Team Notebook

NSU_Team_Aseh

December 4, 2024

Contents

1 —

1.1 0 Sublime Setup

```
// Tools -> Build System -> New Build System (write script &
     save)
// Tools -> Build System (select recently created script)
{
"cmd" : ["g++ -std=c++20 $file_name -o $file_base_name &&
     timeout 10s ./$file_base_name < input.txt > output.txt
     2> debug.txt && rm $file_base_name"],
"selector" : "source.cpp",
"shell": true,
"working_dir" : "$file_path"
}

// Press ‘Alt + Shift + 4‘ to split window in 4 parts.
// save ’inputf.in’, ’outputf.in’, ’debugf.in’
// Press ‘Ctrl + B‘ to run code.

/// Precompile HeaderFile
// just go to file explorer and serach ’stdc++.h’
// go to that folder and open folder in terminal
// sudo g++ -std=c++20 stdc++.h
// stdc++.h.gch is created precompile done

// preferences -> settings add "save_on_window_deactivation
     ": true

// windows
"cmd": [ "g++.exe", "-std=c++14", "${file}", "-o", "${
     file_base_name}.exe", "&&", "${file_base_name}.exe",
"<", "input.txt", ">", "output.txt", "2>", "debug.txt", "&&"
     , "del", "${file_base_name}.exe"],
```

1.2 1 CP Snippet [M]

```
#include <bits/stdc++.h>
using namespace std;
#define endl ’\n’
#define ll long long
#define len(v) (int) v.size()
#define all(v) v.begin(), v.end()

#define input(v) for(auto&x:v)cin>>x;
#define print(v) for(auto&x:v)cout<<x<<’ ’;cout<<endl;
#define dbg(a) cout<<#a<<" = "<<a<<endl;
void solve()
{

}

int32_t main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int t = 1, tc = 1;
    // cin >> t;
    while (t--) {
        // cout << "Case " << tc++ << ": ",
        solve();
    }

    return 0;
}
```

1.3 2 Random Input Generator [M]

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

// Random input generator
auto seed = chrono::high_resolution_clock::now().
     time_since_epoch().count();
std::mt19937 mt(seed);
int myrand(int mod) {
    return mt()%mod;
}
// Generates a random number within 100
// int random_num = myrand(100) + 1;
```

1.4 3 Double Inequality [M]

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

// EPS | Double Inequality
const double eps = 1e-9;
bool isEqual(double a, double b) {return abs(a-b) <= eps;}
     // a == b
bool isSmaller(double a, double b) {return a+eps < b;} // a
     < b
bool isGreater(double a, double b) {return a > b+eps;} // a
     > b
bool isInt(double a) {return isEqual(ceil(a) - a, 0);} //
     isInt(num)
```

1.5 4 Custom Hashing [Bashem]

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

// For gp_hash_table
#include <ext/pb_ds/assoc_container.hpp>

using namespace __gnu_pbds;

using namespace std;

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::
             steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

// Example Use
unordered_map<int, int, custom_hash> mp;

// Faster
gp_hash_table<int, int, custom_hash> mp;
```

1.6 Stress Test - Shell [SA]

```
for ((i = 1; i <= 1000; ++i)); do
    echo Testing $i
    ./gen >in.txt
    ./main <in.txt >out1.txt
    ./brute <in.txt >out2.txt
    diff -w out1.txt out2.txt || break
done
```

2 Data Structures

2.1 1 Big Integer Implementation [M]

```cpp
// big integer or int128
using int128 =  signed __int128;
using uint128 = unsigned __int128;

namespace int128_io {
    inline auto char_to_digit(int chr) {
        return static_cast<int>(isalpha(chr) ? 10+tolower(chr)-'a': chr-'0'); }
    inline auto digit_to_char(int digit) {
        return static_cast<char>(digit > 9 ? 'a'+digit-10: '0'+digit); }

    template<class integer>
    inline auto to_int(const std::string &str, size_t *idx = nullptr, int base = 10) {
        size_t i = idx != nullptr ? *idx : 0;
        const auto n = str.size();
        const auto neg = str[i] == '-';
        integer num = 0;
        if (neg) ++i;
        while (i < n) { num *= base, num += char_to_digit(str[i++]); }
        if (idx != nullptr) *idx = i;
        return neg ? -num : num; }

    template<class integer>
    inline auto to_string(integer num, int base = 10) {
        const auto neg = num < 0;
        std::string str;
        if (neg) num = -num;
        do str += digit_to_char(num%base), num /= base;
        while (num > 0); if (neg) str += '-';
        std::reverse(str.begin(),str.end());
        return str; }

    inline auto next_str(std::istream &stream) { std::string str; stream >> str; return str; }

    template<class integer>
    inline auto& read(std::istream &stream, integer &num) {
        num = to_int<integer>(next_str(stream));
        return stream; }

    template<class integer>
    inline auto& write(std::ostream &stream, integer num) {
        return stream << to_string(num); } }

using namespace std;

inline auto& operator>>(istream &stream, int128 &num) {
    return int128_io::read(stream,num); }
inline auto& operator>>(istream &stream, uint128 &num) {
    return int128_io::read(stream,num); }
inline auto& operator<<(ostream &stream, int128 num) {
    return int128_io::write(stream,num); }
inline auto& operator<<(ostream &stream, uint128 num) {
    return int128_io::write(stream,num); }

inline auto uint128_max() {
    uint128 ans = 0;
    for (uint128 pow = 1; pow > 0; pow <<= 1)
        ans |= pow;
    return ans; }

// (direct assign not supported yet)
// int128 a, b; cin >> a >> b;
// uint128 a, b; cout << a << b;
```

2.2 11 DSU [M]

```cpp
// DSU
struct DSU {
    vector<int> e;
    DSU(int N) { e = vector<int>(N, -1); }

    int size(int x) { return -e[get(x)]; }
    int get(int x) { return e[x] < 0 ? x : e[x] = get(e[x]); }
    bool same_set(int a, int b) { return get(a) == get(b); }

    bool unite(int x, int y) {
        x = get(x), y = get(y);
        if (x == y) return false;
        if (e[x] > e[y]) swap(x, y);
        e[x] += e[y];
        e[y] = x;
        return true;
    }
};
// DSU dsu(n+1);
// dsu.unite(x, y);
// dsu.same_set(x, y);
```

2.3 2 Custom Priority Queue [M]

```cpp
/// Custom Priority Queue
#define pii pair<int, int>
struct comp{
    bool operator()(pii& a, pii& b){
        return a.second < b.second;
    }
};
priority_queue<pii, vector<pii>, comp> pq;
```

2.4 2D Prefix Sum [SA]

```cpp
const int N = 1000, M = 500;
int a[N + 1][M + 1], pref[N + 1][M + 1];
// 1-based
void build() {
    for (int i = 1; i <= N; ++i) {
        for (int j = 1; j <= M; ++j) {
            pref[i][j] = pref[i - 1][j] + pref[i][j - 1] - pref[i - 1][j - 1] + a[i][j];
        }
    }
}
// top_left(i, j), right_bottom(k, l)
auto query(int i, int j, int k, int l) {
    return pref[k][l] - pref[i - 1][l] - pref[k][j - 1] + pref[i - 1][j - 1];
}
```

2.5 3 PBDS Indexed Set (Order Statistics Tree) [M]

```cpp
// pbds set // more like a indexed set
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

typedef tree<int, null_type, less<int>,
rb_tree_tag,tree_order_statistics_node_update> pbds;

/*  pbds s; s.insert(x);
    int value = *s.find_by_order(index);
    int index = s.order_of_key(value); */
```

2.6 4

```
// pbds multiset // more like a indexed multiset
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template<class T>
class _multiset{
    using MS = tree<T, null_type, less_equal<T>,
    rb_tree_tag, tree_order_statistics_node_update>;
    MS s;
public:
    _multiset(){s.clear();}
    void erase(T xx){s.erase(s.upper_bound(xx));}
    typename MS::iterator lower_bound(T xx){return s.
         upper_bound(xx);}
    typename MS::iterator upper_bound(T xx){return s.
         lower_bound(xx);}
    // same
    size_t size(){return s.size();}
    void insert(T xx){s.insert(xx);}
    T find_by_order(int xx){return s.find_by_order(xx);}
    int order_of_key(T xx){return s.order_of_key(xx);}
    void erase(typename MS::iterator xx){s.erase(xx);}
};
```

2.7 5a Segment Tree [M]

```
template <class T>
struct SegmentTree{
private:
 int n;
 vector<T> tree;

 void buildTree(const vector<T>& v, int node, int b, int e){
  if(b==e){tree[node] = v[b];return;}
  int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
  buildTree(v, ln, b, mid);
  buildTree(v, rn, mid+1, e);
  tree[node] = merge(tree[ln],tree[rn]);
 }

 T query(int node, int b, int e, int l, int r){
  if(l > e or r < b) return identity;
  if(l<=b and r>=e) return tree[node];
  int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
  T c1 = query(ln, b, mid, l, r);
  T c2 = query(rn, mid+1, e, l, r);
  return merge(c1,c2);
 }

 void set(int node, int b, int e, int ind, T val){
  if(ind > e or ind < b) return;
  if(ind<=b and ind>=e){
   tree[node] = val;
   return;
  }
  int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
  if (ind <= mid) set(ln, b, mid, ind, val);
  else set(rn, mid+1, e, ind, val);
  tree[node] = merge(tree[ln],tree[rn]);
 }

 void update(int node, int b, int e, int ind, T val){
  if(ind > e or ind < b) return;
  if(ind<=b and ind>=e){
   tree[node] = merge(tree[node], val);
   return;
  }
  int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
  if (ind <= mid) update(ln, b, mid, ind, val);
  else update(rn, mid+1, e, ind, val);
  tree[node] = merge(tree[ln],tree[rn]);
 }

public:
 T query(int l, int r){return query(1, 0, n-1, l, r);}
 void set(int ind, T val){set(1, 0, n-1, ind, val);}
 void update(int ind, T val){update(1, 0, n-1, ind, val);}

 SegmentTree(const vector<T>& input) {
  n = input.size();
  int sz = n<<2; // 4n
  tree.resize(sz);
  buildTree(input, 1, 0, n-1);
 }

 T merge(const T& a, const T& b) { return a + b; }
 T identity = 0;
};

/*
 vector<int> v(n); cin >> v;

 SegmentTree<int> segTree(v); // All 0 based index

 segTree.query(left-1, right-1);
 segTree.set(index-1, value);
 segTree.update(index-1, increasingValue);
*/
```

2.8 5b Segment Tree - Lazy [M]

```
template <class T>
struct LazySegtree{
private:
 int n;
 vector<T> tree;
 vector<T> addTree, setTree;

 void buildTree(const vector<T>& v, int node, int b, int e){
  if(b==e){tree[node] = v[b];return;}
  int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
  buildTree(v, ln, b, mid);
  buildTree(v, rn, mid+1, e);
  tree[node] = merge(tree[ln],tree[rn]);
 }

 void propagate(int node, int b, int e){
  int ln = node<<1, rn = ln+1;
  if(setTree[node]!=set_identity){
   addTree[node]=add_identity;
   tree[node] = setTree[node]*(e-b+1);
   if(b!=e){
    setTree[ln]=setTree[node];
    setTree[rn]=setTree[node];
   }
   setTree[node]=set_identity;
  }
  else{
   if(addTree[node]==add_identity) return;
   tree[node]+=addTree[node]*(e-b+1);
   if(b!=e){
    if(setTree[ln]==set_identity){
     addTree[ln]+=addTree[node];
    }
    else{
     setTree[ln]+=addTree[node];
     addTree[ln]=0;
    }
    if(setTree[rn]==set_identity){
     addTree[rn]+=addTree[node];
    }
    else{
     setTree[rn]+=addTree[node];
     addTree[rn]=0;
    }
```

```
  }
  addTree[node]=add_identity;
 }
}

T query(int node, int b, int e, int l, int r){
 propagate(node, b, e);
 if(l > e or r < b) return identity;
 if(l<=b and r>=e) return tree[node];
 int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
 T c1 = query(ln, b, mid, l, r);
 T c2 = query(rn, mid+1, e, l, r);
 return merge(c1,c2);
}

void range_set(int node, int b, int e, int l, int r, T val)
     {
 propagate(node, b, e);
 if(l > e or r < b) return;
 if(l<=b and r>=e){
  setTree[node]=val;
  propagate(node, b, e);
  return;
 }
 int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
 range_set(ln, b, mid, l, r, val);
 range_set(rn, mid+1, e, l, r, val);

 tree[node]=merge(tree[ln],tree[rn]);
}

void range_update(int node, int b, int e, int l, int r, T
     val){
 propagate(node, b, e);
 if(l > e or r < b) return;
 if(l<=b and r>=e){
  addTree[node]+=val;
  propagate(node, b, e);
  return;
 }
 int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
 range_update(ln, b, mid, l, r, val);
 range_update(rn, mid+1, e, l, r, val);

 tree[node]=merge(tree[ln],tree[rn]);
 return;
}

public:
 T query(int l, int r){return query(1, 0, n-1, l, r);}
 void range_set(int l, int r, T value){ range_set(1, 0, n-1,
      l, r, value);}
 void range_update(int l, int r, T value){range_update(1, 0,
      n-1, l, r, value);}

 LazySegtree(const vector<T>& input) {
  n = input.size();
  int sz = n<<2; // 4n
  tree.resize(sz);
  addTree.resize(sz, add_identity);
  setTree.resize(sz, set_identity);
  buildTree(input, 1, 0, n-1);
 }

 T add_identity = 0;
 T set_identity = 0;
 T identity = 0;
 T merge(const T& a, const T& b) { return a + b; }
};

/*
 LazySegtree<int> segTree(v);

 segTree.query(left-1, right-1);
 segTree.range_set(left-1, right-1, value);
 segTree.range_update(left-1, right-1, value);
*/
```

2.9 5c Merge Sort Tree [M]

```
template <class T>
struct SegmentTree{
private:
    int n;
    vector<vector<T>> tree;

    // Build Tree
    void buildTree(const vector<T>& v, int node, int b, int e
        ){
        if(b==e){tree[node] = {v[b]};return;}
        int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
        buildTree(v, ln, b, mid);
        buildTree(v, rn, mid+1, e);
        tree[node] = merge(tree[ln],tree[rn]);
    }

    // Merge Nodes (just sort two nodes or vectors)
    vector<int> merge(vector<int> &a, vector<int> &b) {
        vector<int> c;
        int i = 0, j = 0;
        while (i < a.size() and j < b.size()) {
            if (a[i] < b[j]) c.push_back(a[i++]);
            else c.push_back(b[j++]);
        }
        while (i < a.size()) c.push_back(a[i++]);
        while (j < b.size()) c.push_back(b[j++]);
        return c;
    }

    // do binary search on the sorted node array(ofc if in
         range)
    int get(vector<int> &v, int k){
        auto it = upper_bound(v.begin(), v.end(), k) - v.
            begin();
        // return it; //number of elements strictly less than
             k in the range
        return v.size() - it; //number of elements strictly
            greater than k in the range
        // return v.size() - it - 1; //number of elements
            strictly greater than or equal to k in the range
    }

    int query(int node, int tL, int tR, int qL, int qR, int k
         ) {
        if (tL >= qL && tR <= qR) {
            return get(tree[node], k);
        }
        if (tR < qL || tL > qR) {
            return 0;
        }
        int mid = (tL + tR) / 2;
        int QL = query(2 * node, tL, mid, qL, qR, k);
        int QR = query(2 * node + 1, mid + 1, tR, qL, qR, k);
        return QL + QR;
    }

public:
    int query(int l, int r, int k){return query(1, 0, n-1, l,
         r, k);}

    SegmentTree(const vector<T>& input) {
        n = input.size();
        int sz = n<<2; // 4n
        // tree.assign(vector<T>());
        tree.resize(sz);
        buildTree(input, 1, 0, n-1);
    }
};
```

```
/*
    vector<int> v(n); cin >> v;

    SegmentTree<int> segTree(v); // All 0 based index

    segTree.query(left - 1, right - 1, k);
*/
```

2.10 5d Merge Sort Tree (w point update) [M]

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template <class T>
class _multiset{
    using MS = tree<T, null_type, less_equal<T>,
    rb_tree_tag, tree_order_statistics_node_update>;
    MS s;
public:
    _multiset(){s.clear();}
    void erase(T xx){s.erase(s.upper_bound(xx));}
    typename MS::iterator lower_bound(T xx){return s.
        upper_bound(xx);}
    typename MS::iterator upper_bound(T xx){return s.
        lower_bound(xx);}
    // same
    size_t size(){return s.size();}
    void insert(T xx){s.insert(xx);}
    T find_by_order(int xx){return s.find_by_order(xx);}
    int order_of_key(T xx){return s.order_of_key(xx);}
    void erase(typename MS::iterator xx){s.erase(xx);}
};

using T = long long;
int N;
vector<T> vec;
vector<_multiset<ll>> segtree;

void buildTree(int node, int b, int e){
    for (int i = b; i <= e; i++) {
        segtree[node].insert(vec[i]);
    }
    if(b==e)return;
    int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
    buildTree(ln, b, mid);
    buildTree(rn, mid+1, e);
}

T query(int node, int b, int e, int l, int r, T val){
    if(l > e or r < b) return 0;
    if(l<=b and r>=e) return segtree[node].order_of_key(val);
    int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
    T c1 = query(ln, b, mid, l, r, val);
    T c2 = query(rn, mid+1, e, l, r, val);
    return c1 + c2;
}

void setValue(int node, int b, int e, int ind, T val){
    segtree[node].erase(vec[ind]);
    segtree[node].insert(val);

    if(b==e)return;

    int mid = (b+e)>>1, ln = node<<1, rn = ln+1;
    if (ind <= mid) setValue(ln, b, mid, ind, val);
    else setValue(rn, mid+1, e, ind, val);
}

void buildTree(vector<T>& input) {
    N = input.size();vec = input;
    int sz = N<<2; // 4n
    segtree.resize(sz);
    buildTree(1, 0, N-1);
}
T query(int l, int r, T val){return query(1, 0, N-1, l, r,
    val);}
void setValue(int ind, T val){
    setValue(1, 0, N-1, ind, val);
    vec[ind] = val;
}

/*
    vector<int> v(n); input(v);
    buildTree(v); // All 0 based index

    query(left-1, right-1, value);
    set(index-1, value);
*/
```

2.11 6 Sparse Table [M]

```
template<class T>
struct SparseTable {
    vector<vector<T>> jmp;

    SparseTable(const vector<T>& V) {
        int n = V.size();
        int log = 32 - __builtin_clz(n); // Maximum depth
        jmp.assign(log, V);

        for (int k = 1, pw = 1; pw * 2 <= n; ++k, pw *= 2) {
            for (int i = 0; i + pw * 2 <= n; ++i) {
                jmp[k][i] = min(jmp[k - 1][i], jmp[k - 1][i +
                    pw]);
            }
        }
    }

    T query(int a, int b) {
        assert(a < b);
        int dep = 31 - __builtin_clz(b - a); // log2(b - a)
        return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
    }
};
// SparseTable<int> table(v);
// table.query(a, b); // [a, b) // index 0 based
```

2.12 7a Sqrt Decomposition [M]

```
// Sqrt Decomposition
struct SqrtDecom {
int block_size;
vector<int> nums;
vector<long long> blocks;
SqrtDecom(int sqrtn, vector<int> &arr) : block_size(sqrtn),
    blocks(sqrtn, 0) {
    nums = arr;
    for (int i = 0; i < nums.size(); i++) { blocks[i /
        block_size] += nums[i]; }
}

/** O(1) update to set nums[i] to v */
void update(int i, int v) {
    blocks[i / block_size] -= nums[i];
    nums[i] = v;
    blocks[i / block_size] += nums[i];
}

/** O(sqrt(n)) query for sum of [0, r) */
long long query(int r) {
    long long res = 0;
    for (int i = 0; i < r / block_size; i++) { res += blocks[
        i]; }
```

```
    for (int i = (r / block_size) * block_size; i < r; i++) {
        res += nums[i]; }
    return res;
}

/** O(sqrt(n)) query for sum of [l, r) */
long long query(int l, int r) { return query(r) - query(l -
    1); }
};
// SqrtDecomp sq((int)ceil(sqrt(n)), v); // O(n)
// sq.query(l, r); // O( sqrt(n) )
// sq.update(i, v); // O(1)
```

2.13 7b Mo's Algorithm [M]

```
// pbds set // more like a indexed set
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>,
rb_tree_tag,tree_order_statistics_node_update> pbds;

void getMoAnswer(vector<int>& v, vector<array<int, 5>>&
    queries, vector<int>& ans) {
    pbds oset; // ordered set
    auto add = [&](int x) -> void { oset.insert(v[x]); };
    auto remove = [&](int x) -> void { oset.erase(v[x]); };
    auto get = [&](int k) -> int { return *oset.find_by_order
        (k-1); };

    sort(all(queries));
    int left = 0, right = -1;
    for (auto& [b, r, l, idx, k] : queries) {
        while(right < r) add(++right); while(right > r)
            remove(right--);
        while(left < l) remove(left++); while(left > l) add
            (--left);
        ans[idx] = get(k);
    }
}
// v = main array, // N = v.size()
queries.push_back({l/sqrtN, r, l, idx, k}); // for each
    query
// sort quiries according to -> starting block, and then r
    wise sort
// gives k'th smallest number's index in [l, r) range
```

2.14 8 Trie Normal [M]

```
// Trie
struct Node {
    Node *links[26];
    int cp = 0, cw = 0;

    bool containsRef(char c) { return links[c - 'a'] != NULL
        ;}
    void putRef(char c, Node *node) { links[c - 'a'] = node;
        }
    Node* getRef(char c) { return links[c - 'a']; }

    void incPrefix() { cp++; }
    void decPrefix() { cp--; }
    int countPrefixes() { return cp; }

    void incWord() { cw++; }
    void decWord() { cw--; }
    int countWords() { return cw; }

};

struct Trie {
    Node *root;
    Trie() { root = new Node(); }

    // O( len(word) )
    void insert(string& word) {
        Node *node = root;
        for (auto& c : word) {
            if (!node->containsRef(c)) {
                node->putRef(c, new Node());
            }
            node = node->getRef(c);
            node->incPrefix();
        }
        node->incWord();
    }

    // O( len(word) )
    void remove(string& word) {
        Node *node = root;
        for (auto& c : word) {
            if (!node->containsRef(c)) return;
            node = node->getRef(c);
            node->decPrefix();
        }
        node->decWord();
    }

    // O( len(word) )
    int countWordsEqualTo(string& word) {
        Node *node = root;
        for (auto& c : word) {
            if (!node->containsRef(c)) return 0;
            node = node->getRef(c);
        }
        return node->countWords();
    }

    // O( len(word) )
    int countWordsStartingWith(string& prefix) {
        Node *node = root;
        for (auto& c : prefix) {
            if (!node->containsRef(c)) return 0;
            node = node->getRef(c);
        }
        return node->countPrefixes();
    }
};
// Trie trie;
```

2.15 8b Trie Bitwise [M][SS]

```
const int N = 3e5 + 9;

struct Trie {
  static const int B = 31;
  struct node {
    node* nxt[2];
    int sz;
    node() {
      nxt[0] = nxt[1] = NULL;
      sz = 0;
    }
  }*root;
  Trie() {
    root = new node();
  }
  void insert(int val) {
    node* cur = root;
    cur -> sz++;
    for (int i = B - 1; i >= 0; i--) {
      int b = val >> i & 1;
      if (cur -> nxt[b] == NULL) cur -> nxt[b] = new node();
      cur = cur -> nxt[b];
      cur -> sz++;
    }
```

```
    }
    int query(int x, int k) { // number of values s.t. val ^ x
        < k
      node* cur = root;
      int ans = 0;
      for (int i = B - 1; i >= 0; i--) {
        if (cur == NULL) break;
        int b1 = x >> i & 1, b2 = k >> i & 1;
        if (b2 == 1) {
          if (cur -> nxt[b1]) ans += cur -> nxt[b1] -> sz;
          cur = cur -> nxt[!b1];
        } else cur = cur -> nxt[b1];
      }
      return ans;
    }
    int get_max(int x) { // returns maximum of val ^ x
      node* cur = root;
      int ans = 0;
      for (int i = B - 1; i >= 0; i--) {
        int k = x >> i & 1;
        if (cur -> nxt[!k]) cur = cur -> nxt[!k], ans <<= 1,
            ans++;
        else cur = cur -> nxt[k], ans <<= 1;
      }
      return ans;
    }
    int get_min(int x) { // returns minimum of val ^ x
      node* cur = root;
      int ans = 0;
      for (int i = B - 1; i >= 0; i--) {
        int k = x >> i & 1;
        if (cur -> nxt[k]) cur = cur -> nxt[k], ans <<= 1;
        else cur = cur -> nxt[!k], ans <<= 1, ans++;
      }
      return ans;
    }
    void del(node* cur) {
      for (int i = 0; i < 2; i++) if (cur -> nxt[i]) del(cur ->
          nxt[i]);
      delete(cur);
    }
} t;

// t.insert(cur);
// t.query(cur, k); count numbers which are (a[i] ^ x < k)
// t.get_max(int x); // gets max of val ^ x
// t.get_min(int x); // gets min of val ^ x
```

2.16 9 Wavelet [M][SS]

```
// Wavelet Tree
const int MAXN = (int)3e5 + 9;
const int MAXV = (int)1e9 + 9; // maximum value of any
     element in array
// array values can be negative too, use appropriate minimum
      and maximum value
struct wavelet_tree {
    int lo, hi;
    wavelet_tree *l, *r;
    int *b, *c, bsz, csz; // c holds the prefix sum of
         elements

    wavelet_tree() {
        lo = 1; hi = 1;
        bsz = csz = 0;
        l = r = NULL;
    }

    void init(int *from, int *to, int x, int y) {
        lo = x, hi = y;
        if (from >= to) return;
        int mid = (lo + hi) >> 1;
        auto f = [mid](int x) { return x <= mid; };
        b = (int *)malloc((to - from + 2) * sizeof(int));
        bsz = 0; b[bsz++] = 0;
        c = (int *)malloc((to - from + 2) * sizeof(int));
        csz = 0; c[csz++] = 0;
        for (auto it = from; it != to; it++) {
            b[bsz] = (b[bsz - 1] + f(*it)); bsz++;
            c[csz] = (c[csz - 1] + (*it)); csz++;
        }
        if (hi == lo) return;
        auto pivot = stable_partition(from, to, f);
        l = new wavelet_tree();
        l->init(from, pivot, lo, mid);
        r = new wavelet_tree();
        r->init(pivot, to, mid + 1, hi);
    }
    // kth smallest element in [l, r]
    int kth(int l, int r, int k) {
        if (l > r) return 0;
        if (lo == hi) return lo;
        int inLeft = b[r] - b[l - 1], lb = b[l - 1], rb = b[r
            ];
        if (k <= inLeft) return this->l->kth(lb + 1, rb, k);
        return this->r->kth(l - lb, r - rb, k - inLeft);
    }
    // count of numbers in [l, r] Less than or equal to k
    int LTE(int l, int r, int k) {
        if (l > r || 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);
    }
    // count of numbers in [l, r] equal to k
    int count(int l, int r, int k) {
        if (l > r || k < lo || k > hi) return 0;
        if (lo == hi) return r - l + 1;
        int lb = b[l - 1], rb = b[r];
        int mid = (lo + hi) >> 1;
        if (k <= mid) return this->l->count(lb + 1, rb, k);
        return this->r->count(l - lb, r - rb, k);
    }
    // sum of numbers in [l ,r] less than or equal to k
    int sum(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->sum(lb + 1, rb, k) + this->r->sum(l -
             lb, r - rb, k);
    }
    ~wavelet_tree() { delete l; delete r; }
};
int a[MAXN]; // declare
wavelet_tree t;

// 1 based -> index, l, r
// int n; cin >> n; // size of array
// for (int i=1; i<=n; i++)cin>>a[i]; // array input

// O (n log ( max_ele(array) )), array a changes after init
// t.init(a + 1, a + n + 1, -MAXV, MAXV);

//     [l, r] range, below O( max_ele(array)
// t.kth(l, r, k); // kth smallest element
// t.LTE(l, r, k); // count values <= k
// t.count(l, r, k); // count values == k
// t.sum(l, r, k); // sum of numbers <= k
```

2.17 Articulation Points in O(N + M) [NK]

```
int n; // number of nodes
```

```
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> tin, low;
int timer;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}
void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

2.18 BIT - Binary Indexed Tree [MB]

```
struct BIT
{
private:
 std::vector<long long> mArray;
public:
 BIT(int sz) // Max size of the array
 {
  mArray.resize(sz + 1, 0);
 }
 void build(const std::vector<long long> &list)
 {
  for (int i = 1; i <= list.size(); i++)
   mArray[i] = list[i];
  for (int ind = 1; ind <= mArray.size(); ind++)
  {
   int ind2 = ind + (ind & -ind);
   if (ind2 <= mArray.size())
    mArray[ind2] += mArray[ind];
  }
 }
 long long prefix_query(int ind)
 {
  int res = 0;
  for (; ind > 0; ind -= (ind & -ind))
   res += mArray[ind];
  return res;
 }
 long long range_query(int from, int to)
 {
  return prefix_query(to) - prefix_query(from - 1);
 }
 void add(int ind, long long add)
 {
  for (; ind < mArray.size(); ind += (ind & -ind))
   mArray[ind] += add;
 }
};
```

2.19 Bridges in O(N + M) [NK]

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> tin, low;
int timer;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}
void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

2.20 Bridges Online [NK]

```
vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
int bridges;
int lca_iteration;
vector<int> last_visit;
void init(int n) {
    par.resize(n);
    dsu_2ecc.resize(n);
    dsu_cc.resize(n);
    dsu_cc_size.resize(n);
    lca_iteration = 0;
    last_visit.assign(n, 0);
    for (int i=0; i<n; ++i) {
        dsu_2ecc[i] = i;
        dsu_cc[i] = i;
        dsu_cc_size[i] = 1;
        par[i] = -1;
    }
    bridges = 0;
}
int find_2ecc(int v) {
    if (v == -1)
        return -1;
    return dsu_2ecc[v] == v ? v : dsu_2ecc[v] = find_2ecc(
        dsu_2ecc[v]);
}
int find_cc(int v) {
    v = find_2ecc(v);
    return dsu_cc[v] == v ? v : dsu_cc[v] = find_cc(dsu_cc[v
        ]);
}
void make_root(int v) {
    v = find_2ecc(v);
    int root = v;
    int child = -1;
    while (v != -1) {
        int p = find_2ecc(par[v]);
        par[v] = child;
        dsu_cc[v] = root;
```

```
        child = v;
        v = p;
    }
    dsu_cc_size[root] = dsu_cc_size[child];
}
void merge_path (int a, int b) {
    ++lca_iteration;
    vector<int> path_a, path_b;
    int lca = -1;
    while (lca == -1) {
        if (a != -1) {
            a = find_2ecc(a);
            path_a.push_back(a);
            if (last_visit[a] == lca_iteration){
                lca = a;
                break;
                }
            last_visit[a] = lca_iteration;
            a = par[a];
        }
        if (b != -1) {
            b = find_2ecc(b);
            path_b.push_back(b);
            if (last_visit[b] == lca_iteration){
                lca = b;
                break;
                }
            last_visit[b] = lca_iteration;
            b = par[b];
        }

    }
    for (int v : path_a) {
        dsu_2ecc[v] = lca;
        if (v == lca)
            break;
        --bridges;
    }
    for (int v : path_b) {
        dsu_2ecc[v] = lca;
        if (v == lca)
            break;
        --bridges;
    }
}

void add_edge(int a, int b) {
    a = find_2ecc(a);
    b = find_2ecc(b);
    if (a == b)
        return;
    int ca = find_cc(a);
    int cb = find_cc(b);
    if (ca != cb) {
        ++bridges;
        if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
            swap(a, b);
            swap(ca, cb);
        }
        make_root(a);
        par[a] = dsu_cc[a] = b;
        dsu_cc_size[cb] += dsu_cc_size[a];
    } else {
        merge_path(a, b);
    }
}
```

2.21 Convex Hull Trick [AlphaQ]

```
typedef long long ll;
const ll IS_QUERY = -(1LL << 62);
struct line {
  ll m, b;
  mutable function <const line*()> succ;
  bool operator < (const line &rhs) const {
    if (rhs.b != IS_QUERY) return m < rhs.m;
    const line *s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s -> b < (s -> m - m) * x;
  }
};
struct HullDynamic : public multiset <line> {
  bool bad (iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y -> m == z -> m && y -> b <= z -> b;
    }
    auto x = prev(y);
    if (z == end()) return y -> m == x -> m && y -> b <= x ->
         b;
    return 1.0 * (x -> b - y -> b) * (z -> m - y -> m) >= 1.0
         * (y -> b - z -> b) * (y -> m - x -> m);
  }
  void insert_line (ll m, ll b) {
    auto y = insert({m, b});
    y -> succ = [=] {return next(y) == end() ? 0 : &*next(y)
        ;};
    if (bad(y)) {erase(y); return;}
    while (next(y) != end() && bad(next(y))) erase(next(y));
    while (y != begin() && bad(prev(y))) erase(prev(y));
  }
  ll eval (ll x) {
    auto l = *lower_bound((line) {x, IS_QUERY});
    return l.m * x + l.b;
  }
};
```

2.22 LCA - Lowest Common Ancestor [MB]

```
struct LCA {
private:
    int n, lg;
    std::vector<int> depth;
    std::vector<std::vector<int>> up;
    std::vector<std::vector<int>> g;
public:
    LCA() : n(0), lg(0) {}

    LCA(int _n) {
        this->n = _n;
        lg = (int)log2(n) + 2;
        depth.resize(n + 5, 0);
        up.resize(n + 5, std::vector<int>(lg, 0));
        g.resize(n + 1);
    }
    LCA(std::vector<std::vector<int>>& graph) : LCA((int)
         graph.size()) {
        for (int i = 0; i < (int)graph.size(); i++)
            g[i] = graph[i];
        dfs(1, 0);
    }
    void dfs(int curr, int p) {
        up[curr][0] = p;
        for (int next : g[curr]) {
            if (next == p)
                continue;
            depth[next] = depth[curr] + 1;
            up[next][0] = curr;
            for (int j = 1; j < lg; j++)
                up[next][j] = up[up[next][j - 1]][j - 1];
            dfs(next, curr);
        }
    }
    void clear_v(int a) {
```

```
        g[a].clear();
    }
    void clear(int n_ = -1) {
        if (n_ == -1)
            n_ = ((int)(g.size())) - 1;

        for (int i = 0; i <= n_; i++) {
            g[i].clear();
        }
    }
    void add(int a, int b) {
        g[a].push_back(b);
    }
    int par(int a) {
        return up[a][0];
    }
    int get_lca(int a, int b) {
        if (depth[a] < depth[b])
            std::swap(a, b);
        int k = depth[a] - depth[b];
        for (int j = lg - 1; j >= 0; j--) {
            if (k & (1 << j))
                a = up[a][j];
        }
        if (a == b)
            return a;
        for (int j = lg - 1; j >= 0; j--)
            if (up[a][j] != up[b][j]) {
                a = up[a][j];
                b = up[b][j];
            }
        return up[a][0];
    }
    int get_dist(int a, int b) {
        return depth[a] + depth[b] - 2 * depth[get_lca(a, b)
            ];
    }
};
```

2.23 LCA - Lowest Common Ancestor [SA]

```
vector<int> dist;
vector<vector<int>> up;
vector<vector<int>> adj;
int lg = -1;
void dfs(int u, int p = -1) {
    up[u][0] = p;
    for (auto v : adj[u]) {
        if (dist[v] != -1) continue;
        dist[v] = 1 + dist[u];
        dfs(v, u);
    }
}
void pre_process(int root, int n) {
    assert(lg != -1);
    dist[root] = 0;
    dfs(root);
    for (int i = 1; i < lg; ++i) {
        for (int j = 1; j <= n; ++j) {// 1-based graph
            int p = up[j][i - 1];
            if (p == -1) continue;
            up[j][i] = up[p][i - 1];
        }
    }
}
int get_lca(int u, int v) {
    if (dist[u] > dist[v])
        swap(u, v);
    int dif = dist[v] - dist[u];
    while (dif > 0) {
        int lg = __lg(dif);
        v = up[v][lg];
        dif -= (1 << lg);
    }
    if (u == v)
        return u;

    for (int i = lg - 1; i >= 0; --i) {
        if (up[u][i] == up[v][i]) continue;
        u = up[u][i];
        v = up[v][i];
    }
    return up[u][0];
}

int get_kth_ancestor(int v, int k) {
    while (k > 0) {
        int lg = __lg(k);
        v = up[v][lg];
        k -= (1 << lg);
    }
    return v;
}
```

2.24 SCC, Condens Graph [NK]

```
vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;
void dfs1(int v) {
    used[v] = true;
    for (auto u : adj[v])
        if (!used[u])
            dfs1(u);
    order.push_back(v);
}
void dfs2(int v) {
    used[v] = true;
    component.push_back(v);
    for (auto u : adj_rev[v])
        if (!used[u])
            dfs2(u);
}
int main() {
    int n;
    // ... read n ...
    for (;;) {
        int a, b;
        // ... read next directed edge (a,b) ...
        adj[a].push_back(b);
        adj_rev[b].push_back(a);
    }
    used.assign(n, false);
    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs1(i);
    used.assign(n, false);
    reverse(order.begin(), order.end());
    for (auto v : order)
        if (!used[v]) {
            dfs2(v);
            // ... processing next component ...
            component.clear();
        }
    vector<int> roots(n, 0);
    vector<int> root_nodes;
    vector<vector<int>> adj_scc(n);
    for (auto v : order)
        if (!used[v]) {
            dfs2(v);
            int root = component.front();
            for (auto u : component) roots[u] = root;
            root_nodes.push_back(root);
            component.clear();
        }
    for (int v = 0; v < n; v++)
        for (auto u : adj[v]) {
            int root_v = roots[v],
```

```
            root_u = roots[u];

        if (root_u != root_v)
            adj_scc[root_v].push_back(root_u);
    }
}
```

3 Equations

3.1 Combinatorics

3.1.1 General

1. $\sum_{0\le k\le n}\binom{n-k}{k} = Fib_{n+1}$

2. $\binom{n}{k} = \binom{n}{n-k}$

3. $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$

4. $k\binom{n}{k} = n\binom{n-1}{k-1}$

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

6. $\sum_{i=0}^{n}\binom{n}{i} = 2^n$

7. $\sum_{i\ge 0}\binom{n}{2i} = 2^{n-1}$

8. $\sum_{i\ge 0}\binom{n}{2i+1} = 2^{n-1}$

9. $\sum_{i=0}^{k}(-1)^i\binom{n}{i} = (-1)^k\binom{n-1}{k}$

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

11. $1\binom{n}{1} + 2\binom{n}{2} + 3\binom{n}{3} + \ldots + n\binom{n}{n} = n2^{n-1}$

12. $1^2\binom{n}{1} + 2^2\binom{n}{2} + 3^2\binom{n}{3} + \ldots + n^2\binom{n}{n} = (n+n^2)2^{n-2}$

13. **Vandermonde's Identify:** $\sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k} = \binom{m+n}{r}$

14. **Hockey-Stick Identify:** $n, r \in N, n > r, \sum_{i=r}^{n}\binom{i}{r} = \binom{n+1}{r+1}$

15. $\sum_{i=0}^{k}\binom{k}{i}^2 = \binom{2k}{k}$

16. $\sum_{k=0}^{n}\binom{n}{k}\binom{n}{n-k} = \binom{2n}{n}$

17. $\sum_{k=q}^{n}\binom{n}{k}\binom{k}{q} = 2^{n-q}\binom{n}{q}$

18. $\sum_{i=0}^{n}k^i\binom{n}{i} = (k+1)^n$

19. $\sum_{i=0}^{n}\binom{2n}{i} = 2^{2n-1} + \frac{1}{2}\binom{2n}{n}$

20. $\sum_{i=1}^{n}\binom{n}{i}\binom{n-1}{i-1} = \binom{2n-1}{n-1}$

21. $\sum_{i=0}^{n}\binom{2n}{i}^2 = \frac{1}{2}\left(\binom{4n}{2n} + \binom{2n}{n}^2\right)$

22. **Highest Power of 2 that divides ${}^{2n}C_n$:** Let x be the number of 1s in the binary representation. Then the number of odd terms will be 2^x.Let it form a sequence. The n-th value in the sequence (starting from $n = 0$) gives the highest power of 2 that divides ${}^{2n}C_n$.

23. **Pascal Triangle**

    (a) In a row p where p is a prime number, all the terms in that row except the 1s are multiples of p.

    (b) Parity: To count odd terms in row n, convert n to binary. Let x be the number of 1s in the binary representation. Then the number of odd terms will be 2^x.

    (c) Every entry in row $2^n - 1, n \ge 0$, is odd.

24. An integer $n \ge 2$ is prime if and only if all the intermediate binomial coefficients $\binom{n}{1}, \binom{n}{2}, \ldots, \binom{n}{n-1}$ are divisible by n.

25. **Kummer's Theorem:** For given integers $n \ge m \ge 0$ and a prime number p, the largest power of p dividing $\binom{n}{m}$ is equal to the number of carries when m is added to n-m in base p. For implementation take inspiration from lucas theorem.

26. Number of different binary sequences of length n such that no two 0's are adjacent=Fib_{n+1}

27. **Combination with repetition:** Let's say we choose k elements from an n-element set, the order doesn't matter and each element can be chosen more than once. In that case, the number of different combinations is: $\binom{n+k-1}{k}$

28. Number of ways to divide n persons in $\frac{n}{k}$ equal groups i.e. each having size k is

$$\frac{n!}{k!^{\frac{n}{k}}\left(\frac{n}{k}\right)!} = \prod_{n\geq k}^{n-=k}\binom{n-1}{k-1}$$

29. The number non-negative solution of the equation: $x_1 + x_2 + x_3 + \ldots + x_k = n$ is $\binom{n+k-1}{n}$

30. Number of ways to choose n ids from 1 to b such that every id has distance at least k = $\left(\frac{b-(n-1)(k-1)}{n}\right)$

31. $\sum_{i=1,3,5,\ldots}^{i\leq n}\binom{n}{i}a^{n-i}b^i = \frac{1}{2}((a+b)^n - (a-b)^n)$

32. $\sum_{i=0}^{n}\frac{\binom{k}{i}}{\binom{n}{i}} = \frac{\binom{n+1}{n-k+1}}{\binom{n}{k}}$

33. Derangement: a permutation of the elements of a set, such that no element appears in its original position. Let $d(n)$ be the number of derangements of the identity permutation fo size n.

$d(n) = (n-1)\cdot(d(n-1)+d(n-2))$ where $d(0) = 1, d(1) = 0$

34. **Involutions:** permutations such that p^2 = identity permutation. $a_0 = a_1 = 1$ and $a_n = a_{n-1}+(n-1)a_{n-2}$ for $n > 1$.

35. Let $T(n,k)$ be the number of permutations of size n for which all cycles have length $\leq k$.

$$T(n,k) = \begin{cases} n! & ; n \leq k \\ n\cdot T(n-1,k) - F(n-1,k)\cdot T(n-k-1,k) & ; n > k \end{cases}$$

Here $F(n,k) = n\cdot(n-1)\cdot\ldots\cdot(n-k+1)$

36. **Lucas Theorem**

(a) If p is prime, then $\binom{p^a}{k} \equiv 0(\mod p)$

(b) For non-negative integers m and n and a prime p, the following congruence relation holds: $\binom{m}{n} \equiv \prod_{i=0}^{k}\binom{m_i}{n_i} (mod\ p)$, where, $m = m_kp^k + m_{k-1}p^{k-1} + \ldots + m_1p + m_0$, and $n = n_kp^k + n_{k-1}p^{k-1} + \ldots + n_1p + n_0$ are the base p expansions of m and n respectively. This uses the convention that $\binom{m}{n} = 0$,when $m < n$.

37. $\sum_{i=0}^{n}\binom{n}{i}\cdot i^k = \sum_{i=0}^{n}\binom{n}{i}\cdot\sum_{j=0}^{k}\left\{{k \atop j}\right\}\cdot i^{\underline{j}} = \sum_{i=0}^{n}\binom{n}{i}\cdot\sum_{j=0}^{k}\left\{{k \atop j}\right\}\cdot j!\binom{n}{i} = \sum_{i=0}^{n}\frac{n!}{(n-i)!}\cdot\sum_{j=0}^{k}\left\{{k \atop j}\right\}\cdot\frac{1}{(i-j)!} = \sum_{i=0}^{n}\sum_{j=0}^{k}\frac{n!}{(n-i)!}\cdot\left\{{k \atop j}\right\}\cdot\frac{1}{(i-j)!} = n!\sum_{i=0}^{n}\sum_{j=0}^{k}\left\{{k \atop j}\right\}\cdot\frac{1}{(n-i)!}\cdot\frac{1}{(i-j)!} = n!\sum_{i=0}^{n}\sum_{j=0}^{k}\left\{{k \atop j}\right\}\cdot\binom{n-j}{n-i}\cdot\frac{1}{(n-j)!} = n!\sum_{j=0}^{k}\left\{{k \atop j}\right\}\cdot\frac{1}{(n-j)!}\sum_{i=0}^{n}\cdot\binom{n-j}{n-i} = \sum_{j=0}^{k}\left\{{k \atop j}\right\}\cdot n^{\underline{j}}\cdot 2^{n-j}$

Here $n^{\underline{j}} = P(n,j) = \frac{n!}{(n-j)!}$ and $\left\{{k \atop j}\right\}$ is stirling number of the second kind.

So, instead of $O(n)$, now you can calculate the original equation in $O(k^2)$ or even in $O(k\log^2 n)$ using NTT.

38. $\sum_{i=0}^{n-1}\binom{i}{j}x^i = x^j(1-x)^{-j-1}\left(1 - x^n\sum_{i=0}^{j}\binom{n}{i}x^{j-i}(1-x)^i\right)$

39. $x_0, x_1, x_2, x_3, \ldots, x_n$ $x_0+x_1, x_1+x_2, x_2+x_3, \ldots x_n \ldots$ If we continuously do this n times then the polynomial of the first column of the n-th row will be

$$p(n) = \sum_{k=0}^{n}\binom{n}{k}\cdot x(k)$$

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

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

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

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

3.1.2 Catalan Numbers

1. $C_n = \frac{1}{n+1}\binom{2n}{n}$

2. $C_0 = 1, C_1 = 1$ and $C_n = \sum_{k=0}^{n-1}C_kC_{n-1-k}$

3. Number of correct bracket sequence consisting of n opening and n closing brackets.

4. The number of ways to completely parenthesize n+1 factors.

5. The number of triangulations of a convex polygon with n+2 sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).

6. The number of ways to connect the $2n$ points on a circle to form n disjoint i.e. non-intersecting chords.

7. The number of monotonic lattice paths from point $(0,0)$ to point (n,n) in a square lattice of size $n\times n$, which do not pass above the main diagonal (i.e. connecting $(0,0)$ to (n,n)).

8. The number of rooted full binary trees with n+1 leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.

9. Number of permutations of $1,\ldots,n$ that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing sub-sequence. For $n = 3$, these permutations are 132, 213, 231, 312 and $321. For n = 4, they are 1432$, 2143, 2413, 2431, 3142, 3214, 3241, 3412, 3421, 4132, 4213, 4231, 4312 and 4321.

10. Balanced Parentheses count with prefix: The count of balanced parentheses sequences consisting of $n + k$ pairs of parentheses where the first k symbols are open brackets. Let the number be $C_n^{(k)}$, then

$$C_n^{(k)} = \frac{k+1}{n+k+1}\binom{2n+k}{n}$$

3.1.3 Narayana numbers

1. $N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$

2. The number of expressions containing n pairs of parentheses, which are correctly matched and which contain k distinct nestings. For instance, $N(4,2) = 6$ as with four pairs of parentheses six sequences can be created which each contain two times the sub-pattern '()'.

3.1.4 Stirling numbers of the first kind

1. The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).

2. $S(n,k)$ counts the number of permutations of n elements with k disjoint cycles.

3. $S(n,k) = (n-1)\cdot S(n-1,k) + S(n-1,k-1)$, $where,\ S(0,0) = 1, S(n,0) = S(0,n) = 0$

4. $\sum_{k=0}^{n} S(n,k) = n!$

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

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

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

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

3.1.5 Stirling numbers of the second kind

1. Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets.

2. $S(n,k) = k\cdot S(n-1,k) + S(n-1,k-1)$, $where\ S(0,0) = 1, S(n,0) = S(0,n) = 0$

3. $S(n,2) = 2^{n-1} - 1$

4. $S(n,k)\cdot k!$ = number of ways to color n nodes using colors from 1 to k such that each color is used at least once.

5. An r-associated Stirling number of the second kind is the number of ways to partition a set of n objects into k subsets, with each subset containing at least r elements. It is denoted by $S_r(n,k)$ and obeys the recurrence relation. $S_r(n+1,k) = kS_r(n,k) + \binom{n}{r-1}S_r(n-r+1,k-1)$

6. Denote the n objects to partition by the integers $1, 2, \ldots, n$. Define the reduced Stirling numbers of the second kind, denoted $S^d(n,k)$, to be the number of ways to partition the integers $1, 2, \ldots, n$ into k nonempty subsets such that all elements in each subset have pairwise distance at least d. That is, for any integers i and j in a given subset, it is required that $|i-j| \ge d$. It has been shown that these numbers satisfy, $S^d(n,k) = S(n-d+1, k-d+1), n \ge k \ge d$

3.1.6 Bell number

1. Counts the number of partitions of a set.

2. $B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}\cdot B_k$

3. $B_n = \sum_{k=0}^{n} S(n,k)$,where $S(n,k)$ is stirling number of second kind.

3.2 Math

3.2.1 General

1. $ab \mod ac = a(b \mod c)$

2. $\sum_{i=0}^{n} i\cdot i! = (n+1)! - 1.$

3. $a^k - b^k = (a-b)\cdot(a^{k-1}b^0 + a^{k-2}b^1 + \ldots + a^0b^{k-1})$

4. $\min(a+b,c) = a + \min(b, c-a)$

5. $|a-b| + |b-c| + |c-a| = 2(\max(a,b,c) - \min(a,b,c))$

6. $a\cdot b \le c \to a \le \left\lfloor \frac{c}{b} \right\rfloor$ is correct

7. $a\cdot b < c \to a < \left\lfloor \frac{c}{b} \right\rfloor$ is incorrect

8. $a\cdot b \ge c \to a \ge \left\lceil \frac{c}{b} \right\rceil$ is correct

9. $a\cdot b > c \to a > \left\lfloor \frac{c}{b} \right\rfloor$ is correct

10. For positive integer n, and arbitrary real numbers m, x,
$\left\lfloor \frac{\lfloor x/m \rfloor}{n} \right\rfloor = \left\lfloor \frac{x}{mn} \right\rfloor$
$\left\lceil \frac{\lceil x/m \rceil}{n} \right\rceil = \left\lceil \frac{x}{mn} \right\rceil$

11. Lagrange's identity:

$$\left(\sum_{k=1}^{n} a_k^2\right)\left(\sum_{k=1}^{n} b_k^2\right) - \left(\sum_{k=1}^{n} a_k b_k\right)^2 = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(a_ib_j - a_jb_i)^2$$
$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}(a_ib_j - a_jb_i)^2$$

12. $\sum_{i=1}^{n} ia^i = \frac{a(na^{n+1}-(n+1)a^n+1)}{(a-1)^2}$

13. Vieta's formulas: Any general polynomial of degree n

$$p(x) = a_nx^n + a_{n-1}x^{n-1} + ... + a_1x + a_0$$

(with the coefficients being real or complex numbers and $a_n \neq 0$) is known by the fundamental theorem of algebra to have n (not necessarily distinct) complex roots $r_1, r_2, \ldots, r_n$.

$$\begin{cases} r_1 + r_2 + ... + r_{n-1} + r_n = -\frac{a_{n-1}}{a_n} \\ (r_1r_2 + r_1r_3 + ... + r_1r_n) + (r_2r_3 + r_2r_4 + ... + r_2r_n) + ... + r_{n-1}r_n = \frac{a_{n-2}}{a_n} \\ \vdots \\ r_1r_2...r_n = (-1)^n\frac{a_0}{a_n}. \end{cases}$$

Vieta's formulas can equivalently be written as

$$\sum_{1\leq i_1<i_2<\ldots<i_k\leq n}\left(\prod_{j=1}^{k} r_{i_j}\right) = (-1)^k\frac{a_{n-k}}{a_n},$$

14. We are given n numbers $a_1, a_2, \ldots, a_n$ and our task is to find a value x that minimizes the sum,

$$|a_1 - x| + |a_2 - x| + ... + |a_n - x|$$

optimal x = median of the array. if n is even x = [left median,right median] i.e. every number in this range will work.

For minimizing

$$(a_1 - x)^2 + (a_2 - x)^2 + ... + (a_n - x)^2$$

optimal $x = \frac{(a_1 + a_2 + \ldots + a_n)}{n}$

15. Given an array a of n non-negative integers. The task is to find the sum of the product of elements of all the possible subsets. It is equal to the product of $(a_i + 1)$ for all a_i

16. Pentagonal number theorem: In mathematics, the pentagonal number theorem states that

$$\prod_{n=1}^{\infty}(1-x^n) = \prod_{k=-\infty}^{\infty}(-1)^k x^{\frac{k(3k-1)}{2}} = 1 + \prod_{k=1}^{\infty}(-1)^k\left(x^{\frac{k(3k+1)}{2}} + x^{\frac{k(3k-1)}{2}}\right).$$

In other words,

$$(1-x)(1-x^2)(1-x^3)\cdots = 1-x-x^2+x^5+x^7-x^{12}-x^{15}+x^{22}+x^{26}-\cdots.$$

The exponents $1, 2, 5, 7, 12, \cdots$ on the right hand side are given by the formula $g_k = \frac{k(3k-1)}{2}$ for $k = 1, -1, 2, -2, 3, \cdots$ and are called (generalized) pentagonal numbers.

It is useful to find the partition number in $O(n\sqrt{n})$

3.2.2 Fibonacci Number

1. $F_0 = 0, F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$

2. $F_n = \sum_{k=0}^{\lfloor\frac{n-1}{2}\rfloor}\binom{n-k-1}{k}$

3. $F_n = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n$

4. $\sum_{i=1}^{n} F_i = F_{n+2} - 1$

5. $\sum_{i=0}^{n-1} F_{2i+1} = F_{2n}$

6. $\sum_{i=1}^{n} F_{2i} = F_{2n+1} - 1$

7. $\sum_{i=1}^{n} F_i^2 = F_nF_{n+1}$

8. $F_mF_{n+1} - F_{m-1}F_n = (-1)^nF_{m-n}$ $F_{2n} = F_{n+1}^2 - F_{n-1}^2 = F_n(F_{n+1} + F_{n-1})$

9. $F_mF_n + F_{m-1}F_{n-1} = F_{m+n-1}$ $F_mF_{n+1} + F_{m-1}F_n = F_{m+n}$

10. A number is Fibonacci if and only if one or both of $(5 \cdot n^2 + 4)$ or $(5 \cdot n^2 - 4)$ is a perfect square

11. Every third number of the sequence is even and more generally, every k^{th} number of the sequence is a multiple of F_k

12. $gcd(F_m, F_n) = F_{gcd(m,n)}$

13. Any three consecutive Fibonacci numbers are pairwise coprime, which means that, for every n, $gcd(F_n, F_{n+1}) = gcd(F_n, F_{n+2}), gcd(F_{n+1}, F_{n+2}) = 1$

14. If the members of the Fibonacci sequence are taken $mod\ n$, the resulting sequence is periodic with period at most $6n$.

3.2.3 Pythagorean Triples

1. A Pythagorean triple consists of three positive integers a, b, and C, such that $a^2 + b^2 = c^2$. Such a triple is commonly written (a, b, c)

2. Euclid's formula is a fundamental formula for generating Pythagorean triples given an arbitrary pair of integers m and n with $m > n > 0$. The formula states that the integers

$$a = m^2 - n^2, b = 2mn, c = m^2 + n^2$$

form a Pythagorean triple. The triple generated by Euclid's formula is primitive if and only if m and n are coprime and not both odd. When both m and n are odd, then a, b, and c will be even, and the triple will not be primitive; however, dividing a, b, and c by 2 will yield a primitive triple when m and n are coprime and both odd.

3. The following will generate all Pythagorean triples uniquely:

$$a = k \cdot \left(m^2 - n^2\right), b = k \cdot (2mn), c = k \cdot \left(m^2 + n^2\right)$$

where m, n, and k are positive integers with $m > n$, and with m and n coprime and not both odd.

4. Theorem: The number of Pythagorean triples a,b,n with $maxa, b, n = n$ is given by

$$\frac{1}{2}\left(\prod_{p^\alpha || n} (2\alpha + 1) - 1\right)$$

where the product is over all prime divisors p of the form $4k + 1$. The notation $p^\alpha || n$ stands for the highest exponent α for which p^α divides n Example: For $n = 2 \cdot 3^2 \cdot 5^3 \cdot 7^4 \cdot 11^5 \cdot 13^6$, the number of Pythagorean triples with hypotenuse n is $\frac{1}{2}(7.13 - 1) = 45$. To obtain a formula for the number of Pythagorean triples with hypotenuse less than a specific positive integer N, we may add the numbers corresponding to each $n < N$ given by the Theorem. There is no simple way to compute this as a function of N.

3.2.4 Sum of Squares Function

1. The function is defined as $r_k(n) = \left|(a_1, a_2, \ldots, a_k) \in \mathbf{Z^k} : n = a_1^2 + a_2^2 + \ldots + a_k^2\right|$

2. The number of ways to write a natural number as sum of two squares is given by $r_2(n)$. It is given explicitly by $r_2(n) = 4(d_1(n) - d_3(n))$ where d1(n) is the number of divisors of n which are congruent with 1 modulo 4 and d3(n) is the number of divisors of n which are congruent with 3 modulo 4. The prime factorization $n = 2^g p_1^{f_1} p_2^{f_2} ... q_1^{h_1} q_2^{h_2} ...$, where p_i are the prime factors of the form $p_i \equiv 1 \pmod 4$, and q_i are the prime factors of the form $q_i \equiv 3 \pmod 4$ gives another formula $r_2(n) = 4(f_1 + 1)(f_2 + 1)\ldots$, if all exponents h_1,h_2,... are even. If one or more h_i are odd, then $r_2(n) = 0$.

3. The number of ways to represent n as the sum of four squares is eight times the sum of all its divisors which are not divisible by 4, i.e. $r_4(n) = 8 \sum d|n; 4dd$ $r8(n) = 16 \sum_{d|n} (-1)^{n+d} d^3$

3.3 Miscellaneous

1. $a + b = a \oplus b + 2(a\&b)$.

2. $a + b = a \mid b + a\&b$

3. $a \oplus b = a \mid b - a\&b$

4. k_{th} bit is set in x iff $x \mod 2^{k-1} \geq 2^k$. It comes handy when you need to look at the bits of the numbers which are pair sums or subset sums etc.

5. k_{th} bit is set in x iff $x \mod 2^{k-1} - x \mod 2^k \neq 0$ ($= 2^k$ to be exact). It comes handy when you need to look at the bits of the numbers which are pair sums or subset sums etc.

6. $n \mod 2^i = n\&(2^i - 1)$

7. $1 \oplus 2 \oplus 3 \oplus \cdots \oplus (4k - 1) = 0$ for any $k \geq 0$

8. Erdos Gallai Theorem: The degree sequence of an undirected graph is the non-increasing sequence of its vertex degrees A sequence of non-negative integers $d_1 \geq d_2 \geq \cdots \geq d_n$ can be represented as the degree sequence of finite simple graph on n vertices if and only if $d_1 + d_2 + \cdots + d_n$ is even and

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

holds for every k in $1 \leq k \leq n$.

3.4 Number Theory

3.4.1 General

1. for $i > j$, $\gcd(i, j) = \gcd(i - j, j) \leq (i - j)$

2. $\sum_{x=1}^{n} [d|x^k] = \left\lfloor \frac{n}{\prod_{i=0} p_i^{\left\lceil \frac{e_i}{k} \right\rceil}} \right\rfloor$,

where $d = \prod_{i=0} p_i^{e_i}$. Here, $[a|b]$ means if a divides b then it is 1, otherwise it is 0.

3. The number of lattice points on segment (x_1, y_1) to (x_2, y_2) is $\gcd(abs(x_1 - x_2), abs(y_1 - y_2)) + 1$

4. $(n - 1)! \mod n = n - 1$ if n is prime, 2 if $n = 4$, 0 otherwise.

5. A number has odd number of divisors if it is perfect square

6. The sum of all divisors of a natural number n is odd if and only if $n = 2^r \cdot k^2$ where r is non-negative and k is positive integer.

7. Let a and b be coprime positive integers, and find integers $a\prime$ and $b\prime$ such that $aa\prime \equiv 1 \mod b$ and $bb\prime \equiv 1 \mod a$. Then the number of representations of a positive integers (n) as a non negative linear combination of a and b is
$$\frac{n}{ab} - \left\{\frac{b\prime n}{a}\right\} - \left\{\frac{a\prime n}{b}\right\} + 1$$
Here, x denotes the fractional part of x.

8.
$$\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{c} d(i{\cdot}j{\cdot}k) = \sum_{\gcd(i,j)=\gcd(j,k)=\gcd(k,i)=1} \left\lfloor\frac{a}{i}\right\rfloor \left\lfloor\frac{b}{j}\right\rfloor \left\lfloor\frac{c}{k}\right\rfloor$$
Here, $d(x) =$ number of divisors of x.

9. Gauss's generalization of Wilson's theorem:, Gauss proved that,
$$\prod_{\substack{k=1\\ \gcd(k,m)=1}}^{m} k \equiv \begin{cases} -1 \pmod m & \text{if } m = 4,\ p^\alpha,\ 2p^\alpha \\ 1 \pmod m & \text{otherwise} \end{cases}$$
where p represents an odd prime and α a positive integer. The values of m for which the product is -1 are precisely the ones where there is a primitive root modulo m.

3.4.2 Divisor Function

1. $\sigma_x(n) = \sum_{d|n} d^x$

2. It is multiplicative i.e if $\gcd(a, b) = 1 \to \sigma_x(ab) = \sigma_x(a)\sigma_x(b)$.

3.
$$\sigma_x(n) = \prod_{i=1}^{\tau} \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$$

4. **Divisor Summatory Function**
   (a) Let $\sigma_0(k)$ be the number of divisors of k.
   (b) $D(x) = \sum_{n\le x} \sigma_0(n)$
   (c) $D(x) = \sum_{k=1}^{x} \lfloor\frac{x}{k}\rfloor = 2\sum_{k=1}^{u} \lfloor\frac{x}{k}\rfloor - u^2$, where $u = \sqrt{x}$
   (d) $D(n)$ =Number of increasing arithmetic progressions where $n + 1$ is the second or later term. (i.e. The last term, starting term can be any positive integer $\le n$. For example, $D(3) = 5$ and there are 5 such arithmetic progressions: $(1, 2, 3, 4); (2, 3, 4); (1, 4); (2, 4); (3, 4)$.

5. Let $\sigma_1(k)$ be the sum of divisors of k. Then, $\sum_{k=1}^{n} \sigma_1(k) = \sum_{k=1}^{n} k \left\lfloor\frac{n}{k}\right\rfloor$

6. $\prod_{d|n} d = n^{\frac{\sigma_0}{2}}$ if n is not a perfect square, and $= \sqrt{n} \cdot n^{\frac{\sigma_0 - 1}{2}}$ if n is a perfect square.

3.4.3 Euler's Totient function

1. The function is multiplicative. This means that if $\gcd(m, n) = 1$, $\phi(m \cdot n) = \phi(m) \cdot \phi(n)$.

2. $\phi(n) = n\prod_{p|n}(1 - \frac{1}{p})$

3. If p is prime and (k $\ge$ 1)$, then, \phi(p^k) = p^{k-1}(p - 1) = p^k(1 - \frac{1}{p})$

4. $J_k(n)$, the Jordan totient function, is the number of k-tuples of positive integers all less than or equal to n that form a coprime $(k + 1)$-tuple together with n. It is a generalization of Euler's totient, $\phi(n) = J_1(n)$. $J_k(n) = n^k \prod_{p|n}(1 - \frac{1}{p^k})$

5. $\sum_{d|n} J_k(d) = n^k$

6. $\sum_{d|n} \phi(d) = n$

7. $\phi(n) = \sum_{d|n} \mu(d) \cdot \frac{n}{d} = n\sum_{d|n} \frac{\mu(d)}{d}$

8. $\phi(n) = \sum_{d|n} d \cdot \mu(\frac{n}{d})$

9. $a|b \to \varphi(a)|\varphi(b)$

10. $n|\varphi(a^n - 1)$ for $a, n > 1$

11. $\varphi(mn) = \varphi(m)\varphi(n) \cdot \frac{d}{\varphi(d)}$ where $d = gcd(m, n)$ Note the special cases
$$\varphi(2m) = \begin{cases} 2\varphi(m) & ; if\ m\ is\ even \\ \varphi(m) & ; if\ m\ is\ odd \end{cases}$$
$$\varphi(n^m) = n^{m-1}\varphi(n)$$

12. $\varphi(lcm(m, n)) \cdot \varphi(gcd(m, n)) = \varphi(m) \cdot \varphi(n)$ Compare this to the formula $lcm(m, n) \cdot gcd(m, n) = m \cdot n$

13. $\varphi(n)$ is even for $n \ge 3$. Moreover, if if n has r distinct odd prime factors, $2^r|\varphi(n)$

14. $\sum_{d|n} \frac{\mu^2(d)}{\varphi(d)} = \frac{n}{\varphi(n)}$

15. $\sum_{1\le k\le n, \gcd(k,n)=1} k = \frac{1}{2}n\varphi(n)$ for $n > 1$

16. $\dfrac{\varphi(n)}{n} = \dfrac{\varphi(rad(n))}{rad(n)}$ where $rad(n) = \prod_{p|n, p\, prime} p$

17. $\phi(m) \geq \log_2 m$

18. $\phi(\phi(m)) \leq \dfrac{m}{2}$

19. When $x \geq \log_2 m$, then
$$n^x \mod m = n^{\phi(m)+x \mod \phi(m)} \mod m$$

20. $\sum_{1\leq k\leq n, gcd(k,n)=1} \gcd(k-1, n) = \varphi(n)d(n)$ where $d(n)$ is number of divisors. Same equation for $\gcd(a \cdot k - 1, n)$ where a and n are coprime.

21. For every n there is at least one other integer $m \neq n$ such that $\varphi(m) = \varphi(n)$.

22. $\sum_{i=1}^{n} \varphi(i) \cdot \lfloor \frac{n}{i} \rfloor = \dfrac{n*(n+1)}{2}$

23. $\sum_{i=1, i\%2\neq 0}^{n} \varphi(i) \cdot \lfloor \frac{n}{i} \rfloor = \sum_{k\geq 1} [\frac{n}{2^k}]^2$. Note that [] is used here to denote round operator not floor or ceil

24.
$$\sum_{i=1}^{n}\sum_{j=1}^{n} ij[\gcd(i,j)=1] = \sum_{i=1}^{n} \varphi(i) i^2$$

25. Average of coprimes of n which are less than n is $\dfrac{n}{2}$.

3.4.4 Mobius Function and Inversion

1. For any positive integer n, define $\mu(n)$ as the sum of the primitive n^{th} roots of unity. It has values in $-1, 0, 1$ depending on the factorization of n into prime factors:
   (a) $\mu(n) = 1$ if n is a square-free positive integer with an even number of prime factors.
   (b) $\mu(n) = -1$ if n is a square-free positive integer with an odd number of prime factors.
   (c) $\mu(n) = 0$ if n has a squared prime factor.

2. It is a multiplicative function.

3.
$$\sum_{d|n} \mu(d) = \begin{cases} 1 & ; n = 1 \\ 0 & ; n > 0 \end{cases}$$

4. $\sum_{n=1}^{N} \mu^2(n) = \sum_{n=1}^{\sqrt{N}} \mu(k) \cdot \left\lfloor \dfrac{N}{k^2} \right\rfloor$ This is also the number of square-free numbers $\leq n$

5. **Mobius inversion theorem:** The classic version states that if g and f are arithmetic functions satisfying $g(n) = \sum_{d|n} f(d)$ for every integer $n \geq 1$ then $g(n) = \sum_{d|n} \mu(d) g\left(\dfrac{n}{d}\right)$ for every integer $n \geq 1$

6. If $F(n) = \prod_{d|n} f(d)$, then $F(n) = \prod_{d|n} F\left(\dfrac{n}{d}\right)^{\mu(d)}$

7. $\sum_{d|n} \mu(d)\phi(d) = \prod_{j=1}^{K} (2 - P_j)$ where p_j is the primes factorization of d

8. If $F(n)$ is multiplicative, $F \not\equiv 0$, then $\sum_{d|n} \mu(d) f(d) = \prod_{i=1} (1 - f(P_i))\cdot$ where p_i are primes of n.

3.4.5 GCD and LCM

1. $\gcd(a, 0) = a$

2. $\gcd(a, b) = \gcd(b, a \mod b)$

3. Every common divisor of a and b is a divisor of $\gcd(a, b)$.

4. if m is any integer, then $\gcd(a + m\cdot b, b) = \gcd(a, b)$

5. The gcd is a multiplicative function in the following sense: if a_1 and a_2 are relatively prime, then $\gcd(a_1 \cdot a_2, b) = \gcd(a_1, b) \cdot \gcd(a_2, b)$.

6. $\gcd(a, b) \cdot \mathrm{lcm}(a, b) = |a \cdot b|$

7. $\gcd(a, \mathrm{lcm}(b, c)) = \mathrm{lcm}(\gcd(a, b), \gcd(a, c))$.

8. $\mathrm{lcm}(a, \gcd(b, c)) = \gcd(\mathrm{lcm}(a, b), \mathrm{lcm}(a, c))$.

9. For non-negative integers a and b, where a and b are not both zero, $\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$

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

11. $\sum_{i=1}^{n} [\gcd(i, n) = k] = \phi\Big(\dfrac{n}{k}\Big)$

12. $\sum_{k=1}^{n} \gcd(k, n) = \sum_{d|n} d \cdot \phi\Big(\dfrac{n}{d}\Big)$

13. $\sum_{k=1}^{n} x^{\gcd(k,n)} = \sum_{d|n} x^d \cdot \phi\Big(\dfrac{n}{d}\Big)$

14. $\sum_{k=1}^{n} \dfrac{1}{\gcd(k, n)} = \sum_{d|n} \dfrac{1}{d} \cdot \phi\Big(\dfrac{n}{d}\Big) = \dfrac{1}{n} \sum_{d|n} d \cdot \phi(d)$

15. $\sum_{k=1}^{n} \dfrac{k}{\gcd(k, n)} = \dfrac{n}{2} \cdot \sum_{d|n} \dfrac{1}{d} \cdot \phi\Big(\dfrac{n}{d}\Big) = \dfrac{n}{2} \cdot \dfrac{1}{n} \cdot \sum_{d|n} d \cdot \phi(d)$

16. $\sum_{k=1}^{n} \dfrac{n}{\gcd(k, n)} = 2 * \sum_{k=1}^{n} \dfrac{k}{\gcd(k, n)} - 1$, for $n > 1$

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

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

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

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

21. $\gcd(\text{lcm}(a,b),\text{lcm}(b,c),\text{lcm}(a,c)) = \text{lcm}(\gcd(a,b),\gcd(b,c),\gcd(a,c))$

22. $\gcd(A_L, A_{L+1},\ldots,A_R) = \gcd(A_L, A_{L+1} - A_L,\ldots,A_R - A_{R-1})$.'

23. Given n, If $SUM = LCM(1,n) + LCM(2,n) + \ldots + LCM(n,n)$ then SUM $= \frac{n}{2}(\sum_{d|n}(\phi(d)\times d) + 1$

3.4.6 Legendre Symbol

1. Let p be an odd prime number. An integer a is a quadratic residue modulo p if it is congruent to a perfect square modulo p and is a quadratic nonresidue modulo p otherwise. The Legendre symbol is a function of a and p defined as

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a quadatric residue modulo } p \text{ and } a \not\equiv 0 \pmod{p}, \\ -1 & \text{if } a \text{ is a non-quadaratic residue modulo } p, \\ 0 & \text{if } a \equiv 0 \pmod{p} \end{cases}$$

2. Legenres's original definition was by means of explicit formula $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}$ *and* $\left(\frac{a}{p}\right) \in -1, 0, 1$.

3. The Legendre symbol is periodic in its first (or top) argument: if $a \equiv b \pmod{p}$, then $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$.

4. The Legendre symbol is a completely multiplicative function of its top argument: $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$

5. The Fibonacci numbers $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ are defined by the recurrence $F_1 = F_2 = 1, F_{n+1} = F_n + F_{n-1}$. If p is a prime number then $F_{p-\left(\frac{p}{5}\right)} \equiv 0 \pmod{p}$, $F_p \equiv \left(\frac{p}{5}\right) \pmod{p}$.

   For example, $\left(\frac{2}{5}\right) = -1, \quad F_3 = 2, \quad F_2 = 1,$

   $\left(\frac{3}{5}\right) = -1, \quad F_4 = 3, \quad F_3 = 2,$

   $\left(\frac{5}{5}\right) = 0, \quad F_5 = 5,$

   $\left(\frac{7}{5}\right) = -1, \quad F_8 = 21, \quad F_7 = 13,$

   $\left(\frac{11}{5}\right) = 1, \quad F_{10} = 55, \quad F_{11} = 89,$

6. Continuing from previous point, $\left(\frac{p}{5}\right) =$ infinite concatenation of the sequence $(1, -1, -1, 1, 0)$ from $p \geq 1$.

7. If $n = k^2$ is perfect square then $\left(\frac{n}{p}\right) = 1$ for every odd prime except $\left(\frac{n}{k}\right) = 0$ if k is an odd prime.

4 Graph

4.1 Edge Remove CC [MB]

```
class DSU {
    std::vector<int> p, csz;
public:
    DSU() {}
    DSU(int dsz) // Max size
    {
        // Default empty
        p.resize(dsz + 5, 0), csz.resize(dsz + 5, 0);
        init(dsz);
    }
    void init(int n) {
        // n = size
        for (int i = 0; i <= n; i++) {
            p[i] = i, csz[i] = 1;
        }
    }
    // Return parent Recursively
    int get(int x) {
        if (p[x] != x)
            p[x] = get(p[x]);
        return p[x];
    }
    // Return Size
    int getSize(int x) { return csz[get(x)]; }
    // Return if Union created Successfully or false if they
        are already in Union
    bool merge(int x, int y) {
        x = get(x), y = get(y);
        if (x == y)
            return false;
        if (csz[x] > csz[y])
            std::swap(x, y);
        p[x] = y;
        csz[y] += csz[x];
        return true;
    }
};
int main() {
    int n, m;
    cin >> n >> m;
    auto g = vec(n + 1, set<int>());
    auto dsu = DSU(n + 1);
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].insert(v);
        g[v].insert(u);
    }
    set<int> elligible;
    for (int i = 1; i <= n; i++) {
        elligible.insert(i);
    }
    int i = 1;
    int cnt = 0;
    while (sz(elligible)) {
        cnt++;
        queue<int> q;
        q.push(*elligible.begin());
        elligible.erase(elligible.begin());
        while (sz(q)) {
            int fr = q.front();
            q.pop();
            auto v = elligible.begin();
```

```
            while (v != elligible.end()) {
                if (g[fr].find(*v) == g[fr].end()) {
                    q.push(*v);
                    v = elligible.erase(v);
                } else
                    v++;
            }
        }
    }
    cout << cnt - 1 << endl;
    return 0;
}
```

4.2 Kruskal's [NK]

```
struct Edge {
    using weight_type = long long;
    static const weight_type bad_w; // Indicates non-existent
         edge

    int u = -1;          // Edge source (vertex id)
    int v = -1;          // Edge destination (vertex id)
    weight_type w = bad_w; // Edge weight

#define DEF_EDGE_OP(op)                                    \
    friend bool operator op(const Edge& lhs, const Edge& rhs)
         { \
        return make_pair(lhs.w, make_pair(lhs.u, lhs.v)) op \
            make_pair(rhs.w, make_pair(rhs.u, rhs.v));     \
    }

    DEF_EDGE_OP(==)
    DEF_EDGE_OP(!=)
    DEF_EDGE_OP(<)
    DEF_EDGE_OP(<=)
    DEF_EDGE_OP(>)
    DEF_EDGE_OP(>=)
};

constexpr Edge::weight_type Edge::bad_w = numeric_limits<
     Edge::weight_type>::max();

template <class EdgeCompare = less<Edge>>
constexpr vector<Edge> kruskal(const int n, vector<Edge>
     edges, EdgeCompare compare = EdgeCompare()) {
    // define dsu part and initlaize forests

    vector<int> parent(n);
    iota(parent.begin(), parent.end(), 0);
    vector<int> size(n, 1);
    auto root = [&](int x) {
        int r = x;
        while (parent[r] != r) {
            r = parent[r];
        }
        while (x != r) {
            int tmp_id = parent[x];
            parent[x] = r;
            x = tmp_id;
        }
        return r;
    };
    auto connect = [&](int u, int v) {
        u = root(u);
        v = root(v);
        if (size[u] > size[v]) {
            swap(u, v);
        }
        parent[v] = u;
        size[u] += size[v];
        size[v] = 0;
    };

    // connect components (trees) with edges in order from
         the sorted list

    sort(edges.begin(), edges.end(), compare);
    vector<Edge> edges_mst;
    int remaining = n - 1;
    for (const Edge& e : edges) {
        if (!remaining) break;
        const int u = root(e.u);
        const int v = root(e.v);
        if (u == v) continue;
        --remaining;
        edges_mst.push_back(e);
        connect(u, v);
    }

    return edges_mst;
}
```

4.3 Re-rooting a Tree [MB]

```
typedef long long ll;
const int N = 2e5 + 5;
vector<int> g[N];
ll sz[N], dist[N], sum[N];

void dfs(int s, int p) {
    sz[s] = 1;
    dist[s] = 0;
    for (int nxt : g[s]) {
        if (nxt == p)
            continue;
        dfs(nxt, s);
        sz[s] += sz[nxt];
        dist[s] += (dist[nxt] + sz[nxt]);
    }
}

void dfs1(int s, int p) {
    if (p != 0) {
        ll my_size = sz[s];
        ll my_contrib = (dist[s] + sz[s]);

        sum[s] = sum[p] - my_contrib + sz[1] - sz[s] + dist[s
             ];
    }
    for (int nxt : g[s]) {
        if (nxt == p)
            continue;
        dfs1(nxt, s);
    }
}

// problem link: https://cses.fi/problemset/task/1133

int main() {
    int n;
    cin >> n;

    for (int i = 1, u, v; i < n; i++)
        cin >> u >> v, g[u].push_back(v), g[v].push_back(u);

    dfs(1, 0);

    sum[1] = dist[1];

    dfs1(1, 0);

    for (int i = 1; i <= n; i++)
        cout << sum[i] << " ";
    cout << endl;

    return 0;
}
```

5 Math, Number Theory, Geometry

5.1 Angle Orientation (Turn) [NK]

```
int orientation(const Point& p, const Point& q, const Point&
     r) {
    /// ||cross(PQ, QR)|| > 0 => left turn (counter-clockwise
        ) => 1
    /// ||cross(PQ, QR)|| < 0 => right turn (clockwise)  =>
        -1
    /// ||cross(PQ, QR)|| = 0 => straight line (collinear) =>
        0

    /// PQ = (Qx - Px, Qy - Py)
    /// QR = (Rx - Qx, Ry - Qy)
    /// cross(PQ, QR) = (Qx - Px) * (Ry - Qy) - (Qy - Py) * (
        Rx - Qx)

    auto v = (q.x - p.x) * (r.y - q.y) - (q.y - p.y) * (r.x -
         q.x);
    return (v > 0) - (v < 0);
}
```

5.2 BinPow - Modular Binary Exponentiation [NK]

```
template <class B, class E, class M>
constexpr B power(B base, E expo, M mod = 0) {
    assert(expo >= 0);
    if (mod == 1) return 0;
    if (base == 0 || base == 1) return base;
    B res = 1;
    if (!mod) {
        while (expo) {
            if (expo & 1) res *= base;
            base *= base;
            expo >>= 1;
        }
    } else {
        assert(mod > 0);
        base %= mod;
        if (base <= 1) return base;
        while (expo) {
            if (expo & 1) res = (res * base) % mod;
            base = (base * base) % mod;
            expo >>= 1;
        }
    }
    return res;
}
```

5.3 Cirle-line Intersection [CPA]

```
// assume the cirlce is centered at the origin
vector<pair<double, double>> circle_line_intersect(double r,
     double a, double b, double c) {
    double x0 = -a * c / (a * a + b * b), y0 = -b * c / (a *
         a + b * b);
    if (c * c > r * r * (a * a + b * b) + EPS) {
        return {};
    } else if (abs(c * c - r * r * (a * a + b * b)) < EPS) {
        return {make_pair(x0, y0)};
    } else {
        double d = r * r - c * c / (a * a + b * b);
        double mult = sqrt(d / (a * a + b * b));
        double ax, ay, bx, by;
        ax = x0 + b * mult;
        bx = x0 - b * mult;
        ay = y0 - a * mult;
        by = y0 + a * mult;
        return {make_pair(ax, ay), make_pair(bx, by)};
    }
}
```

5.4 Combinatrics [MB]

```
struct Combinatrics {
    vector<ll> fact, fact_inv, inv;
    ll mod, nl;

    Combinatrics() {}

    Combinatrics(ll n, ll _mod) {
        this->nl = n;
        this->mod = _mod;
        fact.resize(n + 1, 1), fact_inv.resize(n + 1, 1), inv
            .resize(n + 1, 1);
        init();
    }

    void init() {
        fact[0] = 1;
        for (int i = 1; i <= nl; i++) {
            fact[i] = (fact[i - 1] * i) % mod;
        }

        inv[0] = inv[1] = 1;
        for (int i = 2; i <= nl; i++)
            inv[i] = inv[mod % i] * (mod - mod / i) % mod;

        fact_inv[0] = fact_inv[1] = 1;

        for (int i = 2; i <= nl; i++)
            fact_inv[i] = (inv[i] * fact_inv[i - 1]) % mod;
    }

    ll ncr(ll n, ll r) {
        if (n < r) {
            return 0;
        }

        if (n > nl)
            return ncr(n, r, mod);
        return (((fact[n] * 1LL * fact_inv[r]) % mod) * 1LL *
             fact_inv[n - r]) % mod;
    }

    ll npr(ll n, ll r) {
        if (n < r) {
            return 0;
        }

        if (n > nl)
            return npr(n, r, mod);
        return (fact[n] * 1LL * fact_inv[n - r]) % mod;
    }

    ll big_mod(ll a, ll p, ll m = -1) {
        m = (m == -1 ? mod : m);
        ll res = 1 % m, x = a % m;
        while (p > 0)
            res = ((p & 1) ? ((res * x) % m) : res), x = ((x
                 * x) % m), p >>= 1;
        return res;
    }

    ll mod_inv(ll a, ll p) {
        return big_mod(a, p - 2, p);
    }

    ll ncr(ll n, ll r, ll p) {
        if (n < r)
```

```
        return 0;
    if (r == 0)
        return 1;
    return (((fact[n] * mod_inv(fact[r], p)) % p) *
        mod_inv(fact[n - r], p)) % p;
  }

  ll npr(ll n, ll r, ll p) {
    if (n < r)
        return 0;
    if (r == 0)
        return 1;
    return (fact[n] * mod_inv(fact[n - r], p)) % p;
  }
};
const int N = 1e6, MOD = 998244353;
Combinatrics comb(N, MOD);
```

5.5 Graham's Scan for Convex Hull [CPA]

```
bool cw(Point2D a, Point2D b, Point2D c, bool
    include_collinear) {
  int o = orientation(a, b, c);
  return o < 0 || (include_collinear && o == 0);
}
bool collinear(Point2D a, Point2D b, Point2D c) { return
    orientation(a, b, c) == 0; }

void convex_hull(vector<Point2D>& a, bool include_collinear
    = false) {
  Point2D p0 = *min_element(a.begin(), a.end(), [](Point2D
      a, Point2D b) {
    return make_pair(a.y, a.x) < make_pair(b.y, b.x);
  });
  sort(a.begin(), a.end(), [&p0](const Point2D& a, const
      Point2D& b) {
    int o = orientation(p0, a, b);
    if (o == 0)
        return (p0.x - a.x) * (p0.x - a.x) + (p0.y - a.y)
            * (p0.y - a.y) < (p0.x - b.x) * (p0.x - b.x)
            + (p0.y - b.y) * (p0.y - b.y);
    return o < 0;
  });
  if (include_collinear) {
    int i = (int)a.size() - 1;
    while (i >= 0 && collinear(p0, a[i], a.back())) i--;
    reverse(a.begin() + i + 1, a.end());
  }
  vector<Point2D> st;
  for (int i = 0; i < (int)a.size(); i++) {
    while (st.size() > 1 && !cw(st[st.size() - 2], st.
        back(), a[i], include_collinear))
        st.pop_back();
    st.push_back(a[i]);
  }

  a = st;
}
```

5.6 Mathematical Progression [SA]

```
int arithmetic_nth_term(int a, int n, int d) {
  return a + (n - 1) * d;
}
int arithmetic_sum(int a, int n, int d) {
  return n * (2 * a + (n - 1) * d) / 2;
}
int geometric_nth_term(int a, int n, int r) {
  return a * pow(r, n - 1);
}
int geometric_sum(int a, int n, int r) {
  if (r == 1) return n * a;
  if (r < 1) return a * (1 - pow(r, n)) / (1 - r);
  else return a * (pow(r, n) - 1) / (r - 1);
}
int infinite_geometric_sum(int a, int r) {
  assert(r < 1);
  return a / (1 - r);
}
```

5.7 MatrixExponentiation

```
struct Matrix : vector<vector<ll>>
{
 Matrix(size_t n) : std::vector<std::vector<ll>>(n, std::
     vector<ll>(n, 0)) {}
 Matrix(std::vector<std::vector<ll>> &v) : std::vector<std::
     vector<ll>>(v) {}

 Matrix operator*(const Matrix &other)
 {
  size_t n = size();
  Matrix product(n);
  for (size_t i = 0; i < n; i++)
  {
   for (size_t j = 0; j < n; j++)
   {
    for (size_t k = 0; k < n; k++)
    {
     product[i][k] += (*this)[i][j] * other[j][k];
     product[i][k] %= MOD;
    }
   }
  }
  return product;
 }
};
Matrix big_mod(Matrix a, long long n)
{
 Matrix res = Matrix(a.size());
 for (int i = 0; i < (int)(a.size()); i++)
  res[i][i] = 1;
 if (n <= 0) return res;
 while (n)
 {
  if (n % 2)
  {
   res = res * a;
  }
  n /= 2;
  a = a * a;
 }
 return res;
}
```

5.8 Miller Rabin - Primality Test [SK]

```
typedef long long ll;

ll mulmod(ll a, ll b, ll c) {
  ll x = 0, y = a % c;
  while (b) {
    if (b & 1) x = (x + y) % c;
    y = (y << 1) % c;
    b >>= 1;
  }
  return x % c;
}

ll fastPow(ll x, ll n, ll MOD) {
  ll ret = 1;
  while (n) {
    if (n & 1) ret = mulmod(ret, x, MOD);
    x = mulmod(x, x, MOD);
```

```
        n >>= 1;
    }
    return ret;
}

bool isPrime(ll n) {
    ll d = n - 1;
    int s = 0;
    while (d % 2 == 0) {
        s++;
        d >>= 1;
    }

    // It's guranteed that these values will work for any
        number smaller than 3e18 (3 and 18 zeros)
    int a[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    for (int i = 0; i < 9; i++) {
        bool comp = fastPow(a[i], d, n) != 1;
        if (comp)
            for (int j = 0; j < s; j++) {
                ll fp = fastPow(a[i], (1LL << (ll)j) * d, n);
                if (fp == n - 1) {
                    comp = false;
                    break;
                }
            }
        if (comp) return false;
    }
    return true;
}
```

5.9 Modular Inverse w Ext GCD [NK]

```
template <class Z>
constexpr Z extended_gcd(Z a, Z b, Z& x_ref, Z& y_ref) {
    x_ref = 1, y_ref = 0;
    Z x1 = 0, y1 = 1, tmp = 0, q = 0;
    while (b > 0) {
        q = a / b;
        tmp = a, a = b, b = tmp - (q * b);
        tmp = x_ref, x_ref = x1, x1 = tmp - (q * x1);
        tmp = y_ref, y_ref = y1, y1 = tmp - (q * y1);
    }
    return a;
}

template <class Z>
constexpr Z inverse(Z num, Z mod) {
    assert(mod > 1);
    if (!(0 <= num && num < mod)) {
        num %= mod;
        if (num < 0) num += mod;
    }
    Z res = 1, tmp = 0;
    assert(extended_gcd(num, mod, res, tmp) == 1);
    if (res < 0) res += mod;
    return res;
}
```

5.10 Point 2D, 3D Line [CPA]

```
using ftype = double; // or long long, int, etc.
struct Point2 {
    ftype x, y;
};
struct Point3 {
    ftype x, y, z;
};
// Define natural operator overloads for Point2 and Point3
// +, - with another point
// *, / with an ftype scalar
ftype dot(Point2 a, Point2 b) {
    return a.x * b.x + a.y * b.y;
}
ftype dot(Point3 a, Point3 b) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
ftype norm(Point2 a) {
    return dot(a, a);
}
double abs(Point2 a) {
    return sqrt(norm(a));
}
double proj(Point2 a, Point2 b) {
    return dot(a, b) / abs(b);
}
double angle(Point2 a, Point2 b) {
    return acos(dot(a, b) / abs(a) / abs(b));
}
Point3 cross(Point3 a, Point3 b) {
    return Point3(a.y * b.z - a.z * b.y,
                  a.z * b.x - a.x * b.z,
                  a.x * b.y - a.y * b.x);
}
ftype triple(Point3 a, Point3 b, Point3 c) {
    return dot(a, cross(b, c));
}
ftype cross(Point2 a, Point2 b) {
    return a.x * b.y - a.y * b.x;
}
Point2 lines_intersect(Point2 a1, Point2 d1, Point2 a2,
     Point2 d2) {
    return a1 + cross(a2 - a1, d2) / cross(d1, d2) * d1;
}
Point3 planes_intersect(Point3 a1, Point3 n1, Point3 a2,
     Point3 n2, Point3 a3, Point3 n3) {
    Point3 x(n1.x, n2.x, n3.x);
    Point3 y(n1.y, n2.y, n3.y);
    Point3 z(n1.z, n2.z, n3.z);
    Point3 d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
    return Point3(triple(d, y, z),
                  triple(x, d, z),
                  triple(x, y, d)) /
           triple(n1, n2, n3);
}
```

5.11 Pollard's Rho Algorithm [SK]

```
ll mul(ll x, ll y, ll mod) {
    ll res = 0;
    x %= mod;
    while (y) {
        if (y & 1) res = (res + x) % mod;
        y >>= 1;
        x = (x + x) % mod;
    }
    return res;
}
ll bigmod(ll a, ll m, ll mod) {
    a = a % mod;
    ll res = 1ll;
    while (m > 0) {
        if (m & 1) res = mul(res, a, mod);
        m >>= 1;
        a = mul(a, a, mod);
    }
    return res;
}
bool composite(ll n, ll a, ll s, ll d) {
    ll x = bigmod(a, d, n);
    if (x == 1 or x == n - 1) return false;
    for (int r = 1; r < s; r++) {
        x = mul(x, x, n);
        if (x == n - 1) return false;
    }
    return true;
}
```

```cpp
bool isprime(ll n) {
    if (n < 4) return n == 2 or n == 3;
    if (n % 2 == 0) return false;
    ll d = n - 1;
    ll s = 0;
    while (d % 2 == 0) {
        d /= 2;
        s++;
    }
    for (int i = 0; i < 10; i++) {
        ll a = 2 + rand() % (n - 3);
        if (composite(n, a, s, d)) return false;
    }
    return true;
}
// Polard rho
ll f(ll x, ll c, ll mod) {
    return (mul(x, x, mod) + c) % mod;
}
ll rho(ll n) {
    if (n % 2 == 0) {
        return 2;
    }
    ll x = rand() % n + 1;
    ll y = x;
    ll c = rand() % n + 1;
    ll g = 1;
    while (g == 1) {
        x = f(x, c, n);
        y = f(y, c, n);
        y = f(y, c, n);
        g = __gcd(abs(y - x), n);
    }
    return g;
}
void factorize(ll n, vector<ll>& factors) {
    if (n == 1) {
        return;
    } else if (isprime(n)) {
        factors.push_back(n);
        return;
    }
    ll cur = n;
    for (ll c = 1; cur == n; c++) {
        cur = rho(n);
    }
    factorize(cur, factors), factorize(n / cur, factors);
}
```

5.12 Sieve Phi (Segmented) [NK[

```cpp
vector<int64_t> phi_seg;

void seg_sieve_phi(const int64_t a, const int64_t b) {
    phi_seg.assign(b - a + 2, 0);
    vector<int64_t> factor(b - a + 2, 0);
    for (int64_t i = a; i <= b; i++) {
        auto m = i - a + 1;
        phi_seg[m] = i;
        factor[m] = i;
    }
    auto lim = sqrt(b) + 1;
    sieve(lim);
    for (auto p : primes) {
        int64_t a1 = p * ((a + p - 1) / p);
        for (int64_t j = a1; j <= b; j += p) {
            auto m = j - a + 1;
            while (factor[m] % p == 0) {
                factor[m] /= p;
            }
            phi_seg[m] -= (phi_seg[m] / p);
        }
    }
    for (int64_t i = a; i <= b; i++) {
        auto m = i - a + 1;
        if (factor[m] > 1) {
            phi_seg[m] -= (phi_seg[m] / factor[m]);
            factor[m] = 1;
        }
    }
}
```

5.13 Sieve Phi [MB]

```cpp
struct PrimePhiSieve {
private:
    ll n;
    vector<ll> primes, phi;
    vector<bool> is_prime;

public:
    PrimePhiSieve() {}

    PrimePhiSieve(ll n) {
        this->n = n, is_prime.resize(n + 5, true), phi.resize
            (n + 5, 1);
        phi_sieve();
    }

    void phi_sieve() {
        is_prime[0] = is_prime[1] = false;

        for (ll i = 1; i <= n; i++)
            phi[i] = i;

        for (ll i = 1; i <= n; i++)
            if (is_prime[i]) {
                primes.push_back(i);
                phi[i] *= (i - 1), phi[i] /= i;

                for (ll j = i + i; j <= n; j += i)
                    is_prime[j] = false, phi[j] /= i, phi[j]
                        *= (i - 1);
            }
    }

    ll get_divisors_count(int number, int divisor) {
        return phi[number / divisor];
    }

    ll get_phi(int n) {
        return phi[n];
    }
    // (n/p) * (p-1) => n- (n/p);
    void segmented_phi_sieve(ll l, ll r) {
        vector<ll> current_phi(r - l + 1);
        vector<ll> left_over_prime(r - l + 1);

        for (ll i = l; i <= r; i++)
            current_phi[i - l] = i, left_over_prime[i - l] =
                i;

        for (ll p : primes) {
            ll to = ((l + p - 1) / p) * p;

            if (to == p)
                to += p;

            for (ll i = to; i <= r; i += p) {
                while (left_over_prime[i - l] % p == 0)
                    left_over_prime[i - l] /= p;
                current_phi[i - l] -= current_phi[i - l] / p;
            }
        }

        for (ll i = l; i <= r; i++) {
            if (left_over_prime[i - l] > 1)
                current_phi[i - l] -= current_phi[i - l] /
                    left_over_prime[i - l];
```

```
        cout << current_phi[i - 1] << endl;
    }
}

ll phi_sqrt(ll n) {
    ll res = n;

    for (ll i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            res /= i;
            res *= (i - 1);

            while (n % i == 0)
                n /= i;
        }
    }

    if (n > 1)
        res /= n, res *= (n - 1);
    return res;
}
};
```

5.14 Sieve Phi [NK]

```
vector<int> phi;

void sieve_phi(int n) {
    phi.assign(n + 1, 0);
    iota(phi.begin(), phi.end(), 0);
    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i) {
                phi[j] -= (phi[j] / i);
            }
        }
    }
}
```

5.15 Sieve Primes (Segmented) [NK]

```
vector<bool> isprime_seg;
vector<int64_t> seg_primes;

void seg_sieve(const int64_t a, const int64_t b) {
    isprime_seg.assign(b - a + 1, true);
    int lim = sqrt(b) + 1;
    sieve(lim);
    for (auto p : primes) {
        auto a1 = p * max((int64_t)(p), ((a + p - 1) / p));
        for (auto j = a1; j <= b; j += p) {
            isprime_seg[j - a] = false;
        }
    }
    for (auto i = a; i <= b; i++) {
        if (isprime_seg[i - a]) {
            seg_primes.push_back(i);
        }
    }
}
```

5.16 Sieve Primes [MB]

```
struct PrimeSieve {
public:
    vector<int> primes;
    vector<bool> isprime;
    int n;

    PrimeSieve() {}

    PrimeSieve(int _n) {
        this->n = _n, isprime.resize(_n + 5, true), primes.
            clear();
        sieve();
    }

    void sieve() {
        isprime[0] = isprime[1] = false;

        primes.push_back(2);
        for (int i = 4; i <= n; i += 2)
            isprime[i] = false;

        for (int i = 3; 1LL * i * i <= n; i += 2)
            if (isprime[i])
                for (int j = i * i; j <= n; j += 2 * i)
                    isprime[j] = false;

        for (int i = 3; i <= n; i += 2)
            if (isprime[i])
                primes.push_back(i);
    }

    vector<pll> factorize(ll num) {
        vector<pll> a;
        for (int i = 0; i < (int)primes.size() && primes[i] *
            1LL * primes[i] <= num; i++)
            if (num % primes[i] == 0) {
                int cnt = 0;
                while (num % primes[i] == 0)
                    cnt++, num /= primes[i];
                a.push_back({primes[i], cnt});
            }

        if (num != 1)
            a.push_back({num, 1});
        return a;
    }

    vector<ll> segemented_sieve(ll l, ll r) {
        vector<ll> seg_primes;
        vector<bool> current_primes(r - l + 1, true);
        for (ll p : primes) {
            ll to = (l / p) * p;
            if (to < l)
                to += p;
            if (to == p)
                to += p;
            for (ll i = to; i <= r; i += p) {
                current_primes[i - l] = false;
            }
        }

        for (ll i = l; i <= r; i++) {
            if (i < 2)
                continue;
            if (current_primes[i - l]) {

                seg_primes.push_back(i);
            }
        }
        return seg_primes;
    }
};
```

6 String

6.1 Hashing [MB]

```
const int PRIMES[] = {2147462393, 2147462419, 2147462587,
    2147462633};

// ll base_pow,base_pow_1;
```

```cpp
ll base1 = 43, base2 = 47, mod1 = 1e9 + 7, mod2 = 1e9 + 9;

struct Hash {
public:
    vector<int> base_pow, f_hash, r_hash;
    ll base, mod;

    Hash() {}
    // Update it make it more dynamic like segTree class and
         DSU
    Hash(int mxSize, ll base, ll mod) // Max size
    {
        this->base = base;
        this->mod = mod;
        base_pow.resize(mxSize + 2, 1), f_hash.resize(mxSize
             + 2, 0), r_hash.resize(mxSize + 2, 0);

        for (int i = 1; i <= mxSize; i++) {
            base_pow[i] = base_pow[i - 1] * base % mod;
        }
    }

    void init(string s) {
        int n = s.size();

        for (int i = 1; i <= n; i++) {
            f_hash[i] = (f_hash[i - 1] * base + int(s[i - 1])
                ) % mod;
        }

        for (int i = n; i >= 1; i--) {
            r_hash[i] = (r_hash[i + 1] * base + int(s[i - 1])
                ) % mod;
        }
    }

    int forward_hash(int l, int r) {
        int h = f_hash[r + 1] - (1LL * base_pow[r - l + 1] *
             f_hash[l]) % mod;
        return h < 0 ? mod + h : h;
    }

    int reverse_hash(int l, int r) {
        int h = r_hash[l + 1] - (1LL * base_pow[r - l + 1] *
             r_hash[r + 2]) % mod;
        return h < 0 ? mod + h : h;
    }
};

class DHash {
public:
    Hash sh1, sh2;
    DHash() {}

    DHash(int mx_size) {
        sh1 = Hash(mx_size, base1, mod1);
        sh2 = Hash(mx_size, base2, mod2);
    }

    void init(string s) {
        sh1.init(s);
        sh2.init(s);
    }

    ll forward_hash(int l, int r) {
        return (ll(sh1.forward_hash(l, r)) << 32) | (sh2.
             forward_hash(l, r));
    }

    ll reverse_hash(int l, int r) {
        return ((ll(sh1.reverse_hash(l, r)) << 32) | (sh2.
             reverse_hash(l, r)));
    }
};
```

6.2 String Hashing With Point Updates [SA]

```cpp
struct Node {
    int64_t fwd, rev;
    int len;
    Node(int64_t f, int64_t r, int l) {
        fwd = f, rev = r, len = l;
    }
    Node() {
        fwd = rev = len = 0;
    }
};

const int BASE = 47, MX_N = 1E5 + 5, M = 1E9 + 7;
string a;
Node st[4 * MX_N];
int64_t expo[MX_N];// TODO: compute this beforehand

void build(int node, int tL, int tR) {
    if (tL == tR) {
        st[node] = Node(a[tL], a[tL], 1);
        return;
    }
    int mid = (tL + tR) / 2;
    int left = 2 * node, right = 2 * node + 1;
    build(left, tL, mid);
    build(right, mid + 1, tR);
    st[node] = Node((st[left].fwd * expo[st[right].len] + st[
         right].fwd) % M,
                    (st[right].rev * expo[st[left].len] + st[
                        left].rev) % M,
                    st[left].len + st[right].len);
}

void update(int node, int tL, int tR, int i, int64_t v) {
    if (tL >= i && tR <= i) {
        st[node] = Node(v, v, 1);
        return;
    }
    if (tR < i || tL > i) return;

    int mid = (tL + tR) / 2;
    int left = 2 * node, right = 2 * node + 1;
    update(left, tL, mid, i, v);
    update(right, mid + 1, tR, i, v);
    st[node] = Node((st[left].fwd * expo[st[right].len] + st[
         right].fwd) % M,
                    (st[right].rev * expo[st[left].len] + st[
                        left].rev) % M,
                    st[left].len + st[right].len);
}

Node query(int node, int tL, int tR, int qL, int qR) {
    if (tL >= qL && tR <= qR) {
        return Node(st[node].fwd, st[node].rev, st[node].len)
            ;
    }
    if (tR < qL || tL > qR) {
        return Node(0, 0, 0);
    }
    int mid = (tL + tR) / 2;
    auto QL = query(2 * node, tL, mid, qL, qR);
    auto QR = query(2 * node + 1, mid + 1, tR, qL, qR);
    return Node((QL.fwd * expo[QR.len] + QR.fwd) % M, (QR.rev
         * expo[QL.len] + QL.rev) % M, QL.len + QR.len);
}
```

6.3 Suffix Array LCP

```cpp
// #pragma once
struct SuffixArray {
```

```
vector<int> sa, lcp;

SuffixArray(string& s, int lim = 256) {
    int n = s.size() + 1, k = 0, a, b;
    vector<int> x(s.begin(), s.end()), y(n), ws(max(n,
        lim));
    x.push_back(0), sa = lcp = y;
    iota(sa.begin(), sa.end(), 0);

    // Build suffix array using doubling approach
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim
        = p) {
        p = j;
        iota(y.begin(), y.end(), n - j); // Initialize y
            with indices from n-j to n-1
        for (int i = 0; i < n; i++) if (sa[i] >= j) y[p
            ++] = sa[i] - j;
        fill(ws.begin(), ws.end(), 0); // Reset counting
            array
        for (int i = 0; i < n; i++) ws[x[i]]++; // Count
            occurrences of ranks
        for (int i = 1; i < lim; i++) ws[i] += ws[i - 1];
             // Convert counts to positions
        for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            // Sorting suffixes based on 1st part
        swap(x, y);
        p = 1, x[sa[0]] = 0;
        for (int i = 1; i < n; i++) {
            a = sa[i - 1], b = sa[i];
            x[b] = (y[a] == y[b] && y[a + j] == y[b + j])
                ? p - 1 : p++; // Compare suffixes
        }
    }

    // Compute LCP array
    for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
        for (k && k--, j = sa[x[i] - 1]; s[i + k] == s[j
            + k]; k++);
    }
};

void printSA(SuffixArray& sufa, string& s) {
    auto& lcp = sufa.lcp, sa = sufa.sa;
    for (int i = 1; i <= s.size(); i++)
        cout << lcp[i] << ' ' << sa[i] << ' ' << s.substr(sa[
            i]) << endl;
    cout << endl;
}

// // Create a SuffixArray object
// SuffixArray sufa(s);
// sufa.sa; // Suffix array 1 based
// sufa.lcp; // LCP array 1 based
// printSA(sufa, s); // prints SA, LCP, and substrings
```

6.4 Z-Function [MB]

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

/*
 tested by ac
 submission: https://codeforces.com/contest/432/submission
     /145953901
 problem: https://codeforces.com/contest/432/problem/D
*/
std::vector<int> z_function(const std::string &s)
{
 int n = (int)s.size();
 std::vector<int> z(n, 0);
 for (int i = 1, l = 0, r = 0; i < n; i++)
 {
  if (i <= r)
   z[i] = std::min(r - i + 1, z[i - 1]);
  while (i + z[i] < n && s[z[i]] == s[i + z[i]])
   z[i]++;
  if (i + z[i] - 1 > r)
   l = i, r = i + z[i] - 1;
 }
 return z;
}
```