Competitive Programming Reference

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

John Johnson

By

Sergio Gabriel Sanchez Valencia

gabrielsanv97@gmail.com

searleser97

Contents

Coding Resources

C++

Decimal Precision

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

Include All Libraries

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

Int To Binary

```
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;
}
```

IO Optimization

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

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();
}
```

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;
}
```

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;
}
```

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;
}
```

Random

```cpp
int random(int min, int max) {
  return min + rand() % (max - min + 1);
}
int main() {
  srand(time(0));
  // code ...
}
```

Read Line

```cpp
// 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;
}
```

Sort Pair

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

Sort Vector Of Object

```cpp
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;
}
```

Split String

```cpp
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;
}
```

Typedef

```cpp
typedef TYPE ALIAS;
// example:
typedef int T;
```

Python

Combinations

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

Fast IO

```python
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()))
```

Permutations

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

Random

```python
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)
```

Sort List

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

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)
```

Data Structures

Geometry

K-D Tree

Graphs

UnionFind

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

Ranges

BIT

```
typedef long long int T;
T neutro = 0;
vector<T> bit;
void initVars(int n) {
  bit.assign(++n, neutro);
}

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

// Inverse of F
T I(T a, T b) {
  return a - b;
  // return a / b;
}

// O(N)
void build() {
  for (int i = 1; i < bit.size(); i++) {
    int j = i + (i & -i);
    if (j < bit.size()) bit[j] = F(bit[j], bit[i]);
  }
}

// O(lg(N))
void update(int i, T val) {
  for (i++; i < bit.size(); i += i & -i) bit[i] =
  ↪  F(bit[i], val);
}

// O(lg(N))
T query(int i) {
  T ans = neutro;
  for (i++; i; i -= i & -i) ans = F(ans, bit[i]);
  return ans;
}
```

```
// O(lg(N)), [l, r]
T query(int l, int r) {
  return I(query(r), query(--l));
}
void setValAt(T val, int i) {
  bit[++i] = val;
}
```

BIT Range Update

```
typedef long long int T;
T neutro = 0;
vector<T> bit1, bit2;

void initVars(int n) {
  bit1.assign(++n, neutro);
  bit2 = bit1;
}

// O(lg(N))
void update(vector<T> &bit, int i, T val) {
  for (i++; i < bit.size(); i += i & -i) bit[i] +=
   ↪  val;
}

// O(lg(N)), [l, r]
void update(int l, int r, T val) {
  update(bit1, l, val);
  update(bit1, r + 1, -val);
  update(bit2, r + 1, val * r);
  update(bit2, l, -val * (l - 1));
}

// O(lg(N))
T query(vector<T> &bit, int i) {
  T ans = neutro;
  for (i++; i; i -= i & -i) ans += bit[i];
  return ans;
}

// O(lg(N))
T query(int i) {
  return query(bit1, i) * i + query(bit2, i);
}

// O(lg(N)), [l, r]
T query(int l, int r) {
  return query(r) - query(l - 1);
}
```

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 update(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 update(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; 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, int i) {
  st[i + N] = val;
}
```

Segment Tree Lazy Propagation

```
// st = segment tree, st[1] = root, H = height of d
// 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.

template <class T>
struct SegmentTree {
  T neutro = 0;
  int N, H;
  vector<T> st, d;
  vector<bool> u;

  SegmentTree(int n) : st(2 * n, neutro), d(n), u(n,
   ↪  0) {
    H = sizeof(int) * 8 - __builtin_clz(N = n);
  }
```

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

void apply(int i, T val, int k) {
  st[i] = val * k;  // sum
  // st[i] = val; // min, max, gcd
  // st[i] = pow(a, k); // multiplication
  if (i < N) d[i] = val, u[i] = 1;
}

void calc(int i) {
  if (!u[i]) st[i] = F(st[i << 1], st[i << 1 | 1]);
}

// O(2N)
void build() {
  for (int i = N - 1; i > 0; i--) calc(i);
}

// O(lg(N))
void build(int p) {
  while (p > 1) p >>= 1, calc(p);
}

// O(lg(N))
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, d[i] = neutro;
    }
  }
}

// O(lg(N)), [l, r]
void update(int l, int r, T val) {
  push(l += N);
  push(r += N);
  int ll = l, rr = r, k = 1;
  for (; l <= r; l >>= 1, r >>= 1, k <<= 1) {
    if (l & 1) apply(l++, val, k);
    if (~r & 1) apply(r--, val, k);
  }
  build(ll);
  build(rr);
}
```

```
  // O(lg(2N)), [l, r]
  T query(int l, int r) {
    push(l += N);
    push(r += N);
    T ans = neutro;
    for (; 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, int i) {
    st[i + N] = val;
  }
};
```

Sparse Table

```
// st = sparse table, Arith = Arithmetic
typedef int T;
int neutro = 0;
vector<vector<T>> st;

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

// O(Nlg(N))
void build(vector<T> &arr) {
  st.assign(log2(arr.size()), vector<T>(arr.size()));
  st[0] = arr;
  for (int i = 1; (1 << i) <= arr.size(); i++)
    for (int j = 0; j + (1 << i) <= arr.size(); j++)
  ↪   st[i][j] = F(st[i - 1][j], st[i - 1][j + (1 <<
  ↪   (i - 1))]);
}

// O(1), [l, r]
T query(int l, int r) {
  int i = log2(r - l + 1);
  return F(st[i][l], st[i][r + 1 - (1 << i)]);
}

// O(lg(N)), [l, r]
T queryArith(int l, int r) {
  T ans = neutro;
  while (true) {
    int k = log2(r - l + 1);
    ans = F(ans, st[k][l]);
    l += 1 << k;
    if (l > r) break;
  }
  return ans;
}
```

Strings

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++;
  }

  // O(STR.SIZE)
  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;
  }

  // O(N)
  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);
  }

  // O(STR.SIZE)
  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);
  }
};
```

Trees And Heaps

Treap

Geometry

Max Interval Overlap

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

Graphs

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 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 < ady.size(); 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);
}
```

Connected Components

```
// comp = component
int compId;
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);
}
```

Cycles

Get All Simple Cycles

Get Some Cycles

```
// at least detects one cycle per component
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;
}
```

```cpp
// O(N)
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);
}
```

Has Cycle

```cpp
vector<vector<int>> ady;
vector<int> vis;
bool isDirected = false;

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

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

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

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

Flood Fill

```cpp
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);
}
```

Flow

Max Flow Dinic

```cpp
// 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

typedef int T;
vector<int> level;
vector<vector<int>> ady;
unordered_map<int, unordered_map<int, T>> 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];
}
```

```
T blockingFlow(int u, int t, T currPathMaxFlow) {
  if (u == t) return currPathMaxFlow;
  for (int v : ady[u]) {
    T capleft = cap[u][v] - flow[u][v];
    if ((level[v] == (level[u] + 1)) && (capleft > 0))
    ↪ {
      T 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)
T dinicMaxFlow(int s, int t) {
  if (s == t) return -1;
  T maxFlow = 0;
  while (levelGraph(s, t))
    while (T flow = blockingFlow(s, t, 1 << 30))
    ↪ maxFlow += flow;
  return maxFlow;
}

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

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;
}
```

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;
}
```

LCA

```
// st = sparse table
typedef pair<int, int> T;
int neutro = 0;
vector<vector<T>> st;
vector<int> first;
vector<T> tour;
vector<vector<int>> ady;

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

T F(T a, T b) {
  return a.first < b.first ? a : b;
}

void build() {
  st.assign(log2(tour.size()),
  ↪ vector<T>(tour.size()));
  st[0] = tour;
  for (int i = 1; (1 << i) <= tour.size(); i++)
    for (int j = 0; j + (1 << i) <= tour.size(); j++)
    ↪ st[i][j] = F(st[i - 1][j], st[i - 1][j + (1 <<
    ↪ (i - 1))]);
}
```

```cpp
void eulerTour(int u, int p, int h) {
  first[u] = tour.size();
  tour.push_back({h, u});
  for (int v : ady[u])
    if (v != p) {
      eulerTour(v, u, h + 1);
      tour.push_back({h, u});
    }
}
// O(N * lg(N))
void preprocess() {
  tour.clear();
  first.assign(ady.size(), -1);
  eulerTour(0, 0, 0);
  build();
}
// O(1)
int lca(int u, int v) {
  int l = min(first[u], first[v]);
  int r = max(first[u], first[v]);
  int i = log2(r - l + 1);
  return F(st[i][l], st[i][r + 1 - (1 << i)]).second;
}

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

MST Kruskal

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

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

T kruskal() {
  T 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;
}
```

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

MST Prim

```cpp
// st = spanning tree, p = parent
// vis = visited, dist = distance
typedef int T;
typedef pair<int, int> Edge;
typedef pair<T, Edge> Wedge;
typedef pair<T, int> DistNode;
int INF = 1 << 30;
vector<vector<int>> ady;
unordered_map<int, unordered_map<int, T>> weight;
vector<int> p, vis;
vector<T> 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();
}

// O(E * log(V))
T prim(int s) {
  vector<Wedge> mst;
  vector<set<Edge>::iterator> pos(ady.size());
  vector<T> dist(ady.size(), INF);
  set<Edge> q;
  T 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]) {
      T 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;
}
```

```
T primLazy(int s) {
  vector<Wedge> mst;
  vector<set<Edge>::iterator> pos(ady.size());
  vector<T> dist(ady.size(), INF);
  priority_queue<DistNode, vector<DistNode>,
  ↪  greater<DistNode>> q;
  T 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]) {
      T 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))
T prim() {
  T cost = 0;
  map<int, T> q;
  for (int i = 0; i < ady.size(); i++)
    if (!vis[i]) cost += prim(i);
  return cost;
}

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

ShortestPaths

Bellman Ford

```
// N = number of nodes
// returns {} if there is a negative weight cycle
typedef int T;
int MAXN = 20001, N, INF = 1 << 30, isDirected = true;
vector<vector<int>> ady;
unordered_map<int, unordered_map<int, T>> weight;

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

// O(V * E)
vector<T> bellmanFord(int s) {
  vector<T> 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]) {
        T 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, T w) {
  ady[u].push_back(v);
  weight[u][v] = w;
  if (isDirected) return;
  ady[v].push_back(u);
  weight[v][u] = w;
}
```

Dijkstra

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

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

// O(E * lg(V))
vector<int> dijkstra(int s) {
  vector<set<DistNode>::iterator> pos(ady.size());
  vector<T> 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]) {
      T 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]) {
      T 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, T w) {
  ady[u].push_back(v);
  weight[u][v] = w;
  if (isDirected) return;
  ady[v].push_back(u);
  weight[v][u] = w;
}
```

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 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);
}
```

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 && (vis[v] ||
    ↪  !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] && !toposort(u)) return false;
  return true;
}
```

Maths

Number Theory

Divisibility Criterion

```python
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))
```

Extended Euclidean

```cpp
// 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};
}
```

GCD

```cpp
// recursive
int gcd(int a, int b) {
  return !b ? a : gcd(b, a % b);
}

// iterative
int gcd(int a, int b) {
  while (b) {
    a %= b;
    swap(a, b);
  }
  return a;
}
```

LCM

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

Prime Check Miller Rabin

```python
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
```

Prime Sieve

```cpp
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;
}
```

Strings

KMP

```cpp
// p = pattern, t = text
// f = error function, cf = create error function
// 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();) {
    while (i > -1 && p[i] != t[j]) i = f[i];
    i++, j++;
    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
}
```

Rabin Karp

```cpp
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;
}
```

Techniques

Binary Search

Multiple Queries

Mo

```
// q = query
// qs = queries

struct Query {
  int l, r;
};

int blksize;
vector<Query> qs;
vector<int> arr;

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

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

```
void getResults() {
  blksize = (int)sqrt(arr.size());
  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() {
  initVars(9, 2);
  arr = {1, 1, 2, 1, 3, 4, 5, 2, 8};
  qs = {{0, 8}, {3, 5}};
  getResults();
}
```

SQRT Decomposition

```
// sum of elements in range
int neutro = 0;
vector<int> arr;
vector<int> blks;
void initVars(int n) {
  arr.assign(n, neutro);
  blks.assign(sqrt(n), neutro);
}

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

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

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

int main() {
  initVars(10);
  arr = {1, 5, 2, 4, 6, 1, 3, 5, 7, 10};
  preprocess();
  for (int i = 0; i < blks.size() + 1; i++) cout <<
  ↪  blks[i] << " ";
  // output: 8 11 15 10
  cout << endl;
  cout << query(3, 8) << " ";
  cout << query(1, 6) << " ";
  update(8, 0);
  cout << query(8, 8) << endl;
  // output: 26 21 0
  return 0;
}
```