# HW4: Graphs, Traversals

## Reference

**Example Undirected Graph**

|  |  |  |
| --- | --- | --- |
| Visual | Adjacency List | Adjacency Matrix |
|  | [[], [2], [1]] | [  [0, 0, 0]  [0, 0, 1]  [0, 1, 0]  ] |

Questions start on next page ⤵

**Q1.** Given an undirected graph with 10 vertices but 45 edges, which of the following will be true? Select all that apply.

1. **Every node will have at least one** neighbor
2. The graph will have duplicate labels
3. **An Adjacency List would be more space-efficient to represent this graph vs. a Matrix**
4. **The graph is GUARANTEED to have a cycle in it**
5. None of the above

**Q2.** There are N cities, with M roads, where each road connects a pair of cities. You are given city **x**. We want to see if every city has a direct road leading to **x**. Return true if this condition is met, and false otherwise.

1. If we represent this as a graph problem, what are the nodes and edges?
   1. The nodes would be N cities and the edges would be M roads.
2. Is this a directed or undirected graph? – Please consider the situation between two cities in the real world. The question is not about two places in a city.
   1. This would be an undirected graph. An undirected graph has edges without arrows. By using an undirected graph, we will be able to move back and forth from nodes just like how a real road would work in a city. For example, if there is a road connecting cities A and B, it should be possible to go from A-B and from B-A.
3. Given an adjacency matrix for this graph, describe using words how you would find the answer to this problem. You do not have to write code.
   1. We could check if there is a direct path to city x by traversing through the city x row in the adjacency matrix. If we look at the city x row, and check city x to every other city, and it’s marked with a 1, then they are directly connected to each other. Otherwise, if a city is marked with a 0, then the city is not directly connected to city x.
4. Given an adjacency list for this graph, describe using words how you would find the answer to this problem. You do not have to write code.
   1. We would have to traverse through the list of the city x and if we find all the other cities in in the list, then that would indicate that there is a road connect directly from city x to each city.

**Q3.** Given the graph below, answer the following questions.

A) Represent this graph as an adjacency list.

|  |  |  |  |
| --- | --- | --- | --- |
| Adjacency List | | | |
| 0 |  | | |
| 1 | 0 | 2 | 4 |
| 2 | 0 |  | |
| 3 | 0 |
| 4 | 3 |

B) Represent this graph as an adjacency matrix.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Adjacency Matrix | | | | | |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 2 | 1 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 |

C) What is the ordering of nodes If we run Graph DFS starting on node 1? Assume we visit the smallest neighbor first.

1. First, we start at node 1 and move up to 0. We check if 0 is unvisited and then run DFS(0).
   1. 1 -> 0
2. From 0, move back down the edge to 1, and we see that it has been visited, so move to node 2.
   1. 1 <- 0
   2. 1 -> 2
3. From node 2, check if it’s unvisited and then run DFS(2) and move to node 0. We see that node 0, has been visited so move backwards down to node 1.
   1. 2 ->0
   2. 1 <- 2 <-0
4. Move to the next neighbor, which is node 4, and see that it has not been visited, so run DFS(4).
   1. 1 -> 4
5. From 4, move to node 3, check that it hasn’t been visited and then run DFS (3) and move to Node 0.
   1. 4 -> 3
   2. 4 -> 3 -> 0
6. We see that node 0 has been visited so we move back down to node 1.
   1. 1 <- 4 < - 3 <- 0
7. **Final Output: 1, 0 , 2, 4 ,3**

D) Write your Java code(submit a .java file) to implement the DFS for graph traversal using the adjacency matrix (either recursive or iterative).

For the test case, you can directly use the above example. And you should call the DFS function several times with different starting points to show the different traversal orders.

DFS(graph, 0); // one possible output likes 0

DFS(graph, 1); // one possible output likes 1 0 2 4 3

DFS(graph, 2); // …

DFS(graph, 3); // …

DFS(graph, 4); // …