Team Notebook

NSU_TooMuchAC

January 27, 2023

Contents

1 — 2
- 1.1 Build and Run - Linux [NK] . . . 2
- 1.2 Custom Hash [MB] . . . 2
- 1.3 Debug [MB] . . . 2
- 1.4 GNU PBDS [NK] . . . 2
- 1.5 Macros [NK] . . . 3
- 1.6 Starter [MB] . . . 3
- 1.7 Stress Test - Linux [SA] . . . 3

2 Data Structures 3
- 2.1 2D Prefix Sum [SA] . . . 3
- 2.2 Articulation Points in O(N + M) [NK] . . . 4
- 2.3 BIT - Binary Indexed Tree [MB] . . . 4
- 2.4 Bigint (string) operations [NK[. . . 4
- 2.5 Bridges Online [NK] . . . 5
- 2.6 Bridges in O(N + M) [NK] . . . 6
- 2.7 DSU - Disjoint Set Union [NK] . . . 6
- 2.8 LCA - Lowest Common Ancestor [MB] . . . 6
- 2.9 LCA - Lowest Common Ancestor [SA] . . . 7
- 2.10 Mos Algorithm [MB] . . . 7
- 2.11 SCC, Condens Graph [NK] . . . 8
- 2.12 Segment Tree - Lazy [MB] . . . 8
- 2.13 Segment Tree [MB] . . . 9
- 2.14 Sparse Table [MB] . . . 10
- 2.15 Sparse Table [SA] . . . 10

3 Equations 10
- 3.1 Combinatorics . . . 10
  - 3.1.1 General . . . 10
  - 3.1.2 Catalan Numbers . . . 12
  - 3.1.3 Narayana numbers . . . 12
  - 3.1.4 Stirling numbers of the first kind . . . 12
  - 3.1.5 Stirling numbers of the second kind . . . 13
  - 3.1.6 Bell number . . . 13
- 3.2 Math . . . 13
  - 3.2.1 General . . . 13
  - 3.2.2 Fibonacci Number . . . 14
  - 3.2.3 Pythagorean Triples . . . 14
  - 3.2.4 Sum of Squares Function . . . 15
- 3.3 Miscellaneous . . . 15
- 3.4 Number Theory . . . 15
  - 3.4.1 General . . . 15
  - 3.4.2 Divisor Function . . . 16
  - 3.4.3 Euler's Totient function . . . 16
  - 3.4.4 Mobius Function and Inversion . . . 17
  - 3.4.5 GCD and LCM . . . 17
  - 3.4.6 Legendre Symbol . . . 17

4 Graph 18
- 4.1 Edge Remove CC [MB] . . . 18
- 4.2 Kruskal's [NK] . . . 18
- 4.3 Re-rooting a Tree [MB] . . . 19

5 Math 19
- 5.1 BinPow - Modular Binary Exponentiation [NK] 19
- 5.2 Combinatrics [MB] . . . 20
- 5.3 Mathematical Progression [SA] . . . 20
- 5.4 Miller Rabin - Primality Test [SK] . . . 20
- 5.5 Modular Inverse w Ext GCD [NK] . . . 21
- 5.6 Pollard's Rho Algorithm [SK] . . . 21
- 5.7 Sieve Phi (Segmented) [NK[. . . 21
- 5.8 Sieve Phi [MB] . . . 22
- 5.9 Sieve Phi [NK] . . . 22
- 5.10 Sieve Primes (Segmented) [NK] . . . 23
- 5.11 Sieve Primes [MB] . . . 23

6 String 23
- 6.1 Hashing [MB] . . . 23
- 6.2 String Hashing With Point Updates [SA] . . . 24
- 6.3 Z-Function [MB] . . . 24

1 —

1.1 Build and Run - Linux [NK]

```
#!/bin/bash

# Get the name of the current directory
dir_name=$(basename $(pwd))

# Set the names of the input and output files
in_file="$dir_name.in"
out_file="$dir_name.out"

# Compile the CPP file
g++ "$dir_name.cpp" -o "$dir_name"

# Run the compiled program, redirecting input and output
./"$dir_name" < "$in_file" > "$out_file"

# Delete the binary executable
rm "$dir_name"
```

1.2 Custom Hash [MB]

```
#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;
```

1.3 Debug [MB]

```
template <typename T, typename C = typename T::value_type>
typename enable_if<!is_same<T, string>::value, ostream&>::
     type operator<<(ostream& out, const T& c) {
    for (auto it = c.begin(); it != c.end(); it++)
        out << (it == c.begin() ? "{" : ",") << *it;
    return out << (c.empty() ? "{" : "") << "}";
}

template <typename T, typename S>
ostream& operator<<(ostream& out, const pair<T, S>& p) {
    return out << "{" << p.first << ", " << p.second << "}";
}

#define dbg(...) _dbg_print(#__VA_ARGS__, __VA_ARGS__);

template <typename Arg1>
void _dbg_print(const char* name, Arg1&& arg1) {
    if (name[0] == ' ') name++;
    cout << "[" << name << ": " << arg1 << "]\n";
}

template <typename Arg1, typename... Args>
void _dbg_print(const char* names, Arg1&& arg1, Args&&...
     args) {
    const char* comma = strchr(names + 1, ',');
    cout << "[";
    cout.write(names, comma - names) << ": " << arg1 << "] ";
    _dbg_print(comma + 1, args...);
}
```

1.4 GNU PBDS [NK]

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/trie_policy.hpp>
namespace pbds = __gnu_pbds;

// - `find_by_order(i)` to get the `i`-th element (0-indexed
     )
// - `order_of_key(k)` to get the number of elements/keys
     strictly smaller than the key `k`
template <class Key,
          class Mapped = pbds::null_type,
          class Cmp_Fn = std::less<Key>>
using Ordered_Map = pbds::tree<Key,
                               Mapped,
                               Cmp_Fn,
                               pbds::rb_tree_tag,
                               pbds::
                                   tree_order_statistics_node_update
                                   >;
template <class Key,
          class Cmp_Fn = std::less<Key>>
using Ordered_Set = pbds::tree<Key,
                               pbds::null_type,
                               Cmp_Fn,
                               pbds::rb_tree_tag,
                               pbds::
                                   tree_order_statistics_node_update
                                   >;

template <class Key,
          class Mapped,
          class Hash_Fn = std::hash<Key>,
          class Eq_Fn = std::equal_to<Key>>
using Hash_Map = pbds::gp_hash_table<Key,
                                     Mapped,
                                     Hash_Fn,
                                     Eq_Fn>;
template <class Key,
          class Hash_Fn = std::hash<Key>,
          class Eq_Fn = std::equal_to<Key>>
using Hash_Set = pbds::gp_hash_table<Key,
                                     pbds::null_type,
                                     Hash_Fn,
                                     Eq_Fn>;

// GNU PBDS prefix-search based "PATRICIA" trie:
template <class Key,
          class Mapped,
          class Access_Traits = pbds::
              trie_string_access_traits<>>
using Trie_Map = pbds::trie<Key,
                            Mapped,
                            Access_Traits,
                            pbds::pat_trie_tag,
```

```
                        pbds::
                            trie_prefix_search_node_update
                            >;
template <class Key,
          class Access_Traits = pbds::
              trie_string_access_traits<>>
using Trie_Set = pbds::trie<Key,
                        pbds::null_type,
                        Access_Traits,
                        pbds::pat_trie_tag,
                        pbds::
                            trie_prefix_search_node_update
                            >;

template <class Int_Type = int>
struct Trie_Bits_Access_Traits {
   // Bit-Access Definitions (not in the docs)
   using bit_const_iterator = std::_Bit_const_iterator;
   using bit_field_type = std::_Bit_type;
   static constexpr int bit_field_size = std::_S_word_bit;

   // Key-Type Definitions
   using size_type = int;
   using key_type = Int_Type;
   using const_key_reference = const key_type&;

   // Element-Type Definitions
   using e_type = bool;
   using const_iterator = bit_const_iterator;
   static constexpr int min_e_val = 0;
   static constexpr int max_e_val = 1;
   static constexpr int max_size = 2;

   // Methods
   static constexpr size_type e_pos(e_type e) { return e; }
   static constexpr const_iterator begin(const_key_reference
        r_key) {
      return bit_const_iterator((bit_field_type*)(&r_key),
          0);
   }
   static constexpr const_iterator end(const_key_reference
       r_key) {
      return bit_const_iterator((bit_field_type*)(&r_key),
          bit_field_size - __builtin_clzll(r_key));
   }
};
```

1.5 Macros [NK]

```
#define LT(x, y) (((x) + eps) < (y))
#define GT(x, y) (((x)-eps) > (y))
#define EQ(x, y) (abs((x) - (y)) < eps)
#define LE(x, y) (LT(x, y) || EQ(x, y))
#define GE(x, y) (GT(x, y) || EQ(x, y))
#define NE(x, y) (!EQ(x, y))
#define CSB(x) __builtin_popcountll(staic_cast<unsigned long
      long>(x))
#define CLZ(x) __builtin_clzll(staic_cast<unsigned long long
     >(x))
#define CTZ(x) __builtin_ctzll(staic_cast<unsigned long long
     >(x))
#define ISPOW2(x) ((x) && !((x) & ((x)-1)))
#define LOG2_F(x) (63 - CLZ(x))
#define LOG2_C(x) (LOG2_F(x) + !ISPOW2(x))
#define GETBIT(x, i) (((x) >> (i)) & 1)
#define SETBIT(x, i) ((x) | (1LL << (i)))
#define CLRBIT(x, i) ((x) & ~(1LL << (i)))
#define INVBIT(x, i) ((x) ^ (1LL << (i)))
#define GETBITS(x, i, j) (((x) >> (i)) & ((1LL << ((j) - (i)
     )) - 1))
#define SETBITS(x, i, j) ((x) | (((1LL << ((j) - (i))) - 1)
     << (i)))
#define CLRBITS(x, i, j) ((x) & ~(((1LL << ((j) - (i))) - 1)
      << (i)))
#define INVBITS(x, i, j) ((x) ^ (((1LL << ((j) - (i))) - 1)
     << (i)))
```

1.6 Starter [MB]

```
#if defined LOCAL && !defined ONLINE_JUDGE
#include "debug.cpp"
#else
#include <bits/stdc++.h>
using namespace std;
#define dbg(...) ;
#endif

typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;

#define mem(x, n) memset(x, n, sizeof(x))
#define all(x) x.begin(), x.end()
#define sz(x) ((int)(x).size())
#define vec vector
inline bool read(auto&... a) { return (((cin >> a) ? true :
     false) && ...); }

inline void print(const auto&... a) { ((cout << a), ...); }
inline void println(const auto&... a) { print(a..., '\n'); }

void run_case([[maybe_unused]] const int& TC) {}

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

   int tt = 1;
   read(tt);

   for (int tc = 1; tc <= tt; tc++)
      run_case(tc);

   return 0;
}
```

1.7 Stress Test - Linux [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 2D Prefix Sum [SA]

```
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];
}
```

2.2 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);
    }
}
```

2.3 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;
  }
 }
};
```

2.4 Bigint (string) operations [NK[

```
namespace bigint {
    constexpr int base = 10;

    int digit_value(char c) {
        if (c >= '0' && c <= '9') return (int)(c - '0');
        if (c >= 'A' && c <= 'Z') return (int)(c - 'A' + 10);
        if (c >= 'a' && c <= 'z') return (int)(c - 'a' + 36);
        return -1;
    }

    char digit_char(int n) {
        if (n >= 0 && n <= 9) return (char)(n + '0');
        if (n >= 10 && n <= 35) return (char)(n - 10 + 'A');
        if (n >= 36 && n <= 61) return (char)(n - 36 + 'a');
        return ' ';
    }

    string add(const string& a, const string& b) {
        string sum;
        int i = a.length() - 1, j = b.length() - 1, carry =
            0;
        while (i >= 0 || j >= 0) {
            int temp = carry +
                    (i < 0 ? 0 : digit_value(a[i--])) +
                    (j < 0 ? 0 : digit_value(b[j--]));
            carry = temp / base;
            sum += digit_char(temp % base);
        }
        if (carry > 0) sum += digit_char(carry);
        while (sum.length() > 1 && sum[sum.length() - 1] == '
            0') {
            sum.pop_back();
        }
        reverse(sum.begin(), sum.end());
        return sum;
    }

    string multiply(const string& a, const string& b) {
        string prod = "0";
        int shift = 0, carry = 0;
        for (int j = b.length() - 1; j >= 0; j--) {
            string prod_temp(shift++, '0');
            carry = 0;
```

```
        for (int i = a.length() - 1; i >= 0; i--) {
            int temp = carry + digit_value(a[i]) *
                digit_value(b[j]);
            carry = temp / base;
            prod_temp += digit_char(temp % base);
        }
        if (carry > 0) prod_temp += digit_char(carry);
        reverse(prod_temp.begin(), prod_temp.end());
        prod = add(prod, prod_temp);
    }
    while (prod.length() > 1 && prod[prod.length() - 1]
        == '0') {
        prod.pop_back();
    }
    return prod;
}

struct div_result {
    string quot;
    int64_t rem;
};

div_result divide(const string& num, int64_t divisor) {
    div_result result;
    int64_t remainder = 0;
    for (int i = 0; i < num.length(); i++) {
        remainder = (remainder * base) + digit_value(num[
            i]);
        result.quot += digit_char(remainder / divisor);
        remainder %= divisor;
    }
    int clz = 0;
    while (clz < result.quot.length() - 1 && result.quot[
        clz] == '0') {
        clz++;
    }
    result.quot = result.quot.substr(clz);
    result.rem = remainder;
    return result;
}
} // namespace bigint
```

2.5 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);
    }
}
```

2.6 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);
    }
}
```

2.7 DSU - Disjoint Set Union [NK]

```
struct DSU {
    int n_nodes = 0;
    int n_components = 0;
    vector<int> component_size;
    vector<int> component_root;

    DSU(int n_nodes, bool make_all_nodes = false)
        : n_nodes(n_nodes),
          component_root(n_nodes, -1),
          component_size(n_nodes, 0) {
        if (make_all_nodes) {
            for (int i = 0; i < n_nodes; ++i) {
                make_node(i);
            }
        }
    }

    void make_node(int v) {
        if (component_root[v] == -1) {
            component_root[v] = v;
            component_size[v] = 1;
            ++n_components;
        }
    }

    int root(int v) {
        auto res = v;
        while (component_root[res] != res) {
            res = component_root[res];
        }
        while (v != res) {
            auto u = component_root[v];
            component_root[v] = res;
            v = u;
        }
        return res;
    }

    int connect(int u, int v) {
        u = root(u), v = root(v);
        if (u == v) return u;
        if (component_size[u] < component_size[v]) {
            swap(u, v);
        }
        component_root[v] = u;
        component_size[u] += component_size[v];
        --n_components;
    }
};
```

2.8 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)
         ];
    }
};
```

2.9 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;
}
```

2.10 Mos Algorithm [MB]

```
const int N = 3e4 + 5;
const int blck = sqrt(N) + 1;

struct Query
{
 int l, r, i;
 bool operator<(const Query q) const
 {
  if (this->l / blck == q.l / blck)
   return this->r < q.r;
  return this->l / blck < q.l / blck;
 }
};

vector<int> mos_alogorithm(vector<Query> &queries, vector<
     int> &a)
{
 vector<int> answers(queries.size());
 sort(queries.begin(), queries.end());

 int sza = 1e6 + 5;
 vector<int> freq(sza);

 int cnt = 0;

 auto add = [&](int x) -> void
 {
  freq[x]++;
  if (freq[x] == 1)
   cnt++;
 };

 auto remove = [&](int x) -> void
 {
  freq[x]--;
  if (freq[x] == 0)
   cnt--;
 };

 int l = 0;
 int r = -1;
 for (Query q : queries)
 {
  while (l > q.l)
  {
   l--;
   add(a[l]);
  }
```

```
  while (r < q.r)
  {
   r++;
   add(a[r]);
  }
  while (l < q.l)
  {
   remove(a[l]);
   l++;
  }
  while (r > q.r)
  {
   remove(a[r]);
   r--;
  }
  answers[q.i] = cnt;
 }
 return answers;
}

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

 vector<int> a(n);
 for (int i = 0; i < n; i++)
  cin >> a[i];

 int q;
 cin >> q;

 vector<Query> qr(q);

 for (int i = 0; i < q; i++)
 {
  int l, r;
  cin >> l >> r;

  l--, r--;
  qr[i].l = l, qr[i].r = r, qr[i].i = i;
 }

 vector<int> res = mos_alogorithm(qr, a);

 for (int i = 0; i < q; i++)
  cout << res[i] << endl;

 return 0;
}
```

2.11 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);
        }
}
```

2.12 Segment Tree - Lazy [MB]

```
template <typename T, typename F, T(*op)(T, T), F(*
    lazy_to_lazy)(F, F), T(*lazy_to_seg)(T, F, int, int)>
struct LazySegTree
{
private:
 std::vector<T> segt;
 std::vector<F> lazy;
 int n;
 T neutral;
 F lazyE;
 int left(int si) { return si * 2; }
 int right(int si) { return si * 2 + 1; }
 int midpoint(int ss, int se) { return (ss + (se - ss) / 2);
      }
 T query(int ss, int se, int si, int qs, int qe)
 {
  // **** //
  if (lazy[si] != lazyE)
```

```
 {
  F curr = lazy[si];
  lazy[si] = lazyE;
  segt[si] = lazy_to_seg(segt[si], curr, ss, se);
  if (ss != se)
  {
   lazy[left(si)] = lazy_to_lazy(lazy[left(si)], curr);
   lazy[right(si)] = lazy_to_lazy(lazy[right(si)], curr);
  }
 }
 if (se < qs || qe < ss)
  return neutral;
 if (qs <= ss && qe >= se)
  return segt[si];
 int mid = midpoint(ss, se);
 return op(query(ss, mid, left(si), qs, qe), query(mid + 1,
       se, right(si), qs, qe));
}

void update(int ss, int se, int si, int qs, int qe, F val)
{
 // **** //
 if (lazy[si] != lazyE)
 {
  F curr = lazy[si];
  lazy[si] = lazyE;
  segt[si] = lazy_to_seg(segt[si], curr, ss, se);
  if (ss != se)
  {
   lazy[left(si)] = lazy_to_lazy(lazy[left(si)], curr);
   lazy[right(si)] = lazy_to_lazy(lazy[right(si)], curr);
  }
 }
 if (se < qs || qe < ss)
  return;
 if (qs <= ss && qe >= se)
 {
  // **** //
  segt[si] = lazy_to_seg(segt[si], val, ss, se);

  if (ss != se)
  {
   lazy[left(si)] = lazy_to_lazy(lazy[left(si)], val);
   lazy[right(si)] = lazy_to_lazy(lazy[right(si)], val);
  }
  return;
 }

 int mid = midpoint(ss, se);
 update(mid + 1, se, si * 2 + 1, qs, qe, val);
 update(ss, mid, left(si), qs, qe, val);

 segt[si] = op(segt[left(si)], segt[right(si)]);
}
void build(const std::vector<T> &a, int si, int ss, int se)
{
 if (ss == se)
 {
  segt[si] = a[ss];
  return;
 }
 int mid = midpoint(ss, se);
 build(a, left(si), ss, mid);
 build(a, right(si), mid + 1, se);
 segt[si] = op(segt[left(si)], segt[right(si)]);
}
public:
 LazySegTree() : n(0) {}
 LazySegTree(int sz, T ini, T _neutral, F _lazyE)
 {
  this->n = sz + 1;
  this->neutral = _neutral;
  this->lazyE = _lazyE;
  segt.resize(n * 4 + 5, ini);
  lazy.resize(n * 4 + 5, _lazyE);
 }
 LazySegTree(const std::vector<T> &arr, T ini, T _neutral, F
       _lazyE) : LazySegTree((int)arr.size(), ini, _neutral,
      _lazyE)
 {
  init(arr);
 }
 void init(const std::vector<T> &arr) { this->n = (int)arr.
      size(); build(arr, 1, 0, n - 1); }
 T get(int qs, int qe) { return query(0, n - 1, 1, qs, qe);
      }
 void set(int from, int to, F val) { update(0, n - 1, 1,
      from, to, val); }
};

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

int lazy_to_seg(int seg, int lazy_v, int l, int r)
{
 return seg + (lazy_v * (r - l + 1));
}

int lazy_to_lazy(int curr_lazy, int input_lazy)
{
 return curr_lazy + input_lazy;
}
```

2.13 Segment Tree [MB]

```
template <typename T, T(*op)(T, T)>
struct SegTree
{
private:
 std::vector<T> segt;
 int n;
 T e;
 int left(int si) { return si * 2; }
 int right(int si) { return si * 2 + 1; }
 int midpoint(int ss, int se) { return (ss + (se - ss) / 2);
      }
 T query(int ss, int se, int qs, int qe, int si)
 {
  if (se < qs || qe < ss)
   return e;
  if (qs <= ss && qe >= se)
   return segt[si];
  int mid = midpoint(ss, se);
  return op(query(ss, mid, qs, qe, left(si)), query(mid + 1,
       se, qs, qe, right(si)));
 }
 void update(int ss, int se, int key, int si, T val)
 {
  if (ss == se)
  {
   segt[si] = val;
   return;
  }
  int mid = midpoint(ss, se);
  if (key > mid)
   update(mid + 1, se, key, right(si), val);
  else
   update(ss, mid, key, left(si), val);
  segt[si] = op(segt[left(si)], segt[right(si)]);
 }
 void build(const std::vector<T> &a, int si, int ss, int se)
 {
  if (ss == se)
  {
   segt[si] = a[ss];
   return;
```

```
 }
 int mid = midpoint(ss, se);
 build(a, left(si), ss, mid);
 build(a, right(si), mid + 1, se);
 segt[si] = op(segt[left(si)], segt[right(si)]);
}
public:
 SegTree() : n(0) {}
 SegTree(int sz, T _e)
 {
  this->e = _e;
  this->n = sz;
  segt.resize(n * 4 + 5, _e);
 }
 SegTree(const std::vector<T> &arr, T _e) : SegTree((int)arr
     .size(), _e) { init(arr); }
 void init(const std::vector<T> &arr) { this->n = (int)(arr.
     size());build(arr, 1, 0, n - 1); }
 T get(int qs, int qe) { return query(0, n - 1, qs, qe, 1);
     }
 void set(int key, T val) { update(0, n - 1, key, 1, val); }
};

int op(int a, int b)
{
 return min(a, b);
}
```

2.14 Sparse Table [MB]

```
template <typename T, T (*op)(T, T)>
struct SparseTable {
private:
    std::vector<std::vector<T>> st;
    int n, lg;
    std::vector<int> logs;
    T e;

public:
    SparseTable() : n(0) {}

    SparseTable(int _n) {
        this->n = _n;
        int bit = 0;
        while ((1 << bit) <= n)
            bit++;
        this->lg = bit;

        st.resize(n, std::vector<T>(lg));
        logs.resize(n + 1, 0);
        logs[1] = 0;
        for (int i = 2; i <= n; i++) {
            logs[i] = logs[i / 2] + 1;
        }
    }

    SparseTable(const std::vector<T>& a) : SparseTable((int)a
         .size()) {
        init(a);
    }

    void init(const std::vector<T>& a) {
        this->n = (int)a.size();

        for (int i = 0; i < n; i++) {
            st[i][0] = a[i];
        }

        for (int j = 1; j <= lg; j++) {
            for (int i = 0; i + (1 << j) <= n; i++) {
                st[i][j] = op(st[i][j - 1], st[std::min(i + (1
                        << (j - 1)), n - 1)][j - 1]);
            }
        }
    }

    T get(int l, int r) {
        int j = logs[r - l + 1];
        return op(st[l][j], st[r - (1 << j) + 1][j]);
    }
};

int min(int a, int b) {
    return std::min(a, b);
}
```

2.15 Sparse Table [SA]

```
const int N = 100001, LG = 18;
int st[N][LG];

void sparse_table(vector<int>& a, int n) {
    for (int i = 0; i < n; ++i) {
        st[i][0] = a[i];
    }

    for (int j = 1; j < LG; ++j) {
        for (int i = 0; i + (1 << j) - 1 < n; ++i) {
            st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1)
                )][j - 1]);
        }
    }
}

int rmq(int L, int R) {
    int lg = __lg(R - L + 1);
    return min(st[L][lg], st[R - (1 << lg) + 1][lg]);
}
```

3 Equations

3.1 Combinatorics

3.1.1 General

1. $\displaystyle\sum_{0\le k\le n}\binom{n-k}{k} = Fib_{n+1}$
2. $\displaystyle\binom{n}{k} = \binom{n}{n-k}$
3. $\displaystyle\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$
4. $\displaystyle k\binom{n}{k} = n\binom{n-1}{k-1}$
5. $\displaystyle\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$
6. $\displaystyle\sum_{i=0}^{n}\binom{n}{i} = 2^n$
7. $\displaystyle\sum_{i\ge 0}\binom{n}{2i} = 2^{n-1}$
8. $\displaystyle\sum_{i\ge 0}\binom{n}{2i+1} = 2^{n-1}$
9. $\displaystyle\sum_{i=0}^{k}(-1)^i\binom{n}{i} = (-1)^k\binom{n-1}{k}$

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 $2^n - 1, n \geq 0$, is odd.

24. An integer $n \geq 2$ 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 $n \geq m \geq 0$ 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 $\leq k$.

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

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, $m = m_k p^k + m_{k-1}p^{k-1} + \ldots + m_1 p + m_0$, and $n = n_k p^k + n_{k-1}p^{k-1} + \ldots + n_1 p + n_0$ are the base p expansions of m and n respectively. This uses the convention that $\binom{m}{n} = 0$,when $m < n$.

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

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 $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
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, 3 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. $a^k - b^k = (a-b) \cdot (a^{k-1}b^0 + a^{k-2}b^1 + \ldots + a^0 b^{k-1})$
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. $a \cdot b \ge c \to a \ge \left\lfloor \frac{c}{b} \right\rfloor$ 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:
$$\left(\sum_{k=1}^{n} a_k^2\right)\left(\sum_{k=1}^{n} b_k^2\right) - \left(\sum_{k=1}^{n} a_k b_k\right)^2 = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (a_i b_j - a_j b_i)$$
$$= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} (a_i b_j - a_j$$
12. $\sum_{i=1}^{n} i a^i = \frac{a(na^{n+1} - (n+1)a^n + 1)}{(a-1)^2}$
13. Vieta's formulas: Any general polynomial of degree n
$$p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$
(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 $r_1, r_2, \ldots, r_n$.
$$\begin{cases} r_1 + r_2 + ... + r_{n-1} + r_n = -\frac{a_{n-1}}{a_n} \\ (r_1 r_2 + r_1 r_3 + ... + r_1 r_n) + (r_2 r_3 + r_2 r_4 + ... + r_2 r_n) + ... + \\ \vdots \\ r_1 r_2 ... r_n = (-1)^n \frac{a_0}{a_n}. \end{cases}$$
Vieta's formulas can equivalently be written as
$$\sum_{1 \le i_1 < i_2 < ... < i_k \le n} \left(\prod_{j=1}^{k} r_{i_j}\right) = (-1)^k \frac{a_{n-k}}{a_n},$$

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

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. $F_n = \displaystyle\sum_{k=0}^{\lfloor \frac{n-1}{2} \rfloor} \binom{n-k-1}{k}$
3. $F_n = \dfrac{1}{\sqrt{5}} (\dfrac{1+\sqrt{5}}{2})^n - \dfrac{1}{\sqrt{5}} (\dfrac{1-\sqrt{5}}{2})^n$
4. $\displaystyle\sum_{i=1}^{n} F_i = F_{n+2} - 1$
5. $\displaystyle\sum_{i=0}^{n-1} F_{2i+1} = F_{2n}$
6. $\displaystyle\sum_{i=1}^{n} F_{2i} = F_{2n+1} - 1$
7. $\displaystyle\sum_{i=1}^{n} F_i^2 = F_n F_{n+1}$
8. $F_m F_{n+1} - F_{m-1} F_n = (-1)^n F_{m-n}$ $F_{2n} = F_{n+1}^2 - F_{n-1}^2 = F_n(F_{n+1} + F_{n-1})$
9. $F_m F_n + F_{m-1} F_{n-1} = F_{m+n-1}$ $F_m F_{n+1} + F_{m-1} F_n = F_{m+n}$
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 $n = 2^g p_1^{f_1} p_2^{f_2} \ldots q_1^{h_1} q_2^{h_2} \ldots$, 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 $h_1, h_2, \ldots$ 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) $D(x) = \sum_{n\leq x} \sigma_0(n)$
   (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 $\leq n$. 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 ≥ 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 $n \geq 3$. 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. $\sum_{1\leq k\leq n, \gcd(k,n)=1} k = \frac{1}{2} n\varphi(n)$ for $n > 1$
16. $\frac{\varphi(n)}{n} = \frac{\varphi(rad(n))}{rad(n)}$ where $rad(n) = \prod_{p|n, p\, prime} p$
17. $\phi(m) \geq \log_2 m$
18. $\phi(\phi(m)) \leq \frac{m}{2}$
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. $\sum_{i=1}^{n} \varphi(i) \cdot \lfloor \frac{n}{i} \rfloor = \frac{n * (n+1)}{2}$
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 $\frac{n}{2}$.

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. $\sum_{n=1}^{N} \mu^2(n) = \sum_{n=1}^{\sqrt{N}} \mu(k) \cdot \left\lfloor \frac{N}{k^2} \right\rfloor$ 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 $g(n) = \sum_{d|n} \mu(d) g\left(\frac{n}{d}\right)$ for every integer $n \geq 1$
6. If $F(n) = \prod_{d|n} f(d)$, then $F(n) = \prod_{d|n} F\left(\frac{n}{d}\right)^{\mu(d)}$
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. $\sum_{i=1}^{n} [\gcd(i, n) = k] = \phi\left(\frac{n}{k}\right)$
12. $\sum_{k=1}^{n} \gcd(k, n) = \sum_{d|n} d \cdot \phi\left(\frac{n}{d}\right)$
13. $\sum_{k=1}^{n} x^{\gcd(k,n)} = \sum_{d|n} x^d \cdot \phi\left(\frac{n}{d}\right)$
14. $\sum_{k=1}^{n} \frac{1}{\gcd(k, n)} = \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{1}{n} \sum_{d|n} d \cdot \phi(d)$
15. $\sum_{k=1}^{n} \frac{k}{\gcd(k, n)} = \frac{n}{2} \cdot \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{n}{2} \cdot \frac{1}{n} \cdot \sum_{d|n} d \cdot \phi(d)$
16. $\sum_{k=1}^{n} \frac{n}{\gcd(k, n)} = 2 * \sum_{k=1}^{n} \frac{k}{\gcd(k, n)} - 1$, for $n > 1$
17. $\sum_{i=1}^{n} \sum_{j=1}^{n} [\gcd(i, j) = 1] = \sum_{d=1}^{n} \mu(d) \lfloor \frac{n}{d} \rfloor^2$
18. $\sum_{i=1}^{n} \sum_{j=1}^{n} \gcd(i, j) = \sum_{d=1}^{n} \phi(d) \lfloor \frac{n}{d} \rfloor^2$
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. $F(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathrm{lcm}(i, j) = \sum_{l=1}^{n} \left(\frac{\left(1 + \lfloor \frac{n}{l} \rfloor\right) \left(\lfloor \frac{n}{l} \rfloor\right)}{2} \right)^2 \sum_{d|l} \mu(d)$
21. $\gcd(\mathrm{lcm}(a, b), \mathrm{lcm}(b, c), \mathrm{lcm}(a, c)) = \mathrm{lcm}(\gcd(a, b), \gcd(b, c), \mathrm{g}$
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

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a quadatric residue modulo } p \text{ and } a \\ -1 & \text{if } a \text{ is a non-quadaratic residue modulo } p, \\ 0 & \text{if } a \equiv 0 \pmod{p} \end{cases}$$

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 $F_{p-\left(\frac{p}{5}\right)} \equiv 0 \pmod p$, $\quad F_p \equiv \left(\frac{p}{5}\right) \pmod p$.

   For example, $\left(\frac{2}{5}\right) = -1, \quad F_3 = 2, \quad F_2 = 1,$
   $\left(\frac{3}{5}\right) = -1, \quad F_4 = 3, \quad F_3 = 2,$
   $\left(\frac{5}{5}\right) = \quad 0, \quad F_5 = 5,$
   $\left(\frac{7}{5}\right) = -1, \quad F_8 = 21, \quad F_7 = 13,$
   $\left(\frac{11}{5}\right) = \quad 1, \quad F_{10} = 55, \quad F_{11} = 89,$
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 Succesffully 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]

```cpp
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]

```cpp
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

5.1 BinPow - Modular Binary Exponentiation [NK]

```cpp
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;
}
```

5.2 Combinatrics [MB]

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

using namespace std;

typedef long long ll;

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);
```

5.3 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);
}
```

5.4 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 3*10**18 (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;
}
```

5.5 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;
}
```

5.6 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;
}
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);
}
```

5.7 Sieve Phi (Segmented) [NK[

```
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;
        }
    }
}
```

5.8 Sieve Phi [MB]

```
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];
    }

    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;
    }

    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 - l] << 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;
    }
};
```

5.9 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);
```

```
            }
        }
    }
}
```

5.10 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);
        }
    }
}
```

5.11 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;
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]

```
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 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 - l]);
  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;
}
```