Jan 12th [Class Notes - Sliding Window (Jan 12).pdf

Q1 : Subarray Product Less Than K
(https://course.acciojob.com/idle?question=faddb193-7416-4426-8c42-99dcd35916a8)

```
public int numSubarrayProductLessThanK(int[] nums, int k) {
    int start = 0;
    int end = 0;
    int curr_prod = 1;
    int ans = 0;
    int n = nums.length;

    while(end < n) {
        // 1. expansion
        curr_prod = curr_prod * nums[end];
        end++;

        // 2. contraction
        while(start < end && curr_prod >= k) {
            curr_prod = curr_prod / nums[start];
            start++;
        }

        // 3. calculation
        if(curr_prod < k)
            ans += (end - start);
    }

    return ans;
}
```

TC : O(2N) => O(N) [Sliding window, every ele visited atmost 2 times]
SC : O(1)

Q2 : Maximum Consecutive Ones 2
(https://course.acciojob.com/idle?question=e40d0a42-b09e-4fc8-9be8-2d5fb508fdad)

```
static int maxOne(int arr[], int n,int k) {
    int start = 0;
    int end = 0;
    int flips = 0;
    int ans = 0;

    while(end < n) {
        // 1. expansion => if it is 0, flip and move
        if(arr[end] == 0) flips++;
        end++;

        // 2. contraction => if it is 0, remove that flip
        while(start < end && flips > k) {
            if(arr[start] == 0) flips--;
            start++;
        }

        // 3. calculation
        if(flips <= k)
            ans = Math.max(ans, end - start);
    }

    return ans;
}
```

TC : O(2N) => O(N) [Sliding window, every ele visited atmost 2 times]
SC : O(1)

Q3 : Longest Substring with Unique Characters
(https://course.acciojob.com/idle?question=78adbda5-bbd0-4246-b173-5301ca6a0ac0)

```java
public int longestSubstring(String s) {
    int n = s.length();
    int start = 0;
    int end = 0;
    int[] freq = new int[256];
    int repeat = 0;
    int ans = 0;

    while(end < n) {

        freq[s.charAt(end)]++;
        if(freq[s.charAt(end)] > 1) repeat++;
        end++;

        while(start < end && repeat > 0) {
            if(freq[s.charAt(start)] > 1) repeat--;
            freq[s.charAt(start)]--;
            start++;
        }

        if(repeat == 0)
            ans = Math.max(ans, end - start);
    }

    return ans;
}
```

TC : O(2N) => O(N) [Sliding window, every ele visited atmost 2 times]
SC : O(256) => O(Constant)

Using HashMap

```java
public int longestSubstring(String s) {
    int n = s.length();
    int start = 0;
    int end = 0;
    HashMap<Character, Integer> hm = new HashMap<>();
    int repeat = 0;
    int ans = 0;

    while(end < n) {
        hm.put(s.charAt(end), hm.getOrDefault(s.charAt(end), 0) + 1);
        if(hm.get(s.charAt(end)) > 1) repeat++;
        end++;

        while(start < end && repeat > 0) {
            if(hm.get(s.charAt(start)) > 1) repeat--;
            hm.put(s.charAt(start), hm.get(s.charAt(start)) - 1);
            start++;
        }

        if(repeat == 0)
            ans = Math.max(ans, end - start);
    }

    return ans;
}
```

TC : O(2N) => O(N) [Sliding window, every ele visited atmost 2 times]
SC : O(N) [HashMap]

Q4 : Longest Substring with At Least K Repeating Characters
(https://course.acciojob.com/idle?question=9be96ee1-cf8d-4bcc-936b-899f22782a7f)

```java
private int xUniquek(String s, int x, int k) {
    int start = 0;
    int end = 0;
    int[] freq = new int[123];
    int unique = 0;
    int char_atleastk = 0;
    int ans = 0;
    int n = s.length();

    while(end < n) {

        // 1. expansion
        freq[s.charAt(end)]++;
        // char is coming first time
        if(freq[s.charAt(end)] == 1) unique++;
        // we got a char which is satisfying atleast k
        if(freq[s.charAt(end)] == k) char_atleastk++;
        end++;

        // 2. contraction
        while(start < end && unique > x) {
            freq[s.charAt(start)]--;
            // we lost a unique char
            if(freq[s.charAt(start)] == 0) unique--;
            // we lost a char which was satisfying atleast k
            if(freq[s.charAt(start)] == k - 1) char_atleastk--;
            start++;
        }

        // calc only if there are exactly x unique,
        // and all those x unique are satisfying atleast k
        if(unique == x && char_atleastk == x)
            ans = Math.max(ans, end - start);
    }

    return ans;
}
```

```java
public int longestSubstring(String s, int k) {
    int totalUnq = 0;
    int[] freq = new int[123];
    int n = s.length();
    for(int i = 0; i < n; i++) {
        freq[s.charAt(i)]++;
        if(freq[s.charAt(i)] == 1) totalUnq++;
    }

    int ans = 0;
    for(int x = 1; x <= totalUnq; x++)
        ans = Math.max(ans, xUniquek(s, x, k));

    return ans;
}
```

TC : O(2N * 26) => O(N * 26) => O(N)
SC : O(123) => O(Constant)

Jan 14th [Class Notes - Sliding Window (Jan 14).pdf

Q5 : Minimum Window Substring
(https://course.acciojob.com/idle?question=8c817174-62b5-46e8-8cf7-00cc0d0ffa47)

```java
private boolean check(int[] sfreq, int[] tfreq) {
    for(int i = 65; i < 123; i++) {
        if(tfreq[i] > sfreq[i]) return false;
    }
    return true;
}

public String minWindow(String s, String t) {

    int n = s.length();
    int m = t.length();

    // 1. Build your tfreq
    int[] tfreq = new int[123];
    for(int i = 0; i < m; i++)
        tfreq[t.charAt(i)]++;

    int start = 0;
    int end = 0;
    int[] sfreq = new int[123];
    int ans = Integer.MAX_VALUE;
    int ansStart = -1;
    int ansEnd = -1;

    while(end < n) {
        // 1. Expansion => add it to sfreq
        sfreq[s.charAt(end)]++;
        end++;
        // 2. Calculate and Contract, because we need min window
        while(start < end && check(sfreq, tfreq)) {
            if(ans > end - start) {
                ans = end - start;
                ansStart = start;
                ansEnd = end;
            }
            sfreq[s.charAt(start)]--;
            start++;
        }
    }

    // edge case where we didnt find any window
    if(ans == Integer.MAX_VALUE) return "";
    return s.substring(ansStart, ansEnd);
}
```

TC : O(2N * 123) => O(N * 123) => O(N)
SC : O(2 * 123) => O(Constant)

Q6 : Substring with Concatenation of All Words
(https://course.acciojob.com/idle?question=55e9731b-af80-4a27-af05-c4455e4caf88)

```java
private boolean check_excess(HashMap<String, Integer> shm, HashMap<String, Integer>
thm) {
    for(String s : shm.keySet()) {
        // Word is there in curr window (shm) but not in words(thm)
        // => excessWord is present
        if(thm.containsKey(s) == false) return true;

        // Word is there in both, but word is there in curr window(shm)
        // more than required times => excessWord is present
        if(shm.get(s) > thm.get(s)) return true;
    }

    return false; // no excessWord
}

public List<Integer> findSubstring(String s, String[] words) {
    HashMap<String, Integer> thm = new HashMap<>();
    int totalWords = words.length;
    for(int i = 0; i < totalWords; i++) {
        thm.put(words[i], thm.getOrDefault(words[i], 0) + 1);
    }

    int wordLen = words[0].length();
    List<Integer> ans = new ArrayList<>();
    int n = s.length();
    for(int i = 0; i < wordLen; i++) {
        int start = i;
        int end = i;
        HashMap<String, Integer> shm = new HashMap<>();

        while(end + wordLen <= n) {
            // 1.expansion => add word to map
            String temp = s.substring(end, end + wordLen);
            shm.put(temp, shm.getOrDefault(temp, 0) + 1);
            end += wordLen;
            // 2.contraction => remove excess words
            while(start < end && check_excess(shm, thm)) {
                String str = s.substring(start, start + wordLen);
                shm.put(str, shm.get(str) - 1);
                if(shm.get(str) == 0) shm.remove(str);
                start += wordLen;
            }

            // 3. calculation
            // if(shm.equals(thm)) ans.add(start);
            if(end - start == totalWords * wordLen)
                ans.add(start);
        }
    }

    return ans;
}
```

TC : O(N^2)
SC : O(N)

Q7 : Sliding window Maximum
(https://course.acciojob.com/idle?question=2da7ad22-cccc-497d-864c-a3ea784e1263)

```java
static int[] nextGreaterRightIdx(int[] arr) {
    int n = arr.length;
    int[] nge = new int[n];
    Stack<Integer> st = new Stack<>();

    for(int i = 0; i < n; i++) {
        while(st.size() > 0 && arr[i] > arr[st.peek()]) {
            nge[st.peek()] = i;
            st.pop();
        }
        st.push(i);
    }

    while(st.size() > 0) {
        nge[st.peek()] = n;
        st.pop();
    }
    return nge;
}

static int[] SlidingWindowMaximum(int n, int k, int[] arr){
    int[] nge = nextGreaterRightIdx(arr);
    int[] ans = new int[n - k + 1];

    int j = 0;
    for(int i = 0; i <= n - k; i++) {
        if(j < i) j = i;

        while(nge[j] < i + k) {
            j = nge[j];
        }

        ans[i] = arr[j];
    }
    return ans;
}
```

TC : O(N) and SC : O(N)

Jan 15th [Class Notes - Contest Discussion (Jan 15).pdf]

Q8 : Sum of Leaf Nodes
(https://course.acciojob.com/idle?question=27f200f6-90cd-4bd2-94a7-3cc25ded8048)

Post order method

```
public static int leafIt(Node root)
{
    if(root == null) return 0;
    if(root.left == null && root.right == null)
        return root.data;

    int lsum = leafIt(root.left);
    int rsum = leafIt(root.right);
    return lsum + rsum;
}
```

Pre order method

```
int sum = 0;
private static void solve(Node root) {
    if(root == null) return;
    if(root.left == null && root.right == null) {
        sum += root.data;
        return;
    }
    solve(root.left);
    solve(root.right);
}

public static int leafIt(Node root) {
    solve(root);
    return sum
}
```

TC : O(N)
SC : O(N) [Recursive Stack Space]

Q9 : Min Window with distinct element equal to K
(https://course.acciojob.com/idle?question=b759dd60-d890-43e7-8971-7c24ee8c64fe)

```java
public List<Integer> coldDrink(int arr[], int n, int k) {

    int[] freq = new int[10001];
    int start = 0;
    int end = 0;
    int distinct = 0;

    int ans = Integer.MAX_VALUE;
    int ansStart = -1;
    int ansEnd = -1;

    while(end < n) {
        freq[arr[end]]++;
        if(freq[arr[end]] == 1) distinct++;
        end++;

        while(start < end && distinct == k) {
            if(ans > end - start) {
                ans = end - start;
                ansStart = start;
                ansEnd = end;
            }

            if(freq[arr[start]] == 1) distinct--;
            freq[arr[start]]--;
            start++;
        }
    }

    if(ans == Integer.MAX_VALUE) return Arrays.asList(-1);
    return Arrays.asList(ansStart, ansEnd - 1);
}
```

TC : O(2N) => O(N) [Sliding window, every ele visited atmost 2 times]
SC : O(10^4) => O(Constant)

Q10 : Maximum Sum BST in Binary Tree
(https://course.acciojob.com/idle?question=504e980e-fbca-4776-9612-7c0d4dd7cc2c)

```java
class Info {
    int max;
    int min;
    int sum;
    boolean isBST;

    Info() {}
    Info(int max, int min, int sum, boolean isBST) {
        this.max = max;
        this.min = min;
        this.sum = sum;
        this.isBST = isBST;
    }
}

class Solution {

    int maxSum = 0;

    // max, min, subtree sum, isBST
    Info solve(TreeNode root) {

        if(root == null)
            return new Info(Integer.MIN_VALUE, Integer.MAX_VALUE, 0, true);

        Info lInfo = solve(root.left);
        Info rInfo = solve(root.right);
        Info mInfo = new Info();

        int lmax = lInfo.max;
        int rmax = rInfo.max;
        mInfo.max = Math.max(root.val, Math.max(lmax, rmax));

        int lmin = lInfo.min;
        int rmin = rInfo.min;
        mInfo.min = Math.min(root.val, Math.min(lmin, rmin));
        int lsum = lInfo.sum;
        int rsum = rInfo.sum;
        mInfo.sum = lsum + root.val + rsum;

        boolean lBST = lInfo.isBST;
        boolean rBST = rInfo.isBST;
        mInfo.isBST = lBST && rBST && (lmax < root.val) && (rmin > root.val);

        if(mInfo.isBST) maxSum = Math.max(maxSum, mInfo.sum);
        return mInfo;
    }

    public int maxSumBST(TreeNode root) {
        solve(root);
        return maxSum;
    }
}
```

TC : O(N)
SC : O(N) [Recursive Stack Space]

Q11 : Validate BST
(https://course.acciojob.com/idle?question=3b992182-31dd-4aaa-a4fd-914a6a9669ff)

```
private boolean solve(TreeNode root, long min, long max) {
    if(root == null) return true;
    if(root.val <= min || root.val >= max) return false;
    return solve(root.left, min, root.val) && solve(root.right, root.val, max);
}

public boolean isValidBST(TreeNode root) {
    return solve(root, Long.MIN_VALUE, Long.MAX_VALUE);
}
```

TC : O(N)
SC : O(N) [Recursive Stack Space]

Q12 : Optimized Minimum Window Substring
(https://course.acciojob.com/idle?question=8c817174-62b5-46e8-8cf7-00cc0d0ffa47)

```java
public String minWindow(String s, String t) {

    int n = s.length();
    int m = t.length();
    // 1. Build your tfreq
    int[] tfreq = new int[123];
    int distinctT = 0;
    for(int i = 0; i < m; i++) {
        tfreq[t.charAt(i)]++;
        if(tfreq[t.charAt(i)] == 1)
            distinctT++;
    }

    int start = 0;
    int end = 0;
    int[] sfreq = new int[123];

    int ans = Integer.MAX_VALUE;
    int ansStart = -1;
    int ansEnd = -1;
    int charSatisfied = 0;

    while(end < n) {
        // 1. Expansion => add it to sfreq
        sfreq[s.charAt(end)]++;
        if(sfreq[s.charAt(end)] == tfreq[s.charAt(end)]) charSatisfied++;
        end++;

        // 2. Caculate and shrink
        while(start < end && charSatisfied == distinctT) {
            if(ans > end - start) {
                ans = end - start;
                ansStart = start;
                ansEnd = end;
            }
            if(sfreq[s.charAt(start)] == tfreq[s.charAt(start)]) charSatisfied--;
            sfreq[s.charAt(start)]--;
            start++;
        }
    }

    if(ans == Integer.MAX_VALUE) return "";
    return s.substring(ansStart, ansEnd);
}
```

TC : O(2N) => O(N) [Sliding window, every ele visited atmost 2 times]
SC : O(2 * 123) => O(Constant)

Q13 : BST Using Level Order
(https://course.acciojob.com/idle?question=78d88a42-f9ae-4454-a580-518034855d54)

```java
class pair {
    Node par = null;
    int lr = -(int)1e9, rr = (int)1e9;

    pair(){}
    pair(Node par, int lr, int rr){
        this.par = par;
        this.lr = lr;
        this.rr = rr;
    }
}

Node bstFromLevel(int arr[],int n) {
    LinkedList<pair> que = new LinkedList<>();
    que.add(new pair());
    Node root = null;
    int idx = 0;
    while(que.size()!= 0 && idx < n) {
        pair rem = que.removeFirst();
        int ele = arr[idx];

        if(ele < rem.lr || ele > rem.rr) continue;

        Node node = new Node(ele);
        idx++;

        Node par = rem.par;
        if(par == null) root = node;
        else if(ele < par.data) par.left = node;
        else par.right = node;

        que.addLast(new pair(node, rem.lr, ele));
        que.addLast(new pair(node, ele, rem.rr));
    }
    return root;
}
```

TC : O(N) and SC : O(N)

Jan 16th [Class Notes - Graphs (Jan 16).pdf]

Q14 : Constructing graph

```java
import java.util.*;
class Edge {
    int v;
    int w;
    Edge(int v, int w) {
        this.v = v;
        this.w = w;
    }
}

public class Main {
    public static void display(ArrayList<Edge>[] graph) {
        int vertices = graph.length;
        for (int i = 0; i < vertices; i++) {
            System.out.print(i + " -> ");
            for (Edge e : graph[i]) {
                System.out.print("(" + e.v + ", " + e.w + ")");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int vertices = sc.nextInt();
        int edges = sc.nextInt();

        // we are making an array of [vertices]
        ArrayList<Edge>[] graph = new ArrayList[vertices];
        for (int i = 0; i < vertices; i++) {
            // we are initializing Array list in each box(element)
            graph[i] = new ArrayList<>();
        }
        for (int i = 0; i < edges; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            int w = sc.nextInt();
            graph[u].add(new Edge(v, w));
            graph[v].add(new Edge(u, w));
            // if it is directed you can ignore this line
        }

        // for non weighted no need Edge, simply add only v to the list
        ArrayList<Integer>[] graphNW = new ArrayList[vertices];
        for (int i = 0; i < vertices; i++) {
            graphNW[i] = new ArrayList<>();
        }

        for (int i = 0; i < edges; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            graphNW[u].add(v);
            graphNW[v].add(u);
            // if it is directed you can ignore this line
        }

        // Adjacency Matrix
        int[][] graphMatrix = new int[vertices][vertices];
        for (int i = 0; i < edges; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            int w = sc.nextInt();

            graphMatrix[u][v] = w;
            graphMatrix[v][u] = w;
        }

        display(graph);
        sc.close();
    }
}
```

to construct an adj.list, TC : $O(V + E)$
to construct an adj.Matrix, TC : $O(V^2)$

Q15 : DFS
(https://course.acciojob.com/idle?question=bc51d934-d08b-4b7d-a852-2dec91e07bce)

```java
public static void dfs(int src, ArrayList<Integer>[] adj, boolean[] visited) {
    System.out.print(src + " ");
    visited[src] = true;
    Collections.sort(adj[src]);
    for(int nbr : adj[src]) {
        if(visited[nbr] == false)
            dfs(nbr, adj, visited);
    }
}

public static void DFSTraversal(List<List<Integer>> edges, int vertices) {
    // 1. Build our adjacency list
    ArrayList<Integer>[] graphNW = new ArrayList[vertices];
    for (int i = 0; i < vertices; i++) {
        graphNW[i] = new ArrayList<>();
    }

    for (List<Integer> edge : edges) {
        int u = edge.get(0);
        int v = edge.get(1);
        graphNW[u].add(v);
        graphNW[v].add(u);
    }

    // 2. calling dfs, works for disconnected graph as well
    boolean[] visited = new boolean[vertices];
    for(int i = 0; i < vertices; i++) {
        if(visited[i] == false)
            dfs(i, graphNW, visited);
    }
}
```

TC : O(V + E) => we are travelling each edge and vertex only once

Jan 17th [Class Notes - Graphs (Jan 17, 18).pdf]

Q16 : Path between 2 vertices
(https://course.acciojob.com/idle?question=2b6f51da-cb55-412f-a0b5-47724e221ca3)

```
boolean dfs(int src, int dest, ArrayList<Integer>[] graph, boolean[] visited) {
    if(src == dest) return true;

    visited[src] = true;
    boolean res = false;
    for(int nbr : graph[src]) {
        if(visited[nbr] == false) {
            res = res || dfs(nbr, dest, graph, visited);
        }
    }
    return res;
}

boolean check(int vertices, int edges, ArrayList<ArrayList<Integer>> edgeList, int
src, int dest) {
    ArrayList<Integer>[] graph = new ArrayList[vertices];
    for(int i = 0; i < vertices; i++) {
        graph[i] = new ArrayList<>();
    }

    for(ArrayList<Integer> edge : edgeList) {
        int u = edge.get(0);
        int v = edge.get(1);
        graph[u - 1].add(v - 1);
        graph[v - 1].add(u - 1);
    }

    boolean[] visited = new boolean[vertices];
    return dfs(src - 1, dest - 1, graph, visited);
}
```

TC : O(V + E) [dfs on adj.List]
SC : O(V + E)

Q17 : Count components
(https://course.acciojob.com/idle?question=ea0ba181-faaa-4ec3-89d2-c40d022b321f)

Using DFS

```java
private void dfs(int src, ArrayList<ArrayList<Integer>> adj, boolean[] visited) {
    visited[src] = true;
    int vertices = adj.size();
    for(int nbr = 0; nbr < vertices; nbr++) {
        if(adj.get(src).get(nbr) == 1 && visited[nbr] == false) {
            dfs(nbr, adj, visited);
        }
    }
}

int components(ArrayList<ArrayList<Integer>> adj, int vertices) {
    boolean[] visited = new boolean[vertices];
    int cnt = 0;
    for(int i = 0; i < vertices; i++) {
        if(visited[i] == false) {
            dfs(i, adj, visited);
            cnt++;
        }
    }
    return cnt;
}
```

TC : O(V^2) [dfs on adj.Matrix]
SC : O(V) [for visited array, recursive stack]

Using BFS

```
private void BFS(int src, ArrayList<ArrayList<Integer>> adj, boolean[] visited)
{
    Queue<Integer> q = new LinkedList<>();
    int vertices = adj.size();

    q.add(src);
    visited[src] = true;

    while(q.size() > 0)
    {
        int size = q.size();
        for(int i = 0; i < size; i++)
        {
            int temp = q.remove();
            for(int nbr = 0; nbr < vertices; nbr++)
            {
                if(adj.get(temp).get(nbr) == 1 && visited[nbr] == false) {
                    q.add(nbr);
                    visited[nbr] = true;
                }
            }
        }
    }
}

int components(ArrayList<ArrayList<Integer>> adj, int vertices) {
    boolean[] visited = new boolean[vertices];
    int cnt = 0;
    for(int i = 0; i < vertices; i++) {
        if(visited[i] == false) {
            BFS(i, adj, visited);
            cnt++;
        }
    }
    return cnt;
}
```

TC : O(V^2) [bfs on adj.Matrix]
SC : O(V) [for visited array, queue]

Q18 : Print Paths
(https://course.acciojob.com/idle?question=2b6f51da-cb55-412f-a0b5-47724e221ca3)

```
private static void dfs(int src, int dest, ArrayList<Edge>[]graph , boolean[]
visited, String path) {
    if(src == dest) {
        System.out.println(path);
        return;
    }

    visited[src] = true;
    for(Edge e : graph[src]) {
        if(visited[e.nbr] == false) {
            dfs(e.nbr, dest, graph, visited, path + e.nbr);
        }
    }
    visited[src] = false; // backtrack(erase)
}

public static void printAllPath(ArrayList<Edge>[]graph , int src , int dest , int
vertices){
   boolean[] visited = new boolean[vertices];
   String path = "" + src;
   dfs(src, dest, graph, visited, path);
}
```

Time complexity is O(2^n), Here n is vertices
Think about this worst case scenario : Suppose we have n nodes, labeled 0 to n-1.
And for each pair of nodes i and j, if i < j, let there be an edge between node i and node j
Now, we want to calculate how many paths there are from the 0th node to the (n-1)th node.
Well, notice that each path of length k corresponds to some choice of (k - 1) nodes between 0 and (n-1).
For example, here are two paths of length 2: so any 1 node in between 0 and (n-1)
0->3->(n-1)
0->5->(n-1)
Here is a path of length 3: so any 2 node in between 0 and (n-1)
0->1->5->(n-1)
How many paths of length k are there? The answer is (n-2 choose k-1) because we pick k - 1 nodes between 0 and (n - 1).
The total number of paths is the sum of (n-2 choose k-1) for k = 1, 2, ..., (n-2).
Using the binomial theorem, this sum is equal to 2^(n-2) which is O(2^n).
Space complexity is O(Vertices) [Recursive stack space]

Using List as a path
(https://leetcode.com/problems/all-paths-from-source-to-target/description/)

```
private void dfs(int src, int dest, int[][] graph , boolean[] visited, List<Integer>
path, List<List<Integer>>ans) {
    if(src == dest) {
        List<Integer> temp = new ArrayList<>(path);
        ans.add(temp);
        return;
    }

    visited[src] = true;
    for(int nbr : graph[src]) {
        if(visited[nbr] == false) {
            path.add(nbr);
            dfs(nbr, dest, graph, visited, path, ans);
            path.remove(path.size() - 1);
        }
    }
    visited[src] = false;
}

public List<List<Integer>> allPathsSourceTarget(int[][] graph) {
    List<Integer>path = new ArrayList<>();
    List<List<Integer>>ans = new ArrayList<>();
    int vertices = graph.length;
    boolean[] visited = new boolean[vertices];

    path.add(0);
    dfs(0, vertices - 1, graph, visited, path, ans);
    return ans;
}
```

TC : O($2^{vertices}$)
SC : O($2^{vertices}$) [to store all the paths]

Q20 : BFS Implementation
(https://course.acciojob.com/idle?question=9b2d5eac-703f-487c-8921-a9a641811594)

```java
void BFS(int src) {
    Queue<Integer> q = new LinkedList<>();
    boolean[] visited = new boolean[vertices];

    q.add(src);
    visited[src] = true;
    while(q.size() > 0)
    {
        int size = q.size();
        for(int i = 0; i < size; i++)
        {
            int temp = q.remove();
            System.out.print(temp + " ");
            for(int nbr : adjList[temp])
            {
                if(visited[nbr] == false) {
                    q.add(nbr);
                    visited[nbr] = true;
                }
            }
        }
    }
}
```

TC : O(V + E) [bfs on adj.List]
SC : O(V) [for visited array, queue]

Q21 : Shortest path in Undirected Graph having unit distance
(https://practice.geeksforgeeks.org/problems/shortest-path-in-undirected-graph-having-unit-distance/1)

```java
int[] BFS(int src, ArrayList<Integer>[] graph) {
    int vertices = graph.length;
    Queue<Integer> q = new LinkedList<>();
    boolean[] visited = new boolean[vertices];
    int[] distance = new int[vertices];
    Arrays.fill(distance, -1);

    q.add(src);
    visited[src] = true;
    int level = 0;
    while(q.size() > 0) {
        int size = q.size();
        for(int i = 0; i < size; i++) {
            int temp = q.remove();
            distance[temp] = level;
            for(int nbr : graph[temp]) {
                if(visited[nbr] == false) {
                    q.add(nbr);
                    visited[nbr] = true;
                }
            }
        }
        level++;
    }
    return distance;
}
public int[] shortestPath(int[][] edgeList, int vertices, int edges, int src) {
    ArrayList<Integer>[] graph = new ArrayList[vertices];
    for(int i = 0; i < vertices; i++) graph[i] = new ArrayList<>();
    for(int i = 0; i < edges; i++) {
        int u = edgeList[i][0];
        int v = edgeList[i][1];
        graph[u].add(v);
        graph[v].add(u);
    }
    return BFS(src, graph);
}
```

Q22 : Detect cycle in an undirected graph
(https://course.acciojob.com/idle?question=c5f395d4-c8ed-4c3c-88e9-285a189bddca)

Using DFS

```java
private static boolean checkCycle(int parent, int src, ArrayList<ArrayList<Integer>>
graph, boolean[] visited) {

    visited[src] = true;
    for(int nbr : graph.get(src)) {
        if(visited[nbr] == false) {
            boolean check = checkCycle(src, nbr, graph, visited);
            if(check == true) return true;
        }
        else if(visited[nbr] == true && nbr != parent) return true;
    }
    return false;
}

public static boolean isCycle(int vertices, ArrayList<ArrayList<Integer>> graph) {
    boolean[] visited = new boolean[vertices];
    for(int i = 0; i < vertices; i++) {
        if(visited[i] == false) {
            boolean check = checkCycle(-1, i, graph, visited);
            if(check == true) return true;
        }
    }
    return false;
}
```

TC : O(V + E) [dfs on adj.List]
SC : O(V) [for visited array, recursive stack space]

Using BFS

```java
private static boolean checkCycle(int src, ArrayList<ArrayList<Integer>> graph,
boolean[] visited, int[] parent) {
    Queue<Integer> q = new LinkedList<>();
    q.add(src);
    visited[src] = true;
    parent[src] = -1;

    while(q.size() > 0) {
        int size = q.size();
        for(int i = 0; i < size; i++) {
            int temp = q.remove();
            for(int nbr : graph.get(temp)) {
                if(visited[nbr] == false) {
                    q.add(nbr);
                    visited[nbr] = true;
                    parent[nbr] = temp;
                }
                else if(visited[nbr] == true && nbr != parent[temp]) return true;
            }
        }
    }
    return false;
}

public static boolean isCycle(int vertices, ArrayList<ArrayList<Integer>> graph) {
    boolean[] visited = new boolean[vertices];
    int[] parent = new int[vertices];

    for(int i = 0; i < vertices; i++) {
        if(visited[i] == false) {
            boolean check = checkCycle(i, graph, visited, parent);
            if(check == true) return true;
        }
    }
    return false;
}
```

TC : O(V + E) [bfs on adj.List]
SC : O(V) [for visited array, queue]

Jan 18th [Class Notes - Graphs (Jan 17, 18).pdf]

Q23 : Detect cycle in a directed graph
(https://course.acciojob.com/idle?question=f418b898-6922-4c8a-91f3-9cf060a62957)

```java
private boolean dfs(int src, ArrayList<Integer>[] graph, boolean[] visited, boolean[]
path)
{
    visited[src] = true;
    path[src] = true;
    for(int nbr : graph[src]) {
        if(visited[nbr] == false) {
            boolean check = dfs(nbr, graph, visited, path);
            if(check == true) return true;
        }
        else if(visited[nbr] && path[nbr]) return true;
    }
    path[src] = false;
    return false;
}

public boolean isCyclic(int vertices, ArrayList<Integer>[] graph)
{
    boolean[] visited = new boolean[vertices];
    boolean[] path = new boolean[vertices];
    for(int i = 0; i < vertices; i++) {
        if(visited[i] == false) {
            boolean check = dfs(i, graph, visited, path);
            if(check == true) return true;
        }
    }
    return false;
}
```

TC : O(V + E) [dfs on adj.List]
SC : O(V) [for visited array, path array, recursive stack space]

Q24 : Bipartite Graph
(https://course.acciojob.com/idle?question=717ac001-d919-4e01-ae4e-db1cbc4b8cba)

Using both bfs and dfs approaches

```java
private boolean bfs(int src, ArrayList<Integer>[] graph, boolean[] visited, int[]
color)
{
    Queue<Integer> q = new LinkedList<>();
    q.add(src);
    visited[src] = true;
    color[src] = 0; // red

    while(q.size() > 0)
    {
        int size = q.size();
        for(int i = 0; i < size; i++)
        {
            int temp = q.remove();
            for(int nbr : graph[temp])
            {
                if(visited[nbr] == false)
                {
                    q.add(nbr);
                    visited[nbr] = true;
                    color[nbr] = 1 - color[temp];
                    // 1 - 0 = 1 // 1 - 1 = 0
                }
                else if(visited[nbr] && color[temp] == color[nbr])
                    return false;
            }
        }
    }
    return true;
}
```

```java
private boolean dfs(int src, ArrayList<Integer>[] graph, boolean[] visited, int[]
color, int c)
{
    visited[src] = true;
    color[src] = c;
    for(int nbr : graph[src]) {
        if(visited[nbr] == false) {
            boolean check = dfs(nbr, graph, visited, color, 1 - c);
            if(check == false) return false;
        }
        else if(visited[nbr] && color[src] == color[nbr])
            return false;
    }
    return true;
}

public int possibleBipartition(int n, int[][] dislikes) {
    ArrayList<Integer>[] graph = new ArrayList[n];
    for(int i = 0; i < n; i++) graph[i] = new ArrayList<>();
    for(int[] edge : dislikes) {
        int u = edge[0];
        int v = edge[1];
        // vertices from 1 to n, so make it 0 to n - 1
        graph[u - 1].add(v - 1);
        graph[v - 1].add(u - 1);
    }

    boolean[] visited = new boolean[n];
    int[] color = new int[n];
    for(int i = 0; i < n; i++) {
        if(visited[i] == false) {
            boolean check = dfs(i, graph, visited, color, 0);
            // boolean check = bfs(i, graph, visited, color)
            if(check == false) return 0;
        }
    }
    return 1;
}
```

TC : O(V + E)
SC : O(V)

Q25 : Flood fill
(https://course.acciojob.com/idle?question=842a9b40-7830-4e90-bd25-bffc65a02055)

Using both bfs and dfs approaches

```java
private static int[] dr = new int[]{-1, 0, 1, 0};
private static int[] dc = new int[]{0, 1, 0, -1};

private static void bfs(int r, int c, int[][] grid, boolean[][] visited, int
newColor)
{
    Queue<int[]> q = new LinkedList<>();
    visited[r][c] = true;
    q.add(new int[]{r, c});
    int rows = grid.length;
    int cols = grid[0].length;

    while(q.size() > 0)
    {
        int size = q.size();
        for(int i = 0; i < size; i++)
        {
            int[] temp = q.remove();
            int sr = temp[0];
            int sc = temp[1];
            int myColor = grid[sr][sc];
            grid[sr][sc] = newColor;
            for(int j = 0; j < 4; j++)
            {
                int nr = sr + dr[j];
                int nc = sc + dc[j];
                if(nr < 0 || nr >= rows || nc < 0 || nc >= cols) continue;
                if(visited[nr][nc] == false && grid[nr][nc] == myColor) {
                    q.add(new int[]{nr, nc});
                    visited[nr][nc] = true;
                }
            }
        }
    }
}
```

```java
private static void dfs(int r, int c, int[][] grid, boolean[][] visited, int
newColor)
{
    visited[r][c] = true;
    int myColor = grid[r][c];
    grid[r][c] = newColor;

    int rows = grid.length;
    int cols = grid[0].length;

    for(int i = 0; i < 4; i++) {
        int nr = r + dr[i];
        int nc = c + dc[i];
        if(nr < 0 || nr >= rows || nc < 0 || nc >= cols) continue;
        if(visited[nr][nc] == false && grid[nr][nc] == myColor) {
            dfs(nr, nc, grid, visited, newColor);
        }
    }
}

public static void FloodFill(int[][] grid, int r, int c, int newColor) {
    int rows = grid.length;
    int cols = grid[0].length;
    if(r < 0 || r >= rows || c < 0 || c >= cols || grid[r][c] == newColor)
        return;

    boolean[][] visited = new boolean[rows][cols];
    bfs(r, c, grid, visited, newColor);
    // dfs(r, c, grid, visited, newColor);
    return;
}
```

TC : O(rows * cols) [We can visited each cell only once]
SC : O(rows * cols) [Visited array]

Q26 : Number of islands
(https://practice.geeksforgeeks.org/problems/find-the-number-of-islands/1)
(https://course.acciojob.com/idle?question=89d49818-69fe-4020-8763-4651f1a6f6ae)

Using both bfs and dfs approaches

```
private static int[] dr = new int[]{-1, -1, 0, 1, 1, 1, 0, -1};
private static int[] dc = new int[]{ 0, 1, 1, 1, 0, -1, -1, -1};

private static void bfs(int r, int c, int[][] grid, boolean[][] visited)
{
    Queue<int[]> q = new LinkedList<>();
    visited[r][c] = true;
    q.add(new int[]{r, c});
    int rows = grid.length;
    int cols = grid[0].length;

    while(q.size() > 0)
    {
        int size = q.size();
        for(int i = 0; i < size; i++)
        {
            int[] temp = q.remove();
            int sr = temp[0];
            int sc = temp[1];
            for(int j = 0; j < 8; j++)
            {
                int nr = sr + dr[j];
                int nc = sc + dc[j];
                if(nr < 0 || nr >= rows || nc < 0 || nc >= cols) continue;
                if(visited[nr][nc] == false && grid[nr][nc] == 1) {
                    q.add(new int[]{nr, nc});
                    visited[nr][nc] = true;
                }
            }
        }
    }
}
```

```java
private static void dfs(int r, int c, int[][] grid, boolean[][] visited)
{
    int rows = grid.length;
    int cols = grid[0].length;
    visited[r][c] = true;
    for(int i = 0; i < 8; i++) {
        int nr = r + dr[i];
        int nc = c + dc[i];
        if(nr < 0 || nr >= rows || nc < 0 || nc >= cols) continue;
        if(visited[nr][nc] == false && grid[nr][nc] == 1) {
            dfs(nr, nc, grid, visited);
        }
    }
}

static int numberOfIslands(int[][] grid, int n, int m){
    boolean[][] visited = new boolean[n][m];
    int cnt = 0;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < m; j++) {
            if(grid[i][j] == 1 && visited[i][j] == false) {
                bfs(i, j, grid, visited);
                // dfs(i, j, grid, visited);
                cnt++;
            }
        }
    }
    return cnt;
}
```

TC : O(rows * cols) [We can visited each cell only once]
SC : O(rows * cols) [Visited array]

Q27 : Max Area of Island
(https://leetcode.com/problems/max-area-of-island/description/)

Using dfs, and bfs is homework

```java
private int[] dr = new int[]{-1, 0, 1, 0};
private int[] dc = new int[]{0, 1, 0, -1};

private int dfs(int r, int c, int[][] grid, boolean[][] visited) {
    int rows = grid.length;
    int cols = grid[0].length;

    visited[r][c] = true;
    int cnt = 1;
    for(int i = 0; i < 4; i++) {
        int nr = r + dr[i];
        int nc = c + dc[i];
        if(nr < 0 || nr >= rows || nc < 0 || nc >= cols) continue;
        if(visited[nr][nc] == false && grid[nr][nc] == 1) {
            cnt += dfs(nr, nc, grid, visited);
        }
    }
    return cnt;
}

public int maxAreaOfIsland(int[][] grid) {
    int n = grid.length;
    int m = grid[0].length;
    boolean[][] visited = new boolean[n][m];
    int maxArea = 0;

    for(int i = 0; i < n; i++) {
        for(int j = 0; j < m; j++) {
            if(grid[i][j] == 1 && visited[i][j] == false) {
                int area = dfs(i, j, grid, visited);
                maxArea = Math.max(maxArea, area);
            }
        }
    }
    return maxArea;
}
```

Q28 : Nearest 0 from every cell
(https://leetcode.com/problems/01-matrix/description/)

```java
public int[][] updateMatrix(int[][] grid)
{
    int n = grid.length;
    int m = grid[0].length;
    boolean[][] visited = new boolean[n][m];
    Queue<int[]> q = new LinkedList<>();
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < m; j++) {
            if(grid[i][j] == 0) {
                q.add(new int[]{i, j});
                visited[i][j] = true;
            }
        }
    }

    int level = 0;
    while(q.size() > 0) {
        int size = q.size();
        for(int i = 0; i < size; i++) {
            int[] temp = q.remove();
            int sr = temp[0];
            int sc = temp[1];
            grid[sr][sc] = level;
            for(int j = 0; j < 4; j++) {
                int nr = sr + dr[j];
                int nc = sc + dc[j];
                if(nr < 0 || nr >= n || nc < 0 || nc >= m) continue;
                if(visited[nr][nc] == false) {
                    q.add(new int[]{nr, nc});
                    visited[nr][nc] = true;
                }
            }
        }
        level++;
    }
    return grid;
}
```

Jan 19th [Class Notes - Graphs and PQ (Jan 19).pdf]

Q29 : Rotten Oranges
(https://course.acciojob.com/idle?question=b21cba45-2a97-4492-82f7-5e23ed20ac00)

```
public static int orangesRotting(int[][] grid) {
    int rows = grid.length;
    int cols = grid[0].length;
    Queue<int[]> q = new LinkedList<>();
    int fresh = 0;

    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < cols; j++) {
            if(grid[i][j] == 2) q.add(new int[]{i, j});
            else if(grid[i][j] == 1) fresh++;
        }
    }

    int level = 0;
    while(q.size() > 0) {
        int size = q.size();
        for(int i = 0; i < size; i++) {
            int[] temp = q.remove();
            int r = temp[0];
            int c = temp[1];
            for(int j = 0; j < 4; j++) {
                int nr = r + dr[j];
                int nc = c + dc[j];
                if(nr < 0 || nr >= rows || nc < 0 || nc >= cols) continue;
                if(grid[nr][nc] == 1) {
                    grid[nr][nc] = 2;
                    fresh--;
                    q.add(new int[]{nr, nc});
                }
            }
        }
        level++;
    }
    if(fresh == 0) return level - 1;
    return -1;
}
```

Q30 : Jump Game IV
(https://leetcode.com/problems/jump-game-iv/description/)

```
public int minJumps(int[] arr) {
    HashMap<Integer, List<Integer>> hm = new HashMap<>();
    int n = arr.length;
    for(int i = 0; i < n; i++) {
        if(hm.containsKey(arr[i])) {
            hm.get(arr[i]).add(i);
        }
        else {
            List<Integer> newlist = new ArrayList<>();
            newlist.add(i);
            hm.put(arr[i], newlist);
        }
    }

    boolean[] visited = new boolean[n];
    Queue<Integer> q = new LinkedList<>();
    q.add(0);
    visited[0] = true;
    int level = 0;

    while(q.size() > 0) {
        int size = q.size();
        for(int i = 0; i < size; i++) {
            int idx = q.remove();
            if(idx == n - 1) return level;
            if(idx + 1 < n && visited[idx + 1] == false) {
                q.add(idx + 1);
                visited[idx + 1] = true;
            }
            if(idx - 1 >= 0 && visited[idx - 1] == false) {
                q.add(idx - 1);
                visited[idx - 1] = true;
            }
```

```java
            for(int new_idx : hm.get(arr[idx])) {
                if(visited[new_idx] == false) {
                    q.add(new_idx);
                    visited[new_idx] = true;
                }
            }
            hm.get(arr[idx]).clear();
        }
        level++;
    }
    return -1;
}
```

TC : O(N) [Every ele visited once]
SC : O(N) [Visited array and hashmap]

Q31 : Kth Largest and Smallest
(https://course.acciojob.com/idle?question=432f7ce4-a080-40bc-a2da-b7b2bbea21a0)

```java
public static void kSmallLarge( int arr [], int n, int k) {
    PriorityQueue<Integer> minpq = new PriorityQueue<>();
    PriorityQueue<Integer> maxpq = new PriorityQueue<>(Collections.reverseOrder());

    for(int i = 0; i < n; i++) {
        minpq.add(arr[i]);
        maxpq.add(arr[i]);
        if(minpq.size() > k) minpq.remove();
        if(maxpq.size() > k) maxpq.remove();
    }

    int kthSmall = maxpq.peek();
    int kthLarge = minpq.peek();
    System.out.println(kthSmall);
    System.out.println(kthLarge);
}
```

TC : O(NLogK)
SC : O(K)

Q32 : Min cost connect ropes
(https://course.acciojob.com/idle?question=bb7d3ad1-d2b7-4f65-8268-94e1d64a2ef3)

```
long minCost(long arr[], int n)
{
    PriorityQueue<Long> minpq = new PriorityQueue<>();
    for(int i = 0; i < n; i++)
        minpq.add(arr[i]);

    long cost = 0;
    while(minpq.size() > 1) {
        long r1 = minpq.remove();
        long r2 = minpq.remove();
        cost += r1 + r2;
        minpq.add(r1 + r2);
    }
    return cost;
}
```

TC : O(NLogN)
SC : O(N)

Q33 : Jump game II (https://leetcode.com/problems/jump-game-ii/description/)
(https://course.acciojob.com/idle?question=e1239682-f07f-48e1-8a2f-0973a958f443)

```java
public int jump(int[] arr)
{
    int n = arr.length;
    boolean[] visited = new boolean[n];
    Queue<Integer> q = new LinkedList<>();
    q.add(0);
    visited[0] = true;

    int level = 0;
    while(q.size() > 0)
    {
        int size = q.size();
        for(int i = 0; i < size; i++)
        {
            int idx = q.remove();
            if(idx == n - 1) return level;
            for(int jump = 1; jump <= arr[idx]; jump++)
            {
                if(idx + jump >= n) break;
                if(visited[idx + jump] == false) {
                    q.add(idx + jump);
                    visited[idx + jump] = true;
                }
            }
        }
        level++;
    }
    return -1;
}
```

TC : O(N * maxele(arr)) [due to jump loop]
SC : O(N)

Jan 20th [Class Notes - Graphs (Jan 20).pdf]

Q34 : Implementing Dijkstra
(https://practice.geeksforgeeks.org/problems/implementing-dijkstra-set-1-adjacency-matrix/1)

```
static int[] dijkstra(int vertices, ArrayList<ArrayList<ArrayList<Integer>>> graph,
int src) {
    // (dist, node)
    PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> Integer.compare(a[0],
b[0]));

    int[] distance = new int[vertices];
    Arrays.fill(distance, Integer.MAX_VALUE);
    distance[src] = 0;
    pq.add(new int[]{distance[src], src});

    while(pq.size() > 0) {
        // removing min distance node
        int[] temp = pq.remove();
        int dist_node = temp[0];
        int node = temp[1];

        // the distance in array is lesser than distance in your pq, ignore it
        if(distance[node] < dist_node) continue;

        for(ArrayList<Integer> edge : graph.get(node)) {
            int nbr = edge.get(0);
            int wt = edge.get(1);
            // previously attained distance to nbr is greater
            // than current incoming distance, update and add to pq
            if(distance[nbr] > distance[node] + wt) {
                distance[nbr] = distance[node] + wt;
                pq.add(new int[]{distance[nbr], nbr});
            }
        }
    }
    return distance;
}
```

TC : ((V + E) * LogV)
SC : O(V)

Q35 : Topological sort
(https://leetcode.com/problems/course-schedule-ii/description/)

```
private void complete(int src, ArrayList<Integer>[] graph, boolean[] visited,
ArrayList<Integer> order) {
    visited[src] = true;
    // your nbrs are your prereq complete them first
    for(int nbr : graph[src]) {
        if(visited[nbr] == false)
            complete(nbr, graph, visited, order);
    }
    order.add(src); // your prereq are completed, now you can do this course
}

public int[] findOrder(int numCourses, int[][] prerequisites) {

    ArrayList<Integer>[] graph = new ArrayList[numCourses];
    for(int i = 0; i < numCourses; i++) graph[i] = new ArrayList<>();

    for(int[] edge : prerequisites) {
        int u = edge[0];
        int v = edge[1];
        // graph[v].add(u); // u is a dependent of v (v -> u)
        graph[u].add(v); // v is prereq of u (u -> v)
    }

    boolean cycle = isCyclic(numCourses, graph);
    if(cycle == true) return new int[]{};

    boolean[] visited = new boolean[numCourses];
    ArrayList<Integer> order = new ArrayList<>();
    for(int i = 0; i < numCourses; i++) {
        if(visited[i] == false)
            complete(i, graph, visited, order);
    }

    int ans[] = new int[numCourses];
    for(int i = 0; i < numCourses; i++) ans[i] = order.get(i);
    return ans;
}
```

Using Khan’s BFS

```java
public int[] findOrder(int numCourses, int[][] prerequisites) {

    ArrayList<Integer>[] graph = new ArrayList[numCourses];
    int[] indegree = new int[numCourses];
    for(int i = 0; i < numCourses; i++) {
        graph[i] = new ArrayList<>();
    }

    for(int[] edge : prerequisites) {
        int u = edge[0];
        int v = edge[1];
        graph[v].add(u); // u is a dependent of v (v -> u)
        indegree[u]++;
    }

    Queue<Integer> q = new LinkedList<>();
    for(int i = 0; i < numCourses; i++) {
        if(indegree[i] == 0) q.add(i);
    }

    int[] topo = new int[numCourses];
    int idx = 0;
    while(q.size() > 0) {
        int size = q.size();
        for(int i = 0; i < size; i++) {
            int src = q.remove();
            topo[idx] = src;
            idx++;
            for(int nbr : graph[src]) {
                indegree[nbr]--;
                if(indegree[nbr] == 0) q.add(nbr);
            }
        }
    }

    if(idx == numCourses) return topo;
    return new int[]{}; // this means it has a cycle
}
```

Q36 : Number of distinct Islands
(https://practice.geeksforgeeks.org/problems/number-of-distinct-islands/1)

```
String island;
int[] dr = {-1, 0, 1, 0};
int[] dc = {0, 1, 0, -1};
String[] val = {"T", "R", "D", "L"};

private void dfs(int r, int c, boolean[][] visited, int[][] grid) {
    int rows = grid.length;
    int cols = grid[0].length;

    visited[r][c] = true;
    for(int i = 0; i < 4; i++) {
        int nr = r + dr[i];
        int nc = c + dc[i];
        if(nr < 0 || nr >= rows || nc < 0 || nc >= cols) continue;
        if(visited[nr][nc] == false && grid[nr][nc] == 1) {
            island += val[i];
            dfs(nr, nc, visited, grid);
        }
    }
    island += "X";
}
int countDistinctIslands(int[][] grid) {
    int rows = grid.length;
    int cols = grid[0].length;
    boolean[][] visited = new boolean[rows][cols];
    HashMap<String, Integer> hm = new HashMap<>();
    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < cols; j++) {
            if(visited[i][j] == false && grid[i][j] == 1) {
                island = "";
                dfs(i, j, visited, grid);
                hm.put(island, hm.getOrDefault(island, 0) + 1);
             }
        }
    }
    return hm.size();
}
```

Jan 22nd [Class Notes - Contest Discussion (Jan 22).pdf]

Q37 : Alice and capital city
(https://course.acciojob.com/idle?question=f941e196-c628-446e-bce3-9ec899602d2c)

```java
public int findCenter(int n,int[][] edges) {
    int[] cnt = new int[n + 1];

    for(int[] edge : edges) {
        int u = edge[0];
        int v = edge[1];
        cnt[u]++;
        cnt[v]++;

        if(cnt[u] == n - 1) return u;
        if(cnt[v] == n - 1) return v;
    }
    return -1;
}
```

Q38 : 3 sum equal to 0
(https://course.acciojob.com/idle?question=5bc00c88-0d0b-441b-aebd-302ebec93eb5)

```java
public boolean containsTriplet(int[] arr) {
    Arrays.sort(arr);
    int n = arr.length;
    for(int i = 0; i < n; i++) {
        int start =  i + 1;
        int end = n - 1;
        while(start < end) {
            int triplet = arr[i] + arr[start] + arr[end];
            if(triplet == 0) return true;
            else if(triplet > 0) end--;
            else start++;
        }
    }
    return false;
}
```

TC : O(N^2)

Q39 : Subarray with exactly K distinct integers
(https://course.acciojob.com/idle?question=2a52d218-f7ff-4261-af9d-5d2f4ae556a8)
(https://leetcode.com/problems/subarrays-with-k-different-integers/)

```
private int atmost(int arr[], int n, int k) {
    int start = 0;
    int end = 0;
    int ans = 0;
    int distinct = 0;
    HashMap<Integer, Integer> hm = new HashMap<>();

    while(end < n) {
        hm.put(arr[end], hm.getOrDefault(arr[end], 0) + 1);
        if(hm.get(arr[end]) == 1) distinct++;
        end++;

        while(start < end && distinct > k) {
            if(hm.get(arr[start]) == 1) distinct--;
            hm.put(arr[start], hm.get(arr[start]) - 1);
            start++;
        }

        if(distinct <= k)
            ans += (end - start);
    }

    return ans;
}

public int subarraysWithKDistinct(int[] nums, int k) {
    int n = nums.length;
    return atmost(nums, n, k) - atmost(nums, n, k - 1);
}
```

TC : O(N)
SC : O(N)

Jan 23rd [Class Notes - Bit Manipulation (Jan 23).pdf]

Q40 : Check Kth is set or not
(https://course.acciojob.com/idle?question=90266d47-b043-4a66-b6c8-57d12147a33d)

Using left shift

```
public static Boolean solve(int n,int k) {
    int bitMask = (1 << k);
    int ans = n & bitMask;
    if(ans == 0) return false;
    return true;
}
```

Using right shift

```
public static Boolean solve(int n,int k) {
    int bitMask = (n >> k);
    int ans = bitMask & 1;
    if(ans == 0) return false;
    return true;
}
```

Q41 : Count set bits
(https://course.acciojob.com/idle?question=99b2b8d9-d830-4b9a-a61d-25dc5bc94ab4)

Method 1 :

```
private boolean checkBit(int n, int i) {
    return (n & (1 << i)) != 0;
}

public int setBits(int n) {
    int cnt = 0;
    for(int i = 0; i <= 31; i++) {
        boolean ans = checkBit(n, i);
        if(ans) cnt++;
    }
    return cnt;
}
```

Method 2 :

```
public int setBits(int n) {
    int cnt = 0;
    while(n != 0) {
        if((n & 1) == 1) cnt++;
        n = n >> 1;
    }
    return cnt;
}
```

Q42 : Check 2 power
(https://course.acciojob.com/idle?question=6c89e821-0fac-4811-847e-05d0cd5b3733)

Method 1 :

```
public static int setBits(int n) {
    int cnt = 0;
    while(n != 0) {
        if((n & 1) == 1) cnt++;
        n = n >> 1;
    }
    return cnt;
}

public static boolean isPowerOfTwo(int n) {
    return (setBits(n) == 1);
}
```

Method 2 :

```
public static boolean isPowerOfTwo(int n) {
    return (n & (n - 1)) == 0;
}
```

Jan 24th [Class Notes - Bit Manipulation (Jan 24).pdf]

Q43 : Set kth bit, Unset kth bit, toggle kth bit
(https://practice.geeksforgeeks.org/problems/set-kth-bit3724/1)

```
int setBit(int n, int k) {
    int bitMask = (1 << k);
    return (n | bitMask)
}

int clearBit(int n, int k) {
    int bitMask = ~(1 << k);
    return (n & bitMask);
}

int toggleBit(int n, int k) {
    int bitMask = ~(1 << k);
    return (n ^ bitMask);
}
```

Q44 : Game of XOR
(https://course.acciojob.com/idle?question=1605b6cb-6b60-4bf7-ac4a-8da7cf4bcf1e)

```
static int xorSubarrayXors(int arr[], int n) {
    int ans = 0;
    for(int i = 0; i < n; i++) {
        int freq = (i + 1) * (n - i);
        if((freq % 2) == 1) ans = ans ^ arr[i];
    }
    return ans;
}
```

Q45 : Flip bits
(https://course.acciojob.com/idle?question=fc94cc5e-7796-4534-9ada-abc2b7ed2938)

Method 1 : Using Mask

```
public long flipBits(long n) {
    long bitMask = (1L << 32) - 1;
    return (n ^ bitMask);
}
```

Method 2 : Iterating over all bits

```java
public long flipBits(long n) {
    long ans = 0;
    for(int i = 0; i <= 31; i++) {
        long check = n & (1L << i);
        if(check == 0) ans += (1L << i);
    }
    return ans;
}
```

Q46 : Reverse bits
(https://course.acciojob.com/idle?question=f2de87a9-3254-4fad-a157-e5194ac90f1c)

```java
public static long reverse(long n) {
    long ans = 0;
    for(int i = 0; i <= 31; i++) {
        long check = n & (1L << i);
        if(check != 0) ans += (1L << (31 - i));
    }
    return ans;
}
```

Q47 : Generate Subsets
(https://course.acciojob.com/idle?question=52a234c6-6674-456c-8671-51a3d15ec96d)

```java
public static ArrayList<ArrayList<Integer>> subsets(int[] arr, int n) {
    ArrayList<ArrayList<Integer>> ans = new ArrayList<>();
    int total = (1 << n);
    for(int num = 0; num < total; num++) {
        ArrayList<Integer> subset = new ArrayList<>();
        for(int idx = 0; idx < n; idx++) {
            int bitpos = (n - idx - 1);
            int check = num & (1 << bitpos);
            if(check != 0) subset.add(arr[idx]);
        }
        ans.add(subset);
    }
    return ans;
}
```

Q48 : Single Number
(https://leetcode.com/problems/single-number/)

```java
public int singleNumber(int[] nums) {
    int n = nums.length;
    int ans = 0;
    for(int i = 0; i < n; i++) {
        ans = ans ^ nums[i];
    }
    return ans;
}
```

Q49 : Single Number II
(https://leetcode.com/problems/single-number-ii/description/)

```java
public int singleNumber(int[] nums) {
    int n = nums.length;
    int ans = 0;
    for(int bitpos = 0; bitpos <= 31; bitpos++) {
        int cnt = 0;
        for(int i = 0; i < n; i++) {
            if((nums[i] & (1 << bitpos)) != 0) cnt++;
        }
        if(cnt % 3 == 1) ans += (1 << bitpos);
    }
    return ans;
}
```

Q50 : Single Number III
(https://leetcode.com/problems/single-number-iii/description/)

```
private boolean checkKthBit(int n, int k) {
    int bitmask = (1 << k);
    int check = (n & bitmask);
    return (check != 0);
}

private int rightMostSetBit(int n) {
    for(int i = 0; i <= 31; i++) {
        if(checkKthBit(n, i)) return i;
    }
    return -1;
}

public int[] singleNumber(int[] nums) {
    int n = nums.length;
    int xor = 0;
    for(int i = 0; i < n; i++)
        xor = xor ^ nums[i];

    int bitpos = rightMostSetBit(xor);
    int[] ans = new int[]{0, 0};
    // ans[0] -> g1(0's), ans[1] -> g2(1's)

    for(int i = 0; i < n; i++) {
        if(checkKthBit(nums[i], bitpos)) ans[1] = ans[1] ^ nums[i];
        else ans[0] = ans[0] ^ nums[i];
    }

    return ans;
}
```

Jan 30th [Class Notes - Prefix matrix (Jan 30).pdf]

Q51 : Range Addition
(https://www.lintcode.com/problem/903/)

```
public int[] getModifiedArray(int length, int[][] updates) {
    int[] ans = new int[length];
    for(int[] query : updates) {
        int l = query[0];
        int r = query[1];
        int x = query[2];
        ans[l] += x;
        if(r + 1 < length) ans[r + 1] -= x;
    }

    for(int i = 1; i < length; i++)
        ans[i] += ans[i - 1];

    return ans;
}
```

TC : O(q + n)
SC : O(n)

Q52 : Range Sum Query 2D - Immutable
(https://leetcode.com/problems/range-sum-query-2d-immutable/)

```
class NumMatrix {
    int[][] prefix2D;

    public NumMatrix(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;

        prefix2D = new int[rows][cols];
        // 1. colprefix on mat
        for(int c = 0; c < cols; c++) {
            int csum = 0;
            for(int r = 0; r < rows; r++) {
                csum += matrix[r][c];
                prefix2D[r][c] = csum;
            }
        }
        // 2. rowprefix on prefix2D
        for(int r = 0; r < rows; r++) {
            int csum = 0;
            for(int c = 0; c < cols; c++) {
                csum += prefix2D[r][c];
                prefix2D[r][c] = csum;
            }
        }
    }

    public int sumRegion(int row1, int col1, int row2, int col2) {
        int bigger = prefix2D[row2][col2];
        int left = (col1 - 1 >= 0) ? prefix2D[row2][col1 - 1] : 0;
        int upper = (row1 - 1 >= 0) ? prefix2D[row1 - 1][col2] : 0;
        int intersect = (row1 - 1 >= 0 && col1 - 1 >= 0) ? prefix2D[row1 - 1][col1 -
1] : 0;
        return bigger - left - upper + intersect;
    }
}
```

TC : O(q + (rows * cols))
SC : O(rows * cols)

Q53 : Maximum submatrix sum
(https://practice.geeksforgeeks.org/problems/maximum-sum-rectangle2948/1)

```
private int kadanes(int [] arr) {
    int n = arr.length;
    int csum = 0;
    int gsum = Integer.MIN_VALUE;

    for(int i = 0; i < n; i++) {
        csum = Math.max(arr[i], csum + arr[i]);
        gsum = Math.max(gsum, csum);
    }

    return gsum;
}

int maximumSumRectangle(int rows, int cols, int matrix[][]) {

    int[] colPrefix = new int[cols];
    int ans = Integer.MIN_VALUE;

    for(int slab = 0; slab < rows; slab++)
    {
        Arrays.fill(colPrefix, 0);
        for(int moving = slab; moving < rows; moving++)
        {
            for(int c = 0; c < cols; c++)
            {
                colPrefix[c] += matrix[moving][c];
            }
            ans = Math.max(ans, kadanes(colPrefix));
        }
    }

    return ans;
}
```

TC : O(rows * rows * cols)
SC : O(cols)

Q54 : Number of submatrices with sum = k
(https://leetcode.com/problems/number-of-submatrices-that-sum-to-target/)

```java
private int numSubarraySumTarget(int[] arr, int target) {
    int n = arr.length;
    HashMap<Integer, Integer> hm = new HashMap<>();
    int cnt = 0;
    int sum = 0;
    hm.put(0, 1);

    for(int i = 0; i < n; i++) {
        sum += arr[i];
        if(hm.containsKey(sum - target)) cnt += hm.get(sum - target);
        hm.put(sum, hm.getOrDefault(sum, 0) + 1);
    }
    return cnt;
}

public int numSubmatrixSumTarget(int[][] matrix, int target) {
    int rows = matrix.length;
    int cols = matrix[0].length;
    int[] colPrefix = new int[cols];
    int ans = 0;

    for(int slab = 0; slab < rows; slab++) {
        Arrays.fill(colPrefix, 0);
        for(int moving = slab; moving < rows; moving++) {
            for(int c = 0; c < cols; c++) {
                colPrefix[c] += matrix[moving][c];
            }
            ans += numSubarraySumTarget(colPrefix, target);
        }
    }

    return ans;
}
```

TC : O(rows * rows * cols)
SC : O(cols)

Q55: Maximum Sub-String after at most K changes
(Maximum Sub-String after at most K changes | Practice | GeeksforGeeks)

```java
private int solve(String s, int k, char ch) {
    int start = 0;
    int end = 0;
    int ans = -1;
    int n = s.length();
    int replace = 0;

    while(end < n) {
        if(s.charAt(end) != ch) replace++;
        end++;
        while(start < end && replace > k) {
            if(s.charAt(start) != ch) replace--;
            start++;
        }
        ans = Math.max(ans, end - start);
    }

    return ans;
}

public int characterReplacement(String s, int k) {
    int ans = -1;
    for(char ch = 'A'; ch <= 'Z'; ch++) {
        ans = Math.max(ans, solve(s, k, ch));
    }
    return ans;
}
```

TC : O(26 * n)
SC : O(1)

Jan 31st [Class Notes - Revision (Jan 31).pdf]