COMP 5600 HW1

1. **Advanced Approaches: 10 points**
   1. **Define your score function**
      1. The score functions f(s) will be the number of queens intersecting a given square/cell for each cell. When considering neighbors, we choose cells with the least number of queens already intersecting that cell position.
   2. **Define your neighbors and how to generate them**
      1. Since there are 25 columns, 25 rows, and 25 queens, we know that there will only be a single queen for each column and row. The queens are initialized diagonally through the center, so they start as already belonging to their unique column and row. For this reason, the two most logical choices for the neighborhood would be the x axis or the y axis of each respective queen. For my program, I used the y axis as the neighborhood for each queen. In other words, each queen only considers moving to the cells above or below it.
      2. We generate these neighbors by grabbing cell information above and below the current cell of the queen until we hit the borders of the grid/board.
   3. **Report the best state you can find**
      1. The program I wrote ran with no goal state reached. In theory, we should be able to find a state where the cell of each queen is conflicting with the cells of any other queen.
   4. **Report how many conflicting queens there are in your best result**
      1. In theory there should be 0 conflicting queens by the end. My program reached 4 conflicting queens minimum.
   5. **Discuss the approach you used/designed and your result**
      1. I used a 2D array as a data structure. The value of each cell in the array is the score of the cell (how many queens are intersecting with that cell). Cells that have queens on them have the value 26 + however many queens are intersecting with that cell. Therefore, cells with the value 26 have queens with no conflicting queens intersecting.
      2. To generate cell scores, I created a function that went through every cell/tile and looked in every intercardinal direction (North, NorthEast, East, SouthEast, etc.) for each cell. If a cell finds a queen during this check (>26), it increases its cell score which is initially set to 0.
      3. I used the hillclimbing algorithm because it was the simplest to implement. It is unnecessary to allow queens to move in any possible direction because they already begin initiated in their respective rows or columns. For this reason, I decided to only move queens up and down their columns. Therefore, the neighborhood for each queen is the y axis of the board.
      4. For each column, each queen checks for cells with a lower score than the one it is currently on. Cells that have the same score are added to an array. If more than one cell has the same score, a successor is randomly chosen.
2. **A\* Algorithm: 10 points**
   1. **How many steps did you use to achieve the goal state?**
      1. 16 Steps
   2. **How many states did you explore before achieving the goal state?**
      1. 17 States…beginning with the starting state and ending with a confirmed goal state.
   3. **Report the 5th state you explored**
      1. **A screenshot of a cell phone

         Description automatically generated**
      2. This is state #5 if we consider the initial state to be state #0.
   4. **Report the 5th from the last state you explored**
      1. **A screenshot of a cell phone

         Description automatically generated**
      2. This is state #12 if we consider the initial state to be state #0.
3. **A\* Algorithm: 10 points**
   1. **Prove A\* algorithm is optimal, given the heuristic function h(s) is admissible**
      1. First let us define *optimal* as it is defined in the slides: an algorithm is optimal if it will find the least-cost goal.
      2. A\* is similar to the A algorithm in that it considers real path costs to successor states, and also the heuristic path costs from the successor states to the goal state.
      3. A\* is different from A in that its heuristic function is restricted such that the heuristic cost must not be more than the actual cost from a successor to a goal state. A heuristic function that satisfies this condition is considered *admissible*.
      4. Because this restriction is put on the heuristic function, as long as we have a heuristic function that is accurate *relative* to every node, we will find a least-cost optimal path regardless if the heuristic function is pinpoint accurate or not.
4. **UCS Algorithm: 5 points**
   1. **Clarify the difference between uniformed cost search algorithm and Dijkstra’s algorithm**
      1. Dijkstra’s algorithm generates a shortest path tree which finds the shortest path to every node in a graph by traversing every vertices. Its aim is to optimize a space to be traversed. Uniform cost search is similar, but we are constantly choosing the least cost vertices with the intention of finding a specific goal state. Once this goal state is found, the algorithm is complete.
5. **Open Question: 5 points**
   1. **Define why your heuristic function is ‘better’**
   2. **How many steps did you use to achieve the goal state?**
   3. **How many states did you explore before you achieved the goal state?**
   4. **Report the 5th state you explored**
   5. **Report the 5th from the last state you explored**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2** | **3** | **7** | **4** | **5** |
| **1** | **//** | **11** | **//** | **8** |
| **6** | **10** |  | **12** | **15** |
| **9** | **//** | **14** | **//** | **20** |
| **13** | **16** | **17** | **18** | **19** |

**State 0**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2** | **3** | **7** | **4** | **5** |
| **1** | **//** |  | **//** | **8** |
| **6** | **10** | **11** | **12** | **15** |
| **9** | **//** | **14** | **//** | **20** |
| **13** | **16** | **17** | **18** | **19** |

**State 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2** | **3** |  | **4** | **5** |
| **1** | **//** | **7** | **//** | **8** |
| **6** | **10** | **11** | **12** | **15** |
| **9** | **//** | **14** | **//** | **20** |
| **13** | **16** | **17** | **18** | **19** |

**State 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2** |  | **3** | **4** | **5** |
| **1** | **//** | **7** | **//** | **8** |
| **6** | **10** | **11** | **12** | **15** |
| **9** | **//** | **14** | **//** | **20** |
| **13** | **16** | **17** | **18** | **19** |

**State 3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **2** | **3** | **4** | **5** |
| **1** | **//** | **7** | **//** | **8** |
| **6** | **10** | **11** | **12** | **15** |
| **9** | **//** | **14** | **//** | **20** |
| **13** | **16** | **17** | **18** | **19** |

**State 4**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
|  | **//** | **7** | **//** | **8** |
| **6** | **10** | **11** | **12** | **15** |
| **9** | **//** | **14** | **//** | **20** |
| **13** | **16** | **17** | **18** | **19** |

**State 5**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
| **6** | **//** | **7** | **//** | **8** |
|  | **10** | **11** | **12** | **15** |
| **9** | **//** | **14** | **//** | **20** |
| **13** | **16** | **17** | **18** | **19** |

**State 6**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
| **6** | **//** | **7** | **//** | **8** |
| **9** | **10** | **11** | **12** | **15** |
|  | **//** | **14** | **//** | **20** |
| **13** | **16** | **17** | **18** | **19** |

**State 7**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
| **6** | **//** | **7** | **//** | **8** |
| **9** | **10** | **11** | **12** | **15** |
| **13** | **//** | **14** | **//** | **20** |
|  | **16** | **17** | **18** | **19** |

**State 8**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
| **6** | **//** | **7** | **//** | **8** |
| **9** | **10** | **11** | **12** | **15** |
| **13** | **//** | **14** | **//** | **20** |
| **16** |  | **17** | **18** | **19** |

**State 9**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
| **6** | **//** | **7** | **//** | **8** |
| **9** | **10** | **11** | **12** | **15** |
| **13** | **//** | **14** | **//** | **20** |
| **16** | **17** |  | **18** | **19** |

**State 10**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
| **6** | **//** | **7** | **//** | **8** |
| **9** | **10** | **11** | **12** | **15** |
| **13** | **//** | **14** | **//** | **20** |
| **16** | **17** | **18** |  | **19** |

**State 11**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
| **6** | **//** | **7** | **//** | **8** |
| **9** | **10** | **11** | **12** | **15** |
| **13** | **//** | **14** | **//** | **20** |
| **16** | **17** | **18** | **19** |  |

**State 12**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
| **6** | **//** | **7** | **//** | **8** |
| **9** | **10** | **11** | **12** | **15** |
| **13** | **//** | **14** | **//** |  |
| **16** | **17** | **18** | **19** | **20** |

**State 13**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
| **6** | **//** | **7** | **//** | **8** |
| **9** | **10** | **11** | **12** |  |
| **13** | **//** | **14** | **//** | **15** |
| **16** | **17** | **18** | **19** | **20** |

**State 14**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
| **6** | **//** | **7** | **//** | **8** |
| **9** | **10** | **11** |  | **12** |
| **13** | **//** | **14** | **//** | **15** |
| **16** | **17** | **18** | **19** | **20** |

**State 15**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
| **6** | **//** | **7** | **//** | **8** |
| **9** | **10** |  | **11** | **12** |
| **13** | **//** | **14** | **//** | **15** |
| **16** | **17** | **18** | **19** | **20** |

**State 16**