Defines and Macros

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef long long ll;
typedef long double ld;
typedef int ii;
typedef unsigned long long ull;
typedef long long int lint;
typedef pair<int,int> pi;
typedef pair<ll,ll> pll;
#define shekoo ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
template <typename T>
using ordered_set = tree<T, null_type,less_equal<T>, rb_tree_tag,tree_order_statistics_node_update>;
#define of(m) find_by_order((m)) //random access of ordered set
#define all(v) (v).begin(),(v).end()
#define r_all(v) (v).rbegin(),(v).rend()
#define cnt_0s_lr(x) __builtin_clz(x) //It counts number of zeros before the first occurrence of one(set bit)
#define cnt_0s_rl(x) __builtin_ctz(x) //Count no of zeros ff last tt first occurrence of one(set bit).
#define popcnt(x) __builtin_popcount(x) //Count no of zeros ff last tt first occurrence of one(set bit).
#define s(v) (v).size()
#define endl "\n";
#define NO cout << "NO\n";
#define YES cout << "YES\n";
#define No cout << "No\n";
#define Yes cout << "Yes\n";
#define no cout << "no\n";
#define yes cout << "yes\n";
```

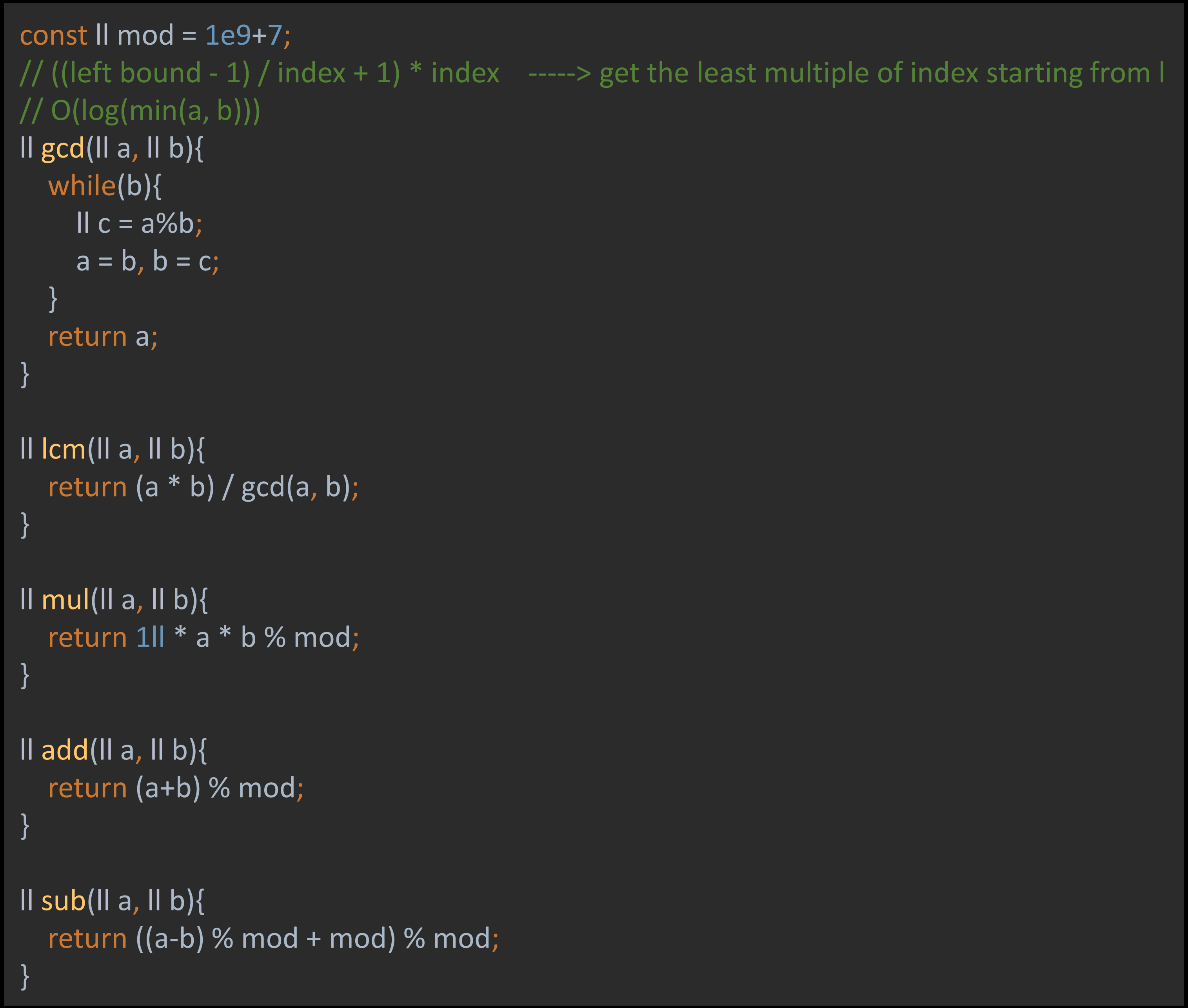
```cpp
#define PF push_front
#define PB push_back
#define MP make_pair
#define F first
#define S second
#define rep(i, m) for(ll i=0;i<(m);++i)
#define rep_inv(i,m) for(ll i=(m);i>=0;--i)
#define repn(i,in, m) for(ll i=(in);i<(m);++i)
#define repn_inv(i,m,in) for(ll i=(in);i>=(m);--i)
#define repa(i, x) for(auto &i: x)
#define clr(a) memset((a),0,sizeof(a))
#define vi vector<int>
#define vll vector<ll>
#define vg vector<pair<ll,ll>>
#define Pi M_PI
#define UNIQUE(c) (c).resize(unique(all(c)) - (c).begin())
#define st_int(s, base) stoi((s), 0 , (base))
#define st_ll(s, base) stoll((s), 0 , (base))
#define ch_st(c) string sch(1, (c))
#define stream(line) istringstream in((line)) //split string by spaces and can be accessed in >> s >> m >> s...
#define least_multiple_of_a(a, l) (((l) - 1) / (a) + 1) * (a) // l -> start value

void print_vec(vector<ll>& v){
   for(ll i=0; i<s(v); i++) cout << v[i] << " ";
   cout << endl
}
```

File Input/Output

```
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);
```

NUMBER THEORY



```cpp
const ll mod = 1e9+7;
// ((left bound - 1) / index + 1) * index    -----> get the least multiple of index starting from l
// O(log(min(a, b)))
ll gcd(ll a, ll b){
    while(b){
        ll c = a%b;
        a = b, b = c;
    }
    return a;
}

ll lcm(ll a, ll b){
    return (a * b) / gcd(a, b);
}

ll mul(ll a, ll b){
    return 1ll * a * b % mod;
}

ll add(ll a, ll b){
    return (a+b) % mod;
}

ll sub(ll a, ll b){
    return ((a-b) % mod + mod) % mod;
}
```

```
ll fast_power(ll b, ll p){
    ll ans = 1;
    while(p){
        if(p&1) ans = mul(ans, b);
        b = mul(b, b);
        p /= 2;
    }
    return ans;
}
// a power -1 we use it in division when using mod in problem
// mod should be prime number
ll fermats_inv(ll a){
    return fast_power(a, mod-2);
}
ll division(ll a, ll d){
    return mul(a, fermats_inv(d));
}
```

```
/// in counting sum -> or , product -> and
/// if you have n nums you can arrange them in n! diffrent wayes (n * n-1 * n-2 * ..1)
/// npr order matters (factorial 3ddhom 3la elli m4 ha5doo) n!/(n-r)!
/// ncr order doesn't matter -> n!/((n-r)!*r!
/// mod in inv should be prime
const ll N = 1e5+5;
ll fact[N], inv[N];
void precompute(){
    fact[0] = inv[0] = 1;
    for(ll i=1; i<N; i++){
        fact[i] = mul(fact[i-1], i);
        inv[i] = fast_power(fact[i], mod-2);
    }
}
ll ncr(ll n, ll r){
    return mul(fact[n], mul(inv[n-r], inv[r]));
}
ll npr(ll n, ll r){
    return mul(fact[n], inv[n-r]);
}
ll NCR(ll n, ll r){
    ll ans = 1;
    r = min(r, n-r);
    if(r == 0) return 1;
    if(r == 1) return n;
    for(ll i=1; i<=r; i++) ans = ans * (n-i+1) / i;
    return ans;
}
```

```
bool is_prime(ll n){
    for(ll i = 2; i*i <= n; i++)
        if(n % i == 0) return false;
    return true;
}

vector<ll> get_divs(ll n){
    vector<ll> divs;
    for(ll i=1; i*i <=n; i++){
        if(n%i == 0) {
            divs.push_back(i);
            if(i != n/i) divs.push_back(n/i);
        }
    }
    return  divs;
}
```

```
vector<pair<ll, ll>> prime_factors(ll n){
    vector<pair<ll,ll>> fact;
    for(ll i=2; i*i <= n; i++){
        ll cnt = 0;
        while(n % i == 0){
            n /= i;
            cnt++;
        }
        if(cnt) fact.push_back({i, cnt});
    }
    if(n > 1) fact.push_back({n, 1});
    return fact;
}

vector<bool> sieve(ll n){
    vector<bool> prime(n+1, 1);
    prime[0] = prime[1] = 0;
    for(ll i = 2; i*i <= n; i++){
        if(prime[i])
            for(ll j= i*i; j <= n; j+=i)
                prime[j] = 0;
    }
    return prime;
}
```

Geometry

```
const ll N = 1e5+5;
ll n, r[N]; // r contain raduis of point i
pll p[N]; // p.F -> x-axis of point i, p.S -> y-axis of point i
ll manhatan_dist(int i, int j) {
    return abs(p[i].F - p[j].S) + abs(p[i].F - p[j].S);
}
ll ecluid_dist(int i, int j) {
    ll x = abs(p[i].F - p[j].S), y = abs(p[i].F - p[j].S);
    return x * x + y * y;
}
// if two circles are one inside another or touches each other
bool valid(int i, int j) {
    return ecluid_dist(i, j) <= r[i] * r[i];
}
```

Shapes Area

```
/// Triangle --> 1/2 * b * h
/// Rectangle --> w * h
/// Trapizuim --> 1/2 * (a+b) * h
/// Ellipse --> pi * a * b
/// square --> a*a
/// parallelogram --> b * h
/// circle ---> pi * r * r
/// Sector = 1/2 * r * r * angle in radians
```

Moving on Grid

```
//knight move
ll dx[] = {2, 1, -1, 2, -1, -2, 1, -2};
ll dy[] = {1, 2, 2, -1, -2, -1, -2, 1};

// up - down - right - left - diagonals with dist 1
ll d[8][2] = {{0, 1}, {0, -1}, {-1, 0}, {1, 0},
              {-1, -1}, {1, -1}, {-1, 1}, {1, 1}};

// up down right left
ll d[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
```

RECURSION AND BACKTRACKING

Problem 1

Rat in a maze: a 2D maze is given where R is the position of a rat, C is the position of a piece of cheese, # is a wall, and . is an empty cell. What is the different paths by which the rat can get to the piece of cheese if it can move down (D) and right (R) on the empty cells.

```
bool valid(int r, int c){
   return r<n && c<m && grid[r][c]!='#';
}

void solve(int r, int c){
   if(r==tr && c==tc){
      cout<<path<<'\n';
   }
   else{
      if(valid(r,c+1)){       // option 1
         path.push_back('R');  // do
         solve(r,c+1);        // recurse
         path.pop_back();      // undo
      }
      if(valid(r+1,c)){       // option 2
         path.push_back('D'); // do
         solve(r+1,c);        // recurse
         path.pop_back();     // undo
      }
   }
}
```

Problem 2

Subset Sum: some unique numbers and a target sum are given. You are asked to print all the subset that sum up to the target sum

```
int sum,n, tmp;
vector<int>arr,path;

void solve(int i){
    if(i==n){
        if(tmp==sum){
            for(int x : path) cout<<x<<' ';
            cout<<'\n';
        }
    }
    else{
        // option 1 : pick
        tmp+=arr[i];
        path.push_back(arr[i]);
        solve(i+1);
        tmp-=arr[i];
        path.pop_back();

        //option 2 : leave
        solve(i+1);
    }
}
```

Problem 3

Find all valid Placements of a queen

```
vector<vector<int>> valid_placements;
vector<pair<int, int>> placements;



bool is_valid(int row, int col)
{
   for(int i = 0; i < placements.size(); i++)
   {
      if(row == placements[i].F) return false;
      if(abs(row-placements[i].F) == abs(placements[i].S-col)) return false;
   }
   return true;
}
```

```
void gen_placements(int c)
{
    if(c == 9)
    {
        vector<int> cur_placements;
        for(int i=0; i<8; i++)
        {
            cur_placements.push_back(placements[i].F);
        }
        valid_placements.push_back(cur_placements);
        return;
    }
    for(int r = 1; r <= 8; r++)
    {
        if(is_valid(r, c))
        {
            placements.push_back({r, c});
            gen_placements(c+1);
            placements.pop_back();
        }
    }
}
```

Problem 4

Prime Ring problem (every to adjacent elements are prime)

```
void solve(ll pos)
{
    if(pos == n-1 && is_prime(ring[n] + ring[n-1]))
    {
        cout << "1";
        for(int i=1; i<n; i++)
        {
            cout << " " << ring[i];
        }
        cout << endl
        return;
    }
    for(ll i=2; i<= n; i++)
    {
        if(!vis[i] && is_prime(ring[pos]+i))
        {
            vis[i] = 1;
            ring[pos+1] = i;
            solve(pos+1);
            vis[i] = 0;
        }
    }
}
```

Segment Tree

```cpp
/// seg tree to get the segment with max sum in a segment tree
struct item{
   ll seg, suf, pre, sum;
};
struct seg_tree3 {
   int size;
   vector<item> values;
   item NEUTRAL_ELEMENT = {0, 0, 0, 0};

   item merge(item a, item b){
      return {
            max({a.seg, b.seg, a.suf + b.pre}),
            max(b.suf, a.suf + b.sum),
            max(a.pre, a.sum+ b.pre),
            a.sum + b.sum
      };
   }
   item single(int v){
      if(v > 0){
         return {v, v, v, v};
      }
      else{
         return {0, 0, 0, v};
      }
   }
```

1

```cpp
void init(int n){
   size = 1;
   while (size < n) size *= 2;
   values.resize(2 * size);
}

void build(vi &a, int x, int lx, int rx){
   if(rx-lx == 1){
      if(lx < (int) a.size()){
         values[x] = single(a[lx]);
      }
      return;
   }
   int mid = (lx+rx)/2;
   build(a, 2*x+1, lx, mid);
   build(a, 2*x+2, mid, rx);
   values[x] = merge(values[2 * x + 1], values[2 * x + 2]); //change
}
void build(vi &a){
   build(a, 0, 0, size);
}
```

2

3

```
    void set(int i, int v, int x, int lx, int rx){
        if(rx-lx == 1){
            values[x] = single(v);
            return;
        }
        int mid = (lx+rx)/2;
        if(i < mid){
            set(i, v, 2*x+1, lx, mid);
        }
        else{
            set(i, v, 2*x+2, mid, rx);
        }
        values[x] = merge(values[2 * x + 1], values[2 * x + 2]); //change
    }
    void set(int i, int v){
        set(i, v, 0, 0, size);
    }

    item calc(int l, int r, int x, int lx, int rx){
        if((lx >= r) || (rx <= l)) return NEUTRAL_ELEMENT; //change
        if((lx >= l) && (rx <= r)) return values[x];
        int mid = (lx+rx)/2;
        item m1 = calc(l, r, 2 * x + 1, lx, mid);
        item m2 = calc(l, r, 2 * x + 2, mid, rx);
        return merge(m1, m2); //change
    }
    item calc(int l, int r){
        return calc(l, r, 0, 0, size);
    }
};
```

```
/// seg tree to find the index of the k-th one
struct seg_tree4 {
   int size;
   vector<int> values;
   int NEUTRAL_ELEMENT = 0;

   int merge(int a, int b){
      return a+b;
   }
   int single(int v){
      return v;
   }

   void init(int n){
      size = 1;
      while (size < n) size *= 2;
      values.resize(2 * size);
   }
```

1

```
void build(vi &a, int x, int lx, int rx){
   if(rx-lx == 1){
      if(lx < (int) a.size()){
         values[x] = single(a[lx]);
      }
      return;
   }
   int mid = (lx+rx)/2;
   build(a, 2*x+1, lx, mid);
   build(a, 2*x+2, mid, rx);
   values[x] = merge(values[2 * x + 1], values[2 * x + 2]); //change
}
void build(vi &a){
   build(a, 0, 0, size);
}
```

2

```cpp
void set(int i, int v, int x, int lx, int rx){
   if(rx-lx == 1){
      values[x] = single(v);
      return;
   }
   int mid = (lx+rx)/2;
   if(i < mid){
      set(i, v, 2*x+1, lx, mid);
   }
   else{
      set(i, v, 2*x+2, mid, rx);
   }
   values[x] = merge(values[2 * x + 1], values[2 * x + 2]); //change
}
void set(int i, int v){
   set(i, v, 0, 0, size);
}
```

3

```cpp
   int find(int k, int x, int lx, int rx){
      if(rx-lx == 1){
         return lx;
      }
      int sl = values[2*x+1];
      int mid = (lx+rx)/2;
      if(k < sl){
         return find(k, 2*x+1, lx, mid);
      } else{
         return find(k-sl, 2*x+2, mid, rx);
      }
   }

   int find(int k){
      return find(k, 0, 0, size);
   }
};
```

4

<u>TRIE</u>

```
const int MAX_CHAR = 26;

struct trie{
   trie* child[MAX_CHAR];
   bool is_leaf;

   trie(){
      memset(child, 0, sizeof child);
      is_leaf = 0;
   }

   void insert(char* str){
      if(*str == '\0')
         is_leaf = 1;
      else{
         int cur = *str - 'a';
         if(child[cur] == 0)
            child[cur] = new trie();
         child[cur]->insert(str+1);
      }
   }
```

```
   bool word_exist(char* str){
      if(*str == '\0')
         return is_leaf;
      int cur = *str - 'a';
      if(child[cur] == 0) return false;
      child[cur]->word_exist(str+1);
   }

   bool is_prefix(char* str){
      if(*str == '\0')
         return true;
      int cur = *str - 'a';
      if(child[cur] == 0) return false;
      child[cur]->is_prefix(str+1);
   }
};
```

KMP

```
vector<ll> prefix_precompute(string pat){
   ll m = s(pat);
   vector<ll> longest_prefix(m);
   for(ll i=1, k = 0; i<m; i++){
      while((k>0) && (pat[k] != pat[i]))
         k = longest_prefix[k-1];
      if(pat[k] == pat[i]) longest_prefix[i] = ++k;
      else longest_prefix[i] = k;
   }
   return longest_prefix;
}
```

```
vector<ll> KMP(string str, string pat){
   ll n = s(str), m = s(pat);
   vector<ll> longest_prefix = prefix_precompute(pat);
   vector<ll> ans;
   for(ll i=0, k=0; i<n; i++){
      while((k>0) && (pat[k] != str[i]))
         k = longest_prefix[k-1];
      if(pat[k] == str[i]) k++;
      if(k == m){
         ans.push_back(i-m+1);
         k = longest_prefix[k-1];
      }
   }
   return  ans;
}
```

```cpp
ll longest_suffix_palindrome(string s){
    string p = s;
    reverse(all(p));
    p+= ("@" + s);
    vector<ll> v = prefix_precompute(p);
    return v[s(v)-1];
}
//minimuim characters to convert str to palindrome by adding chars to the end of string
ll min_chars_str_to_palindrome(string s){
    return s(s) - longest_suffix_palindrome(s);
}
```

DFS and BFS

```cpp
vector<ll> topoSort;
void toposort_bfs(){
    priority_queue<ll> q;
    for(ll i=1; i<=n; i++){
        if(!indeg[i]){
            q.push(-i);
        }
    }
    while(q.size()) {
        ll u = -q.top();
        q.pop();
        topoSort.push_back(u);
        for (auto &v: g[u]) {
            if (!(--indeg[v])) {
                q.push(-v);
            }
        }
    }
}
```

```
const ii M = 1e4+5;
vll g[M];
ll vis[M];

void dfs(ll u){
    vis[u] = 1;
    repa(v, g[u]){
        if(!vis[v]) dfs(v);
    }
}
```

```cpp
const int N = 2e3+5;
ii dist[N][N], n, m, d[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};


void bfs(ll i, ll j){
    dist[i][j] = 0;
    queue<pi> q;
    q.emplace(i, j);
    while(!q.empty()){
        ii x = q.front().F, y = q.front().S; q.pop();
        rep(l, 4){
            ii xx = x + d[l][0], yy = y + d[l][1];
            if(xx >= 0 and xx < n and yy >= 0 and yy < m){
                if(dist[xx][yy] > dist[x][y]+1){
                    dist[xx][yy] = dist[x][y]+1;
                    q.emplace(xx,yy);
                }
            }
        }
    }
}
```

```cpp
// Diameter of a Tree
void dfs(int node, int par, int len = 0) {
    if (mx <= len)
        mx = len, leaf = node;
    for (auto to : g[node]) {
        if (to == par) continue;
        dfs(to, node, len + 1);
    }
}
mx = 0, dfs(1, 1);
mx = 0, dfs(leaf, leaf);
```

DSU and MST

```
const ll N = 2e2+5;
ll rep[N], sz[N], comp, have[N];

void init(){
    rep(i, N) rep[i] = i, sz[i] = 1;
}

ll find(ll u){
    if(rep[u] == u) return u;
    return rep[u] = find(rep[u]);
}

void connect(ll u, ll v){
    u = find(u), v = find(v);
    if(u == v) return;
    if(sz[v] > sz[u]) swap(v, u);
    rep[v] = u;
}

bool is_conneccted(ll u, ll v){
    return (find(u) == find(v));
}
```

```cpp
vector<pair<ll, pll>> edges;

void MST(){
    sort(all(edges));
    rep(i, s(edges)){
        ll x = edges[i].S.F, y = edges[i].S.S, w = edges[i].F;
        if(!is_conneccted(x, y)){
            connect(x, y);
            // could construct the tree by using following operation
            g[x].PB(y);
            g[y].PB(x)
        }
    }
}
```

Shortest Path Algorithms

```
const ll N = 1e5+10;
vg g[N];
ll dist[N], n, m;;
vll p;


void dij(ll src){
    rep(i, n+1) dist[i] = LONG_LONG_MAX;
    priority_queue<pll, vector<pll>, greater<pll>> pq;
    p = vll(n+1, -1);
    dist[src] = 0;
    pq.push({dist[src], src});
    while(!pq.empty()){
        ll node = pq.top().S, cost = pq.top().F; pq.pop();
        if(dist[node] < cost) continue;
        repa(ch, g[node]){
            ll to = ch.F, w = ch.S;
            if(dist[node]+w < dist[to]){
                dist[to] = dist[node]+w;
                p[to] = node;
                pq.push({dist[to], to});
            }
        }
    }
}
```

```
// if it runs to n+1 and still relaxes then there is negative cycle
void floyd(){
    rep(k, n) rep(i, n) rep(j, n) d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
}



vector<pair<ll,pll>> e;
void Bellman(){
    bool negCycle = false;
    dist[0] = 0;
    rep(i, n){
        rep(j, s(e)){
            ll a = e[j].F, b = e[j].S.F, c = e[j].S.S;
            if(dist[b] > dist[a]+c){
                if(i == n-1) negCycle = true;
                dist[b] = dist[a]+c;
            }
        }
    }
}
```

<u>Euler’s Graph</u>

```cpp
// euler cycle start from one node and go through all edges once and return to same node
// in undirected graph if any node have odd number of edges euler cycle fails
// odd degree or ans.size() != m+1
// if directed remove vis from dfs and check that indeg == outdeg for all nodes
void dfs( int node)
{
    while(!g[node].empty())
    {
        int v = g[node].back().F;
        int idx = g[node].back().S;
        g[node].pop_back();
        if(vis[idx])
            continue;
        vis[idx] = 1 ;
        dfs(v);
    }
    ans.push_back(node);
}
```

```
// euler path start from one node and go through all edges once
// in undirected graph if any node have odd number of edges euler cycle fails execpt st and en nodes
// odd degree or ans.size() != m+1
// if directed remove vis from dfs and check that indeg == outdeg for all nodes except
// out[st] != in[st]+1 || in[en] != out[en]+1
void dfs( int node)
{
   while(!g[node].empty())
   {
      int v = g[node].back();
      g[node].pop_back();
      dfs(v);
   }
   ans.push_back(node);
}
```

Tarjan

```cpp
const int N = 3e5 + 2, mod = 1e9 + 7;
int n, m, mx, leaf;
int id, rep[N];
int timer, dfn[N], low[N];
vector<int> g[N], brdg[N];
stack<int> st;
bool in_st[N];
```

```cpp
// create graph with bridges
// call tarjan then compress
void compress() {
   for (int node = 1; node <= n; node++) {
      for (auto to : g[node]) {
         if (rep[node] != rep[to])
            brdg[rep[node]].push_back(rep[to]);
      }
   }
}
```

```cpp
void tarjan(int node, int par = 1) {
   dfn[node] = low[node] = ++timer;
   in_st[node] = 1;
   st.push(node);
   for (auto to : g[node]) {
      if (to == par) continue;
      if (dfn[to] == 0) {
         tarjan(to, node);
         low[node] = min(low[node], low[to]);
      }
      else if (in_st[to])
         low[node] = min(low[node], low[to]);
   }
   if (dfn[node] == low[node]) {
      id++;
      while(true) {
         int cur = st.top();
         st.pop();
         in_st[cur] = 0;
         rep[cur] = id;
         if (cur == node)
            break;
      }
   }
}
```

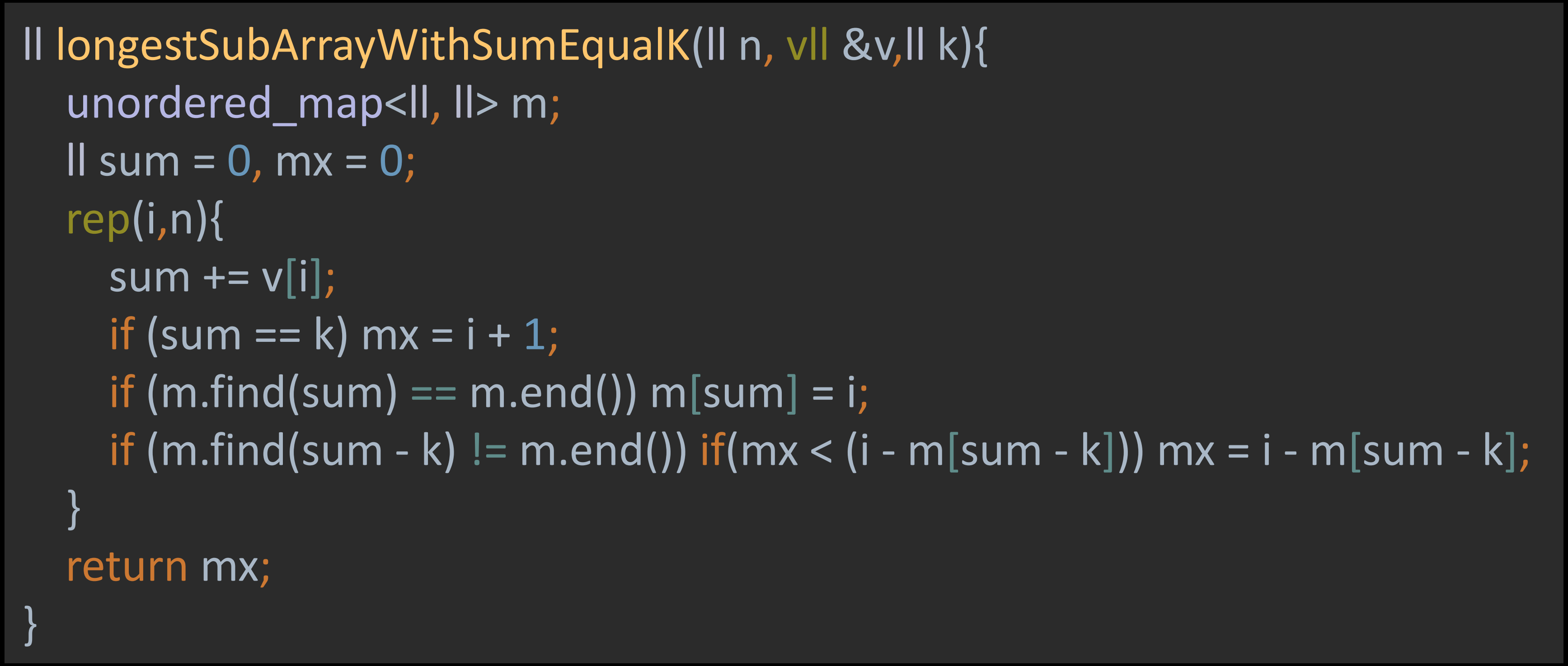
Parallel binary Search

```
void p_bs() {
    priority_queue<pair<int, int>> pq;
    vector<pair<int, int>> inter;
    inter.emplace_back(0, n - 1);
    for (int i = 0; i < inter.size(); i++) {
        int low = inter[i].F, high = inter[i].S;
        if (low > high)
            continue;
        int mid = (low + high) >> 1;
        pq.push({high - low, -mid});
        inter.emplace_back(low, mid - 1);
        inter.emplace_back(mid + 1, high);
    }
    for (int i = 1; i <= n; i++) {
        ans[-pq.top().S] = i;
        pq.pop();
    }
}
```

Various Functions

```cpp
ll longest_subarray_multiple_of_k(vll &v, ll n, ll k)
{
    unordered_map<ll, ll> um;
    ll mod[n], mx = 0;
    ll sum = 0;
    rep(i, n)
    {
        sum += v[i];
        mod[i] = ((sum % k) + k) % k;
        if (mod[i] == 0)
            mx = i + 1;
        else if (um.find(mod[i]) == um.end())
            um[mod[i]] = i;
        else if (mx < (i - um[mod[i]]))
            mx = i - um[mod[i]];
    }
    return mx;
}
```

```
ll f[2000];
void fibonacci()
{
    f[0] = 0, f[1] = 1;
    repn(i, 2, 2000) f[i] = f[i - 1] + f[i - 2];
}
```



```
ll longestSubArrayWithSumEqualK(ll n, vll &v,ll k){
    unordered_map<ll, ll> m;
    ll sum = 0, mx = 0;
    rep(i,n){
        sum += v[i];
        if (sum == k) mx = i + 1;
        if (m.find(sum) == m.end()) m[sum] = i;
        if (m.find(sum - k) != m.end()) if(mx < (i - m[sum - k])) mx = i - m[sum - k];
    }
    return mx;
}
```

```cpp
string hexa_to_binary(string s) {
   string ret = "";
   for (auto i : s) {
      if (i == '0') ret += "0000";
      if (i == '1') ret += "0001";
      if (i == '2') ret += "0010";
      if (i == '3') ret += "0011";
      if (i == '4') ret += "0100";
      if (i == '5') ret += "0101";
      if (i == '6') ret += "0110";
      if (i == '7') ret += "0111";
      if (i == '8') ret += "1000";
      if (i == '9') ret += "1001";
      if (i == 'A') ret += "1010";
      if (i == 'B') ret += "1011";
      if (i == 'C') ret += "1100";
      if (i == 'D') ret += "1101";
      if (i == 'E') ret += "1110";
      if (i == 'F') ret += "1111";
   }
   return ret;
}
```

```
const int N = 5e5 + 2;
ll n, r, k, a[N], pref[N];
bool sliding_window(ll mid) {
    ll sum = 0, rem = k;
    deque<pair<int, ll>> dq;
    for (int i = 1; i <= n; i++) {
        // deque
        if (!dq.empty() && dq.front().F < i) {
            sum -= dq.front().S;
            dq.pop_front();
        }
        // check for need
        ll dx = mid - (a[i] + sum);
        if (dx > 0) {
            if (dx > rem)
                return false;
            rem -= dx, sum += dx;
            dq.emplace_back(i + 2 * r, dx);
        }
    }
    return true;
}
```

```
bool is_palindrome_s(string s) {
    string p = s;
    reverse(all(p));
    if (s == p) return true;
    else return false;
}
bool is_palindrome_n(ll n) {
    ll temp = n, sum =0 ,r;
    while(n>0)
    {
        r=n%10;
        sum = (sum*10)+r;
        n/=10;
    }
    if (temp == sum) return true;
    else return false;
}
```

```
vector<pair<char, ll>> subs(string a){
    vector<pair<char, ll>> ans;
    char cur = a[0];
    ll cnt = 0;
    rep(i, s(a)){
        cnt++;
        if(a[i] != cur){
            ans.emplace_back(cur, cnt);
            cur = a[i];
            cnt = 1;
        }
        if((i == s(a)-1) && (cnt > 0)){
            ans.emplace_back(a[s(a)-1], cnt);
        }
    }
    return ans;
}
```

```
bool is_Subsequence(string a, string b) {
    if (b == a)
        return true;
    if (a.length() > b.length()) {
        swap(a, b);
    }
    int n = b.size();
    int m = a.size();
    int j = 0;
    for (int i = 0; i < n; i++) {
        if (a[j] == b[i])
            j++;
        if (j == a.size())
            return true;
    }
    return false;
}
```

```
bool isPower(ll x, ll y) {
   // logarithm function to calculate value
   ll res1 = log(y) / log(x);
   double res2 = log(y) / log(x); // Note : this is double

   // compare to the result1 or result2 both are equal
   return (res1 == res2);
}
```

String Functions

```
cin.ignore();
getline(cin , string1);

string a = “Omar”, b = “ma”;
a.find(b); //returns 1
string a = “Omar”, b = “ama”;
a.find(b); //returns string::npos(-1)

string a = “Omar”;
a.resize(7, ‘*’); //returns “Omar***”
a.resize(2); //returns “Om”
a.resize(3, ‘*’); //returns “Oma”

string a = “Omar”, b = “”, c;
a.empty(); //returns false (0)
b.empty; //returns true (1)
c.empty(); //returns true (1)
```

```
string name = "Omar";
cout << name.back(); //Outputs ‘r’
name.front() = ‘D’; // makes name equal to “Dmar”

string name = "Omar";
cout << name.substr(1,2); //Outputs “ma”

string name = "Omar";
cout << isupper(name[0]);  // returns true
cout << islower(‘b’);  // returns true
cout << isupper(name[1])  // returns false

string name = "Omar";
name = toupper(name[1]);  // name = “OMar”
cout << tolower(‘B’); // b
```