Ruth Tilahun
CMSC 337
Lab 6

Simple Graph Algorithms

In the real world, relationships between different entities are quite complex. They involve interconnected data, such as social networks, transportation networks, computer networks, and so on, that cannot be represented using lists or arrays. A graph is a non-linear data structure that can model and analyze such relationships.

In this lab, we will explore different ways of representing graphs and implement a graph ADT (abstract data type) in Java. We will use the graph to represent all the states in the US and their neighbors and implement a simple algorithm to traverse through the graph to find a specific path.

Graphs

A graph consists of nodes/vertices and edges that connect pairs of nodes. Nodes/vertices are the entities or objects in a graph and edges represent the relationships between pairs of nodes. Two nodes are said to be adjacent if they are connected by an edge. Edges in a graph can be either undirected (Figure 1a) or directed (Figure 1b).

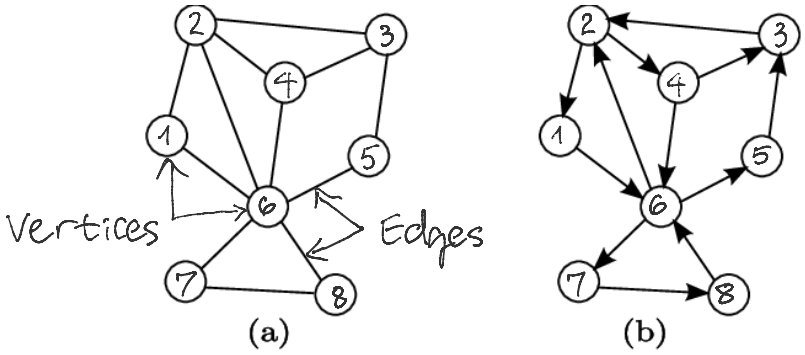

Figure 1: Directed vs. Undirected graphs.

In a directed graph objects can only travel in one direction along the edge. An undirected graph has no restrictions for travelling on it, objects can move freely in either direction.

A cycle in a graph is a path that starts and ends at the same node, revisiting at least one node. A graph without any cycles is called an acyclic graph.

Graph Representation

There are several ways to represent a graph, but we will only look at two of them namely, adjacency lists, and adjacency matrices.

Adjacency List

In an adjacency list representation of a graph, each vertex in the graph is associated with a list of its adjacent or neighboring vertices as in Figure 2. It allows us to efficiently find the collection of adjacent vertices of a given vertex. We could use an n-element array indexed by each vertex whose entry is a list of all the vertices adjacent to it.

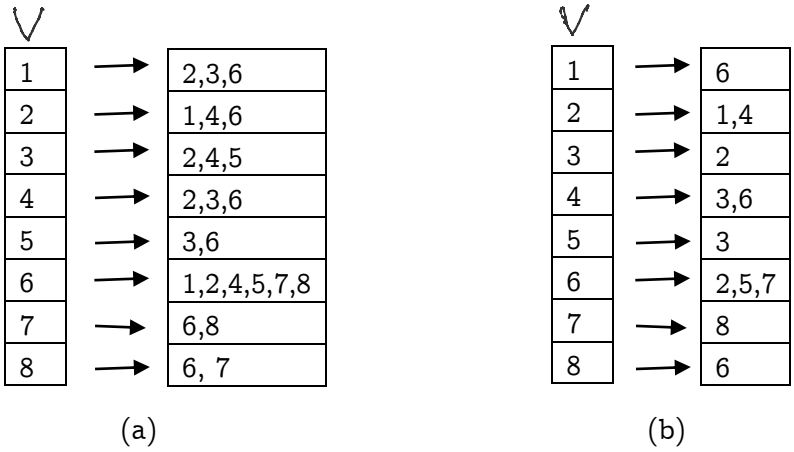

Figure 2: Adjacency list representations of the graphs in Figure 1

For a directed graph such as graph *b* in Figure 1, the list corresponding to each vertex, V contains all the vertices connected to it with arrows going out from V into another vertex. The adjacency list for graph *a* in Figure 1 contains all of the vertices connected to each vertex V.

Adjacency Matrix

In an adjacency matrix representation, we use a two-dimensional matrix or a 2D array of 0s and 1s to represent the graph. We use an n×n adjacency matrix where n is the number of vertices in the graph with each row and each column corresponding to one vertex. In an undirected graph represented with matrix *m*, if there is an edge between vertices *i* and *j,* then both *m[i, j]* and *m[j, i]* are set to 1. If the graph is directed, *m[i, j]* is set to 1 while *m[j, i]* is set to 0.

	1	2	3	4	5	6	7	8
1	0	1	1	0	0	1	0	0
2	1	0	0	1	0	1	0	0
3	0	1	0	1	1	0	0	0
4	0	1	1	0	0	1	0	0
5	0	0	1	0	0	1	0	0
6	1	1	0	1	1	0	1	1
7	0	0	0	0	0	1	0	1
8	0	0	0	0	0	1	1	0

(a)

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	1	0	0
2	1	0	0	1	0	0	0	0
3	0	1	0	0	0	0	0	0
4	0	0	1	0	0	1	0	0
5	0	0	1	0	0	0	0	0
6	0	1	0	0	1	0	1	0
7	0	0	0	0	0	0	0	1
8	0	0	0	0	0	1	0	0

(b)

Figure 3: Adjacency matrix representations of the graphs in Figure 1

Matrix *a* in Figure 3 represents an undirected graph. For instance *a[1,2]* & *a[2,1]* are both 1, and *a[4,6]* & *a[6,4]* are both 1. Matrix *b* shows that the same relationship doesn't hold for a directed graph.

Java Implementation

We will implement our graph ADT based on the adjacency list structure. Since our data has 51 unique names of states in the US, it is convenient to use an array-based data structure to represent each state as a vertex indexed in the array. Therefore, we will use Java's HashMap as it is ideal to map each US state to a unique index in a table/array. As discussed earlier, each entry in the array representing a vertex holds a list of vertices adjacent to it. Thus, we will use Java's LinkedList to represent each list. We could also use an array since we don't need to add or remove an item from the middle of the list. The final product will be a HashMap that maps each vertex/US state to a LinkedList of its adjacent vertices/neighbors. The graph will be undirected since the relationship between neighboring states is symmetrical meaning if Maryland shares a border with Virginia, then Virginia also shares a border with Maryland.

The "Death of a Salesman" puzzle

Now, we will solve the "Death of a Salesman" puzzle where we try to find a path between two states travelling only through states whose names begin with W, L, O, M, A, and N. In order to find a specific path from US state A to B, we need to employ a graph traversal algorithm. For this lab we will implement a straightforward traversal algorithm known as "Breadth First

Traversal/Search" and modify it so that we can find the desired path specified by the puzzle.

```
BreadthFirstTraversal(Graph, startVertex):
    // Create a queue for BFS
    queue = new Queue()

    // Create a boolean array to track visited vertices
    visited = new boolean[Graph.numVertices]

    // Mark the start vertex as visited and enqueue it
    visited[startVertex] = true
    queue.enqueue(startVertex)

    while queue is not empty:
        // Dequeue a vertex from the queue
        currentVertex = queue.dequeue()

        // Process the current vertex (e.g., print it)
        Process(currentVertex)

        // Get the neighbors of the current vertex
        neighbors = Graph.getNeighbors(currentVertex)

        // Enqueue all unvisited neighbors
        for each neighbor in neighbors:
            if not visited[neighbor]:
                visited[neighbor] = true
                queue.enqueue(neighbor)
```

Figure 4: Pseudocode for a breadth first algorithm

The BFS algorithm visits nodes level by level where each level contains a vertex with its adjacent vertices. It starts from a given vertex and explores all its neighbors before moving on to the next level of neighbors. In this algorithm, we use a queue data structure to ensure that vertices are visited in the proper order, since queues order elements in a FIFO (first in first out) manner. Additionally, we use a boolean array, *visited* to keep track of which vertices have been visited to avoid revisiting them and entering an infinite loop.

In Figure 4, we first mark the starting vertex as visited and enqueue/insert it into the queue. Inside the outer loop we dequeue a vertex from the queue and process it. We then enqueue all the unvisited

adjacent vertices of the vertex we just removed. Each time we enqueue a vertex we must set is as visited. We repeat the process by dequeuing a vertex, processing it and enqueuing its adjacent vertices until the queue is empty.

The time complexity of this algorithm is $\Theta(V+E)$ where V is the number of vertices and E is the number of edges in the graph.

Adjusting the algorithm to solve the "Death of a Salesman" problem is quite simple. We only need to change the condition in the inner loop so that we only enqueue a vertex if it starts with one of the specified letters and add another condition to break out of the inner loop when/if we run into the destination US state.*(refer to Appendix A for the path from Washington to Washington DC, and Appendix B for the Java implementation)*

Conclusions

In this lab, we looked at the importance of graphs and two ways to represent them. We used Java's built-in HashMap and LinkedList classes to represent our graph using the adjacency list method where each key-value pair in the HashMap is vertex associated with a list of its adjacent vertices. We also employed the "Breadth First" graph search algorithm and modified it to solve the "Death of a Salesman" puzzle.

Extra Credit

There are paths to Pennsylvania through "salesman" states from almost all states. The states that don't have a path to Pennsylvania are Alaska, North Carolina, South Carolina, and Hawaii.

Appendix A: Sample Runs

Task 1

```
State/s with the most neighbors: [Tennessee, Missouri] with 8 neighbors
Enter a state: Florida
Florida has the following neighbors: [Alabama, Georgia]
Enter a state: Idaho
Idaho has the following neighbors: [Montana, Nevada, Oregon, Utah, Washington,
Wyoming]
Enter a state: Hawaii
Hawaii has no neighbors.
```

Task 2

```
Yes. To get from Washington to District of Columbia, march as follows:
Washington, Oregon, Nevada, Arizona, New Mexico, Oklahoma, Arkansas, Missouri,
Louisiana, Mississippi, Nebraska, Alabama, Wyoming, Montana, North Dakota,
Minnesota, Wisconsin, Michigan, Ohio, West Virginia, Maryland, District of
Columbia
```

Extra Credit

```
1. Yes. To get from Alabama to Pennsylvania march as follows:
Alabama, Mississippi, Arkansas, Louisiana, Missouri, Oklahoma, Nebraska, New
Mexico, Wyoming, Arizona, Montana, Nevada, North Dakota, Oregon, Minnesota,
Washington, Wisconsin, Michigan, Ohio, Pennsylvania
2. Yes. To get from Florida to Pennsylvania march as follows:
Florida, Alabama, Mississippi, Arkansas, Louisiana, Missouri, Oklahoma,
Nebraska, New Mexico, Wyoming, Arizona, Montana, Nevada, North Dakota, Oregon,
Minnesota, Washington, Wisconsin, Michigan, Ohio, Pennsylvania
3. Yes. To get from Georgia to Pennsylvania march as follows:
Georgia, Alabama, North Carolina, Mississippi, Arkansas, Louisiana, Missouri,
Oklahoma, Nebraska, New Mexico, Wyoming, Arizona, Montana, Nevada, North Dakota,
Oregon, Minnesota, Washington, Wisconsin, Michigan, Ohio, Pennsylvania
4. Yes. To get from Mississippi to Pennsylvania march as follows:
Mississippi, Alabama, Arkansas, Louisiana, Missouri, Oklahoma, Nebraska, New
Mexico, Wyoming, Arizona, Montana, Nevada, North Dakota, Oregon, Minnesota,
Washington, Wisconsin, Michigan, Ohio, Pennsylvania
5. Yes. To get from Tennessee to Pennsylvania march as follows:
Tennessee, Alabama, Arkansas, Mississippi, Missouri, North Carolina, Louisiana,
Oklahoma, Nebraska, New Mexico, Wyoming, Arizona, Montana, Nevada, North Dakota,
Oregon, Minnesota, Washington, Wisconsin, Michigan, Ohio, Pennsylvania
6. No. There is no way to get from Alaska to Pennsylvania.
7. Yes. To get from Arizona to Pennsylvania march as follows:
Arizona, Nevada, New Mexico, Oregon, Oklahoma, Washington, Arkansas, Missouri,
Louisiana, Mississippi, Nebraska, Alabama, Wyoming, Montana, North Dakota,
Minnesota, Wisconsin, Michigan, Ohio, Pennsylvania
8. Yes. To get from California to Pennsylvania march as follows:
```

California, Arizona, Nevada, Oregon, New Mexico, Washington, Oklahoma, Arkansas, Missouri, Louisiana, Mississippi, Nebraska, Alabama, Wyoming, Montana, North Dakota, Minnesota, Wisconsin, Michigan, Ohio, Pennsylvania
9. Yes. To get from Colorado to Pennsylvania march as follows:
Colorado, Arizona, Nebraska, New Mexico, Oklahoma, Wyoming, Nevada, Missouri, Arkansas, Montana, Oregon, Louisiana, Mississippi, North Dakota, Washington, Alabama, Minnesota, Wisconsin, Michigan, Ohio, Pennsylvania
10. Yes. To get from Nevada to Pennsylvania march as follows:
Nevada, Arizona, Oregon, New Mexico, Washington, Oklahoma, Arkansas, Missouri, Louisiana, Mississippi, Nebraska, Alabama, Wyoming, Montana, North Dakota, Minnesota, Wisconsin, Michigan, Ohio, Pennsylvania
11. Yes. To get from New Mexico to Pennsylvania march as follows:
New Mexico, Arizona, Oklahoma, Nevada, Arkansas, Missouri, Oregon, Louisiana, Mississippi, Nebraska, Washington, Alabama, Wyoming, Montana, North Dakota, Minnesota, Wisconsin, Michigan, Ohio, Pennsylvania
12. Yes. To get from Utah to Pennsylvania march as follows:
Utah, Arizona, Nevada, New Mexico, Wyoming, Oregon, Oklahoma, Montana, Nebraska, Washington, Arkansas, Missouri, North Dakota, Louisiana, Mississippi, Minnesota, Alabama, Wisconsin, Michigan, Ohio, Pennsylvania
13. Yes. To get from Arkansas to Pennsylvania march as follows:
Arkansas, Louisiana, Mississippi, Missouri, Oklahoma, Alabama, Nebraska, New Mexico, Wyoming, Arizona, Montana, Nevada, North Dakota, Oregon, Minnesota, Washington, Wisconsin, Michigan, Ohio, Pennsylvania
14. Yes. To get from Louisiana to Pennsylvania march as follows:
Louisiana, Arkansas, Mississippi, Missouri, Oklahoma, Alabama, Nebraska, New Mexico, Wyoming, Arizona, Montana, Nevada, North Dakota, Oregon, Minnesota, Washington, Wisconsin, Michigan, Ohio, Pennsylvania
15. Yes. To get from Missouri to Pennsylvania march as follows:
Missouri, Arkansas, Nebraska, Oklahoma, Louisiana, Mississippi, Wyoming, New Mexico, Alabama, Montana, Arizona, North Dakota, Nevada, Minnesota, Oregon, Wisconsin, Washington, Michigan, Ohio, Pennsylvania
16. Yes. To get from Oklahoma to Pennsylvania march as follows:
Oklahoma, Arkansas, Missouri, New Mexico, Louisiana, Mississippi, Nebraska, Arizona, Alabama, Wyoming, Nevada, Montana, Oregon, North Dakota, Washington, Minnesota, Wisconsin, Michigan, Ohio, Pennsylvania
17. Yes. To get from Texas to Pennsylvania march as follows:
Texas, Arkansas, Louisiana, New Mexico, Oklahoma, Mississippi, Missouri, Arizona, Alabama, Nebraska, Nevada, Wyoming, Oregon, Montana, Washington, North Dakota, Minnesota, Wisconsin, Michigan, Ohio, Pennsylvania
18. Yes. To get from Oregon to Pennsylvania march as follows:
Oregon, Nevada, Washington, Arizona, New Mexico, Oklahoma, Arkansas, Missouri, Louisiana, Mississippi, Nebraska, Alabama, Wyoming, Montana, North Dakota, Minnesota, Wisconsin, Michigan, Ohio, Pennsylvania
19. Yes. To get from Kansas to Pennsylvania march as follows:

Kansas, Missouri, Nebraska, Oklahoma, Arkansas, Wyoming, New Mexico, Louisiana, Mississippi, Montana, Arizona, Alabama, North Dakota, Nevada, Minnesota, Oregon, Wisconsin, Washington, Michigan, Ohio, Pennsylvania
20. Yes. To get from Nebraska to Pennsylvania march as follows:
Nebraska, Missouri, Wyoming, Arkansas, Oklahoma, Montana, Louisiana, Mississippi, New Mexico, North Dakota, Alabama, Arizona, Minnesota, Nevada, Wisconsin, Oregon, Michigan, Washington, Ohio, Pennsylvania
21. Yes. To get from Wyoming to Pennsylvania march as follows:
Wyoming, Montana, Nebraska, North Dakota, Missouri, Minnesota, Arkansas, Oklahoma, Wisconsin, Louisiana, Mississippi, New Mexico, Michigan, Alabama, Arizona, Ohio, Nevada, Pennsylvania
22. Yes. To get from Connecticut to Pennsylvania march as follows:
Connecticut, Massachusetts, New York, New Hampshire, New Jersey, Pennsylvania
23. Yes. To get from Massachusetts to Pennsylvania march as follows:
Massachusetts, New Hampshire, New York, Maine, New Jersey, Pennsylvania
24. Yes. To get from New York to Pennsylvania march as follows:
New York, Massachusetts, New Jersey, Pennsylvania
25. Yes. To get from Rhode Island to Pennsylvania march as follows:
Rhode Island, Massachusetts, New Hampshire, New York, Maine, New Jersey, Pennsylvania
26. Yes. To get from District of Columbia to Pennsylvania march as follows:
District of Columbia, Maryland, Pennsylvania
27. Yes. To get from Maryland to Pennsylvania march as follows:
Maryland, Pennsylvania
28. Yes. To get from Virginia to Pennsylvania march as follows:
Virginia, Maryland, North Carolina, West Virginia, Pennsylvania
29. Yes. To get from Delaware to Pennsylvania march as follows:
Delaware, Maryland, New Jersey, Pennsylvania
30. Yes. To get from New Jersey to Pennsylvania march as follows:
New Jersey, New York, Pennsylvania
31. Yes. To get from Pennsylvania to Pennsylvania march as follows:
Pennsylvania
32. No. There is no way to get from North Carolina to Pennsylvania.
33. No. There is no way to get from South Carolina to Pennsylvania.
34. No. There is no way to get from Hawaii to Pennsylvania.
35. Yes. To get from Idaho to Pennsylvania march as follows:
Idaho, Montana, Nevada, Oregon, Washington, Wyoming, North Dakota, Arizona, Nebraska, Minnesota, New Mexico, Missouri, Wisconsin, Oklahoma, Arkansas, Michigan, Louisiana, Mississippi, Ohio, Alabama, Pennsylvania
36. Yes. To get from Montana to Pennsylvania march as follows:
Montana, North Dakota, Wyoming, Minnesota, Nebraska, Wisconsin, Missouri, Michigan, Arkansas, Oklahoma, Ohio, Louisiana, Mississippi, New Mexico, Pennsylvania
37. Yes. To get from Washington to Pennsylvania march as follows:

Washington, Oregon, Nevada, Arizona, New Mexico, Oklahoma, Arkansas, Missouri, Louisiana, Mississippi, Nebraska, Alabama, Wyoming, Montana, North Dakota, Minnesota, Wisconsin, Michigan, Ohio, Pennsylvania

38. Yes. To get from Illinois to Pennsylvania march as follows:
Illinois, Michigan, Missouri, Wisconsin, Ohio, Arkansas, Nebraska, Oklahoma, Minnesota, Pennsylvania

39. Yes. To get from Indiana to Pennsylvania march as follows:
Indiana, Michigan, Ohio, Wisconsin, Pennsylvania

40. Yes. To get from Iowa to Pennsylvania march as follows:
Iowa, Minnesota, Missouri, Nebraska, Wisconsin, North Dakota, Arkansas, Oklahoma, Wyoming, Michigan, Montana, Louisiana, Mississippi, New Mexico, Ohio, Alabama, Arizona, Pennsylvania

41. Yes. To get from Michigan to Pennsylvania march as follows:
Michigan, Ohio, Wisconsin, Pennsylvania

42. Yes. To get from Kentucky to Pennsylvania march as follows:
Kentucky, Missouri, Ohio, West Virginia, Arkansas, Nebraska, Oklahoma, Michigan, Pennsylvania

43. Yes. To get from Wisconsin to Pennsylvania march as follows:
Wisconsin, Michigan, Minnesota, Ohio, North Dakota, Pennsylvania

44. Yes. To get from Ohio to Pennsylvania march as follows:
Ohio, Michigan, Pennsylvania

45. Yes. To get from Minnesota to Pennsylvania march as follows:
Minnesota, North Dakota, Wisconsin, Montana, Michigan, Wyoming, Ohio, Nebraska, Pennsylvania

46. Yes. To get from South Dakota to Pennsylvania march as follows:
South Dakota, Minnesota, Montana, Nebraska, North Dakota, Wyoming, Wisconsin, Missouri, Michigan, Arkansas, Oklahoma, Ohio, Louisiana, Mississippi, New Mexico, Pennsylvania

47. Yes. To get from West Virginia to Pennsylvania march as follows:
West Virginia, Maryland, Ohio, Pennsylvania

48. Yes. To get from Maine to Pennsylvania march as follows:
Maine, New Hampshire, Massachusetts, New York, New Jersey, Pennsylvania

49. Yes. To get from New Hampshire to Pennsylvania march as follows:
New Hampshire, Maine, Massachusetts, New York, New Jersey, Pennsylvania

50. Yes. To get from Vermont to Pennsylvania march as follows:
Vermont, Massachusetts, New Hampshire, New York, Maine, New Jersey, Pennsylvania

51. Yes. To get from North Dakota to Pennsylvania march as follows:
North Dakota, Minnesota, Montana, Wisconsin, Wyoming, Michigan, Nebraska, Ohio, Missouri, Pennsylvania

Appendix B: Java Program from Task 2

```
   public void breadthFirstTraversal(String v) {
        if (!adjList.containsKey(v)) {
            System.out.println("Vertex not found");
            return;
        }
        Queue<String> queue = new LinkedList<>();
        boolean[] visited = new boolean[vertices.size()];
        StringBuilder path = new StringBuilder();
        visited[indexOf(v)] = true;
        queue.add(v);

        while (!queue.isEmpty()) {
            String currentVertex = queue.poll();

            if (currentVertex.equals("District of Columbia")) {
                path.append(currentVertex);
                System.out.println("Yes. To get from Washington to
Distric of Columbia, march as follows: \n" + path.toString());
                break;
            } else
                path.append(currentVertex + ", ");

            for (String neighbor : adjList.get(currentVertex)) {
                if (neighbor.equals("District of Columbia")) {
                    queue.add(neighbor);
                    continue;
                }
                if (isWLOMAN(neighbor) && !visited[indexOf(neighbor)]) {
                    visited[indexOf(neighbor)] = true;
                    queue.add(neighbor);
                }
            }
        }
        System.out.println("No. There is no way to get from Washington
to District of Columbia");
    }
    private boolean isWLOMAN(String v) {
        return v.startsWith("W") || v.startsWith("L") ||
v.startsWith("O") || v.startsWith("M") || v.startsWith("A")
                || v.startsWith("N");
    }
    private int indexOf(String v) {
        return vertices.indexOf(v);
    }
```

Appendix C: Java Program for Extra Credit

In main:

```
for(String vertex : graph.getVertices()) {
            System.out.print(i++ + ". ");
            graph.breadthFirstTraversal(vertex);
        }
```

Breadth-First Traversal:

```
  public void breadthFirstTraversal(String v) {
        if (!adjList.containsKey(v)) {
            System.out.println("Vertex not found");
            return;
        }
        Queue<String> queue = new LinkedList<>();
        boolean[] visited = new boolean[vertices.size()];
        StringBuilder path = new StringBuilder();
        visited[indexOf(v)] = true;
        queue.add(v);

        while (!queue.isEmpty()) {
            String currentVertex = queue.poll();

            if (currentVertex.equals("Pennsylvania")) {
                path.append(currentVertex);
                System.out.println("Yes. To get from " + v + " to
Pennsylvania march as follows: \n" + path.toString());
                break;
            } else
                path.append(currentVertex + ", ");

            for (String neighbor : adjList.get(currentVertex)) {
                if (neighbor.equals("Pennsylvania")) {
                    queue.add(neighbor);
                    continue;
                }
                if (isWLOMAN(neighbor) && !visited[indexOf(neighbor)]) {
                    visited[indexOf(neighbor)] = true;
                    queue.add(neighbor);
                }
            }
        }
        if (!path.toString().contains("Pennsylvania"))
            System.out.println("No. There is no way to get from " + v +
" to Pennsylvania.")}
```