```
#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
#define PI 3.14159265358979323846
#define toLowerCase(s) transform(s.begin(), s.end(), s.begin(), ::tolower);
#define toUpperCase(s) transform(s.begin(), s.end(), s.begin(), ::toupper);
using namespace std;
using namespace __gnu_pbds;
typedef long long ll;
typedef tree<ll, null_type, less_equal<ll>, rb_tree_tag, tree_order_statistics_node_update> ordered_set;
int dx[] = {+1, -1, 0, 0, +1, +1, -1, -1};
int dy[] = {0, 0, -1, +1, +1, -1, +1, -1};
bool check_prime(ll n) {
    if (n == 2) return true;
    if (n < 2 or n % 2 == 0) return false;
    for (ll i = 3; i * i <= n; i += 2) { if (n % i == 0) return false; } return true;
}
bool check_power_of_two(ll n){ return !(n & (n - 1)); }
bool check_perfect_square(ll n){ if (n < 0) return false; ll root = sqrt(n); return (root * root == n); }
bool check_fibonacci(int n) { return check_perfect_square(5*n*n + 4) or check_perfect_square(5*n*n - 4); }
bool check_parity(ll n) { return __builtin_parityll(n); } // returns 1 if the number has odd parity
// bitset <32> b(n) ==> For 32 bit
string binary_representation_of_given_number(ll n) {
    if (n == 0) return "0";
    string binary = bitset<64>(n).to_string();
    return binary.substr(binary.find('1'));
}
ll countLeadingZeros(ll n) { return __builtin_clzll(n); }
ll countTrailing_Zeros(ll n) { return __builtin_ctzll(n); }
ll Number_of_Set_Bits(ll n) { return __builtin_popcountll(n); }
ll clearKthBit(ll n, ll k) { return (n & (~(1ll << k))); }
ll setKthBit(ll n, ll k) { return ((1ll << k) | n); }
ll checkKthBit(ll n, ll k) { return (n & (1ll << k)); }
void idea() { }
int main(){
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    // freopen("input.txt", "r", stdin); freopen("output.txt", "w", stdout);
    int T = 1;
    // cin >> T;
    for(int C = 1; C <= T; C++) {
        // cout << "Case " << C << ": " << '\n';
        idea();
    }
    return 0;
```

```
}
```

Math

```
ll All_Possible_Substring_Sum(string s) {
    int n = (int)s.size();
    vector<ll> digit_sum(n);
    digit_sum[0] = s[0] - '0';
    ll totalSum = digit_sum[0];
    for (int i = 1; i < n; i++) {
        int cur_val = s[i] - '0';
       digit_sum[i] = (i + 1) * cur_val + 10 * digit_sum[i - 1];
        totalSum += digit_sum[i];
    }
    return totalSum; // TC: O(N)
}
```

```
ll nPr(ll n, ll r, ll mod) { // O(log(MOD))
    if (r > n) return -1;
    ll numerator = fact[n] % mod;
    ll denominator = fact[n - r] % mod;
                numerator = (numerator * Modular_Exponentiation(denominator, mod - 2, mod)) % mod;
    return numerator;
}
```

```
ll maximum_subarray_sum(vector <ll> &v) {
    int n = v.size();
    ll maxSum = v[0], currentSum = v[0];
    for (int i = 1; i < n; i++) {
        currentSum = max(currentSum + v[i], v[i]);
        maxSum = max(maxSum, currentSum);
    }
    return maxSum; // TC: O(N)
}
```

```
// Function to calculate nCr % MOD
ll nCr(ll n, ll r, ll mod) { // O(log(MOD))
    if (r == 0) return 1;
    if (r > n) return -1;
    ll numerator = fact[n] % mod;
    ll denominator = (fact[n - r] * fact[r]) % mod;
                numerator = (numerator * Modular_Exponentiation(denominator, mod - 2, mod)) % mod;
    return numerator;
}
```

String

```
void GenerateAndPrintAllSubstrings(string s) {
    int n = s.size();
    for (int i = 0; i < n; i++) {
        string currentSubstring;
        for (int j = i; j < n; j++) {
            currentSubstring += s[j];
            cout << currentSubstring << '\n';
        }
    } // TC: O(N x N)
}
```

```
vector<int> constructTempArray(string pattern) {
    vector<int> lps(pattern.size());
int index = 0;
    for (int i = 1; i < (int)pattern.size();) {
        if (pattern[i] == pattern[index]) {
            lps[i] = index + 1;
```

```
bool KMPMultipleTimes(string text, string pattern) {
    vector<int> lps = constructTempArray(pattern);
    int j = 0, i = 0;
    while (i < (int)text.size()) {
        if (text[i] == pattern[j]) {
            ++i; ++j;
        } else {
            if (j != 0) {
                j = lps[j - 1];
            } else {
                ++i;
            }  }
        if (j == (int) pattern.size()) {
            return true;  }
    }
    return false; // O(N + M)
}
```

```
            ++index;
            ++i;
        } else {
            if (index != 0) {
                index = lps[index - 1];
            } else {
                lps[i] = index;
                ++i;} }  }
    return lps;
}

int LCS(string& s1, string& s2) {
    int len1 = s1.length();
    int len2 = s2.length();
    vector<int> dp(len2 + 1, 0);
    for (int i = 1; i <= len1; i++) {
        int prevDiagonal = 0;
        for (int j = 1; j <= len2; j++) {
            int temp = dp[j];
            if (s1[i - 1] == s2[j - 1]) { dp[j] = prevDiagonal
+ 1; }
            else { dp[j] = max(dp[j], dp[j - 1]); }
            prevDiagonal = temp;
        }
    }
    return dp[len2]; // TC: O(N x M) | SC: O(N)
}
-------------------------------------------------------------
vector <long long> LIS_Path(vector <long long>&
sequence) {
    long long n = sequence.size();
    vector <long long> sub, subIndex;
    vector <long long> path(n, -1);
    for (long long i = 0; i < n; ++i) {
        if (sub.empty() || sub.back() < sequence[i]) {
            path[i] = sub.empty() ? -1 : subIndex.back();
            sub.push_back(sequence[i]);
            subIndex.push_back(i);
        }
        else {
            long long idx = lower_bound(sub.begin(),
sub.end(), sequence[i]) - sub.begin();
            path[i] = (idx == 0) ? -1 : subIndex[idx - 1];
            sub[idx] = sequence[i];
            subIndex[idx] = i;
```

```
int LongestCommonSubstring(string& s1, string& s2)
{
    int len1 = s1.length();
    int len2 = s2.length();
    vector<int> dp(len2 + 1, 0);
    int maxLength = 0;

    for (int i = 1; i <= len1; i++) {
        int prevDiagonal = 0;
        for (int j = 1; j <= len2; j++) {
            int temp = dp[j];
            if (s1[i - 1] == s2[j - 1]) {
                dp[j] = prevDiagonal + 1;
                maxLength = max(maxLength, dp[j]);
            } else {
              dp[j] = 0; // Reset if characters do not match
            }
            prevDiagonal = temp;
        }
    }
    return maxLength; // TC: O(N x M) | SC: O(N)
}

-------------------------------------------------------------
bool is_substring(string child, string mother) {
    if (mother.find(child) != string::npos) return true;
    return false; // TC: O(N)
}
-------------------------------------------------------------
```

String Hashing

```
int for_hash = 0, rev_hash = 0;
vector<int> pw(N);
int build_hash(string s) {
    int hsh = 0;
    f(i, 0, s.size()){
        hsh = (hsh + ((s[i] - 'a' + 1) * pw[i]) % M) % M;
      }
    return hsh;
}
void solve(void){
    string s, tmp;
    cin >> s;
    tmp = s;
    int l = s.size();
```

```
        }
    }
    vector <long long> result;
    long long t = subIndex.back();
    while (t != -1) {
        result.push_back(sequence[t]);
        t = path[t];
    }
    reverse(result.begin(), result.end());
    return result; // TC: O(NLogN)
} // LIS path ends
```

Trie DS

```
#define MAX_NODE 100000
#define MAX_LEN 100
char S[MAX_LEN];
// assuming words only have samll letters ['a', 'z]
int node[MAX_NODE][26];
int root, nnode;
int isWord[MAX_NODE];
// call before inserting any words into trie
void initialize () {
    root = 0; nnode = 0;
    for (int i = 0; i < 26; i++)
        // -1 means no edge for ('a' + i)th character
        node[root][i] = -1;
}
void insert() {
    scanf("%s", S);
    int len = strlen(s);
    int now = root;
    for (int i = 0; i < len; i++) {
        if (node[now][S[i] - 'a'] == -1) {
            node[now][S[i] - 'a'] = ++nnode;
            for (int j = 0; j < 26; j++)
                node[nnode][j] = -1;
        }
        now = node[now][S[i] - 'a'];
    }
    // mark that a word ended at this node.
    isWord[now] = 1;
}
```

```
    reverse(all(tmp));
    int p = 27;
    pw[0] = 1;
    f(i, 1, N) {
        pw[i] = (pw[i - 1] * p) % M;
    }
    for_hash = build_hash(s);
    rev_hash = build_hash(tmp);
    // cout<<for_hash<<" "<<rev_hash;
    int q;
    cin >> q;
    while (q--) {
        char flag, ch;   int k;
        cin >> flag >> ch >> k;
        string ad(k, ch);
        int ad_hash = build_hash(ad);
        if (flag == 'R') {
            rev_hash = (rev_hash * pw[k]) % M;
            rev_hash = (rev_hash + ad_hash) % M;
            ad_hash = (ad_hash * pw[l]) % M;
            for_hash = (for_hash + ad_hash) % M;
            l += k;
        }
        else {
            for_hash = (for_hash * pw[k]) % M;
            for_hash = (for_hash + ad_hash) % M;
            ad_hash = (ad_hash * pw[l]) % M;
            rev_hash = (rev_hash + ad_hash) % M;
            l += k;
        }
    }
}
```

Number Theory

```
const int N = 1e8 + 3;
vector <int> spf(N + 1, 0);
vector <long long> saved_primes;
void linear_sieve() {
   for (int i = 2; i <= N; i += 2) {
      if (spf[i] == 0) {
         spf[i] = 2;
         if (i == 2) {
            saved_primes.push_back(2);
         }
      }
   }
   for (int i = 3; i <= N; i += 2) {
      if (spf[i] == 0) {
         spf[i] = i;
         saved_primes.push_back(i);
      }
       for (int j = 0; j < (int)saved_primes.size() and saved_primes[j] <= spf[i] and i * saved_primes[j] <= N; j++) {
         spf[i * saved_primes[j]] = saved_primes[j];
      }
   }
}
```

```
const int N = 1e8 + 3;
vector <bool> is_prime(N + 1, true);
vector <long long> saved_primes;
void standard_sieve() { // TC: O(N log log N)
   is_prime[0] = is_prime[1] = false;
   for (int i = 3; i * i < N; i += 2) {
      if (is_prime[i]) {
         for (int j = i * i; j < N; j += i + i) {
            is_prime[j] = false;
         }      }     }
   saved_primes.push_back(2);
   for (int i = 3; i < N; i += 2) {
      if (is_prime[i]) {
         saved_primes.push_back(i);
      }
   }
}
```

```
ll Modular_Exponentiation(ll base, ll exp, ll mod) {
   ll res = 1LL; base = base % mod;
   if (base == 0) return 0;
   while (exp > 0) {
      if (exp & 1) res = (res * base) % mod;
      exp = exp >> 1;
      base = (base * base) % mod;
   } return res; }
ll Modular_Addition(ll x, ll y, ll mod) {
   x = x % mod; y = y % mod;
   return (((x + y) % mod) + mod) % mod;
}
ll Modular_Subtraction(ll x, ll y, ll mod) {
   x = x % mod; y = y % mod;
   return (((x - y) % mod) + mod) % mod;
}
ll Modular_Multiplication(ll x, ll y, ll mod) {
   x = x % mod; y = y % mod;
   return (((x * y) % mod) + mod) % mod;
}
ll Modular_Inverse(ll x, ll mod) {
    return Modular_Exponentiation(x, mod - 2LL, mod);
}
```

```
vector <unsigned long long> Prime_Factorization(unsigned long long n) {
   vector <unsigned long long> prime_factors;
    for (size_t i = 0; i < saved_primes.size() && saved_primes[i] * saved_primes[i] <= n; i++) {
      if (n % saved_primes[i] == 0) {
         prime_factors.push_back(saved_primes[i]);
         while (n % saved_primes[i] == 0)
            n /= saved_primes[i];
      }
   }
   if (n > 1) prime_factors.push_back(n);
   return prime_factors;
}
```

Data Structures

```
class DisjointSet
{
    vector<int> par, rank;
public:
    DisjointSet(int n)
    {
        par.resize(n + 1);
        rank.resize(n + 1, 0);
        for (int i = 0; i <= n; i++)
        {
            par[i] = i;
        }
    }
    int find_parent(int node)
    {
        if (node == par[node])
            return node;
        else
            return par[node] = find_parent(par[node]);
    }
```

```
void union_by_rank(int u, int v)
    {
        int ult_u = find_parent(u);
        int ult_v = find_parent(v);
        if (ult_u == ult_v)
            return;
        else if (rank[ult_u] < rank[ult_v])
        {
            par[ult_u] = ult_v;
        }
        else if (rank[ult_u] > rank[ult_v])
        {
            par[ult_v] = ult_u;
        }
        else
        {
            par[ult_u] = ult_v;
            rank[ult_v]++;
        }
    }
};
```

```
// Segment Tree Starts
int lazy[4 * N];
int tree[4 * N];
int arr[N];
void build(int id, int l, int r){
  if (l == r){
    tree[id] = arr[l];
    return;
  }
  int mid = (l + r) / 2;
  build(2 * id , l, mid);
  build(2 * id + 1, mid + 1, r);
  tree[id] = tree[2 * id ] + tree[2 * id + 1];}

void update_tree(int node, int first_node,
int last_node, int lo_rng, int hi_rng, int val){
  if (lazy[node] != 0){
    tree[node] += (last_node - first_node + 1) *
lazy[node];
```

```
int rangeSum(int node, int first_node, int last_node,
int lo_rng, int hi_rng)
{
  if (lazy[node] != 0) {
    tree[node] += (last_node - first_node + 1) *
lazy[node];

    if (first_node != last_node){
      lazy[node * 2 ] += lazy[node];
      lazy[node * 2 + 1] += lazy[node]; }
    lazy[node] = 0; }
  if (first_node>hi_rng || last_node<lo_rng)return 0;
    if (first_node >= lo_rng && last_node <= hi_rng)
      return tree[node];

  int mid = (first_node + last_node) / 2;
  return rangeSum(node * 2 , first_node, mid, lo_rng,
hi_rng) + rangeSum(node * 2 + 1, mid + 1, last_node,
lo_rng, hi_rng);
```

```
    if (first_node != last_node){
      lazy[node * 2 ] += lazy[node];
      lazy[node * 2 + 1] += lazy[node];  }
    lazy[node] = 0; }
  if ( first_node > hi_rng || last_node < lo_rng)
    return;

  if (first_node >= lo_rng && last_node <= hi_rng){
    tree[node] += (last_node - first_node + 1) * val;
    if (first_node != last_node){
      lazy[node * 2 ] += val;
      lazy[node * 2 + 1] += val; }
    return;
  }
  int mid = (first_node + last_node) / 2;
  update_tree(node * 2 , first_node, mid, lo_rng, hi_rng, val);
  update_tree(node * 2 + 1, mid + 1, last_node, lo_rng, hi_rng, val);
  tree[node] = tree[node * 2 ] + tree[node * 2 + 1];
}
```

```
}
void solve(void){
  int n, q, flag, l, r, val, ans;
  cin >> n >> q;
  for (int i = 1; i <= n; i++)
    cin >> arr[i];
  build(1, 1, n);
  //for(int i =0;i<20;i++)cout<<tree[i]<<" ";cout<<endl;
  while (q--){
    cin >> flag;
    if (flag == 1){
      cin >> l >> val;
      update_tree(1, 1, n , l, l, val); }
    else{
    cin >> l >> r;
    ans = rangeSum(1, 1, n, l, r);
    cout << ans << "\n";
     }}
}
// Segment Tree Ends
```

DP

```
const int mx = 1e5 + 123;
ll n, bagSize, weight[123], value[123];
ll dp[123][mx];

ll knapsack (int idx, int value_left)
{
    if (value_left == 0) return 0;
    if (idx > n) return 1e15;
    if (dp[idx][value_left] != -1) return dp[idx][value_left];

    // We can always allowed to not picking an item. But, if we want want to pick an item we need a condition.
    ll pick, no_pick, ans; // We can't assign any value to any of these variable
    // Not picking an item
    no_pick = knapsack(idx + 1, value_left);
    ans = no_pick;
```

```
void solve()
{
    cin >> n >> bagSize;
    for (int i = 1; i <= n; i++)
        cin >> weight[i] >> value[i];

    memset(dp, -1, sizeof(dp));

    // How much weight do I need to carry currValue
    for (int currValue = 1e5 + 5; currValue >= 0; currValue--) {
        ll weight_need = knapsack(1, currValue);
        if (weight_need <= bagSize) {
            cout << currValue << endl;
            break;
        }
    }
}
```

```
    // Picking an item
    if (value_left - value[idx] >= 0) {
      pick = weight[idx] + knapsack(idx + 1, value_left
- value[idx]);
        ans = min(no_pick, pick);
    }

    return dp[idx][value_left] = ans;
}
```

Graph

```
//Parent_using_dfs
int parent[N];
vector<bool>vis(N);
vector<int> graph[N];
void dfs(int vertex,int par){
    parent[vertex] = par;
    vis[vertex]=true;
        for(auto child :graph[vertex]){
            if(vis[child])continue;
            dfs(child,vertex);
        }
}



// Dijkstra's Algorithm
const int INF = 1e18;
int NODE;
vector<int> dis(N, INF);
void dijkstra(int source, vector<pair<int, int>>
graph[])
{
    priority_queue<pair<int, int>, vector<pair<int,
int>>, greater<pair<int, int>>> pq;
    // vector<bool> vis(N, false);
    dis[source] = 0;
    pq.push({0, source});
    int node, distance;
    while (!pq.empty()){
        node = pq.top().second;
        distance = pq.top().first;  pq.pop();
        for (auto it : graph[node]){
            int to_node = it.second;
            int to_distance = it.first;
```

```
//prims algorithm
void prims_cal(vector<vector<pair<int, int>>>
&graph)
{
    priority_queue<pair<int, int>, vector<pair<int,
int>>, greater<pair<int, int>>> pq;

    vector<bool> vis(N, false);
    pair<int, int> zero = {0, 1};
    pq.push(zero);
    int node, weight, total_cost = 0;
    while (!pq.empty())
    {
        node = pq.top().second;
        weight = pq.top().first;

        pq.pop();
        if (vis[node] == true)
            continue;

        vis[node] = true;
        total_cost += weight;
        for (auto it : graph[node])
        {
            if (vis[it.first])
                continue;
            pq.push({it.second, it.first});
        }

    }
    cout << total_cost;
}
void solve(void)
```

```
            if (dis[to_node] > dis[node] + to_distance){
                dis[to_node] = dis[node] + to_distance;
                pq.push({dis[to_node], to_node});
            }
        }
    }
   // lets print the shortest path from source to all node;
    for (int i = 1; i <= NODE; i++)
    {
       cout << "distance from node :" << source << " to
node :" << i << " is :" << dis[i] << "\n";
    }
}
void solve(void)
{
    int edge;
    cin >> NODE >> edge;
    vector<pair<int, int>> graph[NODE + 1];
    for (int i = 0; i < edge; i++) {
        int v1, v2, w;
        cin >> v1 >> v2 >> w;
        graph[v1].pb({w, v2});
        graph[v2].pb({w, v1});
    }
    int source;
    cin >> source;
    dijkstra(source, graph);
}
//connected components
const int mx = 1e5 + 123;
vector<bool> vis(mx, false);
vector<int> adj[mx];

void dfs (int node)
{
    vis[node] = true;

    for (auto &&u : adj[node]) {
        if (vis[u] == false)
            dfs(u);
    }
    return;
}

int main(void)
{
    int node, edge;
    cin >> node >> edge;
    vector<vector<pair<int, int>>> graph(node + 5);
    while (edge--)
    {
        int v1, v2, w;
        cin >> v1 >> v2 >> w;
        graph[v1].pb({v2, w});
        graph[v2].pb({v1, w});
    }
    prims_cal(graph);
}
```

```
//path printing by bfs
const int mx = 1e5 + 123;
vector<int> adj[mx];
int level[mx], parent[mx];
void bfs (int source){
    memset(level, -1, sizeof(level));
    memset(parent, -1, sizeof(parent));
    level[source] = 0;
    queue<int> q;
    q.push(source);
    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (auto &&v: adj[u]) {
            if (level[v] == -1) {
                level[v] = level[u] + 1;
                parent[v] = u;
                q.push(v);
            }
        }
    }
}
int main(void) {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
```

```
{
    int node, edge; cin >> node >> edge;

    for (int i = 1; i <= edge; i++) {
        int u, v; cin >> u >> v;

        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    int ans = 0;
    for (int i = 1; i <= node; i++) {
        if (vis[i] == false) {
            ans++;
            dfs(i);
        }
    }

    cout << ans << endl;
    return 0;
}
```

```
//biparted graph    {keft+rught table}————————————>
const int mx = 1e5 + 123;
vector<int> adj[mx];
int color[mx];

bool isBipartite (int source)
{
    memset(color, -1, sizeof(color));
    color[source] = 1;

    queue<int> q;
    q.push(source);

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

        for (auto &&v: adj[u]) {
            if (color[v] == -1) {
    }
    int s, t;
    cout << "Source: "; cin >> s;
    cout << "Destination: "; cin >> t;  bfs(s);
   if (level[t] == -1) cout << "NO PATH FOUND" <<
endl;
    else {
        cout << "Path size: " << level[t] << endl;
        vector<int> path;
        path.push_back(t);
        while (parent[t] != -1) {
            path.push_back(parent[t]);
            t = parent[t];
        }
        reverse(path.begin(), path.end());
        cout << "Path: ";
        for (auto u: path)
            cout << u << " ";
        cout << endl;
    }
    return 0;
}
```

```
int main(void)
{
    for (int i = 0; i < mx; i++) adj[i].clear();

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

    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;

        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    int s;
    cout << "Source: "; cin >> s;

    if (isBipartite(s)) cout << "Bipartite/Bi-colorable"
<< endl;
```

```
            if (color[u] == 1) color[v] = 2;
            else color[v] = 1;

            q.push(v);
          }
          else if (color[u] == color[v]) return false;
        }
      }

    return true;
}
```

```
    else cout << "Not Bipartite/Bi-colorable" << endl;
    return 0;
}
```

Math formulas(Bangla)

ক্রমিক সংখ্যা বা ধারার পদসংখ্যা, যোগফল ও গড় নির্ণয়ের সূত্র :

১. ধারার পদ সংখ্যা = {(শেষপদ-১ম পদ) ÷ প্রতিপদের পার্থক্য} + ১

২. ধারার যোগফল = {(১ম পদ + শেষপদ)× পদসংখ্যা} ÷ ২

৩. ধারার গড় = (শেষপদ+১ম পদ) ÷ ২

১. আয়তক্ষেত্রের ক্ষেত্রফল = দৈর্ঘ্য × প্রস্থ (বর্গএকক)

২. আয়তক্ষেত্রের পরিসীমা = ২ × (দৈর্ঘ্য + প্রস্থ)

৩. সামান্তরিক ক্ষেত্রের ক্ষেত্রফল = ভূমি × উচ্চতা (বর্গএকক)

৪. বর্গক্ষেত্রের ক্ষেত্রফল = (বাহু)^২ (বর্গ একক)

৫. বর্গক্ষেত্রের পরিসীমা = ৪ × বাহু দৈর্ঘ্য

৬. ত্রিভুজের ক্ষেত্রফল = $^১/_২$ (ভূমি × উচ্চতা) (বর্গ একক)

৭. ত্রিভুজের ক্ষেত্রফল = $\sqrt{s(s-a)(s-b)(s-c)}$ {ত্রিভুজের তিন বাহুর দৈর্ঘ্য a, b, c দেয়া থাকলে, s অর্ধপরিসীমা এবং পরিসীমা $2S=(a+b+c)$ }

৮. সমবাহু ত্রিভুজের ক্ষেত্রফল = $\frac{\sqrt{3}a^2}{4}$ {এখানে, ত্রিভুজের বাহুর দৈর্ঘ্য a}

৯. সমদ্বিবাহু ত্রিভুজের ক্ষেত্রফল = $\frac{2\sqrt{4b^2-a^2}}{4}$ {এখানে, ভূমি a এবং এক বাহুর দৈর্ঘ্য b }

১০. সমকোণী ত্রিভুজের ক্ষেত্রফল = $\frac{1}{2}(a \times b)$ {এখানে, ত্রিভুজের সমকোণ সংলগ্ন বাহুদ্বয় a এবং b }

১১. বৃত্তের ক্ষেত্রফল = ফাই·r2 {এখানে, বৃত্তের ব্যাসার্ধ r }

সমান্তর ধারা

১. একটি সমান্তর ধারার প্রথম পদ a এবং সাধারন অন্তর d হলে,
r -তম পদ $=a+(r-1)d$

২. প্রথম n সংখ্যক স্বাভাবিক সংখ্যার সমষ্টি $=\frac{n(n+1)}{2}$.
অর্থাৎ, $1+2+3+\ldots\ldots\ldots\ldots\ldots\ldots+n=\frac{n(n+1)}{2}$

৩. প্রথম n সংখ্যক স্বাভাবিক সংখ্যার বর্গের সমষ্টি $=\frac{n(n+1)(2n+1)}{6}$
অর্থাৎ, $1^2+2^2+3^2+\ldots\ldots\ldots\ldots\ldots\ldots+n^2=\frac{n(n+1)(2n+1)}{6}$

৪. প্রথম n সংখ্যক স্বাভাবিক সংখ্যার ঘনের সমষ্টি $=\frac{n^2(n+1)^2}{4}$.
অর্থাৎ, $1^3+2^3+3^3+\ldots\ldots\ldots\ldots\ldots\ldots+n^3=\frac{n^2(n+1)^2}{4}$

৫. গুনান্তর/সমানুপাতিক ধারার n তম পদ,
t_n ={প্রথমপদ × (সাধারন অনুপাত)$^{n-1}$ }= ar^{n-1} এবং উহার n সংখ্যক পদের যোগফল, $S_n=\frac{a(r^n-1)}{r-1}$ যখন $r>1$
আবার, $S_n=\frac{a(1-r^n)}{1-r}$, যখন $r<1$

১. কোন আয়তকার ঘন বস্তুর দৈর্ঘ্য, প্রস্থ ও উচ্চতা যথাক্রমে a একক, b একক ও c একক হলে,

@ সমগ্রতলের ক্ষেত্রফল $= 2(ab+bc+ca)$ বর্গ একক।

@ আয়তন $= abc$ ঘন একক।

@ কর্ণ $= \sqrt{a^2+b^2+c^2}$ একক।

২. ঘনকের ক্ষেত্রে, $a=b=c$ হলে,

@ ঘনকের সমগ্রতলের ক্ষেত্রফল $= 6a^2$ বর্গ একক।

@ ঘনকের আয়তন $= a^3$ ঘন একক।

@ ঘনকের কর্ণ $= a\sqrt{3}$ একক।

সমবৃত্তভূমিক সিলিন্ডার / বেলন :

৩. সমবৃত্তভূমিক সিলিন্ডারের ভূমির ব্যাসার্ধ r এবং উচ্চতা h হলে,

@ সিলিন্ডারের বক্রতলের ক্ষেত্রফল $= \pi rl$ বর্গ একক।

@ সিলিন্ডারের সমগ্রতলের ক্ষেত্রফল $= \pi r(r+l)$ বর্গ একক।

@ সিলিন্ডারের আয়তন $= \frac{1}{3}\pi r^2 h$ ঘন একক।

সমবৃত্তভূমিক কোণক :

৪. সমবৃত্তভূমিক কোণকের উচ্চতা h, ভূমির ব্যাসার্ধ r এবং হেলানো তলের উচ্চতা l হলে,

@ কোণকের বক্রতলের ক্ষেত্রফল $= \pi rl$ বর্গ একক।

@ কোণকের সমগ্রতলের ক্ষেত্রফল $= \pi r(r+l)$ বর্গ একক।

@ কোণকের আয়তন $= \frac{1}{3}\pi r^2 h$ ঘন একক।

৫. গোলকের ব্যাসার্ধ r হলে,

@ গোলকের তলের ক্ষেত্রফল $= 4\pi r^2$ বর্গ একক।

@ গোলকের আয়তন $= \frac{4}{3}\pi r^3$ ঘন একক।

@ h উচ্চতায় তলচ্ছেদে উৎপন্ন বৃত্তের ব্যাসার্ধ $= \sqrt{r^2-h^2}$ একক।

Big Integer

```
class BigInt{
   string digits;
public:
   //Constructors:
   BigInt(unsigned long long n = 0);
   BigInt(string &);
   BigInt(const char *);
   BigInt(BigInt &);
   BigInt(const BigInt &);
   //Helper Functions:
   friend void divide_by_2(BigInt &a);
   friend bool Null(const BigInt &);
   friend int Length(const BigInt &);
   int operator[](const int)const;
   //Direct assignment
   BigInt &operator=(const BigInt &);
   //Post/Pre - Incrementation
   BigInt &operator++();
   BigInt operator++(int temp);
   BigInt &operator--();
   BigInt operator--(int temp);
   //Addition and Subtraction
   friend BigInt &operator+=(BigInt &, const BigInt &);
   friend BigInt operator+(const BigInt &, const BigInt &);
   friend BigInt operator-(const BigInt &, const BigInt &);
   friend BigInt &operator-=(BigInt &, const BigInt &);
   //Comparison operators
   friend bool operator==(const BigInt &, const BigInt &);
   friend bool operator!=(const BigInt &, const BigInt &);
   friend bool operator>(const BigInt &, const BigInt &);
   friend bool operator>=(const BigInt &, const BigInt &);
   friend bool operator<(const BigInt &, const BigInt &);
   friend bool operator<=(const BigInt &, const BigInt &);
   //Multiplication and Division
   friend BigInt &operator*=(BigInt &, const BigInt &);
   friend BigInt operator*(const BigInt &, const BigInt &);
   friend BigInt &operator/=(BigInt &, const BigInt &);
   friend BigInt operator/(const BigInt &, const BigInt &);
   //Modulo
   friend BigInt operator%(const BigInt &, const BigInt &);
   friend BigInt &operator%=(BigInt &, const BigInt &);
   //Power Function
```

```
    friend BigInt &operator^=(BigInt &,const BigInt &);
    friend BigInt operator^(BigInt &, const BigInt &);
    //Square Root Function
    friend BigInt sqrt(BigInt &a);
    //Read and Write
    friend ostream &operator<<(ostream &,const BigInt &);
    friend istream &operator>>(istream &, BigInt &);
    //Others
    friend BigInt NthCatalan(int n);
    friend BigInt NthFibonacci(int n);
    friend BigInt Factorial(int n);
};
BigInt::BigInt(string & s){
    digits = ""; int n = s.size();
    for (int i = n - 1; i >= 0;i--) {
        if(!isdigit(s[i])) throw("ERROR");
        digits.push_back(s[i] - '0');
    }
}
BigInt::BigInt(unsigned long long nr){
    do {
        digits.push_back(nr % 10); nr /= 10;
    } while (nr);
}
BigInt::BigInt(const char *s){
    digits = "";
    for (int i = strlen(s) - 1; i >= 0;i--) {
        if(!isdigit(s[i])) throw("ERROR");
        digits.push_back(s[i] - '0');
    }
}
BigInt::BigInt(BigInt & a) { digits = a.digits; }
BigInt::BigInt(const BigInt &a) { digits = a.digits; }
bool Null(const BigInt& a) {
    if(a.digits.size() == 1 && a.digits[0] == 0) return true;
    return false;
}
int Length(const BigInt & a) { return a.digits.size(); }
int BigInt::operator[](const int index)const {
    if(digits.size() <= index || index < 0) throw("ERROR");
    return digits[index];
}
bool operator==(const BigInt &a, const BigInt &b) { return a.digits == b.digits; }
bool operator!=(const BigInt & a,const BigInt &b) { return !(a == b); }
bool operator<(const BigInt&a,const BigInt&b) {
```

```
    int n = Length(a), m = Length(b);
    if(n != m) return n < m;
    while(n--)
        if(a.digits[n] != b.digits[n]) return a.digits[n] < b.digits[n];
    return false;
}
bool operator>(const BigInt&a,const BigInt&b) { return b < a; }
bool operator>=(const BigInt&a,const BigInt&b) { return !(a < b); }
bool operator<=(const BigInt&a,const BigInt&b) { return !(a > b); }
BigInt &BigInt::operator=(const BigInt &a) { digits = a.digits; return *this; }
BigInt &BigInt::operator++() {
    int i, n = digits.size();
    for (i = 0; i < n && digits[i] == 9;i++) digits[i] = 0;
    if(i == n) digits.push_back(1);
    else digits[i]++;
    return *this;
}
BigInt BigInt::operator++(int temp) { BigInt aux; aux = *this; ++(*this); return aux; }
BigInt &BigInt::operator--() {
    if(digits[0] == 0 && digits.size() == 1) throw("UNDERFLOW");
    int i, n = digits.size();
    for (i = 0; digits[i] == 0 && i < n; i++) digits[i] = 9;
    digits[i]--;
    if(n > 1 && digits[n - 1] == 0) digits.pop_back();
    return *this;
}
BigInt BigInt::operator--(int temp){ BigInt aux; aux = *this; --(*this); return aux; }
BigInt &operator+=(BigInt &a,const BigInt& b){
    int t = 0, s, i;
    int n = Length(a), m = Length(b);
    if(m > n) a.digits.append(m - n, 0);
    n = Length(a);
    for (i = 0; i < n; i++) {
        if(i < m) s = (a.digits[i] + b.digits[i]) + t;
        else s = a.digits[i] + t;
        t = s / 10; a.digits[i] = (s % 10);
    }
    if(t) a.digits.push_back(t);
    return a;
}
BigInt operator+(const BigInt &a, const BigInt &b){ BigInt temp; temp = a; temp += b; return temp; }
BigInt &operator-=(BigInt&a,const BigInt &b){
    if(a < b) throw("UNDERFLOW");
    int n = Length(a), m = Length(b);
    int i, t = 0, s;
```

```
    for (i = 0; i < n; i++) {
        if(i < m) s = a.digits[i] - b.digits[i]+ t;
        else s = a.digits[i]+ t;
        if(s < 0) s += 10, t = -1;
        else t = 0;
        a.digits[i] = s;
    }
    while(n > 1 && a.digits[n - 1] == 0) a.digits.pop_back(), n--;
    return a;
}
BigInt operator-(const BigInt& a,const BigInt&b) { BigInt temp; temp = a; temp -= b; return temp; }
BigInt &operator*=(BigInt &a, const BigInt &b) {
    if(Null(a) || Null(b)) { a = BigInt(); return a; }
    int n = a.digits.size(), m = b.digits.size();
    vector<int> v(n + m, 0);
    for (int i = 0; i < n;i++)
        for (int j = 0; j < m;j++) { v[i + j] += (a.digits[i] ) * (b.digits[j]); }
    n += m;
    a.digits.resize(v.size());
    for (int s, i = 0, t = 0; i < n; i++) {
        s = t + v[i]; v[i] = s % 10; t = s / 10; a.digits[i] = v[i];
    }
    for (int i = n - 1; i >= 1 && !v[i];i--) a.digits.pop_back();
    return a;
}
BigInt operator*(const BigInt&a,const BigInt&b) { BigInt temp; temp = a; temp *= b; return temp; }
BigInt &operator/=(BigInt& a,const BigInt &b) {
    if(Null(b)) throw("Arithmetic Error: Division By 0");
    if(a < b) { a = BigInt(); return a; }
    if(a == b) { a = BigInt(1); return a; }
    int i, lgcat = 0, cc;
    int n = Length(a), m = Length(b);
    vector<int> cat(n, 0);
    BigInt t;
    for (i = n - 1; t * 10 + a.digits[i]  < b;i--) { t *= 10; t += a.digits[i]; }
    for (; i >= 0; i--) {
        t = t * 10 + a.digits[i];
        for (cc = 9; cc * b > t; cc--);
        t -= cc * b;
        cat[lgcat++] = cc;
    }
    a.digits.resize(cat.size());
    for (i = 0; i < lgcat; i++) a.digits[i] = cat[lgcat - i - 1];
    a.digits.resize(lgcat);
    return a;
```

```
}
BigInt operator/(const BigInt &a,const BigInt &b) { BigInt temp; temp = a; temp /= b; return temp; }
BigInt &operator%=(BigInt& a,const BigInt &b) {
   if(Null(b)) throw("Arithmetic Error: Division By 0");
   if(a < b) { return a; }
   if(a == b) { a = BigInt(); return a; }
   int i, lgcat = 0, cc;
   int n = Length(a), m = Length(b);
   vector<int> cat(n, 0);
   BigInt t;
   for (i = n - 1; t * 10 + a.digits[i] < b;i--) { t *= 10; t += a.digits[i]; }
   for (; i >= 0; i--) {
      t = t * 10 + a.digits[i];
      for (cc = 9; cc * b > t;cc--);
      t -= cc * b;
      cat[lgcat++] = cc;
   }
   a = t; return a;
}
BigInt operator%(const BigInt &a,const BigInt &b) { BigInt temp; temp = a; temp %= b; return temp; }
BigInt &operator^=(BigInt & a,const BigInt & b) {
   BigInt Exponent, Base(a); Exponent = b; a = 1;
   while(!Null(Exponent)){
      if(Exponent[0] & 1) a *= Base;
      Base *= Base;
      divide_by_2(Exponent);
   }
   return a;
}
BigInt operator^(BigInt & a,BigInt & b) { BigInt temp(a); temp ^= b; return temp; }
void divide_by_2(BigInt & a) {
   int add = 0;
   for (int i = a.digits.size() - 1; i >= 0;i--){
      int digit = (a.digits[i] >> 1) + add;
      add = ((a.digits[i] & 1) * 5);
      a.digits[i] = digit;
   }
   while(a.digits.size() > 1 && !a.digits.back()) a.digits.pop_back();
}
BigInt sqrt(BigInt & a) {
   BigInt left(1), right(a), v(1), mid, prod; divide_by_2(right);
   while(left <= right) {
      mid += left; mid += right; divide_by_2(mid); prod = (mid * mid);
      if(prod <= a) { v = mid; ++mid; left = mid; }
      else { --mid; right = mid; }
```

```
        mid = BigInt();
    }
    return v;
}
BigInt NthCatalan(int n) {
    BigInt a(1),b;
    for (int i = 2; i <= n; i++) a *= i;
    b = a;
    for (int i = n + 1; i <= 2 * n; i++) b *= i;
    a *= a; a *= (n + 1); b /= a;
    return b;
}
BigInt NthFibonacci(int n) {
    BigInt a(1), b(1), c;
    if(!n) return c;
    n--;
    while(n--) { c = a + b; b = a; a = c; }
    return b;
}
BigInt Factorial(int n) {
    BigInt f(1);
    for (int i = 2; i <= n; i++) f *= i;
    return f;
}
istream &operator>>(istream &in, BigInt &a) {
    string s; in >> s; a.digits.clear();
    for (int i = s.size() - 1; i >= 0; i--) {
        if (!isdigit(s[i])) throw("INVALID NUMBER");
        a.digits.push_back(s[i] - '0');
    }
    return in;
}
ostream &operator<<(ostream &out,const BigInt &a) {
    for (int i = a.digits.size() - 1; i >= 0; i--) out << (short)a.digits[i];
    return out;
}

void idea() {

    // take input
    BigInt first_num, Second_num;
    cin >> first_num >> Second_num;

    // check equality
    if (first_num == Second_num) { cout << "Equal" << '\n'; }
```

```
    else { cout << "Not Equal" << '\n'; }

    // comaprison
    if (first_num > Second_num) { cout << "Greater" << '\n'; }
    else { cout << "Smaller" << '\n'; }

    // printing
    cout << first_num << ' ' << Second_num << '\n';

    // vector input
    vector <BigInt> vec = {first_num, Second_num};
    for(auto val : vec) { cout << val << ' '; cout << '\n'; }

    BigInt Fib = NthFibonacci(6); // 6th fibonacci is 8
    BigInt Cat = NthCatalan(10); // 10th catalan is 16796
    BigInt Fact = Factorial(5); // Factorial of 5 is 120
    cout << Fib << ' ' << Cat << ' ' << Fact << '\n';
}
```