + R) * (
                            d + r + R));
119         return area;
120     }
121
```

```
122     public static Line[] tangents(Circle a, Circle b) {
123         List<Line> lines = new ArrayList<>();
124         for (int i = -1; i <= 1; i += 2)
125             for (int j = -1; j <= 1; j += 2)
126                 tangents(new Point(b.x - a.x, b.y - a.y), a.r * i, b.r * j,
                        lines);
127         for (Line line : lines)
128             line.c -= line.a * a.x + line.b * a.y;
129         return lines.toArray(new Line[lines.size()]);
130     }
131 
132     static void tangents(Point center2, double r1, double r2, List<Line> lines
        ) {
133         double r = r2 - r1;
134         double z = center2.x * center2.x + center2.y * center2.y;
135         double d = z - r * r;
136         if (d < -EPS)
137             return;
138         d = Math.sqrt(d < 0 ? 0 : d);
139         lines.add(new Line((center2.x * r + center2.y * d) / z, (center2.y * r
                - center2.x * d) / z, r1));
140     }
141 
142     // min enclosing circle in O(n) on average
143     public static Circle minEnclosingCircle(Point[] pointsArray) {
144         if (pointsArray.length == 0)
145             return new Circle(0, 0, 0);
146         if (pointsArray.length == 1)
147             return new Circle(pointsArray[0].x, pointsArray[0].y, 0);
148         List<Point> points = Arrays.asList(pointsArray);
149         Collections.shuffle(points);
150         Circle circle = getCircumCircle(points.get(0), points.get(1));
151         for (int i = 2; i < points.size(); i++)
152             if (!circle.contains(points.get(i)))
153                 circle = minEnclosingCircleWith1Point(points.subList(0, i),
                        points.get(i));
154         return circle;
155     }
156 
157     static Circle minEnclosingCircleWith1Point(List<Point> points, Point q) {
158         Circle circle = getCircumCircle(points.get(0), q);
159         for (int i = 1; i < points.size(); i++)
160             if (!circle.contains(points.get(i)))
161                 circle = minEnclosingCircleWith2Points(points.subList(0, i),
                        points.get(i), q);
162         return circle;
163     }
164 
165     static Circle minEnclosingCircleWith2Points(List<Point> points, Point q1,
        Point q2) {
166         Circle circle = getCircumCircle(q1, q2);
```

```
167         for (Point point : points)
168             if (!circle.contains(point))
169                 circle = getCircumCircle(q1, q2, point);
170         return circle;
171     }
172 
173     public static Circle getCircumCircle(Point a, Point b) {
174         double x = (a.x + b.x) / 2.;
175         double y = (a.y + b.y) / 2.;
176         double r = fastHypot(a.x - x, a.y - y);
177         return new Circle(x, y, r);
178     }
179 
180     public static Circle getCircumCircle(Point a, Point b, Point c) {
181         double Bx = b.x - a.x;
182         double By = b.y - a.y;
183         double Cx = c.x - a.x;
184         double Cy = c.y - a.y;
185         double d = 2 * (Bx * Cy - By * Cx);
186         if (Math.abs(d) < EPS)
187             return getCircumCircle(new Point(Math.min(a.x, Math.min(b.x, c.x))
                    , Math.min(a.y, Math.min(b.y, c.y))),
188                                    new Point(Math.max(a.x, Math.max(b.x, c.x))
                                       , Math.max(a.y, Math.max(b.y, c.y))));
189         double z1 = Bx * Bx + By * By;
190         double z2 = Cx * Cx + Cy * Cy;
191         double cx = Cy * z1 - By * z2;
192         double cy = Bx * z2 - Cx * z1;
193         double x = cx / d;
194         double y = cy / d;
195         double r = fastHypot(x, y);
196         return new Circle(x + a.x, y + a.y, r);
197     }
198 
199     // Usage example
200     public static void main(String[] args) {
201         Random rnd = new Random(1);
202         for (int step = 0; step < 100_000; step++) {
203             int range = 10;
204             int x = rnd.nextInt(range) - range / 2;
205             int y = rnd.nextInt(range) - range / 2;
206             int r = rnd.nextInt(range);
207 
208             int x1 = rnd.nextInt(range) - range / 2;
209             int y1 = rnd.nextInt(range) - range / 2;
210             int x2 = rnd.nextInt(range) - range / 2;
211             int y2 = rnd.nextInt(range) - range / 2;
212             if (x1 == x2 && y1 == y2)
213                 continue;
214 
215             Point[] p1 = circleLineIntersection(new Circle(x, y, r), new Line(
```

```
                new Point(x1, y1), new Point(x2, y2)));
            Point[] p2 = circleLineIntersection2(new Circle(x, y, r), new Line
                (new Point(x1, y1), new Point(x2, y2)));

            if (p1.length != p2.length || p1.length == 1 && !eq(p1[0], p2[0])
                    || p1.length == 2 && !(eq(p1[0], p2[0]) && eq(p1[1], p2
                        [1]) || eq(p1[0], p2[1]) && eq(p1[1], p2[0])))
                throw new RuntimeException();
        }
    }
    static boolean eq(Point p1, Point p2) {
        return !(fastHypot(p1.x - p2.x, p1.y - p2.y) > 1e-9);
    }
}
```

5.15.4 Circle

```
struct circle {
    pt c;
    double r;
    circle() {}
    circle(pt c, double r) : c(c) , r(r) {}
};


/* returns circumcircle of a triangle
   the radius of circumcircle  -->  intersection point of the perpendicular
                                    bisectors of the three sides */
circle circumCircle(pt a, pt b, pt c) {
    b = b-a, c = c-a; // consider coordinates relative to point a
    assert(cross(b,c) != 0); // no circumcircle if A,B,C are co-linear
    // detecting the intersection point using the same technique used in line
       line intersection
    pt center = a + ( perp( b*dot(c,c) - c*dot(b,b) )/cross(b,c)/2 );
    return {center, norm(center-a)};
}

int sgn(double val) {
    if(val>0) return 1;
    else if(val == 0) return 0;
    else return -1;
}

/*  returns number of intersection points between a line and a circle
    O --> Center
    I,J --> Intersection points
    P -- > Projection of O onto line l
    IP = JP = h , OP = d */
int circleLineIntersection(circle c, line l, pair<pt,pt> &out) {
    double h2 = c.r*c.r - l.sqDist(c.c); // h^2
    if (h2 >= 0) { // the line touches the circle
        pt p = l.proj(c.c); // point P
```

```
35          pt h = l.v*sqrt(h2)/norm(l.v); // vector parallel to l, of length h
36          out = {p-h, p+h}; // {I,J}
37      }
38      return 1 + sgn(h2); // number of intersection points
39  }
40
41  /*  returns number of intersection points between two circles
42      O_i --> Center of circle i
43      I,J --> Intersection points
44      P -- > Projection of O onto line IJ
45      IP = JP = h , O_1O_2 = d */
46  int circleCircleIntersection(circle c1, circle c2, pair<pt,pt> &out) {
47      pt d = c2.c - c1.c; double d2 = dot(d,d); // d^2
48      if (d2 == 0) { // concentric circle
49          assert(c1.r != c2.r); // same circle
50          return 0;
51      }
52      double pd = (d2 + c1.r*c1.r - c2.r*c2.r)/2; // = |O_1P| * d
53      double h2 = c1.r*c1.r - pd*pd/d2; // = h\882
54      if (h2 >= 0) {
55          pt p = c1.c + d*pd/d2, h = perp(d)*sqrt(h2/d2);
56          out = {p-h, p+h};
57      }
58      return 1 + sgn(h2);
59  }
60
61  /*  inner --> if true returns inner tangents
62
63      * if the radius of c2 is 0, returns tangents that go through the center
64        of circle c2 (value of inner is does not matter in this case)
65
66      * if there are 2 tangents, it fills out with two pairs of points: the
           pairs
67        of tangency points on each circle (P1; P2), for each of the tangents
68      * if there is 1 tangent, the circles are tangent to each other at some
           point
69        P, out just contains P 4 times, and the tangent line can be found as
70        line(c1.c,p).perpThrough(p)
71      * if there are 0 tangents, it does nothing
72      * if the circles are identical, it aborts. */
73  int tangents(circle c1, circle c2, bool inner, vector < pair <pt,pt> > &out) {
74      if (inner) c2.r = -c2.r;
75      pt d = c2.c-c1.c;
76      double dr = c1.r-c2.r, d2 = dot(d,d), h2 = d2-dr*dr;
77      if (d2 == 0 || h2 < 0) {
78          //assert(h2 != 0);
79          return 0;
80      }
81      for (double sign : {-1,1}) {
82          pt v = (d*dr + perp(d)*sqrt(h2)*sign)/d2;
83          out.push_back({c1.c + v*c1.r, c2.c + v*c2.r});
```

```
    }
    return 1 + (h2 > 0);
}
```

5.15.5 Convex Hull

```
/***
    ConvexHull : Graham's Scan O( n * lg(n) )

    0 based P and C

    P[]: holds all the Points, C[]: holds Points on the hull(in anti clockwise
        order)

    np: number of Points in P[], nc: number of Points in C[]

    If there are duplicate Points in P, call makeUnique() before
    calling convexHull(), call convexHull() if you have np >= 3

    to remove co-linear Points on hull, call compress() after convexHull()

    Call getBakiPoints() to get all the points that lie in the perimeter of
    the convex hull


    In case you get TLE, you might try changing the data type of point from
        int to double
***/


struct pt {
    double x, y;
    pt() {}
    pt(double x, double y) : x(x) , y(y) {}

    pt operator - (const pt &p) const { return pt( x-p.x , y-p.y ); }

    bool operator == (const pt &p) const { return ( fabs( x - p.x ) < EPS &&
        fabs( y - p.y ) < EPS ); }
    bool operator != (const pt &p) const { return !(pt(x,y) == p); }
};

pt P[MAX], C[MAX], P0;

bool nisi[MAX];

inline double dot(pt u, pt v) { return u.x*v.x + u.y*v.y; }
inline double cross(pt u, pt v) {return u.x*v.y - u.y*v.x;}
inline double triArea2(pt a,pt b,pt c) { return cross(b-a,c-a); }

inline bool comp(const pt &a, const pt &b) {
    double d = triArea2(P0, a, b);
```

```
44      if(d < 0) return false;
45      if(d == 0 && dot(P0-a,P0-a) > dot(P0-b,P0-b)) return false;
46      return true;
47  }
48
49  void convexHull(int &np, int &nc) {
50      int i, j, pos = 0;
51      for(i = 1; i < np; i++)
52          if(P[i].y<P[pos].y || ( fabs(P[i].y - P[pos].y) < EPS && P[i].x<P[pos
                ].x ) )
53              pos = i;
54      swap(P[0], P[pos]);
55      P0 = P[0];
56      sort(P+1, P+np, comp);
57      for(i = 0; i < 3; i++) C[i] = P[i];
58      for(i = j = 3; i < np; i++) {
59          while(triArea2(C[j-2], C[j-1], P[i]) < 0) j--;
60          C[j++] = P[i];
61      }
62      nc = j;
63  }
64
65  inline bool normal(const pt &a, const pt &b) {
66      return ( fabs( a.x - b.x ) < EPS ? a.y < b.y : a.x < b.x);
67  }
68
69  inline void makeUnique(int &np) {
70      sort(P , P+np , normal);
71      np = unique(P , P+np) - P;
72  }
73
74  void compress(int &nc) {
75      int i, j;
76      double d;
77      C[nc] = C[0];
78      for(i=j=1; i < nc; i++) {
79          d = triArea2(C[j-1], C[i], C[i+1]);
80          if( d != 0 || ( d == 0 && C[j-1] == C[i+1]) ) C[j++] = C[i];
81      }
82      nc = j;
83  }
84
85  void getBakiPoints(int &np, int &nc){
86      int j = 0;
87      for(int i=0;i<nc;i++){
88          while( C[i] != P[j] ) j++;
89          nisi[j] = true; /// If the point is already taken
90      }
91
92      int last = nc;
93      for(int i = np-1; i >= 0 ; i--){
```

```
94          if(!nisi[i]){
95              if( P[i] != C[0] && P[i] != C[last-1] )
96                  if( triArea2(P[i],C[0],C[nc-1]) == 0 )
97                      C[nc++] = P[i];
98          }
99      }
100 }
```

5.15.6 Line Operations

```
1  import java.util.*;
2
3  public class LineGeometry
4  {
5      static final double EPS = 1e-10;
6
7      public static int sign(double a)
8      {
9          return a < -EPS ? -1 : a > EPS ? 1 : 0;
10     }
11     public static class Point implements Comparable<Point> {
12         public double x, y;
13         public Point(double x, double y) {
14             this.x = x;
15             this.y = y;
16         }
17         public Point minus(Point b) {
18             return new Point(x - b.x, y - b.y);
19         }
20         public double cross(Point b) {
21             return x * b.y - y * b.x;
22         }
23         public double dot(Point b) {
24             return x * b.x + y * b.y;
25         }
26         public Point rotateCCW(double angle) {
27             return new Point(x * Math.cos(angle) - y * Math.sin(angle), x *
                   Math.sin(angle) + y * Math.cos(angle));
28         }
29
30         @Override
31         public int compareTo(Point o) {
32             // return Double.compare(Math.atan2(y, x), Math.atan2(o.y, o.x));
33             return Double.compare(x, o.x) != 0 ? Double.compare(x, o.x) :
                   Double.compare(y, o.y);
34         }
35     }
36
37     public static class Line {
38         public double a, b, c;
39         public Line(double a, double b, double c) {
40             this.a = a;
```

```
41              this.b = b;
42              this.c = c;
43          }
44
45          public Line(Point p1, Point p2) {
46              a = +(p1.y - p2.y);
47              b = -(p1.x - p2.x);
48              c = p1.x * p2.y - p2.x * p1.y;
49          }
50
51          public Point intersect(Line line) {
52              double d = a * line.b - line.a * b;
53              if (sign(d) == 0) {
54                  return null;
55              }
56              double x = -(c * line.b - line.c * b) / d;
57              double y = -(a * line.c - line.a * c) / d;
58              return new Point(x, y);
59          }
60      }
61
62      // Returns -1 for clockwise, 0 for straight line, 1 for counterclockwise
           order
63      public static int orientation(Point a, Point b, Point c) {
64          Point AB = b.minus(a);
65          Point AC = c.minus(a);
66          return sign(AB.cross(AC));
67      }
68
69      public static boolean cw(Point a, Point b, Point c) {
70          return orientation(a, b, c) < 0;
71      }
72
73      public static boolean ccw(Point a, Point b, Point c) {
74          return orientation(a, b, c) > 0;
75      }
76
77      public static boolean isCrossIntersect(Point a, Point b, Point c, Point d)
            {
78          return orientation(a, b, c) * orientation(a, b, d) < 0 && orientation(
               c, d, a) * orientation(c, d, b) < 0;
79      }
80
81      public static boolean isCrossOrTouchIntersect(Point a, Point b, Point c,
           Point d) {
82          if (Math.max(a.x, b.x) < Math.min(c.x, d.x) - EPS || Math.max(c.x, d.x
               ) < Math.min(a.x, b.x) - EPS
83                  || Math.max(a.y, b.y) < Math.min(c.y, d.y) - EPS || Math.max(c
                       .y, d.y) < Math.min(a.y, b.y) - EPS)
84          {
85              return false;
```

```
86          }
87          return orientation(a, b, c) * orientation(a, b, d) <= 0 && orientation
                (c, d, a) * orientation(c, d, b) <= 0;
88      }
89
90      public static double pointToLineDistance(Point p, Line line) {
91          return Math.abs(line.a * p.x + line.b * p.y + line.c) / fastHypot(line
                .a, line.b);
92      }
93
94      public static double fastHypot(double x, double y) {
95          return Math.sqrt(x * x + y * y);
96      }
97
98      public static double sqr(double x) {
99          return x * x;
100     }
101     public static double angleBetween(Point a, Point b) {
102         return Math.atan2(a.cross(b), a.dot(b));
103     }
104     public static double angle(Line line) {
105         return Math.atan2(-line.a, line.b);
106     }
107     public static double signedArea(Point[] points) {
108         int n = points.length;
109         double area = 0;
110         for (int i = 0, j = n - 1; i < n; j = i++) {
111             area += (points[i].x - points[j].x) * (points[i].y + points[j].y);
112             // area += points[i].x * points[j].y - points[j].x * points[i].y;
113         }
114         return area / 2;
115     }
116
117     public static enum Position {
118         LEFT, RIGHT, BEHIND, BEYOND, ORIGIN, DESTINATION, BETWEEN
119     }
120
121     // Classifies position of point p against vector a
122     public static Position classify(Point p, Point a) {
123         int s = sign(a.cross(p));
124         if (s > 0) {
125             return Position.LEFT;
126         }
127         if (s < 0) {
128             return Position.RIGHT;
129         }
130         if (sign(p.x) == 0 && sign(p.y) == 0) {
131             return Position.ORIGIN;
132         }
133         if (sign(p.x - a.x) == 0 && sign(p.y - a.y) == 0) {
134             return Position.DESTINATION;
```

```
        }
        if (a.x * p.x < 0 || a.y * p.y < 0) {
            return Position.BEYOND;
        }
        if (a.x * a.x + a.y * a.y < p.x * p.x + p.y * p.y) {
            return Position.BEHIND;
        }
        return Position.BETWEEN;
    }

    // cuts right part of poly (returns left part)
    public static Point[] convexCut(Point[] poly, Point p1, Point p2) {
        int n = poly.length;
        List<Point> res = new ArrayList<>();
        for (int i = 0, j = n - 1; i < n; j = i++) {
            int d1 = orientation(p1, p2, poly[j]);
            int d2 = orientation(p1, p2, poly[i]);
            if (d1 >= 0)
                res.add(poly[j]);
            if (d1 * d2 < 0)
                res.add(new Line(p1, p2).intersect(new Line(poly[j], poly[i]))
                    );
        }
        return res.toArray(new Point[res.size()]);
    }
    // Usage example
    public static void main(String[] args)
    {
    }
}
```

5.15.7 Pick's Theorem

```
/***
    A = I + (B/2) - 1;

    A = Area of the polygon(Should be a simple polygon
                            and the vertexes should be lattice points)
    I = Number of Interior Lattice Points
    B = Number of Boundary Lattice Points

    P   =   N umber of lattice points in the line (x_0,y_0)-->(x_1,y_1)
    Suppose, X   =   x_1 - x_0 , Y = y_1 - y_0
    Then,   P   = gcd(X,Y)+1
    The boundary points can be counted by this way.

    Area can be calculated by any standard way.

    Then the only unknown will be I.
***/
```

5.15.8 Point Inside Poly (log n)

```
/***
    * The following code determines if point p is inside the polygon or not
    * works for convex polygon only
    * Complexity O( log n )
***/

struct pt {
    double x, y;
    pt() {}
    pt(double x, double y) : x(x) , y(y) {}
    pt(const pt &p) : x(p.x) , y(p.y) {}

    pt operator + (const pt &p) const { return pt( x+p.x , y+p.y ); }
    pt operator - (const pt &p) const { return pt( x-p.x , y-p.y ); }
    pt operator * (double c) const { return pt( x*c , y*c ); }
};

inline double dot(pt u, pt v) { return u.x*v.x + u.y*v.y; }
inline double cross(pt u, pt v) {return u.x*v.y - u.y*v.x;}
inline double triArea2(pt a,pt b,pt c) { return cross(b-a,c-a); }

inline bool inDisk(pt a, pt b, pt p) { return dot(a-p, b-p) <= 0; }
inline bool onSegment(pt a, pt b, pt p) { return triArea2(a,b,p) == 0 &&
   inDisk(a,b,p); }

// points of the polygon has to be in ccw order
// if strict, returns false when a is on the boundary
inline bool insideConvexPoly(pt* C, int nc, pt  p, bool strict) {
    int st = 1, en = nc - 1, mid;
    while(en - st > 1) {
        mid = (st + en)>>1;
        if(triArea2(C[0], C[mid], p) < 0) en = mid;
        else st = mid;
    }
    if(strict) {
        if(st==1) if(onSegment(C[0],C[st],p)) return false;
        if(en==nc-1) if(onSegment(C[0],C[en],p)) return false;
        if(onSegment(C[st],C[en],p)) return false;
    }
    if(triArea2(C[0], C[st], p) < 0) return false;
    if(triArea2(C[st], C[en], p) < 0) return false;
    if(triArea2(C[en], C[0], p) < 0) return false;
    return true;
}
```

5.15.9 Point Inside Poly (Ray Shooting)

```
/***
    * The following code determines if point a is inside the polygon or not
    * ray is shot from point a to the right of a
    * works for both convex and concave polygon
    * Complexity O(n)
```

```
***/

struct pt {
    double x, y;
    pt() {}
    pt(double x, double y) : x(x) , y(y) {}
    pt(const pt &p) : x(p.x) , y(p.y) {}

    pt operator + (const pt &p) const { return pt( x+p.x , y+p.y ); }
    pt operator - (const pt &p) const { return pt( x-p.x , y-p.y ); }
    pt operator * (double c) const { return pt( x*c , y*c ); }
};

inline double dot(pt u, pt v) { return u.x*v.x + u.y*v.y; }
inline double cross(pt u, pt v) {return u.x*v.y - u.y*v.x;}
inline double triArea2(pt a,pt b,pt c) { return cross(b-a,c-a); }

inline bool inDisk(pt a, pt b, pt p) { return dot(a-p, b-p) <= 0; }
inline bool onSegment(pt a, pt b, pt p) { return triArea2(a,b,p) == 0 &&
   inDisk(a,b,p); }

// check if segment pq crosses ray from point a
inline bool crossesRay(pt a, pt p, pt q) {
    int mul = (q.y>=a.y) - (p.y>=a.y);
    return (mul * triArea2(a,p,q)) > 0;
}

// if strict, returns false when a is on the boundary
inline bool insidePoly(pt *P, int np, pt a, bool strict = true) {
    int numCrossings = 0;
    for (int i = 0; i < np; i++) {
        if (onSegment(P[i], P[(i+1)%np], a)) return !strict;
        numCrossings += crossesRay(a, P[i], P[(i+1)%np]);
    }
    return (numCrossings & 1); // inside if odd number of crossings
}
```

5.15.10 Use of atan2

```
/*** Use of atan2
    atan2f(y,x) --> float
    atan2(y,x) --> double
    atan2l(y,x) ---> long double

    returns angle between positive x axis and the line segment (0,0)-->(x,y)
    Angle is returned in radian and is in range [-pi,+pi]
***/
```

5.16 Matrices

5.16.1 Gauss-Jordan Elimination in GF(2)

```
/***
```

```
    * mat is 0 based
    * n rows(equations) and m columns(variables)
    * ans[i] = solution for the i'th variable (0 based)
    * where[i] = the row index of the pivot element of column i
    * call to GaussJordan() returns the (number of solutions % MOD)

    * In every test case, clear ( mat[i].reset() ) mat first and then do the changes

    * For solving problems on graphs with probability/expectation, make sure the graph
      is connected and a single component. If not, then re-number the vertex and solve
      for each connected component separately.

    * Complexity --> O( min(n,m) * nm )/64 because of using bitset
**/

const int SZ = 105;
const int MOD = 1e9 + 7;

bitset <SZ> mat[SZ];
int where[SZ];
bitset <SZ> ans;

ll bigMod(ll a,ll b,ll m){
    ll ret = 1LL;
    a %= m;
    while (b){
        if (b & 1LL) ret = (ret * a) % m;
        a = (a * a) % m;
        b >>= 1LL;
    }
    return ret;
}

/// n for row, m for column, modulo 2
int GaussJordan(int n,int m) {
    SET(where); /// sets to -1
    for(int r=0,c=0; c<m && r<n; c++) {
        for(int i=r; i<n; i++)
            if( mat[i][c] ){
                swap(mat[i],mat[r]); break;
            }

        if( !mat[r][c] ) continue;

        where[c] = r;

        for (int i=0; i<n; ++i)
            if (i != r && mat[i][c])
```

```
                mat[i] ^= mat[r];
        r++;
    }

    for(int j=0; j<m; j++) {
        if(where[j]!=-1) ans[j] = mat[where[j]][m]/mat[where[j]][j];
        else ans[j] = 0;
    }

    for(int i=0; i<n; i++){
        int sum = 0;
        for(int j=0; j<m; j++) sum ^=  (ans[j] & mat[i][j]);
        if( sum != mat[i][m] ) return 0;  /// no solution
    }

    int cnt = 0;
    for(int j=0; j<m; j++) if (where[j]==-1) cnt++;
    return bigMod(2,cnt,MOD); /// how many solutions modulo some other MOD
}
```

5.16.2 Gauss-Jordan Elimination in GF(P)

```
/***
    * mat is 0 based
    * Keep the elements of matrix between 0 to P-1
    * n rows(equations) and m columns(variables)
    * ans[i] = solution for the i'th variable (0 based)
    * where[i] = the row index of the pivot element of column i
    * call to GaussJordan() returns the (number of solutions % MOD)

    * In every test case, clear mat first and then do the changes

    * For solving problems on graphs with probability/expectation, make sure
       the graph
      is connected and a single component. If not, then re-number the vertex
         and solve
      for each connected component separately.

    * Complexity --> O( min(n,m) * nm )
**/

const int SZ = 105;
const int MOD = 1e9 + 7;

int mat[SZ][SZ], where[SZ], ans[SZ];

ll bigMod(ll a,ll b,ll m){
    ll ret = 1LL;
    a %= m;
    while (b){
        if (b & 1LL) ret = (ret * a) % m;
        a = (a * a) % m;
```

```
29          b >>= 1LL;
30      }
31      return ret;
32  }
33
34  int GaussJordan(int n,int m,int P) {
35      SET(where); /// sets to -1
36      for(int r=0,c=0; c<m && r<n; c++) {
37          int mx = r;
38          for(int i=r; i<n; i++) if( mat[i][c] > mat[mx][c] ) mx = i;
39
40          if( mat[mx][c] == 0 ) continue;
41
42          if(r != mx) for(int j=c; j<=m; j++) swap(mat[r][j],mat[mx][j]);
43
44          where[c] = r;
45
46          int mul,minv = bigMod(mat[r][c],P-2,P);
47          int temp;
48          for(int i=0; i<n; i++){
49              if( i!=r && mat[i][c]!=0){
50                  mul = ( mat[i][c] * (long long) minv ) % P;
51                  for(int j=c; j<=m; j++) {
52                      temp = mat[i][j];
53                      temp -= ( ( mul * (long long) mat[r][j] ) % P );
54                      temp += P;
55                      if( temp >= P ) temp -= P;
56                      mat[i][j] = temp;
57                  }
58              }
59          }
60          r++;
61      }
62
63      for(int j=0; j<m; j++) {
64          if(where[j]!=-1) ans[j] = ( mat[where[j]][m] * 1LL * bigMod(mat[where[
                j]][j],P-2,P) ) % P;
65          else ans[j] = 0;
66      }
67
68      for(int i=0; i<n; i++){
69          int sum = 0;
70          for(int j=0; j<m; j++) {
71              sum += ( ans[j] * 1LL * mat[i][j] ) % P;
72              if(sum >= P) sum -= P;
73          }
74          if( sum != mat[i][m] ) return 0;  /// no solution
75      }
76
77      int cnt = 0;
78      for(int j=0; j<m; j++) if (where[j]==-1) cnt++;
```

```
    return bigMod(P,cnt,MOD);
}
```

5.16.3 Gauss-Jordan Elimination

```
/***
    * mat is 0 based
    * n rows(equations) and m columns(variables)
    * ans[i] = solution for the i'th variable (0 based)
    * where[i] = the row index of the pivot element of column i
    * call to GaussJordan() returns the number of solutions

    * In every test case, clear mat first and then do the changes

    * For solving problems on graphs with probability/expectation, make sure the graph
      is connected and a single component. If not, then re-number the vertex and solve
      for each connected component separately.

    * Complexity --> O( min(n,m) * nm )
***/

const int SZ = 105;
const double EPS = 1e-9;

double mat[SZ][SZ], ans[SZ];
int where[SZ];

int GaussJordan(int n,int m) {
    SET(where); /// sets to -1
    for(int r=0,c=0; c<m && r<n; c++) {
        int mx = r;
        for(int i=r; i<n; i++)
            if( abs(mat[i][c]) > abs(mat[mx][c]) ) mx = i;

        if( abs(mat[mx][c]) < EPS ) continue;

        if(r != mx) for(int j=c; j<=m; j++) swap(mat[r][j],mat[mx][j]);

        where[c] = r;

        for(int i=0; i<n; i++)
            if( i!=r ){
                double mul = mat[i][c]/mat[r][c];
                for(int j=c; j<=m; j++) mat[i][j] -= mul*mat[r][j];
            }
        r++;
    }

    for(int j=0; j<m; j++) {
        if(where[j]!=-1) ans[j] = mat[where[j]][m]/mat[where[j]][j];
```

```
        else ans[j] = 0;
    }

    for(int i=0; i<n; i++){
        double sum = 0;
        for(int j=0; j<m; j++) sum += ans[j] * mat[i][j];
        if( abs(sum - mat[i][m]) > EPS ) return 0; /// no solution
    }

    for(int j=0; j<m; j++) if (where[j]==-1) return INF;
    return 1;
}
```

5.16.4 Gaussian Related Problem 1

```
/***
    http://codeforces.com/blog/entry/9518
    Problem : Given n numbers, you have to take a subset.
              Let X denote the xor of the numbers of the subset.

              You will be given some conditions on the bits of X.
              condition(b,1) means try to make the b'th bit of X 1
              condition(b,0) means try to make the b'th bit of X 0
              The conditions will be ordered

              Suppose 6 conditions are given.
              Y = 100110 denotes condition 1,4,5 are satisfied
              We will try to maximize Y
***/

const int MAX = 100010;

ll ara[MAX];
bitset <MAX> mat[70];
int n; /// Number of input integers/number of columns in matrix
int row, ans[MAX], where[MAX];

void addCondition(int bn,int val){
    ++row;

    mat[row].reset();
    mat[row][n] = val;

    for(int col=0; col<n; col++) mat[row][col]=( (ara[col]>>bn) & 1 );

    for(int col=0; col<n; col++){
        if(mat[row][col]){
            if(where[col]) mat[row] ^= mat[where[col]];
            else break;
        }
    }
    for(int col=0; col<n; col++){
```

```
        if(mat[row][col]){
            where[col]=row;
            return;
        }
    }
    --row;
}

struct data {
    int bitNumber;
    int val; /// preferred value for that bit
};
vector <data> conditions;


/// m denotes maximum number of bits of any number in the input
void solve() {
    CLR(where);
    row = 0;
    for(int i=0;i<conditions.size();i++){
        addCondition(conditions[i].bitNumber,conditions[i].val);
    }
    for(int i=n-1;i>=0;i--){
        if(mat[where[i]][n]){
            ans[i] = 1;
            for(int j=1;j<=row;j++){
                if(mat[j][i]) mat[j].flip(n);
            }
        }
        else ans[i] = 0;
    }
}

int main() {
    /// scan n integer numbers(0 based)
    /// fill conditions vector

    solve();

    /// now ans[i] will be 1 if the i'th integer is taken in the subset
    return 0;
}
```

5.16.5 Gaussian Related Problem 2,3

```
/***
Problem : https://codeforces.com/contest/1101/problem/G

You are given an array a[0...(n-1)] of integer numbers.
Your task is to divide the array into the maximum number of
non empty segments in such a way that :
```

```
there doesn't exist a non-empty subset of segments such that the XOR-sum
of the numbers from them is equal to 0.

Solution :

Build a cumilitive xor array. ( cum[i] = cum[i-1] ^ a[i] )
Build a matrix where row[i] = binary representation of cum[i]
Answer is the rank of the matrix.

If cum[n-1] = 0, no solution.
***/


/***
problem :
Given a set S of size N, find the number of distinct integers that
can be represented using xor over the set of the given elements.

Solution :
This is 2x, where x is the size of the basis of the given set of size N.
To find the basis, use greedy technique and Gaussian elimination, checking
at each step if the current element can be expressed as linear combination
of xor of the elements already in the basis. In fact, the size of the basis
is even equal to the number of pivoted columns in the RREF of the matrix M,
where M is formed using elements of set S as its columns.
***/
```

5.16.6 Gaussian Related Problem 4

```
/***
Problem :
Given an array ara[] of n integers, you will be given some queries.
? L x --> How many subsequences of ara[1:L] has XOR-sum = x

Solution :
Let dp[i][x] be the number of subsequences of the first i elements with xor x.
Let's prove that dp[i][x] is equal for all x belonging to the set!
Let's assume this holds true for i-1 and see what happens in the transition to
    i.
Notice that it holds true for i=0. Let j be the value that dp[i-1][x] is equal
    to
for all x belonging to the set. If a[i] is in the set, and x is in the set,
(x^a[i]) is in the set. Therefore, dp[i-1][x]=j and  which makes dp[i][x]=2*j
   for all x in the set.
Notice that the set doesn't change so dp[i][x]=0 for all x that aren't in the
   set.
If a[i] isn't in the set, we have 3 cases for x. If x is in the set,  isn't in
    the set.
Therefore, dp[i][x]=j+0=j. If x is to be added to the set in this step, is in
   the set.
Therefore, dp[i][x]=0+j=j. Otherwise, dp[i][x]=0.

```

```
Can also be solved using gaussian but pari na.

***/

#include <bits/stdc++.h>
using namespace std;
const int MOD = 1000000007;
const int MAX = 100010;
vector<pair<int,int> > v[MAX];
int arr[MAX], res[MAX];
vector<int> s;
bool b[(1<<20)];
int main() {
    int n, q, L, x;
    cin >> n >> q;
    for (int i=0;i<n;i++) cin >> ara[i];
    for (int i=0;i<q;i++) {
        cin >> L >> x;
        v[L-1].push_back({x,i});
    }
    s.push_back(0);
    b[0]=1;
    int ans=1;
    for (int i=0;i<n;i++) {
        if (b[arr[i]]) ans = (ans + ans) % MOD;
        else {
            int tmp=s.size();
            for (int x=0;x<tmp;x++) {
                s.push_back(s[x]^arr[i]);
                b[s.back()]=1;
            }
        }
        for (int x=0;x<v[i].size();x++)
        res[v[i][x].second] = ans * b[v[i][x].first];
    }
    for (int i=0;i<q;i++) cout << res[i] << endl;
    return 0;
}
```

5.16.7 Matrix Determinant

```
/***
    * mat is 0 based
    * n x n matrix
    * If the entries are integer, the final result will also be integer
      But the returned value is of double type.
      To print it as an interge, do the following.
      cout << (int)round(determinant(n)) << endl;

    * In every test case, clear mat first and then do the changes
    * Complexity --> O( min(n,m) * nm )
***/
```

```cpp

const int SZ = 105;
const double EPS = 1e-9;
double mat[SZ][SZ];

double determinant(int n) {
    int sign = 1;
    for(int r=0,c=0; c<n && r<n; c++) {
        int mx = r;
        for(int i=r; i<n; i++)
            if( abs(mat[i][c]) > abs(mat[mx][c]) ) mx = i;

        if( abs(mat[mx][c]) < EPS ) continue;

        if(r != mx) {
            for(int i=c; i<n; i++) swap(mat[r][i],mat[mx][i]);
            sign *= -1;
        }
        for(int i=0; i<n; i++)
            if( i!=r ){
                double mul = mat[i][c]/mat[r][c];
                for(int j=c; j<n; j++) mat[i][j] -= mul*mat[r][j];
            }
        r++;
    }
    double ret = 1;
    for(int i=0;i<n;i++) ret *= mat[i][i];
    return sign * ret;
}
```

5.16.8 Matrix Exponentiation

```cpp
struct matrix{
    ll mat[100][100];
    int dim;
    matrix(){};
    matrix(int d){
        dim = d;
        for(int i=0;i<dim;i++)
            for(int j=0;j<dim;j++)
                mat[i][j] = 0;
    }
    /// mat = mat * mul
    matrix operator *(const matrix &mul){
        matrix ret = matrix(dim);
        for(int i=0;i<dim;i++){
            for(int j=0;j<dim;j++){
                for(int k=0;k<dim;k++){
                    ret.mat[i][j] += (mat[i][k])*(mul.mat[k][j]) ;
                    ret.mat[i][j] %= MOD ;
                }
            }
```

```
        }
        return ret ;
    }
    matrix operator + (const matrix &add){
        matrix ret = matrix(dim);
        for(int i=0;i<dim;i++){
            for(int j=0;j<dim;j++){
                ret.mat[i][j] = mat[i][j] + add.mat[i][j] ;
                ret.mat[i][j] %= MOD ;
            }
        }
        return ret ;
    }
    matrix operator ^(int p){
        matrix ret = matrix(dim);
        matrix m = *this ;
        for(int i=0;i<dim;i++) ret.mat[i][i] = 1 ; /// identity matrix
        while(p){
            if( p&1 ) ret = ret * m ;
            m = m * m ;
            p >>= 1 ;
        }
        return ret ;
    }
};
```

5.16.9 Matrix Inverse

```
/***
    * In every test case, clear mat first and then do the changes
    * Complexity --> O( min(n,m) * nm )
    * Augmented Matrix -->    [ ORIGINAL_MAT | IDENTITY_MAT ]
    * After elimination -->   [ IDENTITY_MAT | INVERSE_MAT  ]
***/

const int SZ = 105;
const double EPS = 1e-9;
double mat[SZ][SZ+SZ];

void Inverse(int n) {
    for(int mx,r=0,c=0; c<n && r<n; c++) {
        mx = r;
        for(int i=r; i<n; i++) if( abs(mat[i][c]) > abs(mat[mx][c]) ) mx = i;

        if( abs(mat[mx][c]) < EPS ) continue;

        for(int j=c; j<n+n; j++) swap(mat[r][j],mat[mx][j]);

        double mul;
        for(int i=0; i<n; i++)
            if( i!=r ){
                mul = mat[i][c]/mat[r][c];
```

```
                for(int j=c; j<n+n; j++) mat[i][j] -= mul*mat[r][j];
            }
        r++;
    }
    for(int i=0;i<n;i++) {
        for(int j=n;j<n+n;j++) {
            mat[i][j] /= mat[i][i];
        }
    }
}

int main() {
    int n;
    si(n);
    for(int i=0;i<n;i++) {
        mat[i][n+i] = 1;
        for(int j=0;j<n;j++) scanf("%lf",&mat[i][j]);
    }
    Inverse(n);
    for(int i=0;i<n;i++) {
        for(int j=0;j<n;j++) {
            if(j!=0) printf(" ");
            printf("%0.9f",mat[i][n+j]);
        }
        puts("");
    }
    CLR(mat);
    return 0;
}
```

5.16.10 Maximum Xor Subset

```
/***
    The code of the explanation can be found in "Gaussian Related Problem 1.
       cpp"

Problem :   Given n integers, pick a subset of the integers such
            that the xor sum of the subset is maximum.

Solution :

Clearly, if all the input numbers had a different length, the problem
would have a trivial solution: just iterate over the input numbers in
decreasing order by length, choosing each number if and only if XORing
it with the maximum so far increases the maximum, i.e. if and only if
its leading bit is not set in the current maximum.

"""
# Find maximum XOR of input, assuming that all input numbers have
# different length:
let max = 0
for each number n in input (sorted in descending order by length):
```

```
    if max < (max XOR n): let max = (max XOR n)
"""

The tricky part is when the input may contain multiple numbers with the same length,
since then it's not obvious which of them we should choose to include in the XOR.
What we'd like to do is reduce the input list into an equivalent form that doesn't
contain more than one number of the same length.

Conveniently, this is exactly what Gaussian elimination does:
it transforms a list of vectors into another list of vectors which have strictly
decreasing length, as defined above (that is, into a list which is in echelon form),
but which still spans the same linear subspace.

"""
# Preliminary phase: split numbers into buckets by length
for each number x in input:
    let k = length of x
    if k > 0: add x to bucket k

# Gaussian elimination:
for each bucket (in decreasing order by k):
    if this bucket is empty: skip rest of loop
    let x = arbitrary (e.g. first) number in this bucket
    remove x from this bucket
    add x to modified input list

    for each remaining number y in this bucket:
        remove y from this bucket
        let z = y XOR x
        let k = length of z
        if k > 0: add z to bucket k
"""

It's also possible to find the subset of input numbers giving the maximum XOR
using this algorithm: for each modified input number, you just need to somehow keep
track of which original input numbers it was XORed from. This is basically the same
algorithm as using Gaussian elimination to find a matrix inverse, just adapted to bit vectors.


***/
```

5.17 Modular Arithmatic

5.17.1 Bigmod and Modular Inverse

```
/// returns (a^b) % m
ll bigMod(ll a,ll b,ll m){
    ll ret = 1LL;
    a %= m;
    while (b){
        if (b & 1LL) ret = (ret * a) % m;
        a = (a * a) % m;
        b >>= 1LL;
    }
    return ret;
}


/// returns (x,y) of the equation ax + by = gcd(a,b)
PLL extEuclid(ll a,ll b) {
    if(b==0LL) return make_pair(1LL,0LL);
    PLL ret,got;
    got = extEuclid(b,a%b);
    ret = make_pair(got.yy,got.xx-(a/b)*got.yy);
    return ret;
}

/// returns modular invers of a with respect to m
/// inverse exists if and only if a and m are co-prime
ll modularInverse(ll a, ll m){
    ll x, y, inv;
    PLL sol = extEuclid(a,m);
    inv = (sol.xx + m) % m;
    return inv;
}
```

5.17.2 CRT

```
/***
    X = a_1 % m_1
    X = a_2 % m_2
    X = a_3 % m_3

    m_1,m_2,m_3 are pair wise co-prime

    M = m_1*m_2*m_3

    u_i = Modular inverse of (M/m_i) with respect m_i

    X = ( a_1 * (M/m_1) * u_1 + a_2 * (M/m_2) * u_2 + a_3 * (M/m_3) * u_3 ) %
        M
***/
```

5.17.3 Euler Phi

```
/***
    ( a ^ b ) % m = ( a ^ (phi [m] + b % phi [m]) ) % m
    where (b >= phi [m])
```

```
***/
```

5.17.4 Lucas Theorem (Zahin Vai Code)

```cpp
#include <bits/stdtr1c++.h>

#define MAXP 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// Lucas theorem to calculate binomial co-efficients modulo a prime

namespace lc{
    int MOD = 1000000007;
    int fact[MAXP], inv[MAXP];

    /// Call once with the modulo prime
    void init(int prime){
        MOD = prime;
        fact[0] = 1, inv[MOD - 1] = MOD - 1;
        for (int i = 1; i < MOD; i++) fact[i] = ((long long)fact[i - 1] * i) % MOD;
        for (int i = MOD - 2; i >= 0; i--) inv[i] = ((long long)inv[i + 1] * (i + 1)) % MOD;
    }

    inline int count(int n, int k){
        if (k > n) return 0;
        int x = ((long long)inv[n - k] * inv[k]) % MOD;
        return ((long long)x * fact[n]) % MOD;
    }

    /// Lucas theorem, calculates binomial(n, k) modulo MOD, MOD must be a prime
    inline int binomial(long long n, long long k){
        if (k > n) return 0;

        int res = 1;
        k = min(k, n - k);
        while (k && res){
            res = ((long long)res * count(n % MOD, k % MOD)) % MOD;
            n /= MOD, k /= MOD;
        }
        return res;
    }

    /*** Alternate and extended functionalities ***/

    /// Must call init with prime before (Or set lc::MOD = prime)
```

```
    /// Computes (n! / (p ^ (n / p))) % p in O(p log(n)) time, p MUST be a prime
    /// That is, calculating n! without p's powers
    /// For instance factmod(9, 3) = (1 * 2 * 4 * 5 * 2 * 7 * 8 * 1) % 3 = 1
    inline int factmod(long long n, int p){
        int i, res = 1;
        while (n > 1) {
            if ((n / p) & 1) res = ((long long)res * (p - 1)) % p;
            for (i = n % p; i > 1; i--) res = ((long long)res * i) % p;
            n /= p;
        }
        return (res % p);
    }

    inline int expo(int a, int b){
        int res = 1;

        while (b){
            if (b & 1) res = (long long)res * a % MOD;
            a = (long long)a * a % MOD;
            b >>= 1;
        }
        return res;
    }

    /// Trailing zeros of n! in base p, p is a prime
    inline long long fact_ctz(long long n, long long p){
        long long x = p, res = 0;
        while (n >= x){
            res += (n / x);
            x *= p;
        }
        return res;
    }

    /// Calculates binomial(n, k) modulo MOD, MOD must be a prime
    inline int binomial2(long long n, long long k){
        if (k > n) return 0;
        if (fact_ctz(n, MOD) != (fact_ctz(k, MOD) + fact_ctz(n - k, MOD)))
            return 0;
        int a = factmod(n - k, MOD), b = factmod(k, MOD), c = factmod(n, MOD);
        int x = ((long long)expo(a, MOD - 2) * expo(b, MOD - 2)) % MOD;
        return ((long long)x * c) % MOD;
    }
}

int main(){
    lc::init(997);
    printf("%d\n", lc::binomial(10, 5));
    printf("%d\n", lc::binomial(1996, 998));

```

```
    lc::MOD = 10007;
    printf("%d\n", lc::binomial2(10, 5));
    printf("%d\n", lc::binomial2(1996, 998));
    return 0;
}
```

5.17.5 Lucas description and nCr Modulo Composite Number

```
/***
    If we need to find nCr % P where P is a prime but P can be
    less than n or r, we can use Lucas Theorem.

    nCm = ((n_0 C r_0) * (n_1 C r_1) * (n_2 C r_2) * ... * (n_k C r_k)) % P

    Where n_i is the i'th digit in P based representation of n
    and r_i is the i'th digit in P based representation of r

    ** What if P is a composite number? **

    P = (p_0 ^ a_0) * (p_1 ^ a_1) * ... * (p_k ^ a_k)
    where all p_i are prime numbers.

    nCr = (n!)/((r)!*(n-r)!)

    If all a_i are 1, then we can use lucas to find individual mods for
    each p_i and combine those using CRT

    If any a_i is greater than 1,

    Let's Suppose,  n! = (p_i ^ u) * x
                    (n-r)! = (p_i ^ v) * y
                    (r)! = (p_i ^ w) * z
                    (See the code for calculation of x,y,z when
                     n or r has large value)

    Let's suppose p_i ^ a_i = t,
    gcd(t,x) = gcd(t,y) = gcd(t,z) = 1, so, x,y,z will have modular inverse
    with respect to t (see Note 1)

    So, we will find ( x /(y*z) ) % (p_i^a_i) and then multiply the
    result by (p_i ^ s) where s = u - v - w;
    If, s is not smaller than a_i, then the result is 0.

    Then, we will use CRT to combine the result.
    Actually, we don't need Lucas theorem anymore. This technique
    will work for a_i = 1 also.


    *****************Note 1******************
    phi(p^a) = (p^a) - (p^(a-1)) if p is prime

    a ^ phi(p^x) = 1 (mod p^x)  if gcd(a,p) = 1
```

```
    modular inverse of a with respect to p^a is
    a ^ ( phi(p^x) - 1 ) % (p^x)
    ****************************************
***/


/// returns factorail(n) % (p^a) ignoring prime number p
/// can be done using a loop if n is small
/// complexity p^a * log(p^a)

ll fact[MAX]; /// size at least p^a

ll call(ll n,ll p,ll a) {
    ll ret = 1LL;
    ll y,x,m = 1;

    ///m = p^a
    for(ll i=1;i<=a;i++) m *= p;

    fact[0] = 1;
    for(ll i=1;i<=m;i++){
        if(i%p==0) fact[i] = fact[i-1];
        else fact[i] = (fact[i-1]*i)%m;
    }

    while(true){
        if(n==0) break;

        y = n/m;
        ret *= bm(fact[m],y,m);
        ret %= m;

        y = n%m;

        ret *= fact[y];
        ret %= m;
        n /= p;
    }
    return ret;
}
```

5.17.6 Wilson's Theorem

```
/***
    * modInverse of (p-1) modulo p is (p-1)
    * So, (p-1)! = -1 (mod p) if p is a prime
***/
```

5.18 Polynomial Multiplication

5.18.1 FFT Complex-Handwritten

```
/***
    * multiply (7x^2 + 8x^1 + 9x^0) with (6x^1 + 5x^0)
    * ans = (42x^3 + 83x^2 + 94x^1 + 45x^0)
    * A = (9,8,7), B = (5,6)
    * multiply(A,B,res)
    * res will be (45 94 83 42)
*/

/// a + ib
struct cplx {
    double a,b;
    cplx(double a = 0, double b = 0) : a(a), b(b) {}
    const cplx operator + (const cplx &c) const { return cplx( a + c.a, b + c.b ); }
    const cplx operator - (const cplx &c) const { return cplx( a - c.a, b - c.b ); }
    const cplx operator * (const cplx &c) const { return cplx( a * c.a - b * c.b, a * c.b + b * c.a ); }
};

void fft (vector<cplx> &a, bool invert) {
    int n = (int) a.size();
    for (int i=1, j=0; i<n; ++i) {
        int bit = n >> 1;
        for (; j>=bit; bit>>=1)
            j -= bit;
        j += bit;
        if (i < j)
            swap (a[i], a[j]);
    }

    for (int len=2; len<=n; len<<=1) {
        double ang = 2*PI/len * (invert ? -1 : 1);
        cplx wlen (cos(ang), sin(ang));
        for (int i=0; i<n; i+=len) {
            cplx w (1,0);
            for (int j=0; j<len/2; ++j) {
                cplx u = a[i+j],  v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w = w * wlen;
            }
        }
    }
    if (invert)
        for (int i=0; i<n; ++i)
            a[i].a /= n , a[i].b /= n;
}
/// A and B does not change after passing, res can be any vector
/// A==B || B==res || A==res should not create any problem
/// change to long long vector if needed
```

```
49 void multiply (const vector<int> &a, const vector<int> &b, vector<int> &res) {
50     vector<cplx> fa(a.begin(), a.end()),  fb(b.begin(), b.end());
51     size_t n = 1;
52     size_t mx = max( a.size(), b.size() );
53     while( n < mx )  n <<= 1;
54     n <<= 1;
55     fa.resize (n),  fb.resize (n);
56 
57     fft (fa, false),  fft (fb, false);
58     for (size_t i=0; i<n; ++i)
59         fa[i] = fa[i] * fb[i];
60     fft (fa, true);
61     res.resize (n);
62     for (size_t i=0; i<n; ++i)
63         res[i] = int (fa[i].a + 0.5); /// change to ll (fa[i].real() + 0.5) if
              needed
64 
65     /** For Base B multiplication
66     int B = ?, carry = 0;
67     for (size_t i=0; i<n; ++i) {
68         res[i] += carry;
69         carry = res[i] / B;
70         res[i] %= B;
71     }
72     **/
73 
74     /** Removing Leading Zeros
75     while(res.size() && res.back() == 0)
76         res.pop_back();
77     if(res.empty())
78         res.push_back(0);
79     **/
80 }
```

5.18.2 FFT

```
1 /***
2     * multiply (7x^2 + 8x^1 + 9x^0) with (6x^1 + 5x^0)
3     * ans = (42x^3 + 83x^2 + 94x^1 + 45x^0)
4     * A = (9,8,7), B = (5,6)
5     * multiply(A,B,res)
6     * res will be (45 94 83 42)
7 */
8 
9 typedef complex<double> cplx;
10 
11 void fft (vector<cplx> &a, bool invert) {
12     int n = (int) a.size();
13     for (int i=1, j=0; i<n; ++i) {
14         int bit = n >> 1;
15         for (; j>=bit; bit>>=1)
16             j -= bit;
```

```
        j += bit;
        if (i < j)
            swap (a[i], a[j]);
    }

    for (int len=2; len<=n; len<<=1) {
        double ang = 2*PI/len * (invert ? -1 : 1);
        cplx wlen (cos(ang), sin(ang));
        for (int i=0; i<n; i+=len) {
            cplx w (1);
            for (int j=0; j<len/2; ++j) {
                cplx u = a[i+j],  v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert)
        for (int i=0; i<n; ++i)
            a[i] /= n;
}
/// A and B does not change after passing, res can be any vector
/// A==B || B==res || A==res should not create any problem
/// change to long long vector if needed
void multiply (const vector<int> &a, const vector<int> &b, vector<int> &res) {
    vector<cplx> fa(a.begin(), a.end()),  fb(b.begin(), b.end());
    size_t n = 1;
    size_t mx = max( a.size(), b.size() );
    while( n < mx )  n <<= 1;
    n <<= 1;
    fa.resize (n),  fb.resize (n);

    fft (fa, false),  fft (fb, false);
    for (size_t i=0; i<n; ++i)
        fa[i] *= fb[i];
    fft (fa, true);
    res.resize (n);
    for (size_t i=0; i<n; ++i)
        res[i] = int (fa[i].real() + 0.5); /// change to ll (fa[i].real() +
            0.5) if needed

    /** For Base B multiplication
    int B = ?, carry = 0;
    for (size_t i=0; i<n; ++i) {
        res[i] += carry;
        carry = res[i] / B;
        res[i] %= B;
    }
    **/

```

```
    /** Removing Leading Zeros
    while(res.size() && res.back() == 0)
        res.pop_back();
    if(res.empty())
        res.push_back(0);
    **/
}
```

5.18.3 Karatsuba

```
/***
    multiply (7x^2 + 8x^1 + 9x^0) with (6x^1 + 5x^0)
    ans = (42x^3 + 83x^2 + 94x^1 + 45x^0)
    A = [7,8,9], B = [6,5], n = 3, m = 2
    multiply(A,B,res)
    res will be (42 83 94 45)
***/
#define MAX 200000*4 /// Must be a power of 2 (Not really actually, 4*MAX suffices)
#define MOD ?
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

long long ptr = 0;
long long temp[128];
long long buffer[MAX * 6];
const long long INF = 8000000000000000000LL;

void karatsuba(int n, long long *a, long long *b, long long *res){ /// n is a power of 2
    int i, j, s;
    if (n < 33){ /// Reduce recursive calls by setting a threshold
        for (i = 0; i < (n + n); i++) temp[i] = 0;
        for (i = 0; i < n; i++){
            if (a[i]){
                for (j = 0; j < n; j++){
                    temp[i + j] += (a[i] * b[j]);
                    if (temp[i + j] > INF) temp[i + j] %= MOD;
                }
            }
        }
        for (i = 0; i < (n + n); i++) res[i] = temp[i] % MOD;
        return;
    }

    s = n >> 1;
    karatsuba(s, a, b, res);
    karatsuba(s, a + s, b + s, res + n);
    long long *x = buffer + ptr, *y = buffer + ptr + s, *z = buffer + ptr + s + s;

    ptr += (s + s + n);
    for (i = 0; i < s; i++){
```

```
        x[i] = a[i] + a[i + s], y[i] = b[i] + b[i + s];
        if (x[i] >= MOD) x[i] -= MOD;
        if (y[i] >= MOD) y[i] -= MOD;
    }

    karatsuba(s, x, y, z);
    for (i = 0; i < n; i++) z[i] -= (res[i] + res[i + n] - MOD);
    for (i = 0; i < n; i++) res[i + s] = (res[i + s] + z[i] + MOD) % MOD;
    ptr -= (s + s + n);
}

/// multiplies two polynomial a( degree n-1 ) and b( degree m-1 ) and returns the result modulo MOD in a
/// returns (the degree of the multiplied polynomial + 1)
/// note that a and b are changed in the process
int mul(int n, long long *a, int m, long long *b){
    int i, r, c = (n < m ? n : m), d = (n > m ? n : m);
    long long *res = buffer + ptr;
    r = 1 << (32 - __builtin_clz(d) - (__builtin_popcount(d) == 1));
    for (i = d; i < r; i++) a[i] = b[i] = 0;
    for (i = c; i < d && n < m; i++) a[i] = 0;
    for (i = c; i < d && m < n; i++) b[i] = 0;

    ptr += (r << 1), karatsuba(r, a, b, res), ptr -= (r << 1);
    for (i = 0; i < (r << 1); i++) a[i] = res[i];
    return (n + m - 1);
}


/// For a polynomial of degree D ,
/// coeff[0] will contain the coefficient of x^D
/// coeff[D] will contain the coefficient of x^0
/// coeff[D-i] will contain the coefficient of x^i
```

5.18.4 Notes

```
/***
    * Karatsuba, FFT, NTT they all do the same thing.
    * Karatsuba, NTT uses LL where fft uses doubles.
    * So FFT might face loss of precision but karatsuba/NTT won't face that ever.
    * Karatsuba,NTT returns the coefficients%MOD. FFT returns the coefficients as they are.
    * Karatsuba runs O(n^(log2(3)), FFT runs in O(nlog2(n)), NTT runs in O(nlog2(n)). But NTT is slower than FFT due to constant factors.
    * You cannot use NTT for arbitrary mods. NTT works for special mods only. So for the general case, NTT is not an option.

    *** Use karatsuba if you're sure time limit is okay for it. Otherwise go to FFT.

***/
```

5.19 Prime Numbers

5.19.1 Miller Rabin and Pollard Rho

```cpp
#include <bits/stdtr1c++.h>

using namespace std;

namespace rho{
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

    const int MAXP = 1000010;
    const int BASE[] = {2, 450775, 1795265022, 9780504, 28178, 9375, 325};

    ll seq[MAXP];
    int primes[MAXP], spf[MAXP];

    inline ll mod_add(ll x, ll y, ll m){
        return (x += y) < m ? x : x - m;
    }

    inline ll mod_mul(ll x, ll y, ll m){
        ll res = x * y - (ll)((ld)x * y / m + 0.5) * m;
        return res < 0 ? res + m : res;
    }

    inline ll mod_pow(ll x, ll n, ll m){
        ll res = 1 % m;
        for (; n; n >>= 1){
            if (n & 1) res = mod_mul(res, x, m);
            x = mod_mul(x, x, m);
        }
        return res;
    }


    /// O(k logn logn logn), k = number of rounds performed
    inline bool miller_rabin(ll n){
        if (n <= 2 || (n & 1 ^ 1)) return (n == 2);
        if (n < MAXP) return spf[n] == n;

        ll c, d, s = 0, r = n - 1;
        for (; !(r & 1); r >>= 1, s++) {}

        /// each iteration is a round
        for (int i = 0; primes[i] < n && primes[i] < 32; i++){
            c = mod_pow(primes[i], r, n);
            for (int j = 0; j < s; j++){
                d = mod_mul(c, c, n);
                if (d == 1 && c != 1 && c != (n - 1)) return false;
                c = d;
            }
```

```
            if (c != 1) return false;
        }
        return true;
    }

    inline void init(){
        int i, j, k, cnt = 0;
        for (i = 2; i < MAXP; i++){
            if (!spf[i]) primes[cnt++] = spf[i] = i;
            for (j = 0, k; (k = i * primes[j]) < MAXP; j++){
                spf[k] = primes[j];
                if(spf[i] == spf[k]) break;
            }
        }
    }

    /// Expected complexity O(n^(1/4))
    ll pollard_rho(ll n){
        while (1){
            ll x = rand() % n, y = x, c = rand() % n, u = 1, v, t = 0;
            ll *px = seq, *py = seq;

            while (1){
                *py++ = y = mod_add(mod_mul(y, y, n), c, n);
                *py++ = y = mod_add(mod_mul(y, y, n), c, n);
                if((x = *px++) == y) break;

                v = u;
                u = mod_mul(u, abs(y - x), n);

                if (!u) return __gcd(v, n);
                if (++t == 32){
                    t = 0;
                    if ((u = __gcd(u, n)) > 1 && u < n) return u;
                }
            }

            if (t && (u = __gcd(u, n)) > 1 && u < n) return u;
        }
    }

    vector <ll> factorize(ll n){
        if (n == 1) return vector <ll>();
        if (miller_rabin(n)) return vector<ll> {n};

        vector <ll> v, w;
        while (n > 1 && n < MAXP){
            v.push_back(spf[n]);
            n /= spf[n];
        }

```

```
        if (n >= MAXP) {
            ll x = pollard_rho(n);
            v = factorize(x);
            w = factorize(n / x);
            v.insert(v.end(), w.begin(), w.end());
        }

        sort(v.begin(), v.end());
        return v;
    }
}

int main(){
    rho::init();
    vector <ll> v = rho::factorize(n);
    return 0;
}
```

5.19.2 Prime Counting Functions

```
/// http://acganesh.com/blog/2016/12/23/prime-counting
/// http://mathworld.wolfram.com/LegendresFormula.html

namespace pcf{
    ///   Prime-Counting Function
    ///   initialize once by calling init()
    ///   Legendre(n) and Lehmer(n) returns number of primes less than or
       equal to m
    ///   Lehmer(n) is faster

    #define MAXN 1000010 /// initial sieve limit
    #define MAX_PRIMES 1000010 /// max size of the prime array for sieve
    #define PHI_N 100000 ///
    #define PHI_K 100

    int len = 0; /// total number of primes generated by sieve
    int primes[MAX_PRIMES];
    int counter[MAXN]; /// counter[m] --> number of primes <= i
    int phi_dp[PHI_N][PHI_K]; /// precal of phi(n,k)


    bitset <MAXN> isComp;
    void Sieve(int N){
        int  i,j,sq = sqrt(N);
        isComp[1] = true;
        for(i=4;i<=N;i+=2) isComp[i] = true;
        for(i=3;i<=sq;i+=2){
            if(!isComp[i]){
                for(j=i*i;j<=N;j+=i+i) isComp[j] = 1;
            }
        }
        for (i = 1; i <= N; i++){
```

```
32              if (!isComp[i]) primes[len++] = i;
33              counter[i] = len;
34          }
35      }
36
37      void init(){
38          Sieve(MAXN - 1);
39          /// precalculation of phi upto size (PHI_N,PHI_K)
40          int k , n , res;
41          for(n = 0; n < PHI_N; n++) phi_dp[n][0] = n;
42          for (k = 1; k < PHI_K; k++){
43              for (n = 0; n < PHI_N; n++){
44                  phi_dp[n][k] = phi_dp[n][k - 1] - phi_dp[n / primes[k - 1]][k
                        - 1];
45              }
46          }
47      }
48
49      /// returns number of integers less or equal n which are
50      /// not divisible by any of the first k primes
51      /// recurrence --> phi( n , k ) = phi( n , k-1 ) - phi( n / p_k , k-1)
52      long long phi(long long n, int k){
53          if (n < PHI_N && k < PHI_K) return phi_dp[n][k];
54          if (k == 1) return ((++n) >> 1);
55          if (primes[k - 1] >= n) return 1;
56          return phi(n, k - 1) - phi(n / primes[k - 1], k - 1);
57      }
58
59      long long Legendre(long long n){
60          if (n < MAXN) return counter[n];
61
62          int lim = sqrt(n) + 1;
63          int k = upper_bound(primes, primes + len, lim) - primes;
64          return phi(n, k) + (k - 1);
65      }
66
67      long long Lehmer(long long n){
68          if (n < MAXN) return counter[n];
69
70          long long w , res = 0;
71          int i, j, a, b, c, lim;
72          b = sqrt(n), c = Lehmer(cbrt(n)), a = Lehmer(sqrt(b)), b = Lehmer(b);
73          res = phi(n, a) + (((b + a - 2) * (b - a + 1)) >> 1);
74
75          for (i = a; i < b; i++){
76              w = n / primes[i];
77              lim = Lehmer(sqrt(w)), res -= Lehmer(w);
78
79              if (i <= c){
80                  for (j = i; j < lim; j++){
81                      res += j;
```

```
                    res -= Lehmer(w / primes[j]);
                }
            }
        }
        return res;
    }
}
```

5.19.3 Prime Power Factorization

```
/***
    A call to generatePPF(int N) will generate the prime power
    factorization of numbers upto N
    TLE khele vector ke array diye replace korte hobe
    Overall complexity is almost n(log n)
***/

vector <PII> factor[MAX];
/// factor[x] contains (p,i) if prime p divides x i times
bool isComp[MAX]; /// true if a number is composite
int lp[MAX]; /// least prime factor

void Sieve(int N){
    int  i,j,sq = sqrt(N);
    for(i=1;i<=N;i++) lp[i] = i;
    for(i=4;i<=N;i+=2) isComp[i] = true , lp[i] = 2;
    for(i=3;i<=sq;i+=2){
        if(isComp[i]) continue;
        for(j=i*i;j<=N;j+=i+i) isComp[j] = 1 , lp[j] = min(lp[j],i);
    }
}

void generatePPF(int N) {
    Sieve(N);
    int now;
    vector <int> temp;
    for(int num=2;num<=N;num++) {
        now = num;
        temp.clear();
        while(lp[now]!=1){
            temp.pb(lp[now]);
            now = now/lp[now];
        }
        int cnt = 1;
        for(int i=1;i<temp.size();i++){
            if(temp[i]==temp[i-1]) cnt++;
            else{
                factor[num].push_back(mp(temp[i-1],cnt));
                cnt = 1;
            }
        }
        factor[num].push_back(mp(temp[temp.size()-1],cnt));
```

```
    }
}

/***
    Another Way
    Takes more time than the previous but soto code
***/

vector <PII> factor[MAX];
/// factor[x] contains (p,i) if prime p divides x i times
bool isComp[MAX]; /// true if a number is composite

void generatePPF(int N) {
    int i,j,cnt,tmp,x;
    for(i=2;i<=N;i++) {
        if(!isComp[i]) {
            for(j=i;j<=N;j+=i) {
                isComp[j] = true;
                tmp = j , cnt = 0;
                while(true) {
                    x = tmp/i;
                    if(x*i!=tmp) break;
                    tmp = x;
                    cnt++;
                }
                factor[j].pb({i,cnt});
            }
        }
    }
}
```

5.19.4 Sieve

```
bool isComp[MAX];  /// ara[i] is true if i is composite
vector <int> primes;

void Sieve(int N){
    int  i,j,sq = sqrt(N);
    for(i=4;i<=N;i+=2) isComp[i] = true;
    for(i=3;i<=sq;i+=2){
        if(!isComp[i]){
            for(j=i*i;j<=N;j+=i+i) isComp[j] = 1;
        }
    }
    for(i=2;i<=N;i++) if(!isComp[i]) primes.pb(i);
}
```

6 Misc

6.1 Fast IO

```
/// For windows system use getchar() in place of getchar_unlocked()
```

```
inline int RI() {
    int  ret = 0, flag = 1,ip = getchar_unlocked();
    for(; ip < 48 || ip > 57; ip = getchar_unlocked()) {
        if(ip == 45) {
            flag = -1;
            ip = getchar_unlocked();
            break;
        }
    }
    for(; ip > 47 && ip < 58; ip = getchar_unlocked())
        ret = ret * 10 + ip - 48 ;
    return flag * ret;
}

/// scanning syntax
int n = RI();
```

6.2 GP Hash Table

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
gp_hash_table<int, int> table;


/*** Defeating Anti-Hash tests :
    One weakness of hash tables is that mean people can find
    hash collisions offline and blow up the complexity of your hashmap.
    In my opinion, the easiest way of solving this is below.
    There's no need to define your own custom hash function.
***/

const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().
    count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<key, int, chash> table;
```

6.3 Histogram Problem

```
/***
    Finds the largest rectangular area possible in a given histogram where
    the largest rectangle can be made of a number of contiguous bars.
***/


int solve(vector <int> &V) {
    V.push_back(0);
    int n = V.size();
    int ans = 0;
    stack <int> S;
    for(int i=0;i<V.size();i++) {
```

```
        int y = V[i];
        while(!S.empty()) {
            int x = V[S.top()];
            if(x >= y) {
                S.pop();
                if(S.empty()) ans = max(ans,i*x);
                else ans = max(ans,(i-S.top()-1)*x);
            }
            else break;
        }
        S.push(i);
    }
    V.pop_back();
    return ans;
}
```

6.4 Knight Distance in Infinite Chessboard

```
/// Minimum number of knight moves to reach (x,y) from (0,0) in infinite
    chessboard
/// x or y can be negative
/// minimum move to reach from (x1,y1) to (x2,y2) = knight_move( x1-x2 , y1-y2
    )

int knight_move(int x, int y){
    int a, b, z, c, d;
    x = abs(x), y = abs(y);
    if (x < y) a = x, x = y, y = a;
    if (x == 2 && y == 2) return 4;
    if (x == 1 && y == 0) return 3;

    if (y == 0 || (y << 1) < x){
        c = y & 1;
        a = x - (c << 1), b = a & 3;
        return ((a - b) >> 1) + b + c;
    }
    else{
        d = x - ((x - y) >> 1);
        z = ((d % 3) != 0), c = (x - y) & 1;
        return ((d / 3) * 2) + c + (z * 2 * (1 - c));
    }
}
```

6.5 Number of Trees Given n

```
/***
    * Given n nodes numbered from 1 to n , number of different unrooted trees
       =n^(n-2)
    * Given n nodes numbered from 1 to n , number of different rooted
      trees = n * n^(n-2) = n^(n-1)
***/
```

6.6 Output Formatting

```
/***
    double pi = 3.14;
    cout << setprecision(5) << fixed << pi << endl;
    --> 3.14000

    cout << setw(5) << 20;
    --> "   20"

    cout << setiosflags(ios::left) << setw(5) << 20 << endl;
    --> "20   "

    cout << setiosflags(ios::right) << setw(5) << 20 << endl;
    --> "   20"

    cout << setfill('0'); fills the blanks created by setw(w) function


    Printing Date :

    void showDate(int m, int d, int y) {
        cout << setfill('0');
        cout << setw(2) << m << '/' << setw(2) << d << '/' << setw(4) << y <<
           endl;
    }


    Base Conversion :
    int x = 2324534;
    cout << dec << x << endl;
    cout << oct << x << endl;
    cout << hex << x << endl;
***/
```

6.7 Precision

```
/***
    https://www.quora.com/What-are-the-differences-between-the-double-and-
       float-data-types

    Float(32 bits):
        sign bit  : 1 indicates negative, 0 indicates positive or zero
        exponent  : 8 bits
                    x is stored as (x + 127) (example : -10 --> 117)
        precision : 23 bits
                    upto 7-8 significant decimal digits will be accurate

    Double(64 bits):
        sign bit  : 1 indicates negative, 0 indicates positive or zero
        exponent  : 11 bits
                    x is stored as (x + 1023) (example : -5 --> 1018)
```

```
        precision : 52 bits
                    upto 15-16 significant decimal digits will be accurate

    Long Double(80 bits):
        sign bit  : 1 indicates negative, 0 indicates positive or zero
        exponent  : 15 bits
                    x is stored as (x + 16383) (example : 3 --> 16386)
        precision : 64 bits
                    upto 19-20 significant decimal digits will be accurate

    If some data type gives x bits precision, it means the first x bits
    of the number will be exact if the number is contained by that data type
***/
```

6.8 Random Number

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

unsigned int num = rng(); //returns 32 bit unsigned integer

shuffle(V.begin(),V.end(),rng); // A way of random shuffling
```

6.9 Simulated Annealing

```
/***
    s   --> current state
    ns  --> neighboring state of s state

    F(s)--> evaluates some state

    e   --> F(s)
    ne  --> F(ns)

    T   --> Temperature

    P(e,ne,T) --> Probability of moving to the neighboring state ns

    When we are trying to find global maximum :
    if ne >= e :
        P(e,ne,T) = 1
    else :
        P(e,ne,T) = exp((ne-e)/T)

***/

/*** Pseudo Code ***/
    T = sth around 1e9
    mul = 0.997
    for it = 0 to trough lim :
        ns = neighbour(s) ( Pick a random neighbour )
        if P(e,ne,T) >= random(0,1) :
            s = ns
```

```
        if T >= 1e-20:
            T *= mul

/***
    Suppose you can afford LIM iterations given the time limit.
    It's better Run the algorithm several(1<x<10) times, everytime with different
    temperature and the number of iterations each time will be (LIM/x)
***/


/***
    Selecting the parameters

    *  An essential requirement for the neighbour() function is that the diameter
       of the search graph must be small. In the traveling salesman problem,
       the search space for n = 20 cities has n! = 2,432,902,008,176,640,000 states;
       yet the neighbor generator function that swaps two consecutive cities can get
       from any state (tour) to any other state in at most {n(n-1)/2 = 190 steps

    * In TSP, it's better to swap adjacent elements as it does not do any drastic change
      to the current state
***/
```

6.10 Walsh Hadamard Transformation

```
const int MAX = (1<<N) + 10; /// maximum value of n
//const int INF = 2000000000;
//const int MOD = 1000000007;

/***
    * A call to fwht::convolution(n, A, B, operator) returns a vector V of size n
    * A and B are arrays of size n
    * n must be a power of 2
    * For i = 0 to n - 1, j = 0 to n - 1
      V[i operator j] += A[i] * B[j]
***/

/// Fast Walsh-Hadamard Transformation in n log n
namespace fwht{
    const int OR = 0;
    const int AND = 1;
    const int XOR = 2;

    long long P1[MAX], P2[MAX]; /// Adjust the MAX accordingly

```

```
    void walsh_transform(long long* ar, int n, int flag = XOR){
        if (n == 0) return;
        int i, m = n / 2;
        walsh_transform(ar, m, flag);
        walsh_transform(ar + m, m, flag);

        for (i = 0; i < m; i++){ /// Don't forget modulo if required
            long long x = ar[i], y = ar[i + m];
            if (flag == OR) ar[i] = x, ar[i + m] = x + y;
            if (flag == AND) ar[i] = x + y, ar[i + m] = y;
            if (flag == XOR) ar[i] = x + y, ar[i + m] = x - y;
        }
    }

    void inverse_walsh_transform(long long* ar, int n, int flag = XOR){
        if (n == 0) return;

        int i, m = n / 2;
        inverse_walsh_transform(ar, m, flag);
        inverse_walsh_transform(ar + m, m, flag);

        for (i = 0; i < m; i++){ /// Don't forget modulo if required
            long long x = ar[i], y = ar[i + m];
            if (flag == OR) ar[i] = x, ar[i + m] = y - x;
            if (flag == AND) ar[i] = x - y, ar[i + m] = y;
            if (flag == XOR) ar[i] = (x + y) >> 1, ar[i + m] = (x - y) >> 1;
                /// Modular inverse if required here
        }
    }

    vector <long long> convolution(int n, long long* A, long long* B, int flag
        = XOR){
        assert(__builtin_popcount(n) == 1); /// n must be a power of 2
        for (int i = 0; i < n; i++) P1[i] = A[i];
        for (int i = 0; i < n; i++) P2[i] = B[i];

        walsh_transform(P1, n, flag);
        walsh_transform(P2, n, flag);
        for (int i = 0; i < n; i++) P1[i] = P1[i] * P2[i];
        inverse_walsh_transform(P1, n, flag);
        return vector<long long> (P1, P1 + n);
    }
}
```

7 String

7.1 Aho Corasick(Using Array)

```
/***
    Given n patterns and a text T, for every pattern
    you have to output the number of times that pattern
    appears in the text.
```

```
5
6       * Don't forget to call build() after adding all the patterns
7         to the Aho Corasick trie.
8
9       * ans[i] contains the number of occurrences of the i'th pattern
10
11      * link[x] = y means there is a suffix link from node x to node y
12
13      * out_link[x] = y means we can go from x to y using the suffix links
14        suppose the path is as follows : x , a , b, c, ..., y
15        No pattern ends in node a, b, c, ... but some pattern ends at node y.
16
17      * After a call to build(), the trie becomes a DAG(except node 0)
18        next[x][c] = y means if we are currently at node x and the character
19        c arrives, we will go to node y.
20
21
22      * Suppose sum of the length of the characters is N. Text length is also
23        at most N. If all the patterns are unique, total number of occurrences
24        of all the patterns will not be more than " N sqrt(N) ".
25  ***/
26
27  #include <bits/stdc++.h>
28
29  using namespace std;
30
31  const int N = ?; /// Total number of characters in pattern
32  const int A = ?; /// Alphabet size
33
34  struct AC {
35      int nd, pt;
36
37      int next[N][A], link[N], out_link[N], cnt[N], ans[N];
38      vector <int> ed[N], out[N];
39
40      AC(): nd(0), pt(0) { node(); }
41
42      int node() {
43          memset(next[nd], 0, sizeof next[nd]);
44          link[nd] = out_link[nd] = cnt[nd] = 0;
45          ed[nd].clear(), out[nd].clear();
46          return nd++;
47      }
48
49      void clear() {
50          nd = pt = 0;
51          node();
52      }
53
54      inline int get(char c) { return c - 'a'; }
55
```

```
56      void insert(const string &T) {
57          int u = 0;
58          for (char c : T) {
59              if (!next[u][get(c)]) next[u][get(c)] = node();
60              u = next[u][get(c)];
61          }
62          ans[pt] = 0;
63          out[u].push_back(pt++);
64      }
65
66      void build() {
67          queue <int> q;
68          for (q.push(0); !q.empty(); ) {
69              int u = q.front();
70              q.pop();
71              for (int c = 0; c < A; ++c) {
72                  int v = next[u][c];
73                  if (!v) next[u][c] = next[link[u]][c];
74                  else {
75                      link[v] = u ? next[link[u]][c] : 0;
76                      out_link[v] = out[link[v]].empty() ? out_link[link[v]] :
                            link[v];
77                      ed[link[v]].push_back(v);
78                      q.push(v);
79                  }
80              }
81          }
82      }
83
84      void dfs(int s) {
85          for(int x : ed[s]) dfs(x), cnt[s] += cnt[x];
86          for(int e : out[s]) ans[e] = cnt[s];
87      }
88
89      void traverse(const string &S) {
90          int u = 0;
91          for (char c : S) {
92              u = next[u][get(c)];
93              cnt[u]++;
94          }
95          dfs(0);
96      }
97  };
98
99  char str[1000010], pat[505];
100
101 int main() {
102 //    freopen("in.txt","r",stdin);
103     AC aho;
104     int t,T;
105     scanf("%d",&T);
```

```
    for(int t=1;t<=T;t++) {
        int n;
        scanf("%d",&n);
        scanf("%s",str);
        for(int i=1;i<=n;i++) {
            scanf("%s",pat);
            aho.insert(pat);
        }
        aho.build();
        aho.traverse(str);
        printf("Case %d:\n",t);
        for(int i=0;i<n;i++) {
            printf("%d\n",aho.ans[i]);
        }
        aho.clear();
    }
    return 0;
}
```

7.2 Aho Corasick(Using Vector)

```
/***
    Given n patterns and a text T, for every pattern
    you have to output the number of times that pattern
    appears in the text.

    * Don't forget to call build() after adding all the patterns
      to the Aho Corasick trie.

    * ans[i] contains the number of occurrences of the i'th pattern

    * link[x] = y means there is a suffix link from node x to node y

    * out_link[x] = y means we can go from x to y using the suffix links
      suppose the path is as follows : x , a , b, c, ..., y
      No pattern ends in node a, b, c, ... but some pattern ends at node y.

    * After a call to build(), the trie becomes a DAG(except node 0)
      next[x][c] = y means if we are currently at node x and the character
      c arrives, we will go to node y.


    * Suppose sum of the length of the characters is N. Text length is also
      at most N. If all the patterns are unique, total number of occurrences
      of all the patterns will not be more than " N sqrt(N) ".
***/

#include <bits/stdc++.h>

using namespace std;

struct AC {
```

```
32      int N, P;
33      const int A = 26;
34      vector < vector <int> > next;
35      vector <int> link, out_link;
36      vector < vector <int> > out;
37      vector < int > cnt;
38      vector < vector <int> > ed;
39      vector <int> ans;
40
41      AC(): N(0), P(0) { node(); }
42
43      int node() {
44          next.emplace_back(A, 0);
45          link.emplace_back(0);
46          out.emplace_back(0);
47          out_link.emplace_back(0);
48          cnt.emplace_back(0);
49          ed.emplace_back(0);
50          return N++;
51      }
52
53      void clear() {
54          next.clear(), link.clear(), out.clear() , out_link.clear(), ed.clear()
                ;
55          cnt.clear(), ans.clear();
56          N = P = 0;
57          node();
58      }
59
60      inline int get(char c) { return c - 'a'; }
61
62      void insert(const string &T) {
63          int u = 0;
64          for (char c : T) {
65              if (!next[u][get(c)]) next[u][get(c)] = node();
66              u = next[u][get(c)];
67          }
68          out[u].push_back(P);
69          ans.push_back(0);
70          P++;
71      }
72
73      void build() {
74          queue <int> q;
75          for (q.push(0); !q.empty(); ) {
76              int u = q.front();
77              q.pop();
78              for (int c = 0; c < A; ++c) {
79                  int v = next[u][c];
80                  if (!v) next[u][c] = next[link[u]][c];
81                  else {
```

```
                    link[v] = u ? next[link[u]][c] : 0;
                    out_link[v] = out[link[v]].empty() ? out_link[link[v]] :
                        link[v];
                    ed[link[v]].push_back(v);
                    q.push(v);
                }
            }
        }
    }

    void dfs(int s) {
        for(int x : ed[s]) dfs(x), cnt[s] += cnt[x];
        for(int e : out[s]) ans[e] = cnt[s];
    }

    void traverse(const string &S) {
        int u = 0;
        for (char c : S) {
            u = next[u][get(c)];
            cnt[u]++;
        }
        dfs(0);
    }
};

char str[1000010], pat[505];

int main() {
//    freopen("in.txt","r",stdin);

    AC aho;
    int t,T;
    scanf("%d",&T);
    for(int t=1;t<=T;t++) {
        int n;
        scanf("%d",&n);
        scanf("%s",str);
        for(int i=1;i<=n;i++) {
            scanf("%s",pat);
            aho.insert(pat);
        }
        aho.build();
        aho.traverse(str);
        printf("Case %d:\n",t);
        for(int i=0;i<n;i++) {
            printf("%d\n",aho.ans[i]);
        }
        aho.clear();
    }
    return 0;
}
```

7.3 Double Hashing

```
/// l, r are 0 based

struct simplehash{
    int len;
    ll base, mod;
    vector <int> P, H, R;

    simplehash(){}
    simplehash(const string &str, ll b, ll m){
        base = b, mod = m, len = str.size();
        P.resize(len + 3, 1), H.resize(len + 3, 0), R.resize(len + 3, 0);

        for (int i = 1; i <= len; i++) P[i] = (P[i - 1] * base) % mod;
        for (int i = 1; i <= len; i++) H[i] = (H[i - 1] * base + str[i - 1] +
            1007) % mod;
        for (int i = len; i >= 1; i--) R[i] = (R[i + 1] * base + str[i - 1] +
            1007) % mod;
    }

    inline int range_hash(int l, int r) {
        int hashval = H[r + 1] - ((ll)P[r - l + 1] * H[l] % mod);
        return (hashval < 0 ? hashval + mod : hashval);
    }

    inline int reverse_hash(int l, int r) {
        int hashval = R[l + 1] - ((ll)P[r - l + 1] * R[r + 2] % mod);
        return (hashval < 0 ? hashval + mod : hashval);
    }
};

struct stringhash{
    simplehash sh1, sh2;
    stringhash(){}
    // character array can be sent as parameter too
    stringhash(const string &str){
        sh1 = simplehash(str, 1949313259, 2091573227);
        sh2 = simplehash(str, 1997293877, 2117566807);
    }

    inline ll range_hash(int l, int r){
        return ((ll)sh1.range_hash(l, r) << 32) ^ sh2.range_hash(l, r);
    }

    inline ll reverse_hash(int l, int r){
        return ((ll)sh1.reverse_hash(l, r) << 32) ^ sh2.reverse_hash(l, r);
    }
};
```

7.4 Dynamic Aho Corasick(Using Vector)

```cpp
/***
    Suppose you have a list of strings(initially empty)
    There will be some queries.
    Query --> t s

    If t == 1, add s to your list

    else, print the total occurrences of all the
    strings of your list in s.
***/

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

const int MAX = ?; /// maximum possible size of an input string

struct AC {
    int N, P;
    const int A = 26;
    vector < vector <int> > next;
    vector <int> link, out_link;
    vector < vector <int> > out;
    vector < int > prefSum;
    vector <string> pat;

    AC(): N(0), P(0) { node(); }

    int node() {
        next.emplace_back(A, 0);
        link.emplace_back(0);
        out.emplace_back(0);
        out_link.emplace_back(0);
        prefSum.emplace_back(0);
        return N++;
    }

    void clear() {
        next.clear(), link.clear(), out.clear() , out_link.clear();
        prefSum.clear();
        pat.clear();
        N = P = 0;
        node();
    }

    inline int get(char c) { return c - 'a'; }

    void insert(const string &T) {
        int u = 0;
        for (char c : T) {
            if (!next[u][get(c)]) next[u][get(c)] = node();
            u = next[u][get(c)];
```

```
52          }
53          out[u].push_back(P);
54          pat.push_back(T);
55          P++;
56      }
57
58      void build() {
59          queue <int> q;
60          for (q.push(0); !q.empty(); ) {
61              int u = q.front();
62              q.pop();
63              for (int c = 0; c < A; ++c) {
64                  int v = next[u][c];
65                  if (!v) next[u][c] = next[link[u]][c];
66                  else {
67                      link[v] = u ? next[link[u]][c] : 0;
68                      out_link[v] = out[link[v]].empty() ? out_link[link[v]] :
                            link[v];
69                      prefSum[v] = prefSum[link[v]] + out[v].size();
70                      q.push(v);
71                  }
72              }
73          }
74      }
75
76      long long query(const string &S) {
77          long long ret = 0;
78          int u = 0;
79          for (char c : S) {
80              u = next[u][get(c)];
81              ret += prefSum[u];
82          }
83          return ret;
84      }
85  };
86
87  const int LOG = 23; /// log2(Total number of patterns)
88
89  ///dynamic aho-corasick
90  struct DAC {
91      AC aho[LOG];
92
93      void insert(const string &S) {
94          int pos;
95          for(pos = 0; pos < LOG; pos++) if( aho[pos].P == 0 ) break;
96
97          aho[pos].insert(S);
98          for(int i=0;i<pos;i++) {
99              for(string &cur : aho[i].pat) aho[pos].insert(cur);
100             aho[i].clear();
101         }
```

```
        aho[pos].build();
    }

    long long query(const string &S) {
        long long ret = 0;
        for(int i=0;i<LOG;i++)
            ret += aho[i].query(S);
        return ret;
    }
};

char str[MAX];

int main() {
    int n, x;
    scanf("%d",&n);
    DAC aho;
    for(int i=0;i<n;i++) {
        scanf("%d %s",&x,str);
        if(x == 1) {
            aho.insert(str);
        }
        else {
            printf("%lld\n",aho.query(str));
            fflush(stdout);
        }
    }
    return 0;
}
```

7.5 Finding Maimum and Minimum Xor Match

```
/// Find an integer x in the trie such than n^x is minimized
int FindMinMatch(int n,int nob){
    int ret = 0;
    int cur = root , to;
    for(int i=nob ; i>=0 ; i--){
        ret <<= 1;
        to = checkBit(n,i);
        if(nxt[cur][to]){
            ret += to;
            cur = nxt[cur][to];
        }
        else{
            ret += (!to);
            cur = nxt[cur][!to];
        }
    }
    return ret;
}

/// Find an integer x in the trie such than n^x is maximized
```

```
int FindMaxMatch(int n,int nob){
    int ret = 0;
    int cur = root , to;
    for(int i=nob ; i>=0 ; i--){
        ret <<= 1;
        to = checkBit(n,i);
        if(nxt[cur][!to]){
            ret += (!to);
            cur = nxt[cur][!to];
        }
        else{
            ret += (to);
            cur = nxt[cur][to];
        }
    }
    return ret;
}
```

7.6 KMP

```
/***
    * Complexity = O(P+S)
    * Searches pat in str

    * We can avoid the matcher by calculating the prefix function for the
       string = P + "#" + S
    * To find the smallest period of a string ,
      If ( len % ( len \96 pref[len-1] ) ) == 0
      then the string hasa period of length ( len \96 pref[len-1] )
      and it occurs len / ( len \96 pref[len-1] ) times


***/

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

const int MAX = 100010;

char str[MAX],pat[MAX];
int pref[MAX];
int match[MAX];

/// pref[i] =   length of the longest suffix which is also
/// a proper prefix of the original string
void prefixFunction(int P) {
    int j=0;
    for(int i=1; i<P; i++) {
        while(true) {
            if(pat[i]==pat[j]) {
                j = pref[i] = j+1; break;
            }
```

```
            else {
                if(j==0) {
                    pref[i] = 0; break;
                }
                else j = pref[j-1];
            }
        }
    }
}

void KMPMatcher(int S) {
    int j = 0;
    for(int i=0; i<S; i++) {
        while(true) {
            if(str[i]==pat[j]) {
                j = match[i] = j+1; break;
            }
            else {
                if(j==0) {
                    match[i] = 0; break;
                }
                else j = pref[j-1];
            }
        }
    }
}

int main() {
    scanf("%s",str);
    scanf("%s",pat);
    int S = strlen(str);
    int P = strlen(pat);

    prefixFunction(P);
    KMPMatcher(S);
    return 0;
}
```

7.7 Manacher

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/*** Manacher's algorithm to generate longest palindromic substrings for all centers ***/
/// When i is even, pal[i] = largest palindromic substring centered from str[i / 2]
```

```
/// When i is odd, pal[i] = largest palindromic substring centered between str
    [i / 2] and str[i / 2] + 1

vector <int> manacher(char *str){ /// hash = 784265
    int i, j, k, l = strlen(str), n = l << 1;
    vector <int> pal(n);
    for (i = 0, j = 0, k = 0; i < n; j = max(0, j - k), i += k){
        while (j <= i && (i + j + 1) < n && str[(i - j) >> 1] == str[(i + j +
            1) >> 1]) j++;
        for (k = 1, pal[i] = j; k <= i && k <= pal[i] && (pal[i] - k) != pal[i
            - k]; k++){
            pal[i + k] = min(pal[i - k], pal[i] - k);
        }
    }
    pal.pop_back();
    return pal;
}

int main(){
    char str[100];
    while (scanf("%s", str)){
        auto v = manacher(str);
        for (auto it: v) printf("%d ", it);
        puts("");
    }
    return 0;
}
```

7.8 Palindromic Tree (MIST 2019 F)

```
#include <bits/stdc++.h>
using namespace std;
#define CLR(a) memset(a,0,sizeof a)
#define pb push_back

int base, mod;
inline int add(int a,int b) { return (a + 0LL + b) % mod; }
inline int mul(int a,int b) { return (a * 1LL * b) % mod; }

int ans[1000010];

namespace pt {
    const int MAXN = 1000010;
    const int MAXC = 26;

    int n;
    char str[MAXN];
    int len[MAXN], link[MAXN], ed[MAXN][MAXC], occ[MAXN], st[MAXN];

    int nc, suff, pos;

    void init() {
```

```
23          str[0] = -1;
24          nc = 2; suff = 2;
25          len[1] = -1, link[1] = 1;
26          len[2] =  0, link[2] = 1;
27          CLR(ed[1]), CLR(ed[2]);
28          occ[1] = occ[2] = 0;
29      }
30
31      inline int scale(char c) { return c-'a'; }
32
33      inline int nextLink(int cur) {
34          while (str[pos - 1 - len[cur]] != str[pos]) cur = link[cur];
35          return cur;
36      }
37
38      inline bool addLetter(int p) {
39          pos = p;
40          int let = scale(str[pos]);
41          int cur = nextLink(suff);
42
43          if (ed[cur][let]) {
44              suff = ed[cur][let];
45              occ[suff]++;
46              return false;
47          }
48
49          suff = ++nc;
50          CLR(ed[nc]);
51          len[nc] = len[cur] + 2;
52          ed[cur][let] = nc;
53          occ[nc] = 1;
54
55          if (len[nc] == 1) {
56              st[nc] = pos;
57              link[nc] = 2;
58              return true;
59          }
60          link[nc] = ed[nextLink(link[cur])][let];
61          st[nc] = pos-len[nc] + 1;
62          return true;
63      }
64
65      int sub[MAXN], shuru[MAXN], shesh[MAXN];
66      int Time = 0;
67      void dfs(int s) {
68          shuru[s] = ++Time, sub[s] = occ[s];
69          for(int i=0;i<26;i++) {
70              if(ed[s][i]) {
71                  dfs(ed[s][i]);
72                  sub[s] = add(sub[s],sub[ed[s][i]]);
73              }
```

```
        }
        shesh[s] = ++Time;
    }
    vector <int> G[MAXN];
    bool cmp(int a,int b) { return shuru[a] < shuru[b]; }

    void traverse(int s) {
        int done = 0;
        sort(G[s].begin(),G[s].end(),cmp);
        for(int x : G[s]) {
            if(shuru[x]>shuru[s] and shesh[x]<shesh[s]) traverse(x);
            else if(shuru[x] <= done) traverse(x);
            else{
                done = shesh[x];
                sub[s] = add(sub[s],sub[x]);
                traverse(x);
            }
        }
        ans[len[s]] = add(ans[len[s]],mul(occ[s],sub[s]));
    }
    void build(int _n) {
        n = _n;
        init();
        for(int i=1;i<=n;i++) addLetter(i);
        for(int i=nc;i>=3;i--) occ[link[i]] += occ[i];
        occ[1] = occ[2] = 0;
        Time = 0;
        dfs(1), dfs(2);
        for(int i=2;i<=nc;i++) G[link[i]].pb(i);
        traverse(1);
        for(int i=1;i<=nc;i++) G[i].clear();
    }
}

int poww[1000010];

int main() {
//    freopen("in.txt","r",stdin);
//    freopen("out.txt","w",stdout);

    int T,n;
    scanf("%d",&T);
    for(int t=1;t<=T;t++) {
        scanf("%d %d %d",&n,&base,&mod);
        CLR(ans);

        scanf("%s",pt::str+1);
        pt::build(strlen(pt::str+1));

        poww[0] = 1;
        for(int i=1;i<=n;i++) poww[i] = mul(poww[i-1],base);
```

```
        int res = 0;
        for(int i=1;i<=n;i++) {
            res = add(res,mul(poww[n-i],ans[i]));
        }
        printf("Case %d: %d\n",t,res);
    }
    return 0;
}
```

7.9 Palindromic Tree Extended

```
#include <bits/stdc++.h>
using namespace std;
#define CLR(a) memset(a,0,sizeof(a))
/***
    ************************************************************
    * depth[nd] = Number of suffixes of P that are palindrome   *
    *             P --> The palindrome representing node nd     *
    *                                                           *
    * fin[i] = Number of palindromes that end at index i        *
    *                                                           *
    * To know start[i], reverse the string and find fin[1...n]  *
    * Then, start[i] = fin[n-i+1] (of the reversed string)      *
    ************************************************************
***/

namespace pt {
    const int MAXN = 100010; /// maximum possible string size
    const int MAXC = 26; /// Size of the character set

    int n; /// length of str
    char str[MAXN];
    int len[MAXN], link[MAXN], ed[MAXN][MAXC], occ[MAXN], st[MAXN];
    int fin[MAXN], depth[MAXN];

    int nc, suff, pos;
    /// nc -> node count
    /// suff -> Index of the node denoting the longest palindromic proper
                /// suffix of the current prefix

    void init() {
        str[0] = -1;
        nc = 2; suff = 2;
        len[1] = -1, link[1] = 1;
        len[2] =  0, link[2] = 1;
        CLR(ed[1]), CLR(ed[2]);
        occ[1] = occ[2] = 0;
    }

    inline int scale(char c) { return c-'a'; }

    inline int nextLink(int cur) {
```

```
42          while (str[pos - 1 - len[cur]] != str[pos]) cur = link[cur];
43          return cur;
44      }
45
46      inline bool addLetter(int p) {
47          pos = p;
48          int let = scale(str[pos]);
49          int cur = nextLink(suff);
50
51          if (ed[cur][let]) {
52              suff = ed[cur][let];
53              fin[pos] = depth[suff];
54              occ[suff]++;
55              return false;
56          }
57
58          suff = ++nc;
59          CLR(ed[nc]);
60          len[nc] = len[cur] + 2;
61          ed[cur][let] = nc;
62          occ[nc] = 1;
63
64          if (len[nc] == 1) {
65              st[nc] = pos;
66              link[nc] = 2;
67              fin[pos] = depth[nc] = 1;
68              return true;
69          }
70          link[nc] = ed[nextLink(link[cur])][let];
71          fin[pos] = depth[nc] = depth[link[nc]] + 1;
72          st[nc] = pos-len[nc] + 1;
73          return true;
74      }
75
76      void build(int _n) {
77          n = _n;
78          init();
79          for(int i=1;i<=n;i++) addLetter(i);
80          for(int i=nc;i>=3;i--) occ[link[i]] += occ[i];
81          occ[2] = occ[1] = 0;
82      }
83
84      void printTree() {
85          puts(str);
86          cout << "Node\tStart\tLength\tOcc\n";
87          for(int i=3;i<=nc;i++) {
88              cout << i << "\t" << st[i] << "\t" << len[i] << "\t" << occ[i] <<
                    "\n";
89          }
90      }
91  }
```

```
92
93  int main() {
94      scanf("%s",pt::str+1);
95      pt::build(strlen(pt::str+1));
96      return 0;
97  }
```

7.10 Palindromic Tree

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   #define CLR(a) memset(a,0,sizeof(a))
4   /***
5       * str is 1 based
6
7       Each node in the palindromic tree denotes a STRING
8       Node 1 denotes an imaginary string of size -1
9       Node 2 denotes a string of size 0
10      They are the two roots
11      There can be maximum of (string_length + 2) nodes in total
12
13      It's a directed tree. If we reverse the direction of the suffix links,
14      we get a dag. In this DAG, if node v is reachable from node u iff,
15      u is a substring of v.
16
17
18
19      * if ( tree[A].next[x] == B )
20        then, B = xAx
21      * if ( tree[A].suffixLink == B )
22        Then B is the longest possible palindrome which is a proper suffix  of A
23        (node 1 is an exception)
24
25      * occ[i] contains the number of occurrences of the corresponding
            palindrome
26      * st[i] denotes starting index of the first occurrence of the
            corresponding palindrome
27      * st[] or occ[] or both can be ignored if not needed
28
29      * If memory limit is compact, a map has to be used instead of
30        ed[MAXN][MAXC]. Swapping row and column of the matrix will
31        save more memory.
32        Example :
33        map <int,int> ed[MAXC];
34        ed[c][u] = v means, there is an edge from node u to
35        node v that is labeled character c.
36
37
38
39
40  ***/
41
```

```
namespace pt {
    const int MAXN = 100010; /// maximum possible string size
    const int MAXC = 26; /// Size of the character set

    int n; /// length of str
    char str[MAXN];
    int len[MAXN], link[MAXN], ed[MAXN][MAXC], occ[MAXN], st[MAXN];

    int nc, suff, pos;
    /// nc -> node count
    /// suff -> Index of the node denoting the longest palindromic proper
                /// suffix of the current prefix

    void init() {
        str[0] = -1;
        nc = 2; suff = 2;
        len[1] = -1, link[1] = 1;
        len[2] =  0, link[2] = 1;
        CLR(ed[1]), CLR(ed[2]);
        occ[1] = occ[2] = 0;
    }

    inline int scale(char c) { return c-'a'; }

    inline int nextLink(int cur) {
        while (str[pos - 1 - len[cur]] != str[pos]) cur = link[cur];
        return cur;
    }

    inline bool addLetter(int p) {
        pos = p;
        int let = scale(str[pos]);
        int cur = nextLink(suff);

        if (ed[cur][let]) {
            suff = ed[cur][let];
            occ[suff]++;
            return false;
        }

        suff = ++nc;
        CLR(ed[nc]);
        len[nc] = len[cur] + 2;
        ed[cur][let] = nc;
        occ[nc] = 1;

        if (len[nc] == 1) {
            st[nc] = pos;
            link[nc] = 2;
            return true;
        }
```

```
        link[nc] = ed[nextLink(link[cur])][let];
        st[nc] = pos-len[nc] + 1;
        return true;
    }

    void build(int _n) {
        n = _n;
        init();
        for(int i=1;i<=n;i++) addLetter(i);
        for(int i=nc;i>=3;i--) occ[link[i]] += occ[i];
        occ[2] = occ[1] = 0;
    }

    void printTree() {
        puts(str);
        cout << "Node\tStart\tLength\tOcc\n";
        for(int i=3;i<=nc;i++) {
            cout << i << "\t" << st[i] << "\t" << len[i] << "\t" << occ[i] <<
                "\n";
        }
    }
}

int main() {
    scanf("%s",pt::str+1);
    pt::build(strlen(pt::str+1));
    return 0;
}
```

7.11 Persistent Trie

```
/***
    Given an array of size n, each value in array can be expressed using
    20 bits.
    Query : L R K
    max(a_i ^ K) for L <= i <= R
***/


const int MAX = 200010; /// maximum size of array
const int B = 19; /// maximum number of bits in a value - 1
int root[MAX], ptr = 0;

struct node {
    int ara[2], sum;
    node() {}
} tree[ MAX * (B+1) ];

void insert(int prevnode, int &curRoot, int val) {
    curRoot = ++ptr;
    int curnode = curRoot;
    for(int i = B; i >= 0; i--) {
```

```
        bool bit = val & (1 << i);
        tree[curnode] = tree[prevnode];
        tree[curnode].ara[bit] = ++ptr;
        tree[curnode].sum += 1;
        prevnode = tree[prevnode].ara[bit];
        curnode = tree[curnode].ara[bit];
    }
    tree[curnode] = tree[prevnode];
    tree[curnode].sum += 1;
}

int find_xor_max(int prevnode, int curnode, int x) {
    int ans = 0;
    for(int i = B; i >= 0; i--) {
        bool bit = x & (1 << i);
        if(tree[tree[curnode].ara[bit ^ 1]].sum > tree[tree[prevnode].ara[bit
            ^ 1]].sum) {
            curnode = tree[curnode].ara[bit ^ 1];
            prevnode = tree[prevnode].ara[bit ^ 1];
            ans = ans | (1 << i);
        }
        else {
            curnode = tree[curnode].ara[bit];
            prevnode = tree[prevnode].ara[bit];
        }
    }
    return ans;
}

void solve() {
    int n, q, L, R, K;
    cin >> n;
    for(int i=1;i<=n;i++) cin >> ara[i];

    for(int i=1;i<=q;i++) {
        cin >> L >> R >> K;
        cout << find_xor_max(root[L-1],root[R],K) << endl;
    }
}
```

7.12 String Hash + Segment Tree (Point Update, Range Query)

```
/***
    * Everything is 0 based
    * Call precal() once in the program
    * Call update(1,0,n-1,i,j,val) to update the value of position
      i to j to val, here n is the length of the string
    * Call query(1,0,n-1,L,R) to get a node containing hash
      of the position [L:R]
    * Before any update/query
        - Call init(str) where str is the string to be hashed
        - Call build(1,0,n-1)
```

```
***/

namespace strhash {
    int n;
    const int MAX = 100010;
    int ara[MAX];
    const int MOD[] = {2078526727, 2117566807};
    const int BASE[] = {1572872831, 1971536491};


    int BP[2][MAX], CUM[2][MAX];

    void init(char *str) {
        n = strlen(str);
        for(int i=0;i<n;i++) ara[i] = str[i]-'0'+1; /// scale str[i] if needed
    }

    void precal() {
        BP[0][0] = BP[1][0] = 1;
        for(int i=1;i<MAX;i++) {
            BP[0][i] = ( BP[0][i-1] * (long long) BASE[0] ) % MOD[0];
            BP[1][i] = ( BP[1][i-1] * (long long) BASE[1] ) % MOD[1];
        }
    }

    struct node {
        int sz;
        int h[2];
        node() {}
    } tree[4*MAX];

    int lazy[4*MAX];

    inline node Merge(node a,node b) {
        node ret;

        ret.h[0] = ( ( a.h[0] * (long long) BP[0][b.sz] ) + b.h[0] ) % MOD[0];
        ret.h[1] = ( ( a.h[1] * (long long) BP[1][b.sz] ) + b.h[1] ) % MOD[1];

        ret.sz = a.sz + b.sz;

        return ret;
    }

    inline void build(int n,int st,int ed) {
        if(st==ed) {
            tree[n].h[0] = tree[n].h[1] = ara[st];
            tree[n].sz = 1;
            return;
        }
        int mid = (st+ed)>>1;
```

```
        build(n+n,st,mid);
        build(n+n+1,mid+1,ed);

        tree[n] = Merge(tree[n+n],tree[n+n+1]);
    }


    inline void update(int n,int st,int ed,int id,int v) {
        if(st>id or ed<id) return;
        if(st==ed and ed==id) {
            tree[n].h[0] = tree[n].h[1] = v;
            return;
        }
        int mid = (st+ed)>>1;
        update(n+n,st,mid,id,v);
        update(n+n+1,mid+1,ed,id,v);

        tree[n] = Merge(tree[n+n],tree[n+n+1]);
    }

    inline node query(int n,int st,int ed,int i,int j){
        if(st>=i and ed<=j) return tree[n];
        int mid = (st+ed)/2;
        if(mid<i) return query(n+n+1,mid+1,ed,i,j);
        else if(mid>=j) return query(n+n,st,mid,i,j);
        else return Merge(query(n+n,st,mid,i,j),query(n+n+1,mid+1,ed,i,j));
    }
}
```

7.13 String Hash + Segment Tree (Range Update, Range Query)

```
/***
    * Everything is 0 based
    * Call precal() once in the program
    * Call update(1,0,n-1,i,j,val) to update the value of position
      i to j to val, here n is the length of the string
    * Call query(1,0,n-1,L,R) to get a node containing hash
      of the position [L:R]
    * Before any update/query
        - Call init(str) where str is the string to be hashed
        - Call build(1,0,n-1)
***/

#define INVALID_CHAR        -1

namespace strhash {
    int n;
    const int MAX = 100010;
    int ara[MAX];
    const int MOD[] = {2078526727, 2117566807};
    const int BASE[] = {1572872831, 1971536491};

```

```
22
23      int BP[2][MAX], CUM[2][MAX];
24
25      void init(char *str) {
26          n = strlen(str);
27          for(int i=0;i<n;i++) ara[i] = str[i]-'0'+1; /// scale str[i] if needed
28      }
29
30      void precal() {
31          BP[0][0] = BP[1][0] = 1;
32          CUM[0][0] = CUM[1][0] = 1;
33          for(int i=1;i<MAX;i++) {
34              BP[0][i] = ( BP[0][i-1] * (long long) BASE[0] ) % MOD[0];
35              BP[1][i] = ( BP[1][i-1] * (long long) BASE[1] ) % MOD[1];
36
37              CUM[0][i] = ( CUM[0][i-1] + (long long) BP[0][i] ) % MOD[0];
38              CUM[1][i] = ( CUM[1][i-1] + (long long) BP[1][i] ) % MOD[1];
39          }
40      }
41
42      struct node {
43          int sz;
44          int h[2];
45          node() {}
46      } tree[4*MAX];
47
48      int lazy[4*MAX];
49
50      inline void lazyUpdate(int n,int st,int ed) {
51          if(lazy[n]!=INVALID_CHAR){
52
53              tree[n].h[0] = (lazy[n] * (long long) CUM[0][ed-st]) % MOD[0];
54              tree[n].h[1] = (lazy[n] * (long long) CUM[1][ed-st]) % MOD[1];
55
56              if(st!=ed){
57                  lazy[2*n] = lazy[n];
58                  lazy[2*n+1] = lazy[n];
59              }
60              lazy[n] = INVALID_CHAR;
61          }
62      }
63
64      inline node Merge(node a,node b) {
65          node ret;
66
67          ret.h[0] = ( ( a.h[0] * (long long) BP[0][b.sz] ) + b.h[0] ) % MOD[0];
68          ret.h[1] = ( ( a.h[1] * (long long) BP[1][b.sz] ) + b.h[1] ) % MOD[1];
69
70          ret.sz = a.sz + b.sz;
71
72          return ret;
```

```
    }

    inline void build(int n,int st,int ed) {
        lazy[n] = INVALID_CHAR;
        if(st==ed) {
            tree[n].h[0] = tree[n].h[1] = ara[st];
            tree[n].sz = 1;
            return;
        }
        int mid = (st+ed)>>1;
        build(n+n,st,mid);
        build(n+n+1,mid+1,ed);

        tree[n] = Merge(tree[n+n],tree[n+n+1]);
    }


    inline void update(int n,int st,int ed,int i,int j,int v) {
        lazyUpdate(n,st,ed);
        if(st>j or ed<i) return;
        if(st>=i and ed<=j) {
            lazy[n] = v;
            lazyUpdate(n,st,ed);
            return;
        }

        int mid = (st+ed)>>1;
        update(n+n,st,mid,i,j,v);
        update(n+n+1,mid+1,ed,i,j,v);

        tree[n] = Merge(tree[n+n],tree[n+n+1]);
    }

    inline node query(int n,int st,int ed,int i,int j){
        lazyUpdate(n,st,ed);
        if(st>=i and ed<=j) return tree[n];
        int mid = (st+ed)/2;
        if(mid<i) return query(n+n+1,mid+1,ed,i,j);
        else if(mid>=j) return query(n+n,st,mid,i,j);
        else return Merge(query(n+n,st,mid,i,j),query(n+n+1,mid+1,ed,i,j));
    }
}
```

7.14 Suffix Array (O(n))

```
/***
    * scan sa::str
    * n = strlen(sa::str)
    * call sa::build(n)

    * there are n+1 suffixes including the null suffix(denoted as n'th suffix,
        0 based suffix indexing)
```

```
7       * S[0 ... n] is the suffix array ( n+1 elements including the null suffix
            )
8       * rnk[i] denotes the index of the i'th suffix in S[]
9       * lcp[0] = 0, lcp[i] = longest commong prefix( suffix S[i-1], suffix S[i]
            )
10  ***/
11
12  namespace sa {
13
14      const int N = 100010; /// maximum possible string size
15
16      char str[N];
17      int wa[N],wb[N],wv[N],wc[N];
18      int r[N],S[N],rnk[N], lcp[N];
19
20
21      int cmp(int *r,int a,int b,int l) {
22          return r[a] == r[b] && r[a+l] == r[b+l];
23      }
24
25      void da(int *r,int *sa,int n,int m)
26      {
27          int i,j,p,*x=wa,*y=wb,*t;
28          for( i=0; i<m; i++) wc[i]=0;
29          for( i=0; i<n; i++) wc[x[i]=r[i]] ++;
30          for( i=1; i<m; i++) wc[i] += wc[i-1];
31          for( i= n-1; i>=0; i--)S[--wc[x[i]]] = i;
32          for( j= 1,p=1; p<n; j*=2,m=p)
33          {
34              for(p=0,i=n-j; i<n; i++)y[p++] = i;
35              for(i=0; i<n; i++)if(S[i] >= j) y[p++] = S[i] - j;
36              for(i=0; i<n; i++) wv[i] = x[y[i]];
37              for(i=0; i<m; i++) wc[i] = 0;
38              for(i=0; i<n; i++) wc[wv[i]] ++;
39              for(i=1; i<m; i++) wc[i] += wc[i-1];
40              for(i=n-1; i>=0; i--) S[--wc[wv[i]]] = y[i];
41              for(t=x,x=y,y=t,p=1,x[S[0]] = 0,i=1; i<n; i++) x[S[i]]= cmp(y,S[i
                    -1],S[i],j) ? p-1:p++;
42
43          }
44      }
45
46      void calheight(int *r,int *sa,int n) {
47          int i,j,k=0;
48          for(i=1; i<=n; i++) rnk[S[i]] = i;
49          for(i=0; i<n; lcp[rnk[i++]] = k ) {
50              for(k?k--:0, j=S[rnk[i]-1] ; r[i+k] == r[j+k] ; k ++) ;
51          }
52      }
53
54      void build(int n) {
```

```
        for(int i=0;str[i];i++) r[i] = (int)str[i];
        r[n] = 0;
        da(r,S,n+1,128);
        calheight(r,S,n);
    }
}
```

7.15 Suffix Array (n logn)

```
/***
    * str will contain the string
    * L = length of str
    * call generateSA(), the S[] will contain the suffix array

    * Think of suffix as a trie of suffixes.
    * lcp (x, y) = minimum { lcp(x, x + 1), lcp(x + 1, x + 2), ... lcp(y \96
       1, y) }.
***/

const int MAXL = ?;/// 1<< 20
const int MAXLG = ?;/// 20

char str[MAXL];
int L , stp;
int S[MAXL];
int P[MAXLG][MAXL];
/// P[i][j] = position of the suffix starting at character j after sorting
/// on the basis of 2^i characters

struct entry {
    int pr[2]; /// parameters for sorting
    int id; /// starting index of the suffix
} suf[MAXL] , out[MAXL];

int cnt[MAXL] , taken[MAXL] , cum[MAXL];
int special , specialTaken , it;

inline void countingSort(int type) {
    int i;
    CLR(cnt);
    CLR(taken);
    special = specialTaken = 0;
    for(i = 0; i<L ; i++) {
        if(suf[i].pr[type] == -1) special++;
        else cnt[ suf[i].pr[type] ]++;
    }
    cum[0] = special;
    for(i = 1; i <= it ; i++) cum[i] = cum[i-1] + cnt[i-1];
    for(i = 0; i<L ; i++) {
        if(suf[i].pr[type] == -1) out[ specialTaken++ ] = suf[i];
        else out[ cum[ suf[i].pr[type] ] + taken[ suf[i].pr[type] ]++ ] = suf[
           i];
```

```
42      }
43      for(i = 0; i<L ; i++) suf[i] = out[i];
44  }
45
46  /// n * lg n
47  void generateSA(){
48      int now,i;
49      it = 0;
50      for(i=0; i<L; i++){
51          P[0][i] = (int)str[i];
52          it = max(it, P[0][i]);
53      }
54      for(now=1,stp=1 ; now <=L ; stp++,now *= 2){
55          for(i=0; i<L; i++){
56              suf[i].pr[0] = P[stp-1][i];
57              if(i+now<L) suf[i].pr[1] = P[stp-1][i+now];
58              else suf[i].pr[1] = -1;
59              suf[i].id = i;
60          }
61          countingSort(1);
62          countingSort(0);
63          it = -1;
64          for(i=0; i<L; i++){
65              if(i>0 && suf[i].pr[0]==suf[i-1].pr[0] && suf[i].pr[1]==suf[i-1].
                    pr[1])
66                  P[stp][suf[i].id] = it;
67              else
68                  P[stp][suf[i].id] = ++it;
69
70              it = max(it, P[stp][ suf[i].id ]);
71          }
72      }
73      for(i=0;i<L;i++) S[P[stp-1][i]] = i;
74  }
75
76  /// n * lg n
77  inline int getLCP(int x,int y){
78      int ret = 0,add,i;
79      if(x==y) return L-x;
80      for(i = stp-1 ; i >= 0 && x<L && y<L; i--){
81          if(P[i][x]==P[i][y]){
82              ret += (1<<i), x += (1<<i), y += (1<<i);
83          }
84      }
85      return ret;
86  }
```

7.16 Suffix Automaton Extended

```
1   /***
2       * N = maximum possible string size
3       * There won't be more that 2N - 1 nodes
```

```
    * There won't be more that 3N - 4 transitions
    * nodes are numbered from 0 to sz-1

    * scan sa::str
    * n = strlen(str)
    * call sa::build(n)

    * let's suppose sub_i represents the maximum substring that is endpos
       equivalent to node i
    * cnt[i] = number of occurrences of sub_i in str
    * If terminal[i] = true, then sub_i is a suffix of str

    ********************************************************************************
    * dp[i] = number of substrings that has sub_i as prefix                        *
    *         The substrings don't need to be unique                               *
    * lex_kth_substr(k) returns the lexicographically k'th substring of str        *
    ********************************************************************************

***/
namespace sa{
    const int MAXN = 100005 << 1; /// 2 * maximum possible string size
    const int MAXC = 26; /// Size of the character set

    char str[MAXN];

    int n, sz, last; /// sz = number of nodes in the automaton( node indexing
       is 0 based)
    int len[MAXN], link[MAXN], ed[MAXN][MAXC], cnt[MAXN];
    bool terminal[MAXN];
    vector <int> G[MAXN];

    void init() {
        SET(ed[0]);
        len[0] = 0, link[0] = -1, sz = 1, last = 0, terminal[0] = false;
    }

    inline int scale(char c) { return c-'a'; }

    void extend(char c) {
        int cur = sz++;

        terminal[cur] = false;
        cnt[cur] = 1;

        SET(ed[cur]);
        len[cur] = len[last] + 1;
        int p = last;
        while (p != -1 && ed[p][c]==-1) {
            ed[p][c] = cur;
            p = link[p];
        }
```

```
        if (p == -1) link[cur] = 0;
        else {
            int q = ed[p][c];
            if (len[p] + 1 == len[q]) link[cur] = q;
            else {
                int clone = sz++;
                len[clone] = len[p] + 1;
                memcpy(ed[clone],ed[q],sizeof(ed[q]));
                link[clone] = link[q];
                while (p != -1 && ed[p][c] == q) {
                    ed[p][c] = clone;
                    p = link[p];
                }
                link[q] = link[cur] = clone;

                cnt[clone] = 0;
                terminal[clone] = false;
            }
        }
        last = cur;
    }

    /// needed to generate cnt[]
    void dfs(int s) {
        for(auto x : G[s]) dfs(x), cnt[s] += cnt[x];
    }

    ll dp[MAXN];
    ll call(int nd) {
        ll &ret = dp[nd];
        int x;
        if(ret!=-1) return ret;
        ret = cnt[nd];
        for(int i=0;i<MAXC;i++) {
            x = ed[nd][i];
            if(x!=-1) ret += call(x);
        }
        return ret;
    }



    /// returns the lexicographically k'th substring of str
    string lex_kth_substr(ll k) {
        if((k+k) > (n*(n+1LL))) return "No such line.";
        string ret = "";
        int cur = 0, x;
        while(k>0) {
            for(int i=0;i<MAXC;i++) {
                x = ed[cur][i];
                if(x == -1) continue;
```

```
                if(call(x)>=k) {
                    ret += (char)i + 'a';
                    cur = x;
                    k -= cnt[x];
                    break;
                }
                k -= call(x);
            }
        }
        return ret;
    }


    void build() {
        init();
        n = strlen(str);
        for(int i=0;i<n;i++) extend(scale(str[i]));

        /// construction of cnt array
        for(int i=1;i<sz;i++) G[link[i]].pb(i);
        dfs(0);
        for(int i=0;i<sz;i++) G[i].clear();

        /// construction of terminal array
        for(int i=last;i!=-1;i=link[i]) terminal[i] = true;

        /// lex_kth_substr
        SET(dp);
    }
}
```

7.17 Suffix Automaton

```
/***
    * N = maximum possible string size
    * There won't be more that 2N - 1 nodes
    * There won't be more that 3N - 4 transitions
    * nodes are numbered from 0 to sz-1

    * scan sa::str
    * n = strlen(str)
    * call sa::build(n)

    * let's suppose sub_n represents the largest substring that is endpos
       equivalent to node n

    * cnt[i] = number of occurrences of sub_i in str
    * If terminal[i] = true, then sub_i is a suffix of str
    * There suffix link of node x to node y,
      Iff sub_y is the largest suffix of sub_x that is not endpos equivalent
         to node x.
***/
```

```
18 namespace sa{
19     const int MAXN = 100005 << 1; /// 2 * maximum possible string size
20     const int MAXC = 26; /// Size of the character set
21 
22     char str[MAXN];
23 
24     int n, sz, last; /// sz = number of nodes in the automaton( node indexing
           is 0 based)
25     int len[MAXN], link[MAXN], ed[MAXN][MAXC], cnt[MAXN];
26     bool terminal[MAXN];
27     vector <int> G[MAXN];
28 
29     void init() {
30         SET(ed[0]);
31         len[0] = 0, link[0] = -1, sz = 1, last = 0, terminal[0] = false;
32     }
33 
34     inline int scale(char c) { return c-'a'; }
35 
36     void extend(char c) {
37         int cur = sz++;
38 
39         terminal[cur] = false;
40         cnt[cur] = 1;
41 
42         SET(ed[cur]);
43         len[cur] = len[last] + 1;
44         int p = last;
45         while (p != -1 && ed[p][c]==-1) {
46             ed[p][c] = cur;
47             p = link[p];
48         }
49         if (p == -1) link[cur] = 0;
50         else {
51             int q = ed[p][c];
52             if (len[p] + 1 == len[q]) link[cur] = q;
53             else {
54                 int clone = sz++;
55                 len[clone] = len[p] + 1;
56                 memcpy(ed[clone],ed[q],sizeof(ed[q]));
57                 link[clone] = link[q];
58                 while (p != -1 && ed[p][c] == q) {
59                     ed[p][c] = clone;
60                     p = link[p];
61                 }
62                 link[q] = link[cur] = clone;
63 
64                 cnt[clone] = 0;
65                 terminal[clone] = false;
66             }
67         }
```

```
68          last = cur;
69      }
70
71      /// needed to generate cnt[]
72      void dfs(int s) {
73          for(auto x : G[s]) dfs(x), cnt[s] += cnt[x];
74      }
75
76      void build() {
77          init();
78          int n = strlen(str);
79          for(int i=0;i<n;i++) extend(scale(str[i]));
80
81          /// construction of cnt[]
82          for(int i=1;i<sz;i++) G[link[i]].pb(i);
83          dfs(0);
84          for(int i=0;i<sz;i++) G[i].clear();
85
86          /// construction of terminal[]
87          for(int i=last;i!=-1;i=link[i]) terminal[i] = true;
88      }
89  }
```

7.18 Trie (Static Array)

```
1   #define N       200000 /// total number of characters given as input
2   #define S       26
3
4   int root,now;
5   int nxt[N][S], cnt[N];
6
7   /// will be called from main
8   void init(){
9       root = now = 1;
10      CLR(nxt),CLR(cnt);
11  }
12
13  inline int scale(char ch) { return (ch - 'a'); }
14
15  inline void Insert(char s[],int sz){
16      int cur = root, to;
17      for(int i=0 ; i< sz ; i++){
18          to = scale(s[i]) ;
19          if( !nxt[cur][to] ) nxt[cur][to] = ++now;
20          cur = nxt[cur][to];
21      }
22      cnt[cur]++;
23  }
24
25  inline bool Find(char s[],int sz){
26      int cur = root, to;
27      for(int i=0 ; i<sz ; i++){
```

```
28          to = scale(s[i]) ;
29          if( !nxt[cur][to] ) return false;
30          cur = nxt[cur][to];
31      }
32      return (cnt[cur]!=0);
33  }
34
35  /// It's better to call the Delete() after checking if the
36  /// string we wanna delete actually exists in the trie
37  inline void Delete(char s[],int sz){
38      int cur = root, to;
39      for(int i=0 ; i<sz ; i++){
40          to = scale(s[i]) ;
41          cur = nxt[cur][to];
42      }
43      cnt[cur]--;
44  }
```