

# Competitive Programmer's CodeBook

MIST\_EaglesExpr

Syed Mafijul Islam, 202214105  
Md. Tanvin Sarkar Pallab, 202214062  
Shihab Ahmed, 202314049

October 26, 2024

# Contents

<b>1</b>	<b>Useful Tips</b>	<b>3</b>
<b>2</b>	<b>Formula</b>	<b>3</b>
2.1	Area Formula . . . . .	3
2.2	Perimeter Formulas . . . . .	3
2.3	Volume Formula . . . . .	3
2.4	Surface Area Formula . . . . .	3
2.5	Triangles . . . . .	3
2.6	Summation Of Series . . . . .	3
2.7	Miscellaneous . . . . .	4
<b>3</b>	<b>Graph Theory</b>	<b>4</b>
3.1	BFS . . . . .	4
3.2	DFS . . . . .	4
3.3	Dijkstra Algorithm . . . . .	4
3.4	Bellman Ford . . . . .	4
3.5	Floyed Warshall Algorithm . . . . .	5
3.6	Kruskal Algorithm (MST ) . . . . .	5
3.7	Prims Algorithm (MST ) . . . . .	5
3.8	Strongly Connected Components . . . . .	5
3.9	LCA . . . . .	6
3.10	Max Flow . . . . .	7
<b>4</b>	<b>Data Structures</b>	<b>7</b>
4.1	Segment Tree . . . . .	7
4.2	Segment Tree Lazy . . . . .	7
4.3	Fenwick Tree . . . . .	8
4.4	Disjoint Set . . . . .	8
4.5	TRIE . . . . .	8
4.6	Set Balancing . . . . .	9
<b>5</b>	<b>Algorithms</b>	<b>9</b>
5.1	KMP . . . . .	9
5.2	Monotonic Stack (Immediate Small) . . . . .	9
5.3	Kadane's Algorithm . . . . .	10
5.4	2D Prefix Sum . . . . .	10
<b>6</b>	<b>Number Theory</b>	<b>10</b>
6.1	nCr . . . . .	10
6.2	Power . . . . .	10
6.3	Miller Rabin . . . . .	10
6.4	Sieve . . . . .	10
6.5	Bitset Sieve . . . . .	11
6.6	Divisors . . . . .	11
6.7	Euler's Totient Phi Function . . . . .	11
6.8	Log a base b . . . . .	11
6.9	Count 1's from 0 to n . . . . .	11
6.10	Primes Upto $1e9$ . . . . .	11
<b>7</b>	<b>Dynamic Programming</b>	<b>12</b>
7.1	LCS (Longest Common Subsequence) . . . . .	12
7.2	LIS (Longest Increasing Subsequence) . . . . .	13
7.3	SOS (Sum Of Subsets) . . . . .	13



## 2.7 Miscellaneous

- $2^{100} = 2^{50} * 2^{50}$
- $\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$
- $\log n! = \log 1 + \log 2 + \dots + \log n$
- $\gcd(a, b) = \gcd(a - b, b)$
- Number of occurrence of a prime number  $p$  in  $n!$  is  $\lfloor \frac{n}{p} \rfloor + \lfloor \frac{n}{p^2} \rfloor + \lfloor \frac{n}{p^3} \rfloor + \dots 0$
- Formula of Catalan number is  $\binom{2n}{n} - \binom{2n}{n-1}$
- Number of divisors of  $p^x q^y$  where  $p$  and  $q$  are prime is  $(x+1) * (y+1)$
- Sum of divisors of  $p^x q^y$  where  $p$  and  $q$  are prime is  $(1+p+p^2+\dots+p^x)(1+q+q^2+\dots+q^y)$

## 3 Graph Theory

All about graph.

### 3.1 BFS

```
void bfs(int start, int target = -1) {
    queue<int> q;
    q.push(start);
    vis[start] = true;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i : adj[u]) {
            if (!vis[i]) {
                vis[i] = true;
                q.push(i);
            }
        }
    }
}
```

### 3.2 DFS

```
map<int, vector<int>> adj;
map<int, int> visited, parent, level, color;

void dfs(int start)
{
    visited[start]=1;
    for (auto child : adj[start])
    {
        if (!visited[child])
        {
            dfs(child);
        }
    }
}
```

```
visited[start]=2;
}
```

### 3.3 Dijkstra Algorithm

```
void Dijkstra(int start) {
    // vector<pair<int, int>> adj[N];
    priority_queue<pair<int, int>>, greater<
        pair<int, int>>> pq;
    pq.push({0, start});
    while (!pq.empty()) {
        auto it = pq.top();
        pq.pop();
        int wt = it.first;
        int u = it.second;
        if (vis[u])
            continue;
        vis[u] = 1;
        for (pair<int, int> i : adj[u]) {
            int adjWt = i.second;
            int adjNode = i.first;
            if (dist[adjNode] > wt + adjWt)
            {
                dist[adjNode] = wt + adjWt;
                pq.push({dist[adjNode],
                    adjNode});
            }
        }
    }
}
```

### 3.4 Bellman Ford

```
vector<int> dist;
vector<int> parent;
vector<vector<pair<int, int>>> adj;
// resize the vectors from main
function
void bellmanFord(int num_of_nd, int
    src) {
    dist[src] = 0;
    for (int step = 0; step < num_of_nd;
        step++) {
        for (int i = 1; i <= num_of_nd; i
            ++){
            for (auto it : adj[i]) {
                int u = i;
                int v = it.first;
                int wt = it.second;
                if (dist[u] != inf &&
                    ((dist[u] + wt) < dist[v])
                ) {
                    if (step == num_of_nd - 1) {
                        cout << "Negative cycle
                            found\n ";
                        return;
                    }
                    dist[v] = dist[u] + wt;
                    parent[v] = u;
                }
            }
        }
    }
}
```

```

    for (int i = 1; i <= num_of_nd; i++)
        cout << dist[i] << " ";
    cout << endl;
}

```

### 3.5 Floyd Warshall Algorithm

```

typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
typedef vector<int> VI;
typedef vector<VI> VVI;
bool FloydWarshall(VVT &w, VVI &prev)
{
    int n = w.size();
    prev = VVI(n, VI(n, -1));
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (w[i][j] > w[i][k] + w[k][j]) {
                    w[i][j] = w[i][k] + w[k][j];
                    prev[i][j] = k;
                }
            }
        }
    }
    for (int i = 0; i < n; i++)
        if (w[i][i] < 0)
            return false;
    return true;
}

```

### 3.6 Kruskal Algorithm (MST)

```

vector<pair<int, pair<int, int>>>
Kruskal(vector<pair<int, pair<int, int>>> &edges, int n) {
    sort(edges.begin(), edges.end());
    vector<pair<int, pair<int, int>>>
        ans;
    DisjointSet D(n);
    for (auto it : edges) {
        if (D.findUPar(it.second.first) !=
            D.findUPar(it.second.second))
        {
            ans.push_back({it.first, {it.
                second.first, it.second.
                second}});
            D.unionBySize(it.second.first,
                it.second.second);
        }
    }
    return ans;
}

```

### 3.7 Prims Algorithm (MST)

```

void Prims(int start) {
    // map<int, vector<pair<int, int>>>
    adj, ans;
    priority_queue<pair<int, pair<int,
        int>>, vector<pair<int, pair<int,
        int>>>, greater<pair<int, pair
        <int, int>>>> pq;

```

```

pq.push({0, {start, -1}});
while (!pq.empty()) {
    auto it = pq.top();
    pq.pop();
    int wt = it.first;
    int u = it.second.first;
    int v = it.second.second;
    if (vis[u]) continue;
    vis[u] = 1;
    if (v != -1) ans[u].push_back({v,
        wt});
    for (pair<int, int> i : adj[u]) {
        int adjWt = i.second;
        int adjNode = i.first;
        if (!vis[adjNode]) pq.push({
            adjWt, {adjNode, u}});
    }
}
}

```

### 3.8 Strongly Connected Components

```

vector<bool> visited; // keeps track
of which vertices are already
visited

```

```

// runs depth first search starting at
vertex v.

```

```

// each visited vertex is appended to
the output vector when dfs leaves
it.

```

```

void dfs(int v, vector<vector<int>>
    const &adj, vector<int> &output) {
    visited[v] = true;
    for (auto u : adj[v])
        if (!visited[u])
            dfs(u, adj, output);
    output.push_back(v);
}

```

```

// input: adj -- adjacency list of G
// output: components -- the strongly
connected components in G
// output: adj_cond -- adjacency list
of G^SCC (by root vertices)

```

```

void scc(vector<vector<int>> const &
    adj, vector<vector<int>> &
    components, vector<vector<int>> &
    adj_cond) {
    int n = adj.size();
    components.clear(), adj_cond.clear()
    ;

```

```

    vector<int> order; // will be a
sorted list of G's vertices by
exit time

```

```

    visited.assign(n, false);

```

```

    // first series of depth first
searches

```

```

    for (int i = 0; i < n; i++)

```

```

    if (!visited[i])
        dfs(i, adj, order);

    // create adjacency list of G^T
    vector<vector<int>> adj_rev(n);
    for (int v = 0; v < n; v++)
        for (int u : adj[v])
            adj_rev[u].push_back(v);

    visited.assign(n, false);
    reverse(order.begin(), order.end());

    vector<int> roots(n, 0); // gives
                             the root vertex of a vertex's
                             SCC

    // second series of depth first
    searches
    for (auto v : order)
        if (!visited[v]) {
            std::vector<int> component;
            dfs(v, adj_rev, component);
            components.push_back(component);
            int root = *min_element(begin(
                component), end(component));
            for (auto u : component)
                roots[u] = root;
        }

    // add edges to condensation graph
    adj_cond.assign(n, {});
    for (int v = 0; v < n; v++)
        for (auto u : adj[v])
            if (roots[v] != roots[u])
                adj_cond[roots[v]].push_back(
                    roots[u]);
}

```

### 3.9 LCA

```

struct LCA {
    vector<int> height, euler, first,
        segtree, parent;
    vector<bool> visited;
    vector<vector<int>> jump;

    int n;

    LCA(vector<vector<int>> &adj, int
        root = 0) {
        n = adj.size();
        height.resize(n);
        first.resize(n);
        parent.resize(n);
        euler.reserve(n * 2);
        visited.assign(n, false);
        dfs(adj, root);
        int m = euler.size();
        segtree.resize(m * 4);
        build(1, 0, m - 1);

        jump.resize(n, vector<int>(32, -1)
            );
    }
}

```

```

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

    for(int j=1;j<20;j++) {
        for(int i=0;i<n;i++) {
            int mid = jump[i][j-1];
            if(mid != -1) jump[i][j] =
                jump[mid][j-1];
        }
    }
}

void dfs(vector<vector<int>> &adj,
    int node, int h = 0) {
    visited[node] = true;
    height[node] = h;
    first[node] = euler.size();
    euler.push_back(node);
    for (auto to : adj[node]) {
        if (!visited[to]) {
            parent[to] = node;
            dfs(adj, to, h + 1);
            euler.push_back(node);
        }
    }
}

void build(int node, int b, int e) {
    if (b == e) {
        segtree[node] = euler[b];
    } else {
        int mid = (b + e) / 2;
        build(node << 1, b, mid);
        build(node << 1 | 1, mid + 1, e)
            ;
        int l = segtree[node << 1], r =
            segtree[node << 1 | 1];
        segtree[node] = (height[l] <
            height[r]) ? l : r;
    }
}

int query(int node, int b, int e,
    int L, int R) {
    if (b > R || e < L)
        return -1;
    if (b >= L && e <= R)
        return segtree[node];
    int mid = (b + e) >> 1;

    int left = query(node << 1, b, mid
        , L, R);
    int right = query(node << 1 | 1,
        mid + 1, e, L, R);
    if (left == -1)
        return right;
    if (right == -1)
        return left;
    return height[left] < height[right
        ] ? left : right;
}

```

```

int lca(int u, int v) {
    int left = first[u], right = first[v];
    if (left > right)
        swap(left, right);
    return query(1, 0, euler.size() - 1, left, right);
}

int kthParent(int u, int k) {
    for(int i=0; i<19; i++) {
        if(k & (1LL<<i)) u = jump[u][i];
    }
    return u;
}
};

```

### 3.10 Max Flow

```

const int N = 505;
int capacity[N][N];
int vis[N], p[N];
int n, m;

int bfs(int s, int t) {
    memset(vis, 0, sizeof vis);
    queue<int> qu;
    qu.push(s);
    vis[s] = 1;
    while (!qu.empty()) {
        int u = qu.front();
        qu.pop();
        for (int i = 0; i <= n + m + 2; i++) {
            if (capacity[u][i] > 0 && !vis[i]) {
                p[i] = u;
                vis[i] = 1;
                qu.push(i);
            }
        }
    }
    return vis[t] == 1;
}

int maxflow(int s, int t) {
    int cnt = 0;
    while (bfs(s, t)) {
        int cur = t;
        while (cur != s) {
            int prev = p[cur];
            capacity[prev][cur] -= 1;
            capacity[cur][prev] += 1;
            cur = prev;
        }
        cnt++;
    }
    return cnt;
}

```

## 4 Data Structures

Different Data Structures.

### 4.1 Segment Tree

```

constexpr int N = 100005;
int arr[N], seg[N];

void build(int ind, int low, int high)
{
    if (low == high) {
        seg[ind] = arr[low];
        return;
    }
    int mid = (low + high) / 2;
    build(2 * ind + 1, low, mid);
    build(2 * ind + 2, mid + 1, high);
    seg[ind] = seg[2 * ind + 1] + seg[2 * ind + 2];
}

int query(int ind, int low, int high, int l, int r) {
    if (low >= l && high <= r) return seg[ind];
    if (low > r || high < l) return 0;
    int mid = (low + high) / 2;
    int left = query(2 * ind + 1, low, mid, l, r);
    int right = query(2 * ind + 2, mid + 1, high, l, r);
    return left + right;
}

void update(int ind, int low, int high, int node, int val) {
    if (low == high) {
        seg[ind] = val;
        return;
    }
    int mid = (low + high) / 2;
    if (low <= node && node <= mid)
        update(2 * ind + 1, low, mid, node, val);
    else update(2 * ind + 2, mid + 1, high, node, val);
    seg[ind] = seg[2 * ind + 1] + seg[2 * ind + 2];
}

```

### 4.2 Segment Tree Lazy

```

const int N = 1e5 + 5;
int arr[N], seg[N << 2], lz[N << 2];

void pull(int node, int l, int r) {
    seg[node] = seg[node << 1] + seg[node << 1 | 1];
}

void push(int node, int l, int r) {
    int mid = (l + r) >> 1;
    lz[node << 1] += lz[node];
    seg[node << 1] += lz[node] * (mid - l + 1);
    lz[node << 1 | 1] += lz[node];
    seg[node << 1 | 1] += lz[node] * (r - mid);
    lz[node] = 0;
}

```

```

}

void build(int node, int l, int r) {
    if(l == r) {
        seg[node] = arr[l];
        lz[node] = 0;
        return;
    }
    int mid = (l + r) >> 1;
    build(node << 1, l, mid);
    build(node << 1 | 1, mid + 1, r);
    pull(node, l, r);
}

void update(int node, int l, int r,
            int ql, int qr, int val) {
    if(qr < l || r < ql) return;
    if(ql <= l && r <= qr) {
        seg[node] += val;
        lz[node] += val;
        return;
    }
    push(node, l, r);

    int mid = (l + r) >> 1;
    update(node << 1, l, mid, ql, qr,
           val);
    update(node << 1 | 1, mid + 1, r, ql,
           qr, val);

    pull(node, l, r);
}

int query(int node, int l, int r, int
          ql, int qr) {
    if(qr < l || r < ql) return 0;
    if(ql <= l && r <= qr) return seg[
        node];

    push(node, l, r);

    int mid = (l + r) >> 1;
    return query(node << 1, l, mid, ql,
                 qr) + query(node << 1 | 1, mid +
                             1, r, ql, qr);
}

```

### 4.3 Fenwick Tree

```

int fenwick[N];

void update(int ind, int val) {
    while (ind < N) {
        fenwick[ind] += val;
        ind += ind & -ind;
    }
}

int query(int ind) {
    int sum = 0;
    while (ind > 0) {
        sum += fenwick[ind];
        ind -= ind & -ind;
    }
}

```

```

    return sum;
}

```

### 4.4 Disjoint Set

```

class DisjointSet {
    vector<int> parent, sz;

public:
    DisjointSet(int n) {
        sz.resize(n + 1);
        parent.resize(n + 2);
        for (int i = 1; i <= n; i++)
            parent[i] = i, sz[i] = 1;
    }

    int findUPar(int u) { return parent[
        u] == u ? u : parent[u] =
        findUPar(parent[u]); }

    void unionBySize(int u, int v) {
        int a = findUPar(u);
        int b = findUPar(v);
        if (sz[a] < sz[b]) swap(a, b);
        if (a != b) {
            parent[b] = a;
            sz[a] += sz[b];
        }
    }
};

```

### 4.5 TRIE

```

const int N = 26;
class Node {
public:
    int EoW;
    Node* child[N];
    Node() {
        EoW = 0;
        for (int i = 0; i < N; i++) child[
            i] = NULL;
    }
};

void insert(Node* node, string s) {
    for (size_t i = 0; i < s.size(); i
        ++i) {
        int r = s[i] - 'A';
        if (node->child[r] == NULL) node->
            child[r] = new Node();
        node = node->child[r];
    }
    node->EoW += 1;
}

int search(Node* node, string s) {
    for (size_t i = 0; i < s.size(); i
        ++i) {
        int r = s[i] - 'A';
        if (node->child[r] == NULL) return
            0;
    }
    return node->EoW;
}

```



```

void print(Node* node, string s = "")
{
    if (node->EoW) cout << s << "\n";
    for (int i = 0; i < N; i++) {
        if (node->child[i] != NULL) {
            char c = i + 'A';
            print(node->child[i], s + c);
        }
    }
}

bool isChild(Node* node) {
    for (int i = 0; i < N; i++)
        if (node->child[i] != NULL) return true;
    return false;
}

bool isJunc(Node* node) {
    int cnt = 0;
    for (int i = 0; i < N; i++) {
        if (node->child[i] != NULL) cnt++;
    }
    if (cnt > 1) return true;
    return false;
}

int trie_delete(Node* node, string s,
    int k = 0) {
    if (node == NULL) return 0;
    if (k == (int)s.size()) {
        if (node->EoW == 0) return 0;
        if (isChild(node)) {
            node->EoW = 0;
            return 0;
        }
        return 1;
    }
    int r = s[k] - 'A';
    int d = trie_delete(node->child[r],
        s, k + 1);
    int j = isJunc(node);
    if (d) delete node->child[r];
    if (j) return 0;
    return d;
}

void delete_trie(Node* node) {
    for (int i = 0; i < 15; i++) {
        if (node->child[i] != NULL)
            delete_trie(node->child[i]);
    }
    delete node;
}

```

## 4.6 Set Balancing

```

// return middle element of the set
void balance(multiset<int> right,
    multiset<int> &left) {
    while (true) {
        int st = right.size();
        int sl = left.size();

```

```

        if (st == sl || st == sl + 1)
            break;
        if (st < sl) right.insert(left.
            begin()), left.erase(left.
            begin());
        else left.insert(right.rbegin()),
            right.erase(right.rbegin());
    }
}

void insert_in_set(multiset<int> &
    right, multiset<int> &left, int
    value) {
    if (right.empty()) right.insert(
        value);
    else {
        auto it = right.end();
        it--;
        if (value < *it) right.insert(
            value);
        else left.insert(value);
    }
}

```

## 5 Algorithms

All about algorithms.

### 5.1 KMP

```

vector<int> prefix_function(string s)
{
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j])
            j = pi[j - 1];
        if (s[i] == s[j]) j++;
        pi[i] = j;
    }
    return pi;
}

vector<int> find_matches(string text,
    string pat) {
    int n = pat.length(), m = text.
        length();
    string s = pat + "$" + text;
    vector<int> pi = prefix_function(s), ans;
    for (int i = n; i <= n + m; i++) {
        if (pi[i] == n) {
            ans.push_back(i - 2 * n);
        }
    }
    return ans;
}

```

### 5.2 Monotonic Stack (Immediate Small)

```

for (int i = n - 1; i >= 0; i--) {
    while (!stk.empty() && v[i] >= v[stk
        .top()]) stk.pop();
    ind[i] = stk.empty() ? -1 : stk.top
        ();
    stk.push(i);
}
// 3 1 5 4 10
// 2 2 4 4 -1

```

### 5.3 Kadane's Algorithm

```

int maxSubArraySum(vector<int> &a) {
    int size = a.size();
    int maxTill = INT_MIN, maxEnd = 0;
    for (int i = 0; i < size; i++) {
        maxEnd = maxEnd + a[i];
        if (maxTill < maxEnd) maxTill =
            maxEnd;
        if (maxEnd < 0) maxEnd = 0;
    }
    return maxTill;
}

```

### 5.4 2D Prefix Sum

```

pref[i][j] = a[i][j] + pref[i - 1][j]
    + pref[i][j - 1] - pref[i - 1][j -
        1];

```

## 6 Number Theory

All about math.

### 6.1 nCr

```

int inverseMod(int a, int m) { return
    power(a, m - 2); }

int nCr(int n, int r, int m = mod){
    if(r==0) return 1;
    if(r>n) return 0;
    return (fact[n] * inverseMod((fact[
        r] * fact[n-r]) % m , m)) % m;
}

```

### 6.2 Power

```

int power(int base, int n, int m = mod
) {
    if (n == 0) return 1;
    if (n & 1) {
        int x = power(base, n / 2);
        return ((x * x) % m * base) % m;
    }
    else {
        int x = power(base, n / 2);
        return (x * x) % m;
    }
}

```

### 6.3 Miller Rabin

```

using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod)
{
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1) result = (u128)result *
            base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64
    d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1) return
        false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1) return false;
    }
    return true;
};

bool MillerRabin(u64 n, int iter = 5)
{
    // returns true if n is
    // probably prime, else returns false
    if (n < 4) return n == 2 || n == 3;
    int s = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        s++;
    }

    for (int i = 0; i < iter; i++) {
        int a = 2 + rand() % (n - 3);
        if (check_composite(n, a, d, s))
            return false;
    }
    return true;
}

```

### 6.4 Sieve

```

const int N = 1e7 + 3;
vector<int> primes;
int notprime[N];

void sieve() {
    primes.push_back(2);
    for (int i = 2; i < N; i += 2) {
        notprime[i] = true;
    }
    for (int i = 3; i < N; i += 2) {
        if (!notprime[i]) {
            primes.push_back(i);
            for (int j = i * i; j < N; j +=
                2 * i) {

```

```

        notprime[j] = true;
    }
}
}
}

```

## 6.5 Bitset Sieve

```

const int sieve_size = 10000006;
bitset<sieve_size> sieve;

void Sieve() {
    sieve.flip();
    int finalBit = sqrt(sieve.size()) +
        1;
    for (int i = 2; i < finalBit; ++i) {
        if (sieve.test(i))
            for (int j = 2 * i; j <
                sieve_size; j += i) sieve.
                reset(j);
    }
}

```

## 6.6 Divisors

```

constexpr int N = 1000005;

int Prime[N + 4], kk;
bool notPrime[N + 5];
void SieveOf() {
    notPrime[1] = true;
    Prime[kk++] = 2;
    for (int i = 4; i <= N; i += 2)
        notPrime[i] = true;
    for (int i = 3; i <= N; i += 2) {
        if (!notPrime[i]) {
            Prime[kk++] = i;
            for (int j = i * i; j <= N; j +=
                2 * i) notPrime[j] = true;
        }
    }
}

void Divisors(int n) {
    int sum = 1, total = 1;
    int mnP = INT_MAX, mxP = INT_MIN,
        cntP = 0, totalP = 0;
    for (int i = 0; i <= N && Prime[i] *
        Prime[i] <= n; i++) {
        if (n % Prime[i] == 0) {
            mnP = min(mnP, Prime[i]);
            mxP = max(mnP, Prime[i]);
            int k = 0;
            cntP++;
            while (n % Prime[i] == 0) {
                k++;
                n /= Prime[i];
            }

            sum *= (k + 1); // NOD
            totalP += k;
            int s = 0, p = 1;
            while (k-- >= 0) {
                s += p;

```

```

                p *= Prime[i];
            };
            total *= s; // SOD
        }
    }
    if (n > 1) {
        cntP++, totalP++;
        sum *= 2;
        total *= (1 + n);
        mnP = min(mnP, n);
        mxP = max(mnP, n);
    }
    cout << mnP << " " << mxP << " " <<
        cntP << " " << totalP << " " <<
        sum << " " << total << "\n";
}

```

## 6.7 Euler's Totient Phi Function

```

const int N = 5000005;
int phi[N];
unsigned long long phiSum[N];
void phiCalc() {
    for (int i = 2; 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;
            }
        }
    }
    for (int i = 2; i < N; i++) {
        phiSum[i] = (unsigned long long)
            phi[i] * (unsigned long long)
            phi[i] + phiSum[i - 1];
    }
}

```

## 6.8 Log a base b

```

int logab (int a, int b){
    return log2(a) / log2(b);
}

```

## 6.9 Count 1's from 0 to n

```

int cntOnes(int n) {
    int cnt = 0;
    for(int i=1;i<=n;i<=1) {
        int x = (n + 1) / (i << 1);
        cnt += x * i;
        if((n + 1) % i && n & i) cnt += (n
            + 1) % i;
    }
    return cnt;
}

```

## 6.10 Primes Upto 1e9

```

// credit: min_25
// takes 0.5s for n = 1e9
vector<int> sieve(const int N, const
    int Q = 17, const int L = 1 << 15)
{

```

```

static const int rs[] = {1, 7, 11,
    13, 17, 19, 23, 29};
struct P {
    P(int p) : p(p) {}
    int p; int pos[8];
};
auto approx_prime_count = [] (const
    int N) -> int {
    return N > 60184 ? N / (log(N) -
        1.1)
        : max(1., N / (
            log(N) -
            1.11)) + 1;
};

const int v = sqrt(N), vv = sqrt(v);
vector<bool> isp(v + 1, true);
for (int i = 2; i <= vv; ++i) if (
    isp[i]) {
    for (int j = i * i; j <= v; j += i
        ) isp[j] = false;
}

const int rsize = approx_prime_count
    (N + 30);
vector<int> primes = {2, 3, 5}; int
    psize = 3;
primes.resize(rsize);

vector<P> sprimes; size_t pbeg = 0;
int prod = 1;
for (int p = 7; p <= v; ++p) {
    if (!isp[p]) continue;
    if (p <= Q) prod *= p, ++pbeg,
        primes[psize++] = p;
    auto pp = P(p);
    for (int t = 0; t < 8; ++t) {
        int j = (p <= Q) ? p : p * p;
        while (j % 30 != rs[t]) j += p
            << 1;
        pp.pos[t] = j / 30;
    }
    sprimes.push_back(pp);
}

vector<unsigned char> pre(prod, 0xFF
    );
for (size_t pi = 0; pi < pbeg; ++pi)
    {
        auto pp = sprimes[pi]; const int p
            = pp.p;
        for (int t = 0; t < 8; ++t) {
            const unsigned char m = ~(1 << t
                );
            for (int i = pp.pos[t]; i < prod
                ; i += p) pre[i] &= m;
        }
    }

const int block_size = (L + prod -
    1) / prod * prod;
vector<unsigned char> block(
    block_size); unsigned char*

```

```

    pblock = block.data();
    const int M = (N + 29) / 30;

    for (int beg = 0; beg < M; beg +=
        block_size, pblock -= block_size
        ) {
        int end = min(M, beg + block_size)
            ;
        for (int i = beg; i < end; i +=
            prod) {
            copy(pre.begin(), pre.end(),
                pblock + i);
        }
        if (beg == 0) pblock[0] &= 0xFE;
        for (size_t pi = pbeg; pi <
            sprimes.size(); ++pi) {
            auto& pp = sprimes[pi];
            const int p = pp.p;
            for (int t = 0; t < 8; ++t) {
                int i = pp.pos[t]; const
                    unsigned char m = ~(1 << t
                        );
                for (; i < end; i += p) pblock
                    [i] &= m;
                pp.pos[t] = i;
            }
        }
        for (int i = beg; i < end; ++i) {
            for (int m = pblock[i]; m > 0; m
                &= m - 1) {
                primes[psize++] = i * 30 + rs[
                    __builtin_ctz(m)];
            }
        }
        assert(psize <= rsize);
        while (psize > 0 && primes[psize -
            1] > N) --psize;
        primes.resize(psize);
        return primes;
    }
}

```

## 7 Dynamic Programming

### 7.1 LCS (Longest Common Subsequence)

```

string s, t;
vector<vector<int>> dp(3003, vector<
    int>(3003, -1));
vector<vector<int>> mark(3003, vector<
    int>(3003));

int f(int i, int j) {
    if (i < 0 || j < 0) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    int res = 0;
    if (s[i] == t[j]) {
        mark[i][j] = 1;
        res = 1 + f(i - 1, j - 1);
    }
    else {
        int iC = f(i - 1, j);

```

```

    int jC = f(i, j - 1);
    if (iC > jC) mark[i][j] = 2;
    else mark[i][j] = 3;
    res = max(iC, jC);
}
return dp[i][j] = res;
}

void printWay(int i, int j) {
    if (i < 0 || j < 0) return;
    if (mark[i][j] == 1) printWay(i - 1,
        j - 1), cout << s[i];
    else if (mark[i][j] == 2) printWay(i
        - 1, j);
    else if (mark[i][j] == 3) printWay(i
        , j - 1);
}

```

## 7.2 LIS (Longest Increasing Subsequence)

```

void lis(vector<int> &v) {
    int n = v.size();
    vector<int> dp(n + 1, 1), hash(n);
    int mx = 1, lastInd = 0;
    for (int i = 0; i < n; i++) {
        hash[i] = i;
        for (int prev = 0; prev < i; prev
            ++){
            if (v[i] > v[prev] && 1 + dp[
                prev] > dp[i]) {
                dp[i] = 1 + dp[prev];
                hash[i] = prev;
            }
        }
        if (mx < dp[i]) {
            mx = dp[i];
            lastInd = i;
        }
    }
    vector<int> printSeq;

```

```

    printSeq.push_back(v[lastInd]);
    while (hash[lastInd] != lastInd) {
        lastInd = hash[lastInd];
        printSeq.push_back(v[lastInd]);
    }
    reverse(printSeq.begin(), printSeq.
        end());
    cout << mx << "\n";
    for (int i : printSeq) cout << i <<
        " ";
    cout << "\n";
}

```

## 7.3 SOS (Sum Of Subsets)

```

void SOS (vector<int> &v) {
    const int BITS = log2(*max_element(v
        .begin(), v.end())) + 1;
    vector<int> freq(1 << BITS, 0);
    for (int mask : v) freq[mask]++;

    vector<int> dp(1 << BITS);
    for (int mask = 0; mask < 1 << BITS;
        mask++) {
        dp[mask] = 0;
        int submask = mask;
        do {
            dp[mask] += freq[submask];
            submask = (submask - 1) & mask;
        } while (submask);
    }

    for (int mask : v) cout << dp[mask]
        << '\n';
}

// If k = number of 1's in the integer
// n. Then total submask = 2^k.
// 5
// 3 7 2 9 2
// for 3 -> 3, 2, 2 are submasks.

```