

# Competitive Programming Reference

First, solve the problem. Then, write the code.

---

John Johnson

By

Sergio Gabriel Sanchez Valencia

[gabrielsanv97@gmail.com](mailto:gabrielsanv97@gmail.com)

searleser97

# Contents

<b>1 Coding Resources</b>	<b>3</b>	<b>4 Maths</b>	<b>13</b>
1.1 C++	3	4.1 Number Theory	13
1.1.1 Decimal Precision	3	4.1.1 Divisibility Criterion	13
1.1.2 IO Optimization	3	4.1.2 Extended Euclidean	13
1.1.3 Int To Binary	3	4.1.3 GCD	14
1.1.4 Map Value To Int	3	4.1.4 LCM	14
1.1.5 Permutations	3	4.1.5 Prime Check Miller Rabin	14
1.1.6 Print Vector	3	4.1.6 Prime Sieve	14
1.1.7 Priority Queue Of Object	3	<b>5 Strings</b>	<b>14</b>
1.1.8 Random	3	5.1 KMP	14
1.1.9 Read Line	4	5.2 Rabin Karp	15
1.1.10 Sort Pair	4	<b>6 Techniques</b>	<b>15</b>
1.1.11 Sort Vector Of Object	4	6.1 Binary Search	15
1.1.12 Split String	4	6.2 Multiple Queries	15
1.1.13 Typedef	4	6.2.1 Mo	15
1.2 Python	4	6.2.2 SQRT Decomposition	16
1.2.1 Combinations	4		
1.2.2 Fast IO	4		
1.2.3 Permutations	4		
1.2.4 Random	4		
1.2.5 Sort List	4		
1.2.6 Sort List Of Object	5		
<b>2 Data Structures</b>	<b>5</b>		
2.1 Graphs	5		
2.1.1 UnionFind	5		
2.2 Ranges	5		
2.2.1 BIT	5		
2.2.2 Interval Tree	5		
2.2.3 Segment Tree	5		
2.2.4 Segment Tree Lazy Propagation	6		
2.2.5 Sparse Table	7		
2.3 Strings	7		
2.3.1 Trie	7		
2.4 Trees	8		
2.4.1 Treap	8		
<b>3 Graphs</b>	<b>8</b>		
3.1 Articulation Points And Bridges	8		
3.2 Connected Components	9		
3.3 Flood Fill	9		
3.4 Flow	9		
3.4.1 Max Flow Dinic	9		
3.4.2 Maximum Bipartite Matching	10		
3.5 Has Cycle	10		
3.6 Is Bipartite	10		
3.7 LCA	10		
3.8 MST Kruskal	11		
3.9 MST Prim	11		
3.10 ShortestPaths	12		
3.10.1 Bellman Ford	12		
3.10.2 Dijkstra	12		
3.11 Strongly Connected Components	12		
3.12 Topological Sort	13		

# Coding Resources

## C++

### 1.1.1 Decimal Precision

```
// rounds up the decimal number
cout << setprecision(N) << n << endl;
// specify N fixed number of decimals
cout << fixed << setprecision(N) << n << endl;
```

### 1.1.2 IO Optimization

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

### 1.1.3 Int To Binary

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int lli;

lli bitsInInt(lli n) { // clz = count leading zeroes
    return sizeof(n) * 8 - __builtin_clzll(n);
}
```

```
vector<bool> intToBitsArray(lli n) {
    n = abs(n);
    if (!n) return {};
    int length = bitsInInt(n), lastPos = length - 1;
    vector<bool> v(length);
    for (lli i = lastPos, j = 0; i > -1LL; i--, j++)
        ↪ v[j] = (n >> i) & 1LL;
    return v;
}
```

```
int main() {
    vector<bool> ans = intToBitsArray(8LL);
    for (auto a : ans) cout << a << " ";
    cout << endl;
    return 0;
}
```

### 1.1.4 Map Value To Int

```
// val = value
typedef string Val;
unordered_map<Val, int> intForVal;
unordered_map<int, Val> valForInt;
int mapId = 0;

int Map(Val val) {
    valForInt[mapId] = val;
    return intForVal.count(val) ? intForVal[val] :
        ↪ intForVal[val] = mapId++;
}

int IMap(int val) {
```

```
    return valForInt[val];
}
```

```
void initMapping() {
    mapId = 0;
    intForVal.clear();
    valForInt.clear();
}
```

### 1.1.5 Permutations

```
typedef vector<int> T; // typedef string T;
vector<T> permutations(T v) {
    vector<vector<int>> ans;
    sort(v.begin(), v.end());
    do
        ans.push_back(v);
    while (next_permutation(v.begin(), v.end()));
    return ans;
}
```

### 1.1.6 Print Vector

```
void printv(vector<int> v) {
    if (v.size() == 0) {
        cout << "[]" << endl;
        return;
    }
    cout << "[" << v[0];
    for (int i = 1; i < v.size(); i++) {
        cout << ", " << v[i];
    }
    cout << "]" << endl;
}
```

### 1.1.7 Priority Queue Of Object

```
struct Object {
    char first;
    int second;
};

int main() {
    auto cmp = [](const Object& a,
                  const Object& b) {
        return a.second > b.second;
    };
    priority_queue<Object, vector<Object>,
                  decltype(cmp)>
        pq(cmp);
    vector<Object> v = {
        {'c', 3}, {'a', 1}, {'b', 2}};
    sort(v.begin(), v.end(), cmp);
    return 0;
}
```

### 1.1.8 Random

```
int random(int min, int max) {
    return min + rand() % (max - min + 1);
}
```

```

}

int main() {
    srand(time(0));
    // code
}

```

### 1.1.9 Read Line

```

// when reading lines, don't mix 'cin' with
// 'getline' just use getline and split
string input() {
    string ans;
    // cin >> ws; // eats all whitespaces.
    getline(cin, ans);
    return ans;
}

```

### 1.1.10 Sort Pair

```

pair<int, int> p;
sort(p.begin(), p.end());
// sorts array on the basis of the first element

```

### 1.1.11 Sort Vector Of Object

```

struct Object {
    char first;
    int second;
};

bool cmp(const Object& a, const Object& b) {
    return a.second > b.second;
}

int main() {
    vector<Object> v = {
        {'c', 3}, {'a', 1}, {'b', 2}};
    sort(v.begin(), v.end(), cmp);
    printv(v);
    return 0;
}

```

### 1.1.12 Split String

```

vector<string> split(string str, char token) {
    stringstream test(str);
    string seg;
    vector<string> seglist;
    while (getline(test, seg, token))
        seglist.push_back(seg);
    return seglist;
}

```

### 1.1.13 Typedef

```

typedef TYPE ALIAS
// e.g.
typedef int T;

```

## 1.2 Python

### 1.2.1 Combinations

```

import itertools
# from arr choose k = > combinations(arr, k)
print(list(itertools.combinations([1, 2, 3], 3)))

```

### 1.2.2 Fast IO

```

from sys import stdin, stdout

N = 10
# Reads N chars from stdin (it counts '\n' as char)
stdin.read(N)
# Reads until '\n' or EOF
line = stdin.readline()
# Reads all lines in stdin until EOF
lines = stdin.readlines()
# Writes a string to stdout, it doesn't add '\n'
stdout.write(line)
# Writes a list of strings to stdout
stdout.writelines(lines)
# Reads numbers separated by space in a line
numbers = list(map(int, stdin.readline().split()))

```

### 1.2.3 Permutations

```

import itertools
print(list(itertools.permutations([1, 2, 3])))

```

### 1.2.4 Random

```

import random
# Initialize the random number generator.
random.seed(None)
# Returns a random integer N such that a <= N <= b.
random.randint(a, b)
# Returns a random integer N such that 0 <= N < b
random.randrange(b)
# Returns a random integer N such that a <= N < b.
random.randrange(a, b)
# Returns and integer with k random bits.
random.getrandbits(k)
# shuffles a list
random.shuffle(li)

```

### 1.2.5 Sort List

```

li = ['a', 'c', 'b']
# sorts inplace in descending order
li.sort(reverse=True)
# returns sorted list ascending order
ol = sorted(li)

```

### 1.2.6 Sort List Of Object

```
class MyObject :
    def __init__(self, first, second, third):
        self.first = first
        self.second = second
        self.third = third

li = [MyObject('b', 3, 1), MyObject('a', 3, 2),
      ↪ MyObject('b', 3, 3)]
# returns list sorted by first then by second then by
↪ third in increasing order
ol = sorted(li, key = lambda x: (x.first, x.second,
    ↪ x.third), reverse=False)
# sorts inplace by first then by second then by third
↪ in increasing order
li.sort(key = lambda x: (x.first, x.second, x.third),
    ↪ reverse=False)
```

## 2 Data Structures

### 2.1 Graphs

#### 2.1.1 UnionFind

```
struct UnionFind {
    vector<int> dad, size;
    int n;
    UnionFind(int N=0) : n(N), dad(N), size(N, 1) {
        while (N--> 0) dad[N] = N;
    }

    int root(int u) {
        if (dad[u] == u) return u;
        return dad[u] = root(dad[u]);
    }

    bool areConnected(int u, int v) {
        return root(u) == root(v);
    }

    void join(int u, int v) {
        int Ru = root(u), Rv = root(v);
        if (Ru == Rv) return;
        --n, dad[Ru] = Rv;
        size[Rv] += size[Ru];
    }

    int getSize(int u) {
        return size[root(u)];
    }

    int numberOfSets() {
        return n;
    }
};
```

### 2.2 Ranges

#### 2.2.1 BIT

#### 2.2.2 Interval Tree

#### 2.2.3 Segment Tree

```
// st = segment tree. st[1] = root;
// neutro = operation neutral value
// e.g. for sum is 0, for multiplication
// is 1, for gcd is 0, for min is INF, etc.
typedef int T;
T neutro = 0;
int N;
vector<T> st;

void initVars(int n) {
    st.assign(2 * (N = n), neutro);
}

T F(T a, T b) {
    return a + b;
    // return __gcd(a, b);
    // return a * b;
    // return min(a, b);
}

// O(2N)
void build() {
    for (int i = N - 1; i > 0; i--) st[i] = F(st[i <<
        ↪ 1], st[i << 1 | 1]);
}

// O(lg(2N))
void updateNode(int i, T val) {
    for (st[i += N] = val; i > 1; i >>= 1) st[i >> 1] =
        ↪ F(st[i], st[i ^ 1]);
}

// O(3N), [l, r]
void updateRange(int l, int r, T val) {
    for (l += N, r += N; l <= r; l++) st[l] = val;
    build();
}

// O(lg(2N)), [l, r]
T query(int l, int r) {
    T ans = neutro;
    for (l += N, r += N + 1; l < r; l >>= 1, r >>= 1) {
        if (l & 1) ans = F(ans, st[l++]);
        if (r & 1) ans = F(ans, st[--r]);
    }
    return ans;
}
```

```

void setValAt(T val, unsigned pos) {
    st[pos + N] = val;
}

T getValAt(unsigned pos) {
    return st[pos + N];
}

```

## 2.2.4 Segment Tree Lazy Propagation

```

// st = segment tree, st[1] = root, H = height;
// u = updates, d = delayed updates
// neutro = operation neutral val
// e.g. for sum is 0, for multiplication
// is 1, for gcd is 0, for min is INF, etc.
#include <bits/stdc++.h>
using namespace std;
typedef int T;
T neutro = 0;
int N, H;
vector<T> st, d;
vector<bool> u;

void initVars(int n) {
    u.assign(N = n, 0);
    d.assign(n, neutro);
    st.assign(2 * n, neutro);
    H = sizeof(int) * 8 - __builtin_clz(n);
}

T F(T a, T b) {
    return a + b;
    // return __gcd(a, b);
    // return a * b;
    // return min(a, b);
}

T kTimesF(T a, T k) {
    return a * k; // sum
    // return a // min, max, gcd
    // return pow(a, k); // multiplication
}

void calc(int p, int k) {
    if (u[p])
        st[p] = kTimesF(d[p], k);
    else
        st[p] = F(st[p << 1], st[p << 1 | 1]);
}

void apply(int p, int val, int k) {
    st[p] = kTimesF(val, k);
    if (p < N) d[p] = val, u[p] = 1;
}

void build() {
    for (int i = N - 1; i > 0; i--) st[i] = F(st[i << 1], st[i << 1 | 1]);
}

```

```

void push(int p) {
    for (int s = H, k = 1 << (H - 1); s > 0; s--, k >>= 1) {
        int i = p >> s;
        if (u[i]) {
            apply(i << 1, d[i], k);
            apply(i << 1 | 1, d[i], k);
            u[i] = 0;
        }
    }
}

// [l, r]
void modify(int l, int r, int val) {
    push(l += N);
    push(r += N);
    int cl = 0, cr = 0, k = 1;
    for (r++; l < r; l >>= 1, r >>= 1, k <<= 1) {
        if (cl) calc(l - 1, k);
        if (cr) calc(r, k);
        if (l & 1) apply(l++, val, k), cl = 1;
        if (r & 1) apply(--r, val, k), cr = 1;
    }
    for (l--; r > 0; l >>= 1, r >>= 1, k <<= 1) {
        if (cl) calc(l, k);
        if (cr && (!cl || l != r)) calc(r, k);
    }
}

// [l, r]
T query(int l, int r) {
    push(l += N);
    push(r += N);
    T ans = neutro;
    for (r++; l < r; l >>= 1, r >>= 1) {
        if (l & 1) ans = F(ans, st[l++]);
        if (r & 1) ans = F(ans, st[--r]);
    }
    return ans;
}

void setValAt(T val, unsigned pos) {
    st[pos + N] = val;
}

T getValAt(unsigned pos) {
    return st[pos + N];
}

int main() {
    initVars(9);
    for (int i = 1; i < 10; i++) setValAt(i, i - 1);
    build();
    modify(6, 6, 0);
    cout << query(7, 7) << endl;
    for (auto i : st) {
        cout << i << " ";
    }
}

```

```

    cout << endl;
    return 0;
}

```

### 2.2.5 Sparse Table

```

#include <bits/stdc++.h>

using namespace std;
// st = sparse table
typedef int T;

int MAXN = 100, N;
vector<vector<T>> st;
vector<T> arr;

void initVars() {
    st = vector<vector<T>>(MAXN, vector<T>(log2(MAXN) +
    ↪ 1));
    arr = vector<T>(MAXN);
}

static T F1(T a, T b) {
    // return min(a, b);
    return __gcd(a, b);
}

static T F2(T a, T b) {
    return a + b;
    // return a * b;
}

// O(N lg N)
void buildSparseTab(T F(T, T)) {
    st[0] = arr;
    for (int i = 1; (1 << i) <= N; i++)
        for (int j = 0; j + (1 << i) <= N; j++) st[i][j] =
            ↪ F(st[i - 1][j], st[i - 1][j + (1 << i) -
            ↪ 1));
}

// O(1)
T query(int L, int R) {
    int i = log2(R - L + 1);
    return F1(st[i][L], st[i][R + 1 - (1 << i)]);
}

// O(lg N)
T queryArith(int L, int R) {
    // Neutral Element
    T ans = 0; // for sum
    // T ans = 1; for multiplication
    while (true) {
        int k = log2(R - L + 1);
        ans = F2(ans, st[k][L]);
        L += 1 << k;
        if (L > R) break;
    }
    return ans;
}

```

```

}

int main() {
    initVars();
    N = 9;
    arr = {7, 2, 3, 0, 5, 10, 3, 12, 18};
    buildSparseTab(F1);

    cout << query(0, 2) << endl;
    cout << query(1, 3) << endl;
    cout << query(4, 5) << endl;

    initVars();
    N = 6;
    arr = {3, 7, 2, 5, 8, 9};
    buildSparseTab(F2);
    cout << queryArith(0, 5) << endl;
    cout << queryArith(3, 5) << endl;
    cout << queryArith(2, 4) << endl;
    return 0;
}

```

## 2.3 Strings

### 2.3.1 Trie

```

// wpt = number of words passing through
// w = number of words ending in the node
// c = character
struct Trie {
    struct Node {
        // for lexicographical order use 'map'
        // map<char, Node*> ch;
        unordered_map<char, Node*> ch;
        int w = 0, wpt = 0;
    };

    Node *root = new Node();

    // O(STR.SIZE)
    void insert(string str) {
        Node *curr = root;
        for (auto &c : str) {
            curr->wpt++;
            if (!curr->ch.count(c)) curr->ch[c] = new
            ↪ Node();
            curr = curr->ch[c];
        }
        curr->wpt++;
        curr->w++;
    }

    Node *find(string &str) {
        Node *curr = root;
        for (auto &c : str) {
            if (!curr->ch.count(c)) return nullptr;
            curr = curr->ch[c];
        }
        return curr;
    }
}

```

```

// number of words with given prefix O(N)
int prefixCount(string prefix) {
    Node *node = find(prefix);
    return node ? node->wpt : 0;
}

// number of words matching str O(N)
int strCount(string str) {
    Node *node = find(str);
    return node ? node->w : 0;
}

void getWords(Node *curr, vector<string> &words,
    ↪ string &word) {
    if (!curr) return;
    if (curr->w) words.push_back(word);
    for (auto &c : curr->ch) {
        getWords(c.second, words, word += c.first);
        word.pop_back();
    }
}

// O(N)
vector<string> getWords() {
    vector<string> words;
    string word = "";
    getWords(root, words, word);
    return words;
}

// O(N)
vector<string> getWordsByPrefix(string prefix) {
    vector<string> words;
    getWords(find(prefix), words, prefix);
}

bool remove(Node *curr, string &str, int &i) {
    if (i == str.size()) {
        curr->wpt--;
        return curr->w ? !(curr->w = 0) : 0;
    }
    int c = str[i];
    if (!curr->ch.count(c)) return false;
    if (remove(curr->ch[c], str, ++i)) {
        if (!curr->ch[c]->wpt) curr->wpt--,
            ↪ curr->ch.erase(c);
        return true;
    }
    return false;
}

// O(STR.SIZE)
int remove(string str) {
    int i = 0;
    return remove(root, str, i);
}
};

```

## 2.4 Trees

### 2.4.1 Treap

## 3 Graphs

### 3.1 Articulation Points And Bridges

```

// APB = articulation points and bridges
// ap = Articulation Point
// br = bridges
// p = parent
// disc = discovery time
// low = lowTime
// ch = children

```

```

typedef pair<int, int> Edge;
int MAXN = 101, Time;
vector<vector<int>> ady;
vector<int> disc, low, ap;
vector<Edge> br;

void initVars(int N) {
    ady.assign(N, vector<int>());
}

int dfsAPB(int u, int p) {
    int ch = 0;
    low[u] = disc[u] = ++Time;
    for (int &v : ady[u]) {
        if (v == p) continue;
        if (!disc[v]) {
            ch++;
            dfsAPB(v, u);
            if (disc[u] <= low[v]) ap[u]++;
            if (disc[u] < low[v]) br.push_back({u, v});
            low[u] = min(low[u], low[v]);
        } else
            low[u] = min(low[u], disc[v]);
    }
    return ch;
}

// O(N)
void APB() {
    br.clear();
    ap = low = disc = vector<int>(ady.size());
    Time = 0;
    for (int u = 0; u < N; u++)
        if (!disc[u]) ap[u] = dfsAPB(u, u) > 1;
}

void addEdge(int u, int v) {
    ady[u].push_back(v);
    ady[v].push_back(u);
}

```



## 3.2 Connected Components

```
// comp = component
int MAXN = 26, N, compId = 0;
vector<vector<int>> ady;
vector<int> getComp;

void initVars(int N) {
    ady.assign(N, vector<int>());
    getComp.assign(N, -1);
    compId = 0;
}

void dfsCC(int u, vector<int> &comp) {
    if (getComp[u] > -1) return;
    getComp[u] = compId;
    comp.push_back(u);
    for (auto &v : ady[u]) dfsCC(v, comp);
}

// O(N)
vector<vector<int>> connectedComponents() {
    vector<vector<int>> comps;
    for (int u = 0; u < ady.size(); u++) {
        vector<int> comp;
        dfsCC(u, comp);
        if (!comp.empty()) comps.push_back(comp),
        ↪ compId++;
    }
    return comps;
}

void addEdge(int u, int v) {
    ady[u].push_back(v);
    ady[v].push_back(u);
}
```

## 3.3 Flood Fill

```
int n, m, oldColor = 0, color = 1;
vector<vector<int>> mat;

vector<vector<int>> movs = {
    {1, 0}, {0, 1}, {-1, 0}, {0, -1}};

void floodFill(int i, int j) {
    if (i >= mat.size() || i < 0 ||
        j >= mat[i].size() || j < 0 ||
        mat[i][j] != oldColor)
        return;
    mat[i][j] = color;
    for (auto move : movs)
        floodFill(i + move[1], j + move[0]);
}

void floodFill() {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (mat[i][j] == oldColor) floodFill(i, j);
}
```

```
}
```

## 3.4 Flow

### 3.4.1 Max Flow Dinic

```
// cap[a][b] = Capacity from a to b
// flow[a][b] = flow occupied from a to b
// level[a] = level in graph of node a
// Num = number
typedef int Num;
int N, MAXN = 101;
vector<int> level;
vector<vector<int>> ady;
unordered_map<int, unordered_map<int, Num>> cap, flow;

void initVars(int N) {
    ady.assign(N, vector<int>());
    cap.clear();
    flow.clear();
}

bool levelGraph(int s, int t) {
    level = vector<int>(ady.size());
    level[s] = 1;
    queue<int> q;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int &v : ady[u]) {
            if (!level[v] && flow[u][v] < cap[u][v]) {
                q.push(v);
                level[v] = level[u] + 1;
            }
        }
    }
    return level[t];
}

Num blockingFlow(int u, int t, Num currPathMaxFlow) {
    if (u == t) return currPathMaxFlow;
    for (int v : ady[u]) {
        Num capleft = cap[u][v] - flow[u][v];
        if ((level[v] == (level[u] + 1)) && (capleft > 0))
            ↪ {
                Num pathMaxFlow = blockingFlow(v, t,
                ↪ min(currPathMaxFlow, capleft));
                if (pathMaxFlow > 0) {
                    flow[u][v] += pathMaxFlow;
                    flow[v][u] -= pathMaxFlow;
                    return pathMaxFlow;
                }
            }
    }
    return 0;
}

// O(E * V^2)
```

```

Num dinicMaxFlow(int s, int t) {
    if (s == t) return -1;
    Num maxFlow = 0;
    while (levelGraph(s, t))
        while (Num flow = blockingFlow(s, t, 1 << 30))
            ↪ maxFlow += flow;
    return maxFlow;
}

void addEdge(int u, int v, Num capacity) {
    cap[u][v] = capacity;
    ady[u].push_back(v);
}

```

### 3.4.2 Maximum Bipartite Matching

```

#include "Max Flow Dinic.cpp"
void addEdge(int u, int v) {
    cap[u][v] = 1;
    ady[u].push_back(v);
}

int main() {
    int n, s = 0, t = 1;
    cin >> n;
    initVars(n);
    while (n--) {
        int u, v;
        cin >> u >> v;
        addEdge(u += 2, v += 2);
        addEdge(s, u);
        addEdge(v, t);
    }
    cout << dinicMaxFlow(s, t) << endl;
    return 0;
}

```

### 3.5 Has Cycle

```

vector<vector<int>> ady, cycles;
vector<int> vis, cycle;
bool flag = false, isDirected = false;
int root = -1;

void initVars(int N) {
    ady.assign(N, vector<int>());
    vis.assign(N, 0);
    cycles.clear();
    root = -1, flag = false;
}

bool hasCycle(int u, int prev) {
    vis[u] = 1;
    for (auto &v : ady[u]) {
        if (v == u || vis[v] == 2 || (!isDirected && v ==
            ↪ prev)) continue;
        if (flag) {
            if (!vis[v]) hasCycle(v, u);
            continue;
        }
    }
}

```

```

    }
    if (vis[v] || hasCycle(v, u)) {
        if (root == -1) root = v, flag = true;
        cycle.push_back(u);
        if (root == u) flag = false, root = -1,
            ↪ cycles.push_back(cycle), cycle.clear();
    }
}
vis[u] = 2;
return flag;
}

// O(N)
bool hasCycle() {
    for (int u = 0; u < ady.size(); u++)
        if (!vis[u]) cycle.clear(), hasCycle(u, -1);
    return cycles.size() > 0;
}

void addEdge(int u, int v) {
    ady[u].push_back(v);
    if (!isDirected) ady[v].push_back(u);
}

```

### 3.6 Is Bipartite

```

vector<vector<int>> ady;

void initVars(int N) {
    ady.assign(N, vector<int>());
}

// O(N)
bool isBipartite() {
    vector<int> color(ady.size(), -1);
    for (int s = 0; s < ady.size(); s++) {
        if (color[s] > -1) continue;
        color[s] = 0;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int &v : ady[u]) {
                if (color[v] < 0) q.push(v), color[v] =
                    ↪ !color[u];
                if (color[v] == color[u]) return false;
            }
        }
    }
    return true;
}

```

### 3.7 LCA

### 3.8 MST Kruskal

```
// N = number of nodes
#include "../Data Structures/Graphs/UnionFind.cpp"
typedef int Num;
typedef pair<int, int> Edge;
typedef pair<Num, Edge> Wedge;
vector<Wedge> Wedges; // gets filled from input;
vector<Wedge> mst;
UnionFind uf(0);

void initVars(int N) {
    mst.clear();
    Wedges.clear();
    uf = UnionFind(N);
}

Num kruskal() {
    Num cost = 0;
    sort(Wedges.begin(), Wedges.end());
    // reverse(Wedges.begin(), Wedges.end());
    for (Wedge &wedge : Wedges) {
        int u = wedge.second.first, v =
            ↪ wedge.second.second;
        if (!uf.areConnected(u, v)) uf.join(u, v),
            ↪ mst.push_back(wedge), cost += wedge.first;
    }
    return cost;
}

void addEdge(int u, int v, Num w) {
    Wedges.push_back({w, {u, v}});
}
```

### 3.9 MST Prim

```
// st = spanning tree, p = parent
// vis = visited, dist = distance
typedef int Num;
typedef pair<int, int> Edge;
typedef pair<Num, Edge> Wedge;
typedef pair<Num, int> DistNode;
int MAXN = 20001, INF = 1 << 30;
vector<vector<int>>> ady;
unordered_map<int, unordered_map<int, Num>>> weight;
vector<int> p, vis;
vector<Num> dist;
vector<vector<Wedge>>> msts;

void initVars(int N) {
    ady.assign(N, vector<int>());
    p.assign(N, 0);
    vis.assign(N, 0);
    dist.assign(N, INF);
    weight.clear();
    msts.clear();
}

Num prim(int s) {
```

```
    vector<Wedge> mst;
    vector<set<Edge>::iterator> pos(ady.size());
    vector<Num> dist(ady.size(), INF);
    set<Edge> q;
    Num cost = dist[s] = 0;
    q.insert({0, s});
    while (q.size()) {
        int u = q.begin()->second;
        q.erase(q.begin());
        vis[u] = 1, cost += dist[u];
        mst.push_back({dist[u], {p[u], u}});
        for (int &v : ady[u]) {
            Num w = weight[u][v];
            if (!vis[v] && w < dist[v]) {
                if (dist[v] != INF) q.erase(pos[v]);
                pos[v] = q.insert({dist[v] = w, v}).first;
            }
        }
    }
    msts.push_back(vector<Wedge>(mst.begin() + 1,
        ↪ mst.end()));
    return cost;
}

Num primLazy(int s) {
    vector<Wedge> mst;
    vector<set<Edge>::iterator> pos(ady.size());
    vector<Num> dist(ady.size(), INF);
    priority_queue<DistNode, vector<DistNode>,
        ↪ greater<DistNode>>> q;
    Num cost = dist[s] = 0;
    q.push({0, s});
    while (q.size()) {
        pair<int, int> aux = q.top();
        int u = aux.second;
        q.pop();
        if (dist[u] < aux.first) continue;
        vis[u] = 1, cost += dist[u];
        mst.push_back({dist[u], {p[u], u}});
        for (int &v : ady[u]) {
            Num w = weight[u][v];
            if (!vis[v] && w < dist[v]) q.push({dist[v] = w,
                ↪ v});
        }
    }
    msts.push_back(vector<Wedge>(mst.begin() + 1,
        ↪ mst.end()));
    return cost;
}

// O(V + E * log(V))
Num prim() {
    Num cost = 0;
    map<int, Num> q;
    for (int i = 0; i < ady.size(); i++)
        if (!vis[i]) cost += prim(i);
    return cost;
}
```

```
void addEdge(int u, int v, Num w) {
    ady[u].push_back(v);
    weight[u][v] = w;
    ady[v].push_back(u);
    weight[v][u] = w;
}
```

### 3.10 ShortestPaths

#### 3.10.1 Bellman Ford

*// N = number of nodes*  
*// returns {} if there is a negative weight cycle*

```
typedef int Num;
int MAXN = 20001, N, INF = 1 << 30, isDirected = true;
vector<vector<int>>> ady;
unordered_map<int, unordered_map<int, Num>>> weight;
```

```
void initVars(int N) {
    ady.assign(N, vector<int>());
    weight.clear();
}
```

*// O(V \* E)*

```
vector<Num> bellmanFord(int s) {
    vector<Num> dist(ady.size(), INF);
    dist[s] = 0;
    for (int i = 1; i <= ady.size(); i++)
        for (int u = 0; u < ady.size(); u++)
            for (auto &v : ady[u]) {
                Num w = weight[u][v];
                if (dist[u] != INF && dist[u] + w < dist[v]) {
                    if (i == ady.size()) return {};
                    dist[v] = dist[u] + w;
                }
            }
    return dist;
}
```

```
void addEdge(int u, int v, Num w) {
    ady[u].push_back(v);
    weight[u][v] = w;
    if (isDirected) return;
    ady[v].push_back(u);
    weight[v][u] = w;
}
```

#### 3.10.2 Dijkstra

```
typedef int Num;
typedef pair<Num, int> DistNode;
int MAXN = 20001, INF = 1 << 30, isDirected = false;
vector<vector<int>>> ady;
unordered_map<int, unordered_map<int, Num>>> weight;
```

```
void initVars(int N) {
    ady.assign(N, vector<int>());
    weight.clear();
}
```

*// O(E \* log(V))*

```
vector<int> dijkstra(int s) {
    vector<set<DistNode>::iterator> pos(ady.size());
    vector<Num> dist(ady.size(), INF);
    set<DistNode> q;
    q.insert({0, s}), dist[s] = 0;
    while (q.size()) {
        int u = q.begin()->second;
        q.erase(q.begin());
        for (int &v : ady[u]) {
            Num w = weight[u][v];
            if (dist[u] + w < dist[v]) {
                if (dist[v] != INF) q.erase(pos[v]);
                pos[v] = q.insert({dist[v] = dist[u] + w,
                    ↪ v}).first;
            }
        }
    }
    return dist;
}
```

```
vector<int> dijkstraLazy(int s) {
    vector<int> dist(ady.size(), INF);
    priority_queue<DistNode, vector<DistNode>,
        ↪ greater<DistNode>>> q;
    q.push({0, s}), dist[s] = 0;
    while (q.size()) {
        DistNode top = q.top(); q.pop();
        int u = top.second;
        if (dist[u] < top.first) continue;
        for (int &v : ady[u]) {
            Num w = weight[u][v];
            if (dist[u] + w < dist[v]) q.push({dist[v] =
                ↪ dist[u] + w, v});
        }
    }
    return dist;
}
```

```
void addEdge(int u, int v, Num w) {
    ady[u].push_back(v);
    weight[u][v] = w;
    if (isDirected) return;
    ady[v].push_back(u);
    weight[v][u] = w;
}
```

### 3.11 Strongly Connected Components

*// tv = top value from stack*  
*// sccs = strongly connected components*  
*// scc = strongly connected component*  
*// disc = discovery time*  
*// low = low time*  
*// s = stack*  
*// top = top index of the stack*  
 int MAXN = 101, Time, top;

```

vector<vector<int>> ady, sccs;
vector<int> disc, low, s;

void initVars(int N) {
    ady.assign(N, vector<int>());
}

void dfsSCCS(int u) {
    if (disc[u]) return;
    low[u] = disc[u] = ++Time;
    s[++top] = u;
    for (int &v : ady[u]) dfsSCCS(v), low[u] =
        ↪ min(low[u], low[v]);
    if (disc[u] == low[u]) {
        vector<int> scc;
        while (true) {
            int tv = s[top--];
            scc.push_back(tv);
            low[tv] = ady.size();
            if (tv == u) break;
        }
        sccs.push_back(scc);
    }
}

// O(N)
void SCCS() {
    s = low = disc = vector<int>(ady.size());
    Time = 0, top = -1, sccs.clear();
    for (int u = 0; u < ady.size(); u++) dfsSCCS(u);
}

void addEdge(int u, int v) {
    ady[u].push_back(v);
}

```

### 3.12 Topological Sort

```

// vis = visited
vector<vector<int>> ady;
vector<int> vis, toposorted;

void initVars(int N) {
    ady.assign(N, vector<int>());
    vis.assign(N, 0);
    toposorted.clear();
}

// returns false if there is a cycle
bool toposort(int u) {
    vis[u] = 1;
    for (auto &v : ady[u]) {
        if (v == u || vis[v] == 2) continue;
        if (vis[v] == 1 || !toposort(v)) return false;
    }
    vis[u] = 2;
    toposorted.push_back(u);
    return true;
}

```

```

// O(N)
bool toposort() {
    vis.clear();
    for (int u = 0; u < ady.size(); u++)
        if (!vis[u])
            if (!toposort(u)) return false;
    return true;
}

```

## 4 Maths

### 4.1 Number Theory

#### 4.1.1 Divisibility Criterion

```

def divisorCriteria(n, lim):
    results = []
    tenElevated = 1
    for i in range(lim):
        # remainder = pow(10, i, n)
        remainder = tenElevated % n
        negremainder = remainder - n
        if (remainder <= abs(negremainder)):
            results.append(remainder)
        else:
            results.append(negremainder)
        tenElevated *= 10
    return results

```

```

def testDivisibility(dividend, divisor,
    ↪ divisor_criteria):
    dividend = str(dividend)
    addition = 0
    dividendSize = len(dividend)
    i = dividendSize - 1
    j = 0
    while j < dividendSize:
        addition += int(dividend[i]) *
            ↪ divisor_criteria[j]
        i -= 1
        j += 1
    return addition % divisor == 0

```

```

if __name__ == '__main__':
    dividend, divisor = map(int, input().split())
    divisor_criteria = divisorCriteria(divisor,
        ↪ len(str(dividend)))
    print(divisor_criteria)
    print(testDivisibility(dividend, divisor,
        ↪ divisor_criteria))

```

#### 4.1.2 Extended Euclidean

```

// gcd(a, b) = ax + by
vector<long long int> extendedGCD(
    long long int a, long long int b) {

```

```

if (a > 0LL && b == 0LL) {
    return {a, 1LL, 0LL};
}
long long int x = 1LL, y = 0LL, prevx = 0LL,
               prevy = 1LL, q, remainder;
while (true) {
    q = a / b;
    remainder = a - b * q;
    if (remainder == 0LL) break;
    a = b;
    b = remainder;
    x = x - prevx * q;
    swap(x, prevx);
    y = y - prevy * q;
    swap(y, prevy);
}
// gcd = b, x = prevx, y = prevy
return {b, prevx, prevy};
}

```

#### 4.1.3 GCD

```

int gcd(int a, int b) {
    return !b ? a : gcd(b, a % b);
}

int gcdI(int a, int b) {
    while (b) {
        a %= b;
        swap(a, b);
    }
    return a;
}

```

#### 4.1.4 LCM

```

int lcm(int a, int b) {
    int c = gcd(a, b);
    return c ? a / c * b : 0;
}

```

#### 4.1.5 Prime Check Miller Rabin

```

from random import randrange

```

```

def is_prime(p):
    k = 100
    if p == 2 or p == 3:
        return True
    if (p & 1) == 0 or p == 1:
        return False
    phi = p - 1
    d = phi
    r = 0
    while (d & 1) == 0:
        d = int(d >> 1)
        r += 1
    for i in range(k):

```

```

        a = randrange(2, p - 2)
        exp = pow(a, d, p)
        if exp == 1 or exp == p - 1:
            continue
        flag = False
        for j in range(r - 1):
            exp = pow(exp, 2, p)
            if exp == 1:
                return False
            if exp == p - 1:
                flag = True
                break
        if flag:
            continue
        else:
            return False
    return True

```

#### 4.1.6 Prime Sieve

```

vector<int> primeSieve(int n) {
    vector<int> sieve(n + 1);
    for (int i = 4; i <= n; i += 2) sieve[i] = 2;
    for (int i = 3; i * i <= n; i += 2)
        if (!sieve[i])
            for (int j = i * i; j <= n; j += 2 * i)
                if (!sieve[j]) sieve[j] = i;
    return sieve;
}

```

## 5 Strings

### 5.1 KMP

```

// f = error function
// cf = create error function
// p = pattern
// t = text
// pos = positions where pattern is found in text

```

```

int MAXN = 1000000;
vector<int> f(MAXN + 1);

vector<int> kmp(string &p, string &t, int cf) {
    vector<int> pos;
    if (cf) f[0] = -1;
    for (int i = cf, j = 0; j < t.size(); i = f[i], j++)
        while (i > -1 && p[i] != t[j]) i = f[i];
        if (cf) f[j] = i;
        if (!cf && i == p.size()) pos.push_back(j - i), i = f[i];
    }
    return pos;
}

vector<int> search(string &p, string &t) {
    kmp(p, p, -1); // create error function
}

```

```
    return kmp(p, t, 0); // search in text
}
```

## 5.2 Rabin Karp

```
class RollingHash {
public:
    vector<unsigned long long int> pow;
    vector<unsigned long long int> hash;
    unsigned long long int B;
    RollingHash(const string &text) : B(257) {
        int N = text.size();
        pow.resize(N + 1);
        hash.resize(N + 1);
        pow[0] = 1;
        hash[0] = 0;
        for (int i = 1; i <= N; ++i) {
            // in c++ an unsigned long long int is
            // automatically modulated by 2^64
            pow[i] = pow[i - 1] * B;
            hash[i] = hash[i - 1] * B + text[i - 1];
        }
    }

    unsigned long long int getWordHash() {
        return hash[hash.size() - 1];
    }

    unsigned long long int getSubstrHash(int begin, int
    ↪ end) {
        return hash[end] - hash[begin - 1] * pow[end -
    ↪ begin + 1];
    }

    int size() {
        return hash.size();
    }
};

vector<int> rabinKarp(RollingHash &rhStr, string
    ↪ &pattern) {
    vector<int> positions;
    RollingHash rhPattern(pattern);
    unsigned long long int patternHash =
    ↪ rhPattern.getWordHash();
    int windowSize = pattern.size(), end = windowSize;
    for (int i = 1; end < rhStr.size(); i++) {
        if (patternHash == rhStr.getSubstrHash(i, end))
            ↪ positions.push_back(i);
        end = i + windowSize;
    }
    return positions;
}
```

## 6 Techniques

### 6.1 Binary Search

## 6.2 Multiple Queries

### 6.2.1 Mo

```
#include <bits/stdc++.h>

using namespace std;

// q = query
// qs = queries

struct Query {
    int l, r;
};

int N, M, blksize;
vector<Query> qs;
vector<int> arr;

void initVars() {
    qs = vector<Query>(M);
    arr = vector<int>(N);
}

bool cmp(Query &a, Query &b) {
    if (a.l == b.l) return a.r < b.r;
    return a.l / blksize < b.l / blksize;
}

void getResult() {
    blksize = (int)sqrt(N);
    sort(qs.begin(), qs.end(), cmp);
    int prevL = 0, prevR = -1;
    int sum = 0;
    for (auto &q : qs) {
        int L = q.l, R = q.r;
        while (prevL < L) {
            sum -= arr[prevL]; // problem specific
            prevL++;
        }
        while (prevL > L) {
            prevL--;
            sum += arr[prevL]; // problem specific
        }
        while (prevR < R) {
            prevR++;
            sum += arr[prevR]; // problem specific
        }
        while (prevR > R) {
            sum -= arr[prevR]; // problem specific
            prevR--;
        }

        cout << "sum[" << L << ", " << R
            << "] = " << sum << endl;
    }
}
```

```
int main() {
    arr = {1, 1, 2, 1, 3, 4, 5, 2, 8};
    N = arr.size();
    qs = {{0, 8}, {3, 5}};
    M = qs.size();
    getResult();
}
```

### 6.2.2 SQRT Decomposition

```
// sum of elements in range
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
int N, blksize;
int MAXN = 100, MAXSQR = (int)sqrt(MAXN);
```

```
vector<int> arr(MAXN);
vector<int> blks(MAXSQR + 1);
```

```
void preprocess() {
    blksize = sqrt(N);
    for (int i = 0, j = 0; i < N; i++) {
        if (i == blksize * j) j++;
        blks[j - 1] += arr[i]; // problem specific
    }
}
```

```
// problem specific
```

```
void update(int i, int val) {
    blks[i / blksize] += val - arr[i];
    arr[i] = val;
}
```

```
int query(int l, int r) {
    int sum = 0;
    int lblk = l / blksize;
    if (l != blksize * lblk++) {
        while (l < r && l != lblk * blksize) {
            sum += arr[l]; // problem specific
            l++;
        }

        while (l + blksize <= r) {
            sum += blks[l / blksize]; // problem specific
            l += blksize;
        }

        while (l <= r) {
            sum += arr[l]; // problem specific
            l++;
        }
        return sum;
    }
}
```

```
int main() {
    N = 10;
    arr = {1, 5, 2, 4, 6, 1, 3, 5, 7, 10};
    preprocess();
```

```
    for (int i = 0; i < blksize + 1; i++)
        cout << blks[i] << " ";
    // 8 11 15 10
    cout << endl;
    cout << query(3, 8) << " ";
    cout << query(1, 6) << " ";
    update(8, 0);
    cout << query(8, 8) << endl;
    // 26 21 0
    return 0;
}
```