Juspay Questions Round 1

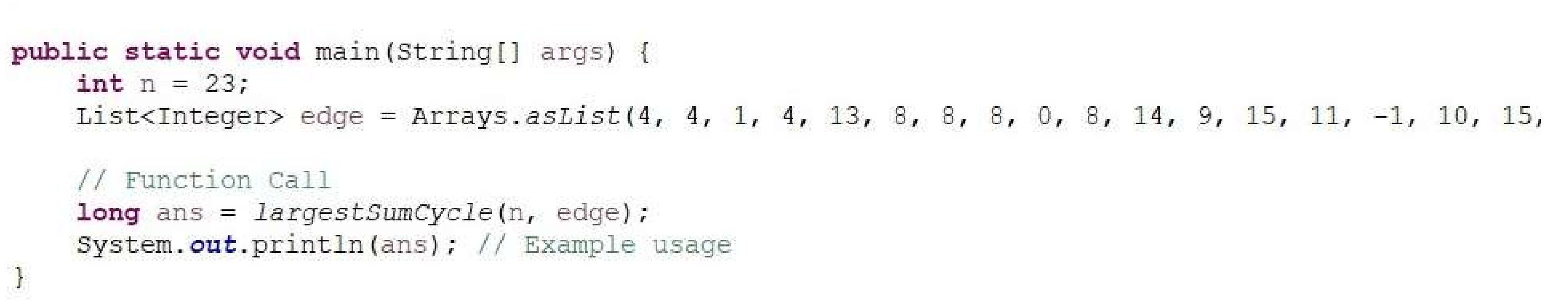
05 July 2024 15:17

Nearest Meeting cell

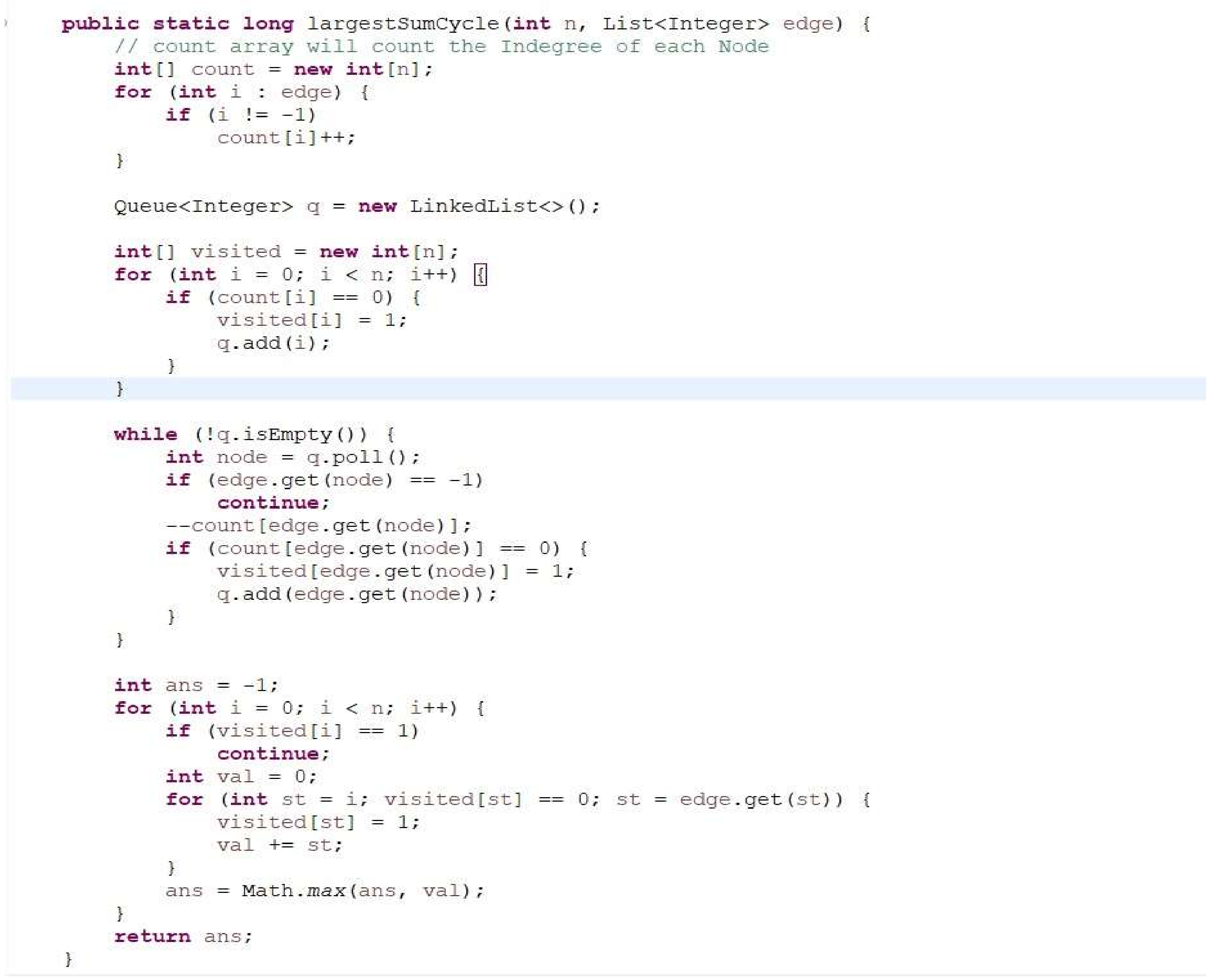
```java
private static void dijkstra(int start, List<List<Integer>> graph, long[] distances){
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    pq.offer(start);
    distances[start]=0;
    while(!pq.isEmpty()){
        int curr = pq.poll();
        for(int neighbor : graph.get(curr)){
            long distance = distances[curr]+1; //all edges have same weight 1
            if(distance<distances[neighbor]){
                distances[neighbor]=distance;
                pq.offer(neighbor);
            }
        }
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int n = scanner.nextInt();
    int[] edges = new int[n];
    for(int i=0;i<n;i++){
        edges[i] = scanner.nextInt();
    }
    int C1 = scanner.nextInt();
    int C2 = scanner.nextInt();
    System.out.println(minimumWeight(n,edges,C1,C2));
}
```

```java
public static int minimumWeight(int n, int[] edges, int C1, int C2) {
    //Create directed graph from the array given in input
    List<List<Integer>> graph = new ArrayList<>();
    for(int i=0;i<n;i++){
        graph.add(new ArrayList<Integer>());
    }
    for(int i=0;i<n;i++){
        if(edges[i]!=-1){
            graph.get(i).add(edges[i]);
        }
    }
    //Create two arrays A and B for storing min distance from C1 and C2
    long[] A = new long[n];
    long[] B = new long[n];
    Arrays.fill(A,Long.MAX_VALUE);
    Arrays.fill(B,Long.MAX_VALUE);
    //Part 1 and Part 2 of Algo -> Implement a dijkstra function and call it for both arrays A and B
    dijkstra(C1,graph,A);
    dijkstra(C2,graph,B);
    //Now comes Part 3 part of algo-> loop through and get node with min(A[i]+B[i])
    int node=0;
    long dist=Long.MAX_VALUE;
    for(int i=0;i<n;i++){
        //if node is not accessible from any of them ignore it
        if(A[i]==Long.MAX_VALUE || B[i]==Long.MAX_VALUE) continue;
        // sauravhathi
        if(dist>A[i]+B[i]){
            dist= A[i]+B[i];
            node=i;
        }
    }
    if(dist==Long.MAX_VALUE) return -1; //if no meeting point is found
    return node;
```

Largest Sum Cycle

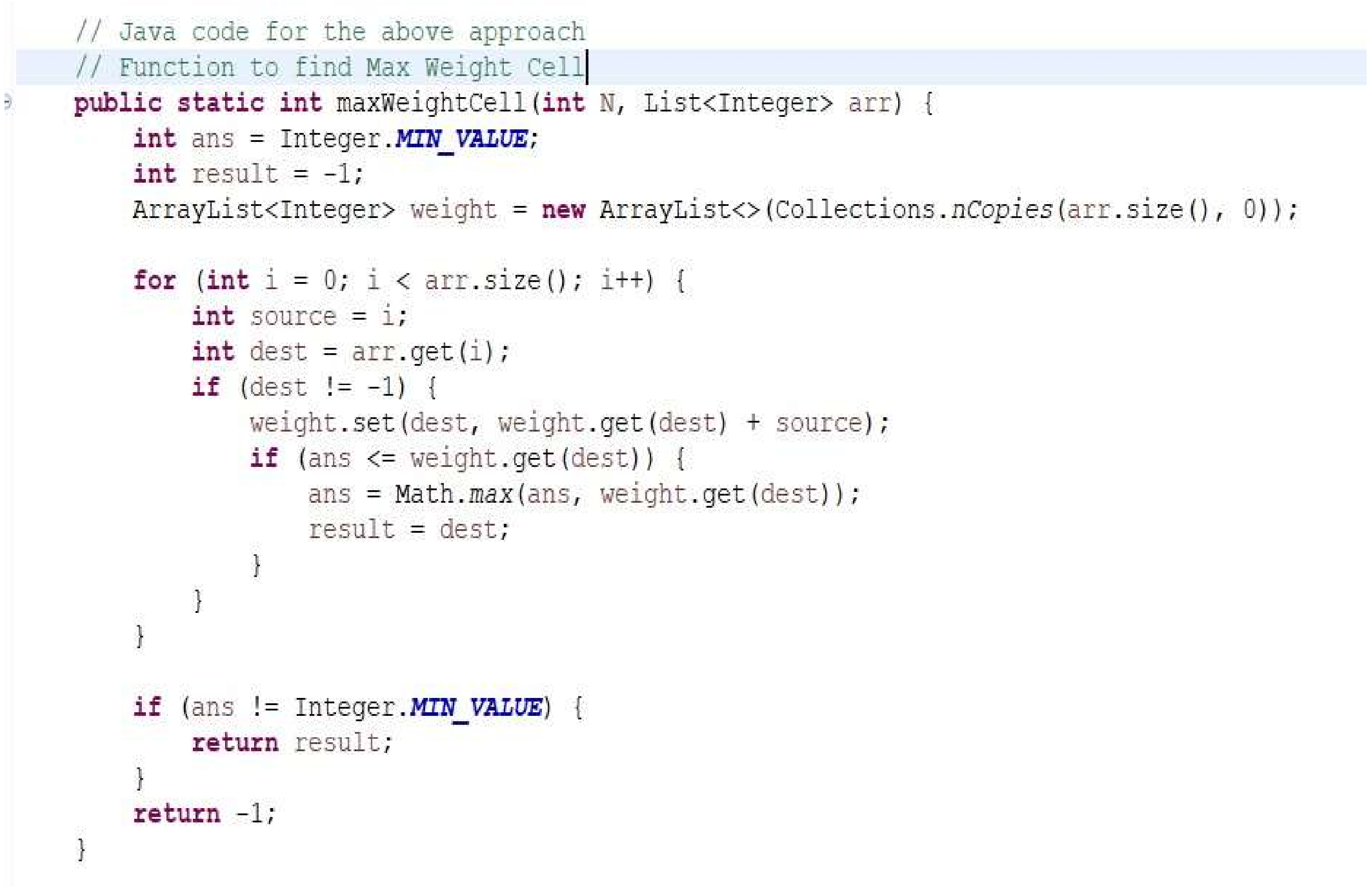


```java
public static void main(String[] args) {
    int n = 23;
    List<Integer> edge = Arrays.asList(4, 4, 1, 4, 13, 8, 8, 8, 0, 8, 14, 9, 15, 11, -1, 10, 15,

    // Function Call
    long ans = largestSumCycle(n, edge);
    System.out.println(ans); // Example usage
}
```



```java
public static long largestSumCycle(int n, List<Integer> edge) {
    // count array will count the Indegree of each Node
    int[] count = new int[n];
    for (int i : edge) {
        if (i != -1)
            count[i]++;
    }

    Queue<Integer> q = new LinkedList<>();

    int[] visited = new int[n];
    for (int i = 0; i < n; i++) {
        if (count[i] == 0) {
            visited[i] = 1;
            q.add(i);
        }
    }

    while (!q.isEmpty()) {
        int node = q.poll();
        if (edge.get(node) == -1)
            continue;
        --count[edge.get(node)];
        if (count[edge.get(node)] == 0) {
            visited[edge.get(node)] = 1;
            q.add(edge.get(node));
        }
    }

    int ans = -1;
    for (int i = 0; i < n; i++) {
        if (visited[i] == 1)
            continue;
        int val = 0;
        for (int st = i; visited[st] == 0; st = edge.get(st)) {
            visited[st] = 1;
            val += st;
        }
        ans = Math.max(ans, val);
    }
    return ans;
}
```

Maximum Weight Cell



```java
// Java code for the above approach
// Function to find Max Weight Cell
public static int maxWeightCell(int N, List<Integer> arr) {
    int ans = Integer.MIN_VALUE;
    int result = -1;
    ArrayList<Integer> weight = new ArrayList<>(Collections.nCopies(arr.size(), 0));

    for (int i = 0; i < arr.size(); i++) {
        int source = i;
        int dest = arr.get(i);
        if (dest != -1) {
            weight.set(dest, weight.get(dest) + source);
            if (ans <= weight.get(dest)) {
                ans = Math.max(ans, weight.get(dest));
                result = dest;
            }
        }
    }

    if (ans != Integer.MIN_VALUE) {
        return result;
    }
    return -1;
}
```