Contents

1 Number theory

1.1 Bigmod, Inverse Mod

```
ll bigmod(ll a, ll b, ll mod){
    ll res = 1;
    while (b > 0){
        if (b & 1)  res = (res * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}

ll inverse_mod(ll a, ll b) {
 return 1 < a ? b - inverse_mod(b % a, a) * b / a : 1;
}
ll inv[N];  // inverse modulo pre calculate
void imod() {
 inv[1] = 1;
 for (ll i = 2; i < N; i++) inv[i] = (mod - (mod / i) * inv[mod % i]) % mod;
}
// another way to find inverse modulo of n is
ll inv_of_n = bigmod(n, mod - 2, mod);
```

1.2 inclusion-exclusion - number of multiples of divs in [l, r]

```
ll iep(int l, int r) {
 if (l > r) return 0;
 ll sum = 0, sz = divs.size();
 for (ll j = 1; j < (1LL << sz); j++) {
   ll gun = 1, one = 0;
   for (ll i = 0; i < sz; i++)
     if (j & (1LL << i)) {
       one++; gun *= divs[i];
      }
   ll mult = (r / gun) - ((l - 1) / gun);
   if (one % 2 == 1) sum += mult;
   else sum -= mult;
 }
 return sum;
}

```

1.3 Linear Sieve

```
const ll N = 10000000;
vector<ll> lp(N+1), pr;

for (ll i=2; i <= N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i; pr.push_back(i);
    }
    for (ll j=0; j < (ll)pr.size() && pr[j] <= lp[i] && i*pr[j] <= N; ++j) {
        lp[i * pr[j]] = pr[j];
    }
}
```

1.4 Phi Function[n]/ Phi Sieve[1 to n]

```
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)  n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}

void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++)
        if (phi[i] == i)
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
}
```

1.5 Extended Euclid

```
ll extended_euclid(ll a, ll b, ll &x, ll &y) {
  if (b == 0) {
    x = 1; y = 0;
    return a;
  }
```

```
  ll x1, y1;
  ll d = extended_euclid(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
```

1.6 MatExpo

```
const ll mod = 1e9 + 7;
#define REP(i, n) for(int i = 0; i < (n); i++)
ll sz ;
struct Matrix {
        vector<vector<ll>> a = vector<vector<ll>>(sz,vector<ll>(sz)) ;
        Matrix() {
                REP(i, sz) {
                        REP(j, sz) {
                                a[i][j] = 0;
                        }
                }
        }
        Matrix operator *(Matrix other) {
                Matrix product = Matrix();
                REP(i, sz) {
                        REP(j, sz) {
                                REP(k, sz) {
                                        product.a[i][k] += a[i][j] * other.a[j][k];
                                        product.a[i][k] %= mod;
                                }
                        }
                }
                return product;
        }
};
Matrix expo_power(Matrix a, long long n) {
        Matrix res = Matrix();
        for (ll i =0 ;i< sz ; i++) res.a[i][i] = 1 ;
        while(n) {
                if(n % 2) {
                        res = res * a;
                }
                n /= 2;
                a = a * a;
        }
        return res;
}
int main() {
        sz = 2 ;ll k = 7 ;
                Matrix single = Matrix();
        single.a[0][0] = 1 ; single.a[0][1] = 1 ; single.a[1][0] = 1 ; single.a[1][1] = 0 ;
        Matrix ans = expo_power(single,k);
}


```

1.7 CRT

```
using T = __int128;
// ax + by = __gcd(a, b)
// returns __gcd(a, b)
T extended_euclid(T a, T b, T &x, T &y) {
  T xx = y = 0;
  T yy = x = 1;
  while (b) {
    T q = a / b;
    T t = b; b = a % b; a = t;
    t = xx; xx = x - q * xx; x = t;
    t = yy; yy = y - q * yy; y = t;
  }
  return a;
}
// finds x such that x % m1 = a1, x % m2 = a2. m1 and m2 may not be coprime
// here, x is unique modulo m = lcm(m1, m2). returns (x, m). on failure, m = -1.
pair<T, T> CRT(T a1, T m1, T a2, T m2) {
  T p, q;
  T g = extended_euclid(m1, m2, p, q);
  if (a1 % g != a2 % g) return make_pair(0, -1);
  T m = m1 / g * m2;
  p = (p % m + m) % m;
  q = (q % m + m) % m;
  return make_pair((p * a2 % m * (m1 / g) % m + q * a1 % m * (m2 / g) % m) %  m, m);
}
//cout << (int)CRT(1, 31, 0, 7).first << '\n';
```

1.8 Miller Rabin Primality Test

```
/* Miller Rabin Primality Test for <= 10^18 */
#define ll long long

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

```
        y = (y << 1) % c;
        b >>= 1;
    }
    return x % c;
}
ll fastPow(ll x, ll n, ll MOD) {
    ll ret = 1;
    while (n) {
        if (n & 1) ret = mulmod(ret, x, MOD);
        x = mulmod(x, x, MOD);
        n >>= 1;
    }
    return ret % MOD;
}
bool isPrime(ll n) {
    if (n == 2 || n == 3) return true;
    if (n == 1 || !(n & 1)) return false;
    ll d = n - 1;
    int s = 0;
    while (d % 2 == 0) {
        s++;
        d /= 2;
    }

    int a[9] = { 2, 3, 5, 7, 11, 13, 17, 19, 23 };
    for (int i = 0; i < 9; i++)  {
        if (n == a[i]) return true;
        bool comp = fastPow(a[i], d, n) != 1;
        if (comp) for (int j = 0; j < s; j++)  {
                ll fp = fastPow(a[i], (1LL << (ll)j) * d, n);
                if (fp == n - 1) {
                    comp = false;
                    break;
                }
            }
        if (comp) return false;
    }
    return true;
}
```

1.9 FFT

```
/***
* Multiply (7x^2 + 8x^1 + 9x^0) with (6x^1 + 5x^0)
* ans = 42x^3 + 83x^2 + 94x^1 + 45x^0
* A = {9, 8, 7}
* B = {5, 6}
* V = multiply(A,B)
* V = {45, 94, 83, 42}
***/
/*** Tricks
* Use vector < bool > if you need to check only the status of the sum
* Use bigmod if the power is over same polynomial && power is big
* Use long double if you need more precision
* Use long long for overflow
***/
typedef vector<int> vi;
const double PI = 2.0 * acos(0.0);
using cd = complex<double>;
void fft(vector<cd> &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) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < n; i += len) {
      cd w(1);
      for (int j = 0; j < len / 2; j++) {
        cd u = a[i + j], v = a[i + j + len / 2] * w;
        a[i + j] = u + v;
        a[i + j + len / 2] = u - v;
        w *= wlen;
      }
    }
  }
  if (invert) {
    for (cd &x : a)
      x /= n;
  }
}

void ifft(vector<cd> &p) { fft(p, 1); }

vi multiply(vi const &a, vi const &b) {
  vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
  int n = 1;
  while (n < a.size() + b.size())
    n <<= 1;
  fa.resize(n);
  fb.resize(n);
```

```

  fft(fa);
  fft(fb);
  for (int i = 0; i < n; i++)
    fa[i] *= fb[i];
  ifft(fa);

  vi result(n);
  for (int i = 0; i < n; i++)
    result[i] = round(fa[i].real());
  return result;
}
```

1.10 NTT

```
/**
Iterative Implementation of Number Theoretic Transform
Complexity: O(N log N)
Slower than regular fft
Possible Optimizations:
1. Remove trailing zeroes
2. Keep the mod const

Suggested mods (mod, root, inv, pw) :
7340033, 5, 4404020, 1<<20
13631489, 11799463,6244495, 1<<20
23068673, 177147,17187657, 1<<21
463470593, 428228038, 182429, 1<<21
415236097, 73362476, 247718523, 1<<22
918552577, 86995699, 324602258, 1<<22
998244353, 15311432, 469870224, 1<<23
167772161, 243, 114609789, 1<<25
469762049, 2187, 410692747, 1<<26
If required mod is not above, use nttdata function OFFLINE.
If pw=1<<k, a polynomial can have at most (1<<k) degree.
**/
#include<bits/stdc++.h>
using namespace std;

namespace ntt {
    int N;
    vector<int> perm;
    vector<int>wp[2][30];
    const int mod = 998244353, root = 15311432, inv = 469870224, pw = 1<<23;

    int power(int a, int p) {
        if (p==0)   return 1;
        int ans = power(a, p/2);
        ans = (ans * 1LL * ans)%mod;
        if (p%2)    ans = (ans * 1LL * a)%mod;
        return ans;
    }

    void precalculate() {
        perm.resize(N);
        perm[0] = 0;

        for (int k=1; k<N; k<<=1) {
            for (int i=0; i<k; i++) {
                perm[i] <<= 1;
                perm[i+k] = 1 + perm[i];
            }
        }

        for (int b=0; b<2; b++) {
            for (int idx = 0, len = 2; len <= N; idx++, len <<= 1) {
                int factor = b ? inv : root;
                for (int i = len; i < pw; i <<= 1)
                    factor = (factor*1LL*factor)%mod;

                wp[b][idx].resize(N);
                wp[b][idx][0] = 1;
                for (int i = 1; i < len; i++)
                    wp[b][idx][i] = (wp[b][idx][i-1]*1LL*factor)%mod;
            }
        }
    }

    void fft(vector<int> &v, bool invert = false) {
        if (v.size() != perm.size()) {
            N = v.size();
            assert(N && (N&(N-1)) == 0);
            precalculate();
        }

        for (int i=0; i<N; i++)
            if (i < perm[i])
                swap(v[i], v[perm[i]]);

        for (int idx = 0, len = 2; len <= N; idx++, len <<= 1) {
            for (int i=0; i<N; i+=len) {
                for (int j=0; j<len/2; j++) {
                    int x = v[i+j];
                    int y = (wp[invert][idx][j]*1LL*v[i+j+len/2])%mod;
                    v[i+j] = (x+y>=mod ? x+y-mod : x+y);
                    v[i+j+len/2] = (x-y>=0 ? x-y : x-y+mod);
                }
            }
```

```
        }
        if (invert) {
            int n1 = power(N, mod-2);
            for (int &x : v) x = (x*1LL*n1)%mod;
        }
    }

    vector<int> multiply(vector<int> a, vector<int> b) {
        int n = 1;
        while(a.back() == 0 && !a.empty()) a.pop_back();
        while(b.back() == 0 && !b.empty()) b.pop_back();

        while (n < a.size()+ b.size())  n<<=1;
        a.resize(n), b.resize(n);

        fft(a), fft(b);
        for (int i=0; i<n; i++) a[i] = (a[i] * 1LL * b[i])%mod;
        fft(a, true);
        return a;
    }
};

const int M = 998244353, N = 2e6;
int main() {
    std::ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);
    vector<int> a(N), b(N);
    long long asum = 0, bsum = 0, csum = 0;
    for (int i=0; i<N; i++) asum += (a[i] = rand()%M);
    for (int i=0; i<N; i++) bsum += (b[i] = rand()%M);

    vector<int> c = NTT::multiply(a, b);
    for (int x: c)   csum += x;
    cout<<csum<<endl;
}

int power(int a, int p, int mod) {
    if (p==0)   return 1;
    int ans = power(a, p/2, mod);
    ans = (ans * 1LL * ans)%mod;
    if (p%2)    ans = (ans * 1LL * a)%mod;
    return ans;
}
/** Find primitive root of p assuming p is prime.
if not, we must add calculation of phi(p).
Complexity : O(Ans * log (phi(n)) * log n + sqrt(p)) (if exists)
             O(p * log (phi(n)) * log n + sqrt(p))   (if does not exist)
Returns -1 if not found.
*/
int primitive_root(int p) {
    if (p == 2) return 1;
    vector<int> factor;
    int phi = p-1,  n = phi;

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

    if (n>1)  factor.push_back(n);

    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (int i=0; i<factor.size() && ok; ++i)
            ok &= power(res, phi/factor[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
/**
  Generates necessary info for NTT (for offline usage :3).
  Returns maximum k such that 2^k % mod = 1,
  NTT can only be applied for arrays not larger than this size.
  mod MUST BE PRIME!!!!!
  We use the fact that if primes have the form p=c*2^k+1,
  there always exists the 2^k-th root of unity.
  It can be shown that g^c is such a 2^k-th root
  of unity, where g is a primitive root of p.
*/
int nttdata(int mod, int &root, int &inv, int &pw) {
    int c = 0, n = mod-1;
    while (n%2 == 0) c++, n/=2;
    pw = (mod-1)/n;
    int g = primitive_root(mod);
    if(g == -1) return -1; // No primitive root exists
    root = power(g, n, mod);
    inv = power(root, mod-2, mod);
    return c;
}
```

2 Data Structure

2.1 BIT

```
ll bit[N]; //Add(0)--> RTE
void Add(ll x,ll v) { for(;x < N;x += x & (-x)) bit[x] = (bit[x] + v) ; }
ll Sum(ll x) { ll r = 0; for(;x;x -= x & (-x)) r = (r + bit[x]) ; return r; }
```

```

```

2.2 Sack

```
int cnt[maxn];
bool big[maxn];
void add(int v, int p, int x){
    cnt[ col[v] ] += x;
    for(auto u: g[v])
        if(u != p && !big[u])
            add(u, v, x)
}
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
          mx = sz[u], bigChild = u;
    for(auto u : g[v])
        if(u != p && u != bigChild)
            dfs(u, v, 0);  // run a dfs on small childs and clear them from cnt
    if(bigChild != -1)
        dfs(bigChild, v, 1), big[bigChild] = 1;  // bigChild marked as big and not cleared from cnt
    add(v, p, 1);
    //now cnt[c] is the number of vertices in subtree of vertex v that has color c. You can answer the queries easily.
    if(bigChild != -1)
        big[bigChild] = 0;
    if(keep == 0)
        add(v, p, -1);
}
```

2.3 Centroid Decomposition

```
/// decompose(1, -1)   //For 1 rooted tree

#define ll long long
#define pb push_back
const ll MAX = 1e5;
vector <ll> g[MAX + 9];
ll del[MAX + 9], sz[MAX + 9], par[MAX + 9], curSize;

void dfs(ll u, ll p)

{
    sz[u] = 1;
    for(ll i = 0; i < g[u].size(); i++) {
        ll nd = g[u][i];
        if(nd == p || del[nd])
            continue;
        dfs(nd, u);
        sz[u] += sz[nd];
    }
}

ll findCentroid(ll u, ll p)

{
    for(ll i = 0; i < g[u].size(); i++) {
        ll nd = g[u][i];
        if(nd == p || del[nd] || sz[nd] <= curSize / 2)
            continue;

        return findCentroid(nd, u);
    }
    return u;
}

void decompose(ll u, ll p)

{
    dfs(u, -1);
    curSize = sz[u];
    ll cen = findCentroid(u, -1);
    if(p == -1) p = cen;
    par[cen] = p, del[cen] = 1;

    for(ll i = 0; i < g[cen].size(); i++) {
        ll nd = g[cen][i];

        if(!del[nd])
            decompose(nd, cen);
    }
}
```

2.4 Lca with Sparse Table

```
int n, l, timer;
vector<vector<int>> adj, up;
vector<int> tin, tout;
void dfs(int v, int p) {
    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 (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }
    tout[v] = ++timer;
}
bool is_ancestor(int u, int v){
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}
int lca(int u, int v){
    if (is_ancestor(u, v)) return u;
    if (is_ancestor(v, u)) return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}
void preprocess(int root) {
    tin.resize(n); tout.resize(n);
    timer = 0; l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}

```

2.5 PBDS/Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using ordered_set = tree<T, null_type, less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;

    ordered_set <pair<int,int>> st;
        st.insert({10, 1});

        cout << (st.find_by_order(2) -> first) << endl; //print element in k-th
    index
        cout << st.order_of_key({10, 2}) << endl; //print number of items < k
        auto print = [&] () {
                for (auto z : st) cout << z.first << ' ';
        };
        print();
        st.erase(st.lower_bound({0, 1}));

// #define ordered_set tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update>
// #define ordered_set tree<int, null_type, less_equal<int>, rb_tree_tag,
    tree_order_statistics_node_update>
//less than or equal -> use             less_equal<int>
// #define ordered_set tree<pair<int,int>, null_type, pair<int,int>,
    rb_tree_tag, tree_order_statistics_node_update>
//to use multiset -> use pair<int, int> where first element is value and second
    element is index

```

2.6 DSU

```
void make_set(int v) {
    parent[v] = v;
}
int find_set(int v) {
    if (v == parent[v]) return v;
    return parent[v] = find_set(parent[v]);
}
void union_sets(int a, int b) {
    a = find_set(a); b = find_set(b);
    if (a != b)
        parent[b] = a;
}
```

2.7 Mo's Algo/ Sqrt decomposition

```
void remove(idx);  // TODO: remove value at idx from data structure
void add(idx);     // TODO: add value at idx from data structure
int get_answer();  // TODO: extract the current answer of the data structure
int block_size;
struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};
vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the range [cur_l, cur_r]
```

```
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}

```

3 Graph Theory

3.1 0-1 BFS

```
vector<int> d(n, INF);
d[s] = 0;
deque<int> q;
q.push_front(s);
while (!q.empty()) {
    int v = q.front();
    q.pop_front();
    for (auto edge : adj[v]) {
        int u = edge.first;
        int w = edge.second;
        if (d[v] + w < d[u]) {
            d[u] = d[v] + w;
            if (w == 1)
                q.push_back(u);
            else
                q.push_front(u);
        }
    }
}
```

3.2 Bellman Ford Algorithm to find Negative Cycle

```
//bellman ford with negative cycle print
struct edge {
  ll v, w;
};
const ll N = 3e3 + 6, inf = 1LL << 60;
ll n, m, dis[N], par[N];
vector<edge> g[N];

int bellman_ford() {
  lop(n + 1) dis[i] = inf;
  dis[1] = 0;
  int cy;

  lop(n + 1) {
    cy = -1;
    for (int u = 1; u <= n; u++) {
      for (auto z : g[u]) {
        ll v = z.v, w = z.w;
        if (dis[u] + w < dis[v]) {
          dis[v] = dis[u] + w;
          par[v] = u;
          cy = v; // if(u == n) negative cycle;
        }
      }
    }
  }
  return cy; //cy is a adjacent node or a node of negative cycle
}

int main() {
  cin >> n >> m;
  lop(m) {
    ll u, v, w;
    cin >> u >> v >> w;
    g[u].pb({v, w});
  }

  int x = bellman_ford();
  if (x == -1) {
    //no negative cycle
    return 0;
  }

//x can be not a part of cycle, so if we go through //path sometimes, x will be
↪  a node of cycle
  lop(n) x = par[x];
  vector<int> cycle;
  int i = x;
```

```
  while (i != x or cycle.size() <= 1) {
    cycle.pb(i);//retrieving cycle
    i = par[i];
  }
  cycle.pb(i);
  reverse(all(cycle));
  for (int z : cycle)
    cout << z << ' ';
  return 0;
}
```

3.3 Kruskal MST

```
struct edge {
	int u, v, w;
	bool operator < (const edge &b) const {
		return w > b.w;
	}
};

const int N = 2e5;
int n, m, par[N];
vector <edge> eg;

int findpar(int x) {
	return par[x] = par[x] == x ? x : findpar(par[x]);
}

void Union(int u, int v) {
	par[findpar(u)] = findpar(v);
}

int kruskal() {
	sort(eg.begin(), eg.end());
	iota(par, par + n, 0);

	int cost = 0, connected = 0;
	while (connected != n - 1) {
		edge z = eg.back();
		eg.pop_back();

		int x = findpar(z.u), y = findpar(z.v);
		if (x != y) {
			connected++;
			cost += z.w;
			Union(x, y);
		}
	}

	return cost;
}


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

	cin >> n >> m;
	for (int i = 0; i < m; i++) {
		int u, v, w;
		cin >> u >> v >> w;
		eg.push_back({u, v, w});
	}

	cout << kruskal() << '\n';

	return 0;
}
```

3.4 Articulation Bridge and Point

```
const int N = 1e4 + 10;
int n, m, root, Time, low[N], d[N];
bool vis[N], is_arti[N]; // is articulation point
vector <int> g[N];
vector <pair <int, int>> arti_bridge;

void dfs(int p, int u) {
	Time++;
	vis[u] = 1;
	d[u] = low[u] = Time;

	int child = 0;
	for (int v : g[u]) {
		if (v == p) continue;

		if (vis[v]) low[u] = min(low[u], d[v]);
		else {
			child++;
			dfs(u, v);

			low[u] = min(low[u], low[v]);
			if (low[v] >= d[u] and u != root)
				is_arti[u] = 1;

			if (d[u] < low[v])
				arti_bridge.push_back({min(u, v), max(u, v)});
```

```
            }
    }

    if (u == root and child > 1)
            is_arti[u] = 1;
}

int main () {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
            int u, v; cin >> u >> v;
            g[u].push_back(v);
            g[v].push_back(u);
    }

    Time = root = 1;
    memset(vis, 0, sizeof vis);
    memset(is_arti, 0, sizeof is_arti);
    dfs(root, root);
}
```

3.5 Online Articulation Bridge Finding

```
vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
int bridges;
int lca_iteration;
vector<int> last_visit;

void init(int n) {
    par.resize(n);
    dsu_2ecc.resize(n);
    dsu_cc.resize(n);
    dsu_cc_size.resize(n);
    lca_iteration = 0;
    last_visit.assign(n, 0);
    for (int i = 0; i < n; ++i) {
        dsu_2ecc[i] = i;
        dsu_cc[i] = i;
        dsu_cc_size[i] = 1;
        par[i] = -1;
    }
    bridges = 0;
}

int find_2ecc(int v) {
    if (v == -1)
        return -1;
    return dsu_2ecc[v] == v ? v : dsu_2ecc[v] = find_2ecc(dsu_2ecc[v]);
}

int find_cc(int v) {
    v = find_2ecc(v);
    return dsu_cc[v] == v ? v : dsu_cc[v] = find_cc(dsu_cc[v]);
}

void make_root(int v) {
    v = find_2ecc(v);
    int root = v;
    int child = -1;
    while (v != -1) {
        int p = find_2ecc(par[v]);
        par[v] = child;
        dsu_cc[v] = root;
        child = v;
        v = p;
    }
    dsu_cc_size[root] = dsu_cc_size[child];
}

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

    }

    for (int v : path_a) {
```

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

void add_edge(int a, int b) {
    a = find_2ecc(a);
    b = find_2ecc(b);
    if (a == b)
        return;

    int ca = find_cc(a);
    int cb = find_cc(b);

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

3.6 Strongly Connected Components

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

// given a directed graph return the minimum number of edges to be added so that
↪ the whole graph  become an SCC
bool vis[N];
vector<int> g[N], r[N], G[N], vec; //G is the condensed graph
void dfs1(int u) {
  vis[u] = 1;
  for (auto v : g[u]) if (!vis[v]) dfs1(v);
  vec.push_back(u);
}

vector<int> comp;
void dfs2(int u) {
  comp.push_back(u);
  vis[u] = 1;
  for (auto v : r[u]) if (!vis[v]) dfs2(v);
}

int idx[N], in[N], out[N];
int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  int n, m;
  cin >> n >> m;
  for (int i = 1; i <= m; i++) {
    int u, v;
    cin >> u >> v;
    g[u].push_back(v);
    r[v].push_back(u);
  }
  for (int i = 1; i <= n; i++) if (!vis[i]) dfs1(i);
  reverse(vec.begin(), vec.end());
  memset(vis, 0, sizeof vis);
  int scc = 0;
  for (auto u : vec) {
    if (!vis[u]) {
      comp.clear();
      dfs2(u);
      scc++;
      for (auto x : comp) idx[x] = scc;
    }
  }
  for (int u = 1; u <= n; u++) {
    for (auto v : g[u]) {
      if (idx[u] != idx[v]) {
        in[idx[v]]++, out[idx[u]]++;
        G[idx[u]].push_back(idx[v]);
      }
    }
  }
  int needed_in = 0, needed_out = 0;
  for (int i = 1; i <= scc; i++) {
    if (!in[i]) needed_in++;
    if (!out[i]) needed_out++;
  }
  int ans = max(needed_in, needed_out);
  if (scc == 1) ans = 0;
  cout << ans << '\n';
```

```
  return 0;
}
```

3.7 2 SAT

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

/*
zero Indexed
we have vars variables
F=(x_0 XXX y_0) and (x_1 XXX y_1) and ... (x_{vars-1} XXX y_{vars-1})
here {x_i,y_i} are variables
and XXX belongs to {OR,XOR}
is there any assignment of variables such that F=true
*/
struct twosat {
  int n;  // total size combining +, -. must be even.
  vector< vector<int> > g, gt;
  vector<bool> vis, res;
  vector<int> comp;
  stack<int> ts;
  twosat(int vars = 0) {
    n = vars << 1;
    g.resize(n);
    gt.resize(n);
  }

  //zero indexed, be careful
  //if you want to force variable a to be true in OR or XOR combination
  //add addOR (a,1,a,1);
  //if you want to force variable a to be false in OR or XOR combination
  //add addOR (a,0,a,0);

  //(x_a or (not x_b))-> af=1,bf=0
  void addOR(int a, bool af, int b, bool bf) {
    a += a + (af ^ 1);
    b += b + (bf ^ 1);
    g[a ^ 1].push_back(b);  // !a => b
    g[b ^ 1].push_back(a);  // !b => a
    gt[b].push_back(a ^ 1);
    gt[a].push_back(b ^ 1);
  }
  //(!x_a xor !x_b)-> af=0, bf=0
  void addXOR(int a, bool af, int b, bool bf) {
    addOR(a, af, b, bf);
    addOR(a, !af, b, !bf);
  }
  //add this type of condition->
  //add(a,af,b,bf) means if a is af then b must need to be bf
  void add(int a, bool af, int b, bool bf) {
    a += a + (af ^ 1);
    b += b + (bf ^ 1);
    g[a].push_back(b);
    gt[b].push_back(a);
  }
  void dfs1(int u) {
    vis[u] = true;
    for (int v : g[u]) if (!vis[v]) dfs1(v);
    ts.push(u);
  }
  void dfs2(int u, int c) {
    comp[u] = c;
    for (int v : gt[u]) if (comp[v] == -1) dfs2(v, c);
  }
  bool ok() {
    vis.resize(n, false);
    for (int i = 0; i < n; ++i) if (!vis[i]) dfs1(i);
    int scc = 0;
    comp.resize(n, -1);
    while (!ts.empty()) {
      int u = ts.top();
      ts.pop();
      if (comp[u] == -1) dfs2(u, scc++);
    }
    res.resize(n / 2);
    for (int i = 0; i < n; i += 2) {
      if (comp[i] == comp[i + 1]) return false;
      res[i / 2] = (comp[i] > comp[i + 1]);
    }
    return true;
  }
};

int main() {
  int n, m; cin >> n >> m;
  twosat ts(n);
  for (int i = 0; i < m; i++) {
    int u, v, k; cin >> u >> v >> k;
    --u; --v;
    if (k) ts.add(u, 0, v, 0), ts.add(u, 1, v, 1), ts.add(v, 0, u, 0), ts.add(v,
 1, u, 1);
    else ts.add(u, 0, v, 1), ts.add(u, 1, v, 0), ts.add(v, 0, u, 1), ts.add(v,
 1, u, 0);
  }
  int k = ts.ok();
  if (!k) cout << "Impossible\n";
  else {
    vector<int> v;
```

```
    for (int i = 0; i < n; i++) if (ts.res[i]) v.push_back(i);
    cout << (int)v.size() << '\n';
    for (auto x : v) cout << x + 1 << ' ';
    cout << '\n';
  }
  return 0;
}
```

4 Dynamic Programming

4.1 LIS

```
int lis(vector<int> const& a) {
 int n = a.size();
 const int INF = 1e9; //INF must be > max(a)
 vector<int> d(n + 1, INF);
 d[0] = -INF;
 for (int i = 0; i < n; i++) {
   int j = upper_bound(d.begin(), d.end(), a[i]) - d.begin();
   if (d[j - 1] <= a[i] && a[i] <= d[j]) d[j] = a[i];
 }
 int ans = 0;
 for (int i = 0; i <= n; i++) {
   if (d[i] < INF) ans = i;
 }
 return ans;
}

```

4.2 SOS DP

```
for(int i = 0; i<(1<<N); ++i)
        F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
        if(mask & (1<<i))
                F[mask] += F[mask^(1<<i)];
}
// The above algorithm runs in O(N2N) time.
```

4.3 Digit DP

```
/// How many numbers x are there in the range a to b, where the digit d occurs
↪ exactly k times in x?
vector<int> num;
int a, b, d, k;
int DP[12][12][2];
/// DP[p][c][f] = Number of valid numbers <= b from this state
/// p = current position from left side (zero based)
/// c = number of times we have placed the digit d so far
/// f = the number we are building has already become smaller than b? [0 = no, 1
↪ = yes]

int call(int pos, int cnt, int f){
    if(cnt > k) return 0;
    if(pos == num.size()){
        if(cnt == k) return 1;
        return 0;
    }
    if(DP[pos][cnt][f] != -1) return DP[pos][cnt][f];
    int res = 0; int LMT;
    if(f == 0) LMT = num[pos]; else  LMT = 9;
    for(int dgt = 0; dgt<=LMT; dgt++){
        int nf = f; int ncnt = cnt;
        if(f == 0 && dgt < LMT) nf = 1; /// The number is getting smaller at
↪ this position
        if(dgt == d) ncnt++;
        if(ncnt <= k) res += call(pos+1, ncnt, nf);
    }
    return DP[pos][cnt][f] = res;
}

int solve(int b){
    num.clear();
    while(b>0){
        num.push_back(b%10);
        b/=10;
    }
    reverse(num.begin(), num.end());
    memset(DP, -1, sizeof(DP));
    int res = call(0, 0, 0);
    return res;
}

int main () {
    cin >> a >> b >> d >> k;
    int res = solve(b) - solve(a-1);
    cout << res << endl;
}
```

5 String

5.1 Hashing

```
const ll N = 1e6 + 10, mod = 2e9 + 63, base1 = 1e9 + 21, base2 = 1e9 + 181;
ll pw1[N], pw2[N], hp1[N], hp2[N], hs1[N], hs2[N], n, q;
string s;

void pw_cal() {
  pw1[0] = pw2[0] = 1;
  for (int i = 1; i < N; i++) {
    pw1[i] = (pw1[i - 1] * base1) % mod;
    pw2[i] = (pw2[i - 1] * base2) % mod;
  }
}
void init() {
  hp1[0] = hp2[0] = hs1[n + 1] = hs2[n + 1] = 0;
  for (int i = 1; i <= n; i++) {
    hp1[i] = (hp1[i - 1] * base1 + s[i - 1]) % mod;
    hp2[i] = (hp2[i - 1] * base2 + s[i - 1]) % mod;
  }
  for (int i = n; i > 0; i--) {
    hs1[i] = (hs1[i + 1] * base1 + s[i - 1]) % mod;
    hs2[i] = (hs2[i + 1] * base2 + s[i - 1]) % mod;
  }
}

ll hashp(int l, int r) {
  ll hash1 = ( hp1[r] - hp1[l - 1] * pw1[r - l + 1]) % mod;
  if (hash1 < 0) hash1 += mod;
  ll hash2 = ( hp2[r] - hp2[l - 1] * pw2[r - l + 1]) % mod;
  if (hash2 < 0) hash2 += mod;
  return (hash1 << 32) | hash2;
}
ll hashs(int l, int r) {
  ll hash1 = (hs1[l] - hs1[r + 1] * pw1[r - l + 1]) % mod;
  if (hash1 < 0) hash1 += mod;
  ll hash2 = (hs2[l] - hs2[r + 1] * pw2[r - l + 1]) % mod;
  if (hash2 < 0) hash2 += mod;
  return (hash1 << 32) | hash2;
}

bool ispal(int l, int r) {
  int mid = (r + l) / 2;
  ll x = hashp(l, mid), y = hashs(mid, r);
  return x == y;
}
```

5.2 Trie

```
const int N = 1e5+10, M = 26;
int trie[N][M], nnode;
bool isword[N];

void reset(int k) {
        for (int i = 0; i < M; i++)
                trie[k][i] = -1;
}


void Insert(string &s) {
        int n = s.size(), node = 0;
        for (int i = 0; i < n; i++) {
                if (trie[node][s[i] - 'a'] == -1) {
                        trie[node][s[i] - 'a'] = ++nnode;
                        reset(nnode);
                }
                node = trie[node][s[i] - 'a'];
        }
        isword[node] = 1;
}

bool Search(string &s) {
        int n = s.size(), node = 0;
        for (int i = 0; i < s.size(); i++) {
                if (trie[node][s[i] - 'a'] == -1) return 0;
                node = trie[node][s[i] - 'a'];
        }
        return isword[node];
}
//find maximum subarray xor sum
int doxor(int s) {
  int nw = 0, t = 0;
  for (int i = 31; i >= 0; i--) {
    bool p = (1 << i) & s;
    if (node[nw][p ^ 1] != -1) {
      t |= 1 << i;
      nw = node[nw][p ^ 1];
    } else
      nw = node[nw][p];
  }
  return t;
}
//minimum subarray xor sum
int doxor2(int s) {
  int nw = 0, t = 0;
  for (int i = 31; i >= 0; i--) {
    bool p = (1 << i) & s;
```

```
    if (node[nw][p] != -1) nw = node[nw][p];
    else {
      t |= 1 << i;
      nw = node[nw][p ^ 1];
    }
  }
  return t;
}
//at first insert(0), then calculate xor before inserting each element of the array
//calculate number of subarray having xor>=k
int doxor(int s) {
  int nw = 0, t = 0;
  for (int i = 31; i >= 0; i--) {
    bool p = (1 << i) & s;
    bool q = (1 << i) & k;
    if (!q) {
      t += (node[nw][p ^ 1] != -1 ? word[node[nw][p ^ 1]] : 0);
      nw = node[nw][p];
    } else
      nw = node[nw][p ^ 1];
    if (nw == -1)
      break;
  }
  if (nw != -1)
    t += word[nw];
  return t;
}
//insert(0), sum returned value, insert prefix xor

int main() {
        reset(0);
        int n; cin >> n;
        for (int i = 0; i < n; i++) {
                string s;
                cin >> s;
                Insert(s);
        }
        int q; cin >> q;
        while (q--) {
                string s;
                cin >> s;
                cout << Search(s) << endl;
        }
}
```

5.3 Z-Algo //0-based Indexing

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

- The Z-function for this string s is an array of length n where the i-th element is equal to the greatest number of characters starting from the position i that coincides with the first characters of s.

5.4 KMP

```
#include <iostream>
using namespace std;
int pattern[200005],text[200005],f[200005]/*failure_table*/;
int pattern_size,text_size;
void build_table(){
    f[0]=0;
    f[1]=0;
    int i,j;
    for(i=2 ; i<=pattern_size ; i++){
        j=f[i-1];
        while(1){
            if(pattern[j]==pattern[i-1]){
                f[i]=j+1;
                break;
            }
            if(j==0){
                f[i]=0;
                break;
            }
            j=f[j];
        }
    }
}
int kmp(){
    build_table();
    int i=0,j=0,ret=0;
    while(1){
```

```
        if(j==text_size){
            break;
        }
        if(text[j]==pattern[i]){
            i++;
            j++;
            if(i==pattern_size) {
                ret++;
                i=f[i];
            }
        }
        else{
            if(i==0){
                j++;
            }
            else{
                i=f[i];
            }
        }
    }
  return ret;
}

int main(){
    text_size=3;
    pattern_size=2;
    text[0]=1;
    text[1]=1;
    text[2]=2;
    pattern[0]=1;
    pattern[1]=2;

    cout<<kmp();

}

```

5.5 Aho-Corasick

```
int nxt[N][26],cnt[N],f[N],vis[N],num;
vector <int> g[N]; //fail node tree

char s[1000005],pat[505]; //given string and pattern

//adding patterns to the trie
void add(){
    int node=0,i,len=strlen(pat);

    for(i=0 ; i<len ; i++){
        int ch=pat[i]-'a';

        if(nxt[node][ch]){
            node=nxt[node][ch];
        }
        else{
            nxt[node][ch] = ++num;
            node = num;
        }

    }
    qrnode.pb(node);
}

//building automaton
void build_aho(){
    queue <int> q;
    int i,u,v,curr;

    for(i=0 ; i<26 ; i++){
        if(nxt[0][i]) q.push(nxt[0][i]);
    }

    while(q.size()){
        curr = q.front();
        q.pop();

        for(i=0 ; i<26 ; i++){
            if(nxt[curr][i]){
                u=nxt[curr][i];
                q.push(u);
                v=f[curr];
                while(v && nxt[v][i]==0){
                    v=f[v];
                }
                f[u]=nxt[v][i];
                g[f[u]].pb(u);
            }
        }
    }

}
//searching the string
void search(){
    int j=0,i,len=strlen(s);

    for(i=0 ; i<len ; i++){
        int ch=s[i]-'a';

```

```
        while(j && nxt[j][ch]==0){
            j = f[j];
        }

        j=nxt[j][ch];

        cnt[j]++;
    }
}
//dfs to add cnts of fail node subtree of a node
void dfs(int node){
    vis[node]=1;
    for(auto to : g[node]){
        if(!vis[to]) dfs(to);
        cnt[node]+=cnt[to];
    }


}
```

5.6 Palindromic Tree

5.7 Suffix Array

```
#define MAX_N 1000020
int n, t;
// char s[500099];
string s;
int SA[MAX_N], LCP[MAX_N];
int RA[MAX_N], tempRA[MAX_N];
int tempSA[MAX_N];
int c[MAX_N];
int Phi[MAX_N], PLCP[MAX_N];
// second approach: O(n log n)
// the input string, up to 100K characters
// the length of input string
// rank array and temporary rank array
// suffix array and temporary suffix array
// for counting/radix sort
void countingSort(int k) {    // O(n)
    int i, sum, maxi = max(300, n);
    // up to 255 ASCII chars or length of n
    memset(c, 0, sizeof c);
    // clear frequency table
    for (i = 0; i < n; i++)
        // count the frequency of each integer rank
        c[i + k < n ? RA[i + k] : 0]++;
    for (i = sum = 0; i < maxi; i++) {
        int t = c[i]; c[i] = sum; sum += t;
    }
    for (i = 0; i < n; i++)
        // shuffle the suffix array if necessary
        tempSA[c[SA[i] + k < n ? RA[SA[i] + k] : 0]++] = SA[i];

    for (i = 0; i < n; i++)
        // update the suffix array SA
        SA[i] = tempSA[i];
}

void buildSA() {
    int i, k, r;
    for (i = 0; i < n; i++) RA[i] = s[i];
    // initial rankings
    for (i = 0; i < n; i++) SA[i] = i;
    // initial SA: {0, 1, 2, ..., n-1}
    for (k = 1; k < n; k <<= 1) {
        // repeat sorting process log n times
        countingSort(k); // actually radix sort: sort based on the second item
        countingSort(0);
        // then (stable) sort based on the first item
        tempRA[SA[0]] = r = 0;
        // re-ranking; start from rank r = 0
        for (i = 1; i < n; i++)
            // compare adjacent suffixes
            tempRA[SA[i]] = // if same pair => same rank r; otherwise, increase r
                (RA[SA[i]] == RA[SA[i - 1]] && RA[SA[i] + k] == RA[SA[i - 1] + k]) ? r : ++r;
        for (i = 0; i < n; i++)
            // update the rank array RA
            RA[i] = tempRA[i];

        if (RA[SA[n - 1]] == n - 1) break;
        // nice optimization trick
    }
}

void buildLCP() {
    int i, L;
    Phi[SA[0]] = -1;
    // default value
    for (i = 1; i < n; i++)
        // compute Phi in O(n)
        Phi[SA[i]] = SA[i - 1];
    // remember which suffix is behind this suffix
    for (i = L = 0; i < n; i++) {
        // compute Permuted LCP in O(n)
        if (Phi[i] == -1) { PLCP[i] = 0; continue; }
```

```
        // special case
        while (s[i + L] == s[Phi[i] + L]) L++;
        // L increased max n times
        PLCP[i] = L;
        L = max(L - 1, 0);
        // L decreased max n times
    }
    for (i = 0; i < n; i++)
        // compute LCP in O(n)
        LCP[i] = PLCP[SA[i]];
    // put the permuted LCP to the correct position
}
// n = string length + 1
// s = the string
// memset(LCP, 0, sizeof(LCP));  setting all index of LCP to zero
// buildSA(); for building suffix array
// buildLCP(); for building LCP array
// LCP is the longest common prefix with the previous suffix here
// SA[0] holds the empty suffix "\0".

int main()
{
    s = "banana";
    s += "$";
    n = s.size();

    memset(LCP, 0, sizeof(LCP));
    buildSA();
    buildLCP();

    for (int i = 0; i < n; i++) cout << SA[i] << ' ' << s.substr(SA[i], n -
↪   SA[i]) << endl;;
    printf("\n");
    for (int i = 0; i < n; i++) printf("%d ", LCP[i]);
    printf("\n");

    return 0;
}
```

6 Geometry

6.1 Convex Hull

```
struct pt {
    double x, y;
};

int orientation(pt a, pt b, pt c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool ccw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o > 0 || (include_collinear && o == 0);
}

void convex_hull(vector<pt>& a, bool include_collinear = false) {
    if (a.size() == 1)
        return;

    sort(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.x, a.y) < make_pair(b.x, b.y);
    });
    pt p1 = a[0], p2 = a.back();
    vector<pt> up, down;
    up.push_back(p1);
    down.push_back(p1);
    for (int i = 1; i < (int)a.size(); i++) {
        if (i == a.size() - 1 || cw(p1, a[i], p2, include_collinear)) {
            while (up.size() >= 2 && !cw(up[up.size()-2], up[up.size()-1], a[i],
↪   include_collinear))
                up.pop_back();
            up.push_back(a[i]);
        }
        if (i == a.size() - 1 || ccw(p1, a[i], p2, include_collinear)) {
            while (down.size() >= 2 && !ccw(down[down.size()-2],
↪   down[down.size()-1], a[i], include_collinear))
                down.pop_back();
            down.push_back(a[i]);
        }
    }

    if (include_collinear && up.size() == a.size()) {
        reverse(a.begin(), a.end());
        return;
    }
    a.clear();
    for (int i = 0; i < (int)up.size(); i++)
        a.push_back(up[i]);
    for (int i = down.size() - 2; i > 0; i--)
        a.push_back(down[i]);
```

```
}
```

6.2 Polar Sort

```
inline bool up (point p) {
  return p.y > 0 or (p.y == 0 and p.x >= 0);
}

sort(v.begin(), v.end(), [] (point a, point b) {
  return up(a) == up(b) ? a.x * b.y > a.y * b.x : up(a) < up(b);
});
```

7 Miscellaneous

7.1 C++17 Sublime Build

```
{
"cmd" : ["g++ -std=c++17 -Wall -fsanitize=address $file_name -o
    $file_base_name && timeout 10s ./$file_base_name <in>out"],
"selector" : "source.cpp",
"shell": true,
"working_dir" : "$file_path"
}
```

7.2 Test Case Generator with FASTIO

```
#pragma GCC optimize("Ofast,unroll-loops")
#pragma GCC target("avx,avx2,fma")

//generator to generate testcase
#include <bits/stdc++.h>
using namespace std;
#define ll long long
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
// return a random number in [l, r] range
ll rand(ll l, ll r) {
  return uniform_int_distribution<ll>(l, r)(rng);
}
void tree() {
  int n = rand(1, 10);
  int t = rand(1, 5);

  vector<int> p(n);
  for (int i = 0; i < n; i++)
    if (i > 0) p[i] = rand(i, t);

  printf("%d\n", n);
  vector<int> perm(n);
  for (int i = 0; i < n; i++) perm[i] = i;
  shuffle(perm.begin() + 1, perm.end(), rng);
  vector<pair<int, int> > edges;

  for (int i = 1; i < n; i++)
    if (rand(0, 1))
      edges.push_back(make_pair(perm[i], perm[p[i]]));
    else
      edges.push_back(make_pair(perm[p[i]], perm[i]));

  shuffle(edges.begin(), edges.end(), rng);
  for (int i = 0; i + 1 < n; i++)
    printf("%d %d\n", edges[i].first + 1, edges[i].second + 1);
}

int main(int argc, char* argv[]) {
  ios_base::sync_with_stdio(0); cin.tie(0);

  ll t = rand(1, 1);
  cout << t << endl;
  while (t--) {
    ll n = rand(1, 15);
    cout << n << endl;
  }
  return 0;
}
```

7.3 Output Checker Bash Script

```
//Bash script to auto check output
for((i = 1; ; ++i)); do
    echo $i
    ./gen $i > in
    # ./a < in > out1
    # ./brute < in > out2
    # diff -w out1 out2 || break
    diff -w <(./sol < in) <(./brute < in) || break
done
echo case
cat in
#create and build a bruteforce code named brute.cpp, main solution code sol.cpp
↪   and a random test case generator gen.cpp. To make this script runable, run
↪   this command chmod +x s.sh (s.sh is this bash script name). Then run the
↪   script by ./s.sh or bash s.sh
```

7.4 Custom Hash for unordered map

```
// To prevent collision in unordered_map
#include <bits/stdc++.h>
using namespace std;
struct custom_hash {
        static uint64_t splitmix64(uint64_t x) {
// http://xorshift.di.unimi.it/splitmix64.c
                x += 0x9e3779b97f4a7c15;
                x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
                x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
                return x ^ (x >> 31);
        }
        size_t operator()(uint64_t x) const {
                static const uint64_t FIXED_RANDOM =
                    chrono::steady_clock::now().time_since_epoch().
                    count();
                return splitmix64(x + FIXED_RANDOM);
        }
};
/// Declaration: unordered_map <int, int, custom_hash > numbers;
/// Usage: same as normal unordered_map
/// Ex: numbers[5] = 2;
/// *** To use gp_hash_table (faster than unordered_map) **** ///
/// Add these extra two lines:
/// #include <ext/pb_ds/assoc_container.hpp>
/// using namespace __gnu_pbds;
/// Declaration: gp_hash_table<int, int, custom_hash > numbers;
/// Usage: Same as unordered_map
```

7.5 Release vector memory

```
vector<int> Elements ;
// fill the vector up
vector<int>().swap(Elements);
```

7.6 Set of Structure [Operator overload]

```
struct seg{
    ll l , r , c;
    bool operator < (const seg &S) const {
        return l < S.l ;
    }
};
set<seg> s ; s.insert({0,0,0}) ;
```

7.7 Graph and Bitmask Operation

```
#pragma GCC optimize("Ofast,unroll-loops")
#pragma GCC target("avx,avx2,fma")
/* Infos */
~ 4 Direction
int dr[] = {1, -1, 0, 0};
int dc[] = {0, 0, 1, -1};
~ 8 Direction
int dr[] = {1, -1, 0, 0, 1, 1, -1, -1};
int dc[] = {0, 0, 1, -1, 1, -1, 1, -1};
~ Knight Direction
int dr[] = {1, -1, 1, -1, 2, 2, -2, -2};
int dc[] = {2, 2, -2, -2, 1, -1, 1, -1};
~ Hexagonal Direction
int dr[] = {2, -2, 1, 1, -1, -1};
int dc[] = {0, 0, 1, -1, 1, -1};
~ bitmask operations
int Set(int n, int pos) { return n = n | (1 << pos); }
int reset(int n, int pos) { return n = n & ~(1 << pos); }
bool check(int n, int pos) { return (bool)(n & (1 << pos)); }
int toggle(int n, int pos) { return n = (n ^ (1 << pos));}
bool isPower2(int x) { return (x && !(x & (x - 1))); }
ll LargestPower2LessEqualX(ll x) { for (int i = 1; i <= x / 2; i *= 2) x = x |
↪ (x >> i); return (x + 1) / 2;}
// for unlimited stack, run the command in terminal and run the code in terminal
ulimit - s unlimited
```

8 Notes

8.1 Stars and Bars Theorem

1. The number of ways to put n identical objects into k labeled boxes is $\binom{n+k-1}{n}$
2. Suppose, there are n objects to be placed into k bins, ways = $\binom{n-1}{k-1}$
3. Statement of 1no. and empty bins are valid, ways = $\binom{n+k-1}{k-1}$

8.2 GCD

1. gcd(a, b) = gcd(a, a - b) [a > b]
2. gcd(F(a), F(b)) = F(gcd(a,b)) [F=fibonacci]
3. gcd(a,b)=$\sum \phi(k)$ where k are all common divisors of a and b

8.3 Geometric Formula

1. Point Slope Form: $y - y_1 = m \cdot (x - x_1)$
2. Slope, $m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$
3. Slope from line, $m = -\frac{A}{B}$
4. Angle, $tan\theta = \frac{m_1 - m_2}{1 + m_1 \cdot m_2}$
5. Distance from a Point (x_0, y_0) to a Line $(Ax + By + C = 0) = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$
6. Area of segment in radian angle: $A = \frac{1}{2} \cdot r^2(\theta – Sin\theta)$
7. The sum of interior angles of a polygon with n sides $= 180 \cdot (n - 2)$
8. Number of diagonals of a $n-$sided polygon $= \frac{n(n-3)}{2}$
9. The measure of interior angles of a regular $n-$sided polygon $= \frac{(n-2) \cdot 180}{n}$
10. The measure of exterior angles of a regular $n-$sided polygon $= \frac{360}{n}$
11. Picks theorem: $A = I + \frac{B}{2} - 1$ where A = Area of Polygon, B = Number of integral points on edges of polygon, I = Number of integral points strictly inside the polygon.

8.4 Series/Progression

1. Sum of first n positive number $= \frac{n(n+1)}{2}$
2. Sum of first n odd number $= n^2$
3. Sum of first n even number $= n \cdot (n + 1)$
4. Arithmetic Progression: $n-$th term $= a + (n - 1) \cdot d$, sum $= \frac{n}{2}\{2a + (n - 1) \cdot d\}$; a = first element, d = difference between two elements
5. Geometric Progression: $n-$th term $= a \cdot r^{n-1}$, sum $= a \cdot \frac{r^{n-1}}{r-1}$; a = fist element, r = ratio of two elements
6. Catalan Numbers: $1, 1, 2, 5, 14, 42, 132 \ldots C_n = \frac{(2n)!}{n! \cdot (n+1)!}; n >= 0$

8.5 Combinatorial formulas

1. $\sum_{k=0}^{n} k^2 = n(n + 1)(2n + 1)/6$
2. $\sum_{k=0}^{n} k^3 = n^2(n + 1)^2/4$
3. $\sum_{k=0}^{n} k^4 = (6n^5 + 15n^4 + 10n^3 - n)/30$
4. $\sum_{k=0}^{n} k^5 = (2n^6 + 6n^5 + 5n^4 - n^2)/12$
5. $\sum_{k=0}^{n} x^k = (x^{n+1} - 1)/(x - 1)$
6. $\sum_{k=0}^{n} kx^k = (x - (n + 1)x^{n+1} + nx^{n+2})/(x - 1)^2$
7. $\binom{n}{k} = \frac{n!}{(n-k)!k!}$
8. $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$
9. $\binom{n}{k} = \frac{n}{n-k}\binom{n-1}{k}$
10. $\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$
11. $\binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$
12. $\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$
13. $\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}$
14. $\sum_{k=1}^{n} k^2\binom{n}{k} = (n + n^2)2^{n-2}$
15. $\binom{m+n}{r} = \sum_{k=0}^{r} \binom{m}{k}\binom{n}{r-k}$
16. $\binom{n}{k} = \prod_{i=1}^{k} \frac{n-k+i}{i}$
17. $a^{\phi(n)} = 1\%n$ where $\phi(n)$ is Euler Totient Function.
18. $a^b\%m = a^{b\%\phi(m)}\%m$ where a and m are coprime.