

Solutions

8.1 Representation

Reference example

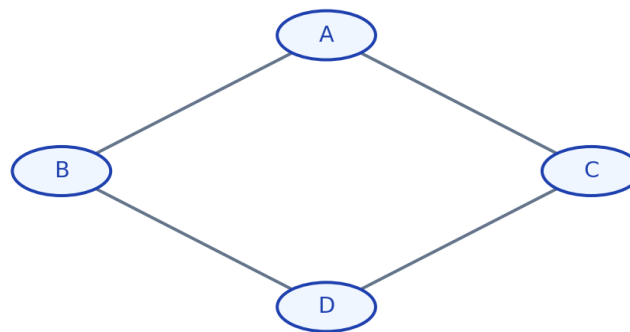
Undirected edges: A-B, A-C, B-D, C-D

Adjacency of A: B, C

Adjacency of D: B, C

Each undirected edge appears at both endpoints

An undirected graph



Edges A-B, A-C, B-D, and C-D; visit order depends on traversal strategy and neighbor order.

8.1.1 - Explain

For 5 vertices and 4 undirected edges, how many neighbor entries does a standard adjacency list store?

There are 8 neighbor entries, two per undirected edge, in addition to the vertex-list structure.

8.1.2 - Explain

Compare adjacency-list and adjacency-matrix space in terms of V and E .

An adjacency list uses $O(V + E)$ space. An adjacency matrix uses $O(V^2)$ space.

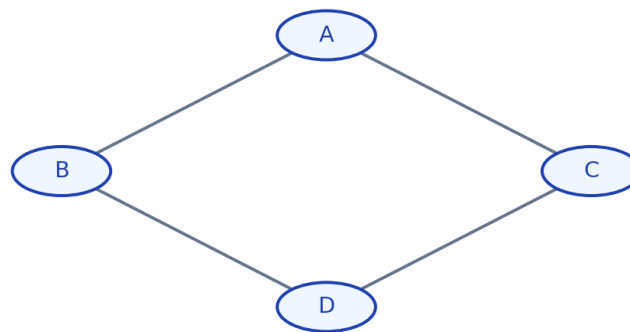
Solutions

8.2 Breadth-First Search

Reference example

Graph: A-B, A-C, B-D, C-D
Start A; neighbors processed alphabetically
Queue initially: A
Mark each vertex when enqueued

An undirected graph



Edges A-B, A-C, B-D, and C-D; visit order depends on traversal strategy and neighbor order.

8.2.1 - Explain

Give BFS order and the edge distance from A to D for the displayed graph.

The order is A, B, C, D under alphabetical neighbor processing. D has distance 2 from A.

Solutions

8.2.2 - Write code

Implement BFS from A for the displayed graph, processing each adjacency list alphabetically. Mark vertices when enqueued.

A queue preserves distance layers and a visited set prevents duplicate enqueues.

```
Map<String,List<String>> graph = Map.of(
    "A", List.of("B","C"), "B", List.of("A","D"),
    "C", List.of("A","D"), "D", List.of("B","C"));
Queue<String> queue = new ArrayDeque<String>();
Set<String> seen = new HashSet<String>();
queue.offer("A"); seen.add("A");
while (!queue.isEmpty()) {
    String node = queue.remove();
    System.out.print(node + " ");
    for (String next : graph.get(node))
        if (seen.add(next)) queue.offer(next);
}
```

Expected output

A B C D

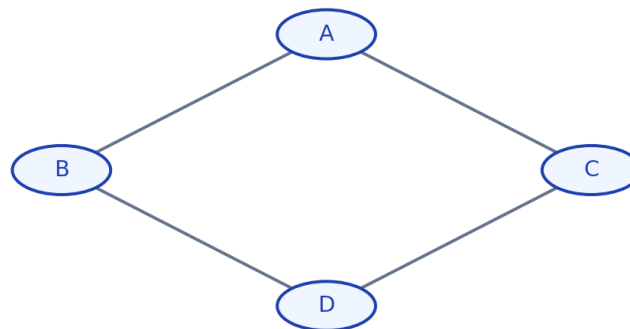
Solutions

8.3 Depth-First Search

Reference example

```
Map<String,List<String>> graph = Map.of(
    "A", List.of("B","C"), "B", List.of("A","D"),
    "C", List.of("A","D"), "D", List.of("B","C"));
class Walker {
    Set<String> seen = new HashSet<String>();
    void visit(String node) {
        if (!seen.add(node)) return;
        System.out.print(node + " ");
        for (String next : graph.get(node)) visit(next);
    }
}
new Walker().visit("A");
```

An undirected graph



Edges A-B, A-C, B-D, and C-D; visit order depends on traversal strategy and neighbor order.

8.3.1 - Explain

Give recursive DFS visit order from A using alphabetical neighbor order.

A, B, D, C. From D, C is the next unvisited neighbor before recursion returns to A.

8.3.2 - Explain

Does DFS necessarily find the minimum-edge path to a target?

No. It explores depth rather than distance layers. BFS is the standard choice for shortest edge-count paths in an unweighted graph.

Solutions

8.4 Shortest Paths and Spanning Trees

Reference example

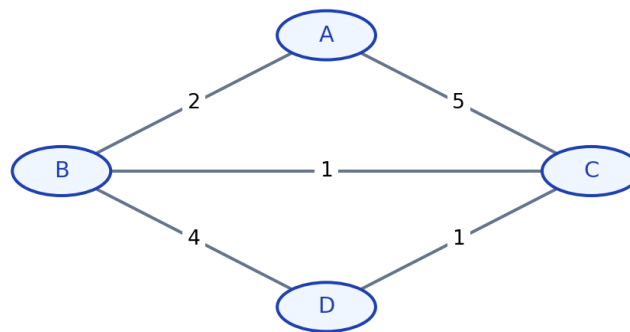
Weighted edges: A-B 2, A-C 5, B-C 1, B-D 4, C-D 1

All weights are positive

Goal 1: shortest path from A to D

Goal 2: minimum total weight connecting all vertices

A weighted undirected graph



A-B-C-D has total weight 4; A-B-D has total weight 6.

8.4.1 - Explain

Compare the A-B-D path with A-B-C-D using the displayed weights.

A-B-D costs $2 + 4 = 6$. A-B-C-D costs $2 + 1 + 1 = 4$, which is shorter.

8.4.2 - Explain

Why does a minimum spanning tree not generally preserve every original shortest path?

It minimizes the total weight needed to connect all vertices without cycles. That global objective differs from minimizing route length between particular endpoints.