Module 2 – Shyam Natarajan

1. The initial configuration can be described by a list of positions of the hinges of the arm. The final configuration can be described the same way with the condition that the end of the arm must be in goal state.

2. Problems with a larger search space and ones that do not have a direct or "greedy" path to the solution are harder.

3. Any state where the arm is not touching an obstacle and the end of the arm is at the goal is valid. A way to describe any state is from the position of the "nodes" representing the hinges and the beginning and end of the arm.

4. The maze has O(n^2) states, while the tile problem has O(n!) states. The arm has a much larger search space which is probably exponential relative to the number of nodes.

5. The neighborhood for the maze is simply the 4 cardinal directions – the number of obstacles adjacent in those 4 directions. For the tile problem the neighborhood depends on where the empty space is, but it is between 2 and 4 states. For the arm problem once again the neighborhood is extremely large as we could choose to move any of the links freely and there is no notion of what exactly represents a successive state since maybe moving one link over a long distance while keeping the others in the same place might be a neighbor, or just dragging a link and letting other nodes move as they do might also be one.

9.
```
stack = [first]
Visited = []
While queue:
    Cur = stack.pop
    If cur == goal
        makePath
    else
        visited.add(cur)
    for neighbor in cur.neighbors:
        if neighbor not in visited:
            stack.push (neighbor)
return fail
```

Using recursion we are able to implement a general algorithm that is agnostic of the current state or type of problem and that can be used for any random state.

10. If the graph in general has many edges, then BFS would take more memory as assuming each node has ~neighbors we would have O(k^2) nodes in our queue. In DFS, we would have the ~k neighbors of each node along the path we explore in depth, which in itself may contain many more than k nodes so for a large search space it would be O(mk) m>k. So if the graph is

dense but small BFS would take more memory and if it is sparse but has long paths DFS would take more memory.

11. BFS uses a queue while DFS uses a stack. We can implement both of these with O(1) time to push and pop.

13. The cost based approach is faster in finding a path, but BFS guarantees the shortest path.

14. We can use a min heap for removeBest() which would take O(logn) time to add each node in the worst case and constant time on average, and would be able to pop the best in constant time. However, the time to update the cost of a node would be O(n), so in order to update costs of the node we can instead use a map from node to cost, which would make the lookup and update O(1), and then we can simply push the new cost onto the heap. This means we would have to check our visited when we pop from the heap, but we can use space to tradeoff the time complexity to check visited, to allow indexing by nodes to make this O(1) as well.

15. Euclidean or manhattan distance can be used in the maze problem, but runs into the issue of the goal not being reachable through a straightline path, making our cost to goal estimate wrong for nodes closer to the goal but that do not have a direct path to it.

17. A* takes both fewer moves and takes a shorter time to find a valid path in all cases, but is far better in the harder cases.

18. We could use Manhattan distance instead since we cannot take diagonal paths in the maze.

19. DFS is generally worse in any case where the first path to be explored in depth contains the goal state, but is suboptimal. We can consider a simple case where there are 3 nodes in the shape of a triangle, and our goal state is a node adjacent to our start state. However with DFS if we expand the other neighbor of the start state we would have a path that is of length 2 where the optimal solution is length 1.

22. Each state is represented by the previous state, the node that was moved, and the movement which means we can reach the same state in multiple ways and they are not identical for the purpose of the problem.

23. The L value is broken up into the x and y components by cos and sin, and this means that we are essentially summing up x components into the x value and y components into the y value making it true.

24. From state 1 to 2, both theta 0 and 1 are positive. From state 2 to 3, theta 0 does not change and 1 is negative. From state 3 to 4 theta 0 is negative and theta 1 is positive

26. An example maze would be one where the shortest path to the goal contains obstacles and we instead require a winding path that is optimal and requires moving away from the goal at many states.