CS404 ASSIGNMENT 1: SEARCH

Can Korkmaz

Start State: Start state is the user entered input, representing a matrix of balls, number of empty and full tubes are also user inputs. 0 in the matrix correspond to empty spots, and positive integers represent colours of the balls.

States: I modelled the states for the ball sorting program as objects of a class named *State,* consisting of 4 and 5 attributes for Uniform Cost Search and A* Search implementations, respectively. Two of the attributes for each of the implementation can be neglected, that are expandth_num and level attributes, and are not used for decision making.

Attributes:

state: A nested list variable of integers. Positive integers correspond to a colour, and 0 value correspond to an empty slot.

cost: An integer type variable, representing the cost of the path from currentState to that State. For both UCS and A* Search algorithms, path cost from any state to its successor states is 1.

Heuristic cost: For A* Search, representing an heuristic calculation of estimated cost to the goal State from that State. Calculated using heuristic() and tubeHeuristic() functions. tubeHeuristic() funciton calculates the cost of a single tube, and heuristic() function calls tubeHeuristic() for each tube in *state* attribute. Heuristic cost for single tube is calculated by looking at the bottom ball colour and starts counitng the cost after founding the first discrepancy from the bottle. Each coloured ball after and including first discrepancy adds cost of 2 to the cost summation, and one for each empty spot in a tube. This heuristic function is found to be a very efficient and good heuristic function, significantly decreasing the running time of the A* Search compared to the Uniform Search Cost program.

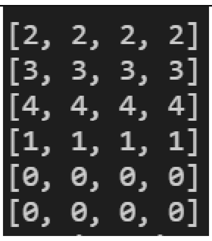
Successor State Function: Successor state function finds out the potential successor states and add them to the frontier list. Then a check is implemented for choosing the smallest cost State inside the frontier. Choosing the smallest State is implemented in two differing ways. UCS algorithm only checks the cost attribute, and the A* Search also checks the heuristic cost of that state and add it to the cost, then do the comparisons for finding the smallest total cost State.

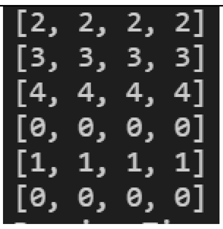
Goal Test: Goal test is the check procedure consisting of checks for each tube in the State, which is done by the isSorted() function, which calls another function isHomogenous() for checking balls in a tube, returns true if all balls are identical in a tube with 4 balls.

RESULTS

Uniform Cost Search:

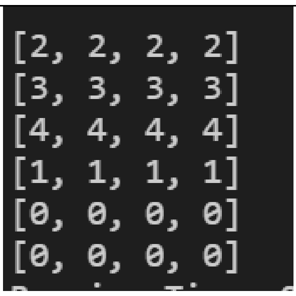
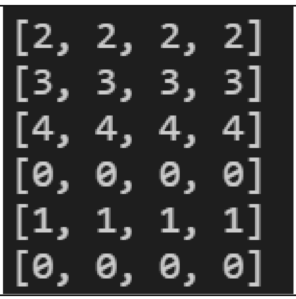
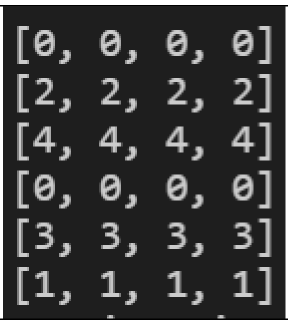
INPUT		OUTPUT	RUNNING TIME

4 2
2 2 2 1
3 3 3 2
4 4 4 3
1 1 1 4
		
		`[2, 2, 2, 2]`
		`[3, 3, 3, 3]`
		`[4, 4, 4, 4]`
		`[1, 1, 1, 1]`
		`[0, 0, 0, 0]`
		`[0, 0, 0, 0]`
			Running Time: 0.13503003120422363

4 2
2 2 2 1
3 3 3 2
4 4 4 3
1 4 1 1
		
		`[2, 2, 2, 2]`
		`[3, 3, 3, 3]`
		`[4, 4, 4, 4]`
		`[0, 0, 0, 0]`
		`[1, 1, 1, 1]`
		`[0, 0, 0, 0]`
			Running Time: 16.10819697380066

4 2 2 1 2 2 2 3 3 3 4 4 4 3 1 4 1 1		Didn't get a result, program kept on running for long durations of time without an output.	NaN

A* Search:

INPUT		OUTPUT	Running Time
4 2 2 2 2 1 3 3 3 2 4 4 4 3 1 1 1 4		 [2, 2, 2, 2] [3, 3, 3, 3] [4, 4, 4, 4] [1, 1, 1, 1] [0, 0, 0, 0] [0, 0, 0, 0]	Running Time: 0.12007880210876465
4 2 2 2 2 1 3 3 3 2 4 4 4 3 1 4 1 1		 [2, 2, 2, 2] [3, 3, 3, 3] [4, 4, 4, 4] [0, 0, 0, 0] [1, 1, 1, 1] [0, 0, 0, 0]	Running Time: 0.45609450340270996
4 2 2 1 2 2 2 3 3 3 4 4 4 3 1 4 1 1		 [0, 0, 0, 0] [2, 2, 2, 2] [4, 4, 4, 4] [0, 0, 0, 0] [3, 3, 3, 3] [1, 1, 1, 1]	Running Time: 19.083547353744507

Result Analysis:

Results clearly display a lower running time for the A* Search algorithm when compared to the UCS algorithm for ball sorting problem. The smallest difference is observed the first input, which only showed a difference of 0.15 second between A* and UCS algorithms, higher running time is observed with UCS algorithm. Second test was done with an input that was more complex for both of the algorithms to sort, and a big difference is observed for the running times of the two algorithms. UCS algorithm took 16.1 seconds, while A* Search only took 0.45 second, 36 times faster than the UCS algorithm. The third test was done with an even more complex to sort input, and the UCS algorithm didn’t give an output within a given 3-5 minute wait time, while A* Search with its intelligently designed heuristic function managed to sort the input in only 19 seconds. These differences in performance were expected since A* Search also used a heuristic function for decision making, and it showed a huge gain in performance.