PDP ONLINE

DSA – Data Structures

Undirected Graph

Course Planning

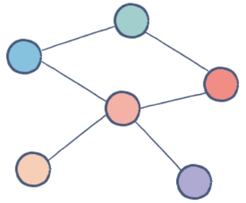
Algorithms	Data Structures	Algorithmic Approaches	Interview Practices
1.Introduction	1.Asymptotic Analysis	1.Search Algorithms	1.In-place Reversal
2.Number 1	2.Dynamic Array	2.Sort Algorithms	2.Two Heaps
3.Number 2	3.LinkedList	3.Dac Algorithms	3.Subsets
4.String 1	4.Stack	4.Recursion	4.Modified BS
5.String 2	5.Queue	5.Sliding Window	5.Bitwise XOR
6.Array 1	6.HashTable	6.Two Pointers	6.Top 'K' Elements
7.Array 2	7.Tree	7.Fast & Slow	7.K-way Merge
8.Matrix	8.Trie	8.Cyclic Sort	8.Knapsack Problem
9.DP 1	9.Directed Graph	9.Breadth First Search	9.Topological Sort
10.DP 2	10.Undirected Graph	10.Depth First Search	10.Mock Interview

https://kurbanovxurshid.medium.com/data-structures-and-algorithms-course-plan-1ab912ec1e17

What are Graphs?

Undirected Graph:

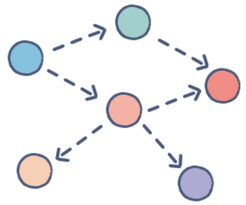
In an undirected graph, nodes are connected by edges that are all bidirectional. For example if an edge connects node 1 and 2, we can traverse from node 1 to node 2, and from node 2 to 1.



Undirected

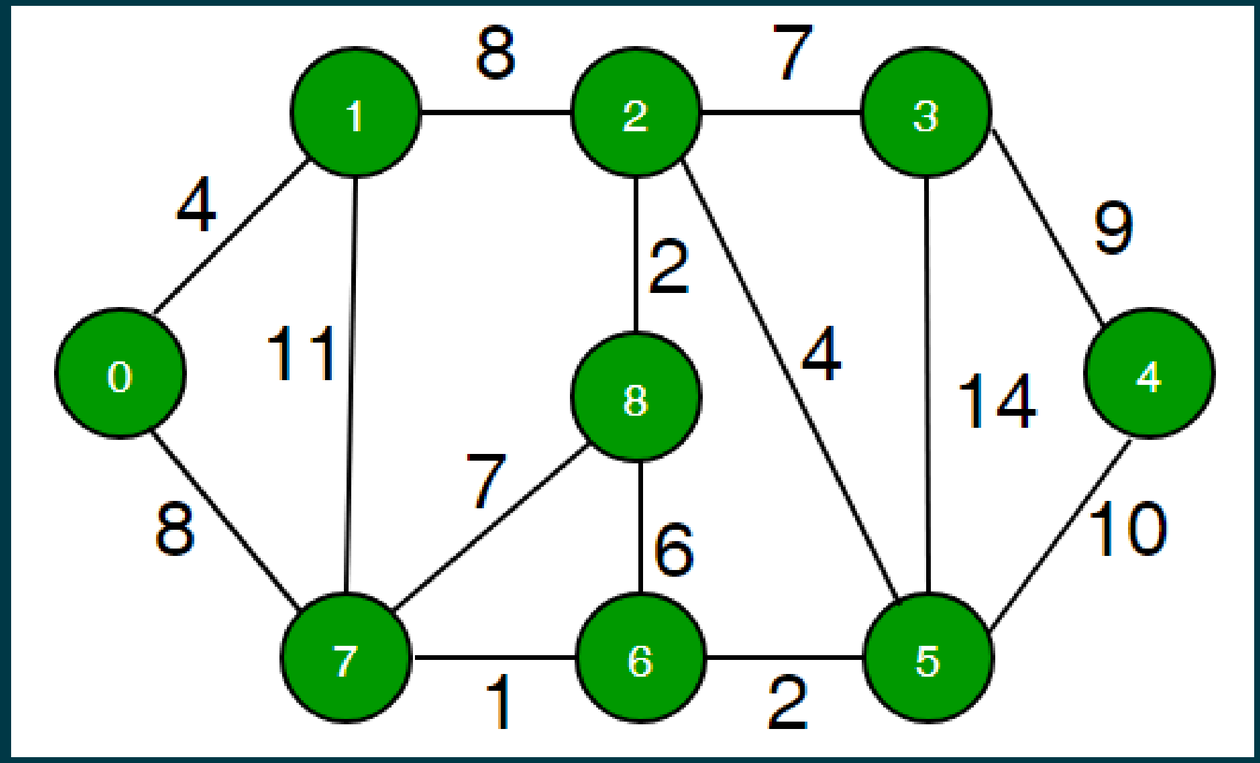
Directed Graph

In a directed graph, nodes are connected by directed edges – they only go in one direction. For example, if an edge connects node 1 and 2, but the arrow head points towards 2, we can only traverse from node 1 to node 2 – not in the opposite direction.



Directed

Weighted Graph



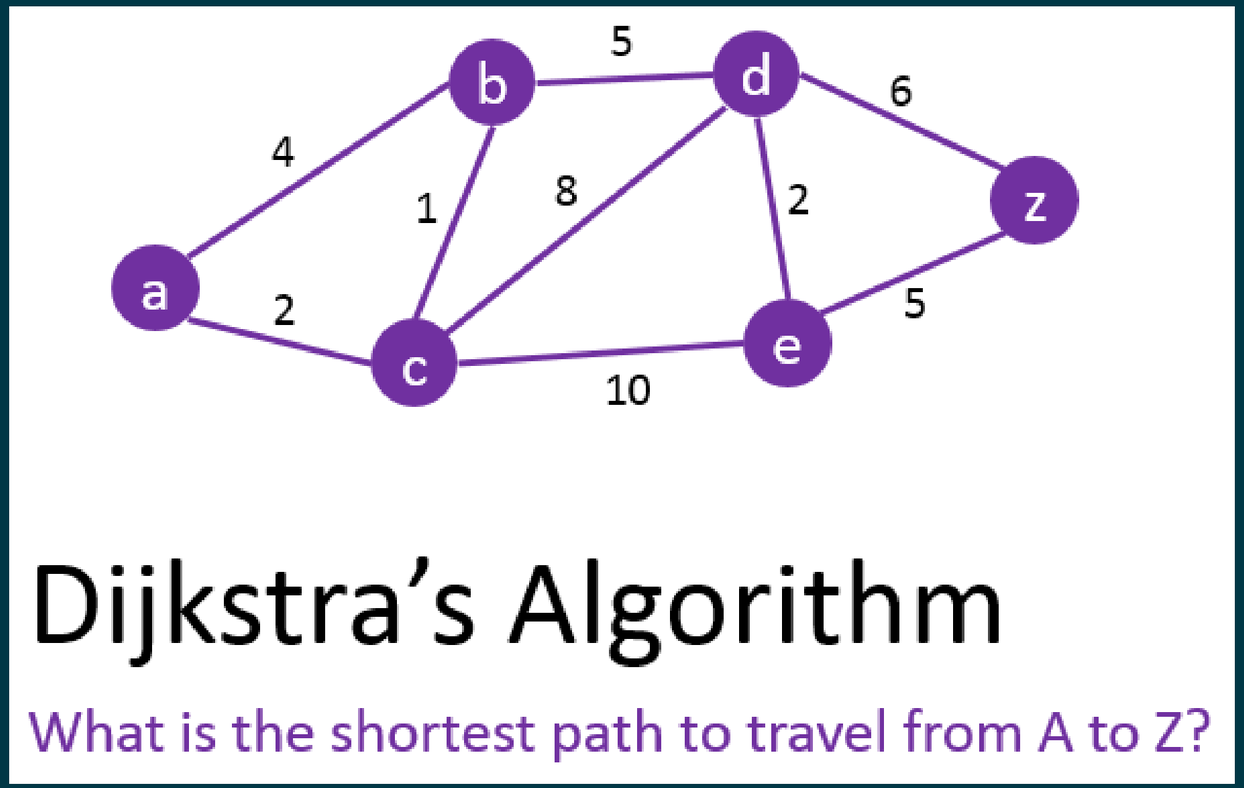
Create Weighted Graph

```
private class Edge {
    private Node from;
    private Node to;
    private int weight;

    public Edge(Node from, Node to, int weight) {
        this.from = from;
        this.to = to;
        this.weight = weight;
    }

    @Override
    public String toString() {
        return from + "->" + to; // A->B
    }
}

private Map<String, Node> nodes = new HashMap<>();
private Map<Node, List<Edge>> adjacencyList = new HashMap<>();


public void addNode(String label) {
    var node = new Node(label);
    nodes.putIfAbsent(label, node);
    adjacencyList.put(node, new ArrayList<>());
}

public void addEdge(String from, String to, int weight) {
    var fromNode = nodes.get(from);
    if(fromNode == null) throw new IllegalArgumentException();
    var toNode = nodes.get(to);
    if(toNode == null) throw new IllegalArgumentException();

    adjacencyList.get(fromNode).add(new Edge(fromNode, toNode, weight));
    adjacencyList.get(toNode).add(new Edge(toNode, fromNode, weight));
}
```

Change Weighted Graph

```
private class Node{
    private String value;
    private List<Edge> edges = new ArrayList<>();

    public Node(String value) {
        this.value = value;
    }

    public String toString() {
        return value;
    }

    public void addEdge(Node to, int weight) {
        edges.add(new Edge(this, to, weight));
    }

    public List<Edge> getEdges(){
        return edges;
    }
}

private class Edge {
private Map<String, Node> nodes = new HashMap<>();

private class NodeEntry{
    private Node node;
    private int priority;

    public NodeEntry(Node node, int priority) {
        this.node = node;
        this.priority = priority;
    }
}
public void addNode(String label) {
    nodes.putIfAbsent(label, new Node(label));
}
public void addEdge(String from, String to, int weight) {
    var fromNode = nodes.get(from);
    if(fromNode == null) throw new IllegalArgumentException();
    var toNode = nodes.get(to);
    if(toNode == null) throw new IllegalArgumentException();

    fromNode.addEdge(toNode, weight);
    toNode.addEdge(fromNode, weight);
}
```

Dijkstra`s Algorithm



The Shortest Distance

```
public int getShortestDistance(String from, String to) {

    var fromNode = nodes.get(from);

    Map<Node, Integer> distances = new HashMap<>();
    for(var node: nodes.values()) {
        distances.put(node, Integer.MAX_VALUE);
    }
    distances.replace(nodes.get(from), 0);

    Set<Node> visited = new HashSet<>();
    PriorityQueue<NodeEntry> queue = new PriorityQueue<>(
            Comparator.comparingInt(ne -> ne.priority)
    );
    queue.add(new NodeEntry(fromNode, 0));

    while(!queue.isEmpty()) {
        var current = queue.remove().node;
        visited.add(current);

        for(var edge: current.getEdges()) {
            if(visited.contains(edge.to))
                continue;

            var newDistance = distances.get(current) + edge.weight;
            if(newDistance < distances.get(edge.to)) {
                distances.replace(edge.to, newDistance);
                queue.add(new NodeEntry(edge.to, newDistance));
            }
        }
    }

    return distances.get(nodes.get(to));
}
```

The Shortest Path

```
        Map<Node, Integer> distances = new HashMap<>();
        for(var node: nodes.values()) {
            distances.put(node, Integer.MAX_VALUE);
        }
        distances.replace(nodes.get(from), 0);

        Map<Node, Node> previousNodes = new HashMap<>(); // 1

        Set<Node> visited = new HashSet<>();
        PriorityQueue<NodeEntry> queue = new PriorityQueue<>(
                Comparator.comparingInt(ne -> ne.priority)
        );
        queue.add(new NodeEntry(fromNode, 0));

        while(!queue.isEmpty()) {
            var current = queue.remove().node;
            visited.add(current);

            for(var edge: current.getEdges()) {
                if(visited.contains(edge.to))
                    continue;

                var newDistance = distances.get(current) + edge.weight;
                if(newDistance < distances.get(edge.to)) {
                    distances.replace(edge.to, newDistance);
                    previousNodes.put(edge.to, current); //2
                    queue.add(new NodeEntry(edge.to, newDistance));
                }
            }
        }

        Stack<Node> stack = new Stack<>(); //3
        stack.push(toNode);
        var previous = previousNodes.get(toNode);
        while(previous != null) {
            stack.push(previous);
            previous = previousNodes.get(previous);
        }

        var path = new Path();
        while(!stack.isEmpty()) {
            path.add(stack.pop().value);
        }
        return path;
    }
```

Task 1

Graph larda Spanning Tree degan tushuncha mavjud va uni misollarda mustaqil o`rganib tushuntiring.