

```
1 package wiproassignment;
2 import java.util.List;
3 import java.util.ArrayList;
4 class Graph {
5     private int V;
6     private List<Integer>[] adj;
7
8     Graph(int V) {
9         this.V = V;
10        adj = new ArrayList[V];
11        for (int i = 0; i < V; ++i)
12            adj[i] = new ArrayList<>();
13    }
14
15    void addEdge(int u, int v) {
16        adj[u].add(v);
17    }
18
19    boolean isCyclicUtil(int v, boolean[] visited, boolean[] recStack) {
20        if (!visited[v]) {
21            visited[v] = true;
22            recStack[v] = true;
23
24            for (int neighbor : adj[v]) {
25                if (!visited[neighbor] && isCyclicUtil(neighbor, visited, recStack))
26                    return true;
27                else if (recStack[neighbor])
28
29                    return true;
30            }
31            recStack[v] = false;
32            return false;
33        }
34
35        boolean isCyclic() {
36            boolean[] visited = new boolean[V];
37            boolean[] recStack = new boolean[V];
38
39            for (int i = 0; i < V; i++)
40                if (isCyclicUtil(i, visited, recStack))
41                    return true;
42
43            return false;
44        }
45
46        boolean addEdgeAndCheckCycle(int u, int v) {
47            addEdge(u, v);
48            return !isCyclic();
49        }
50    }
51}
```

```

wipro - wiproAssignment/src/wiproassignment/GraphAddEdge.java - Eclipse IDE
File Edit Source Refactor Navigate Project Run Window Help
Package Explorer X KMPAlg.java LinkedListM... QueueSorter... StackSorter... RemoveDuplicat... SequenceInS... MergeSorted... CircularQue... GraphAddEdge...
DataStructures
  IRE System Library [JavaSE-17]
    src
      com.wipro.assignments
        MiddleList.java
        com.wipro.ds
        com.wipro.graphalg
        com.wipro.graphs
        com.wipro.linear
        com.wipro.nonlinear
        com.wipro.pattern
        com.wipro.search
        com.wipro.sort
  IRE System Library [jdk1.8.0_202]
    src
      com.wiproassignment
        CircularQueueBinarySearch.java
        GraphAddEdge.java
        LinkedListMiddle.java
        MergeSortedList.java
        QueueSorter.java
        RemoveDuplicate.java
        SequenceInStack.java
        StackSorter.java
  module-info.java
Console X <terminated> GraphAddEdge [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (01-Jun-2024, 2:27:48 pm - 2:27:49 pm) [pid: 28772]
Adding edge 3 -> 0 creates a cycle? false

```

36°C Mostly cloudy 14:28 01-06-2024

Task 5: Breadth-First Search (BFS) Implementation

For a given undirected graph, implement BFS to traverse the graph starting from a given node and print each node in the order it is visited.

```

wipro - wiproAssignment/src/wiproassignment/BFS.java - Eclipse IDE
File Edit Source Refactor Navigate Project Run Window Help
Package Explorer X LinkedListM... QueueSorter... StackSorter... RemoveDuplicat... SequenceInS... MergeSorted... CircularQue... BFS.java X
DataStructures
  IRE System Library [JavaSE-17]
    src
      com.wipro.assignments
        MiddleList.java
        com.wipro.ds
        com.wipro.graphalg
        com.wipro.graphs
        com.wipro.linear
        com.wipro.nonlinear
        com.wipro.pattern
        com.wipro.search
        com.wipro.sort
  IRE System Library [jdk1.8.0_202]
    src
      com.wiproassignment
        BFS.java
        CircularQueueBinarySearch.java
        GraphAddEdge.java
        LinkedListMiddle.java
        MergeSortedList.java
        QueueSorter.java
        RemoveDuplicate.java
        SequenceInStack.java
        StackSorter.java
  module-info.java
Console X <terminated> BFS [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (01-Jun-2024, 4:22:48 pm - 4:22:53 pm) [pid: 28772]

```

36°C Mostly cloudy 14:39 01-06-2024

The screenshot shows the Eclipse IDE interface with the file `BFS.java` open in the editor. The code implements Breadth-First Search (BFS) on an undirected graph represented by an adjacency list. The `main` method creates a `BFSGraph` object, adds edges between nodes 0, 1, 2, and 3, and then performs a BFS starting from node 0, printing the visited nodes in the order they are reached.

```
29     visited.add(start);
30     queue.add(start);
31
32     while (!queue.isEmpty()) {
33         int node = queue.poll();
34         System.out.print(node + " ");
35
36         for (int neighbor : adjList.get(node)) {
37             if (!visited.contains(neighbor)) {
38                 visited.add(neighbor);
39                 queue.add(neighbor);
40             }
41         }
42     }
43 }
44 public static void main(String[] args) {
45     BFS BFSGraph = new BFS();
46     BFSGraph.addEdge(0, 1);
47     BFSGraph.addEdge(0, 2);
48     BFSGraph.addEdge(1, 2);
49     BFSGraph.addEdge(2, 3);
50
51     System.out.println("BFS starting from node 0:");
52     BFSGraph.bfs(0);
53 }
54 }
```

The screenshot shows the Eclipse IDE interface with the file `BFS.java` open in the editor. The code is identical to the one in the previous screenshot. Below the editor, the `Console` tab is active, displaying the output of the `bfs` method. The output shows the message "BFS starting from node 0:" followed by the sequence of visited nodes: 0 1 2 3.

```
35 |
36     for (int neighbor : adjList.get(node)) {
37         if (!visited.contains(neighbor)) {
38             visited.add(neighbor);
39             queue.add(neighbor);
40         }
41     }
42 }
43 }
44 public static void main(String[] args) {
45     BFS BFSGraph = new BFS();
46     BFSGraph.addEdge(0, 1);
47     BFSGraph.addEdge(0, 2);
48     BFSGraph.addEdge(1, 2);
49     BFSGraph.addEdge(2, 3);
50
51     System.out.println("BFS starting from node 0:");
52     BFSGraph.bfs(0);
53 }
54 }
55
```

Console output:

```
<terminated> BFS [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (01-Jun-2024, 2:40:30 pm – 2:40:31 pm) [pid: 24020]
BFS starting from node 0:
0 1 2 3
```

Task 6: Depth-First Search (DFS) Recursive

Write a recursive DFS function for a given undirected graph. The function should visit every node and print it out.

wipro - wiproAssignment/src/wiproassignment/DFS.java - Eclipse IDE

```

1 package wiproassignment;
2 import java.util.List;
3 import java.util.Map;
4 import java.util.ArrayList;
5 import java.util.HashMap;
6 import java.util.Set;
7 import java.util.HashSet;
8 public class DFS {
9     private final Map<Integer, List<Integer>> adjList = new HashMap<>();
10    public void addNode(int node) {
11        adjList.putIfAbsent(node, new ArrayList<>());
12    }
13    public void addEdge(int from, int to) {
14        addNode(from);
15        addNode(to);
16        adjList.get(from).add(to);
17        adjList.get(to).add(from);
18    }
19    public void dfs(int start) {
20        Set<Integer> visited = new HashSet<>();
21        dfsUtil(start, visited);
22    }
23    private void dfsUtil(int node, Set<Integer> visited) {
24        visited.add(node);
25        System.out.print(node + " ");
26        for (int neighbor : adjList.get(node)) {
27            if (!visited.contains(neighbor)) {
28                dfsUtil(neighbor, visited);
29            }
30        }
31    }
32    public static void main(String[] args) {
33        DFS DFSGraph = new DFS();
34        DFSGraph.addEdge(0, 3);
35        DFSGraph.addEdge(0, 5);
36        DFSGraph.addEdge(1, 2);
37        DFSGraph.addEdge(3, 5);
38        System.out.println("DFS starting from node 0:");
39        DFSGraph.dfs(0);
40    }
41 }
42 
```

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the package structure under 'wiproAssignment'. The 'src' folder contains several packages: DataStructures, wiproAssignment, and wiproAssignment. The 'DataStructures' package contains classes like QueueSorter, StackSorter, RemoveDuplicate, SequenceInStack, MergeSorted, CircularQueue, GraphAddEdge, BFS.java, and DFS.java.
- Code Editor:** The 'DFS.java' file is open, displaying Java code for Depth-First Search (DFS). The code includes imports for java.util.List, java.util.LinkedList, and com.wipro.ds. It defines a class 'DFS' with a static main method that creates a 'DFSGraph' object, adds edges between nodes 0, 3, 5, 1, and 2, and then performs a DFS starting from node 0, printing the result to the console.
- Terminal:** The 'Console' tab shows the output of the executed program: "DFS starting from node 0:" followed by the sequence of visited nodes: 0 3 5.
- System Tray:** Shows the date and time as 01-Jun-2024 2:48:55 pm, the temperature as 36°C, and the weather as mostly cloudy.

Task 1: Dijkstra's Shortest Path Finder

Code Dijkstra's algorithm to find the shortest path from a start node to every other node in a weighted graph with positive weights.

wipro - wiproAssignment/src/wiproassignment/Dijkstra.java - Eclipse IDE

```

1 package wiproassignment;
2
3 import java.util.*;
4
5 public class Dijkstra {
6     static class Node implements Comparable<Node> {
7         int vertex;
8         int weight;
9
10        Node(int vertex, int weight) {
11            this.vertex = vertex;
12            this.weight = weight;
13        }
14
15        public int compareTo(Node other) {
16            return Integer.compare(this.weight, other.weight);
17        }
18    }
19
20    static void dijkstra(int[][] graph, int start) {
21        int V = graph.length;
22        PriorityQueue<Node> pq = new PriorityQueue<>();
23        int[] dist = new int[V];
24        Arrays.fill(dist, Integer.MAX_VALUE);
25        boolean[] visited = new boolean[V];
26
27        dist[start] = 0;
28        pq.offer(new Node(start, 0));
29
30        while (!pq.isEmpty()) {
31            int u = pq.poll().vertex;
32            visited[u] = true;
33
34            for (int v = 0; v < V; v++) {
35                if (graph[u][v] != 0 && !visited[v] && dist[u] != Integer.MAX_VALUE && dist[v] > dist[u] + graph[u][v]) {
36                    dist[v] = dist[u] + graph[u][v];
37                    pq.offer(new Node(v, dist[v]));
38                }
39            }
40        }
41
42        // Print shortest distances from start node to every other node
43        System.out.println("Shortest distances from node " + start + ":");
44        for (int i = 0; i < V; i++) {
45            System.out.println("Node " + i + ": " + dist[i]);
46        }
47    }
48
49    public static void main(String[] args) {
50        int[][] graph = {
51            {0, 4, 0, 0, 0, 0, 0, 8, 0},
52            {4, 0, 8, 0, 0, 0, 0, 11, 0},
53            {0, 8, 0, 7, 0, 4, 0, 0, 2}
54        };
55    }
56}

```

CAD/INR +0.55%

wipro - wiproAssignment/src/wiproassignment/Dijkstra.java - Eclipse IDE

```

27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52

```

CAD/INR +0.55%

The screenshot shows two instances of the Eclipse IDE interface. Both instances have the same workspace structure:

- Project Explorer:** Contains packages like DataStructures, JRE System Library [JavaSE-17], and SRC.
- SRC Package:** Contains sub-packages com.wipro.assignments, com.wipro.ds, com.wipro.graphalg, and com.wipro.graphs, each with various Java files (e.g., Dijkstra.java, Graphs.java).
- Dijkstra.java Content:** The code implements Dijkstra's algorithm. It defines a graph as an int[11][11] matrix and calls the dijkstra() method.
- Console Output:** Shows the shortest distances from node 0 to all other nodes (1 to 8) in ascending order of distance.

The top instance shows the output for nodes 0 to 5, while the bottom instance shows the output for nodes 0 to 8.

```
int[][] graph = {
    {0, 4, 0, 0, 0, 0, 0, 8, 0},
    {4, 0, 8, 0, 0, 0, 0, 11, 0},
    {0, 8, 0, 7, 0, 4, 0, 0, 2},
    {0, 0, 7, 0, 9, 14, 0, 0, 0},
    {0, 0, 0, 9, 0, 10, 0, 0, 0},
    {0, 0, 4, 14, 10, 0, 2, 0, 0},
    {0, 0, 0, 0, 2, 0, 1, 6},
    {8, 11, 0, 0, 0, 1, 0, 7},
    {0, 0, 2, 0, 0, 0, 6, 7, 0}
};

int startNode = 0; // Start node for Dijkstra's algorithm

dijkstra(graph, startNode);
```

```
Shortest distances from node 0:
Node 0: 0
Node 1: 4
Node 2: 12
Node 3: 19
Node 4: 21
Node 5: 11
```

```
Shortest distances from node 0:
Node 0: 0
Node 1: 4
Node 2: 12
Node 3: 19
Node 4: 21
Node 5: 11
Node 6: 9
Node 7: 8
Node 8: 14
```

Task 2: Kruskal's Algorithm for MST

Implement Kruskal's algorithm to find the minimum spanning tree of a given connected, undirected graph with non-negative edge weights.

The image shows two side-by-side views of the Eclipse IDE interface, both displaying the same Java code for the Kruskal algorithm. The code is part of a package named `wiproassignment`.

```
1 package wiproassignment;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.List;
6
7 class DisjointSet {
8     int[] parent, rank;
9
10    public DisjointSet(int n) {
11        parent = new int[n];
12        rank = new int[n];
13        for (int i = 0; i < n; i++) {
14            parent[i] = i;
15            rank[i] = 0;
16        }
17    }
18
19    public int find(int u) {
20        if (parent[u] != u) {
21            parent[u] = find(parent[u]);
22        }
23        return parent[u];
24    }
25
26    public void union(int u, int v) {
27        int rootU = find(u);
28        int rootV = find(v);
29        if (rootU != rootV) {
30            if (rank[rootU] > rank[rootV]) {
31                parent[rootV] = rootU;
32            } else if (rank[rootU] < rank[rootV]) {
33                parent[rootU] = rootV;
34            } else {
35                parent[rootV] = rootU;
36                rank[rootU]++;
37            }
38        }
39    }
40
41    public class Kruskal {
42        static class Edge implements Comparable<Edge> {
43            int u, v, weight;
44
45            public Edge(int u, int v, int weight) {
46                this.u = u;
47                this.v = v;
48                this.weight = weight;
49            }
50        }
51    }
52}
```

The code defines a `DisjointSet` class with methods for finding the root of a set and performing union operations. It also contains a nested `Kruskal` class which holds a static inner class `Edge` that implements the `Comparable` interface.

The screenshot shows two identical instances of the Eclipse IDE running on a Windows operating system. Both instances are displaying the same Java code for the Kruskal algorithm, which is used to find a Minimum Spanning Tree (MST) in a weighted graph.

Kruskal.java Code:

```
public int compareTo(Edge other) {
    return this.weight - other.weight;
}

public static List<Edge> kruskal(int n, List<Edge> edges) {
    Collections.sort(edges);
    DisjointSet ds = new DisjointSet(n);
    List<Edge> mst = new ArrayList<>();
    int mstCost = 0;

    for (Edge edge : edges) {
        if (ds.find(edge.u) != ds.find(edge.v)) {
            ds.union(edge.u, edge.v);
            mst.add(edge);
            mstCost += edge.weight;
        }
    }

    System.out.println("Total cost of MST: " + mstCost);
    return mst;
}

public static void main(String[] args) {
    int n = 4;
    List<Edge> edges = new ArrayList<>();
    edges.add(new Edge(0, 1, 10));
    edges.add(new Edge(0, 2, 6));
    edges.add(new Edge(0, 3, 5));
    edges.add(new Edge(1, 3, 15));
    edges.add(new Edge(2, 3, 4));

    List<Edge> mst = kruskal(n, edges);
    System.out.println("Edges in MST:");
    for (Edge edge : mst) {
        System.out.println(edge.u + " - " + edge.v + ": " + edge.weight);
    }
}
```

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure under "wiproAssignment".
- Kruskal.java:** The active file contains Java code for a Minimum Spanning Tree (MST) using Kruskal's algorithm.
- Output:** The "Console" tab shows the execution results:

```
Total cost of MST: 19
Edges in MST:
2 - 3: 4
0 - 3: 5
0 - 1: 10
```
- System Information:** The taskbar at the bottom shows the date (01-06-2024), time (15:10), battery level (36°C), and weather (Partly sunny).

Task 3: Union-Find for Cycle Detection

Write a Union-Find data structure with path compression. Use this data structure to detect a cycle in an undirected graph.

```
1 package com.wipro.graphalg;
2
3 import java.util.Arrays;
4
5 class UnionFind {
6     int[] parent;
7     int[] rank;
8
9     UnionFind(int n) {
10         parent = new int[n];
11         rank = new int[n];
12         Arrays.fill(rank, 1);
13         for(int i=0; i<n ;i++) {
14             parent[i] = i;
15         }
16     }
17
18     int find(int i) {
19         if (parent[i] != i) {
20             parent[i] = find(parent[i]);
21         }
22         return parent[i];
23     }
24
25     void union(int x, int y) {
26         int rootX = find(x);
27
28         int find(int i) {
29             if (parent[i] != i) {
30                 parent[i] = find(parent[i]);
31             }
32             return parent[i];
33         }
34
35         void union(int x, int y) {
36             int rootX = find(x);
37             int rootY = find(y);
38
39             if (rootX != rootY) {
40                 if (rank[rootX] < rank[rootY]) { // 1<2
41                     parent[rootX] = rootY;
42                 } else if (rank[rootX] > rank[rootY]) {
43                     parent[rootY] = rootX;
44                 } else {
45                     parent[rootY] = rootX;
46                     rank[rootX]++;
47                 }
48             }
49         }
50     }
51 }
```

```
45
46     class Graph {
47         int V, E;
48         Edge[] edges;
49
50         class Edge {
51             int src, dest;
52         }
53
54         Graph(int v, int e) {
55             this.V = v;
56             this.E = e;
57             this.edges = new Edge[E];
58             for (int i = 0; i < e; i++) {
59                 edges[i] = new Edge();
60                 System.out.println(edges[i].src + " -- " + edges[i].dest);
61             }
62         }
63
64
65         public boolean isCycleFound(Graph graph) {
66             UnionFind uf = new UnionFind(V);
67             for(int i=0; i< E ; ++i) {
68                 int x = find(uf, graph.edges[i].src);
69                 int y = find(uf, graph.edges[i].dest);
70
71                 if(x==y) {
72                     return true;
73                 }
74                 uf.union(x, y);
75             }
76             return false;
77         }
78
79         private int find(UnionFind uf, int i) {
80             return uf.find(i);
81         }
82     }
83
84 }
85
86 public class CycleDetect {
87     public static void main(String[] args) {
88         //int V = 3, E = 3;
89         int V = 3, E = 2;
90         Graph graph = new Graph(V, E);
91         graph.edges[0].src = 0;
92         graph.edges[0].dest = 1;
93         graph.edges[1].src = 1;
94         graph.edges[1].dest = 2;
95     }
96 }
```

```
wipro - DataStructures/src/com/wipro/graphalg/CycleDetect.java - Eclipse IDE
File Edit Source Refactor Search Project Run Window Help
Package Explorer X CycleDetect.java SequenceInS... MergeSorte... CircularQue... GraphAddEdge... BFSjava DFSjava DijkstraJava KruskalJava
DataStructures
  IRE System Library [JavaSE-17]
    src
      com.wipro.assignments
        MiddleList.java
        com.wipro.ds
        com.wipro.graphalg
          CycleDetectJava
          FloydAlg.java
          KruskalAlg.java
        com.wipro.graphs
          Graphs.java
          com.wipro.linear
          com.wipro.nonlinear
          com.wipro.pattern
          com.wipro.search
          com.wipro.sort
    IRE System Library [jdk1.8.0_202]
      gems
      ListJava
      ora
      practice
    wiproAssignment
      IRE System Library [JavaSE-17]
        src
          wiproassignment
            BFS.java
            CircularQueueBFS.java
            Cycle-DetectJava
            DFS.java
            Dijkstra.java
            GraphAddEdge.java
            Kruskal.java
            LinkedListMiddle.java
            MergeSorteLlists.java
            QueueSorter.java
            RemoveDuplicate.java
CycleDetect.java
graph.edges[1].src = 1;
graph.edges[1].dest = 2;
//graph.edges[2].src = 0;
//graph.edges[2].dest = 2;
System.out.println(graph.V + " -- " + graph.E);
for (int i = 0; i < E; i++) {
}
System.out.println(graph.edges[i].src + " -- " + graph.edges[i].dest);
}
if(graph.isCycleFound(graph)) {
  System.out.println("Cycle Found");
} else {
  System.out.println("Cycle Not Found...");
}
}
108 }
109 }
110 }

Console X
<terminated> CycleDetect [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (01-Jun-2024, 325:26 pm - 325:26 pm) [pid: 18816]
0 -- 0
0 -- 0
3 -- 2
0 -- 1
1 -- 2
Cycle Not Found...
```

Task 1: String Operations

Write a method that takes two strings, concatenates them, reverses the result, and then extracts the middle substring of the given length. Ensure your method handles edge cases, such as an empty string or a substring length larger than the concatenated string.

```

1 package wiproassignment;
2 public class StringOperations {
3     public static String middleSubstring(String str1, String str2, int length) {
4         String concatenated = str1.concat(str2);
5         StringBuilder reversed = new StringBuilder(concatenated).reverse();
6         int reversedLength = reversed.length();
7         if (reversedLength == 0 || length > reversedLength) {
8             return "";
9         }
10        int startIndex = (reversedLength - length) / 2;
11        String middleSubstring = reversed.substring(startIndex, startIndex + length);
12        return middleSubstring;
13    }
14
15    public static void main(String[] args) {
16        String str1 = "Hello";
17        String str2 = "World";
18        int length = 5;
19
20        String result = middleSubstring(str1, str2, length);
21        System.out.println("Middle substring: " + result);
22    }
23}

```

Console output:

```
Middle substring: rowOl
```

Task 2: Naive Pattern Search

Implement the naive pattern searching algorithm to find all occurrences of a pattern within a given text string. Count the number of comparisons made during the search to evaluate the efficiency of the algorithm.

wipro - DataStructures/src/com/wipro/pattern/NaivePatternSearching.java - Eclipse IDE

```
1 package com.wipro.pattern;
2 public class NaivePatternSearching {
3     public static void main(String[] args) {
4         String text = "I Love Cats";
5         String pattern = "Cats";
6         search(text, pattern);
7     }
8     private static void search(String text, String pattern) {
9         int strleng = text.length();
10        int patleng = pattern.length();
11        for (int i = 0; i <= strleng - patleng; i++) {
12            int j;
13            for (j = 0; j < patleng; j++) {
14                if (text.charAt(i + j) != pattern.charAt(j)) {
15                    break;
16                }
17            }
18            if (j == patleng) {
19                System.out.println("pattern found at index:" + i);
20            }
21        }
22    }
23 }
24 }
25 }
```

wipro - DataStructures/src/com/wipro/pattern/NaivePatternSearching.java - Eclipse IDE

```
14
15
16
17
18
19
20
21
22
23
24
25
26
```

pattern found at index:7