

ACM-ICPC-REFERENCE

Searlese

July 2018

Contents

1 Coding Resources	3		
1.1 C++	3	5.3.1 Combinations	11
1.1.1 DecimalPrecision	3	5.3.2 Permutations	11
1.1.2 IOoptimizationCPP	3	6 Rare Topics	11
1.1.3 IntToBinary	3	7 Strings	11
1.1.4 MapValueToInt	3	7.1 KMP	11
1.1.5 Permutations	3	7.2 RabinKarp	11
1.1.6 PrintVector	3	8 Techniques	12
1.1.7 PriorityQueueOfClass	3	8.1 BinarySearch	12
1.1.8 Random	3	8.2 DP	12
1.1.9 ReadLineCpp	4	8.3 Multiple Queries	12
1.1.10 SortPair	4	8.3.1 Mo	12
1.1.11 SortVectorOfClass	4	8.3.2 SparseTable	12
1.1.12 SplitString	4	8.3.3 SqrtDecomposition	12
1.1.13 Typedef	4	9 Faster But Longer	13
1.2 Python	4	9.1 BellmanFerrari	13
1.2.1 Combinations	4	9.2 KMP	13
1.2.2 Fast IO	4		
1.2.3 Permutations	4		
1.2.4 Random	4		
1.2.5 SortList	4		
1.2.6 SortListOfClass	4		
2 Data Structures	5		
2.1 BIT	5		
2.2 IntervalTree	5		
2.3 SegmentTree	5		
2.4 Trie	5		
2.5 UnionFind	5		
3 Geometry	6		
4 Graphs	6		
4.1 ArticulationPointsAndBridges	6		
4.2 ConnectedComponents	6		
4.3 CycleInDirectedGraph	7		
4.4 CycleInUndirectedGraph	7		
4.5 FloodFill	7		
4.6 Flow	7		
4.6.1 MaxFlowDinic	7		
4.7 IsBipartite	8		
4.8 KruskalMST	8		
4.9 LCA	8		
4.10 ShortestPaths	8		
4.10.1 BellmanFord	8		
4.10.2 Dijkstra	9		
4.11 StronglyConnectedComponents	9		
4.12 TopologicalSort	9		
5 Maths	10		
5.1 Game Theory	10		
5.2 Number Theory	10		
5.2.1 DivisibilityCriterion	10		
5.2.2 ExtendedEuclidean	10		
5.2.3 GCD	10		
5.2.4 LCM	10		
5.2.5 PrimeCheckMillerRabin	10		
5.2.6 PrimeSieve	11		
5.3 Probability	11		

Coding Resources

C++

DecimalPrecision

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

IOOptimizationCPP

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

IntToBinary

```
typedef long long int lli;
```

```
lli bitsInInt(lli n) {
    return floor(log2(n) + 1LL);
}
```

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

MapValueToInt

```
typedef string Key;
unordered_map<Key, int> val;
unordered_map<int, Key> getKey;
int mapId = 0;
```

```
int Map(Key key) {
    getKey[mapId] = key;
    return val.count(key) ? val[key]
        : val[key] = mapId++;
}
```

```
void initMapping() {
    mapId = 0;
    val.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;
}
```

PrintVector

```
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;
}
```

PriorityQueueOfClass

```
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

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

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

1.1.9 ReadLineCpp

```
// 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 SortPair

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

1.1.11 SortVectorOfClass

```
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 SplitString

```
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 an integer with k random bits.
random.getrandbits(k)
# shuffles a list
random.shuffle(li)
```

1.2.5 SortList

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

1.2.6 SortListOfClass

```
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 BIT

2.2 IntervalTree

2.3 SegmentTree

2.4 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();

    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
int prefixCount(string prefix) {
    Node *node = find(prefix);
    return node ? node->wpt : 0;
}

// number of words matching str
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();
    }
}

```

```

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

```

```

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;
}

```

```

int remove(string str) {
    int i = 0;
    return remove(root, str, i);
}
};

```

2.5 UnionFind

```

struct UnionFind {
    vector<int> dad, size;
}

```

```

int n;
UnionFind(int N) : 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;
}
};

```

3 Geometry

4 Graphs

4.1 ArticulationPointsAndBridges

```

// 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, N = 7, Time;
vector<vector<int>> ady;
vector<int> disc, low, ap;
vector<Edge> br;

void initVars() {
    ady = vector<vector<int>>(MAXN, 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;
}

void APB() {
    br.clear();
    ap = low = disc = vector<int>(MAXN);
    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);
}

```

4.2 ConnectedComponents

```

// comp = component
int MAXN = 26, N, compId = 1;
vector<vector<int>> ady;
vector<int> getComp;

void initVars() {
    ady = vector<vector<int>>(MAXN, vector<int>());
    getComp = vector<int>(MAXN);
}

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

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

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

```

4.3 CycleInDirectedGraph

```
int n; // max node id >= 0
vector<vector<int>> ady; // ady.resize(n)
vector<int> vis; // vis.resize(n)
vector<vector<int>> cycles;
vector<int> cycle;
bool flag = false;
int rootNode = -1;

bool hasDirectedCycle(int u) {
    vis[u] = 1;
    for (auto &v : ady[u]) {
        if (v == u || vis[v] == 2) continue;
        if (vis[v] == 1 || hasDirectedCycle(v)) {
            if (rootNode == -1)
                rootNode = v, flag = true;
            if (flag) {
                cycle.push_back(u);
                if (rootNode == u) flag = false;
            }
            return true;
        }
    }
    vis[u] = 2;
    return false;
}

bool hasDirectedCycle() {
    vis.clear();
    for (int u = 0; u < n; u++)
        if (!vis[u]) {
            cycle.clear();
            if (hasDirectedCycle(u))
                cycles.push_back(cycle);
        }
    return cycles.size() > 0;
}
```

4.4 CycleInUndirectedGraph

```
int n; // max node id >= 0
vector<vector<int>> ady; // ady.resize(n)
vector<bool> vis; // vis.resize(n)
vector<vector<int>> cycles;
vector<int> cycle;
bool flag = false;
int rootNode = -1;

bool hasUndirectedCycle(int u, int prev) {
    vis[u] = true;
    for (auto &v : ady[u]) {
        if (v == u || v == prev) continue;
        if (vis[v] || hasUndirectedCycle(v, u)) {
            if (rootNode == -1)
                rootNode = v, flag = true;
            if (flag) {
                cycle.push_back(u);
                if (rootNode == u) flag = false;
            }
        }
    }
}
```

```
        return true;
    }
}
return false;
}

bool hasUndirectedCycle() {
    vis.clear();
    for (int u = 0; u < n; u++)
        if (!vis[u]) {
            cycle.clear();
            if (hasUndirectedCycle(u, -1))
                cycles.push_back(cycle);
        }
    return cycles.size() > 0;
}
```

4.5 FloodFill

```
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);
}
```

4.6 Flow

4.6.1 MaxFlowDinic

```
// 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(MAXN, vector<int>()),
    cap(MAXN, vector<int>(MAXN)),
    flow(MAXN, vector<int>(MAXN));

bool levelGraph(int s, int t) {
    level = vector<int>(MAXN);
    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;
}

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);
}

```

4.7 IsBipartite

```

int n; // max node id >= 0
vector<vector<int>> ady; // ady.resize(n)

bool isBipartite() {
    vector<int> color(n, -1);
    for (int s = 0; s < n; 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;
}

```

4.8 KruskalMST

```

typedef int Weight;
typedef pair<int, int> Edge;
typedef pair<Weight, Edge> Wedge;

vector<Wedge> Wedges; // gets filled from input;
vector<Wedge> mst;

int kruskal() {
    int cost = 0;
    sort(Wedges.begin(), Wedges.end());
    // reverse(Wedges.begin(), Wedges.end());
    UnionFind uf(N);
    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;
}

```

4.9 LCA

4.10 ShortestPaths

4.10.1 BellmanFord

```

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

void initVars() {
    ady = vector<vector<int>>(MAXN, vector<int>());
    weight = vector<vector<int>>(
        MAXN, vector<int>(MAXN, INF));
}

vector<Weight> bellmanFord(int s) {
    vector<Weight> dist(MAXN, INF);
    dist[s] = 0;

```



```

for (int i = 0; i <= N; i++)
    for (int u = 0; u < N; u++)
        for (auto &v : ady[u]) {
            Weight w = weight[u][v];
            if (dist[u] != INF &&
                dist[v] > dist[u] + w) {
                if (i == N) return vector<Weight>();
                dist[v] = dist[u] + w;
            }
        }
return dist;
}

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

```

4.10.2 Dijkstra

```

typedef int Weight;
typedef pair<Weight, int> NodeDist;
int MAXN = 20001, INF = 1 << 30,
    isDirected = false;
vector<vector<int>> ady, weight;

void initVars() {
    ady = vector<vector<int>>(MAXN, vector<int>());
    weight = vector<vector<int>>(
        MAXN, vector<int>(MAXN, INF));
}

vector<Weight> dijkstra(int s) {
    vector<int> dist(MAXN, INF);
    set<NodeDist> q;
    q.insert({0, s});
    dist[s] = 0;
    while (!q.empty()) {
        NodeDist nd = *q.begin();
        q.erase(nd);
        int u = nd.second;
        for (int &v : ady[u]) {
            Weight w = weight[u][v];
            if (dist[v] > dist[u] + w) {
                if (dist[v] != INF) q.erase({dist[v], v});
                dist[v] = dist[u] + w;
                q.insert({dist[v], v});
            }
        }
    }
    return dist;
}

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

```

```

    ady[v].push_back(u);
    weight[v][u] = w;
}

```

4.11 StronglyConnectedComponents

```

// 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, N = 7, Time, top;
vector<vector<int>> ady, sccs;
vector<int> disc, low, s;

void initVars() {
    ady = vector<vector<int>>(MAXN, 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] = N;
            if (tv == u) break;
        }
        sccs.push_back(scc);
    }
}

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

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

```

4.12 TopologicalSort

```

int n; // max node id >= 0
vector<vector<int>> ady; // ady.resize(n)
vector<int> vis; // vis.resize(n)
vector<int> toposorted;

```

```

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;
}

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

```

5 Maths

5.1 Game Theory

5.2 Number Theory

5.2.1 DivisibilityCriteria

```

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))

```

5.2.2 ExtendedEuclidean

```

// 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};
}

```

5.2.3 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;
}

```

5.2.4 LCM

```

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

```

5.2.5 PrimeCheckMillerRabin

```

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

```

5.2.6 PrimeSieve

```

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;
}

```

5.3 Probability

5.3.1 Combinations

5.3.2 Permutations

6 Rare Topics

7 Strings

7.1 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(); j++) {
        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
}

```

7.2 RabinKarp

```

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;
}

```

8 Techniques

8.1 BinarySearch

8.2 DP

8.3 Multiple Queries

8.3.1 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 getResults() {
    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();
    getResults();
}

```

8.3.2 SparseTable

```
#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;
}
```

```
void buildSparseTabe(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))]); // problem specific
    }

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

    T queryArith(int L, int R) {
        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;
    }
}

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

using namespace std;

int N, blksize;
int MAXN = 100, MAXSQRT = (int)sqrt(MAXN);

vector<int> arr(MAXN);
vector<int> blks(MAXSQRT + 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
    }
}

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;
}

```

8.3.3 SqrtDecomposition

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

using namespace std;

int N, blksize;
int MAXN = 100, MAXSQRT = (int)sqrt(MAXN);

vector<int> arr(MAXN);
vector<int> blks(MAXSQRT + 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
    }
}

```

9 Faster But Longer

9.1 BellmanFerrari

// will be with queue

9.2 KMP