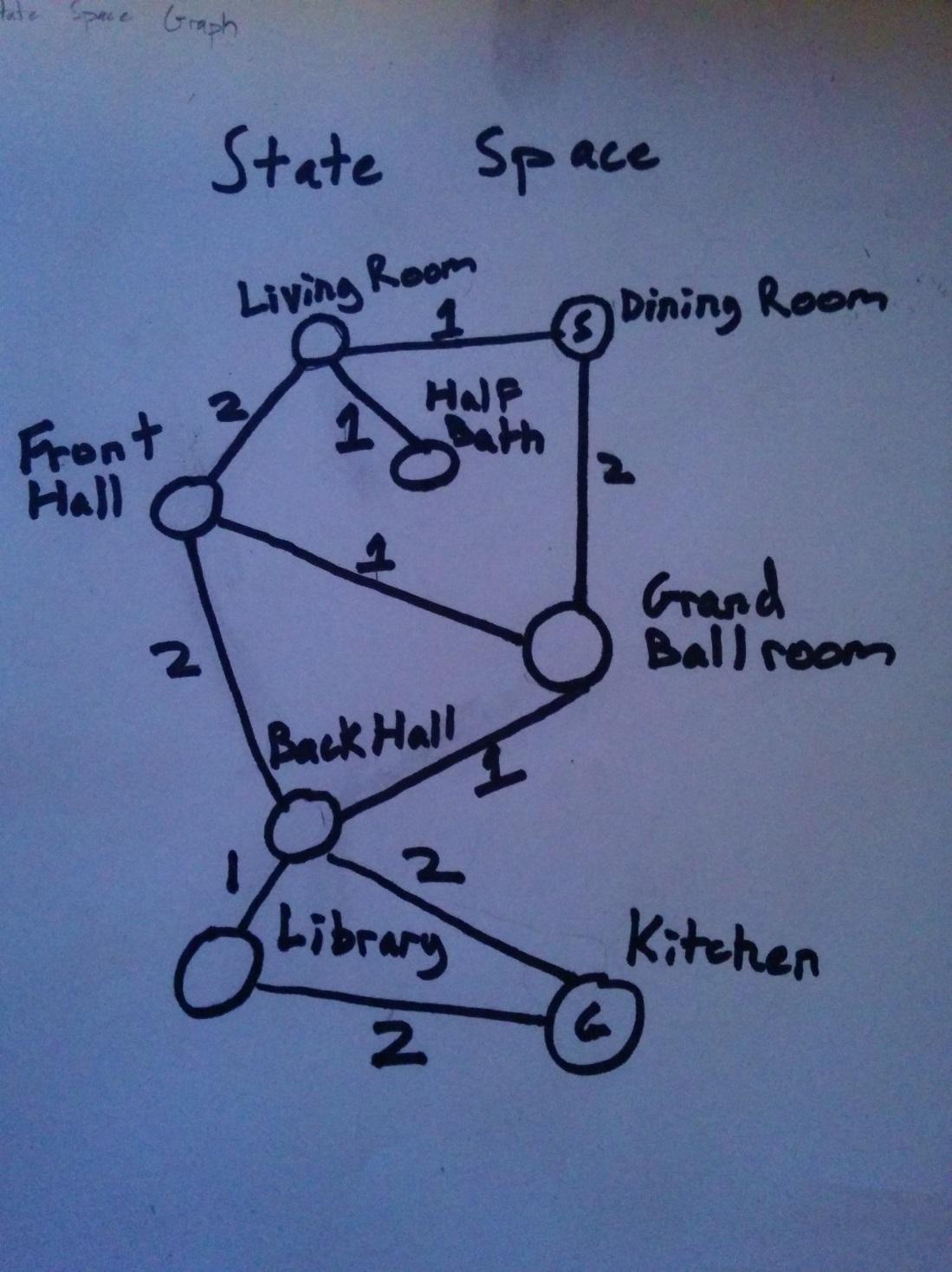
Ian Wilkes

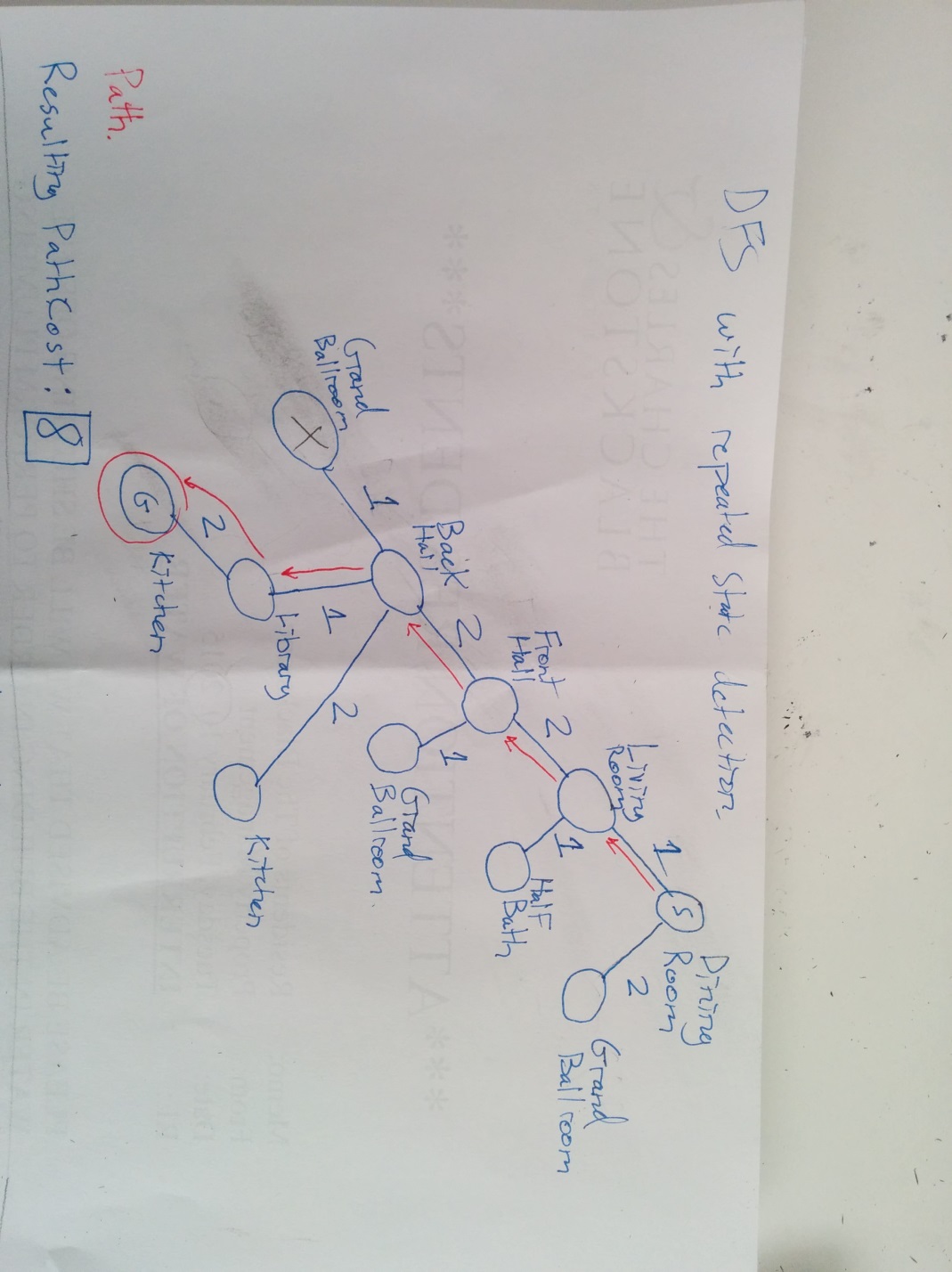
600.335 Artificial Intelligence Spring 2015

Homework 1: Search

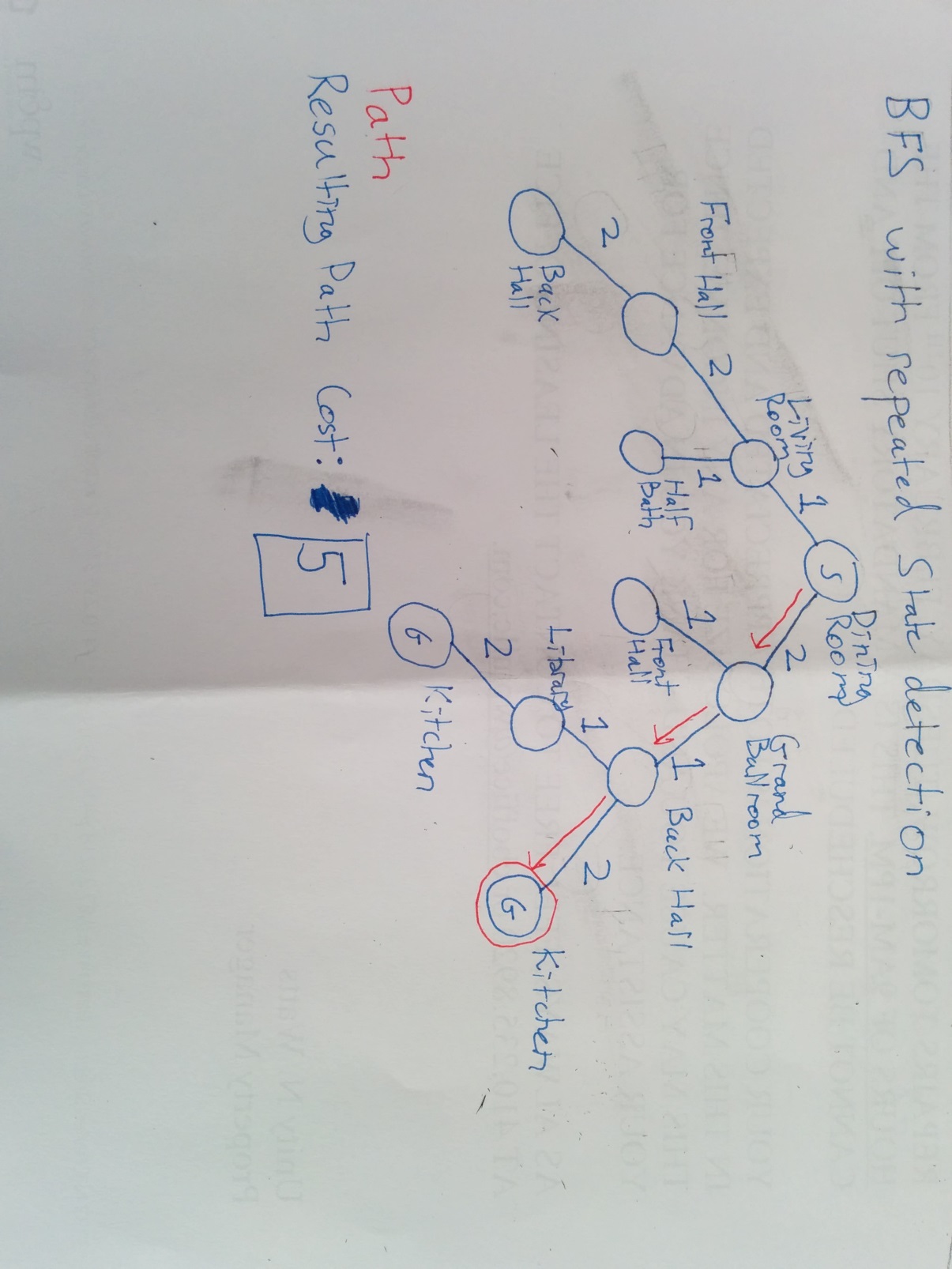
1. The following image contains the state-space for the robot navigation problem.



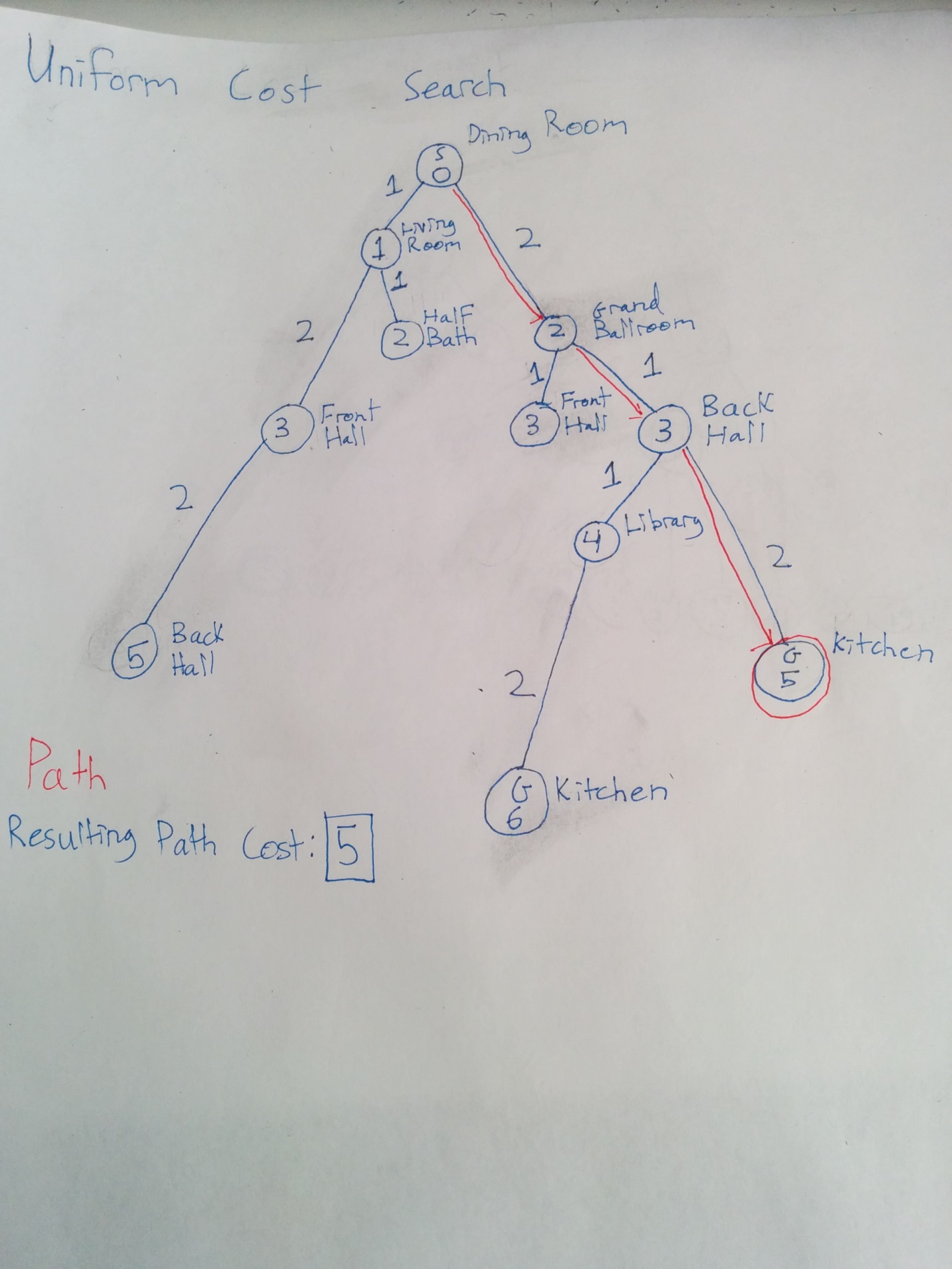
1. The following image contains the DFS search tree. For my DFS, I have included nodes which would have been added to the stack of search nodes, but I have also used repeated state detection, which skips nodes which have already been visited, and does not add nodes which have been previously visited to the search stack. Nodes are considered to be visited when they come out of the search stack and their children are expanded. Also, I have assumed that the first door the robot scans is the first one which it will explore. The path is drawn in red, and the total cost of the path found was 8.



1. The Following image contains the BFS search tree I have created. I have included nodes which would be added to the search queue, but only new nodes which have not been visited previously. I count a node as visited only when it has come out of the search queue and its children are expanded. So when the robot expands nodes, it only includes those which have not previously been visited. Also, if the robot tries to visit a node which has already been visited, it does not revisit that node or expand its children, as this has already been done elsewhere on the search tree. The path used is drawn in red on the image. The total cost of that path was 5.



1. The following is an image of a uniform cost search tree. This uniform cost search had the same repeated state detection as the two previous searches. Nodes with the same cost were visited in the order in which they were added to the priority queue. The path found is drawn in red, and the total cost of that path is 5.



1. The results below are for the performance of the BFS, DFS and A\* algorithms I have implemented. For all three I have included code to prevent nodes from being visited multiple times. A node is considered to be visited if it comes out of the search data structure. If a node has already been visited, it is not expanded again, and if a visited node comes out of the search data structure to be visited again, it is skipped.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| map number | BFS: | | | DFS: | | | A\*: | | |
| nodes | time (s) | cost | Nodes | time (s) | cost | nodes | time (s) | cost |
| 1. | 8 | 0.049 | 7 | 8 | 0.052 | 7 | 8 | 0.090 | 7 |
| 2. | 15 | 0.052 | 14 | 15 | 0.051 | 14 | 15 | 0.053 | 14 |
| 3. | 34 | 0.051 | 14 | 21 | 0.050 | 14 | 28 | 0.049 | 14 |
| 4. | 45 | 0.056 | 22 | 44 | 0.063 | 26 | 45 | 0.082 | 22 |
| 5. | 64 | 0.051 | 15 | 64 | 0.095 | 21 | 56 | 0.094 | 15 |
| 6. | 1844449 | 1.417 | 9499 | 1230805 | 1.709 | 610433 | 1283490 | 1.431 | 9499 |
| 7. | 39 | 0.050 | ∞ | 39 | 0.093 | ∞ | 39 | 0.106 | ∞ |
| 8. | 25 | 0.055 | 21 | 18 | 0.062 | 23 | 20 | 0.048 | 15 |

1. For my A\* heuristic I used the sum of the Manhattan and Euclidean distances all divided by two. Both are individually admissible heuristics, as they always estimate at most the actual distance to the goal, and never more, so by summing the two heuristics, the result can never be more than twice the actual distance to the goal. Thus, dividing by two, we again have an admissible heuristic. This heuristic also dominates the Euclidean distance, as Manhattan distance is always greater than or equal to the Euclidean distance. I chose this instead of regular Manhattan distance as by this method, there are fewer nodes with exactly the same heuristic value than would be the case for regular Manhattan distance, thus making this new heuristic more useable. My version of A\* either matched the path cost of both the BFS and DFS searches in the latter maps, or provided better paths. In almost all cases, the A\* algorithm expanded many fewer nodes than BFS, and most of the time fewer than the DFS, except on a few occasions. On the latter maps, the running time of A\* is approximately comparable with that of DFS, and a little bit slower than DFS.
2. Basic DFS which allows for nodes to be expanded multiple times can get stuck when it encounters a cycle, so a graph with a cycle is an example where DFS may fail, but BFS will succeed. Since only one branch of the search tree is explored, DFS can get stuck continually going through this cycle. BFS will not encounter this same problem, because it explores all branches of the search tree, so if a goal is reachable, BFS will eventually reach it. Essentially, BFS is therefore complete, while DFS is not.
3. Basic DFS is much more memory efficient than DFS, so on large graphs where the branching factor for the search tree is large, DFS may be able to find a solution when BFS may run out of memory, due to the exponential nature of the memory usage of BFS. This is because BFS stores all of the possible paths to all the nodes being explored, while for DFS, all the paths to be explored come off of the same branch of the search tree, so fewer nodes must be stored. Thus, the example of a graph which BFS may fail on is one where there are no cycles, and there is a high branching factor, and a high search tree depth to the first goal.
4. In terms of time, BFS may outperform A\* on problems in which the heuristic used can be slow.

This heuristic is not admissible, because if the goal is one space away from the start, then the actual cost to reach the goal is 1, so the heuristic overestimates.

This heuristic is admissible, because the value is always less than or equal to the total distance to the goal, but it is not useful, because all non-goal nodes will have the same value, which defeats the purpose of the heuristic. This will result in A\* performing almost exactly the same as uniform cost search.

This is a fairly good heuristic, because it is admissible, as Euclidean distance is always less than or equal to the total path length to a goal in a gridworld. It is also fairly useful, as nearly all spaces in the gridworld will have distinct values in thus making the priority queue used in A\* more useful at distinguishing better paths.

This heuristic is not admissible, as in the case of map1 from the project. The Euclidean distance from the start to the goal is 7, but twice that is more than the actual distance to the goal, so this is not an admissible heuristic.

This heuristic is admissible, as in a gridworld, the minimum path length is always dictated by the Manhattan distance to the goal. It is fairly useful, as it will always predict the total distance to reach the goal from the current node, but because there are many configurations which can produce the same Manhattan distance, it may not be extremely useful, as many nodes will be expanded because they will be equally good in the priority queue of A\*.

This heuristic is also admissible, as the Manhattan distance is equal to the path length to the goal, and half that distance is always less than or equal to that distance. This heuristic runs into the same problem as the Manhattan distance, but is slightly worse, because it is dominated by the Manhattan distance, so less emphasis is placed on nodes being close to the goal, and more is placed on simply being farther away from the start node in the A\* search. This search will almost certainly result in more nodes being expanded than for the regular Manhattan distance heuristic.

1. The Euclidean distance between two locations on the globe may take aircraft through said globe. This is somewhat of a problem when trying to fly. A better heuristic could be the shortest arc length on the surface of the earth from the origin to the destination.