**ANSWERS OF THE QUESTIONS:**

1. ANSWER OF FIRST QUESTION:

**Initial state:** A board matrix measured m x n with one goal state(G), one or two (depending on the orientation) initial state of the block [(S), (S, S); only one of these two], untouchable tiles(X) and the ones that allows block to move (O).

**States:** 1) Block being vertical and occupying only one safe tile, block being horizontal and occupying two safe tiles with 2) moving right would keep the orientation horizontal and with 3) moving right would change the orientation to vertical.

**Successor states:** Any coordinate that the block can move to, as long as the coordinates are consisting of safe adjacent tiles.

**Goal test:** The condition to be satisfied at the end of the game, and in this case being the vertical state of the block on the coordinates of G (goal state).

**Step cost:** 1 for each step.

1. AS INDICATED IN THE README, USE MY ATTACHED PYTHON CODE.
2. PROVE THE ADMISSIBILITY.

I use half of the Euclidean distance as a heuristic function. It is admissible because it never overestimates the real cost. We cannot use the Euclidean distance per se, since it may sometimes overestimate the real cost. So, basically, we take the half of it to be on the safe side.

For example, if we were to be on (4,4) and goal node is (4,7) heuristic function of Euclidean distance would say that the cost is 3. However, since we are standing vertically on (4,4) just 2 moves would suffice to get to the goal.

That’s why I took my heuristic function as the half of it, to make sure that it never overestimates the cost and thus, is admissible.

1. AS INDICATED IN THE README, USE MY ATTACHED PYTHON CODE. (You will see the outputs consecutively for Q2 and Q4.)
2. HERE:

**INFORMATION NEEDED FOR UNIFORM COST SEARCH:**

|  |
| --- |
| Line # Mem usage Increment Line Contents |
| ================================================ |
| 174 13.8 MiB 13.8 MiB @profile |
| 175 def uniform\_cost\_search(Graph): |
| 176 |
| 177 #function to make Uniform Cost search |
| 178 |
| 179 13.8 MiB 0.0 MiB frontier = [node(Graph.getStartCoord(), [], 0)] |
| 180 13.8 MiB 0.0 MiB visited = {} |
| 181 |
| 182 13.8 MiB 0.0 MiB goalState = node(Graph.getGoalCoord()) # Goal never changes |
| 183 |
| 184 13.8 MiB 0.0 MiB while frontier: |
| 185 13.8 MiB 0.0 MiB current = heapq.heappop(frontier) |
| 186 # frontier.pop() //It does not work this way. |
| 187 13.8 MiB 0.0 MiB path = current.getParent() + [current.getCor()] |
| 188 |
| 189 13.8 MiB 0.0 MiB if current == goalState: |
| 190 13.8 MiB 0.0 MiB print("Number of nodes visited in UCS algorithm: ", len(visited)) |
| 191 13.8 MiB 0.0 MiB return path |
| 192 |
| 193 13.8 MiB 0.0 MiB currentCoord = str(current.getCor()) |
| 194 13.8 MiB 0.0 MiB visited[currentCoord] = current.getCost() |
| 195 |
| 196 13.8 MiB 0.0 MiB expansion = getAvailableFrontier(Graph, current.getCor()) |
| 197 13.8 MiB 0.0 MiB successorStates = [] |
| 198 |
| 199 13.8 MiB 0.0 MiB for y in expansion: |
| 200 13.8 MiB 0.0 MiB successorStates.append(node(y, path, 1 + current.getCost())) |
| 201 |
| 202 13.8 MiB 0.0 MiB for successor in successorStates: |
| 203 13.8 MiB 0.0 MiB cost = successor.getCost() |
| 204 13.8 MiB 0.0 MiB try: |
| 205 13.8 MiB 0.0 MiB index = frontier.index(successor) |
| 206 13.8 MiB 0.0 MiB except Exception: |
| 207 13.8 MiB 0.0 MiB pass |
| 208 |
| 209 13.8 MiB 0.0 MiB v = str(successor.getCor()) |
| 210 13.8 MiB 0.0 MiB if v not in visited and successor not in frontier: |
| 211 13.8 MiB 0.0 MiB heapq.heappush(frontier, successor) |
| 212 # tmp = copy.deepcopy(successor) |
| 213 # frontier.append(tmp) |
| 214 # SEE MY NOTES FOR THIS |
| 215 13.8 MiB 0.0 MiB elif successor in frontier and frontier[index].getCost() > cost: |
| 216 frontier[index].setCost(cost) |
| 217 frontier[index].setParent(successor.getParent()) |
| 218 |
| 219 print("Visited size is ", visited.\_\_sizeof\_\_()) |
| 220 return None |

**Number of nodes visited in UCS algorithm:** 35

[[2, 3], [3, 3]]

[[2, 4], [3, 4]]

[[2, 5], [3, 5]]

[[2, 6], [3, 6]]

[[2, 7], [3, 7]]

[[4, 7]]

**Run Time of UCS:** 0.004021644592285156

**INFORMATION NEEDED FOR A\* SEARCH:**

|  |
| --- |
| Line # Mem usage Increment Line Contents |
| ================================================ |
| 222 13.8 MiB 13.8 MiB @profile |
| 223 def aStar\_function(Graph): |
| 224 13.8 MiB 0.0 MiB frontier = [node(Graph.getStartCoord(), [], 0, 0)] |
| 225 13.8 MiB 0.0 MiB visited = {} |
| 226 |
| 227 13.8 MiB 0.0 MiB goalState = node(Graph.getGoalCoord()) |
| 228 |
| 229 13.8 MiB 0.0 MiB while frontier: |
| 230 13.8 MiB 0.0 MiB current = heapq.heappop(frontier) |
| 231 # frontier.pop() //It does not work this way. |
| 232 13.8 MiB 0.0 MiB path = current.getParent() + [current.getCor()] |
| 233 |
| 234 13.8 MiB 0.0 MiB if current == goalState: |
| 235 13.8 MiB 0.0 MiB print("Number of nodes visited in A\* algorithm: ", len(visited)) |
| 236 13.8 MiB 0.0 MiB return path |
| 237 |
| 238 13.8 MiB 0.0 MiB currentCoord = str(current.getCor()) |
| 239 |
| 240 13.8 MiB 0.0 MiB expansion = getAvailableFrontier(Graph, current.getCor()) |
| 241 13.8 MiB 0.0 MiB successorStates = [] |
| 242 |
| 243 13.8 MiB 0.0 MiB for y in expansion: |
| 244 13.8 MiB 0.0 MiB successorStates.append( |
| 245 13.8 MiB 0.0 MiB node(y, path, heuristic\_function\_calculator(Graph.getGoalCoord(), y) + 1 + current.getGcost(), 1 + current.getCost())) |
| 246 |
| 247 13.8 MiB 0.0 MiB for successor in successorStates: |
| 248 13.8 MiB 0.0 MiB cost = successor.getCost() |
| 249 13.8 MiB 0.0 MiB try: |
| 250 13.8 MiB 0.0 MiB index = frontier.index(successor) |
| 251 13.8 MiB 0.0 MiB except Exception: |
| 252 13.8 MiB 0.0 MiB pass |
| 253 13.8 MiB 0.0 MiB z = str(successor.getCor()) |
| 254 13.8 MiB 0.0 MiB if z not in visited and successor not in frontier: |
| 255 13.8 MiB 0.0 MiB heapq.heappush(frontier, successor) |
| 256 # tmp = copy.deepcopy(successor) |
| 257 # frontier.append(tmp) |
| 258 # SEE MY NOTES FOR THIS |
| 259 13.8 MiB 0.0 MiB elif successor in frontier and frontier[index].getCost() < cost: |
| 260 13.8 MiB 0.0 MiB frontier[index].setCost(cost) |
| 261 13.8 MiB 0.0 MiB frontier[index].setParent(successor.getParent()) |
| 262 13.8 MiB 0.0 MiB for x in visited: |
| 263 13.8 MiB 0.0 MiB if x == successor.getCor() and visited[x] > cost: |
| 264 newNode = node(x, path, visited[x]) |
| 265 heapq.heappush(frontier, newNode) |
| 266 |
| 267 13.8 MiB 0.0 MiB visited[currentCoord] = current.getCost() |
| 268 print("Visited size is ", visited.\_\_sizeof\_\_()) |
| 269 return None |

**Number of nodes visited in A\* algorithm:** 21

[[2, 3], [3, 3]]

[[2, 4], [3, 4]]

[[2, 5], [3, 5]]

[[2, 6], [3, 6]]

[[2, 7], [3, 7]]

[[4, 7]]

**Run Time of A\*:** 0.002992391586303711

**COMMENTS ON MEMORY AND TIME:**

As can be seen from the output of one sample program, memory consumptions of both algorithms are almost the same. Although I tested in many and put here only one output for convenience, in the other tests, I observed that UCS and A\* Search have almost the same memory consumption for even different boards.

However, sometimes, A\* has slightly less memory consumption. That little relative difference comes from the total number of nodes visited. Since in A\* search, we use an admissible heuristic function, search is not blind, and we have an idea of goal. That helps us to move more towards the goal and makes it (mostly) more efficient than UCS. In my sample cases, it is not that visible for memory since my boards are not that big to see a considerable difference in both algorithms.

As for time, situation is a little different. However, the reason is almost the same. Heuristic function makes the algorithm better and faster. Although both are fast to the human eye, UCS is almost always slower than A\*. (in the case I presented above around 1.5 times slower.)

If you change the board, of course, times will be different since the complexity will change (possibly increase) and it will take more time to find both, however, oftentimes we will observe that A\* takes less time than UCS.