ACM - ICPC 2022
Team Notebook
Can Tho University

Contents

1 Contest

1.1 C++

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

#ifdef LOCAL
#include "cp/debug.h"
#else
#define debug(...)
#endif

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

const int MOD = (int) 1e9 + 7;
const int INF = 0x3f3f3f3f;

int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    // freopen("input.txt", "r", stdin);
    // freopen("output.txt", "w", stdout);

    return 0;
}
```

1.2 Debug

```cpp
#define debug(...) { string _s = #__VA_ARGS__; replace(begin(_s), end(_s),
    ',', ' '); stringstream _ss(_s); istream_iterator<string> _it(_ss);
    out_error(_it, __VA_ARGS__);}

void out_error(istream_iterator<string> it) { cerr << '\n'; }

template<typename T, typename ...Args>
void out_error(istream_iterator<string> it, T a, Args... args) {
    cerr << " [" << *it << " = " << a << "] ";
    out_error(++it, args...);
}

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

template<class Con, class = decltype(begin(declval<Con>()))>
typename enable_if<!is_same<Con, string>::value, ostream&>::type
operator<<(ostream& os, const Con& container) {
    os << "{";
    for (auto it = container.begin(); it != container.end(); ++it)
        os << (it == container.begin() ? "" : ", ") << *it;
    return os << "}";
}
```

1.3 Java

```java
import java.io.BufferedReader;
import java.util.StringTokenizer;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        FastScanner fs = new FastScanner();
        PrintWriter out = new PrintWriter(System.out);
        int n = fs.nextInt();
        out.println(n);
        out.close(); // don't forget this line.
    }
    static class FastScanner {
        BufferedReader br;
        StringTokenizer st;
        public FastScanner() {
            br = new BufferedReader(new InputStreamReader(System.in));
            st = null;
        }
        public String next() {
            while (st == null || st.hasMoreTokens() == false) {
                try {
                    st = new StringTokenizer(br.readLine());
                }
                catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
            return st.nextToken();
        }

        public int nextInt() {
            return Integer.parseInt(next());
        }

        public long nextLong() {
            return Long.parseLong(next());
        }

        public double nextDouble() {
            return Double.parseDouble(next());
        }
    }
}
```

2 Mathematics

2.1 Trigonometry

2.1.1 Sum - difference identities

$$\sin(u \pm v) = \sin(u)\cos(v) \pm \cos(u)\sin(v)$$
$$\cos(u \pm v) = \cos(u)\cos(v) \mp \sin(u)\sin(v)$$
$$\tan(u \pm v) = \frac{\tan(u) \pm \tan(v)}{1 \mp \tan(u)\tan(v)}$$

2.1.2 Sum to product identities

$$\cos(u) + \cos(v) = 2\cos(\frac{u+v}{2})\cos(\frac{u-v}{2})$$
$$\cos(u) - \cos(v) = -2\sin(\frac{u+v}{2})\sin(\frac{u-v}{2})$$
$$\sin(u) + \sin(v) = 2\sin(\frac{u+v}{2})\cos(\frac{u-v}{2})$$
$$\sin(u) - \sin(v) = 2\cos(\frac{u+v}{2})\sin(\frac{u-v}{2})$$

2.1.3 Product identities

$$\cos(u)\cos(v) = \frac{1}{2}[\cos(u+v) + \cos(u-v)]$$
$$\sin(u)\sin(v) = -\frac{1}{2}[\cos(u+v) - \cos(u-v)]$$
$$\sin(u)\cos(v) = \frac{1}{2}[\sin(u+v) + \sin(u-v)]$$

2.1.4 Double - triple angle identities

$$\sin(2u) = 2\sin(u)\cos(u)$$
$$\cos(2u) = 2\cos^2(u) - 1 = 1 - 2\sin^2(u)$$
$$\tan(2u) = \frac{2\tan(u)}{1 - \tan^2(u)}$$
$$\sin(3u) = 3\sin(u) - 4\sin^3(u)$$
$$\cos(3u) = 4\cos^3(u) - 3\cos(u)$$
$$\tan(3u) = \frac{3\tan(u) - \tan^3(u)}{1 - 3\tan^2(u)}$$

2.2 Sums

$$n^a + n^{a+1} + \cdots + n^b = \frac{n^{b+1} - n^a}{n - 1},\ n \neq 1$$
$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2}\right)^2$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2 + 3n - 1)}{30}$$

3 Data structures

3.1 Sparse table

```cpp
int st[MAXN][K + 1];
for (int i = 0; i < N; i++) {
    st[i][0] = f(array[i]);
}
for (int j = 1; j <= K; j++) {
    for (int i = 0; i + (1 << j) <= N; i++) {
        st[i][j] = f(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
    }
}
// Range Minimum Queries.
int lg[MAXN + 1];
lg[1] = 0;
for (int i = 2; i <= MAXN; i++) {
    lg[i] = lg[i / 2] + 1;
}
int j = lg[R - L + 1];
int minimum = min(st[L][j], st[R - (1 << j) + 1][j]);
// Range Sum Queries.
long long sum = 0;
for (int j = K; j >= 0; j--) {
    if ((1 << j) <= R - L + 1) {
        sum += st[L][j];
        L += 1 << j;
    }
}
```

3.2 Ordered set

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

```

```cpp
template<typename key_type>
using set_t = tree<key_type, null_type, less<key_type>, rb_tree_tag,
    tree_order_statistics_node_update>;

void example() {
    vector<int> nums = {1, 2, 3, 5, 10};
    set_t<int> st(nums.begin(), nums.end());

    cout << *st.find_by_order(0) << '\n'; // 1
    assert(st.find_by_order(-INF) == st.end());
    assert(st.find_by_order(INF) == st.end());

    cout << st.order_of_key(2) << '\n'; // 1
    cout << st.order_of_key(4) << '\n'; //  3
    cout << st.order_of_key(9) << '\n'; // 4
    cout << st.order_of_key(-INF) << '\n'; // 0
    cout << st.order_of_key(INF) << '\n'; // 5
}
```

3.3   Dsu

```cpp
struct Dsu {
    int n;
    vector<int> par, sz;
    Dsu(int _n) : n(_n) {
        sz.resize(n, 1);
        par.resize(n);
        iota(par.begin(), par.end(), 0);
    }
    int find(int v) {
        // finding leader/parrent of set that contains the element v.
        // with {path compression optimization}.
        return (v == par[v] ? v : par[v] = find(par[v]));
    }
    bool same(int u, int v) {
        return find(u) == find(v);
    }
    bool unite(int u, int v) {
        u = find(u); v = find(v);
        if (u == v) return false;
        if (sz[u] < sz[v]) swap(u, v);
        par[v] = u;
        sz[u] += sz[v];
        return true;
    }
    vector<vector<int>> groups() {
        // returns the list of the "list of the vertices in a connected
        component".
        vector<int> leader(n);
        for (int i = 0; i < n; ++i) {
            leader[i] = find(i);
        }
        vector<int> id(n, -1);
        int count = 0;
        for (int i = 0; i < n; ++i) {
            if (id[leader[i]] == -1) {
                id[leader[i]] = count++;
            }
        }
        vector<vector<int>> result(count);
        for (int i = 0; i < n; ++i) {
            result[id[leader[i]]].push_back(i);
        }
        return result;
    }
};
```

3.4   Segment tree

```cpp
/**
 * Description: A segment tree with range updates and sum queries that
     supports three types of operations:
 *   + Increase each value in range [l, r] by x (i.e. a[i] += x).
 *   + Set each value in range [l, r] to x (i.e. a[i] = x).
 *   + Determine the sum of values in range [l, r].
 */
struct SegmentTree {
    int n;
    vector<long long> tree, lazy_add, lazy_set;
    SegmentTree(int _n) : n(_n) {
        int p = 1;
        while (p < n) p *= 2;
        tree.resize(p * 2);
        lazy_add.resize(p * 2);
        lazy_set.resize(p * 2);
    }
    long long merge(const long long &left, const long long &right) {
        return left + right;
    }
    void build(int id, int l, int r, const vector<int> &arr) {
        if (l == r) {
            tree[id] += arr[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(id * 2, l, mid, arr);
        build(id * 2 + 1, mid + 1, r, arr);
        tree[id] = merge(tree[id * 2], tree[id * 2 + 1]);
    }
    void push(int id, int l, int r) {
        if (lazy_set[id] == 0 && lazy_add[id] == 0) return;
        int mid = (l + r) >> 1;
        for (int child : {id * 2, id * 2 + 1}) {
            int range = (child == id * 2 ? mid - l + 1 : r - mid);
            if (lazy_set[id] != 0) {
                lazy_add[child] = 0;
```

```
37                   lazy_set[child] = lazy_set[id];
38                   tree[child] = range * lazy_set[id];
39               }
40               lazy_add[child] += lazy_add[id];
41               tree[child] += range * lazy_add[id];
42           }
43           lazy_add[id] = lazy_set[id] = 0;
44
45       }
46       void update(int id, int l, int r, int u, int v, int amount, bool
         set_value = false) {
47           if (r < u || l > v) return;
48           if (u <= l && r <= v) {
49               if (set_value) {
50                   tree[id] = 1LL * amount * (r - l + 1);
51                   lazy_set[id] = amount;
52                   lazy_add[id] = 0; // clear all previous updates.
53               }
54               else {
55                   tree[id] += 1LL * amount * (r - l + 1);
56                   lazy_add[id] += amount;
57               }
58               return;
59           }
60           push(id, l, r);
61           int mid = (l + r) >> 1;
62           update(id * 2, l, mid, u, v, amount, set_value);
63           update(id * 2 + 1, mid + 1, r, u, v, amount, set_value);
64           tree[id] = merge(tree[id * 2], tree[id * 2 + 1]);
65       }
66       long long get(int id, int l, int r, int u, int v) {
67           if (r < u || l > v) return 0;
68           if (u <= l && r <= v) {
69               return tree[id];
70           }
71           push(id, l, r);
72           int mid = (l + r) >> 1;
73           long long left = get(id * 2, l, mid, u, v);
74           long long right = get(id * 2 + 1, mid + 1, r, u, v);
75           return merge(left, right);
76       }
77 };
```

3.5   Efficient segment tree

```
1  template<typename T> struct SegmentTree {
2      int n;
3      vector<T> tree;
4      SegmentTree(int _n) : n(_n), tree(2 * n) {}
5      T merge(const T &left, const T &right) {
6          return left + right;
7      }
8      template<typename G>
```

```
9      void build(const vector<G> &initial) {
10         assert((int) initial.size() == n);
11         for (int i = 0; i < n; ++i) {
12             tree[i + n] = initial[i];
13         }
14         for (int i = n - 1; i > 0; --i) {
15             tree[i] = merge(tree[i * 2], tree[i * 2 + 1]);
16         }
17     }
18     void modify(int i, int v) {
19         tree[i += n] = v;
20         for (i /= 2; i > 0; i /= 2) {
21             tree[i] = merge(tree[i * 2], tree[i * 2 + 1]);
22         }
23     }
24     T get_sum(int l, int r) {
25         // sum of elements from l to r - 1.
26         T ret{};
27         for (l += n, r += n; l < r; l /= 2, r /= 2) {
28             if (l & 1) ret = merge(ret, tree[l++]);
29             if (r & 1) ret = merge(ret, tree[--r]);
30         }
31         return ret;
32     }
33 };
```

3.6   Persistent lazy segment tree

```
1  struct Vertex {
2      int l, r;
3      long long val, lazy;
4      bool has_changed = false;
5      Vertex() {}
6      Vertex(int _l, int _r, long long _val, int _lazy = 0) : l(_l), r(_r),
         val(_val), lazy(_lazy) {}
7  };
8  struct PerSegmentTree {
9      vector<Vertex> tree;
10     vector<int> root;
11     int build(const vector<int> &arr, int l, int r) {
12         if (l == r) {
13             tree.emplace_back(-1, -1, arr[l]);
14             return tree.size() - 1;
15         }
16         int mid = (l + r) / 2;
17         int left = build(arr, l, mid);
18         int right = build(arr, mid + 1, r);
19         tree.emplace_back(left, right, tree[left].val + tree[right].val);
20         return tree.size() - 1;
21     }
22     int add(int x, int l, int r, int u, int v, int amt) {
23         if (l > v || r < u) return x;
24         if (u <= l && r <= v) {
```

```
25          tree.emplace_back(tree[x].l, tree[x].r, tree[x].val + 1LL * amt
    * (r - l + 1), tree[x].lazy + amt);
26          tree.back().has_changed = true;
27          return tree.size() - 1;
28        }
29        int mid = (l + r) >> 1;
30        push(x, l, mid, r);
31        int left = add(tree[x].l, l, mid, u, v, amt);
32        int right = add(tree[x].r, mid + 1, r, u, v, amt);
33        tree.emplace_back(left, right, tree[left].val + tree[right].val, 0)
    ;
34        return tree.size() - 1;
35    }
36    long long get_sum(int x, int l, int r, int u, int v) {
37        if (r < u || l > v) return 0;
38        if (u <= l && r <= v) return tree[x].val;
39        int mid = (l + r) / 2;
40        push(x, l, mid, r);
41        return get_sum(tree[x].l, l, mid, u, v) + get_sum(tree[x].r, mid +
    1, r, u, v);
42    }
43    void push(int x, int l, int mid, int r) {
44        if (!tree[x].has_changed) return;
45        Vertex left = tree[tree[x].l];
46        Vertex right = tree[tree[x].r];
47        tree.emplace_back(left);
48        tree[x].l = tree.size() - 1;
49        tree.emplace_back(right);
50        tree[x].r = tree.size() - 1;
51
52        tree[tree[x].l].val += tree[x].lazy * (mid - l + 1);
53        tree[tree[x].l].lazy += tree[x].lazy;
54
55        tree[tree[x].r].val += tree[x].lazy * (r - mid);
56        tree[tree[x].r].lazy += tree[x].lazy;
57
58        tree[tree[x].l].has_changed = true;
59        tree[tree[x].r].has_changed = true;
60        tree[x].lazy = 0;
61        tree[x].has_changed = false;
62    }
63 };
```

3.7 Fenwick tree

```
1 using tree_type = long long;
2 struct FenwickTree {
3    int n;
4    vector<tree_type> fenw_coeff, fenw;
5    FenwickTree() {}
6    FenwickTree(int _n) : n(_n) {
7        fenw_coeff.assign(n, 0); // fenwick tree with coefficient (n - i).
8        fenw.assign(n, 0); // normal fenwick tree.
```

```
9    }
10   void build(const vector<int> &A) {
11       assert((int) A.size() == n);
12       vector<int> diff(n);
13       diff[0] = A[0];
14       for (int i = 1; i < n; ++i) {
15           diff[i] = A[i] - A[i - 1];
16       }
17       fenw_coeff[0] = (long long) diff[0] * n;
18       fenw[0] = diff[0];
19       for (int i = 1; i < n; ++i) {
20           fenw_coeff[i] = fenw_coeff[i - 1] + (long long) diff[i] * (n -
    i);
21           fenw[i] = fenw[i - 1] + diff[i];
22       }
23       for (int i = n - 1; i >= 0; --i) {
24           int j = (i & (i + 1)) - 1;
25           if (j >= 0) {
26               fenw_coeff[i] -= fenw_coeff[j];
27               fenw[i] -= fenw[j];
28           }
29       }
30   }
31   void add(vector<tree_type> &fenw, int i, tree_type val) {
32       while (i < n) {
33           fenw[i] += val;
34           i |= (i + 1);
35       }
36   }
37   tree_type __prefix_sum(vector<tree_type> &fenw, int i) {
38       tree_type res{};
39       while (i >= 0) {
40           res += fenw[i];
41           i = (i & (i + 1)) - 1;
42       }
43       return res;
44   }
45   tree_type prefix_sum(int i) {
46       return __prefix_sum(fenw_coeff, i) - __prefix_sum(fenw, i) * (n - i
    - 1);
47   }
48   void range_add(int l, int r, tree_type val) {
49       add(fenw_coeff, l, (n - l) * val);
50       add(fenw_coeff, r + 1, (n - r - 1) * (-val));
51       add(fenw, l, val);
52       add(fenw, r + 1, -val);
53   }
54   tree_type range_sum(int l, int r) {
55       return prefix_sum(r) - prefix_sum(l - 1);
56   }
57 };
```

4 String

4.1 Prefix function

```
1  /**
2   * Description: The prefix function of a string 's' is defined as an array
       pi of length n,
3   * where pi[i] is the length of the longest proper prefix of the substring
4   * s[0..i] which is also a suffix of this substring.
5   * Time complexity: O(|S|).
6   */
7  vector<int> prefix_function(const string &s) {
8      int n = (int) s.length();
9      vector<int> pi(n);
10     pi[0] = 0;
11     for (int i = 1; i < n; ++i) {
12         int j = pi[i - 1]; // try length pi[i - 1] + 1.
13         while (j > 0 && s[j] != s[i]) {
14             j = pi[j - 1];
15         }
16         if (s[j] == s[i]) {
17             pi[i] = j + 1;
18         }
19     }
20     return pi;
21 }
```

4.2 Counting occurrences of each prefix

```
1  vector<int> count_occurrences(const string &s) {
2      vector<int> pi = prefix_function(s);
3      int n = (int) s.size();
4      vector<int> ans(n + 1);
5      for (int i = 0; i < n; ++i) {
6          ans[pi[i]]++;
7      }
8      for (int i = n - 1; i > 0; --i) {
9          ans[pi[i - 1]] += ans[i];
10     }
11     for (int i = 0; i <= n; ++i) {
12         ans[i]++;
13     }
14     return ans;
15     // Input: ABACABA
16     // Output: 4 2 2 1 1 1 1
17 }
```

4.3 Knuth–Morris–Pratt algorithm

```
1  /**
2   * Searching for a substring in a string.
3   * Time complexity: O(N + M).
4   */
5  vector<int> KMP(const string &text, const string &pattern) {
6      int n = (int) text.length();
7      int m = (int) pattern.length();
8      string s = pattern + '$' + text;
9      vector<int> pi = prefix_function(s);
10     vector<int> indices;
11     for (int i = 0; i < (int) s.length(); ++i) {
12         if (pi[i] == m) {
13             indices.push_back(i - 2 * m);
14         }
15     }
16     return indices;
17 }
```

4.4 Manacher's algorithm

```
1  /**
2   * Description: for each position, computes d[0][i] = half length of
3   longest palindrome centered on i (rounded up), d[1][i] = half length of
4   longest palindrome centered on i and i - 1.
5   * Time complexity: O(N).
6   * Tested: https://judge.yosupo.jp/problem/enumerate_palindromes, stress-
       tested.
7   */
8  array<vector<int>, 2> manacher(const string &s) {
9      int n = (int) s.size();
10     array<vector<int>, 2> d;
11     for (int z = 0; z < 2; ++z) {
12         d[z].resize(n);
13         int l = 0, r = 0;
14         for (int i = 0; i < n; ++i) {
15             int mirror = l + r - i + z;
16             d[z][i] = (i > r ? 0 : min(d[z][mirror], r - i));
17             int L = i - d[z][i] - z, R = i + d[z][i];
18             while (L >= 0 && R < n && s[L] == s[R]) {
19                 d[z][i]++; L--; R++;
20             }
21             if (R > r) {
22                 l = L; r = R;
23             }
24         }
25     }
26     return d;
27 }
```

4.5 Trie

```
1  struct Trie {
2      const static int ALPHABET = 26;
3      const static char minChar = 'a';
4      struct Vertex {
5          int next[ALPHABET];
6          bool leaf;
7          Vertex() {
```

```
 8              leaf = false;
 9              fill(next, next + ALPHABET, -1);
10          }
11      };
12      vector<Vertex> trie;
13      Trie() { trie.emplace_back(); }
14
15      void insert(const string &s) {
16          int i = 0;
17          for (const char &ch : s) {
18              int j = ch - minChar;
19              if (trie[i].next[j] == -1) {
20                  trie[i].next[j] = trie.size();
21                  trie.emplace_back();
22              }
23              i = trie[i].next[j];
24          }
25          trie[i].leaf = true;
26      }
27      bool find(const string &s) {
28          int i = 0;
29          for (const char &ch : s) {
30              int j = ch - minChar;
31              if (trie[i].next[j] == -1) {
32                  return false;
33              }
34              i = trie[i].next[j];
35          }
36          return (trie[i].leaf ? true : false);
37      }
38 };
```

4.6   Hashing

```
 1 struct Hash61 {
 2     static const uint64_t MOD = (1LL << 61) - 1;
 3     static uint64_t BASE;
 4     static vector<uint64_t> pw;
 5     uint64_t addmod(uint64_t a, uint64_t b) const {
 6         a += b;
 7         if (a >= MOD) a -= MOD;
 8         return a;
 9     }
10     uint64_t submod(uint64_t a, uint64_t b) const {
11         a += MOD - b;
12         if (a >= MOD) a -= MOD;
13         return a;
14     }
15     uint64_t mulmod(uint64_t a, uint64_t b) const {
16         uint64_t low1 = (uint32_t) a, high1 = (a >> 32);
17         uint64_t low2 = (uint32_t) b, high2 = (b >> 32);
18
19         uint64_t low = low1 * low2;
20         uint64_t mid = low1 * high2 + low2 * high1;
21         uint64_t high = high1 * high2;
22
23         uint64_t ret = (low & MOD) + (low >> 61) + (high << 3) + (mid >>
   29) + (mid << 35 >> 3) + 1;
24         // ret %= MOD:
25         ret = (ret >> 61) + (ret & MOD);
26         ret = (ret >> 61) + (ret & MOD);
27         return ret - 1;
28     }
29     void ensure_pw(int m) {
30         int n = (int) pw.size();
31         if (n >= m) return;
32         pw.resize(m);
33         for (int i = n; i < m; ++i) {
34             pw[i] = mulmod(pw[i - 1], BASE);
35         }
36     }
37
38     vector<uint64_t> pref;
39     int n;
40     template<typename T> Hash61(const T &s) { // strings or arrays.
41         n = (int) s.size();
42         ensure_pw(n);
43         pref.resize(n + 1);
44         pref[0] = 0;
45         for (int i = 0; i < n; ++i) {
46             pref[i + 1] = addmod(mulmod(pref[i], BASE), s[i]);
47         }
48     }
49     inline uint64_t operator()(const int from, const int to) const {
50         assert(0 <= from && from <= to && to < n);
51         // pref[to + 1] - pref[from] * pw[to - from + 1]
52         return submod(pref[to + 1], mulmod(pref[from], pw[to - from + 1]));
53     }
54 };
55 mt19937 rng((unsigned int) chrono::steady_clock::now().time_since_epoch().
   count());
56 uint64_t Hash61::BASE = (MOD >> 2) + rng() % (MOD >> 1);
57 vector<uint64_t> Hash61::pw = vector<uint64_t>(1, 1);
```

5   Number Theory

5.1   Euler's totient function

- Euler's totient function, also known as **phi-function** $\phi(n)$ counts the number of integers between 1 and n inclusive, that are **coprime to** n.

- Properties:

    - Divisor sum property: $\sum_{d|n} \phi(d) = n$.

– $\phi(n)$ is a **prime number** when $n = 3, 4, 6$.

– If p is a prime number, then $\phi(p) = p - 1$.

– If p is a prime number and $k \geq 1$, then $\phi(p^k) = p^k - p^{k-1}$.

– If a and b are **coprime**, then $\phi(ab) = \phi(a) \cdot \phi(b)$.

– In general, for **not coprime** a and b, with $d = gcd(a, b)$ this equation holds: $\phi(ab) = \phi(a) \cdot \phi(b) \cdot \dfrac{d}{\phi(d)}$.

– With $n = p_1^{k_1} \cdot p_2^{k_2} \cdots p_m^{k_m}$:

$$\phi(n) = \phi(p_1^{k_1}) \cdot \phi(p_2^{k_2}) \cdots \phi(p_m^{k_m})$$
$$= n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_m}\right)$$

- Application in Euler's theorem:

  – If $gcd(a, M) = 1$, then:

  $$a^{\phi(M)} \equiv 1 \ (\text{mod } M) \Rightarrow a^n \equiv a^{n \ \text{mod } M} \ (\text{mod } M)$$

  – In general, for arbitrary a, M and $n \geq \log_2 M$:

  $$a^n \equiv a^{\phi(M) + [n \ \text{mod} \ \phi(M)]} \ (\text{mod } M)$$

5.2 Mobius function

- For a positive integer $n = p_1^{k_1} \cdot p_2^{k_2} \cdots p_m^{k_m}$:

$$\mu(n) = \begin{cases} 1, & \text{if } n = 1 \\ 0, & \text{if } \exists k_i > 1 \\ (-1)^m & \text{otherwise} \end{cases}$$

- Properties:

  – $\sum_{d|n} \mu(d) = [n = 1]$.

  – If a and b are **coprime**, then $\mu(ab) = \mu(a) \cdot \mu(b)$.

  – Mobius inversion: let f and g be arithmetic functions:

  $$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu\left(\frac{n}{d}\right) g(d)$$

  .

5.3 Primes

Approximating the number of primes up to n:

n	$\pi(n)$	$\dfrac{n}{\ln n - 1}$
100 ($1e^2$)	25	28
500 ($5e^2$)	95	96
1000 ($1e^3$)	168	169
5000 ($5e^3$)	669	665
10000 ($1e^4$)	1229	1218
50000 ($5e^4$)	5133	5092
100000 ($1e^5$)	9592	9512
500000 ($5e^5$)	41538	41246
1000000 ($1e^6$)	78498	78030
5000000 ($5e^6$)	348513	346622

($\pi(n)$ = the number of primes less than or equal to n, $\dfrac{n}{\ln n - 1}$ is used to approximate $\pi(n)$).

5.4 Wilson's theorem

A positive integer n is a prime if and only if:

$$(n - 1)! \equiv n - 1 \ (\text{mod } n)$$

5.5 Zeckendorf's theorem

The Zeckendorf's theorem states that every positive integer n can be represented uniquely as a sum of one or more distinct non-consecutive Fibonacci numbers. For example:

$$64 = 55 + 8 + 1$$
$$85 = 55 + 21 + 8 + 1$$

```cpp
vector<int> zeckendoft_theorem(int n) {
    vector<int> fibs = {1, 1};
    int sz = 2;
    while (fibs.back() <= n) {
        fibs.push_back(fibs[sz - 1] + fibs[s - 2]);
        sz++:
    }
    fibs.pop_back();
    vector<int> nums;
    int p = sz - 1;
    while (n > 0) {
        if (n >= fibs[p]) {
            nums.push_back(fibs[p]);
            n -= fibs[p];
        }
```

```
16          p--;
17      }
18      return nums;
19 }
```

5.6   Bitwise operation

- $a + b = (a \oplus b) + 2(a \mathbin{\&} b)$
- $a \mid b = (a \oplus b) + (a \mathbin{\&} b)$
- $a \mathbin{\&} (b \oplus c) = (a \mathbin{\&} b) \oplus (a \mathbin{\&} c)$
- $a \mid (b \mathbin{\&} c) = (a \mid b) \mathbin{\&} (a \mid c)$
- $a \mathbin{\&} (b \mid c) = (a \mathbin{\&} b) \mid (a \mathbin{\&} c)$

- $a \mid (a \mathbin{\&} b) = a$
- $a \mathbin{\&} (a \mid b) = a$
- $n = 2^k \Leftrightarrow !(n \mathbin{\&} (n - 1)) = 1$
- $-a = \sim a + 1$
- $(4i) \oplus (4i+1) \oplus (4i+2) \oplus (4i+3) = 0$

- Iterating over all subsets of a set and iterating over all submasks of a mask:

```
1 for (int mask = 0; mask < (1 << n); ++mask) {
2     for (int i = 0; i < n; ++i) {
3         if (mask & (1 << i)) {
4             // do something...
5         }
6     }
7     // Time complexity: O(n * 2^n).
8 }
9 for (int mask = 0; mask < (1 << n); ++mask) {
10     for (int submask = mask; ; submask = (submask - 1) & mask) {
11         // do something...
12         if (submask == 0) break;
13     }
14     // Time complexity: O(3^n).
15 }
```

5.7   Combinatorics

5.7.1   Catalan numbers

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

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}, \; C_0 = 1, \; C_n = \frac{4n-2}{n+1}C_{n-1}$$

- The first 12 Catalan numbers ($n = 0, 1, 2, \ldots, 12$):

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786$$

- Applications of Catalan numbers:

  - difference binary search trees with n vertices from 1 to n.

  - rooted binary trees with $n + 1$ leaves (vertices are not numbered).
  - correct bracket sequence of length $2 * n$.
  - permutation $[n]$ with no 3-term increasing subsequence (i.e. doesn't exist $i < j < k$ for which $a[i] < a[j] < a[k]$).
  - ways a convex polygon of $n + 2$ sides can split into triangles by connecting vertices.

5.7.2   Stirling numbers of the second kind

Partitions of n distinct elements into exactly k non-empty groups.

$$S(n,k) = S(n-1, k-1) + kS(n-1, k)$$

$$S(n,1) = S(n,n) = 1$$

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}i^n$$

5.7.3   Derangements

Permutation of the elements of a set, such that no element appears in its original position (no fixed point). Recursive formulas:

$$D(n) = (n-1)[D(n-1) + D(n-2)] = nD(n-1) + (-1)^n$$

5.8   Pollard's rho algorithm

```
1 const int PRIME_MAX = (int) 4e4; // for handle numbers <= 1e9.
2 const int LIMIT = (int) 1e9;
3 vector<int> primes;
4
5 void linear_sieve(int n);
6 num_type mulmod(num_type a, num_type b, num_type mod);
7 num_type powmod(num_type a, num_type n, num_type mod);
8
9 bool miller_rabin(num_type a, num_type d, int s, num_type mod) {
10     // mod - 1 = a ^ (d * 2^s).
11     num_type x = powmod(a, d, mod);
12     if (x == 1 || x == mod - 1) return true;
13     for (int i = 1; i <= s - 1; ++i) {
14         x = mulmod(x, x, mod);
15         if (x == mod - 1) return true;
16     }
17     return false;
18 }
19 bool is_prime(num_type n, int ITERATION = 10) {
20     if (n < 4) return (n == 2 || n == 3);
21     if (n % 2 == 0 || n % 3 == 0) return false;
22     num_type d = n - 1;
23     int s = 0;
24     while (d % 2 == 0) {
25         d /= 2;
```

```
26          s++;
27      }
28      for (int i = 0; i < ITERATION; ++i) {
29          num_type a = (num_type) (rand() % (n - 2)) + 2;
30          if (miller_rabin(a, d, s, n) == false) {
31              return false;
32          }
33      }
34      return true;
35  }
36  num_type f(num_type x, int c, num_type mod) { // f(x) = (x^2 + c) % mod.
37      x = mulmod(x, x, mod);
38      x += c;
39      if (x >= mod) x -= mod;
40      return x;
41  }
42  num_type pollard_rho(num_type n, int c) {
43      // algorithm to find a random divisor of 'n'.
44      // using random function: f(x) = (x^2 + c) % n.
45
46      // *************** Floyd's cycle detection algorithm ***************
47      // move 1 step and 2 steps.
48      // num_type x = 2, y = 2, d;
49      // while (true) {
50      //     x = f(x, c, n);
51      //     y = f(y, c, n);
52      //     y = f(y, c, n);
53      //     d = __gcd(llabs(x - y), n);
54      //     if (d > 1) break;
55      // }
56      // return d;
57
58      // *************** Brent's cycle detection algorithm ***************
59      // move power of two steps.
60      num_type x = 2, y = x, d;
61      long long p = 1;
62      int dist = 0;
63      while (true) {
64          y = f(y, c, n);
65          dist++;
66          d = __gcd(llabs(x - y), n);
67          if (d > 1) break;
68          if (dist == p) { dist = 0; p *= 2; x = y; }
69      }
70      return d;
71  }
72  void factorize(int n, vector<num_type> &factors);
73  void llfactorize(num_type n, vector<num_type> &factors) {
74      if (n < 2) return;
75      if (n < LIMIT) {
76          factorize(n, factors);
77          return;
```

```
78      }
79      if (is_prime(n)) {
80          factors.emplace_back(n);
81          return;
82      }
83      num_type d = n;
84      for (int c = 2; d == n; c++) {
85          d = pollard_rho(n, c);
86      }
87      llfactorize(d, factors);
88      llfactorize(n / d, factors);
89  }
90  vector<num_type> gen_divisors(vector<pair<num_type, int>> &factors) {
91      vector<num_type> divisors = {1};
92      for (auto &x : factors) {
93          int sz = (int) divisors.size();
94          for (int i = 0; i < sz; ++i) {
95              num_type cur = divisors[i];
96              for (int j = 0; j < x.second; ++j) {
97                  cur *= x.first;
98                  divisors.push_back(cur);
99              }
100         }
101     }
102     return divisors; // this array is NOT sorted yet.
103 }
```

6 Linear algebra
6.1 Gauss elimination

```
1  const double EPS = 1e-9;
2  const int INF = 2; // it doesn't actually have to be infinity or a big
       number
3  int gauss (vector < vector<double> > a, vector<double> & ans) {
4      int n = (int) a.size();
5      int m = (int) a[0].size() - 1;
6      vector<int> where (m, -1);
7      for (int col=0, row=0; col<m && row<n; ++col) {
8          int sel = row;
9          for (int i=row; i<n; ++i)
10             if (abs (a[i][col]) > abs (a[sel][col]))
11                 sel = i;
12         if (abs (a[sel][col]) < EPS)
13             continue;
14         for (int i=col; i<=m; ++i)
15             swap (a[sel][i], a[row][i]);
16         where[col] = row;
17
18         for (int i=0; i<n; ++i)
19             if (i != row) {
20                 double c = a[i][col] / a[row][col];
```

```
21              for (int j=col; j<=m; ++j)
22                  a[i][j] -= a[row][j] * c;
23          }
24      ++row;
25      }
26  ans.assign (m, 0);
27  for (int i=0; i<m; ++i)
28      if (where[i] != -1)
29          ans[i] = a[where[i]][m] / a[where[i]][i];
30  for (int i=0; i<n; ++i) {
31      double sum = 0;
32      for (int j=0; j<m; ++j)
33          sum += ans[j] * a[i][j];
34      if (abs (sum - a[i][m]) > EPS)
35          return 0;
36  }
37  for (int i=0; i<m; ++i)
38      if (where[i] == -1)
39          return INF;
40  return 1;
41 }
```

7 Geometry

7.1 Fundamentals

7.1.1 Point

```
1  const double PI = acos(-1);
2  const double EPS = 1e-9;
3  typedef double ftype;
4  struct point {
5      ftype x, y;
6      point(ftype _x = 0, ftype _y = 0): x(_x), y(_y) {}
7      point& operator+=(const point& other) {
8          x += other.x; y += other.y; return *this;
9      }
10     point& operator-=(const point& other) {
11         x -= other.x; y -= other.y; return *this;
12     }
13     point& operator*=(ftype t) {
14         x *= t; y *= t; return *this;
15     }
16     point& operator/=(ftype t) {
17         x /= t; y /= t; return *this;
18     }
19     point operator+(const point& other) const {
20         return point(*this) += other;
21     }
22     point operator-(const point& other) const {
23         return point(*this) -= other;
24     }
25     point operator*(ftype t) const {
```

```
26         return point(*this) *= t;
27     }
28     point operator/(ftype t) const {
29         return point(*this) /= t;
30     }
31     point rotate(double angle) const {
32         return point(x * cos(angle) - y * sin(angle), x * sin(angle) + y *
   cos(angle));
33     }
34     friend istream& operator>>(istream &in, point &t);
35     friend ostream& operator<<(ostream &out, const point& t);
36     bool operator<(const point& other) const {
37         if (fabs(x - other.x) < EPS)
38             return y < other.y;
39         return x < other.x;
40     }
41 };
42
43 istream& operator>>(istream &in, point &t) {
44     in >> t.x >> t.y;
45     return in;
46 }
47 ostream& operator<<(ostream &out, const point& t) {
48     out << t.x << ' ' << t.y;
49     return out;
50 }
51
52 ftype dot(point a, point b) {return a.x * b.x + a.y * b.y;}
53 ftype norm(point a) {return dot(a, a);}
54 ftype abs(point a) {return sqrt(norm(a));}
55 ftype angle(point a, point b) {return acos(dot(a, b) / (abs(a) * abs(b)));}
56 ftype proj(point a, point b) {return dot(a, b) / abs(b);}
57 ftype cross(point a, point b) {return a.x * b.y - a.y * b.x;}
58 bool ccw(point a, point b, point c) {return cross(b - a, c - a) > EPS;}
59 bool collinear(point a, point b, point c) {return fabs(cross(b - a, c - a))
        < EPS;}
60 point intersect(point a1, point d1, point a2, point d2) {
61     double t = cross(a2 - a1, d2) / cross(d1, d2);
62     return a1 + d1 * t;
63 }
```

7.1.2 Line

```
1  struct line {
2      double a, b, c;
3      line (double _a = 0, double _b = 0, double _c = 0): a(_a), b(_b), c(_c)
        {}
4      friend ostream & operator<<(ostream& out, const line& l);
5  };
6  ostream & operator<<(ostream& out, const line& l) {
7      out << l.a << ' ' << l.b << ' ' << l.c;
8      return out;
9  }
```

```cpp
10  void pointsToLine(const point& p1, const point& p2, line& l) {
11      if (fabs(p1.x - p2.x) < EPS)
12          l = {1.0, 0.0, -p1.x};
13      else {
14          l.a = - (double)(p1.y - p2.y) / (p1.x - p2.x);
15          l.b = 1.0;
16          l.c =  - l.a * p1.x - l.b * p1.y;
17      }
18  }
19  void pointsSlopeToLine(const point& p, double m, line& l) {
20      l.a = -m;
21      l.b = 1;
22      l.c = -l.a * p.x - l.b * p.y;
23  }
24  bool areParallel(const line& l1, const line& l2) {
25      return fabs(l1.a - l2.a) < EPS && fabs(l1.b - l2.b) < EPS;
26  }
27  bool areSame(const line& l1, const line& l2) {
28      return areParallel(l1, l2) && fabs(l1.c - l2.c) < EPS;
29  }
30  bool areIntersect(line l1, line l2, point& p) {
31      if (areParallel(l1, l2)) return false;
32      p.x = - (l1.c * l2.b - l1.b * l2.c) / (l1.a * l2.b - l1.b * l2.a);
33      if (fabs(l1.b) > EPS) p.y = - (l1.c + l1.a * p.x);
34      else p.y = - (l2.c + l2.a * p.x);
35      return 1;
36  }
37  double distToLine(point p, point a, point b, point& c) {
38      double t = dot(p - a, b - a) / norm(b - a);
39      c = a + (b - a) * t;
40      return abs(c - p);
41  }
42  double distToSegment(point p, point a, point b, point& c) {
43      double t = dot(p - a, b - a) / norm(b - a);
44      if (t > 1.0)
45          c = point(b.x, b.y);
46      else if (t < 0.0)
47          c = point(a.x, a.y);
48      else
49          c = a + (b - a) * t;
50      return abs(c - p);
51  }
52  bool intersectTwoSegment(point a, point b, point c, point d) {
53      ftype ABxAC = cross(b - a, c - a);
54      ftype ABxAD = cross(b - a, d - a);
55      ftype CDxCA = cross(d - c, a - c);
56      ftype CDxCB = cross(d - c, b - c);
57      if (ABxAC == 0 || ABxAD == 0 || CDxCA == 0 || CDxCB == 0) {
58          if (ABxAC == 0 && dot(a - c, b - c) <= 0) return true;
59          if (ABxAD == 0 && dot(a - d, b - d) <= 0) return true;
60          if (CDxCA == 0 && dot(c - a, d - a) <= 0) return true;
61          if (CDxCB == 0 && dot(c - b, d - b) <= 0) return true;
62          return false;
63      }
64      return (ABxAC * ABxAD < 0 && CDxCA * CDxCB < 0);
65  }
66  void perpendicular(line l1, point p, line& l2) {
67      if (fabs(l1.a) < EPS)
68          l2 = {1.0, 0.0, -p.x};
69      else {
70          l2.a = -l1.b / l1.a;
71          l2.b = 1.0;
72          l2.c = -l2.a * p.x - l2.b * p.y;
73      }
74  }
```

7.1.3   Circle

```cpp
1   int insideCircle(const point& p, const point& center, ftype r) {
2       ftype d = norm(p - center);
3       ftype rSq = r * r;
4       return fabs(d - rSq) < EPS ? 0 : (d - rSq >= EPS ? 1 : -1);
5   }
6   bool circle2PointsR(const point& p1, const point& p2, ftype r, point& c) {
7       double h = r * r - norm(p1 - p2) / 4.0;
8       if (fabs(h) < 0) return false;
9       h = sqrt(h);
10      point perp = (p2 - p1).rotate(PI / 2.0);
11      point m = (p1 + p2) / 2.0;
12      c = m + perp * (h / abs(perp));
13      return true;
14  }
```

7.1.4   Triangle

```cpp
1   double areaTriangle(double ab, double bc, double ca) {
2       double p = (ab + bc + ca) / 2;
3       return sqrt(p) * sqrt(p - ab) * sqrt(p - bc) * sqrt(p - ca);
4   }
5   double rInCircle(double ab, double bc, double ca) {
6       double p = (ab + bc + ca) / 2;
7       return areaTriangle(ab, bc, ca) / p;
8   }
9   double rInCircle(point a, point b, point c) {
10      return rInCircle(abs(a - b), abs(b - c), abs(c - a));
11  }
12  bool inCircle(point p1, point p2, point p3, point &ctr, double &r) {
13      r = rInCircle(p1, p2, p3);
14      if (fabs(r) < EPS) return false;
15      line l1, l2;
16      double ratio = abs(p2 - p1) / abs(p3 - p1);
17      point p = p2 + (p3 - p2) * (ratio / (1 + ratio));
18      pointsToLine(p1, p, l1);
19      ratio = abs(p1 - p2) / abs(p2 - p3);
20      p = p1 + (p3 - p1) * ( ratio / (1 + ratio));
21      pointsToLine(p2, p, l2);
```

```cpp
22        areIntersect(l1, l2, ctr);
23        return true;
24 }
25 double rCircumCircle(double ab, double bc, double ca) {
26        return ab * bc * ca / (4.0 * areaTriangle(ab, bc, ca));
27 }
28 double rCircumCircle(point a, point b, point c) {
29        return rCircumCircle(abs(b - a), abs(c - b), abs(a - c));
30 }
```

7.1.5   Convex hull

```cpp
1 vector<point> CH_Andrew(vector<point> &Pts) { // overall O(n log n)
2        int n = Pts.size(), k = 0;
3        vector<point> H(2 * n);
4        sort(Pts.begin(), Pts.end());
5        for (int i = 0; i < n; ++i) {
6            while ((k >= 2) && !ccw(H[k - 2], H[k - 1], Pts[i])) --k;
7            H[k++] = Pts[i];
8        }
9        for (int i = n - 2, t = k + 1; i >= 0; --i) {
10           while ((k >= t) && !ccw(H[k - 2], H[k - 1], Pts[i])) --k;
11           H[k++] = Pts[i];
12       }
13       H.resize(k);
14       return H;
15 }
```

7.1.6   Polygon

```cpp
1 double perimeter(const vector<point> &P) {
2        double ans = 0.0;
3        for (int i = 0; i < (int)P.size() - 1; ++i)
4            ans += abs(P[i] - P[i + 1]);
5        return ans;
6 }
7 double area(const vector<point> &P) {
8        double ans = 0.0;
9        for (int i = 0; i < (int)P.size() - 1; ++i)
10           ans += (P[i].x * P[i + 1].y - P[i + 1].x * P[i].y);
11       return fabs(ans) / 2.0;
12 }
13 bool isConvex(const vector<point> &P) {
14       int n = (int)P.size();
15       if (n <= 3) return false;
16       bool firstTurn = ccw(P[0], P[1], P[2]);
17       for (int i = 1; i < n - 1; ++i)
18           if (ccw(P[i], P[i + 1], P[(i + 2) == n ? 1 : i + 2]) != firstTurn)
19               return false;
20       return true;
21 }
22 int insidePolygon(point pt, const vector<point> &P) {
23       int n = (int)P.size();
24       if (n <= 3) return -1;
```

```cpp
25       bool on_polygon = false;
26       for (int i = 0; i < n - 1; ++i)
27           if (fabs(abs(P[i] - pt) + abs(pt - P[i + 1]) - abs(P[i] - P[i + 1])
28   ) < EPS)
29               on_polygon = true;
30       if (on_polygon) return 0;
31       double sum = 0.0;
32       for (int i = 0; i < n - 1; ++i) {
33           if (ccw(pt, P[i], P[i + 1]))
34               sum += angle(P[i] - pt, P[i + 1] - pt);
35           else
36               sum -= angle(P[i] - pt, P[i + 1] - pt);
37       }
38       return fabs(sum) > PI ? 1 : -1;
39 }
```

7.2   Minimum enclosing circle

```cpp
1 /**
2  * Description: computes the minimum circle that encloses all the given
3     points.
4  */
5 double abs(point a) { return sqrt(a.X * a.X + a.Y * a.Y); }
6
7 point center_from(double bx, double by, double cx, double cy) {
8        double B = bx * bx + by * by, C = cx * cx + cy * cy, D = bx * cy - by *
9         cx;
10       return point((cy * B - by * C) / (2 * D), (bx * C - cx * B) / (2 * D));
11 }
12
13 circle circle_from(point A, point B, point C) {
14       point I = center_from(B.X - A.X, B.Y - A.Y, C.X - A.X, C.Y - A.Y);
15       return circle(I + A, abs(I));
16 }
17
18 const int N = 100005;
19 int n, x[N], y[N];
20 point a[N];
21
22 circle emo_welzl(int n, vector<point> T) {
23       if (T.size() == 3 || n == 0) {
24           if (T.size() == 0) return circle(point(0, 0), -1);
25           if (T.size() == 1) return circle(T[0], 0);
26           if (T.size() == 2) return circle((T[0] + T[1]) / 2, abs(T[0] - T
27   [1]) / 2);
28           return circle_from(T[0], T[1], T[2]);
29       }
30       random_shuffle(a + 1, a + n + 1);
31       circle Result = emo_welzl(0, T);
32       for (int i = 1; i <= n; i++)
33           if (abs(Result.X - a[i]) > Result.Y + 1e-9) {
34               T.push_back(a[i]);
35               Result = emo_welzl(i - 1, T);
```

```
33              T.pop_back();
34          }
35      return Result;
36 }
```

8   Graph

8.1   K-th smallest shortest path

```
1 /** Finding the k-th smallest shortest path from vertex s to vertex t,
2  *  each vertex can be visited more than once.
3  */
4 using adj_list = vector<vector<pair<int, int>>>;
5 vector<int> k_smallest(const adj_list &g, int k, int s, int t) {
6     int n = (int) g.size();
7     vector<long long> ans;
8     vector<int> cnt(n);
9     using pli = pair<long long, int>;
10    priority_queue<pli, vector<pli>, greater<pli>> pq;
11    pq.emplace(0, s);
12    while (!pq.empty() && cnt[t] < k) {
13        int u = pq.top().second;
14        long long d = pq.top().first;
15        pq.pop();
16        if (cnt[u] == k) continue;
17        cnt[u]++;
18        if (u == t) {
19            ans.push_back(d);
20        }
21        for (auto [v, cost] : g[u]) {
22            pq.emplace(d + cost, v);
23        }
24    }
25    assert(ans.size() == k);
26    return ans;
27 }
```

8.2   Eulerian path

8.2.1   Directed graph

```
1 /**
2  * Hierholzer's algorithm.
3  * Description: An Eulerian path in a directed graph is a path that visits
       all edges exactly once.
4  *  An Eulerian cycle is a Eulerian path that is a cycle.
5  * Time complexity: O(|E|).
6  */
7 vector<int> find_path_directed(const vector<vector<int>> &g, int s) {
8     int n = (int) g.size();
9     vector<int> stack, cur_edge(n), vertices;
10    stack.push_back(s);
11    while (!stack.empty()) {
12        int u = stack.back();
```

```
13        stack.pop_back();
14        while (cur_edge[u] < (int) g[u].size()) {
15            stack.push_back(u);
16            u = g[u][cur_edge[u]++];
17        }
18        vertices.push_back(u);
19    }
20    reverse(vertices.begin(), vertices.end());
21    return vertices;
22 }
```

8.2.2   Undirected graph

```
1 /**
2  * Hierholzer's algorithm.
3  * Description: An Eulerian path in a undirected graph is a path that
       visits all edges exactly once.
4  *  An Eulerian cycle is a Eulerian path that is a cycle.
5  * Time complexity: O(|E|).
6  */
7 struct Edge {
8     int to;
9     list<Edge>::iterator reverse_edge;
10    Edge(int _to) : to(_to) {}
11 };
12 vector<int> vertices;
13 void find_path(vector<list<Edge>> &g, int u) {
14    while (!g[u].empty()) {
15        int v = g[u].front().to;
16        g[v].erase(g[u].front().reverse_edge);
17        g[u].pop_front();
18        find_path(g, v);
19    }
20    vertices.emplace_back(u); // reversion list.
21 }
22 void add_edge(int u, int v) {
23    g[u].emplace_front(v);
24    g[v].emplace_front(u);
25    g[u].front().reverse_edge = g[v].begin();
26    g[v].front().reverse_edge = g[u].begin();
27 }
```

9   Misc.

9.1   Ternary search

```
1 const double eps = 1e-9;
2 double ternary_search_max(double l, double r) {
3     // find x0 such that: f(x0) > f(x), \all x: l <= x <= r.
4     while (r - l > eps) {
5         double mid1 = l + (r - l) / 3;
6         double mid2 = r - (r - l) / 3;
7         if (f(mid1) < f(mid2)) l = mid1;
8         else r = mid2;
```

```
 9        }
10        return l;
11 }
12 double ternary_search_min(double l, double r) {
13        // find x0 such that: f(x0) < f(x), \all x: l <= x <= r.
14        while (r - l > eps) {
15            double mid1 = l + (r - l) / 3;
16            double mid2 = r - (r - l) / 3;
17            if (f(mid1) > f(mid2)) l = mid1;
18            else r = mid2;
19        }
20        return l;
21 }
```

9.2   Dutch flag national problem

```
 1 void dutch_flag_national(vector<int> &arr) {
 2        // All elements that are LESS than pivot are moved to the LEFT.
 3        // All elements that are GREATER than pivot are moved to the RIGHT.
 4        // E.g. [1, 2, 0, 0, 2, 2, 1], pivot = 1 -> [0, 0, 1, 1, 2, 2, 2].
 5        int n = (int) arr.size();
 6        int i = 0, j = 0, k = n - 1;
 7        int pivot = 1;
 8        // 0....i....j....k....n
 9        while (j <= k) {
10            if (arr[j] < pivot) {
11                swap(arr[i], arr[j]);
12                i++;
13                j++;
14            }
15            else if (arr[j] > pivot) {
16                swap(arr[j], arr[k]);
17                k--;
18            }
19            else {
20                j++;
21            }
22        }
23        // 0 <= index <= i - 1: arr[index] < mid.
24        // i <= index <= k: arr[index] = mid.
25        // k + 1 <= index < sz: arr[index] > mid.
26 }
```

9.3   Matrix

```
 1 struct Matrix {
 2        static const matrix_type INF = numeric_limits<matrix_type>::max();
 3        int N, M;
 4        vector<vector<matrix_type>> mat;
 5
 6        Matrix(int _N, int _M, matrix_type v = 0) : N(_N), M(_M) {
 7            mat.assign(N, vector<matrix_type>(M, v));
 8        }
```

```
 9        static Matrix identity(int n) { // return identity matrix.
10            Matrix I(n, n);
11            for (int i = 0; i < n; ++i) {
12                I[i][i] = 1;
13            }
14            return I;
15        }
16
17        vector<matrix_type>& operator[](int r) { return mat[r]; }
18        const vector<matrix_type>& operator[](int r) const { return mat[r]; }
19
20        Matrix& operator*=(const Matrix &other) {
21            assert(M == other.N); // [N x M] [other.N x other.M]
22            Matrix res(N, other.M);
23            for (int r = 0; r < N; ++r) {
24                for (int c = 0; c < other.M; ++c) {
25                    long long square_mod = (long long) MOD * MOD;
26                    long long sum = 0;
27                    for (int g = 0; g < M; ++g) {
28                        sum += (long long) mat[r][g] * other[g][c];
29                        if (sum >= square_mod) sum -= square_mod;
30                    }
31                    res[r][c] = sum % MOD;
32                }
33            }
34            mat.swap(res.mat); return *this;
35        }
36 };
```