# ACM-ICPC Team Reference Document
# Tula State University (Fursov, Perezyabov, Vasin)

## Contents



# 1 Templates

## 1.1 C++ Template

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

using namespace std;


// defines
#define precision(x) cout << fixed << setprecision(x);
#define fast cin.tie(0); ios::sync_with_stdio(0)
#define all(x) x.begin(), x.end()
#define rall(x) x.rbegin(), x.rend()
#define ff first
#define ss second
// #define nl endl
#define nl "\n"
#define sp " "
#define yes "Yes"
#define no "No"
#define int long long
//////////////////////////////////////////


// constants
const int inf = 1e18;

// const int mod = 1e9 + 7;
// const int pmod = 1e9 + 6;

// const int mod = 998244353;
// const int pmod = 998244352;
// const int root = 3;
//////////////////////////////////////////

// iofstreams
ifstream in("input.txt");
ofstream out("output.txt");
//////////////////////////////////////////

// random generator
random_device rdev;
mt19937 reng(rdev());
uniform_int_distribution<mt19937::result_type> dist(1, 1e9
     + 7);
//////////////////////////////////////////

// globals

//////////////////////////////////////////

// solution
void solve() {

}


// preprocessing
void prepr() { }


// etrance
signed main() {
```

```
    precision(15);
    fast;
    prepr();
    int t = 1;
    //cin >> t;
    while (t--) solve();
}

// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

// Мыповзрослелиимытеперьпосебесами
// Превратилисьвзатылкипередчужимиглазами
// Где- тотамдалеко, застекломизольда
// Дрогнулипровода, пронеслисьпоезда
// Ибольшенетникогодома
// Сорнякинаасфальте, недостроенныеновостройки
// Зазаросшейжелезнойдорогойгаражиипомойки
// Тамтеперьновыймоствновыевремена
// Поднимммыоставимсвоиимена
// Рядомсматернымсловом
// Нассдетстванаучилавыживатьинесдаваться
// Огромнаяипротиеворечиваястрана
// Таксложнонайтиместовней, такпростопотеряться
// Зимапогорло, всердцевесна
// Связанныеэтойпрочнойневидимойниткой
// Всемхолоднымветрамивсемувовселеннойоткрытки
// Иззабытыхстихов, изразбитыхвитрин
// Изсжатыхпружининатянутыхжил
// Сломаны, сломаны, сломаны,
// Нонепобедимы
// Нассдетстванаучилавыживатьинесдаваться
// Огромнаяипротиворечиваястрана
// Таксложнонайтиместовней, такпростопотеряться
// Зимапогорло, всердцевесна
// Нассдетстванаучилавыживатьинесдаваться
// Огромнаяипротиворечиваястрана
// Таксложнонайтиместовней, такпростопотеряться
// Зимапогорло, всердцевесна
// Нассдетстванаучилавыживатьинесдаваться
// Огромнаяипротиворечиваястрана
// Таксложнонайтиместовней, такпростопотеряться
```

## 1.2 C++ Include

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <random>
#include <cmath>
#include <algorithm>
#include <string>
#include <vector>
#include <set>
#include <unordered_set>
#include <map>
#include <unordered_map>
#include <queue>
#include <deque>
#include <stack>
#include <list>
#include <bitset>
```

## 1.3 Py Template

```
from math import sqrt, ceil, floor, gcd
from random import randint
import sys


def inpt():
    return sys.stdin.readline().strip()


input = inpt


inf = 1e18;

# mod = 1e9 + 7
# pmod = 1e9 + 6

# mod = 998244353
# pmod = 998244352
# root = 3


def solve():
    pass


t = 1
# t = int(input())
for _ in range(t):
    solve()
```

# 2 Data Structures

## 2.1 Disjoint Set Union

```
// Theme: Disjoint Set Union

struct dsu {
    vector<int> p, size;

    dsu(int n) {
        p.assign(n, 0); size.assign(n, 0);
        for (int i = 0; i < n; i++) {
            p[i] = i;
            size[i] = 1;
        }
    }

    int get(int v) {
        if (p[v] != v) p[v] = get(p[v]);
        return p[v];
    }

    void unite(int u, int v) {
        auto x = get(u), y = get(v);
        if (x == y) return;
        if (size[x] > size[y]) swap(x, y);
        p[x] = y; size[y] += size[x];
    }
};
```

## 2.2 Segment Tree

```
// Theme: Segment Tree

struct segtree {
   int size;
   vector<int> tree;

   void init(int n) {
      size = 1;
      while (size < n) size <<= 1;
      tree.assign(2 * size - 1, 0);
   }

   void build(vector<int> &a, int x, int lx, int rx) {
      if (rx - lx == 1) {
         if (lx < a.size()) tree[x] = a[lx];
         return;
      }
      int m = (lx + rx) / 2;
      build(a, 2 * x + 1, lx, m);
      build(a, 2 * x + 2, m, rx);
      tree[x] = tree[2 * x + 1] + tree[2 * x + 2];
   }
   void build(vector<int> &a) {
      init(a.size());
      build(a, 0, 0, size);
   }

   // O(log(n))
   void set(int i, int v, int x, int lx, int rx) {
      if (rx - lx == 1) {
         tree[x] = v;
         return;
      }
      int m = (lx + rx) / 2;
      if (i < m) set(i, v, 2 * x + 1, lx, m);
      else set(i, v, 2 * x + 2, m, rx);
      tree[x] = tree[2 * x + 1] + tree[2 * x + 2];
   }
   void set(int i, int v) {
      set(i, v, 0, 0, size);
   }

   // O(log(n))
   int sum(int l, int r, int x, int lx, int rx) {
      if (l <= lx && rx <= r) return tree[x];
      if (l >= rx || r <= lx) return 0;
      int m = (lx + rx) / 2;
      return sum(l, r, 2 * x + 1, lx, m) +
         sum(l, r, 2 * x + 2, m, rx);
   }
   int sum(int l, int r) {
      return sum(l, r, 0, 0, size);
   }
};
```

## 2.3 Segment Tree Propagate

```
// Theme: Segment Tree With Propagation

struct segtree_prop {
   int size;
   vector<int> tree;

   void init(int n) {
      size = 1;
      while (size < n) size <<= 1;
      tree.assign(2 * size - 1, 0);
   }

   void build(vector<int> &a, int x, int lx, int rx) {
      if (rx - lx == 1) {
         if (lx < a.size()) tree[x] = a[lx];
         return;
      }
      int m = (lx + rx) / 2;
      build(a, 2 * x + 1, lx, m);
      build(a, 2 * x + 2, m, rx);
      tree[x] = tree[2 * x + 1] + tree[2 * x + 2];
   }
   void build(vector<int> &a) {
      init(a.size());
      build(a, 0, 0, size);
   }

   void push(int x, int lx, int rx) {
      if (rx - lx == 1) return;
      tree[2 * x + 1] += tree[x];
      tree[2 * x + 2] += tree[x];
      tree[x] = 0;
   }

   // O(log(n))
   void add(int v, int l, int r, int x, int lx, int rx) {
      push(x, lx, rx);
      if (rx <= l || r <= lx) return;
      if (l <= lx && rx <= r) {
         tree[x] += v;
         return;
      }
      int m = (lx + rx) / 2;
      add(v, l, r, 2 * x + 1, lx, m);
      add(v, l, r, 2 * x + 2, m, rx);
   }
   void add(int v, int l, int r) {
      add(v, l, r, 0, 0, size);
   }

   // O(log(n))
   int get(int i, int x, int lx, int rx) {
      push(x, lx, rx);
      if (rx - lx == 1) return tree[x];
      int m = (lx + rx) / 2;
      if (i < m) return get(i, 2 * x + 1, lx, m);
      else return get(i, 2 * x + 2, m, rx);
   }
   int get(int i) {
      return get(i, 0, 0, size);
   }
};
```

# 3 Algebra

## 3.1 Primes Sieve

```
// Theme: Prime Numbers

// Alrotihm: Eratosthenes' Sieve
// Complexity: O(N*log(log(N)))

auto get_sieve(int n) {
   vector<int> sieve(n); // Sieve, 0 - Prime, Another -
       Lowest Prime Divisor
   sieve[0] = sieve[1] = 1;

   for (int i = 2; i * i <= n; i++)
      if (!sieve[i])
         for (int j = i * i; j < n; j += i)
            sieve[j] = i;

   return sieve;
}

// Alrotihm: Prime Numbers Wirh Sieve
// Complexity: O(N*log(log(N)))

auto get_primes(int n) {
   vector<int> primes, sieve = get_sieve(n);

   for (int i = 2; i < sieve.size(); i++)
      if (!sieve[i])
         primes.push_back(i);

   return primes;
}

// Alrotihm: Linear Eratosthenes' Sieve
// Complexity: O(N)

auto get_sieve_primes(int n, vector<int> &primes) {
   vector<int> sieve(n);
   sieve[0] = sieve[1] = 1;

   for (int i = 2; i <= n; i++) {
      if (!sieve[i]) {
         sieve[i] = i;
         primes.push_back(i);
      }
      for (int j = 0; j < primes.size() && primes[j] <=
          sieve[i] && i * primes[j] < n; j++)
```

```
         sieve[i * primes[j]] = primes[j];
   }

   return sieve;
}
```

## 3.2 Factorization

```
// Theme: Factorization

// Alrotihm: Trivial Algorithm
// Complexity: O(sqrt(N))

auto factors(int n) {
   vector<int> factors;

   for (int i = 2; i * i <= n; i++) {
      if (n % i) continue;
      while (n % i == 0) n /= i;
      factors.push_back(i);
   }

   if (n != 1) factors.push_back(n);

   return factors;
}

// Alrotihm: Factorization With Sieve
// Complexity: O(N*log(log(N)))

auto factors_sieve(int n) {
   vector<int> factors, sieve = get_sieve(n + 1);

   while (sieve[n]) {
      factors.push_back(sieve[n]);
      n /= sieve[n];
   }

   if (n != 1) factors.push_back(n);

   return factors;
}

// Alrotihm: Factorization With Primes
// Complexity: O(sqrt(N)/log(qsrt(N)))

auto factors_primes(int n) {
   vector<int> factors, primes = get_primes(n + 1);

   for (auto &i : primes) {
      if (i * i > n) break;
      if (n % i) continue;
      while (n % i == 0) n /= i;
      factors.push_back(i);
   }

   if (n != 1) factors.push_back(n);

   return factors;
}

// Alrotihm: Ferma's Test
// Complexity: O(K*log(N))

bool ferma(int n) {
   if (n == 2) return true;

   uniform_int_distribution<int> distA(2, n - 1);

   for (int i = 0; i < 1000; i++) {
      int a = distA(reng);
      if (gcd(a, n) != 1 ||
      binpow(a, n - 1, n) != 1)
         return false;
   }

   return true;
}

// Alrotihm: Pollard's Rho Algorithm
// Complexity: O(N^(1/4))

int f(int x, int c, int n) {
   return ((x * x) % n + c) % n;
}

int pollard_rho(int n) {
   if (n % 2 == 0) return 2;

   uniform_int_distribution<int> distC(1, n), distX(1, n);

   int c = distC(reng), x = distX(reng);
   int y = x;

   int g = 1;

   while (g == 1) {
      x = f(x, c, n);
      y = f(f(y, c, n), c, n);
      g = gcd(abs(x - y), n);
   }

   return g;
}

// Alrotihm: Factorization With Pollard's Rho And Ferma's
     Test
// Complexity: O(N^(1/4)*log(N))

void factors_pollard_rho(int n, vector<int> &factors) {
   if (n == 1) return;

   if (ferma(n)) {
      factors.push_back(n);
      return;
   }

   int d = pollard_rho(n);

   factors_pollard_rho(d, factors);
   factors_pollard_rho(n / d, factors);
}
```

## 3.3 Euler Totient Function

```
// Theme: Euler's Totient Function
// Alrotihm: Euler's Product Formula
// Complexity: O(sqrt(N))
// Idea:
// phi = n(1 - 1 / pi), i = 1,...

int phi(int n) {
   if (n == 1) return 1;

   auto f = factors(n);

   int res = n;
   for (auto &p : f)
      res -= res / p;

   return res;
}
```

## 3.4 Greatest Common Divisor

```
// Theme: Greatest Common Divisor

// Alrotihm: Simple Euclidean Algorithm
// Complexity: O(log(N))

int gcd(int a, int b) {
   while (a && b)
      a > b ? a %= b : b %= a;
   return a + b;
}

// Alrotihm: Extended Euclidean Algorithm
// Complexity: O(log(N))
// Idea
// d = gcd(a, b)
// x * a + y * b = d
// returns {d, x, y}

vector<int> euclid(int a, int b) {
   if (!a) return { b, 0, 1 };
   auto v = euclid(b % a, a);
   int d = v[0], x = v[1], y = v[2];
```

```
    return { d, y - (b / a) * x, x };
}
```

## 3.5 Binary Operations

```
// Theme: Binary Operations

// Alrotihm: Binary Multiplication
// Complexity: O(log(b))

int binmul(int a, int b, int p = 0) {
    int res = 0;
    while (b) {
        if (b & 1) res = p ? (res + a) % p : (res + a);
        a = p ? (a + a) % p : (a + a);
        b >>= 1;
    }
    return res;
}

// Alrotihm: Binary Exponentiation
// Complexity: O(log(b))

int binpow(int a, int b, int p = 0) {
    int res = 1;
    while (b) {
        if (b & 1) res = p ? (res * a) % p : (res * a);
        a = p ? (a * a) % p : (a * a);
        b >>= 1;
    }
    return res;
}
```

## 3.6 Matrices

```
// Theme: Matrix Opeations

template <typename T>
using row = vector<T>;
template <typename T>
using matrix = vector<vector<T>>;

// Alrotihm: Matrix-Matrix Multiplication
// Complexity: O(N*K*M)

auto matrmul(matrix<int> &a, matrix<int> &b, int p) {
    int n = a.size(), k = a[0].size(), m = b[0].size();

    matrix<int> res(n, row<int>(m));

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            for (int z = 0; z < k; z++)
                res[i][j] = p ? (res[i][j] + a[i][z] * b[z][j]
                        % p) % p : (res[i][j] + a[i][z] * b[z][
                        j]);

    return res;
}

// Alrotihm: Matrix-Vector Multiplication
// Complexity: O(N*M)

auto matrmul(matrix<int> &a, row<int> &b, int p) {
    int n = a.size(), m = b.size();

    row<int> res(n);

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            res[i] = p ? (res[i] + a[i][j] * b[j] % p) % p :
                (res[i] + a[i][j] * b[j]);

    return res;
}

// Alrotihm: Fast Matrix Exponentiation
// Complexity: O(N^3*log(N))

auto matrbinpow(matrix<int> a, int x, int p = 0) {
    int n = a.size();
    matrix<int> res(n, row<int>(n));
    for (int i = 0; i < n; i++) res[i][i] = 1;

    while (n) {
        if (n & 1) res = matrmul(res, a, p);
        a = matrmul(a, a, p);
        n >>= 1;
    }

    return res;
}
```

## 3.7 Fibonacci

```
// Theme: Fibonacci Sequence
// Alrotihm: Fibonacci Numbers With Matrix Exponentiation
// Complexity: O(log(N))

int fibonacci(int n) {
    row<int> first_three = { 0, 0, 1 };
    if (n <= 3) return first_three[n - 1];

    matrix<int> fib(2, row<int>(2, 0));
    fib[0][0] = 0; fib[0][1] = 1;
    fib[1][0] = 1; fib[1][1] = 1;

    row<int> last_two = { first_three[1], first_three[2] };

    fib = m_binpow(fib, n - 3);

    last_two = m_prod(fib, last_two);

    return last_two[1];
}
```

## 3.8 Baby Step Giant Step

```
// Theme: Descrete Logarithm
// Alrotihm: Baby-Step Giant-Step Algorithm
// Complexity: O(sqrt(p)*log(p))
// Idea:
// a ^ (x) = b (mod p), (a, p) = 1
// a ^ (i * m + j) = b (mod p), m = ceil(sqrt(p))
// a ^ (i * m) = b * a ^ (-j) (mod p)

int baby_giant_step(int a, int b, int p) {
    int m = ceil(sqrt(p)), _a = binpow(a, p - 2, p); // a ^
        (-1) = a ^ (p - 2) mod p

    unordered_map<int, int> s;
    for (int j = 0, t = b; j < m; j++, t = t * _a % p) s[t]
        = j; // s[b * a ^ (-j)] = j

    for (int i = 0; i < m; i++) {
        auto f = s.find(binpow(a, i * m, p)); // s.find(a ^
            (i * m))
        if (f != s.end()) return i * m + f->ss; // i * m + j
    }

    return -1;
}
```

## 3.9 Combinations

```
// Theme: Combination Number

// Alrotihm: Online Multiplication-Division
// Complexity: O(k)

int C(int n, int k) { // C_n^k - from n by k
    int res = 1;

    for (int i = 1; i <= k; i++) {
        res *= n - k + i;
        res /= i;
    }

    return res;
}
```

```
// Alrotihm: Pascal Triangle Preprocessing
// Complexity: O(N^2)

auto pascal(int n) {
   vector<vector<int>> C(n + 1, vector<int>(n + 1, 1)); //
        C[i][j] = C_i+j^i
   for (int i = 1; i < n + 1; i++)
      for (int j = 1; j < n + 1; j++)
         C[i][j] = C[i - 1][j] + C[i][j - 1];

   return C;
}
```

## 3.10 Permutation

```
// Theme: Permmutations
// Alrotihm: Next Lexicological Permutation
// Complexity: O(N)

bool perm(vector<int> &v) {
   int n = v.size();

   for (int i = n - 1; i >= 1; i--) {
      if (v[i - 1] < v[i]) {
         reverse(v.begin() + i, v.end());
         int j = distance(v.begin(), upper_bound(v.begin()
              + i, v.end(), v[i - 1]));
         swap(v[i - 1], v[j]);
         return true;
      }
   }

   return false;
}
```

## 3.11 Fast Fourier Transform

```
// Theme: Discrete Fourier Transform
// Alrotihm: Fast Fourier Transform
// Complexity: O(N*log(N))

const int mod = 7340033; // Module (7 * (2 ^ 20) + 1)
const int proot = 5; // Primary Root (5 ^ (2 ^ 20) == 1 mod
      7340033)
const int proot_1 = 4404020; // Inverse Primary Root (5 *
     4404020 == 1 mod 7340033)
const int pw = 1 << 20; // Maximum Degree Of Two (2 ^ 20)

// const int mod = 998244353; // Module (7 * 17 * (2 ^ 23)
     + 1)
// const int proot = 31; // Primary Root (31 ^ (2 ^ 23) ==
     1 mod 998244353)
// const int proot_1 = 128805723; // Inverse Primary Root
     (31 * 128805723 == 1 mod 998244353)
// const int pw = 1 << 23; // Maximum Degree Of Two (2 ^
     23)

auto fft(vector<int> &a, bool invert = 0) {
   int n = a.size(); // n = 2 ^ x

   for (int i = 1, j = 0; i < n; i++) { // Bit-Reversal
        Permutation (0000, 1000, 0100, 1100, 0010, ...)
      int bit = n >> 1;
      for (; j >= bit; bit >>= 1) j -= bit;
      j += bit;
      if (i < j) swap(a[i], a[j]);
   }

   for (int len = 2; len <= n; len <<= 1) {
      int lroot = invert ? proot_1 : proot; // Prmary Root
            Or Inverse Root (Inverse Transform)

      for (int i = len; i < pw; i <<= 1)
         lroot = (lroot * lroot) % mod; // Current Primary
              Root

      for (int i = 0; i < n; i += len) {
         int root = 1;
         for (int j = 0; j < len / 2; j++) {
            int u = a[i + j], v = a[i + j + len / 2] *
                 root % mod;
               a[i + j] = (u + v) % mod;
               a[i + j + len / 2] = (u - v + mod) % mod;
               root = (root * lroot) % mod;
            }

         }
      }

   if (invert) {
      int _n = 1;
      for (int i = 1; i <= mod - 2; i++) _n = (_n * n) %
          mod;
      for (int i = 0; i < n; i++) a[i] = (a[i] * _n) % mod
          ;
   }
}
```

## 3.12 Number Decomposition

```
// Theme: Integer Numbers Decomposition With Composite
    Module

int m; // Module
vector<int> p; // Prime Divisors Of Module
                  // m = (p1 ^ m1) * (p2 ^ m2) * ... * (
                       pn ^ mn)

struct num {
   int x; // GCD(x, m) = 1
   vector<int> a; // Powers Of Primes
                  // n = (p1 ^ a1) * (p2 ^ a2) * ... * (
                       pn ^ an) * x

   num() : x(0), a(vector<int>(p.size())) { }

   num(int n) : x(0), a(vector<int>(p.size())) {
      if (!n) return;
      for (int i = 0; i < p.size(); i++) {
         int ai = 0;
         while (n % p[i] == 0) {
            n /= p[i];
            ai++;
         }
         a[i] = ai;
      }
      x = n;
   }

   num operator*(const num &nm) {
      vector<int> new_a(p.size());
      for (int i = 0; i < p.size(); i++)
         new_a[i] = a[i] + nm.a[i];
      num res; res.a = new_a;
      res.x = x * nm.x % m;
      return res;
   }

   num operator/(const num &nm) {
      vector<int> new_a(p.size());
      for (int i = 0; i < p.size(); i++)
         new_a[i] = a[i] - nm.a[i];
      num res; res.a = new_a;
      int g = euclid(nm.x, m)[1];
      g += m; g %= m;
      res.x = x * g % m;
      return res;
   }

   int toint() {
      int res = x;
      for (int i = 0; i < p.size(); i++)
         res = res * binpow(p[i], a[i], m) % m;
      return res;
   }
};
```

## 3.13 Formulae

### Combinations.

$C_n^k = \frac{n!}{(n-k)!k!}$

$C_n^0 + C_n^1 + ... + C_n^n = 2^n$
$C_{n+1}^{k+1} = C_n^{k+1} + C_n^k$
$C_n^k = \frac{n}{k} C_{n-1}^{k-1}$

**Striling approximation.**
$n! \approx \sqrt{2\pi n} \frac{n}{e}^n$

**Euler's theorem.**
$a^{\phi(m)} \equiv 1 \bmod m$, $gcd(a, m) = 1$

**Ferma's little theorem.**
$a^{p-1} \equiv 1 \bmod p$, $gcd(a, p) = 1$, $p$ - prime.

**Catalan number.**

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

**Arithmetic progression.**
$S_n = \frac{a_1+a_n}{2} n = \frac{2a_1+d(n-1)}{2} n$

**Geometric progression.**
$S_n = \frac{b_1(1-q^n)}{1-q} n$

**Infinitely decreasing geometric progression.**
$S_n = \frac{b_1}{1-q} n$

**Sums.**
$\sum_{i=1}^n i = \frac{n(n+1)}{2}$,
$\sum_{i=1}^n i^2 = \frac{n(2n+1)(n+1)}{6}$,
$\sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$,
$\sum_{i=1}^n i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$,
$\sum_{i=a}^b c^i = \frac{c^{b+1}-c^a}{c-1}, c \neq 1.$

# 4 Geometry

## 4.1 Vector

```
// Theme: Methematical 3-D Vector

template <typename T>
struct vec {
   T x, y, z;
   vec(T x = 0, T y = 0, T z = 0) : x(x), y(y), z(z) { }
   vec<T> operator+(const vec<T> &v) const { return vec<T>(
        x + v.x, y + v.y, z + v.z); }
   vec<T>operator-(const vec<T> &v) const { return vec<T>(x
        - v.x, y - v.y, z - v.z); }
   vec<T>operator*(T k) const { return vec<T>(k * x, k * y,
        k * z); }
   friend vec<T> operator*(T k, const vec<T> &v) { return
        vec<T>(v.x * k, v.y * k, v.z * k); }
   vec<T> operator/(T k) { return vec<T>(x / k, y / k, z /
        k); }
   T operator*(const vec<T> &v) const { return x * v.x + y
        * v.y + z * v.z; }
   vec<T> operator^(const vec<T> &v) const { return { y * v
        .z - z * v.y, z * v.x - x * v.z, x * v.y - y * v.x
        }; }
   auto operator<=>(const vec<T> &v) const = default;
   bool operator==(const vec<T> &v) const = default;
   T norm() const { return x * x + y * y + z * z; }
   double abs() const { return sqrt(norm()); }
   double cos(const vec<T> &v) const { return ((*this) * v)
        / (abs() * v.abs()); }
   friend ostream &operator<<(ostream &out, const vec<T> &v
        ) { return out << v.x << sp << v.y << sp << v.z; }
   friend istream &operator>>(istream &in, vec<T> &v) {
        return in >> v.x >> v.y >> v.z; }
};
```

## 4.2 Planimetry

```
// Theme: Planimetry Objects

// Subtheme: Point

template <typename T>
struct point {
   T x, y;

   point() : x(0), y(0) { }
   point(T x, T y) : x(x), y(y) { }
};

// Subtheme: Rectangle

template <typename T>
struct rectangle {
   point<T> ld, ru;

   rectangle(const point<T> &ld, const point<T> &ru) : ld(
        ld), ru(ru) { }
};
```

## 4.3 Graham

```
// Theme: Convex Hull
// Alrotihm: Graham's Algorithm
// Complexity: O(N*log(N))

auto graham(const vector<vec<int>> &points) {
   vec<int> p0 = points[0];

   for (auto p : points)
      if (p.y < p0.y || p.y == p0.y && p.x > p0.x) p0 = p;

   for (auto &p : points) {
      p.x -= p0.x;
      p.y -= p0.y;
   }

   sort(all(points), [] (vec<int> &p1, vec<int> &p2) {
      return (p1 ^ p2).z > 0 || (p1 ^ p2).z == 0 && p1.
           norm() > p2.norm();
   });

   vector<vec<int>> hull;
   for (auto &p : points) {
      while (hull.size() >= 2 &&
      (((p - hull.back()) ^ (hull[hull.size() - 1] - hull[
           hull.size() - 2]))).z >= 0)
         hull.pop_back();
      hull.push_back(p);
   }

   for (auto &p : hull) {
      p.x += p0.x;
      p.y += p0.y;
   }

   return hull;
}
```

## 4.4 Formulae

**Triangles.**

*Radius of circumscribed circle:*
$R = \frac{abc}{4S}$.
*Radius of inscribed circle:*
$r = \frac{S}{p}$.
*Side via medians:*
$a = \frac{2}{3}\sqrt{2(m_b^2 + m_c^2) - m_a^2}$.
*Median via sides:*
$m_a = \frac{1}{2}\sqrt{2(b^2 + c^2) - a^2}$.
*Bisector via sides:*
$l_a = \frac{2\sqrt{bcp(p-a)}}{b+c}$.
*Bisector via two sides and angle:*
$l_a = \frac{2bc\cos\frac{\alpha}{2}}{b+c}$.
*Bisector via two sides and divided side:*
$l_a = \sqrt{bc - a_b a_c}$.

**Right triangles**.
$a$, $b$ - cathets, $c$ - hypotenuse.
$h$ - height to hypotenuse, divides $c$ to $c_a$ and $c_b$.

$$\begin{cases} h^2 = c_a \cdot c_b, \\ a^2 = c_a \cdot c, \\ b^2 = c_b \cdot c. \end{cases}$$

**Quadrangles**.
*Sides of circumscribed quadrangle:*
$a + c = b + d$.
*Square of circumscribed quadrangle:*
$S = \frac{Pr}{2} = pr$.
*Angles of inscribed quadrangle:*
$\alpha + \gamma = \beta + \delta = 180°$.
*Square of inscribed quadrangle:*
$S = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

**Circles**.
*Intersection of circle and line:*

$$\begin{cases} (x - x_0)^2 + (y - y_0)^2 = R^2 \\ y = ax + b \end{cases}$$

Task comes to solution of $\alpha x^2 + \beta x + \gamma = 0$, where

$$\begin{cases} \alpha = (1 + a^2), \\ \beta = (2a(b - y_0) - 2x_0), \\ \gamma = (x_0^2 + (b - y_0)^2 - R^2). \end{cases}$$

*Intersection of circle and circle:*

$$\begin{cases} (x - x_0)^2 + (y - y_0)^2 = R_0^2 \\ (x - x_1)^2 + (y - y_1)^2 = R_1^2 \end{cases}$$

$y = \frac{1}{2}\frac{(R_1^2 - R_0^2) + (x_0^2 - x_1^2) + (y_0^2 - y_1^2)}{y_0 - y_1} - \frac{x_0 - x_1}{y_0 - y_1}x$
Task comes to intersection of circle and line.

# 5 Stringology

## 5.1 Z Function

```
// Theme: Z-Function
// Alrotihm: Linear Algorithm
// Complexity: O(N)

auto z_func(const string &s) {
   vector<int> z(s.size());

   for (int i = 1, l = 0, r = 0; i < s.size(); i++) {
      if (i <= r) z[i] = min(r - i + 1, z[i - l]);

      while (i + z[i] < s.size() && s[z[i]] == s[i + z[i
           ]]) z[i]++;

      if (i + z[i] - 1 > r) {
         l = i;
         r = i + z[i] - 1;
      }
   }

   return z;
}
```

## 5.2 Manacher

```
// Theme: Palindromes
// Alrotihm: Manacher's Algorithm
// Complexity: O(N)

int manacher(const string s) {
   int l, r, n = s.size();
   vector<int> d1(n), d2(n);

   l = 0, r = -1;
   for (int i = 0; i < n; i++) {
      int k = i > r ? 1 : min(d1[l + r - i], r - i + 1);
      while (i + k < n && i - k >= 0 && s[i + k] == s[i -
           k]) k++;
      d1[i] = k;
      if (i + k - 1 > r) {
         l = i - k + 1;
         r = i + k - 1;
      }
   }

   l = 0, r = -1;
   for (int i = 0; i < n; i++) {
      int k = i > r ? 0 : min(d2[l + r - i + 1], r - i +
           1);
      while (i + k < n && i - k - 1 >= 0 && s[i + k] == s[
           i - k - 1]) k++;
      d2[i] = k;
      if (i + k - 1 > r) {
         l = i - k;
         r = i + k - 1;
      }
   }

   int res = 0;
   for (int i = 0; i < n; i++) {
      res += ((d1[i] > 1) ? d1[i] - 1 : 0) + d2[i];
   }

   return res;
}
```

## 5.3 Trie

```
// Theme: Trie
// Algorithm: Aho-Corasick
// Complexity: O(N)

struct trie {
   struct vertex { // Vertex
      vector<int> next;
      bool leaf;
   };

   static const int K = 26; // Alphabet size
   static const int N = 2e5 + 1; // Maximum Vertex Number

   vector<vertex> t; // Vertices Vector
   int sz;

   trie(): sz(1) {
      t.resize(N);
      t[0].next.assign(K, -1);
```

```
    }

    void add_str(const string &s) {
        int v = 0;
        for (int i = 0; i < s.length(); i++) {
            char c = s[i] - 'a';
            if (t[v].next[c] == -1) {
                t[sz].next.assign(K, -1);
                t[v].next[c] = sz++;
            }
            v = t[v].next[c];
        }
        t[v].leaf = true;
    }
};
```

## 5.4 Prefix Function

```
// Theme: Prefix function
// Alrotihm: Prefix Function Algoritms
// Complexity: O(N)

auto pref_func(const string &s) {
    int n = s.length();
    vector<int> pi(n);

    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];

        while (j > 0 && s[i] != s[j]) j = pi[j - 1];

        if (s[i] == s[j]) j++;

        pi[i] = j;
    }

    return pi;
}
```

## 5.5 Suffix Array

```
// suffix array algo with count sort

void count_sort(vector<int> &p, vector<int> &c) {
    int n = p.size();
    vector<int> cnt(n), p_new(n), pos(n);

    for (auto x : c) cnt[x]++;

    pos[0] = 0;
    for (int i = 1; i < n; i++)
        pos[i] = pos[i - 1] + cnt[i - 1];

    for (auto x : p) {
        int i = c[x];
        p_new[pos[i]] = x;
        pos[i]++;
    }

    p = p_new;
}

auto suffix_array(const string &str) {
    string s = str + '$';
    int n = s.length();

    vector<int> p(n), c(n);
    vector<pair<char, int>> a(n);

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

    sort(a.begin(), a.end());

    for (int i = 0; i < n; i++) p[i] = a[i].second;

    c[p[0]] = 0;
    for (int i = 1; i < n; i++)
        c[p[i]] = c[p[i - 1]] + (a[i].first != a[i - 1].
              first);

    int k = 0;
    while ((1 << k) < n) {
        for (int i = 0; i < n; i++)
            p[i] = (p[i] - (1 << k) + n) % n;

        count_sort(p, c);

        vector<int> c_new(n);

        c_new[p[0]] = 0;
        for (int i = 1; i < n; i++) {
            pair<int, int> prev = { c[p[i - 1]], c[(p[i - 1]
                  + (1 << k)) % n] };
            pair<int, int> now = { c[p[i]], c[(p[i] + (1 << k
                  )) % n] };
            c_new[p[i]] = c_new[p[i - 1]] + (now != prev);
        }

        c = c_new;
        k++;
    }

    return p;
}
```

# 6 Dynamic Programming

## 6.1 Increasing Subsequence

```
// Theme: Longest Increasing Subsequence
// Alrotihm: Binary Search Algorithm
// Complexity: O(N)

auto inc_seq(const vector<int> &a) {
    int n = a.size();
    vector<int> dp(n + 1, inf), pos(n + 1), prev(n), path;

    int len = 0;
    dp[0] = -inf;
    pos[0] = -1;

    for (int i = 0; i < n; i++) {
        int j = distance(dp.begin(), upper_bound(all(dp), a[
              i]));
        if (dp[j - 1] < a[i] && a[i] < dp[j]) {
            dp[j] = a[i];
            pos[j] = i;
            prev[i] = pos[j - 1];
            len = max(len, j);
        }
    }

    int p = pos[len];
    while (p != -1) {
        path.push_back(a[p]);
        p = prev[p];
    }
    reverse(path.begin(), path.end());

    return path;
}
```

# 7 Graphs

## 7.1 Graph Travesing

```
// Theme: Graph Traversing

vector<vector<int>> g; // Graph
vector<int> u; // Used

// Algorithm: Breadth-First Search
// Complexity: O(N + M)

void bfs(int v) {
    queue<int> q; // Queue

    u[v] = 1;
    q.push(v);
```

```
   while (q.size()) {
      int w = q.front(); q.pop();

      for (auto &to : g[w]) {
         if (u[to]) continue;
         u[to] = 1;
         q.push(to);
      }
   }
}

// Algorithm: Depth-First Search
// Complexity: O(N + M)

void dfs(int v, int p = -1) {
   u[v] = 1;

   for (auto &to : g[v]) {
      if (to == p || u[to]) continue;
      dfs(to, v);
   }
}
```

## 7.2 Topological Sort

```
// Theme: Graph Topological Sorting
// Algorithm: DFS Based Algorithm
// Complexity: O(N + M)

vector<vector<int>> g; // Graph
vector<int> u; // Used
vector<int> ans; // Sorted Vertices

void dfs(int v, int p = -1) {
   u[v] = 1;

   for (auto &to : g[v]) {
      if (to == p || u[to]) continue;
      dfs(to, v);
   }

   ans.push_back(v);
}

void topsort(int n) {
   for (int i = 0; i < n; i++)
      if (!u[i])
         dfs(i);

   reverse(all(ans));
}
```

## 7.3 Dijkstra

```
// Theme: Shortest Paths From Vertex
// Alrotihm: Dijkstra's Algorithm
// Complexity: O(M*log(N))

const int inf = 1e18; // Infinity Value

vector<vector<pair<int, int>>> g; // Graph <Vertex, Length>
vector<int> d; // Result Distances
vector<int> p; // Path Back

void dijkstra(int v, int n) {
   priority_queue<pair<int, int>> q; // Priority Queue <-
        Distance, Vetex>

   d.assign(n, inf); d[v] = 0;
   p.assign(n, 0);

   q.push({ 0, v });

   while (q.size()) {
      int dist = -q.top().ff, w = q.top().ss; q.pop();
      if (dist > d[w]) continue;

      for (auto &to: g[w])
         if (d[w] + to.ss < d[to]) {
            d[to] = d[w] + to.ss;
            p[to] = w;
            q.push({ -d[to], to });
         }
   }
}
```

## 7.4 Belman Ford Algorithm

```
// Theme: Shortest Paths From Vertex
// Alrotihm: Belman-Ford's
// Complexity: O(N*M)

const int inf = 1e18;

vector<pair<int, <int, int>>> g; // Graph <Weight, <Vertex,
      Vertex>>

auto bfa(int v, int n) {
   int m = g.size();

   vector<int> d(n, inf); d[v] = 0;

   for (;;) {
      bool any = false;
      for (int j = 0; j < m; ++j)
         if (d[g[j].ss.ff] < inf &&
            d[g[j].ss.ff] + g[j].ff < d[g[j].ss.ss]) {
            d[g[j].ss.ss] = d[g[j].ss.ff] + g[j].ff;
            any = true;
         }
      if (!any) break;
   }

   return d;
}
```

## 7.5 Floyd Warshall Algorithm

```
// Theme: Shortest Paths From All Vertices
// Alrotihm: Floyd-Warshall's
// Complexity: O(N^3)

const int inf = 1e18;

vector<vector<int>> g; // Graph, Matrix

auto fwa(int n) {
   vector<vector<int>> d(n, vector<int>(n, inf));

   for (int k = 0; k < n; k++)
      for (int i = 0; i < n; i++)
         for (int j = 0; j < n; j++)
            if (d[i][k] != inf && d[k][j] != inf)
               d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

   return d;
}
```

## 7.6 Articulation Points

```
// Theme: All Graph Articulation Points
// Algorithm: DFS Based Algorithm
// Complexity: O(N + M)

vector<vector<int>> g; // Graph
vector<int> u; // Used
vector<int> tin, tup; // Enter And Exit Time
vector<int> ap; // Articulation Points

int timer; // Timer

void dfs(int v, int p = -1) {
   u[v] = 1;
   tin[v] = tup[v] = timer++;

   int children = 0;

   for (auto &to: g[v]) {
      if (to == p) continue;
      if (u[to]) tup[v] = min(tup[v], tin[to]);
      else {
         dfs(to, v);
         tup[v] = min(tup[v], tup[to]);
```

```
            if (tup[to] >= tin[v] && p != -1) result.insert(v
                 );
            children++;
         }
      }

   if (p == -1 && children > 1) result.insert(v);
}

void find_ap(int n) {
   timer = 0;
   for (int i = 0; i < n; i++)
      if (!u[i])
         dfs(i);
}
```

## 7.7 Bridges

```
// Theme: All Graph Bridges
// Algorithm: DFS Based Algorithm
// Complexity: O(N + M)

vector<vector<int>> g; // Graph
vector<int> u; // Used
vector<int> tin, tup; // Enter And Exit Time
vector<pair<int, int>> b; // Bridges <Vertex, Vertex>

int timer; // Timer

void dfs(int v, int p = -1) {
   u[v] = 1;
   tin[v] = tup[v] = timer++;

   for (auto &to: g[v]) {
      if (to == p) continue;
      if (u[to]) tup[v] = min(tup[v], tin[to]);
      else {
         dfs(to, v);
         tup[v] = min(tup[v], tup[to]);
         if (tup[to] > tin[v] && count(all(g[v]), to) ==
              1)
            b.push_back({ min(v, to), max(v, to) });
      }
   }
}

void bridges(int n) {
   timer = 0;
   for (int i = 0; i < n; i++)
      if (!u[i])
         dfs(i);
}
```

## 7.8 Vertex In Cycle

```
// Theme: Vertex In Cycle
// Algorithm: DFS Based Algorithm
// Complexity: O(N + M)

vector<vector<int>> g; // Graph
vector<int> u; // Used
vector<int> c; // Cycle Vertices
vector<int> p; // Path Back

int vs = -1; // Start Vertex
int ve = 0; // End Vertex

bool dfs(int v, int p = -1) {
   u[v] = 1;

   for (auto &to : g[v]) {
      if (to == p) continue;
      if (u[to] == 0 && dfs(to)) {
         p[to] = v;
         return true;
      }
      else if (u[to] == 1) {
         vs = to;
         ve = v;
         return true;
      }
   }
   u[v] = 2;
   return false;
}

bool find_cycle(int v) {
   if (dfs(v)) {
      for (int w = ve; w != vs; w = p[w])
         c.push_back(w);
      c.push_back(vs);
      reverse(all(c));
      return true;
   }
   else
      return false;
}
```

## 7.9 Connectivity Components

```
// Theme: Graph Connectivity Components

// Subtheme: Graph Connectivity Components Count
// Algorithm: DFS Based Algorithm
// Complexity: O(N + M)

vector<vector<int>> g; // Graph
vector<int> u; // Used

void dfs(int v, int p = -1) {
   u[v] = 1;

   for (auto &to : g[v]) {
      if (to == p || u[to]) continue;
      dfs(to, v);
   }
}

int cc(int n) {
   int count = 0;

   for (int i = 0; i < n; i++)
      if (!u[i]) {
         dfs(i);
         count++;
      }

   return count;
}

// Subtheme: Graph Strong Connectivity Components
// Algorithm: DFS Based Algorithm
// Complexity: O(N + M)

vector<vector<int>> g; // Graph
vector<vector<int>> gr; // Reversed Edges Graph
vector<int> u; // Used
vector<int> order; // Edges Order
vector<int> component; // SCC

void dfs1(int v, int p = -1) {
   u[v] = 1;

   for (auto &to : g[v]) {
      if (to == p || u[to]) continue;
      dfs(to, v);
   }

   order.push_back(v);
}

void dfs2(int v, int p = -1) {
   u[v] = 1;

   component.push_back(v);
   for (auto &to : gr[v])
      if (to != p && !u[to]) dfs2(to, v);
}


void scc(int n) {
   u.assign(n, 0);
   for (int i = 0; i < n; i++)
      if (!u[i])
         dfs1(i);
```

```
    u.assign(n, 0);
    for (int i = 0; i < n; i++) {
        int v = order[n - i - 1];
        if (!u[i]) {
            dfs2(v);
            component.clear();
        }
    }
}
```

## 7.10 Kruscal

```
// Theme: Minimum Spanning Tree
// Algorithm: Kruskal's Algorithm
// Complexity: O(M * log(N))


struct dsu { // Disjoint Set Union
    // ...
};

vector<pair<int, pair<int, int>>> g; // Graph <Weight, <
     Vertex, Vertex>>

auto kruskal(int n) {
    dsu d(n);
    vector<pair<int, pair<int, int>>> spt;

    sort(all(g));

    for (auto &e : g) {
        int w = e.ff, v = e.ss.ff, u = e.ss.ss;
        if (d.get(v) != d.get(u)) {
            res.push_back(e);
            d.unite(v, u);
        }
    }

    return spt;
}
```

## 7.11 Lowest Common Ancestor

```
// Theme: Minimum Spanning Tree
// Algorithm: Binary Lifting Method
// Complexity: O(N * log(N) + log(N))

vector<vector<int>> g; // Graph
vector<vector<int>> up; // Ancestors
vector<int> tin, tout; // Enter And Exit Time

int timer; // Timer

int l; // l == log(N) (~20)

void dfs(int v, int p = -1) {
    tin[v] = timer++;

    up[v][0] = p;
    for (int i = 1; i <= l; i++)
        up[v][i] = up[up[v][i - 1]][i - 1];

    for (auto &to : g[v]) {
        if (to == p) continue;
        dfs(to, v);
    }

    tout[v] = timer++;
}

void preprocess(int n, int r) {
    l = (int) ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    timer = 0;
    dfs(r, r);
}

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

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

## 7.12 Eulerian Path

```
// Theme: Eulerian Path (All Edges)
// Algorithm: Iterative Method
// Complexity: O(M)

vector<vector<int>> g; // Graph, Matrix
vector<int> eul; // Eulerian Path

// 0 - path not exist
// 1 - cycle exits
// 2 - path exists
int euler_path(int n) {
    vector<int> deg; // Vertex Degrees
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; ++j)
            deg[i] += g[i][j];

    int v1 = -1, v2 = -1;
    for (int i = 0; i < n; i++)
        if (deg[i] & 1)
            if (v1 == -1) v1 = i;
            else if (v2 == -1) v2 = i;
            else return 0;

    if (v1 != -1) { g[v1][v2]++; g[v2][v1]++; }

    int first = 0; while (!deg[first]) first++;

    stack<int> st; st.push(first);

    while (!st.empty()) {
        int v = st.top();
        int i; for (i = 0; i < n && !g[v][i]; i++);
        if (i == n) {
            eul.push_back(v);
            st.pop();
        }
        else {
            g[v][i]++; g[i][v]++;
            st.push(i);
        }
    }

    int res = 2;

    if (v1 != -1) {
        res = 1;

        for (int i = 0; i + 1 < eul.size(); i++)
            if (eul[i] == v1 && eul[i + 1] == v2 || eul[i] ==
                  v2 && eul[i + 1] == v1) {
                vector<int> t_eul;
                for (int j = i + 1; j < eul.size(); j++) t_eul
                     .push_back(eul[j]);
                for (int j = 1; j <= i; j++) t_eul.push_back(
                     eul[j]);
                eul = t_eul;
                break;
            }
    }

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (g[i][j]) return 0;

    return res;
}
```

## 7.13 Kuhn

```
// Theme: Maximum Matching
// Algorithm: Kuhn's Algorithm
// Complexity: O(N^3)

vector<vector<int>> g; // Graph, N -> K
vector<int> u; // Used

bool kuhn(int v) {
   if (u[v]) return false;
   u[v] = true;
   for (auto &to : g[v]) {
      if (mt[to] == -1 || kuhn(mt[to])) {
         mt[to] = v;
         return true;
      }
   }
   return false;
}

auto maxmatch(int n, int k) {
   vector<int> mt; // Edges, From Right Ro Left
   mt.assign(k, -1);

   for (int i = 0; i < n; i++) {
      u.assign(n, 0);
      kuhn(i);
   }

   return mt;
}
```

# 8 Miscellaneous

## 8.1 Ternary Search

```
// Theme: Ternary Search

// Alrotihm: Continuous Ternary Search With Goled Ratio
// Complexity: O(log(N))

double phi = (1 + sqrt(5)) / 2; // Golden Ratio

double cont_ternary_search(double l, double r) {
  double m1 = l + (r - l) / (1 + phi), m2 = r - (r - l) /
      (1 + phi);
  double f1 = f(m1), f2 = f(m2);

  int count = 200;
  while (count--) {
    if (f1 < f2) {
      r = m2;
      m2 = m1;
      f2 = f1;
      m1 = l + (r - l) / (1 + phi);
      f1 = f(m1);
    }
    else {
      l = m1;
      m1 = m2;
      f1 = f2;
      m2 = r - (r - l) / (1 + phi);
      f2 = f(m2);
    }
  }

  return f((l + r) / 2);
}

// Alrotihm: Descrete Ternary Search
// Complexity: O(log(N))

double discr_ternary_search(int l, int r) {
  int m1 = l + (r - l) / 3, m2 = r - (r - l) / 3;

  while (r - l > 2) {
    if (f(m1) < f(m2))
      r = m2;
    else
      l = m1;
    m1 = l + (r - l) / 3;
    m2 = r - (r - l) / 3;
  }

  return min(f(l), min(f(l + 1), f(r)));
}
```