Assignment 6: Adjecency list and matrix and breadth-first- and depth-first- search traversal

Simen Westegaard Larsen

April 19, 2024

Exercise 1

The given adjecency-list representation we can make an adjecency-matrix representation. The start points are indicated on the left size and the end point is on the top side.

Start point \ *Endpoint*	1	2	3	4	5	6	7
1	0	1	0	1	0	1	0
2	0	0	1	0	0	0	0
3	1	0	0	0	0	0	0
4	0	1	0	0	1	0	0
5	0	1	0	0	0	0	0
6	0	0	1	0	0	0	1
7	0	0	1	0	0	0	0

Exercise 2

2.1

Traversal of each node using breadth first search traversal. For breadth first traversal we need two queues one to handle which nodes we are looking at currently and one for keeping track of the nodes we have already visited. Let us call the queue for the current nodes we are looking at "Queue" and the queue keeping track of the already visited "visited".

Step 1: Initially the two queues are empty.

Step 2: Push node A to queue and mark it as visited.

visited: A

queue: A

Step 3: Remove A from the queue and visit its unvisited neighbours and add them to the queue.
visited: A, B, C

queue: B, C

Step 4: Remove B from the queue and visit its unvisited neighbours and add them to the queue.
visited: A, B, C, D, E

queue: C, D, E

Step 5: Remove C from the queue and visit its unvisited neighbours and add them to the queue.
visited: A, B, C, D, E, F

queue: D, E, F

Step 6: Remove D from the queue and visit its unvisited neighbours and add them to the queue.
visited: A, B, C, D, E, F

queue: E, F

Step 7: Remove E from the queue and visit its unvisited neighbours and add them to the queue.
visited: A, B, C, D, E, F

queue: F
Step 8: Remove F from the queue and visit its unvisited neighbours and add them to the queue.
visited: A, B, C, D, E, F

queue:

Now that the queue is empty the algorithm will break and all nodes have been visited.

2.2

Traversal of each node using depth first search traversal. For depth first traversal we need a stack to keep track of the nodes we want to visit next by using a stack we will get to the depth before the breadth, and as in BFS we will also keep track of the nodes we have visited using an array.

Step 1: Initially stack and visited array are empty. visited:
stack:

Step 2: Push A to visited and its neighbours that we have not yet visited to the stack: visited: A
stack: B, C

Step 3: Push B to visited and its unvisited neighbours to the stack: visited: A, B
stack: D, E, C

Step 4: Push D to visited and its unvisited neighbours to the stack: visited: A, B, D
stack: E, C

Step 5: Push E to visited and its unvisited neighbours to the stack: visited: A, B, D, E
stack:C

Step 6: Push C to visited and its unvisited neighbours to the stack: visited: A, B, D, E, C
stack: F

Step 7: Push F to visited and its unvisited neighbours to the stack: visited: A, B, D, E, C, F
stack:

Stack is now empty so we have visited all the nodes and the DFS traversal is over.