Reference Notebook

MD. SHOJIBUR RAHMAN FAHAD(22303266)
Department of CSE, IUBAT

binpow ncr npr modinv

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e6, mod = 1e9 + 7;

int power(long long n, long long k) {
    int ans = 1 % mod; n %= mod; if (n < 0) n += mod;
    while (k) {
        if (k & 1) ans = (long long) ans * n % mod;
        n = (long long) n * n % mod;
        k >>= 1;
    }
    return ans;
}
int f[N], invf[N];
int nCr(int n, int r) {
    if (n < r or n < 0) return 0;
    return 1LL * f[n] * invf[r] % mod * invf[n - r] % mod;
}
int nPr(int n, int r) {
    if (n < r or n < 0) return 0;
    return 1LL * f[n] * invf[n - r] % mod;
}
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    f[0] = 1;
    for (int i = 1; i < N; i++) {
        f[i] = 1LL * i * f[i - 1] % mod;
    }
    invf[N - 1] = power(f[N - 1], mod - 2);
    for (int i = N - 2; i >= 0; i--) {
        invf[i] = 1LL * invf[i + 1] * (i + 1) % mod;
    }
    cout << nCr(6, 2) << '\n';
    cout << nPr(6, 2) << '\n';
    return 0;
}

// Fahad MMI = binExpIter(a, m-2);
int binExpIter(int a, int b){
   int ans=1;
   while(b){
      if(b&1)
         ans= (ans * 1LL * a)%M;
      a=(a*1LL*a)%M;
      b>>=1;
   }
   return ans;
}

int MMI(int n) {
    return binExpIter(n, MOD - 2);
}
```

Dijkstra by GM

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

const int N = 3e5 + 9, mod = 998244353;

int n, m;
vector<pair<int, int>> g[N], r[N];
```

```
vector<long long> dijkstra(int s, int t, vector<int> &cnt) {
    const long long inf = 1e18;
    priority_queue<pair<long long, int>, vector<pair<long long,
int>>, greater<pair<long long, int>>> q;
    vector<long long> d(n + 1, inf);
    vector<bool> vis(n + 1, 0);
    q.push({0, s});
    d[s] = 0;
    cnt.resize(n + 1, 0); // number of shortest paths
    cnt[s] = 1;
    while(!q.empty()) {
        auto x = q.top();
        q.pop();
        int u = x.second;
        if(vis[u]) continue;
        vis[u] = 1;
        for(auto y: g[u]) {
        int v = y.first;
        long long w = y.second;
        if(d[u] + w < d[v]) {
            d[v] = d[u] + w;
            q.push({d[v], v});
            cnt[v] = cnt[u];
        } else if(d[u] + w == d[v]) cnt[v] = (cnt[v] + cnt[u])
% mod;
        }
    }
    return d;
}

int u[N], v[N], w[N];
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int s, t;
    cin >> n >> m >> s >> t;
    for(int i = 1; i <= m; i++) {
        cin >> u[i] >> v[i] >> w[i];
        g[u[i]].push_back({v[i], w[i]});
        r[v[i]].push_back({u[i], w[i]});
    }
    vector<int> cnt1, cnt2;
    auto d1 = dijkstra(s, t, cnt1);
    auto d2 = dijkstra(t, s, cnt2);

    long long ans = d1[t];
    for(int i = 1; i <= m; i++) {
        int x = u[i], y = v[i];
        long long nw = d1[x] + w[i] + d2[y];
        if(nw == ans && 1LL * cnt1[x] * cnt2[y] % mod ==
cnt1[t]) cout << "YES\n";
        else if(nw - ans + 1 < w[i]) cout << "CAN " << nw - ans
+ 1 << '\n';
        else cout << "NO\n";
    }
    return 0;
}
```

Dijkstra by Tanvir

```
// Dijkstra for weighted graph with non-negative weights
void dijkstra(int src, vector<vector<pair<int,int>>>& adj,
vector<int>& dist){
    int n = adj.size(); // Nodes are 0-based
    dist.assign(n, INT_MAX);
    priority_queue<pair<int,int>, vector<pair<int,int>>,
greater<>> pq;
    dist[src] = 0;
    pq.push({0, src});
    while (!pq.empty()) {
        auto [d, u] = pq.top(); pq.pop();
        if (d > dist[u]) continue; // Outdated entry
        for (auto [v, w] : adj[u]) {
```

```
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}
// Assumes adj[u] = vector of {neighbor, weight}
// Make sure graph has no negative weights
```

DSU

```
// 1-based indexing
class DSU {
    public:
    vector<int> parent, rating;
    int cc_count;
    DSU(int n) {
        cc_count = n;
        parent.assign(n+10, 0);
        rating.assign(n+1, 1);
        for(int i=1; i<=n; i++){
            parent[i]=i;
            rating[i]=1;
        }
    }
    int find_parent(int node){
        return parent[node] = (parent[node]==node ? node:
find_parent(parent[node]));
    }
    bool is_in_same(int u, int v) {
        int par1 = find_parent(u);
        int par2 = find_parent(v);
        if(par1==par2){
            return true;
        }
        return false;
    }
    // Returns size of component
    int get_size(int u) {return rating[find_parent(u)];}
    // False means already connected
    bool merge(int u, int v){
        if(is_in_same(u, v)) return false;
        int par1 = find_parent(u);
        int par2 = find_parent(v);
        if(rating[par1] >= rating[par2]){
            parent[par2] = par1;
            rating[par1] += rating[par2];
        }
        else{
            parent[par1]=par2;
            rating[par2]+= rating[par1];
        }
        --cc_count;
        return true;
    }
}
```

Equation Solve

```
// a1*x + b1*y = c1 | a2*x + b2*y = c2
bool solve_2_variable(double a1, double b1, double c1,
                      double a2, double b2, double c2,
                      double &x, double &y) {
    double det = a1 * b2 - a2 * b1;
    if (abs(det) < 1e-9) return false; // No unique solution
    x = (c1 * b2 - c2 * b1) / det;
    y = (a1 * c2 - a2 * c1) / det;
    return true;
}
   // a1*x + b1*y + c1*z = d1 | a2*x + b2*y + c2*z = d2 | a3*x
+ b3*y + c3*z = d3
bool solve_3_variable(double a1, double b1, double c1, double
d1,
```

```
                    double a2, double b2, double c2, double
d2,
                    double a3, double b3, double c3, double
d3,
                    double &x, double &y, double &z) {
    double D = a1*(b2*c3 - b3*c2) - b1*(a2*c3 - a3*c2) +
c1*(a2*b3 - a3*b2);
    if (abs(D) < 1e-9) return false; // No unique solution
    double Dx = d1*(b2*c3 - b3*c2) - b1*(d2*c3 - d3*c2) +
c1*(d2*b3 - d3*b2);
    double Dy = a1*(d2*c3 - d3*c2) - d1*(a2*c3 - a3*c2) +
c1*(a2*d3 - a3*d2);
    double Dz = a1*(b2*d3 - b3*d2) - b1*(a2*d3 - a3*d2) +
d1*(a2*b3 - a3*b2);
    x = Dx / D;
    y = Dy / D;
    z = Dz / D;
    return true;
}
```

KMP

```
//finding frequency of P in S, as a substring.
// call build_failure_function() first
#define MAX 1000005
int failure[MAX];
//longest prefix that also mactchs current suffix
void build_failure_function(string pattern, int m) {
    failure[0] = 0;
    failure[1] = 0; //base case

    for(int i = 2; i <= pattern.size(); i++) {
        int j = failure[i - 1];
        while(true) {
        if(pattern[j] == pattern[i - 1]) {
            failure[i] = j + 1;
            break;
        }
        if(j == 0) {
            failure[i] = 0;
            break;
        }
        j = failure[j];
        }
    }
}

int frequency_of_P_in_S(string &s, stirng &p){
    build_failure_function(p, p.size());
    int total_mtc = 0;
    int ans=0;
    for(int i=0; i<s.size(); i++){
        if(s[i]==p[total_mtc]) total_mtc++;
        else{
            while(total_mtc > 0){
                total_mtc = failure[total_mtc];
                if(s[i]==p[total_mtc]){
                    total_mtc++;
                    break;
                }
            }
        }
        if(p.size() == total_mtc){
            ans++;
            total_mtc = failure[total_mtc];
        }
    }
    return ans;
}
```

Knapsack 1

```
int n, w; cin>>n>>w;
vector<int> dp(w+1, 0);
for(int i=0; i<n; i++){
```

```
    int weight, val; cin>>weight>> val;
    for(int prev_weight=w-weight; prev_weight>=0; prev_weight--
){
        dp[prev_weight+weight] = max(dp[prev_weight+weight],
dp[prev_weight]+val);
    }
}
cout<<dp[w]<<nl;
```

LCA with Binary Lifting

```
const int N = 3e5 + 9, LG = 18;
vector<int> g[N];
int par[N][LG + 1], dep[N], sz[N];
// Call at first
void dfs(int u, int p = 0) {
  par[u][0] = p;
  dep[u] = dep[p] + 1;
  sz[u] = 1;
  for (int i = 1; i <= LG; i++) par[u][i] = par[par[u][i -
1]][i - 1];
  for (auto v: g[u]) if (v != p) {
    dfs(v, u);
    sz[u] += sz[v];
  }
}
// Lowest Common Ancestor
int lca(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  for (int k = LG; k >= 0; k--) if (dep[par[u][k]] >= dep[v]) u
= par[u][k];
  if (u == v) return u;
  for (int k = LG; k >= 0; k--) if (par[u][k] != par[v][k]) u =
par[u][k], v = par[v][k];
  return par[u][0];
}
// K-th root from u
int kth(int u, int k) {
  assert(k >= 0);
  for (int i = 0; i <= LG; i++) if (k & (1 << i)) u =
par[u][i];
  return u;
}
// Calculate distance between u and v
int dist(int u, int v) {
  int l = lca(u, v);
  return dep[u] + dep[v] - (dep[l] << 1);
}
//kth node from u to v, 0th node is u
int go(int u, int v, int k) {
  int l = lca(u, v);
  int d = dep[u] + dep[v] - (dep[l] << 1);
  assert(k <= d);
  if (dep[l] + k <= dep[u]) return kth(u, k);
  k -= dep[u] - dep[l];
  return kth(v, dep[v] - dep[l] - k);
}
int32_t main() {
  int n; cin >> n;
  for (int i = 1; i < n; i++) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
  }
  dfs(1);
  int q; cin >> q;
  while (q--) {
    int u, v; cin >> u >> v;
    cout << dist(u, v) << '\n';
  }
  return 0;
}
```

LCS - longest common subsequences

```
int L[m + 1][n + 1];
```

```
for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
        if (i == 0 || j == 0) L[i][j] = 0;
        else if (X[i - 1] == Y[j - 1]) L[i][j] = L[i - 1][j -1]
+ 1;
        else L[i][j] = max(L[i - 1][j], L[i][j - 1]);
    }
}
```

LIS - Longest Increasing Subsequence

```
// LIS - Longest increasing subsequence
// O(n*logn)
const int INF=INT_MAX;
vector<int> LIS(vector<int> & inp){
    int n=inp.size();
    vector<int> a(n+2, INF);
    vector<int> res(n,1);
    a[0]=-INF;
    int lis=0;
    for(int i=0; i<n; i++){
        int low = lower_bound(a.begin(), a.end(), inp[i])-
a.begin();
        a[low] = inp[i];
        lis = low;
        res[i] = lis;
    }
    return res;
}
```

Mo's Algorithm

```
const int M = 1e9+7;
int n, q;
vector<int> v;
struct query{int l,r,idx;};
int block;
bool comp1(query p,query q){
    if (p.l / block != q.l / block) {
        if(p.l==q.l) return p.r<q.r;
        return p.l < q.l;
    }
    return (p.l / block & 1) ? (p.r < q.r) : (p.r > q.r);
}
vector<int> fre(1000001, 0);
int no_of_distinct=0;
void add(int idx){
    fre[v[idx]]++;
    if(fre[v[idx]]==1) no_of_distinct++;
}
void rmv(int idx){
    fre[v[idx]]--;
    if(fre[v[idx]]==0) no_of_distinct--;
}
int get_answer(){
    return no_of_distinct;
}
void mos_algorithm(int n, vector<query>&queries){
    vector<int> answers(queries.size());
    block = (int)sqrt(n);
    sort(queries.begin(), queries.end(),comp1);
    int cur_l = 0;
    int cur_r = -1;
    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) {rmv(cur_l);cur_l++;}
        while (cur_r > q.r) {rmv(cur_r);cur_r--;}
        answers[q.idx] = get_answer();
    }
    for(int i:answers) {cout<<i<<"\n";}
}
void srfahad2021(){
    cin>>n;
    v.assign(n, 0);
```

```
    for(int i=0; i<n; i++){
        cin>>v[i];
    }
    vector<query> queries;
    cin>>q;
    for(int i=0; i<q; i++){
        int l, r; cin>>l>>r;
        l--; r--;
        queries.push_back({l, r, i});
    }
    mos_algorithm(q, queries);
}
int32_t main(){
    fast();
    int test=1;
    // cin>>test;
    for(int i=1; i<=test; i++){
        // cout<<"Case "<<i<<": ";
        srfahad2021();
    }
    return 0;
}
```

MST Kruskal's

```
// DSU is required
// sort the edge according to your problem(asc, dsc)
// Asc -> Minimum spanning tree
// Dsc -> Maximum spanning tree
    int n, m; cin >> n >> m;
    vector<array<int, 3>> ed;
    for(int i = 1; i <= m; i++){
        int u, v, w; cin >> u >> v >> w;
        ed.push_back({w, u , v});
    }
    sort(ed.begin(), ed.end());
    long long ans = 0;
    dsu d(n);
    for (auto e: ed){
        int u = e[1], v = e[2], w = e[0];
        if (d.same(u, v)) continue;
        ans += w;
        d.merge(u, v);
    }
    cout << ans << '\n';
```

ordered set

```
// Ordered Set
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;

template <typename T> using o_set = tree<T, null_type,
less<T>,rb_tree_tag, tree_order_statistics_node_update>;
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    o_set<int> se;
    cout << se.order_of_key(5) << ’\n’; // number of elements <
5
    se.erase(3); // erase by value
    cout << (*se.find_by_order(1)) << ’\n’; // if you imagine
this as a 0-indexed vector, what is se[1]?
    return 0;
}
```

Segment Tree Lazy

```
struct node
{
    ll sum;
    ll setAll;
    ll increment;
    bool setAllValid;
    node(){
```

```
        sum = 0;
        setAllValid = 0;
        increment = 0;
    }
    void Reset(){
        setAllValid = increment = 0;
    }
};
// 0-based indexing
class segtree{
    public:
    int range;
    vector<node> tree;
    void build(vector<int>& v){
        range = v.size();
        tree.assign(4*range, node());
        build_recur(v, 0, range-1, 0);
    }
    void build_recur(vector<int>& v, int l, int r, int
node_no){
        if(l == r){
            if(l < v.size())
                tree[node_no].sum = v[l];
            else tree[node_no].sum = 0;
            return;
        }
        int mid = (l+r)/2;
        build_recur(v, l, mid, 2*node_no + 1);
        build_recur(v, mid + 1, r, 2*node_no + 2);
        tree[node_no].sum = tree[2*node_no + 1].sum +
tree[2*node_no + 2].sum;
    }
    ll range_query(int L, int R){
        return range_query_recur(0, 0, range - 1, L, R);
    }
    void incUpdate_recur(int node, int l, int r, int& L, int&
R, int& X){
        if(r < L || R < l || l >= range)
            return;
        if(L <= l && R >= r)
        {
            tree[node].increment += X;
            return;
        }
        applyAggr(node,l,r);
        int mid = (l+r)/2;
        incUpdate_recur(2*node+1,l,mid,L,R,X);
        incUpdate_recur(2*node+2,mid+1,r,L,R,X);
        applyAggr(2*node+1, l, mid);
        applyAggr(2*node+2, mid+1, r);
        tree[node].sum = tree[2*node+1].sum +
tree[2*node+2].sum;
    }
    void incUpdate(int L, int R, int X){
        incUpdate_recur(0,0,range-1,L,R,X);
    }
    void setUpdate_recur(int node, int l, int r, int& L, int&
R, int& X){
        if(r < L || R < l || l >= range)
            return;
        if(L <= l && R >= r)
        {
            tree[node].setAllValid = 1;
            tree[node].setAll = X;
            tree[node].increment = 0;
            return;
        }
        applyAggr(node,l,r);
        int mid = (l+r)/2;
        setUpdate_recur(2*node+1,l,mid,L,R,X);
        setUpdate_recur(2*node+2,mid+1,r,L,R,X);
```

```
        applyAggr(2*node+1, l, mid);
        applyAggr(2*node+2, mid+1, r);
        tree[node].sum = tree[2*node+1].sum +
tree[2*node+2].sum;
    }
    void setUpdate(int L, int R, int X){
        setUpdate_recur(0,0,range-1,L,R,X);
    }
    void compose(int par, int child){
        if(tree[par].setAllValid){
            tree[child].setAllValid = 1;
            tree[child].setAll = tree[par].setAll;
            tree[child].increment = tree[par].increment;
        }
        else tree[child].increment += tree[par].increment;
    }
    void applyAggr(int node, int l, int r){
        if(tree[node].setAllValid)
            tree[node].sum = (r-l+1)*tree[node].setAll;
        tree[node].sum += (r-l+1)*tree[node].increment;
        if(l != r){
            compose(node, 2*node + 1);
            compose(node, 2*node + 2);
        }
        tree[node].Reset();
    }
    ll range_query_recur(int node, int l, int r, int& L, int&
R)
    {
        if(r < L || R < l || l >= range)
            return 0;
        applyAggr(node, l, r);
        if(L <= l && R >= r)
            return tree[node].sum;
        int mid = (l+r)/2;
        return range_query_recur(2*node + 1, l, mid, L, R) +
range_query_recur(2*node + 2, mid+1, r, L, R);
    }
};
```

Segment Tree

```
class SegmentTree{
    public:
        vector<vector<int>> tree; //0-mn, 1-mx, 2-sum
        SegmentTree(int n){
            tree.assign(3, vector<int>(4*n+1, 0));
        }
        void build(vector<int> &a, int l, int r, int node){
            if(l>r) return;
            if(l==r){
                tree[0][node]=a[l];
                tree[1][node]=a[l];
                tree[2][node] =a[l];
                return;
            }
            int mid = (l+r) /2;
            build(a, l, mid, 2*node);
            build(a, mid+1, r, 2*node+1);

            tree[0][node] = min(tree[0][2*node] ,
tree[0][2*node + 1]);
            tree[1][node] = max(tree[1][2*node] ,
tree[1][2*node + 1]);
            tree[2][node] = tree[2][2*node] +
tree[2][2*node+1];
        }

        void update( int node, int l, int r, int idx, int
value){
            if(l>r) return;
            if(l==r){
                tree[0][node]=value;
```

```
            tree[1][node]=value;
            tree[2][node] =value;
            return;
        }
        int mid = (l+r)/2;
        if(idx >=l and idx<=mid){
            update(2*node, l, mid, idx, value);
        }
        else{
            update(2*node+1, mid+1, r, idx, value);
        }
        tree[0][node] = min(tree[0][2*node] ,
tree[0][2*node + 1]);
        tree[1][node] = max(tree[1][2*node] ,
tree[1][2*node + 1]);
        tree[2][node] = tree[2][2*node] +
tree[2][2*node+1];
    }

    //b=begin, e=end
    int query_mn(int node, int l, int r, int b, int e){
        if(l>=b and r<=e) return tree[0][node];
        if(l>e or r<b) return 1e18; // return invalid
value;
        int mid = (l+r) /2;
        int left = query_mn(2*node, l, mid, b, e);
        int right = query_mn(2*node+1, mid+1, r, b, e);
        return min(left, right);
    }
    int query_mx(int node, int l, int r, int b, int e){
        if(l>=b and r<=e) return tree[1][node];
        if(l>e or r<b) return -1e18; // return invalid
value;
        int mid = (l+r) /2;
        int left = query_mx(2*node, l, mid, b, e);
        int right = query_mx(2*node+1, mid+1, r, b, e);
        return max(left, right);
    }
    int query_sum(int node, int l, int r, int b, int e){
        if(l>=b and r<=e) return tree[2][node];
        if(l>e or r<b) return 0; // return invalid value;
        int mid = (l+r) /2;
        int left = query_sum(2*node, l, mid, b, e);
        int right = query_sum(2*node+1, mid+1, r, b, e);
        return left + right;
    }


};
```

Sieve of Eratosthenes

```
// remember to call the function
vector<bool> prime(1e7+10, true);

void SieveOfEratosthenes(){
    prime[0]=prime[1]=false;
    for (int p = 2; p * p <= 1e7+10; p++) {
        if (prime[p] == true) {
            for (int i = p * p; i <= 1e7+10; i += p)
                prime[i] = false;
        }
    }
}
```

string hashing

```
// Don't Use int=long long
// call prec() funtion from the main function
// create an object of Hashing with a (string)parameter first
const int N = 1e6 + 9;
int power(long long n, long long k, const int mod) {
    int ans = 1 % mod;
    n %= mod;
    if (n < 0) n += mod;
```

```
    while (k) {
        if (k & 1) ans = (long long) ans * n % mod;
        n = (long long) n * n % mod;
        k >>= 1;
    }
    return ans;
}
// extra mod 999999989, 1e9+9, 1e9+7
const int MOD1 = 127657753, MOD2 = 987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void prec() {
    pw[0] =  {1, 1};
    for (int i = 1; i < N; i++) {
        pw[i].first = 1LL * pw[i - 1].first * p1 % MOD1;
        pw[i].second = 1LL * pw[i - 1].second * p2 % MOD2;
    }
    ip1 = power(p1, MOD1 - 2, MOD1);
    ip2 = power(p2, MOD2 - 2, MOD2);
    ipw[0] =  {1, 1};
    for (int i = 1; i < N; i++) {
        ipw[i].first = 1LL * ipw[i - 1].first * ip1 % MOD1;
        ipw[i].second = 1LL * ipw[i - 1].second * ip2 % MOD2;
    }

}
struct Hashing {
    int n;
    string s; // 0 - indexed
    vector<pair<int, int>> hs; // 1 - indexed
    Hashing() {}
    Hashing(string _s) {
        n = _s.size();
        s = _s;
        hs.emplace_back(0, 0);
        for (int i = 0; i < n; i++) {
        pair<int, int> p;
        p.first = (hs[i].first + 1LL * pw[i].first * s[i] %
MOD1) % MOD1;
        p.second = (hs[i].second + 1LL * pw[i].second * s[i] %
MOD2) % MOD2;
        hs.push_back(p);
        }
    }
    pair<int, int> get_hash(int l, int r) { // 1 - indexed
        assert(1 <= l && l <= r && r <= n);
        pair<int, int> ans;
        ans.first = (hs[r].first - hs[l - 1].first + MOD1) *
1LL * ipw[l - 1].first % MOD1;
        ans.second = (hs[r].second - hs[l - 1].second + MOD2) *
1LL * ipw[l - 1].second % MOD2;
        return ans;
    }
    pair<int, int> get_hash() {
        return get_hash(1, n);
    }
};
```

ternary_search

```
double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);      //evaluates the function at m1
        double f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
```

```
    return f(l);                    //return the maximum of
f(x) in [l, r]
}
```

Topological Sort

```
vector<vector<int>> adjList;
vector<int> topoLogicalSort(vector<pair<int, int>> edges, int
nodeCount) {
    vector<int> inDegree(nodeCount + 1, 0);
    vector<vector<int>> adjList(nodeCount + 1);
    for(auto edge : edges) {
        inDegree[edge.second] += 1;
        adjList[edge.first].push_back(edge.second);
    }
    queue<int> nodesWithIndegreeZero;
    for(int i = 0; i <nodeCount; i++) {
        if(inDegree[i] == 0) {
            nodesWithIndegreeZero.push(i);
        }
    }
    vector<int> topoLogicallySortedNodes;
    while (!nodesWithIndegreeZero.empty()) {
        int node = nodesWithIndegreeZero.front();
        topoLogicallySortedNodes.push_back(node);
        nodesWithIndegreeZero.pop();
        for (auto x: adjList[node]) {
            inDegree[x]--;
            if (inDegree[x] == 0) {
                nodesWithIndegreeZero.push(x);
            }
        }
    }
    return topoLogicallySortedNodes;
}
```

Extra

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=1}^{n} k^3 = \left(\frac{n(n+1)}{2}\right)^2$$