ICPC Codebook

CodeCruXaders (BPHC)

March 29, 2024

Contents

1 Basics

```
//Fast IO
ios::sync_with_stdio(false);
cin.tie(nullptr);cout.tie(nullptr);

//File open and close
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);

//Floating Point - rounded to six decimal
    places (rounding half to even)
ans*=1e7;
int res = ans;
if(res==ans && res/10 ==5){
    res/=10;
    if(res%2)res++;
    res*=10;
}
ans = res/1e7;
cout<<fixed<<setprecision(6)<<ans;

//Remove Duplicates
vector<int> vec{1, 1, 1, 2, 3, 3};
sort(vec.begin(), vec.end());
vec.erase(unique(vec.begin(), vec.end()),
    vec.end());

//Lambda Comparators
/* Sets, priority queues & sorting
   vectors with custom comparators */
vector<pair<int, int>> v;
sort(v.begin(), v.end(), [](const auto& A
    , const auto& B) {
    // ...
    // for pairs
    return A.second == B.second?
                A.first < B.first : A.
                    second < B.second;
});
// STL set and priority_queue comparators
auto cmp = [](const auto& A, const auto&
   B) {
    return A < B;
};
set<T, decltype(cmp)> s(cmp);
priority_queue<T, vector<T>, decltype(cmp
    )> pq(cmp);

/* GP hash table, an alternative to
   unordered_map w/ custom hash */
#include <ext/pb_ds/assoc_container.hpp>
namespace Hashing {
    using hash_t = pair<int, uint64_t>;
    static const uint64_t FIXED_RANDOM =
        chrono::steady_clock::now().
        time_since_epoch().count();

    // Custom hash using splitmix64 (https
        ://codeforces.com/blog/entry
        /62393)
    struct custom_hash {
        static uint64_t splitmix64(
            uint64_t x) {
            x += 0x9e3779b97f4a7c15;
            x = (x ^ (x >> 30)) * 0
                xbf58476d1ce4e5b9;
            x = (x ^ (x >> 27)) * 0
                x94d049bb133111eb;
            return x ^ (x >> 31);
        }
        size_t operator()(uint64_t x)
            const {
            return splitmix64(x +
                FIXED_RANDOM);
        }
        size_t operator()(const hash_t& x)
             const {
            return splitmix64(FIXED_RANDOM
                 + x.second)
                    ^ (splitmix64(
                        FIXED_RANDOM + x.
                        first) << 1);
        }
    };

    // gp_hash_table benchmarks vs
       unordered_map (https://codeforces.
       com/blog/entry/60737)
    template<typename K, typename V,
        typename Hash = custom_hash>
    using hash_map = __gnu_pbds::
        gp_hash_table<K, V, Hash>;

    template<typename K, typename Hash =
        custom_hash>
    using hash_set = hash_map<K,
        __gnu_pbds::null_type, Hash>;
}

/* Policy Based Data Structures -
   ordered_set */
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<class key, class value =
   null_type, class cmp = std::less<key>>
using ordered_set = tree<key, value, cmp,
    rb_tree_tag,
   tree_order_statistics_node_update>;
#define ook order_of_key // count of
   elements strictly smaller than k
#define fbo find_by_order // iterator to
   kth element starting from 0

ordered_set<int> os;
ordered_set<pair<int, int>> os;
ordered_set<int, int> o_map;
ordered_set<int, null_type, std::greater<
   int>> os_greater;
```

2 Combinatorics

2.1 Balanced paranthesis

```
//Given a balanced sequence, we have to
   find the next (in lexicographical
   order) balanced sequence. O(n)
bool next_balanced_sequence(string & s) {
    int n = s.size();
    int depth = 0;
    for (int i = n - 1; i >= 0; i--) {
        if (s[i] == '(')
```

```
            depth--;
        else
            depth++;

        if (s[i] == '(' && depth > 0) {
            depth--;
            int open = (n - i - 1 - depth)
                / 2;
            int close = n - i - 1 - open;
            string next = s.substr(0, i) +
                ')' + string(open, '(') +
                string(close, ')');
            s.swap(next);
            return true;
        }
    }
    return false;
}
//Given a balanced bracket sequence with
   n pairs of brackets. We have to find
   its index in the lexicographically
   ordered list of all balanced sequences
    with n bracket pairs.
string kth_balanced(int n, int k) {
    vector<vector<int>> d(2*n+1, vector<
        int>(n+1, 0));
    d[0][0] = 1;
    for (int i = 1; i <= 2*n; i++) {
        d[i][0] = d[i-1][1];
        for (int j = 1; j < n; j++)
            d[i][j] = d[i-1][j-1] + d[i
                -1][j+1];
        d[i][n] = d[i-1][n-1];
    }

    string ans;
    int depth = 0;
    for (int i = 0; i < 2*n; i++) {
        if (depth + 1 <= n && d[2*n-i-1][
            depth+1] >= k) {
            ans += '(';
            depth++;
        } else {
            ans += ')';
            if (depth + 1 <= n)
                k -= d[2*n-i-1][depth+1];
            depth--;
        }
    }
    return ans;
}
```

2.2 Collorary

2.2.1 Binomial Properties

Recurrence formula:
$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$
Symmetry rule:
$\binom{n}{k} = \binom{n}{n-k}$
Factoring in:
$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$
Sum over k:
$\sum_{k=0}^{n}\binom{n}{k} = 2^n$
Sum over n:
$\sum_{m=0}^{n}\binom{m}{k} = \binom{n+1}{k+1}$
Sum over n and k:
$\sum_{k=0}^{m}\binom{n+k}{k} = \binom{n+m+1}{m}$
Sum of the squares:
$\binom{n}{0}^2 + \binom{n}{1}^2 + \cdots + \binom{n}{n}^2 = \binom{2n}{n}$
Weighted sum:
$1\binom{n}{1} + 2\binom{n}{2} + \cdots + n\binom{n}{n} = n2^{n-1}$
Connection with the Fibonacci numbers:
$\binom{n}{0} + \binom{n-1}{1} + \cdots + \binom{n-k}{k} + \cdots + \binom{0}{n} = F_{n+1}$

2.2.2 Catalan Numbers

Number of correct bracket sequence consisting of n opening and n closing brackets. The number of ways to connect the $2n$ points on a circle to form n disjoint chords. The number of non-isomorphic full binary trees with n internal nodes (i.e. nodes having at least one son). $C_n = \frac{1}{n+1}\binom{2n}{n}$

2.2.3 Stars and bars

The number of ways to put n identical objects into k labeled boxes is $\binom{n+k-1}{n}$ (Try to think of it as stars and bars || |)

2.2.4 Labelled Graph with k-connected components

Sum for all K: $G_n = 2^{\frac{n(n-1)}{2}}$ k = 1: $C_n = G_n - \frac{1}{n}\sum_{k=1}^{n-1} k\binom{n}{k}C_kG_{n-k}$ For any k: $D[n][k] = \sum_{s=1}^{n}\binom{n-1}{s-1}C_sD[n-s][k-1]$

2.3 Dearrangement

```
//Number of permutations without Fixed
   points
const int N = 1e6;
vector<int> dearrange_dp(N+1);
int dearrange(int x){
    if(dearrange_dp[0]!=0)return
        dearrange_dp[x];
    dearrange_dp[0] = 1;
    dearrange_dp[1] = 0;

    for(int i=2;i<=N;i++){
        dearrange_dp[i] = ((i-1)*(
            dearrange_dp[i-1] +dearrange_dp
            [i-2]))%mod;
    }
    return dearrange_dp[x];
}
// Also we can do n! * (1-1/1! + 1/2! -
   1/3! + ... +- 1/n!) ~ n!/e
```

2.4 Factorial and Binomial

```
//Factorial with mod
const int N = 2*1e6;
factorial[0] = 1;
for (int i = 1; i <= N; i++) {
    factorial[i] = factorial[i - 1] * i %
        m;
```

```
}

//You are given two numbers n and k. Find
    the largest power of k x such that n!
    is divisible by k^x.
//For Prime k
int fact_pow (int n, int k) {
    int res = 0;
    while (n) {
        n /= k;
        res += n;
    }
    return res;
}
//For Non-Prime k
//Represent k = k_1^p_1 ... k_m^p_m and
   then ans is min(i=1..m) a_i/p_i where
   a_i is fact_pow(n,k_1)

//Binomial Using Precalculated Factorial
   and inverse_Factorial using Mod
   inverse for Array
long long binomial_coefficient(int n, int
    k) {
    return factorial[n] *
        inverse_factorial[k] % m *
        inverse_factorial[n - k] % m;
}
```

3 Data Structures

3.1 Fenwick Tree

```
template <typename T>
struct Fenwick {
    int n;
    vector<T> a;

    Fenwick(int n_ = 0) {
        init(n_);
    }

    void init(int n_) {
        n = n_;
        a.assign(n, T{});
    }

    void add(int x, const T &v) {
        for (int i = x + 1; i <= n; i += i
            & -i) {
            a[i - 1] = a[i - 1] + v;
        }
    }

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

    T sum(int l, int r) {
        return sum(r) - sum(l);
    }
};
```

3.2 Lazy Propogation

```
template<class Info, class Tag>
struct LazySegmentTree {
    int n;
    vector<Info> info;
    vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info
        ()) {
        init(n_, v_);
    }
    template<class T>
    LazySegmentTree(vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(vector(n_, v_));
    }
    template<class T>
    void init(vector<T> init_) {
        n = init_.size();
        info.assign(4 << __lg(n), Info());
        tag.assign(4 << __lg(n), Tag());
        function<void(int, int, int)>
            build = [&](int p, int lx, int
            rx) {
            if (rx == lx) {
                info[p] = init_[lx];
                return;
            }
            int m = (lx + rx) / 2;
            build(2 * p + 1, lx, m);
            build(2 * p + 2, m + 1, rx);
            pull(p);
        };
        build(0, 0, n - 1);
    }
    void pull(int p) {
        info[p] = info[2 * p + 1] + info[2
            * p + 2];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p + 1, tag[p]);
        apply(2 * p + 2, tag[p]);
        tag[p] = Tag();
    }
    void update(int p, int lx, int rx, int
         x, const Info &v) {
        if (rx == lx) {
            info[p] = v;
            return;
        }
        int m = (lx + rx) / 2;
        push(p);
        if (x <= m) {
            update(2 * p + 1, lx, m, x, v)
                ;
        } else {
```

```
        update(2 * p + 2, m + 1, rx, x
            , v);
    }
    pull(p);
}
void update(int p, const Info &v) {
    update(0, 0, n - 1, p, v);
}
Info query(int p, int lx, int rx, int
    l, int r) {
    if (lx > r || rx < l) {
        return Info();
    }
    if (l <= lx && rx <= r) {
        return info[p];
    }
    int m = (lx + rx) / 2;
    push(p);
    return query(2 * p + 1, lx, m, l,
        r) + query(2 * p + 2, m + 1, rx
        , l, r);
}
Info query(int l, int r) {
    return query(0, 0, n - 1, l, r);
}
void rangeUpdate(int p, int lx, int rx
    , int l, int r, const Tag &v) {
    if (lx > r || rx < l) {
        return;
    }
    if (l <= lx && rx <= r) {
        apply(p, v);
        return;
    }
    int m = (lx + rx) / 2;
    push(p);
    rangeUpdate(2 * p + 1, lx, m, l, r
        , v);
    rangeUpdate(2 * p + 2, m + 1, rx,
        l, r, v);
    pull(p);
}
void rangeUpdate(int l, int r, const
    Tag &v) {
    return rangeUpdate(0, 0, n - 1, l,
        r, v);
}
template<class F>
int findFirst(int p, int lx, int rx,
    int l, int r, F pred) {
    if (lx > r || rx < l || !pred(info
        [p])) {
        return -1;
    }
    if (rx == lx) {
        return lx;
    }
    int m = (lx + rx) / 2;
    push(p);
    int res = findFirst(2 * p + 1, lx,
        m, l, r, pred);
    if (res == -1) {
        res = findFirst(2 * p + 2, m +
            1, rx, l, r, pred);
    }
    return res;
}
template<class F>
int findFirst(int l, int r, F pred) {
    return findFirst(0, 0, n - 1, l, r
        , pred);
}
template<class F>
int findLast(int p, int lx, int rx,
    int l, int r, F pred) {
    if (lx > r || rx < l || !pred(info
        [p])) {
        return -1;
    }
    if (rx == lx) {
        return lx;
    }
    int m = (lx + rx) / 2;
    push(p);
    int res = findLast(2 * p + 2, m +
        1, rx, l, r, pred);
    if (res == -1) {
        res = findLast(2 * p + 1, lx,
            m, l, r, pred);
    }
    return res;
}
template<class F>
int findLast(int l, int r, F pred) {
    return findLast(0, 0, n - 1, l, r,
        pred);
}
};

struct Tag {
    int add = 0;

    Tag() {}
    Tag(int s) : add(s) {}

    void apply(const Tag &t) {
        add += t.add;
    }
};

struct Info {
    int sum = 0;

    Info() {}
    Info(int s) : sum(s) {}

    void apply(const Tag &t) {
        sum += t.add;
    }
};

Info operator+(const Info &a, const Info
    &b) {
    Info c{a.sum + b.sum};
    return c;
}
```

3.3 Merge Sort Tree

```
const int N=1e5+5;
int n,q,a[N];
vector<int> seg[4*N+5];
```

```
//build merge sort tree
void build(int node,int l,int r){
    //if range length is 1 there's only
       one element to add and no children
    if (l==r){
        seg[node].push_back(a[l]);
        return;
    }int mid=(l+r)/2;
    build(node*2,l,mid);
    build(node*2+1,mid+1,r);
    int i=0,j=0;
    // use two pointers to merge the two
       vectors in O(r-l+1)
    while (i<seg[node*2].size() && j<seg[
       node*2+1].size()){
        if (seg[node*2][i]<seg[node*2+1][j
           ]) seg[node].push_back(seg[node
           *2][i++]);
        else seg[node].push_back(seg[node
           *2+1][j++]);
    }
    while (i<seg[node*2].size()) seg[node
       ].push_back(seg[node*2][i++]);
    while (j<seg[node*2+1].size()) seg[
       node].push_back(seg[node*2+1][j
       ++]);
    return;
}
//query
int query(int node,int l,int r,int lx,int
     rx,int x){
    //if outside -> 0
    if (l>rx || r<lx) return 0;
    //if inside do binary search
    if (l>=lx && r<=rx){
        int L=0,R=seg[node].size()-1,mid,
           ans=0;
        while (L<=R){
            mid=(L+R)/2;
            if (seg[node][mid]<x){
                ans=mid+1;
                L=mid+1;
            }else R=mid-1;
        }return ans;
    }
    int mid=(l+r)/2;
    return query(node*2,l,mid,lx,rx,x)+
       query(node*2+1,mid+1,r,lx,rx,x);
}
```

3.4 Mex

```
// O(1) Mex, O(logN) update, O(NlogN)
   precompute
class Mex {
private:
    map<int, int> frequency;
    set<int> missing_numbers;
    vector<int> A;
public:
    Mex(vector<int> const& A) : A(A) {
        for (int i = 0; i <= A.size(); i
           ++)
            missing_numbers.insert(i);

        for (int x : A) {
            ++frequency[x];
            missing_numbers.erase(x);
        }
    }
    int mex() {
        return *missing_numbers.begin();
    }

    void update(int idx, int new_value) {
        if (--frequency[A[idx]] == 0)
            missing_numbers.insert(A[idx])
               ;
        A[idx] = new_value;
        ++frequency[new_value];
        missing_numbers.erase(new_value);
    }
};
```

3.5 Min Max Stack

```
struct mxstack{
    vector<int> s1,s1min={LLONG_MAX},s1max
       ={LLONG_MIN};
    vector<int> s2,s2min={LLONG_MAX},s2max
       ={LLONG_MIN};
    void push(int x){
        s2.push_back(x);
        s2min.push_back(::min(x,s2min.back
           ()));
        s2max.push_back(::max(x,s2max.back
           ()));
    }
    int pop(){
        if(s1.empty()){
            while(!s2.empty()){
                s1min.push_back(::min(s1min
                   .back(),s2.back()));
                s2min.pop_back();
                s1max.push_back(::max(s1max
                   .back(),s2.back()));
                s2max.pop_back();
                s1.push_back(s2.back());
                s2.pop_back();
            }
        }
        int res = s1.back();
        s1.pop_back();
        s1min.pop_back();
        s1max.pop_back();
        return res;
    }
    bool empty(){
        return s1.empty()&&s2.empty();
    }
    int min(){
        return ::min(s1min.back(),s2min.
           back());
    }
    int max(){
        return ::max(s1max.back(),s2max.
           back());
    }
};
```

3.6 Mo's Algorithm

```cpp
// void remove(idx); // TODO: remove
   value at idx from data structure
// void add(idx);   // TODO: add value at
    idx from data structure
// int get_answer(); // TODO: extract the
    current answer of the data structure
const int BLOCK_SIZE=700;
struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        if (l / BLOCK_SIZE != other.l /
            BLOCK_SIZE)return make_pair(l /
             BLOCK_SIZE, r) < make_pair(
            other.l / BLOCK_SIZE, other.r);
        return (l / BLOCK_SIZE & 1) ? (r <
             other.r) : (r > other.r);
    }
};
vector<int> mo_s_algorithm(vector<Query>
   queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
    // TODO: initialize data structure
    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will
       always reflect the range [cur_l,
       cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

3.7 Segment Tree

```cpp
template<class Info>
struct SegmentTree {
    int n;
    vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info())
       {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(vector(n_, v_));
    }
    template<class T>
    void init(vector<T> init_) {
        n = init_.size();
        info.assign(4 << __lg(n), Info());
        function<void(int, int, int)>
           build = [&](int p, int lx, int
           rx) {
            if (rx == lx) {
                info[p] = init_[lx];
                return;
            }
            int m = (lx + rx) / 2;
            build(2 * p + 1, lx, m);
            build(2 * p + 2, m + 1, rx);
            pull(p);
        };
        build(0, 0, n - 1);
    }
    void pull(int p) {
        info[p] = info[2 * p + 1] + info[2
             * p + 2];
    }
    void update(int p, int lx, int rx, int
         x, const Info &v) {
        if (rx == lx) {
            info[p] = v;
            return;
        }
        int m = (lx + rx) / 2;
        if (x <= m) {
            update(2 * p + 1, lx, m, x, v)
               ;
        } else {
            update(2 * p + 2, m + 1, rx, x
               , v);
        }
        pull(p);
    }
    void update(int p, const Info &v) {
        update(0, 0, n - 1, p, v);
    }
    Info query(int p, int lx, int rx, int
       l, int r) {
        if (lx > r || rx < l) {
            return Info();
        }
        if (lx >= l && rx <= r) {
            return info[p];
        }
        int m = (lx + rx) / 2;
        return query(2 * p + 1, lx, m, l,
            r) + query(2 * p + 2, m + 1, rx
            , l, r);
    }
    Info query(int l, int r) {
        return query(0, 0, n - 1, l, r);
    }
    template<class F>
    int findFirst(int p, int lx, int rx,
       int l, int r, F pred) {
```

```
    if (lx > r || rx < l || !pred(info
        [p])) {
        return -1;
    }
    if (rx == lx) {
        return lx;
    }
    int m = (lx + rx) / 2;
    int res = findFirst(2 * p + 1, lx,
         m, l, r, pred);
    if (res == -1) {
        res = findFirst(2 * p + 2, m +
             1, rx, l, r, pred);
    }
    return res;
}
template<class F>
int findFirst(int l, int r, F pred) {
    return findFirst(0, 0, n - 1, l, r
        , pred);
}
template<class F>
int findLast(int p, int lx, int rx,
    int l, int r, F pred) {
    if (lx > r || rx < l || !pred(info
        [p])) {
        return -1;
    }
    if (rx == lx) {
        return lx;
    }
    int m = (lx + rx) / 2;
    int res = findLast(2 * p + 2, m +
        1, rx, l, r, pred);
    if (res == -1) {
        res = findLast(2 * p + 1, lx,
            m, l, r, pred);
    }
    return res;
}
template<class F>
int findLast(int l, int r, F pred) {
    return findLast(0, 0, n - 1, l, r,
         pred);
    }
};

struct Info {
    int sum = 0;

    Info() {}
    Info(int s) : sum(s) {}
};

Info operator+(Info a, Info b) {
    return Info(a.sum + b.sum);
}
```

3.8 Sparse Table

```
/* st[i][j]: answer for range [j, j + 2^i
    - 1] (both inclusive)
 * so [j, j+2^i-1] splits into [j, j+2^(i
    -1)-1] and [j+2^(i-1), j+2^i-1]
 */
int st[K + 1][MAXN];  // K >= __lg(MAXN)
std::copy(array.begin(), array.end(), st
    [0]);
for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j
        ++)
        st[i][j] = f(st[i - 1][j], st[i -
            1][j + (1 << (i - 1))]);
        // st[i][j] = st[i - 1][j] + st[i -
            1][j + (1 << (i - 1))]; for sum
            query
// find answer for query
long long sum = 0;
for (int i = K; i >= 0; i--) {
    if ((1 << i) <= R - L + 1) {
        sum += st[i][L];
        L += 1 << i;
    }
}

struct RMQ{
    vector<vector<int>> pre;
    int n;
    void init(int _n){
        n = _n;
        pre.resize(n+1,vector<int>((int)
            log2(n)+1,0));
    }
    void init(vector<int> &a){
        init(a.size());
        build(a);
    }
    void build(vector<int> &a){
        for(int i=0;i<n;i++){
            pre[i][0] = a[i];
        }
        for(int j=1;j<=log2(n);j++){
            for(int i=0; (i+(1ll<<j)-1)<=n
                ; i++){
                pre[i][j]=min(pre[i][j-1],
                    pre[i + (1ll<<(j-1))][j
                    -1]);
            }
        }
    }
    int calc(int x,int y){
        if(x==y)return pre[x-1][0];
        int j = log2(y-x);
        return min(pre[x-1][j],pre[y- (1ll
            <<j)][j]);
    }

};
```

3.9 Sqrt Decomposition

```
// input data
int n;
vector<int> a (n);
// preprocessing
int len = (int) sqrt (n + .0) + 1; //
    size of the block and the number of
    blocks
vector<int> b (len);
for (int i=0; i<n; ++i)
    b[i / len] += a[i];
```

```
// answering the queries
for (;;) {
    int l, r;
  // read input data for the next query
    int sum = 0;
    for (int i=l; i<=r; )
        if (i % len == 0 && i + len - 1 <=
             r) {
            // if the whole block starting
                 at i belongs to [l, r]
            sum += b[i / len];
            i += len;
        }
        else {
            sum += a[i];
            ++i;
        }
}
```

3.10 Treap

```
/* Treap (with lazy prop) */
mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
template<class T>
struct Treap {
    T *root;
    using key_t = T::key_t;
    operator T*() const { return root; }
    explicit Treap(T *t = nullptr): root(t
        ) {}
    Treap(const auto& a): root(build(a, 0,
         a.size())) {}
    template<class... Args>
    Treap(int N, Args&&... args): Treap(
        vector<T>(N, T(forward<Args>(args)
        ...))) {}
    void heapify(T *t) {
        if(!t) return;
        T *mx = t;
        if(t->l and t->l->prio > mx->prio)
            mx = t->l;
        if(t->r and t->r->prio > mx->prio)
            mx = t->r;
        if(mx != t) {
            swap(t->prio, mx->prio);
            heapify(mx);
        }
    }
    // [l, r)
    T *build(const auto& a, int l, int r)
        {
        if(r <= l) return nullptr;
        int mid = l + r >> 1;
        T *t = new T(a[mid]);
        t->l = build(a, l, mid);
        t->r = build(a, mid + 1, r);
        t->pull();
        return heapify(t), t;
    }
    // l has subtree size sz
    static void split_by_sz(T *t, int sz,
        T* &l, T* &r) {
        if(!t) return l = r = nullptr,
            void();
        t->prop();
        if(sz <= (t->l? t->l->sz : 0))
            split_by_sz(t->l, sz, l, t->l)
                , r = t;
        else
            split_by_sz(t->r, sz - (t->l?
                t->l->sz : 0) - 1, t->r, r)
                , l = t;
        t->pull();
    }
    // r starts with key
    static void split_by_key(T *t, key_t
        key, T* &l, T* &r) {
        if(!t) return l = r = nullptr,
            void();
        t->prop();
        if(key <= t->key)
            split_by_key(t->l, key, l, t->
                l), r = t;
        else
            split_by_key(t->r, key, t->r,
                r), l = t;
        t->pull();
    }
    static void meld(T* &t, T *l, T *r) {
        if(l) l->prop();
        if(r) r->prop();
        if(!l or !r) t = l? l : r;
        else if(l->prio > r->prio)
            meld(l->r, l->r, r), t = l;
        else
            meld(r->l, l, r->l), t = r;
        if(t) t->pull();
    }
    // pos(i.e. sz) -> 0-based
    static void insert_at_pos(T* &t, int
        sz, T *nd) {
        if(!t) t = nd;
        else if(nd->prio > t->prio)
            split_by_sz(t, sz, nd->l, nd->
                r), t = nd;
        else {
            t->prop();
            int lsz = t->l? t->l->sz : 0;
            if(sz <= lsz)
                insert_at_pos(t->l, sz, nd)
                    ;
            else
                insert_at_pos(t->r, sz -
                    lsz - 1, nd);
        }
        t->pull();
    }
    // returns pos inserted (0-based)
    static int insert_key(T* &t, T *nd) {
        int res = 0;
        if(!t) t = nd;
        else if(nd->prio > t->prio) {
            split_by_key(t, nd->key, nd->l
                , nd->r);
            t = nd; res += (t->l? t->l->sz
                 : 0);
        } else {
            t->prop();
            if(nd->key > t->key)
                res += 1 + (t->l? t->l->sz
                    : 0);
```

```
        insert_key(nd->key < t->key? t
            ->l : t->r, nd);
    }
    t->pull();
    return res;
}
// pos(i.e. sz) -> 0-based
static void erase_at_pos(T* &t, int sz
    ) {
    if(!t) return;
    t->prop();
    int lsz = t->l? t->l->sz : 0;
    if(sz == lsz)
        meld(t, t->l, t->r);
    else if(sz < lsz)
        erase_at_pos(t->l, sz);
    else
        erase_at_pos(t->r, sz - lsz -
            1);
    if(t) t->pull();
}
// returns pos erased (0-based)
static int erase_key(T* &t, key_t key)
     {
    if(!t) return 0;
    int res = 0;
    t->prop();
    if(key == t->key)
        res += (t->l? t->l->sz : 0),
            meld(t, t->l, t->r);
    else {
        if(key > t->key)
            res += 1 + (t->l? t->l->sz
                : 0);
        erase_key(key < t->key ? t->l
            : t->r, key);
    }
    if(t) t->pull();
    return res;
}
// 0-based [lpos, rpos)
void erase_interval(int lpos, int rpos
    ) {
    T *x{nullptr}, *t{root}, *y{
        nullptr};
    if(rpos < root->sz)
        split_by_sz(root, rpos, t, y);
    if(lpos > 0)
        split_by_sz(t, lpos, x, t);
    meld(root, x, y);
}
// pos -> 0-based
void insert_at_pos(int pos, T *nd) {
    insert_at_pos(root, pos, nd); }
void erase_at_pos(int pos) {
    erase_at_pos(root, pos); }
void insert_key(key_t key) {
    insert_key(root, new T(key)); }
void erase_key(key_t key) { erase_key(
    root, key); }
// 0-based [lpos, rpos)
T query(int lpos, int rpos) {
    T *x{nullptr}, *t{root}, *y{
        nullptr};
    if(rpos < root->sz)
        split_by_sz(root, rpos, t, y);
    if(lpos > 0)
        split_by_sz(t, lpos, x, t);
    T res = *t;
    meld(t, x, t);
    meld(root, t, y);
    return res;
}
// 0-based [lpos, rpos)
template<class... Args>
void update(int lpos, int rpos, Args
    &&... args) {
    T *x{nullptr}, *t{root}, *y{
        nullptr};
    if(rpos < root->sz)
        split_by_sz(root, rpos, t, y);
    if(lpos > 0)
        split_by_sz(t, lpos, x, t);
    t->update(forward<Args>(args)...);
    meld(t, x, t);
    meld(root, t, y);
}
    void inorder(T *t, vector<T*>& res) {
        if(!t) return;
        t->prop();
        inorder(t->l, res);
        res.push_back(t);
        inorder(t->r, res);
        t->pull();
    }
    vector<T*> to_vector() {
        if(!root) return {};
        vector<T*> res;
        res.reserve(root->sz);
        inorder(root, res);
        return res;
    }
};
struct treap_node {
    int sz{1};
    mt19937::result_type prio{rng()};
    treap_node *l{nullptr}, *r{nullptr};
    using key_t = int;
    key_t key;
    treap_node(key_t key): key(key) {}
    void prop() {}
    void pull() {
        sz = 1;
        if(l) {
            l->prop();
            sz += l->sz;
        }
        if(r) {
            r->prop();
            sz += r->sz;
        }
    }
    void update() {}
};
```

4 Geometry

4.1 Area of simple Polygon

```
double area(const vector<point>& fig) {
```

```
    double res = 0;
    for (unsigned i = 0; i < fig.size(); i
        ++) {
        point p = i ? fig[i - 1] : fig.
            back();
        point q = fig[i];
        res += (p.x - q.x) * (p.y + q.y);
    }
    return fabs(res) / 2;
}
```

4.2 Check Point Belongs to Convex Polygon

```
struct pt {
    long long x, y;
    pt() {}
    pt(long long _x, long long _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); }
    long long cross(const pt &p) const {
        return x * p.y - y * p.x; }
    long long dot(const pt &p) const {
        return x * p.x + y * p.y; }
    long long cross(const pt &a, const pt
        &b) const { return (a - *this).
        cross(b - *this); }
    long long dot(const pt &a, const pt &b
        ) const { return (a - *this).dot(b
         - *this); }
    long long sqrLen() const { return this
        ->dot(*this); }
};
bool lexComp(const pt &l, const pt &r) {
    return l.x < r.x || (l.x == r.x && l.y
         < r.y);
}
int sgn(long long val) { return val > 0 ?
     1 : (val == 0 ? 0 : -1); }
vector<pt> seq;
pt translation;
int n;
bool pointInTriangle(pt a, pt b, pt c, pt
     point) {
    long long s1 = abs(a.cross(b, c));
    long long s2 = abs(point.cross(a, b))
        + abs(point.cross(b, c)) + abs(
        point.cross(c, a));
    return s1 == s2;
}
void prepare(vector<pt> &points) {
    n = points.size();
    int pos = 0;
    for (int i = 1; i < n; i++) {
        if (lexComp(points[i], points[pos
            ]))
            pos = i;
    }
    rotate(points.begin(), points.begin()
        + pos, points.end());
    n--;
    seq.resize(n);
    for (int i = 0; i < n; i++)
        seq[i] = points[i + 1] - points
            [0];
    translation = points[0];
}
bool pointInConvexPolygon(pt point) {
    point = point - translation;
    if (seq[0].cross(point) != 0 &&
            sgn(seq[0].cross(point)) !=
                sgn(seq[0].cross(seq[n -
                1])))
        return false;
    if (seq[n - 1].cross(point) != 0 &&
            sgn(seq[n - 1].cross(point))
                != sgn(seq[n - 1].cross(seq
                [0])))
        return false;
    if (seq[0].cross(point) == 0)
        return seq[0].sqrLen() >= point.
            sqrLen();
    int l = 0, r = n - 1;
    while (r - l > 1) {
        int mid = (l + r) / 2;
        int pos = mid;
        if (seq[pos].cross(point) >= 0)
            l = mid;
        else
            r = mid;
    }
    int pos = l;
    return pointInTriangle(seq[pos], seq[
        pos + 1], pt(0, 0), point);
```

4.3 Convex Hull Construction

```
struct pt {
    double x, y;
};
int orientation(pt a, pt b, pt c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)
        +c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-
        clockwise
    return 0;
}
bool cw(pt a, pt b, pt c, bool
    include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear &&
        o == 0);
}
bool ccw(pt a, pt b, pt c, bool
    include_collinear) {
    int o = orientation(a, b, c);
    return o > 0 || (include_collinear &&
        o == 0);
}
void convex_hull(vector<pt>& a, bool
    include_collinear = false) {
    if (a.size() == 1)
        return;
    sort(a.begin(), a.end(), [](pt a, pt b
        ) {
```

```
        return make_pair(a.x, a.y) <
            make_pair(b.x, b.y);
    });
    pt p1 = a[0], p2 = a.back();
    vector<pt> up, down;
    up.push_back(p1);
    down.push_back(p1);
    for (int i = 1; i < (int)a.size(); i
        ++) {
        if (i == a.size() - 1 || cw(p1, a[
            i], p2, include_collinear)) {
            while (up.size() >= 2 && !cw(
                up[up.size()-2], up[up.size
                ()-1], a[i],
                include_collinear))
                up.pop_back();
            up.push_back(a[i]);
        }
        if (i == a.size() - 1 || ccw(p1, a
            [i], p2, include_collinear)) {
            while (down.size() >= 2 && !
                ccw(down[down.size()-2],
                down[down.size()-1], a[i],
                include_collinear))
                down.pop_back();
            down.push_back(a[i]);
        }
    }
    if (include_collinear && up.size() ==
        a.size()) {
        reverse(a.begin(), a.end());
        return;
    }
    a.clear();
    for (int i = 0; i < (int)up.size(); i
        ++)
        a.push_back(up[i]);
    for (int i = down.size() - 2; i > 0; i
        --)
        a.push_back(down[i]);
}
```

4.4 Intersection of Line segments

```
struct pt {
    long long x, y;
    pt() {}
    pt(long long _x, long long _y) : x(_x)
        , y(_y) {}
    pt operator-(const pt& p) const {
        return pt(x - p.x, y - p.y); }
    long long cross(const pt& p) const {
        return x * p.y - y * p.x; }
    long long cross(const pt& a, const pt&
        b) const { return (a - *this).
        cross(b - *this); }
};
int sgn(const long long& x) { return x >=
    0 ? x ? 1 : 0 : -1; }
bool inter1(long long a, long long b,
    long long c, long long d) {
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return max(a, c) <= min(b, d);
}
bool check_inter(const pt& a, const pt& b
    , const pt& c, const pt& d) {
    if (c.cross(a, d) == 0 && c.cross(b, d
        ) == 0)
        return inter1(a.x, b.x, c.x, d.x)
            && inter1(a.y, b.y, c.y, d.y);
    return sgn(a.cross(b, c)) != sgn(a.
        cross(b, d)) &&
            sgn(c.cross(d, a)) != sgn(c.
                cross(d, b));
}
```

4.5 Intersection Point of Lines

```
struct pt {
    double x, y;
};
struct line {
    double a, b, c;
};
const double EPS = 1e-9;
double det(double a, double b, double c,
    double d) {
    return a*d - b*c;
}
bool intersect(line m, line n, pt & res)
    {
    double zn = det(m.a, m.b, n.a, n.b);
    if (abs(zn) < EPS)
        return false;
    res.x = -det(m.c, m.b, n.c, n.b) / zn;
    res.y = -det(m.a, m.c, n.a, n.c) / zn;
    return true;
}
bool parallel(line m, line n) {
    return abs(det(m.a, m.b, n.a, n.b)) <
        EPS;
}
bool equivalent(line m, line n) {
    return abs(det(m.a, m.b, n.a, n.b)) <
        EPS
         && abs(det(m.a, m.c, n.a, n.c)) <
            EPS
         && abs(det(m.b, m.c, n.b, n.c)) <
            EPS;
}
```

4.6 Sweep Line (Intersection of array of line segments)

```
const double EPS = 1E-9;
struct pt {
    double x, y;
};
struct seg {
    pt p, q;
    int id;

    double get_y(double x) const {
        if (abs(p.x - q.x) < EPS)
            return p.y;
```

```
        return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
    }
};
bool intersect1d(double l1, double r1, double l2, double r2) {
    if (l1 > r1)
        swap(l1, r1);
    if (l2 > r2)
        swap(l2, r2);
    return max(l1, l2) <= min(r1, r2) + EPS;
}
int vec(const pt& a, const pt& b, const pt& c) {
    double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
    return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
}
bool intersect(const seg& a, const seg& b)
{
    return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x) &&
           intersect1d(a.p.y, a.q.y, b.p.y, b.q.y) &&
           vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
           vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
}
bool operator<(const seg& a, const seg& b)
{
    double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
    return a.get_y(x) < b.get_y(x) - EPS;
}
struct event {
    double x;
    int tp, id;
    event() {}
    event(double x, int tp, int id) : x(x), tp(tp), id(id) {}
    bool operator<(const event& e) const {
        if (abs(x - e.x) > EPS)
            return x < e.x;
        return tp > e.tp;
    }
};
set<seg> s;
vector<set<seg>::iterator> where;
set<seg>::iterator prev(set<seg>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}
set<seg>::iterator next(set<seg>::iterator it) {
    return ++it;
}
pair<int, int> solve(const vector<seg>& a) {
    int n = (int)a.size();
    vector<event> e;
    for (int i = 0; i < n; ++i) {
        e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
        e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
    }
    sort(e.begin(), e.end());

    s.clear();
    where.resize(a.size());
    for (size_t i = 0; i < e.size(); ++i) {
        int id = e[i].id;
        if (e[i].tp == +1) {
            set<seg>::iterator nxt = s.lower_bound(a[id]), prv = prev(nxt);
            if (nxt != s.end() && intersect(*nxt, a[id]))
                return make_pair(nxt->id, id);
            if (prv != s.end() && intersect(*prv, a[id]))
                return make_pair(prv->id, id);
            where[id] = s.insert(nxt, a[id]);
        } else {
            set<seg>::iterator nxt = next(where[id]), prv = prev(where[id]);
            if (nxt != s.end() && prv != s.end() && intersect(*nxt, *prv))
                return make_pair(prv->id, nxt->id);
            s.erase(where[id]);
        }
    }

    return make_pair(-1, -1);
}
```

5 Graphs

5.1 Bellman Ford

```
int n,m;
cin>>n>>m;
vector<tuple<int,int,int>> edges(m);
vector<vector<int>> adj(n+1);
for(auto &i:edges){
    int x,y,z;cin>>x>>y>>z;
    i=make_tuple(x,y,z);
    adj[y].PB(x);
}

int distance[n+1]={};
int visited[n+1]={};

for (int i = 1; i <= n; i++) distance[i] = LLONG_MAX;
    distance[1] = 0;
    for (int i = 1; i <= n-1; i++){
```

```
    for (auto e : edges) {
    int a, b, w;
    tie(a, b, w) = e;
    if (distance[a] != LLONG_MAX &&
        distance[a] + w < distance[b])
         distance[b] = distance[a]+w;
    }
}

for (auto e : edges) {
    int a, b, w;
    tie(a, b, w) = e;
    if (distance[a] != LLONG_MAX &&
        distance[a] + w < distance[b])
    {
         cout<<"CYCLE"<<nl;
    }
}

cout<<distance[n]<<nl;
```

5.2 Dijkstras

```
int n,m;
vector<vector<pair<int,int>>> adj(n+1);
int st=1,ed=n;
priority_queue<pair<int,int>> q;
int dist[n+1];
int cnt[n+1]={};
int mn[n+1]={};
int mx[n+1]={};
bool visited[n+1]={};
for(int i=0;i<=n;i++){
    dist[i]=4e18;
}
q.push({0,st});
dist[st]=0;
cnt[st]=1;
while(!q.empty()){
    int x=q.top().SS;
    q.pop();
    if(visited[x])continue;
    visited[x]=true;
    // cout<<x+1<<sp<<dist[x]<<nl;
    for(auto i:adj[x]){
        if(dist[x]+i.SS == dist[i.FF]){
            cnt[i.FF]= (cnt[x]+cnt[i.FF])%
                mod;
            ckmin(mn[i.FF],mn[x]+1);
            ckmax(mx[i.FF],mx[x]+1);
        }
        else if(dist[x]+i.SS < dist[i.FF])
            {
            cnt[i.FF]= cnt[x];
            mn[i.FF]=mn[x]+1;
            mx[i.FF]=mx[x]+1;
            dist[i.FF]=dist[x]+i.SS;
            q.push({-dist[i.FF],i.FF});
        }
    }
}
cout<<dist[ed]<<sp<<cnt[ed]<<sp<<mn[ed]<<
    sp<<mx[ed]<<nl;
```

5.3 DSU

```
class unf{
    public :
    vector<int> link;
    vector<int> size;
    int comp=0;
    unf(int n = -1){
        link.assign(n+1,-1);
        size.assign(n+1,1);
        comp=n;
    }
    int find(int x){
        return link[x]==-1?x:find(link[x])
            ;
    }
    void unite(int x,int y){
        x=find(x);
        y=find(y);
        if(x==y)return ;
        if(size[x]<size[y])swap(x,y);
        size[x]+=size[y];
        link[y]=x;
        comp--;
    }

};
```

5.4 Eulerian Circuit

```
/* Eulerian Circuit (directed) */
// Hierholzers Algorithm
vector<int> indeg(n), outdeg(n);
vector<vector<int>> g(n);
for(i = 0; i < m; i++) {
    int u, v; cin >> u >> v;
    outdeg[--u]++; indeg[--v]++;
    g[u].push_back(v);
}
bool bad = false;
for(i = 1; i < n-1; i++)
    bad |= outdeg[i] ^ indeg[i];
bad |= outdeg[0]-indeg[0] ^ 1;
bad |= indeg[n-1]-outdeg[n-1] ^ 1;
if(bad)
    // IMPOSSIBLE
vector<int> tour;
tour.reserve(m+1);
Y([&](auto dfs, int v) -> void {
    while(outdeg[v]-- > 0)
        dfs(g[v][outdeg[v]]);
    tour.push_back(v);
})(0);     // dfs(0);

for(i = 0; i < n; i++)
    if(outdeg[i] > 0)
        // IMPOSSIBLE
reverse(tour.begin(), tour.end());

/* Eulerian Circuit (undirected) */
// Hierholzers Algorithm
struct edge_t {
    int u{-1}, v{-1}; bool done{false};
    edge_t() = default;
    edge_t(int u, int v) : u(u), v(v) {}
```

```
};
vector<int> deg(n);
vector<edge_t> edges;
edges.reserve(m);
vector<vector<edge_t*>> g(n);
for(i = 0; i < m; i++) {
    int u, v; cin >> u >> v;
    deg[--u]++; deg[--v]++;
    edges.emplace_back(u, v);
    g[u].push_back(&edges.back());
    g[v].push_back(&edges.back());
}
bool bad = false;
for(i = 0; i < n; i++)
    bad |= deg[i] & 1;
if(bad)
    // IMPOSSIBLE
vector<int> tour;
tour.reserve(m+1);
Y([&](auto dfs, int v) -> void {
    while(deg[v]-- > 0) {
        auto e = g[v][deg[v]];
        if(!e->done)
            e->done = true, dfs(e->u ^ e->
                v ^ v);
    } tour.push_back(v);
})(0);     // dfs(0);
for(i = 0; i < n; i++)
    if(deg[i] > 0)
        // IMPOSSIBLE
reverse(tour.begin(), tour.end());
```

5.5 Floyd Warshall

```
int n,m,q;
cin>>n>>m>>q;
int adj[n+1][n+1]={};
REP0(i,m){
    int x,y,z;cin>>x>>y>>z;
    if(adj[x][y]==0)
    adj[x][y]=z;
    else
    ckmin(adj[x][y],z);
    if(adj[y][x]==0)
    adj[y][x]=z;
    else
    ckmin(adj[y][x],z);
}
int distance[n+1][n+1]={};
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        if (i == j) distance[i][j] = 0;
        else if (adj[i][j]) distance[i][j]
            = adj[i][j];
        else distance[i][j] = 4e18;
    }
}
for (int k = 1; k <= n; k++) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            distance[i][j] = min(distance[
                i][j],
            distance[i][k]+distance[k][j])
                ;
        }
    }
}
while(q--){
    int x,y;cin>>x>>y;
    if(distance[x][y]>=4e18){
        cout<<-1<<nl;
    }
    else
        cout<<distance[x][y]<<nl;
}
```

5.6 Kruskal Algorithm for MST

```
int cost = 0;
sort(edges.begin(), edges.end(),[&](array
    <int,3> a,array<int,3> b){
    return a[2]<b[2];
});
for (auto e : edges) {
    if (find_set(e[0]) != find_set(e[1]))
        {
        cost += e[2];
        result.push_back(e);
        union_sets(e[0], e[1]);
    }
}
```

5.7 LCA

```
const int LG = 20;
struct LCA {
    vector<array<int, LG>> up;
    vector<int> tin, tout, depth;
    int n, timer;

    LCA(vector<vector<int>>& adj, int root
         = 0) : n(adj.size()) {
        up.resize(n);
        tin.resize(n);
        tout.resize(n);
        depth.resize(n);
        timer = 0;

        dfs(adj, root, root);
    };

    void dfs(vector<vector<int>>& adj, int
         u, int p) {
        tin[u] = ++timer;
        up[u][0] = p;

        if (u != p)
            depth[u] = depth[p] + 1;
        for (int i = 1; i < LG; ++i)
            up[u][i] = up[up[u][i - 1]][i
                - 1];

        for (int i : adj[u]) {
            if (i == p) continue;
            dfs(adj, i, u);
        }

        tout[u] = ++timer;
```

```
    }

    bool is_ancestor(int u, int v) {
        return tin[u] <= tin[v] and tout[u
            ] >= tout[v];
    }

    int query(int u,int t){
        int cnt=0;
        while(t){
            if(t&1){
                u = up[u][cnt];
            }
            cnt++;
            t>>=1;
        }
        return u;
    }

    int lca(int u, int v) {
        if (is_ancestor(u, v))
            return u;
        if (is_ancestor(v, u))
            return v;
        for (int i = LG - 1; i >= 0; --i)
            {
            if (!is_ancestor(up[u][i], v))
                u = up[u][i];
        }
        return up[u][0];
    }
};
```

5.8 SCC

```
class SCC{
public:
    int N; vector<vector<int>> adj, radj;
    vector<int> todo, comps, comp; vector<
        bool> vis;
    SCC(int _N) { N = _N;
        adj.resize(N), radj.resize(N),
            comp = vector<int>(N,-1), vis.
            resize(N); }
    void add(int x, int y) { adj[x].PB(y),
        radj[y].PB(x); }
    void dfs(int x) {
        vis[x] = 1; for(auto y:adj[x]) if
            (!vis[y]) dfs(y);
        todo.PB(x); }
    void dfs2(int x, int v) {
        comp[x] = v;
        for(auto y:radj[x]) if (comp[y] ==
            -1) dfs2(y,v); }
    void gen() {
        for(int i=0;i<N;i++) if (!vis[i])
            dfs(i);
        reverse(all(todo));
        for(auto x:todo) if (comp[x] ==
            -1) {
            dfs2(x,x); comps.PB(x); }
    }
};
```

5.9 Topological Sort

```
/* Topological sort */
int indeg[N];
vector<int> g[N];

for(i = 0; i < m; i++) {
    int u, v; cin >> u >> v; u--, v--;
    g[u].push_back(v);
    indeg[v]++;
}

auto topsort = [&]() {
    queue<int> q; vector<int> order;
    order.reserve(n);
    for(i = 0; i < n; i++)
        if(!indeg[i]) q.push(i);
    while(!q.empty()) {
        auto v = q.front(); q.pop();
        order.push_back(v);
        for(auto& x: g[v])
            if(!--indeg[x]) q.push(x);
    } return order.size() == n;   // cycle
        ? false : true
};
```

6 Misc

6.1 Bit Manipulation

```
//Check if Bit is set
bool is_set(unsigned int number, int x) {
    return (number >> x) & 1;
}

//Check if n is divisible by 2^k
bool isDivisibleByPowerOf2(int n, int k)
    {
    int powerOf2 = 1 << k;
    return (n & (powerOf2 - 1)) == 0;
}

//Check if n is a power of 2
bool isPowerOfTwo(unsigned int n) {
    return n && !(n & (n - 1));
}

//Clear the right most set bit
n&(n-1)

//Count set bits in n -> Brian Kernighan'
    s algorithm
int countSetBits(int n)
{
    int count = 0;
    while (n)
    {
        n = n & (n - 1);
        count++;
    }
    return count;
}

//Clear all trailing ones
n&(n+1)
```

```
//Sets the last cleared bit
n|(n+1)

//Extracts the last set bit
n&-n

//Built in functions
__builtin_popcount(unsigned int) returns
   the number of set bits (
   __builtin_popcount(0b0001'0010'1100)
   == 4)
__builtin_ffs(int) finds the index of the
    first (most right) set bit (
   __builtin_ffs(0b0001'0010'1100) == 3)
__builtin_clz(unsigned int) the count of
   leading zeros (__builtin_clz(0b0001'
   0010'1100) == 23)
__builtin_ctz(unsigned int) the count of
   trailing zeros (__builtin_ctz(0b0001'
   0010'1100) == 2)
__builtin_parity(x) the parity (even or
   odd) of the number of ones in the bit
   representation

//Iterating through all masks with their
   submasks. Complexity O(3^n)
for (int m=0; m<(1<<n); ++m)
    for (int s=m; s; s=(s-1)&m)
 ... s and m ...
```

6.2 Centroid Decomposition

```
int n,k;
vector<int> adj[n];
int sub[n] = {};
int rem[n] = {};
function<void(int,int)> get_subtree =[&]
    (int u,int p){
    sub[u] = 1;
    for(auto v:adj[u]){
        if(v==p || rem[v])continue;
        get_subtree(v,u);
        sub[u] += sub[v];
    }
};
function<int(int,int,int)> get_centroid
    =[&] (int u,int p,int sz){
    for (int v:adj[u]) {
        if(v==p ||rem[v])continue;
        if(sub[v]*2 > sz) {
            return get_centroid(v,u,sz);
        }
    }
    return u;
};
function<void(int,int,vector<int>&,int)>
    dfs = [&](int u,int p,vector<int>& sep
    ,int l){
    sep.push_back(l);
    for (int v:adj[u]) {
        if(v==p ||rem[v])continue;
        dfs(v,u,sep,l+1);
    }
};
int ans = 0;
function<void(int)> centroid_decomp =
    [&](int u){
    get_subtree(u,-1);
    int centroid = get_centroid(u,-1,sub[u
        ]);
    vector<vector<int>> st;
    for(auto v:adj[centroid]){
        if(rem[v])continue;
        vector<int> sep;
        dfs(v,centroid,sep,1);
        st.push_back(sep);
    }

    int sz = sub[u]+1;
    int mp[sz] = {};
    for(auto i:st){
        for(auto j:i)mp[j]++;
    }
    int sol = 0;
    for(auto i:st){
        for(auto j:i)mp[j]--;
        for(auto j:i)if(k-j>=0 && k-j<sz)
            sol += mp[k-j];
        for(auto j:i)mp[j]++;
    }
    sol/=2;
    ans+=sol;
    if(k<sz)ans+=mp[k];

    rem[centroid] = 1;
    for(auto v:adj[centroid]){
        if(rem[v])continue;
        centroid_decomp(v);
    }
};
centroid_decomp(0);
cout<<ans<<endl;
```

6.3 Convex Hull Trick for DP

```
// m1*x+c1 = m2*x+c2
// x = (c2-c1)/(m1-m2)
// mx+c
struct Line {
 int m, c;
 int value(int x) {
  return m*x + c;
 }
 bool better(Line l1, Line l2) {    //
    return true if the next line that is
    l2 is better than l1
  // intersection point between 2nd last
     line and last line.
  int numer1 = l1.c - c;
  int denom1 = m - l1.m;
  // intersection point between 2nd last
     line and cur line.
  int numer2 = l2.c - c;
  int denom2 = m - l2.m;
  // return true if the second
     intersection point <= first
     intersection point.
  return numer2*denom1 <= numer1*denom2;
 }
};
```

```
vi dp(N);
deque<Line> dq;
for(int i = 0; i < N; ++i) {
 while(dq.size() >= 2 && dq[0].value(h[i
    ]) >= dq[1].value(h[i])) dq.pop_front
    ();

 if(dq.size()) dp[i] = dq[0].value(h[i])
    + h[i]*h[i] + C;

 Line cur = {-2*h[i], dp[i] + h[i]*h[i]};
 while(dq.size() >= 2 && dq.end()[-2].
    better(dq.end()[-1], cur)) dq.
    pop_back();
 dq.push_back(cur);
}
cout << dp[N-1];
```

6.4 FFT

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

void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    if (n == 1)
        return;

    vector<cd> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++) {
        a0[i] = a[2*i];
        a1[i] = a[2*i+1];
    }
    fft(a0, invert);
    fft(a1, invert);

    double ang = 2 * PI / n * (invert ? -1
         : 1);
    cd w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2 * i < n; i++) {
        a[i] = a0[i] + w * a1[i];
        a[i + n/2] = a0[i] - w * a1[i];
        if (invert) {
            a[i] /= 2;
            a[i + n/2] /= 2;
        }
        w *= wn;
    }
}

//Polynomial Multiplication
vector<int> multiply(vector<int> const& a
    , vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(
       b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

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

//Long integer Multiplication
int carry = 0;
for (int i = 0; i < n; i++)
    result[i] += carry;
    carry = result[i] / 10;
    result[i] %= 10;
}
```

6.5 Gray Code

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

6.6 String multiplication

```
string multiply(string num1, string num2)
{
    int len1 = num1.size();
    int len2 = num2.size();
    if (len1 == 0 || len2 == 0)
        return "0";
    vector<int> result(len1 + len2, 0);
    int i_n1 = 0;
    int i_n2 = 0;
    for (int i = len1 - 1; i >= 0; i--)
    {
        int carry = 0;
        int n1 = num1[i] - '0';
        i_n2 = 0;
        for (int j = len2 - 1; j >= 0; j
            --)
        {
            int n2 = num2[j] - '0';
            int sum = n1 * n2 + result[
               i_n1 + i_n2] + carry;
            carry = sum / 10;
            result[i_n1 + i_n2] = sum %
               10;
            i_n2++;
        }
        if (carry > 0)
            result[i_n1 + i_n2] += carry;
        i_n1++;
    }
    int i = result.size() - 1;
    while (i >= 0 && result[i] == 0)
        i--;
    if (i == -1)
        return "0";
    string s = "";
    while (i >= 0)
        s += std::to_string(result[i--]);
```

```
    return s;
}
```

6.7 Ternary Search

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

6.8 Trie

```
class Trie {
public:
    Trie* children[26];
    bool check;
    Trie() {
        for(int i=0;i<26;i++)children[i]=
            NULL;
        check = false;
    }

    void insert(string word) {
        Trie* cur = this;
        for(auto i:word){
            if(!cur->children[i-'a']) cur
                ->children[i-'a'] = new
                Trie();
            cur = cur->children[i-'a'];
        }
        cur->check = true;
    }

    bool search(string word) {
        Trie* cur = this;
        for(auto i:word){
            if(!cur->children[i-'a'])
                return false;
            cur = cur->children[i-'a'];
        }
        return cur->check;
    }

    bool startsWith(string prefix) {
        Trie* cur = this;
        for(auto i:prefix){
            if(!cur->children[i-'a'])
                return false;
            cur = cur->children[i-'a'];
        }
        return true;
    }
};
```

7 Number Theory

7.1 Chinese Remainder Theorem

```
struct Congruence {
    long long a, m;
};
//Only Works for coprime modulo O(n*log
    ^3(n))
long long chinese_remainder_theorem(
    vector<Congruence> const& congruences)
     {
    long long M = 1;
    for (auto const& congruence :
        congruences) {
        M *= congruence.m;
    }

    long long solution = 0;
    for (auto const& congruence :
        congruences) {
        long long a_i = congruence.a;
        long long M_i = M / congruence.m;
        long long N_i = mod_inv(M_i,
            congruence.m);
        solution = (solution + a_i * M_i %
             M * N_i) % M;
    }
    return solution;
}
//Garner's Algorithm -> each digit has
   different modulo -> x1,x2...,xk has
   modulo p1,p2,...,pk respectively
// r_ij = (pi)^-1 (mod pj)
for (int i = 0; i < k; ++i) {
    x[i] = a[i];
    for (int j = 0; j < i; ++j) {
        x[i] = r[j][i] * (x[i] - x[j]);

        x[i] = x[i] % p[i];
        if (x[i] < 0)
            x[i] += p[i];
    }
}
```

7.2 Collorary

7.2.1 Number of Divisors,Sum of Divisors

$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$
$d(n) = (e_1+1) \cdot (e_2+1) \cdots (e_k+1)$

$$\sigma(n) = \frac{p_1^{e_1+1}-1}{p_1-1} \cdot \frac{p_2^{e_2+1}-1}{p_2-1} \cdots \frac{p_k^{e_k+1}-1}{p_k-1}$$

7.2.2 Euler Totient Function

$a^n \equiv a^{n \bmod \phi(m)} \pmod m$
$x^n \equiv x^{\phi(m)+[n \bmod \phi(m)]} \mod m$

7.2.3 Mobius Function

$\sum_{dn} \mu(d) = e(n) = [n = 1]$

Example:

$f(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} [\gcd(i, j) = 1]$
$f(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{d(\sec(h,j)} \mu(d)$
$f(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{d=1}^{n} [d \mid \gcd(i, j)]\mu(d)$
$f(n) = \sum_{i=1}^{n} \mu(d) \left(\sum_{i=1}^{n} [d \mid i]\right) \left(\sum_{j=1}^{n} [d \mid i]\right)$
$f(n) = \sum_{d=1}^{n} \mu(d) \left\lfloor \frac{n}{d} \right\rfloor^2$

7.2.4 Pólya enumeration theorem

$|\text{Classes}| = \frac{1}{|G|} \sum_{\pi \in G} k^{C(\pi)}$
$C(\pi)$ is number of cycles in the permutation $\pi, k is the number of values that each representation element can take.$

7.3 Extended Euclidean

```
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q
            * x1);
        tie(y, y1) = make_tuple(y1, y - q
            * y1);
        tie(a1, b1) = make_tuple(b1, a1 -
            q * b1);
    }
    return a1;
}
```

7.4 Fast Fibonnachi

```
pair<int,int> fib(int n){
    // return fib(n) and fib(n+1)
    if(n==0)return {0,1};
    auto [i,j] = fib(n/2);
    int c = (i * ((2*j - i+mod)%mod))%mod;
    int d = ((i *i%mod) + (j*j%mod))%mod;
    if(n%2)return {d,(c+d)%mod};
    return {c,d};
}
```

7.5 Linear Diophantine Equation

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

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

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

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

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

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

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

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

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

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

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

7.6 Matrix Expo

```
struct matrix {
    int n,m;
    vector<vector<int>> mat;
    matrix(int n,int m){
        this->n = n;
        this->m = m;
        mat.resize(n,vector<int>(m,0));
    }
    matrix(int n) : matrix(n,n){}
    void identity(){
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(i==j)mat[i][j] = 1;
            }
        }
    }
    matrix friend operator *(const matrix
        &a, const matrix &b){
        matrix c(a.n,b.m);
        for (int i = 0; i < a.n; i++) {
          for (int j = 0; j < b.m; j++) {
              c.mat[i][j] = 0;
              for (int k = 0; k < b.n; k++)
                  {
                  (c.mat[i][j] += a.mat[i][
                      k] * b.mat[k][j])%=mod
                      ;
              }
          }
        }
        return c;
    }
};
matrix matpow(matrix base, long long n) {
    matrix ans(base.n);
    ans.identity();
    while (n) {
        if(n&1)
            ans = ans*base;
        base = base*base;
        n >>= 1;
    }
    return ans;
}
```

7.7 Mod Expo

```
const int MOD = 1e9 + 7;
// O(logp)
int mod_expo(int x, int p, int mod = MOD)
    {
    int res = 1; x = ((x % mod) + mod) %
        mod;
    while (p > 0) {
        if (p & 1) res = (1LL * res * x) %
            mod;
        x = (1LL * x * x) % mod; p /= 2;
    }
    return res;
}
//for mod-2 for prime modulo, for coprime
    use phi(mod) - 1
inline int mod_inv(int x, int mod = MOD)
    { return mod_expo(x, mod - 2, mod); }
```

7.8 Mod Inverse

```
//Quick Mod inverse O(50)
int mod_inv(int a,int mod = mod) {
    return a <= 1 ? a : mod - (long long)(
        mod/a) * mod_inv(mod % a) % mod;
}

//Quick Mod inverse with DP (1...N)
const int N = 1e6;
vector<int> inv_dp(N+1);
int mod_inv(int a,int mod = mod) {
    if(inv_dp[a])return inv_dp[a];
    return inv_dp[a] = a <= 1 ? a : mod -
        (long long)(mod/a) * mod_inv(mod %
         a) % mod;
}

//Finding Mod Inverse for Array of
    integers in O(n)
std::vector<int> invs(const std::vector<
    int> &a, int m) {
    int n = a.size();
    if (n == 0) return {};
    std::vector<int> b(n);
    int v = 1;
    for (int i = 0; i != n; ++i) {
        b[i] = v;
        v = static_cast<long long>(v) * a[
            i] % m;
    }
    int x, y;
    gcd(v, m, x, y);
    x = (x % m + m) % m;
    for (int i = n - 1; i >= 0; --i) {
        b[i] = static_cast<long long>(x) *
             b[i] % m;
        x = static_cast<long long>(x) * a[
            i] % m;
    }
    return b;
}
```

7.9 nCk$_{d}$*ouble*

```
// nCk for doubles. n can go till 1e5
int mxN = 1e5+7;
double fac[mxN];
double choose(int n, int k) {
 if(k < 0 || k > n) return 0;
 return exp(-(fac[n] - fac[k] - fac[n-k])
     );
}

void preprocess() {

 fac[0] = 0.0;
 for(int i = 1; i < mxN; ++i) {
  fac[i] = fac[i-1] + log(i);
 }
}
```

7.10 Permutation Expo and Enumeration

```
//Applying a set of permutations k times
    O(nlogk)
using Permutation = vector<int>;
void operator*=(Permutation& p,
    Permutation const& q) {
    Permutation copy = p;
    for (int i = 0; i < p.size(); i++)
        p[i] = copy[q[i]];
}
//Permutation Expo
Permutation permute(Permutation sequence,
     Permutation permutation, long long k)
     {
    while (k > 0) {
        if (k & 1) {
            sequence *= permutation;
        }
        permutation *= permutation;
        k >>= 1;
    }
    return sequence;
}
//Count Cycles in Permutation
int count_cycles(Permutation p) {
    int cnt = 0;
    for (int i = 0; i < p.size(); i++) {
        if (p[i] != -1) {
            cnt++;
            for (int j = i; p[j] != -1;) {
                int next = p[j];
                p[j] = -1;
                j = next;
            }
        }
    }
    return cnt;
}
//A group should be invariant under
    transformation
int solve(int n, int m) {
    Permutation p(n*m), p1(n*m), p2(n*m),
        p3(n*m);
    for (int i = 0; i < n*m; i++) {
        p[i] = i;
        p1[i] = (i % n + 1) % n + i / n *
            n;
        p2[i] = (i / n + 1) % m * n + i %
            n;
        p3[i] = (m - 1 - i / n) * n + (n -
             1 - i % n);
    }

    set<Permutation> s;
    for (int i1 = 0; i1 < n; i1++) {
        for (int i2 = 0; i2 < m; i2++) {
            for (int i3 = 0; i3 < 2; i3++)
                 {
                s.insert(p);
                p *= p3;
            }
            p *= p2;
        }
        p *= p1;
    }

    int sum = 0;
    for (Permutation const& p : s) {
        sum += 1 << count_cycles(p);
    }
    return sum / s.size();
}
```

7.11 Prime and Prime Factorization

```
uint64_t modmul(uint64_t a, uint64_t b,
    uint64_t M) {
    int64_t ret = a * b - M * uint64_t(1.L
         / M * a * b);
    return ret + M * (ret < 0) - M * (ret
        >= (int64_t)M);
}

//modpow for long long numbers
uint64_t modpow(uint64_t b, uint64_t e,
    uint64_t mod) {
    uint64_t ans = 1;
    for(; e; b = modmul(b, b, mod), e /=
        2)
        if(e & 1) ans = modmul(ans, b, mod
            );
    return ans;
}

//MillerRabbin Prime checking O(21*log^3(
    n))
bool isPrime(uint64_t n) {
    if(n < 2 || n % 6 % 4 != 1) return (n
        | 1) == 3;
    uint64_t A[] = {2, 325, 9375, 28178,
        450775, 9780504, 1795265022},
        s = __builtin_ctzll(n-1), d = n >>
            s;
    for(uint64_t a: A) {
        uint64_t p = modpow(a%n, d, n), i
            = s;
        while(p != 1 && p != n - 1 && a %
            n && i--)
            p = modmul(p, p, n);
        if(p != n-1 && i != s) return 0;
    } return 1;
}

//Strongest Prime Factorization technique
uint64_t pollard(uint64_t n) {
    auto f = [n](uint64_t x) { return
        modmul(x, x, n) + 1; };
    uint64_t x = 0, y = 0, t = 0, prd = 2,
         i = 1, q;
    while(t++ % 40 || __gcd(prd, n) == 1)
        {
        if(x == y) x = ++i, y = f(x);
        if((q = modmul(prd, max(x,y) - min
            (x,y), n))) prd = q;
        x = f(x), y = f(f(y));
    } return __gcd(prd, n);
}

vector<uint64_t> factor(uint64_t n) {
    if(n == 1) return {};
    if(isPrime(n)) return {n};
    uint64_t x = pollard(n);
    auto l = factor(x), r = factor(n / x);
```

```
    l.insert(l.end(), r.begin(), r.end());
        sort(l.begin(), l.end());
    return l;
}
// int64_t x = 69696969234234 24;
// auto factors = factor(x);
// ~ {2, 2, 2, 2, 2, 2, 3, 37, 281, 2029,
    1720769}
```

7.12 Sieve

```
//Sieve of Eratosthenes O(nloglogn)
int n;
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i * i <= n; i++) {
    if (is_prime[i]) {
        for (int j = i * i; j <= n; j += i
            )
            is_prime[j] = false;
    }
}

// Linear Sieve O(n) but large constant
   factor
// lp[i] Minimum prime factor of i,
   useful for factorizing
const int N = 1e6;
vector<int> lp(N+1);
vector<int> pr;
for (int i=2; i <= N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i;
        pr.push_back(i);
    }
    for (int j = 0; i * pr[j] <= N; ++j) {
        lp[i * pr[j]] = pr[j];
        if (pr[j] == lp[i]) {
            break;
        }
    }
}
```

```
//Segmented Sieve to Find Primes between
    L and R, O((R-L+1 + sqrt(R))loglogR)
vector<char> segmentedSieve(long long L,
    long long R) {
    // generate all primes up to sqrt(R)
    long long lim = sqrt(R);
    vector<char> mark(lim + 1, false);
    vector<long long> primes;
    for (long long i = 2; i <= lim; ++i) {
        if (!mark[i]) {
            primes.emplace_back(i);
            for (long long j = i * i; j <=
                lim; j += i)
                mark[j] = true;
        }
    }
    vector<char> isPrime(R - L + 1, true);
    for (long long i : primes)
        for (long long j = max(i * i, (L +
            i - 1) / i * i); j <= R; j +=
            i)
            isPrime[j - L] = false;
    if (L == 1)
        isPrime[0] = false;
    return isPrime;
}

//Harmonic -> O(2*sqrt(n))
for (int i = 1, la; i <= n; i = la + 1) {
    la = n / (n / i);
    //n / x yields the same value for i <=
        x <= la.
}

// Euler Totient function O(sqrt(n))
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
```

```
    if (n > 1)
        result -= result / n;
    return result;
}

// Euler Totient function pre-calc using
   sieve O(nloglogn)
void phi(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i
                )
                phi[j] -= phi[j] / i;
        }
    }
}

//Linear Sieve to find Mobius function O(
   n)
const int N = 1e6;
vector<int> lp(N+1);
vector<int> mob(N+1);
vector<int> pr;

mob[1] = 1;
for (int i=2; i <= N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i;
        mob[i] = -1;
        pr.push_back(i);
    }
    for (int j = 0; i * pr[j] <= N; ++j) {
        lp[i * pr[j]] = pr[j];
        if(i%pr[j])mob[i * pr[j]] = mob[i
            ]*mob[pr[j]];
        else mob[i * pr[j]] = 0;

        if (pr[j] == lp[i]) {
            break;
        }
    }
```

```
}
```

8 Strings

8.1 KMP

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

8.2 Manacher

```
vector<int> manacher(string s) {
    string t = "#";
    for(auto c: s) {
        t += c + string("#");
    }
    int n = t.size();

    vector<int> res(n);
    for(int i = 0, j = 0; i < n; i++) {
        if(2*j - i >= 0 && j + res[j] > i)
            {
            res[i] = min(res[2*j - i], j +
                res[j] - i);
        }
        while(i - res[i] >= 0 && i + res[i
            ] < n && t[i - res[i]] == t[i +
            res[i]]) {
            res[i] += 1;
        }
        if(i + res[i] > j + res[j]) {
            j = i;
        }
    }
    return res;
}

// both l and r are inclusive
// res is return vector of manacher
bool checkPalindrome(int l, int r, vector
    <int>& res) {
    return res[l + r + 1] >= (r - l + 1);
}
```

8.3 Rabin Karp

```
//Rabin Karp -> O(s)
vector<int> rabin_karp(string const& s,
    string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();

    vector<long long> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size();
         i++)
        p_pow[i] = (p_pow[i-1] * p) % m;

    vector<long long> h(T + 1, 0);
    for (int i = 0; i < T; i++)
        h[i+1] = (h[i] + (t[i] - 'a' + 1)
            * p_pow[i]) % m;
    long long h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - 'a' + 1) *
            p_pow[i]) % m;

    vector<int> occurrences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (h[i+S] + m - h[
            i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            occurrences.push_back(i);
    }
    return occurrences;
}
```

8.4 String Hashing

```
/* String hashing w/ Roling/Polynomial
   hash */
// Single hash
namespace Hashing {
#ifndef __MOD_BASE
    #define __MOD_BASE
    constexpr int _mod = 1e9 + 123; //
        default mod
    mt19937 rng(chrono::steady_clock::now
        ().time_since_epoch().count());
    static const int _base =
        uniform_int_distribution<int>(256,
        _mod - 2)(rng) | 1; // random
        base
#endif
    // use base and primary mod of your
        choice
    template<const int& base = _base, int
        mod = _mod>
    struct single_hash {
        static inline vector<int> pows{1};
        int n;
        vector<int> suf;
        void build(const string& s) {
            n = s.size();
            assert(base < mod);
            suf.resize(n + 1); pows.
                reserve(n + 1);
            while(pows.size() <= n)
                pows.push_back(1LL * pows.
                    back() * base % mod);
            for(int i = n - 1; ~i; i--)
                suf[i] = (1ll * suf[i + 1]
                    * base + s[i]) % mod;
        }
        // hash [l, r)
        int operator()(int l, int r) const
            {
```

```
            int res = suf[l] - 1ll * suf[r
                ] * pows[r - l] % mod;
            return res < 0? res + mod :
                res;
        }
        int operator()() const { return (*
            this)(0, n); }
    };
}
// double hash
namespace Hashing {
#ifndef __MOD_BASE
    #define __MOD_BASE
    constexpr int _mod = 1e9 + 123;
    mt19937 rng(chrono::steady_clock::now
        ().time_since_epoch().count());
    static const int _base =
        uniform_int_distribution<int>(256,
         _mod - 2)(rng) | 1;
#endif
    using hash_t = pair<int, uint64_t>;
    vector<uint64_t> pow2{1};
    template<const int& base = _base, int
        mod = _mod>
    struct double_hash {
        static inline vector<int> pow1{1};
        int n;
        vector<int> suf1;
        vector<uint64_t> suf2;
        void build(const string& s) {
            n = s.size();
            assert(base < mod);
            suf1.resize(n + 1); suf2.
                resize(n + 1);
            pow1.reserve(n + 1); pow2.
                reserve(n + 1);
            while(pow1.size() <= n)
                pow1.push_back(1LL * pow1.
                    back() * base % mod);
            while(pow2.size() <= n)
                pow2.push_back(pow2.back()
                    * base);
            for(int i = n - 1; ~i; i--) {
                suf1[i] = (1ll * suf1[i +
                    1] * base + s[i]) % mod;
                suf2[i] = suf2[i + 1] *
                    base + s[i];
            }
        }
        // hash [l, r)
        hash_t operator()(int l, int r)
            const {
            int res1 = suf1[l] - 1ll *
                suf1[r] * pow1[r - l] % mod
                ;
            if(res1 < 0) res1 += mod;
            uint64_t res2 = suf2[l] - suf2
                [r] * pow2[r - l];
            return {res1, res2};
        }
        hash_t operator()() const { return
             (*this)(0, n); }
    };
}
/*
int main() {
    string s = "absedfd$%#&@sdA01";
    static const int base = 23;
    Hashing::single_hash<base, int(1e9+7)>
        A(s);
    Hashing::double_hash<> B(s);

    cout << A(3, 4) << '\n';
    auto [x, y] = B();
    cout << x << ' ' << y;
}
*/
```

8.5 Z function

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