Can Gözpınar 68965

Q1) DFS is complete whenever cycles are prevented whereas, BFS is always complete assuming that a finite solution exists. Optimality wise, DFS doesn't always give the most optimal lowest costing path whereas, BFS is optimal if the costs are equal since, you dind solution at the shallowest depth. DFS has time complexity of O(b^m) and BFS has O(b^s) (m is the maximum s is the shallowest) so, DFS is more preferable over BFS whenever, space is a issue and optimality of the solution is not a major concern since, DFS has linear space complexity and BFS has exponential space complexity. BFS is prefered over DFS whenever, space is not a concern and optimality and time complexity are a major concern. In my code also, DFS expanded fewer nodes quickly and didn't guarantee optimality of the solution whereas, BFS expanded more nodes but guaranteed optimal solution.

Q2)UCS is slow and steady but gives the optimal solution. By combining pro's of UCS with greedy first search's fast aspect we sort of get A* search. A* search expands both the low cost and the promising nodes using heuristics.A* expands fewer nodes and faster than UCS. A* should be used whenever we have some sort of understanding of the domain so that, developer can come up with heuristics to reduce number of nodes expanded.UCS can be prefered if a viable heuristic can not be formulated or space and time complexity is not a major issue.

Q3) my state kept track of the visited and unvisited corners. Since, the goal was to visit all the nodes. By looking at my state I was able to deduct which corners were already visited up to that point given the path and which corners were not. Also by immediatly knowing which corners were not visited I was able to create a heuristic regarding the cumulative mazeDistance to those unvisited corners with using a few lines of if statements.

Q4) At first, I tried using cumulative manhattan distances of the pacman's coordinates to all of the unvisited corners up to that state. This helped me to reduce the nodes expanded but in order to reduce it more, I decided to use mazeDistance() method which was supplied to us in the already existing pacman code. Since, this method calculated "real" shortest distance meaning that it took walls into consideration this was much more accurate and improved my heuristics. This is admissible and consistent since, all consistent heuristics are admissible as well and because h(A)- h(c) <= cost(A to C) always holds in my case.

Q5) I combined two parameters for better heuristics. First I took the distance of the state's coordinates to the furthest food coordinate. This is admissible on its own since heuristics can increase or decrease between two adjacent state's max one unit. Next I took the state's pacman coordinates having a food at that state as a criterion which meant that at that state a food was eaten which directly reflected our ultimate goal of eating all the dots. This was admissible as heuristics on its own since its heuristic value differed between the value of one and zero since this is lower bound by the cost of moving to an adjacent state it was admissible. Next I combined these two and using an if else statement made sure the heuristic value I passed could differ max one unit which made it admissible since it fits the requirement of consistency h(A) -h(C) <= cost (A to C).

Q6) inadmissable heuristics may find solutions faster by expanding fewer nodes but it doesn't guarantee optimality therefore, the solution may be faster but it does not mean it has to be the best optimal path. Consistent heuristics may expand more nodes and work slower but, it guarantees optimality. Also, finding consistent heuristics may be harder to find than an inadmissible one too. Prefer inadmissible herusitics when optimality of the solution is not as important than finding a working solution fast. Consistent heuristics can be preferred whenever, optimality of the solution is a major concern and being slow, expanding more nodes is not an major concern.