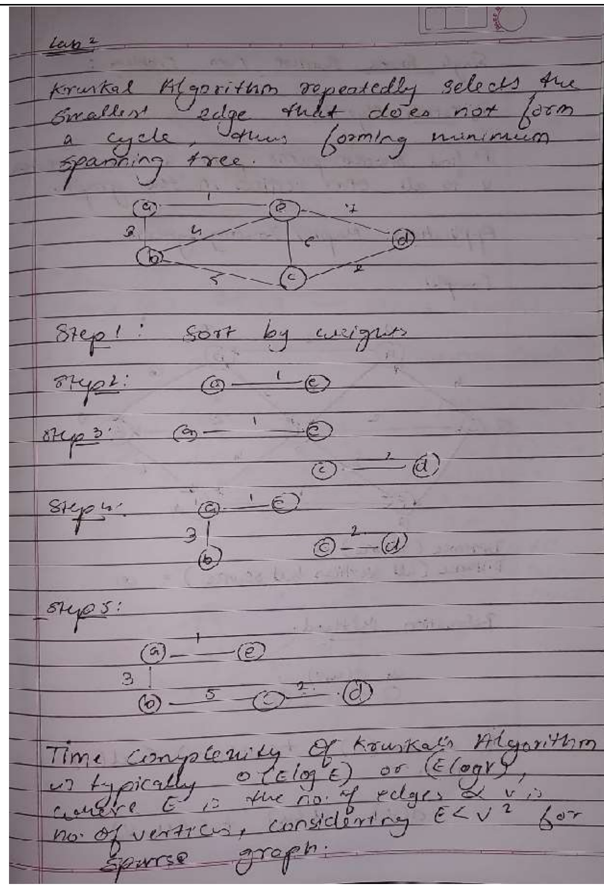
2Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.
Department of Master of Computer Applications

Experiment	2
Aim	To understand and implement greedy Approach
Objective	1) Learn GREEDY Approach 2) Implement Greedy approach problem 3) Solve Greedy approach problem
Name	Vivek Tiwari
UCID	2023510059
Class	FY MCA
Batch	C
Date of Submission	18/02/24

Algorithm and explanation of the technique used	Sort all the edges in non-decreasing order of their weight. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it. Repeat step#2 until there are (V-1) edges in the spanning tree **Kruskal's Algorithm:** KRUSKAL(G): A = ∅ For each vertex v ∈ G. V: MAKE-SET(v) For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v): if FIND-SET(u) ≠ FIND-SET(v): A = A ∪ {(u, v)} UNION(u, v) return A



Code

```
import java.util.*;
class Edge implements Comparable<Edge> {
    char src, dest;
    int weight;

    Edge(char src, char dest, int weight) {
        this.src = src;
        this.dest = dest;
        this.weight = weight;
    }
    public int compareTo(Edge compareEdge) {
        return this.weight - compareEdge.weight;
    }
}

class DisjointSet {
    Map<Character, Character> parent;

    DisjointSet() {
        parent = new HashMap<>();
    }

    void makeSet(char vertex) {
        parent.put(vertex, vertex);
```

```
    }
    char find(char vertex) {
        if (parent.get(vertex) == vertex)
            return vertex;
        return find(parent.get(vertex));
    }
    void union(char vertex1, char vertex2) {
        char parent1 = find(vertex1);
        char parent2 = find(vertex2);
        parent.put(parent1, parent2);
    }
}

class Main {
    public static void main(String[] args) {
        Edge[] edges = {
                new Edge('a', 'b', 3),
                new Edge('a', 'e', 1),
                new Edge('b', 'c', 5),
                new Edge('b', 'e', 4),
                new Edge('c', 'd', 2),
                new Edge('e', 'd', 7),
                new Edge('e', 'c', 6)
        };

        Arrays.sort(edges);
        DisjointSet disjointSet = new DisjointSet();
        for (char c = 'a'; c <= 'h'; c++) {
            disjointSet.makeSet(c);
        }
        // Kruskal's algorithm
        List<Edge> minimumSpanningTree = new ArrayList<>();
        for (Edge edge : edges) {
            char srcParent = disjointSet.find(edge.src);
            char destParent = disjointSet.find(edge.dest);
            if (srcParent != destParent) {
                minimumSpanningTree.add(edge);
                disjointSet.union(edge.src, edge.dest);
            }
        }
        for (Edge edge : minimumSpanningTree) {
            System.out.println(edge.src + " - " + edge.dest + ": " +
edge.weight);
        }
    }
}
```

Output	`"C:\Program Files\Java\jdk-21\bin\java.exe"`
	`a - e: 1`
	`c - d: 2`
	`a - b: 3`
	`b - c: 5`

Justification of the complexity calculated	The time complexity of Kruskal's algorithm is mainly influenced by the sorting of edges, which takes O(E log E) time, where E represents the number of edges in the graph. Additionally, the union-find operations contribute a nearly constant factor per operation. Overall, the time complexity of Kruskal's algorithm is O(E log E), which can be simplified to O(E log V) for sparse graphs, where V is the number of vertices in the graph.
Conclusion	Kruskal's Algorithm is one technique to find out minimum spanning tree from a graph, a tree containing all the vertices of the graph and V-1 edges with minimum cost. The complexity of this graph is (VlogE) or (ElogV). With a time complexity of O(E log E) or O(E log V), it offers a practical solution for a wide range of graph structures, making it a versatile and effective tool for network optimization problems.