ACM-ICPC Team Reference Document

Tula State University (Baklanov, Nechoroshev, Vasin)

Contents

1 Templates

1.1 C++ Template

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

#define int long long
#define endl "\n"
#define all(x) x.begin(),x.end()

using namespace std;

const double PI = 3.14159265358979323846;

signed main()
{
    cin.tie(0);
    cout.tie(0);
    ios_base::sync_with_stdio(false);
    int _t = 1;
    cin>>_t;
    for (int _i = 0; _i < _t; _i++)
    {

    }
    return 0;
}
```

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>
```

2 Data Structures

2.1 Disjoint Set Unioin

```
struct DisjointSets {
   vector<int> parent;
   vector<int> size;
   vector<long long> sum;
   vector<int> Max;
   vector<int> Min;
   void init(int n) {
      parent.resize(n + 1);
      Min.resize(n+1);
      Max.resize(n+1);
      for (int i = 1; i <= n; i++) {
         parent[i] = i;
         Min[i] = i;
         Max[i] = i;
      }
      size.assign(n + 1, 1);
   }
   int get(int v) {
      if (v == parent[v])
         return v;
      return parent[v] = get(parent[v]);
   }
   void Union(int a, int b) {
      a = get(a);
      b = get(b);
      if (a != b) {
         if (size[a] < size[b])
            swap(a, b);
         parent[b] = a;
         Min[a] = min(Min[a], Min[b]);
         Max[a] = max(Max[a], Max[b]);
         size[a] += size[b];
      }
   }
};
```

2.2 Segtree Sum

```
struct TreeSum
{
   vector<long long> tree;
   int size;
   void init(int n) {
      size = 1;
      while (size < n) {
         size *= 2;
      }
      tree.assign(2 * size - 1, 0);
   }
   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);
   }
   void build(vector<int> &a, int x, int lx, int rx) {
      if (rx - lx == 1) {
         if (lx < a.size()) tree[x] = a[lx];
      }
      else {
         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);
      return;
   }
   long long sum(int l, int r, int x, int lx, int rx) {
      if (l >= rx || lx >= r) return 0;
      if (lx >= l && rx <= r) return tree[x];
      int m = (lx + rx) / 2;
      long long sum1 = sum(l, r, 2 * x + 1, lx, m);
      long long sum2 = sum(l, r, 2 * x + 2, m, rx);
      return sum1 + sum2;
   }
   long long sum(int l, int r) {
      return sum(l, r, 0, 0, size);
   }
};
```

2.3 Segtree Countmin

```
struct TreeMin
{
   struct node {
      int min;
      int count;
   };
   node combine(node a, node b) {
      if (a.min<b.min) return a;
      if (a.min>b.min) return b;
      return {a.min, a.count+b.count};
   }
   const node ZERO = {INT_MAX,0};
   vector<node> tree;
   int size;
   void init(int n) {
      size = 1;
      while (size < n)
      {
         size *= 2;
      }
      tree.assign(2 * size - 1, {0,0});
   }
   node calc(int l, int r, int x, int lx, int rx) {
      if (l>=rx || lx>=r) return ZERO;
      if (lx>=l&&rx<=r) return tree[x];
      int m = (lx + rx) / 2;
      node sum1 = calc(l, r, 2 * x + 1, lx, m);
      node sum2 = calc(l, r, 2 * x + 2, m, rx);
      return combine(sum1, sum2);
   }
   node calc(int l, int r) {
      return calc(l, r, 0, 0, size);
   }
};
```

2.4 Segtree First Above

```
struct first_above_tree
{
   // tree_max
```

```
    int first_above(int v, int x, int lx, int rx) {
        if (tree[x] < v) return -1;
        if (rx - lx == 1) return lx;
        int m = (lx + rx) / 2;
        int res = first_above(v, 2 * x + 1, lx, m);
        if (res == -1) res = first_above(v, 2 * x + 2, m, rx
            );
        return res;
    }
    int first_above(int v) {
        return first_above(v, 0, 0, size);
    }
};
struct first_above_left_tree {
    // tree_max
    int first_above(int v, int l, int x, int lx, int rx) {
        if (tree[x] < v || rx <= l) return -1;
        if (rx - lx == 1) return lx;
        int m = (lx + rx) / 2;
        int res = first_above(v, l, 2 * x + 1, lx, m);
        if (res == -1) res = first_above(v, l, 2 * x + 2, m,
            rx);
        return res;
    }
    int first_above(int v, int l) {
        return first_above(v, l, 0, 0, size);
    }
};
```

2.5 Segtree K Ones

```
struct k_ones_tree {
    // tree_sum
    int find(int k, int x, int lx, int rx) {
        if (rx - lx == 1) return lx;
        int m = (rx + lx) / 2;
        if (k < tree[2 * x + 1]) return find(k, 2 * x + 1,lx
            , m);
        else
            return find(k - tree[2 * x + 1], 2 * x + 2, m, rx
                );
    }
    int find(int k) {
        return find(k, 0, 0, size);
    }
};
```

2.6 Segtree Segmentmaxsum

```
struct TreeMin {
    struct node {
        long long seg, pref, suf, sum;
    };
    node combine(node a, node b) {
        return {
            /*seg*/ max(a.seg, max(b.seg, a.suf+b.pref)),
            /*pref*/ max(a.pref, a.sum+b.pref),
            /*suf*/ max(b.suf, b.sum+a.suf),
            /*sum*/ a.sum+b.sum
        };
    }
    const node ZERO = {0,0,0,0};
    node one_eleme(int x){
        return {
            max(x,0), //seg
            max(x,0), //pref
            max(x,0), //suf
            x //sum
        };
    }
    vector<node> tree;
    int size;
    void init(int n) {
        size = 1;
        while (size < n)
        {
            size *= 2;
        }
        tree.assign(2 * size - 1, {0,0});
    }
    node calc(int l, int r, int x, int lx, int rx) {
        if (l>=rx || lx>=r) return ZERO;
        if (lx>=l&&rx<=r) return tree[x];
        int m = (lx + rx) / 2;
        node sum1 = calc(l, r, 2 * x + 1, lx, m);
        node sum2 = calc(l, r, 2 * x + 2, m, rx);
        return combine(sum1, sum2);
    }
    node calc(int l, int r) {
        return calc(l, r, 0, 0, size);
    }
};
```

2.7 Segrree Lazypropagate

```
// mass assignment
struct lazy_seg_tree {
    vector<int> tree, lazy;
    int size;
    init(int n) {
        size = 1;
        while (size < n) size <<= 1;
        tree.assign(2 * size - 1, 0);
        lazy.assign(2 * size - 1, 0);
    }
    void push(int x) {
        tree[2 * x + 1] = lazy[x];
        lazy[2 * x + 1] = lazy[x];
        tree[2 * x + 2] = lazy[x];
        lazy[2 * x + 2] = lazy[x];
        lazy[x] = -1;
    }
    void update(int v, int l, int r, int x, int lx, int rx)
         {
        if (rx <= l && r <= lx) return;
        if (l <= lx && rx <= r) {
            push(x);
            tree[x] = v;
            lazy[x] = v;
            return;
        }
        int m = (lx + rx) / 2;
        tree[x] = v;
        lazy[x] = v;
        update(v, l, r, 2 * x + 1, lx, m);
        update(v, l, r, 2 * x + 2, m, rx);
    }
    void update(int v, int l, int r) {
        update(v, l, r, 0, 0, size);
    }
    int get(int i, int x, int lx, int rx) {
        if (rx - lx == 1) return tree[x];
        int m = (lx + rx) / 2;
        if (i < m) get(i, 2 * x + 1, lx, m);
        else get(i, 2 * x + 2, m, rx);
    }
    int get(int i) {
        return get(i, 0, 0, size);
    }
};
```

2.8 Segtree Propagetesum

```
struct TreeSeg {
    struct node {
        int set;
        int sum;
    };
    vector<node> tree;
    int size;
    int MOD = 1e9 + 7;
    int NETRAL = 0;
    int NO_OPERATION = LLONG_MIN;
    int operat_modify(int a, int b, int len) {
        if (b == NO_OPERATION)
            return a;
        return b * len;
    }
    int operat_min(int a, int b) {
        return a + b;
    }
    void propagate(int x, int lx, int rx) {
        if (tree[x].set == NO_OPERATION || rx - lx == 1)
            return;
```

```
        int m = (lx + rx) / 2;
        tree[2 * x + 1].set = operat_modify(tree[2 * x + 1].
             set, tree[x].set, 1);
        tree[2 * x + 1].sum = operat_modify(tree[2 * x + 1].
             sum, tree[x].set, m - lx);
        tree[2 * x + 2].set = operat_modify(tree[2 * x + 1].
             set, tree[x].set, 1);
        tree[2 * x + 2].sum = operat_modify(tree[2 * x + 1].
             sum, tree[x].set, rx - m);
        tree[x].set = NO_OPERATION;
    }
    void init(int n) {
        size = 1;
        while (size < n) size *= 2;
        tree.assign(2 * size - 1, {0, 0});
    }
    int suma(int l, int r, int x, int lx, int rx) {
        propagate(x, lx, rx);
        if (l >= rx || lx >= r) return NETRAL;
        if (lx >= l && rx <= r) return tree[x].sum;
        int m = (lx + rx) / 2;
        int m1 = suma(l, r, 2 * x + 1, lx, m);
        int m2 = suma(l, r, 2 * x + 2, m, rx);
        int res = operat_min(m1, m2);
        return res;
    }
    int suma(int l, int r) {
        return suma(l, r, 0, 0, size);
    }
    void build(vector<int> &a, int x, int lx, int rx) {
        if (rx - lx == 1) {
            if (lx < a.size()) tree[x].sum = a[lx];
        }
        else {
            int m = (lx + rx) / 2;
            build(a, 2 * x + 1, lx, m);
            build(a, 2 * x + 2, m, rx);
            tree[x].sum = operat_min(tree[2 * x + 1].sum,
                 tree[2 * x + 2].sum);
        }
    }
    void build(vector<int> &a) {
        init(a.size());
        build(a, 0, 0, size);
        return;
    }
    void modify(int l, int r, int v, int x, int lx, int rx)
         {
        propagate(x, lx, rx);
        if (l >= rx || lx >= r) return;
        if (lx >= l && rx <= r) {
            tree[x].set = operat_modify(tree[x].set, v, 1);
            tree[x].sum = operat_modify(tree[x].sum, v, (rx -
                 lx));
            return;
        }
        int m = (lx + rx) / 2;
        modify(l, r, v, 2 * x + 1, lx, m);
        modify(l, r, v, 2 * x + 2, m, rx);
        tree[x].sum = operat_min(tree[2 * x + 1].sum, tree[2
             * x + 2].sum);
    }
    void modify(int l, int r, int v) {
        return modify(l, r, v, 0, 0, size);
    }
};
```

3 Algebra

3.1 Binpow

```
int binpow_mod(int a, int pow, int MOD) {
    if (!pow) {
        return 1;
    }
    if (pow % 2 == 1) {
        return (__int128_t)binpow_mod(a, pow - 1, MOD) * a %
             MOD;
    }
    else {
        __int128_t tmp = binpow_mod(a, pow / 2, MOD);
        __int128_t temp = (tmp * tmp) % MOD;
        return temp;
    }
}

int binpow(int a, int n) {
    int res = 1;
    while (n) {
        if (n & 1)
            res *= a;
        a *= a;
        n >>= 1;
    }
    return res;
}

int binmul(int a, int b) {
    int res = 0;
    while (b)
    {
        if (b & 1)
            res += a;
        a *= 2;
        b >>= 1;
    }
    return res;
}
```

3.2 Factorization

```
vector<int> factorization(long long n)
{
    vector<int> result;
    for (int i = 2; i * i < n; i++)
        while (n % i == 0)
        {
            result.push_back(i);
            n /= i;
        }
    if (n != 1)
        result.push_back(n);
    return result;
}
```

3.3 Gcd Rev Elem

```
//simple gcd
int gcd(int a, int b) {
    while (b)
    {
        a %= b;
        swap(a, b);
    }
    return a;
}
// euclidean algorithm
int advanced_gcd(int a, int b, int &x, int &y) {
    if (a == 0)
    {
        x = 0;
        y = 1;
        return b;
    }
    int x1, y1;
    int d = advanced_gcd(b % a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}

//rev element
int rev_elem(int a, int m) {
    int x, y;
    int g = advanced_gcd(a, m, x, y);
    if (g!=1) return 0;
    else return (x % m + m) % m;
}
```

3.4 FFT

```
const int fft_mod = 998244353;
const int fft_root = 31; // 31 ^ (2^23) == 1 mod 998244353
const int fft_root_1 = 128805723; // 31 * == 1 mod
     998244353
const int fft_pw = 1 << 23;
// const int fft_mod = 7340033; // 7 * 2^20 + 1
// const int fft_root = 5; // 5 ^ (2^20) == 1 mod 7340033
// const int fft_root_1 = 4404020; // 5 * 4404020 == 1 mod
     7340033
// const int fft_pw = 1 << 20;

vector<int> fft(vector<int> a, bool invert = 0)
{
   int n = a.size();

   for (int i = 1, j = 0; i < n; i++)
   {
      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 root_len = invert ? fft_root_1 : fft_root;
      for (int i = len; i < fft_pw; i <<= 1)
         root_len = (root_len * root_len) % fft_mod;
      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 % fft_mod;
            a[i + j] = (u + v) % fft_mod;
            a[i + j + len / 2] = (u - v + fft_mod) %
                 fft_mod;
            root = (root * root_len) % fft_mod;
         }
      }
   }

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

   return a;
}
```

3.5 Matrics

```
vector<vector<int>> matrix_production(vector<vector<int>>&
     a, vector<vector<int>>& b, int mod) {
   vector<vector<int>> result(a.size(), vector<int>(b[0].
        size()));
   for (int i = 0; i < a.size(); i++) {
      for (int j = 0; j < b[0].size(); j++) {
         for (int k = 0; k < b.size(); k++) {
            result[i][j] += a[i][k] * b[k][j];
            if (mod) result[i][j] %= mod;
         }
      }
   }
   return result;
}

vector<vector<int>> fast_pow(vector<vector<int>>& a, int n)
     {
   if (n == 0) {
      vector<vector<int>> temp(a.size(), vector<int>(a[0].
           size()));
      for (int i = 0; i < a.size(); i++) {
         temp[i][i] = 1;
      }
      return temp;
   }
   if (n % 2 == 1) {
      vector<vector<int>> temp = fast_pow(a, n - 1);
      return matrix_production(temp, a, 1e7 + 7);
   }
   else {
      vector<vector<int>> b = fast_pow(a, n / 2);
      return matrix_production(b, b, 1e7 + 7);
   }
}
```

3.6 Fibonacci

```
signed fibonacci()
{
   int n = 0, m = 0;
   cin >> n >> m;
   vector<vector<int>> mass(2, vector<int>(2));
   mass[0][0] = 0;
   mass[0][1] = 1;
   mass[1][0] = 1;
   mass[1][1] = 1;
   if (n == 1)
   {
      cout << 1 << endl;
      return 0;
   }
   if (n == 2)
   {
      cout << 1 << endl;
      return 0;
   }
   if (n == 3)
   {
      cout << 2 << endl;
      return 0;
   }
   vector<vector<int>> powed = fast_pow(mass, n - 3, m);
   int result = 0;
   for (int i = 0; i < 2; i++)
   {
      for (int j = 0; j < 2; j++)
         result += powed[i][j];
   }
   cout << result % m << endl;
}
```

3.7 Euler Totient Fun

```
// number of numbers x < n so that gcd(x, n) = 1
int phi(int n)
{
   if (n == 1)
      return 1;
   // f = vector<pair<prime, count>>
   auto f = factorization(n);
   int res = n;
   for (auto p : f)
   {
      res = res - res / p.first;
   }
   return res;
}
```

3.8 Combinations

```
int c(int n, int k)
{
   int result = 1;
   for (int i = 1; i <= k; i++)
   {
      result *= n - i + 1;
      result /= i;
   }
   return result;
}
//triangle pascal
const int N = 20;
vector<vector<int>> C(N + 1, vector<int>(N + 1, 1));
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];
```

3.9 Extended Euclidean Alg

```
// ax + by = c
bool solve_eq(int a, int b, int c, int &x, int &y, int &g)
{
   solve_eq(a, b, x, y, g);
   if (c % g != 0)
      return false;
   x *= c / g;
   y *= c / g;
   return true;
}

// finds a solution (x, y) so that x >= 0 and x is minimal
bool solve_eq_non_neg_x(int a, int b, int c, int &x, int &y
     , int &g)
{
   if (!solve_eq(a, b, c, x, y, g))
      return false;
   int k = x * g / b;
   x = x - k * b / g;
   y = y + k * a / g;
   if (x < 0)
   {
      x += b / g;
      y -= a / g;
   }
   return true;
}
```

3.10 Eratosthenes

```
#include <iostream>
#include <vector>

std::vector<int> sieve_of_eratosthenes(int n, int m) {
  std::vector<int> primes;
  std::vector<bool> is_prime(m + 1, true);

  is_prime[0] = is_prime[1] = false;

  for (int p = 2; p * p <= m; p++) {
    if (is_prime[p]) {
      for (int i = p * p; i <= m; i += p) {
        is_prime[i] = false;
      }
    }
  }

  for (int i = n; i <= m; i++) {
    if (is_prime[i]) {
      primes.emplace_back(i);
    }
  }

  return primes;
}
```

3.11 Polard

```
int get_random_number(int l, int r) {
   random_device random_device;
   mt19937 generator(random_device());
   uniform_int_distribution<int> distribution(l, r);

   return distribution(generator);
}

int gcd(int a, int b) {
   if (b == 0) {
      return a;
   }

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

int polard(int n) {
   int g = 1;
   for (int i = 0; i < 5; i++) {
      int x = get_random_number(1, n);
      int c = get_random_number(1, n);
      int h = 0;
      while (g == 1) {
         x = f(x, c, n) % n;
         int y = f(f(x, c, n), c, n) % n;
         g = gcd(abs(x - y), n);
         if (g == n) {
            g = 1;
         }
         h += 1;
         if (h > 4 * (int)pow(n, 1.0 / 4)) {
            break;
         }
      }

      if (g > 1) {
         return g;
      }
   }
   return -1;
}
signed main()
{
   int n = 0;
   cin >> n;
   vector<int> a;
   while (n > 1) {
      int m = ff(n);
      if (m > 0) {
         n = n / m;
         a.push_back(m);
      }
      else {
         break;
      }
   }
   vector<int> ans;
   a.push_back(n);
   for (auto& it : a) {
      int i = 2;
      int m = it;
      while (i * i <= m) {
         if (m % i == 0) {
            ans.push_back(i);
            m = m / i;
         }
         else {
            i += 1;
         }
      }
      ans.push_back(m);
   }
   sort(all(ans));
   for (int i = 0; i < ans.size(); i++) {
      cout << ans[i] << " ";
   }
   cout << endl;
   return 0;
}
```

3.12 Test Milera Rabera

```
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
  int ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (int)M);
}
ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2) {
    if (e & 1) {
      ans = modmul(ans, b, mod);
    }
  }
  return ans;
}

bool isPrime(ull n) {
  if (n < 2 || (n % 6) % 4 != 1) {
    return (n | 1) == 3;
  }
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504,
       1795265022},
```

```
    s = __builtin_ctzll(n - 1), d = n >> s;
  for (ull a : A) { // ^ count trailing zeroes
    ull p = modpow(a % n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--) {
      p = modmul(p, p, n);
    }
    if (p != n - 1 && i != s) {
      return 0;
    }
  }

  return 1;
}
```

3.13 Baby Step Giant

```
// 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) {
   // a ^ (-1) = a ^ (p - 2) mod p
    int m = ceil(sqrt(p)), _a = binpow(a, p - 2, p);
    // s[b * a ^ (-j)] = j
    unordered_map<int, int> s;
    for (int j = 0, t = b; j < m; j++, t = t * _a % p)
        s[t] = j;
    for (int i = 0; i < m; i++) {
        // s.find(a ^ (i * m))
        auto f = s.find(binpow(a, i * m, p));
        // i * m + j
        if (f != s.end())
            return i * m + f->ss;
    }
    return -1;
}
```

3.14 Code Grey

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

3.15 Factor Mod

```
int factmod (int n, int p) {
       int res = 1;
       while (n > 1) {
              res = (res * ((n/p) % 2 ? p-1 : 1)) % p;
              for (int i=2; i<=n%p; ++i)
                     res = (res * i) % p;
              n /= p;
       }
       return res % p;
}
```

3.16 Primitive Roots

```
int powmod (int a, int b, int p) {
       int res = 1;
       while (b)
              if (b & 1)
                     res = int (res * 1ll * a % p), --b;
              else
                     a = int (a * 1ll * a % p), b >>= 1;
       return res;
}
int generator (int p) {
       vector<int> fact;
       int phi = p-1, n = phi;
       for (int i=2; i*i<=n; ++i)
              if (n % i == 0) {
                     fact.push_back (i);
                     while (n % i == 0)
                            n /= i;
              }
       if (n > 1)
              fact.push_back (n);
       for (int res=2; res<=p; ++res) {
              bool ok = true;
              for (size_t i=0; i<fact.size() && ok; ++i)
                     ok &= powmod (res, phi / fact[i], p) !=
                           1;
              if (ok) return res;
       }
       return -1;
}
```

3.17 Catalan

```
const int MX = 1000005;
const int MD = 100000007;
ll powr(ll n, ll p) {
    if (p == 0)
        return 1;
    if (p == 1)
        return n;
    if (p & 1LL)
        return (powr(n, p - 1) * n) % MD;
    else
    {
        ll x = powr(n, p / 2) % MD;
        return (x * x) % MD;
    }
}
ll inverse(ll n) {
    return (powr(n, MD - 2)) % MD;
}
ll ft[MX];
//first vizov (preprocessing)
void fact() {
    ll i;
    ft[0] = 1;
    for (i = 1; i < MX; i++)
    {
        ft[i] = (ft[i - 1] * i) % MD;
    }
}

ll nCr(ll n, ll r) {
    ll x = ft[n];
    ll y = inverse((ft[r] * ft[n - r]) % MD) % MD;
    return (x * y) % MD;
}

ll catalan(ll n)
{
    ll x = nCr(2 * n, n);
    return (x * inverse((n + 1))) % MD;
}
```

3.18 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 Graham

```
struct point {
  int x, y;
};
point operator-(point a, point b) { return {a.x - b.x, a.y
     - b.y}; }
bool operator==(point a, point b) { return (a.x == b.x) &&
     (a.y == b.y); }
int operator^(point a, point b) { return a.x * b.y - a.y *
     b.x; }
bool comp(point &a, point &b) {
  return ((a ^ b) > 0) ||
       ((a ^ b) == 0 && a.x * a.x + a.y * a.y > b.x * b.x
            + b.y * b.y);
}
std::vector<point> graham(std::vector<point> points) {
  point p0 = points[0];
  for (point p : points) {
    if (p.y < p0.y || (p.y == p0.y && p.x > p0.x)) {
      p0 = p;
    }
  }
  for (point &p : points) {
    p.x -= p0.x;
    p.y -= p0.y;
  }
  std::sort(points.begin(), points.end(), comp);
  std::vector<point> hull;
  for (point p : points) {
    while (hull.size() >= 2 &&
          ((p - hull.back()) ^ (hull[hull.size() - 2] - hull
               .back())) <= 0) {
      hull.pop_back();
    }
    hull.push_back(p);
  }
  for (point &p : hull) {
    p.x += p0.x;
    p.y += p0.y;
  }
  return hull;
}
int main() {
  std::vector<point> points = {{1, 2}, {3, 4}, {5, 6}, {7,
       8}};
  std::vector<point> hull = graham(points);
  for (point p : hull) {
    std::cout << "(" << p.x << ", " << p.y << ")" << std::
         endl;
  }
  return 0;
}
```

4.2 Circle Line Intersection

```
// ax + by + c = 0, radius is at (0, 0)
double r, a, b, c;
// If the center is not at (0, 0), fix the constant c to
translate
   everything so that center is at(0, 0) double x0 = -a * c
        / (a * a + b * b),
                                        y0 = -b * c / (a *
                                             a + b * b);
if (c * c > r * r * (a * a + b * b) + eps)
  puts("no points");
else if (abs(c * c - r * r * (a * a + b * b)) < eps) {
  puts("1 point");
  cout << x0 << ' ' << y0 << '\n';
} else {
  double d = r * r - c * c / (a * a + b * b);
  double mult = sqrt(d / (a * a + b * b));
  double ax, ay, bx, by;
  ax = x0 + b * mult;
  bx = x0 - b * mult;
  ay = y0 - a * mult;
  by = y0 + a * mult;
  puts("2 points");
  cout << ax << ' ' << ay << '\n'
      << bx << ' ' << by << '\n'
}
```

4.3 7zip Cord

```
ll dfs(vector<vector<int>> &Map, int i, int j, vector<
     vector<bool>> &used, vector<int> &Xvalue, vector<int>
     &Yvalue) {
   used[i][j] = true;
   bool flag = false;
   ll sum = Xvalue[i] * Yvalue[j];
   int a[] = {0, -1, 1, 0};
   int b[] = {-1, 0, 0, 1};
   for (int h = 0; h < 4; h++)
      if (Map[i + a[h]][j + b[h]] == 0 && !used[i + a[h]][
           j + b[h]]) {
         flag = true;
         sum += dfs(Map, i + a[h], j + b[h], used, Xvalue,
              Yvalue);
      }
   if (!flag) {
      return Xvalue[i] * Yvalue[j];
   }
   return sum;
}
int main() {
   int w, h, n;
   cin >> w >> h >> n;
   set<int> x, y;
   unordered_map<int, int> X, Y;
   vector<vector<int>> lines;
   vector<int> Xvalue, Yvalue;
   x.insert(0);
   y.insert(0);
   x.insert(w);
   y.insert(h);
   for (int i = 0; i < n; i++) {
      int x1, y1, x2, y2;
      cin >> x1 >> y1 >> x2 >> y2;
```

```
        if (x1 < 0) x1 = 0;
        if (x1 > w) x1 = w;
        if (y1 < 0) y1 = 0;
        if (y1 > h) y1 = h;
        if (x2 < 0) x2 = 0;
        if (x2 > w) x2 = w;
        if (y2 < 0) y2 = 0;
        if (y2 > h) y2 = h;
        lines.push_back({x1, y1, x2, y2});
        x.insert(x1);
        x.insert(x2);
        y.insert(y1);
        y.insert(y2);
    }
    int index = 0;
    for (auto _x : x) {
        X[_x] = index;
        index += 2;
    }
    index = 0;
    for (auto _y : y) {
        Y[_y] = index;
        index += 2;
    }
    int prev = 0;
    for (auto _x = ++x.begin(); _x != x.end(); _x++) {
        Xvalue.push_back(0);
        Xvalue.push_back(*_x - prev);
        prev = *_x;
    }
    Xvalue.push_back(0);
    prev = 0;
    for (auto _y = ++y.begin(); _y != y.end(); _y++) {
        Yvalue.push_back(0);
        Yvalue.push_back(*_y - prev);
        prev = *_y;
    }
    Yvalue.push_back(0);
    int Xs = Xvalue.size();
    int Ys = Yvalue.size();
    vector<vector<int>> Map(Xs, vector<int>(Ys, 0));
    for (int i = 0; i < Xs; i++) {
        Map[i][0] = 1;
        Map[i][Ys - 1] = 1;
    }
    for (int i = 0; i < Ys; i++) {
        Map[0][i] = 1;
        Map[Xs - 1][i] = 1;
    }
    for (int i = 0; i < n; i++) {
        if (lines[i][0] == lines[i][2])
        {
            int x = X[lines[i][0]];
            int y1 = Y[lines[i][1]];
            int y2 = Y[lines[i][3]];
            if (y1 > y2)
                y1 ^= y2 ^= y1 ^= y2;
            for (int i = y1; i <= y2; i++)
                Map[x][i] = 1;
        }
        else {
            int y = Y[lines[i][1]];
            int x1 = X[lines[i][0]];
            int x2 = X[lines[i][2]];
            if (x1 > x2)
                x1 ^= x2 ^= x1 ^= x2;
            for (int i = x1; i <= x2; i++)
                Map[i][y] = 1;
        }
    }
    vector<ll> s;
    vector<vector<bool>> used(Xs, vector<bool>(Ys, false));
    for (int i = 1; i < Xs - 1; i++) {
        for (int j = 1; j < Ys - 1; j++) {
            if (Map[i][j] == 0 && !used[i][j])
                s.push_back(dfs(Map, i, j, used, Xvalue,
                            Yvalue));
        }
    }
    sort(s.rbegin(), s.rend());
    for (auto _s : s)
        cout << _s << "\n";
}
```

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:
$\alpha + \gamma = \beta + \delta = 180^\circ$.
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

```
string z_func()
{
   string str;
   cin >> str;
   vector<int> Z(str.length(), 0);
   int n = str.length();
   int l = 0, r = 0;
   for (int i = 1; i < n; i++)
   {
      if (r >= i)
      {
         Z[i] = min(Z[i - l], r - i + 1);
      }
      while (Z[i] + i < n && str[Z[i]] == str[Z[i] + i])
         Z[i]++;

      if (r < i + Z[i] - 1)
      {
         l = i;
         r = i + Z[i] - 1;
      }
   }
}
```

5.2 Manaker

```
signed manaker()
{
   string s;
   cin >> s;
   int n = s.length();
   vector<int> d1(n);
   int 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;
   }
   vector<int> d2(n);
   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 sum = 0;
   for (int i = 0; i < n; i++)
   {
      sum += ((d1[i] > 1) ? d1[i] - 1 : 0) + d2[i];
   }
   cout << sum << '\n';
}
```

5.3 Suffix Array

```
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;
}
signed suffix_array()
{
   string str;
   cin >> str;
   str += "&";
   int len = str.length();
   vector<int> p(len), c(len);
   vector<pair<char, int>> a(len);
   for (int i = 0; i < len; i++)
      a[i] = {str[i], i};
   sort(a.begin(), a.end());
   for (int i = 0; i < len; i++)
      p[i] = a[i].second;
   c[p[0]] = 0;
   for (int i = 1; i < len; i++)
      if (a[i].first == a[i - 1].first)
         c[p[i]] = c[p[i - 1]];
      else
         c[p[i]] = c[p[i - 1]] + 1;
   int k = 0;
   while ((1 << k) < len)
   {
      for (int i = 0; i < len; i++)
         p[i] = (p[i] - (1 << k) + len) % len;
      count_sort(p, c);
      vector<int> c_new(len);
      c_new[p[0]] = 0;
      for (int i = 1; i < len; i++)
      {
         pair<int, int> prev = {c[p[i - 1]], c[(p[i - 1] +
               (1 << k)) % len]};
         pair<int, int> now = {c[p[i]], c[(p[i] + (1 << k)
              ) % len]};
         if (now == prev)
            c_new[p[i]] = c_new[p[i - 1]];
         else
            c_new[p[i]] = c_new[p[i - 1]] + 1;
      }
      c = c_new;
      k++;
   }
   for (int i = 0; i < len; i++)
      cout << p[i] << " ";
}
```

5.4 Bor

```
// Построениебора, поисксловвлексиграфическомпорядке(dfs)

int K = 26;
// int MAXN = 10;
int MAXN = 2 * 1e5 + 1;

struct vertex
{
   vector<int> next;
   vector<int> count_v;
   bool leaf;
};

vector<vertex> t(MAXN);
int sz;

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

```
    t[v].leaf = true;
}

string dfs(int k)
{
    string result = "";
    int init = 0;
    while (k != 0)
    {
        int temp = 0;
        for (int i = 0; i < t[init].next.size(); i++)
        {
            if (t[init].count_v[i] && t[init].count_v[i] +
                 temp >= k)
            {
                init = t[init].next[i];
                k -= temp;
                if (t[init].leaf)
                {
                    k--;
                }
                result += char(i + 'a');
                break;
            }
            else if (t[init].count_v[i])
            {
                temp += t[init].count_v[i];
            }
        }
    }

    return result;
}

signed main()
{
    long long _t = 1;
    // cin >> _t;
    t[0].next.assign(K, -1);
    t[0].count_v.assign(K, 0);
    sz = 1;
    for (int _i = 0; _i < _t; _i++)
    {
        int n = 0;
        cin >> n;
        for (int i = 0; i < n; i++)
        {
            string s = "";
            cin >> s;
            bool flag = true;
            for (int i = 0; i < s.size(); i++)
            {
                if (!isdigit(s[i]))
                {
                    flag = false;
                    break;
                }
            }
            if (flag)
            {
                int k = stoi(s);
                cout << dfs(k) << endl;
            }
            else
            {
                add_string(s);
            }
        }
    }
    return 0;
}
```

6 Dynamic Programming

6.1 Increasing Subsequence

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    int n;
    std::cin >> n;
    std::vector<int> arr(n);
    std::copy_n(std::istream_iterator<int>(std::cin), n, arr
         .begin());

    std::vector<int> cur_longest_subsequence = {arr[0]};
    std::vector<int> longest_subs_in_position(n, 1);

    for (int i = 1; i < arr.size(); ++i) {
        if (cur_longest_subsequence.back() < arr[i]) {
            cur_longest_subsequence.emplace_back(arr[i]);
            longest_subs_in_position[i] =
                 cur_longest_subsequence.size();
        } else {
            auto it = std::lower_bound(
                 cur_longest_subsequence.begin(),
                                cur_longest_subsequence.end
                                     (), arr[i]);
            *it = arr[i];
            longest_subs_in_position[i] = std::distance(
                 cur_longest_subsequence.begin(), it) + 1;
        }
    }

    int length_of_lis = (int)cur_longest_subsequence.size();

    // Print longest subsequence
    std::cout << length_of_lis << "\n";
    std::vector<int> longest_subsequence;

    for (int i = (int)arr.size() - 1; i >= 0; --i) {
        if (longest_subs_in_position[i] == length_of_lis) {
            longest_subsequence.push_back(arr[i]);
            length_of_lis--;
        }
    }

    std::reverse(longest_subsequence.begin(),
         longest_subsequence.end());

    for (const auto& elem : longest_subsequence) {
        std::cout << elem << " ";
    }

    std::cout << std::endl;

    return 0;
}
```

6.2 General Backpack

```
#include <algorithm>
#include <iostream>
#include <vector>

using matrix = std::vector<std::vector<int>>;

int knapsack(int max_weight, const std::vector<int> &
     weights,
          const std::vector<int> &values,
          std::vector<int> &selected_indices) {
    int n = (int)weights.size();
    matrix dp(n + 1, std::vector<int>(max_weight + 1));
    matrix selected(n + 1, std::vector<int>(max_weight + 1,
         0));
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= max_weight; ++j) {
            if (weights[i - 1] <= j) {
                dp[i][j] = std::max(values[i - 1] + dp[i - 1][
                     j - weights[i - 1]],
                                    dp[i - 1][j]);
                if (values[i - 1] + dp[i - 1][j - weights[i -
                     1]] > dp[i - 1][j]) {
                    selected[i][j] = 1;
                }
            } else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }

    int i = n;
    int j = max_weight;
    while (i > 0 && j > 0) {
        if (selected[i][j] == 1) {
```

```
            selected_indices.push_back(i - 1);
            j -= weights[i - 1];
        }
        i--;
    }
    return dp[n][max_weight];
}

int main() {
    // At the entrance we get the number of items, the
        capacity of the backpack,
    // then the weight and value of the items
    int n, max_weight;
    std::cin >> n >> max_weight;
    std::vector<int> weights(n), values(n);
    for (int i = 0; i < n; ++i) {
        std::cin >> weights[i] >> values[i];
    }

    std::vector<int> selected_indices;
    int max_value = knapsack(max_weight, weights, values,
        selected_indices);
    std::cout << max_value << "\n";
    std::sort(selected_indices.begin(), selected_indices.end
        ());

    for (int index : selected_indices) {
        std::cout << index + 1 << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

6.3 K Elements Backpack

```
#include <algorithm>
#include <iostream>
#include <vector>

using matrix = std::vector<std::vector<int>>;

int knapsack(int max_weight, const std::vector<int> &
    weights,
             const std::vector<int> &values,
             std::vector<int> &selected_indices, int k) {
    int n = (int)weights.size();
    matrix dp(n + 1, std::vector<int>(max_weight + 1));
    matrix selected(n + 1, std::vector<int>(max_weight + 1,
        0));
    for (int i = 1; i <= std::min(n, k); ++i) {
        for (int j = 1; j <= max_weight; ++j) {
            if (weights[i - 1] <= j) {
                dp[i][j] = std::max(values[i - 1] + dp[i - 1][
                    j - weights[i - 1]],
                                    dp[i - 1][j]);
                if (values[i - 1] + dp[i - 1][j - weights[i -
                    1]] > dp[i - 1][j]) {
                    selected[i][j] = i;
                }
            } else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }

    int i = std::min(n, k);
    int j = max_weight;
    while (i > 0 && j > 0) {
        if (selected[i][j] != 0) {
            selected_indices.push_back(selected[i][j] - 1);
            j -= weights[selected[i][j] - 1];
        }
        i--;
    }
    return dp[std::min(n, k)][max_weight];
}

int main() {
    int n, max_weight, k;
    std::cin >> n >> max_weight >> k;
    std::vector<int> weights(n), values(n);
    for (int i = 0; i < n; ++i) {
        std::cin >> weights[i] >> values[i];
    }

    std::vector<int> selected_indices;
    int max_value = knapsack(max_weight, weights, values,
        selected_indices, k);
    std::cout << max_value << "\n";
    std::sort(selected_indices.begin(), selected_indices.end
        ());

    for (int index : selected_indices) {
        std::cout << index + 1 << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

6.4 Count Coin Changes

```
int change(int amount, const std::vector<int>& coins) {
    int numCoins = coins.size();
    std::vector<std::vector<int>> wayToChange(numCoins, std
        ::vector<int>(amount + 1, 0));

    for (int i = 0; i < numCoins; i++) {
        wayToChange[i][0] = 1;
    }

    for (int i = numCoins - 1; i >= 0; i--) {
        for (int j = 1; j <= amount; j++) {
            if (j - coins[i] >= 0) {
                wayToChange[i][j] += wayToChange[i][j - coins[
                    i]];
            }
            if (i + 1 < numCoins) {
                wayToChange[i][j] += wayToChange[i + 1][j];
            }
        }
    }

    return wa
```

6.5 Count Palindromes

```
int palindromes(std::string s) {
    int n = s.length();
    std::vector<std::vector<int>> dp(n, std::vector<int>(n,
        0));
    std::vector<std::vector<int>> pal(n, std::vector<int>(n,
        1));

    for (int i = n - 1; i >= 0; i--) {
        for (int j = i; j < n; j++) {
            if (i == j) {
                dp[i][j] = 1;
            } else {
                pal[i][j] = pal[i + 1][j - 1] && (s[i] == s[j
                    ]);
                dp[i][j] = dp[i][j - 1] + dp[i + 1][j] - dp[i
                    + 1][j - 1] + pal[i][j];
            }
        }
    }

    return dp[0][n - 1];
}
```

6.6 Longest Common Subsequence

```
int longestCommonSubsequence(const std::string& text1,
    const std::string& text2) {
      int n = text1.length();
      int m = text2.length();
      std::vector<std::vector<int>>
          longestSubseqInPosition(n + 1, std::vector<int
          >(m + 1, 0));

      for (int i = n - 1; i >= 0; i--) {
          for (int j = m - 1; j >= 0; j--) {
              if (text1[i] == text2[j]) {
```

```
                longestSubseqInPosition[i][j] = 1 +
                     longestSubseqInPosition[i + 1][j +
                     1];
            } else {
                longestSubseqInPosition[i][j] = std::max(
                    longestSubseqInPosition[i][j + 1],
                    longestSubseqInPosition[i + 1][j]
                );
            }
        }
    }

    return longestSubseqInPosition[0][0];
}
```

6.7 Pyramid

```
#include <iostream>
#include <vector>

int pyramid(int n) {
    int m = 0, result = 0;
    std::vector<std::vector<int>> mass(n + 1, std::vector<
         int>(n + 1, 0));
    mass[0][0] = 1;

    for (int i = 1; i < n + 1; i++) {
        for (int j = 1; j < n + 1; j++) {
            if (j > i) continue;
            for (int m = 0; m < j; m++) {
                mass[i][j] += mass[i - j][m];
            }
        }
    }

    for (int i = 1; i < n + 1; i++) {
        result += mass[n][i];
    }

    return result;
}
```

6.8 Domino 1

```
#pragma GCC optimize("O2,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
// const int N = 1e4;

bool compare(int& _i, int& _j, int& size) {
    int count = 0;
    bitset<16> j_prof = _j, i_prof = _i;
    for (int i = 0; i < size; i++) {
        if (j_prof[i] && !i_prof[i]) {
            if (count % 2 != 0) {
                return false;
            }
            else {
                count = 0;
                continue;
            }
        }

        if (j_prof[i] && i_prof[i]) {
            return false;
        }

        if (!i_prof[i]) {
            count++;
            continue;
        }

        if (i_prof[i]) {
            if (count % 2) {
                return false;
            }
            else {
                count = 0;
                continue;
            }
        }
    }
    return !(count % 2);
}

bool lastOrNo(int& i, int& size) {
    bitset<16> a = i;
    int count = 0;
    for (int j = 0; j < size; j++) {
        if (!a[j]) {
            count++;
        }
        else if (a[j]) {
            if (count % 2) {
                return false;
            }
            else {
                count = 0;
                continue;
            }
        }
    }

    return !(count % 2);
}

long long dp[4096][4096];

signed main() {
    int m, n;
    cin >> n >> m;
    if (n % 2 && m % 2) {
        cout << 0 << endl;
        return 0;
    }
    if (n > m) {
        swap(m, n);
    }
    int size_N = (1 << m);
    int N = (1 << n);

    dp[0][0] = 1;
    for (int k = 1; k < m; k++) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                dp[k][i] += dp[k - 1][j] * compare(j, i, n);
            }
        }
    }

    long long ans = 0;
    for (int i = 0; i < N; i++) {
        if (lastOrNo(i, n)) {
            ans += dp[m - 1][i];
        }
    }

    cout<<ans;

}
```

6.9 Domino 2

```
ull dp[20][20][5000];
ull binpow(ull a, unsigned long long int b, ull p = 0) {
    ull 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;
}
int main() {
    ull n, m, i, j, k, l2, r;
    cin >> n >> m;
    char tiling[20][20];
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)tiling[i][j] = '.';
    }
    for (k = 0; k < n + 1;k++) {
        for (j = 0; j < m; j++) {
            for (ull mask = 0; mask < (ull)pow(2, m); mask++)
                 {
                if (k != 0 || j != 0 || mask != 0)dp[k][j][
                     mask] = 0;
                else dp[k][j][mask] = 1;
```

```
            }
        }
    }
    for (k = 0; k < n; k++) {
        for (j = 0; j < m; j++) {
            for (ull mask = 0; mask < (ull)pow(2, m); mask++)
                {
                if (k < n - 1 && tiling[k][j] == '.' && tiling
                    [k + 1][j] == '.' && (mask & (1 << j))
                    == 0)
                    dp[k + ((j + 1) / m)][(j + 1) % m][mask +
                        (1 << j)] += dp[k][j][mask];
                if (j < m - 1 && tiling[k][j] == '.' && tiling
                    [k][j + 1] == '.' && (mask & (3 << j))
                    == 0)
                    dp[k + ((j + 1) / m)][(j + 1) % m][mask +
                        (2 << j)] += dp[k][j][mask];
                if (((1 << j) & mask) != 0 || tiling[k][j] !=
                    '.')
                    dp[k + ((j + 1) / m)][(j + 1) % m][(mask |
                         (1 << j)) - (1 << j)] += dp[k][j][
                        mask];
            }
        }
    }
    cout << binpow(2, n * m / 2, 1000000007) * (dp[n][0][0]
        % 1000000007) % 1000000007;
    return 0;
}
```

7 Graphs

7.1 Articulation Point

```
#include <iostream>
#include <set>
#include <vector>

using namespace std;

vector<vector<int>> g;
vector<bool> used;
int timer = 0;
vector<int> tin, fup;
set<int> result;

void dfs(int v, int p = -1) {
  used[v] = true;
  tin[v] = fup[v] = timer++;
  int children = 0;
  for (size_t i = 0; i < g[v].size(); ++i) {
    int to = g[v][i];
    if (to == p)
      continue;
    if (used[to])
      fup[v] = min(fup[v], tin[to]);
    else {
      dfs(to, v);
      fup[v] = min(fup[v], fup[to]);
      if (fup[to] >= tin[v] && p != -1)
        result.insert(v);
      children++;
    }
  }
  if (p == -1 && children > 1)
    result.insert(v);
}

int main() {
  int n, m, k;
  cin >> n >> m;
  g.resize(n);
  used.assign(n, false);
  tin.resize(n);
  fup.resize(n);

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

  for (int i = 0; i < n; i++) {
    dfs(i);
  }

  for (int it : result) {
    cout << it + 1 << " ";
  }

  return 0;
}
```

7.2 Dfs

```
void dfs(vector<vector<int>> &adj, int start, vector<bool>
     &visited) {
  stack<int> s;
  visited[start] = true;
  s.push(start);

  while (!s.empty()) {
    int current = s.top();
    s.pop();

    cout << current + 1 << " ";

    for (int neighbor : adj[current]) {
      if (!visited[neighbor]) {
        visited[neighbor] = true;
        s.push(neighbor);
      }
    }
  }
}
```

7.3 Bfs

```
void bfs(vector<vector<int>> &adj, int start, vector<bool>
     &visited) {
  queue<int> q;
  visited[start] = true;
  q.push(start);

  while (!q.empty()) {
    int current = q.front();
    q.pop();

    cout << current + 1 << " ";

    for (int neighbor : adj[current]) {
      if (!visited[neighbor]) {
        visited[neighbor] = true;
        q.push(neighbor);
      }
    }
  }
}
```

7.4 Find Bridges

```
#include <algorithm>
#include <iostream>
#include <vector>

using namespace std;

vector<vector<int>> g;
vector<bool> used;
int timer = 0;
vector<int> tin, fup;
vector<pair<int, int>> result;

void dfs(int v, int p = -1) {
  used[v] = true;
  tin[v] = fup[v] = timer++;
  for (int i = 0; i < g[v].size(); i++) {
    int to = g[v][i];
    if (to == p)
```

```
      continue;
    if (used[to])
      fup[v] = min(fup[v], tin[to]);
    else {
      dfs(to, v);
      fup[v] = min(fup[v], fup[to]);
      if (fup[to] > tin[v] && count(g[v].begin(), g[v].end()
          , to) == 1) {
        result.push_back({min(v, to), max(to, v)});
      }
    }
  }
}

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

int main() {
  int n;
  cin >> n;
  g.resize(n);
  used.assign(n, false);
  tin.resize(n);
  fup.resize(n);
  cin.ignore();

  for (int i = 0; i < n; i++) {
    int current = 0, count = 0;
    cin >> current >> count;
    for (int j = 0; j < count; j++) {
      int temp = 0;
      cin >> temp;
      g[current].push_back(temp);
    }
  }

  find_bridges(n);

  if (!result.empty()) {
    sort(result.begin(), result.end());
    for (auto bridge : result) {
      cout << bridge.first << " " << bridge.second << endl;
    }
  } else {
    cout << "Empty" << endl;
  }

  return 0;
}
```

7.5 Components Of Strong Connectivity

```
#include <iostream>
#include <vector>

using namespace std;

vector<vector<int>> g, gr;
vector<bool> used;
vector<int> order, component;

void dfs1(int v) {
  used[v] = true;
  for (size_t i = 0; i < g[v].size(); ++i) {
    if (!used[g[v][i]]) {
      dfs1(g[v][i]);
    }
  }
  order.push_back(v);
}

void dfs2(int v) {
  used[v] = true;
  component.push_back(v);
  for (size_t i = 0; i < gr[v].size(); ++i) {
    if (!used[gr[v][i]]) {
      dfs2(gr[v][i]);
    }
  }
}

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

  g.resize(n);
  gr.resize(n);
  used.assign(n, false);

  for (int i = 0; i < n; i++) {
    int a = 0, b = 0;
    cin >> a >> b;
    g[a].push_back(b);
    gr[b].push_back(a);
  }

  for (int i = 0; i < n; ++i) {
    if (!used[i]) {
      dfs1(i);
    }
  }

  used.assign(n, false);

  for (int i = n - 1; i >= 0; --i) {
    int v = order[i];
    if (!used[v]) {
      dfs2(v);
      for (int j = 0; j < component.size(); j++) {
        cout << component[j] << " ";
      }
      cout << '\n';
      component.clear();
    }
  }

  return 0;
}
```

7.6 Connected Components

```
#include <iostream>
#include <vector>

using namespace std;

void dfs(vector<vector<int>> &mass, vector<bool> &used, int
      vertex) {
  used[vertex] = true;
  for (int i = 0; i < mass[vertex].size(); i++) {
    int neighbor = mass[vertex][i];
    if (!used[neighbor]) {
      dfs(mass, used, neighbor);
    }
  }
}

int main() {
  int n = 0, m = 0;
  cin >> n >> m;

  vector<vector<int>> mass(n);
  vector<bool> used(n, false);

  for (int i = 0; i < m; i++) {
    int first, second;
    cin >> first >> second;
    first--;
    second--;
    mass[first].push_back(second);
    mass[second].push_back(first);
  }

  int result = 0;
  for (int i = 0; i < n; i++) {
    if (!used[i]) {
      dfs(mass, used, i);
      result++;
    }
  }

  cout << result << '\n';
```

```
  return 0;
}
```

7.7 Find Cycles

```
#include <iostream>
#include <vector>

using namespace std;

int cycle_start = -1, cycle_end = 0;
vector<int> p;

bool dfs(vector<vector<int>> &g, vector<bool> &used, vector
     <int> &color,
       int vertex) {
  color[vertex] = 1;
  for (int i = 0; i < g[vertex].size(); i++) {
    int to = g[vertex][i];
    if (color[to] == 0) {
      if (dfs(g, used, color, to)) {
        p[to] = vertex;
        return true;
      }
    } else if (color[to] == 1) {
      cycle_start = to;
      cycle_end = vertex;
      return true;
    }
  }
  color[vertex] = 2;
  return false;
}

int main() {
  int n = 0, m = 0;
  cin >> m >> n;

  vector<vector<int>> mass(n);
  vector<bool> used(n, false);
  vector<int> color(n, 0);
  vector<int> cycle;

  p.assign(n, -1);

  for (int i = 0; i < m; i++) {
    int first, second;
    cin >> first >> second;
    first--;
    second--;
    mass[first].push_back(second);
  }

  for (int i = 0; i < n; i++) {
    if (dfs(mass, used, color, i)) {
      break;
    }
  }

  if (cycle_start == -1) {
    cout << "No" << endl;
  } else {
    cout << "Yes" << endl;
    cycle.push_back(cycle_start);
    for (int v = cycle_end; v != cycle_start; v = p[v]) {
      cycle.push_back(v);
    }
    cycle.push_back(cycle_start);
    reverse(cycle.begin(), cycle.end());
    for (int i = 0; i < cycle.size(); i++) {
      cout << cycle[i] + 1 << " ";
    }
    cout << endl;
  }

  return 0;
}
```

7.8 Cruscal

```
#include <algorithm>
#include <iostream>
#include <random>
#include <vector>

using namespace std;

int findRoot(vector<int> &parent, int v) {
  if (parent[v] == v) {
    return v;
  } else {
    parent[v] = findRoot(parent, parent[v]);
    return parent[v];
  }
}

bool connected(vector<int> &parent, int v1, int v2) {
  return findRoot(parent, v1) == findRoot(parent, v2);
}

void merge(vector<int> &parent, int v1, int v2) {
  int r1 = findRoot(parent, v1);
  int r2 = findRoot(parent, v2);
  if (r1 != r2) {
    if (rand() % 2 == 0) {
      parent[r1] = r2;
    } else {
      parent[r2] = r1;
    }
  }
}

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

  vector<vector<int>> mst(n);
  vector<int> parent(n);
  vector<pair<int, pair<int, int>>> G(m);

  for (int i = 0; i < m; i++) {
    int v, u, cost;
    cin >> v >> u >> cost;
    G[i] = {cost, {v - 1, u - 1}};
  }

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

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

  int cost = 0;
  int all_sum = 0;

  for (int i = 0; i < m; i++) {
    int a = G[i].second.first;
    int b = G[i].second.second;
    int l = G[i].first;

    if (!connected(parent, a, b)) {
      mst[a].push_back(l + 1);
      mst[b].push_back(l + 1);
      merge(parent, a, b);
      all_sum += l;
    }
  }

  cout << all_sum << endl;

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < mst[i].size(); j++) {
      cout << i + 1 << " " << mst[i][j] << endl;
    }
  }

  return 0;
}
```

7.9 Prim Algorithm

```
#include <algorithm>
#include <iostream>
#include <map>
#include <vector>

using namespace std;
```

```
int main() {
  int mass[100001];
  vector<int> check;
  vector<int> result;
  vector<vector<pair<int, int>>> mass(100001);

  int n, m;
  cin >> n >> m;

  for (int i = 0; i < n - 1; i++) {
    check.push_back(i + 1);
  }

  result.push_back(0);

  for (int i = 0; i < m; i++) {
    int first, second, third;
    cin >> first >> second >> third;
    first--;
    second--;
    mass[first].push_back(make_pair(second, third));
    mass[second].push_back(make_pair(first, third));
  }

  while (!check.empty()) {
    int temp = 10e5;
    int top = 10e5;
    int parent = 0;

    for (int i = 0; i < result.size(); i++) {
      for (int j = 0; j < mass[result[i]].size(); j++) {
        if (mass[result[i]][j].second < temp &&
            find(check.begin(), check.end(), mass[result[i]][
                j].first) !=
                check.end()) {
          temp = mass[result[i]][j].second;
          top = mass[result[i]][j].first;
          parent = result[i];
        }
      }
    }

    result.push_back(top);

    int count = 0;
    for (int k = 0; k < check.size(); k++) {
      if (check[k] == top) {
        count = k;
        break;
      }
    }

    check.erase(check.begin() + count);

    int sum = 0;
    count = 0;
    temp = 10e5;
    top = 10e5;
  }

  cout << sum << endl;

  return 0;
}
```

7.10 Lca Using Segment Tree

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <unordered_map>
#include <map>

using namespace std;

typedef vector < vector<int> > graph;
typedef vector<int>::const_iterator const_graph_iter;


vector<int> lca_h, lca_dfs_list, lca_first, lca_tree;
vector<char> lca_dfs_used;

void lca_dfs(const graph& g, int v, int h = 1)
{
    lca_dfs_used[v] = true;
    lca_h[v] = h;
    lca_dfs_list.push_back(v);
    for (const_graph_iter i = g[v].begin(); i != g[v].end();
         ++i)
        if (!lca_dfs_used[*i])
        {
            lca_dfs(g, *i, h + 1);
            lca_dfs_list.push_back(v);
        }
}

void lca_build_tree(int i, int l, int r)
{
    if (l == r)
        lca_tree[i] = lca_dfs_list[l];
    else
    {
        int m = (l + r) >> 1;
        lca_build_tree(i + i, l, m);
        lca_build_tree(i + i + 1, m + 1, r);
        if (lca_h[lca_tree[i + i]] < lca_h[lca_tree[i + i +
            1]])
            lca_tree[i] = lca_tree[i + i];
        else
            lca_tree[i] = lca_tree[i + i + 1];
    }
}

void lca_prepare(const graph& g, int root)
{
    int n = (int)g.size();
    lca_h.resize(n);
    lca_dfs_list.reserve(n * 2);
    lca_dfs_used.assign(n, 0);

    lca_dfs(g, root);

    int m = (int)lca_dfs_list.size();
    lca_tree.assign(lca_dfs_list.size() * 4 + 1, -1);
    lca_build_tree(1, 0, m - 1);

    lca_first.assign(n, -1);
    for (int i = 0; i < m; ++i)
    {
        int v = lca_dfs_list[i];
        if (lca_first[v] == -1)
            lca_first[v] = i;
    }
}

int lca_tree_min(int i, int sl, int sr, int l, int r)
{
    if (sl == l && sr == r)
        return lca_tree[i];
    int sm = (sl + sr) >> 1;
    if (r <= sm)
        return lca_tree_min(i + i, sl, sm, l, r);
    if (l > sm)
        return lca_tree_min(i + i + 1, sm + 1, sr, l, r);
    int ans1 = lca_tree_min(i + i, sl, sm, l, sm);
    int ans2 = lca_tree_min(i + i + 1, sm + 1, sr, sm + 1, r
        );
    return lca_h[ans1] < lca_h[ans2] ? ans1 : ans2;
}

int lca(int a, int b)
{
    int left = lca_first[a],
        right = lca_first[b];
    if (left > right) swap(left, right);
    return lca_tree_min(1, 0, (int)lca_dfs_list.size() - 1,
         left, right);
}
int main() {
    // чтениеграфа
    int n;
    cin >> n;
    vector <vector<int>> graph(n, vector<int>());
    int top, m, tmp;
    for (int i = 0; i < n; i++) {
        cin >> top >> m;
        top--;
        for (int j = 0; j < m; j++) {
            cin >> tmp;
            graph[top].push_back(tmp - 1);
        }
    }
```

```
    // выполнениепрепроцессинга
    lca_prepare(graph, 0);

    // чтениеиответыназапросы
    int q;
    cin >> q;
    int from, to;
    for (int i = 0; i < n; i++) {
        cin >> from >> to;
        from--;
        to--;
    }
}
```

7.11 Algo Floyd

```
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)
        continue;
      d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
    }
  }
}
```

8 Miscellaneous

8.1 Ternary Search

```
double phi = 1 + (1 + sqrt(5)) / 2;
// continuous ternary search
double cont_ternary_search(double l, double r)
{
    double m1 = l + (r - l) / phi, m2 = r - (r - l) / 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) / phi;
            f1 = f(m1);
        }
        else {
            l = m1;
            m1 = m2;
            f1 = f2;
            m2 = r - (r - l) / phi;
            f2 = f(m2);
        }
    }
    return f((l + r) / 2);
}
// discrete ternary search
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)));
}
```

8.2 Binary Search Float

```
double sqrtnWithBinSearch(double a, int n) {
        double l = 0, r = a;
        for (int _ = 0; _ < 200; _++) {
                double mid = (r + l) / 2;
                if (pow(mid, n) > a) {
                        r = mid;
                }
                else l = mid;
        }
        return l;
}
```