**Question 1:** Depth First Search

Using a stack(imported from util) to store the node states(node position), a list as the exploredlist to store the visited node state, a list(or stack but list is enough) to store the list of actions to each node(to clarify, it is a list storing list). Each time pop out one node(means to explore) and add it to explored list, then judge if it is the goal state, if is not, then get the successors of this state and add them all into the stack one by one. But since DFS will explore newborn nodes first and to save the space, I modify that if the successor is explored, then we will not put it in stack. If has no successor or the find the goal state, then pop out the action list to get or drop the actions to this node.

**Question 2:** Breadth First Search

The only difference DFS and BFS is that BFS uses queues(from util) to store the state and action list separately. Using a list to store explored nodes. And since BFS visits nodes in order, when pop out a state from queue and then judge if it is explored, if is, then do not visit it.

**Question 3:** Uniform Cost Search

Using priority queue to store state(a set of node position and action list to this node), each time pop one state from priority queue, if this node is not explored, get its successor and add the successor’s position, action list and the cost of the action list into priority queue. If this node is goal, get action list from state.

**Question 4:** A Star Search

The implement of A star search is very similar to uniform cost search. The only difference is in A star search, we add the cost of action list and the distance of this node from heuristic function together as the total cost and add it into priority queue.

**Question 4.1:** Comparisons of Four Search Algorithms

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Cost / Expanded** | **DFS** | **BFS** | **UCS** | **A\* + ManH** | **A\* + EucH** |
| tinyMaze | 10 / 15 | 8/15 | 8/15 | 8/14 | 8/13 |
| mediumMaze | 130 / 146 | 68/269 | 68/269 | 68/221 | 68/226 |
| bigMaze | 210 / 390 | 210/620 | 210/620 | 210/549 | 210/557 |

**Question 5:** Representation for Corners problem

First, initialize variables like walls, corners. And I add a list(exploredCornerList) to store the visited corners. State is defined as a set of node’s position and list of visited corners. Only the node is a unvisited node and it has visited other three corners before. To get successors, first get the visited corners and if the successor is a unvisited corner, add it into exploredCornerList to show all successor generated by this node has visited this corner. Then get successor’s positon, action and cost.

**Question 6:** Heuristics for Corners Problem

First, since it is four corners now, not a single goal node. We can not use Manhattan distance directly. However, all we try to figure out is a path from start to the end, the end must be one of corners. And the path must travel through four corners and the length of this path should be least. Then, I divide the path into four parts, from start to nearest corner1, corner1 to nearest corner2,corner2 to nearest corner3, corner3 to end. Since all are corners, this way can make the total cost of path least. And heuristic distance of one point is the total cost of the path to all its unvisited corners(one of which is end). This heuristic distance is admissible since for each point, distance is the least sum of Manhattan distance to the end which is surely less than actual cost. Also it is consistent, since for each two node, their heuristic distance difference is surely less than actual cost difference.

**Question 6.1:** check Heruristic

Q1: == Q2:non-trivial Q3:admissible Q4: > Q5:consistent

**Question 7:** Food Heuristic

First using the same idea from Question6, in this problems, the number of goals changes from four to n(get from foodGrid). And the heuristic distance of one node can be its distance to its closest food1 plus the distance of food1 to its nearest food2……till to the end. But, a big difference between Question7 and Question 6 is that food is more and not only appears in the corner, so get the least parts and add them all do not guarantee the total path is least(shortcoming of Dijkstra). So for each node, we take account of each node’s food near. And calculate the total cost of path from the node to this food plus this food to the end(during calculation from this food to end, only using the nearest food as the next food to reduce time complexity). And the sort all these paths to get the shortest one. In this way, heuristic distance become more efficient since it can urge search to the path with shortest cost more. And since it is the improvement of algorithm in question6, with more optimal heuristic distance, it is adimissible and consistent.

**Question 8:** Hill-Climbing Search for 8-Queens Problem

**Implement getNumberOfAttacks:**

From the first column, find the queen in this column and count the number of queens in its same line(only queen on the right) and in its up and down diagonals(also only queens on the right of this queen). Then move to next column and count again. In the end add all nums.

**Implement getBetterBoard:**

Using the given fuction getCostBoard, and get the current cost board and use random.choice to choose new queen with least value in cost board randomly. Then remove previous queen in chosen queen’s column, and add new queen in squareArray by changing values in squareArray.

Then generate new board with new squareArray and return new board, minium number of attacks, new queen’s row and column.

**Modify the Stop Criterion:**

Previous logic is when current number of attacks is less than or equals new board’s number of attacks then stop. Modify this as, when current number of attacks equals zero, stop. Only when current number of attacks is less than or the value of i(represents exploring times) is larger than 100 then stop.