

Casey Easter, Justin Roberts

Dr. Pears – 5210 – Project 2

10/19/21

Intro:

In the development of this project, we were able to explore new methods of data storage, how to interact with said structures, and most relevant to the prompt; how to use a uniformed search method (IDS) to find the data in the order that we want.

Part A

1. From the map given in the prompt, we concluded that a binary search tree data structure would be best suited for minimizing movement between divisions. A binary search tree orders nodes in a hierarchal fashion, which is exactly what we need for our divisions. Each node has a key value, in this case the division number (1-15) as well as values pointing to the child nodes.
2. The proposed data structure is populated by a function that generates another tree within a specified node. This function uses a loop to iteratively create a left-child node and a right-child node for all applicable parent nodes in the range. These values are stored in ordered lists that are made available to other functions by their indexed values. (Eg. Parent node at: x, has Child nodes y & z)

The Approach

Our program is organized into 4 python files: agent.py, main.py, project_utils.py, and warehouse.py. Each of these files has some degree of dependency on another. Two files, project_utils.py and warehouse.py make use of the predefined libraries “math” and “random”. In file Warehouse.py we defined the divisions 1-15, populated the divisions with shelves (1-63) and created our order generator.

3. Our binary tree representing the divisions in the warehouse, is populated by a user defined function “generate_tree”. This function uses a while loop to iteratively create values for parent and child nodes and then append the values to separate lists designated for parent values and child values.

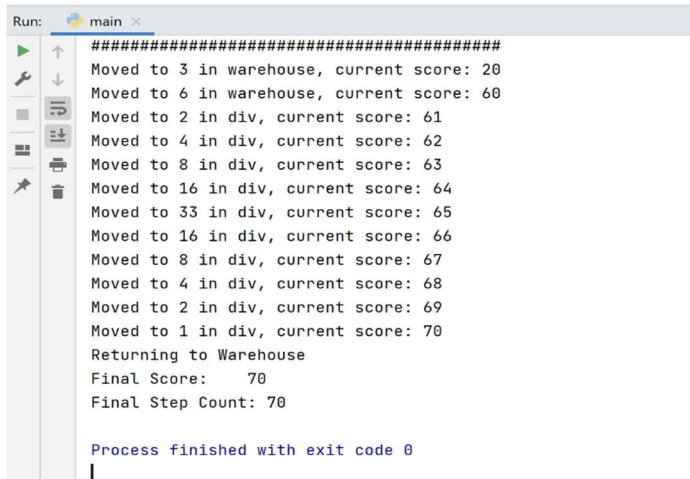
The agent.py file is where the actual IDS search method is implemented. The file starts by defining an “agent” class and initializing variables for score tracking, step count, and minimum and maximum path values. The first defined function within the agent handles the primary outline of a session. Next in the file, we have a warehouse search function that finds the location of a division within the warehouse, and a division search function that finds desired shelves within a division. Following those functions, is the movement function, responsible for moving the agent and keeping track of its path cost in the division.

4. The last remaining functions in the agent class are responsible for the execution of the iterative deepening search. The first is the “id_search” function which is used to iteratively increase the depth at which nodes will be expanded. Next, is the “dl_search” which implements the elements of a depth first search, expanding nodes on the frontier at

the given level. Lastly, the “expand” function *expands* the desired nodes on the frontier in a breadth first fashion.

The “project_utils.py consists of four more functions, all being *utilities* that the agent can use to interact with the environment. The “path_trace” function gives the agent the ability to traceback to the root node after completing an order. The “path_merge” function combines paths that converge at a common node when tracing back to the root node. The following function “find_common”, which uses an if-else branch to compare nodes and determines if they are the same, is what allows the previous function to properly merge paths on the traceback. The last function used in this file is the “path_forger” function, which as the name implies makes a path for the agent to follow. The function creates a priority que based on step length and uses this information to create a path from the agent’s current location to the goal node.

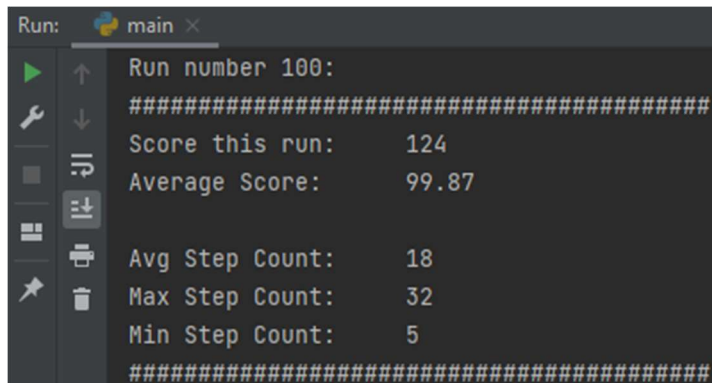
5.



```
Run: main x
#####
Moved to 3 in warehouse, current score: 20
Moved to 6 in warehouse, current score: 60
Moved to 2 in div, current score: 61
Moved to 4 in div, current score: 62
Moved to 8 in div, current score: 63
Moved to 16 in div, current score: 64
Moved to 33 in div, current score: 65
Moved to 16 in div, current score: 66
Moved to 8 in div, current score: 67
Moved to 4 in div, current score: 68
Moved to 2 in div, current score: 69
Moved to 1 in div, current score: 70
Returning to Warehouse
Final Score: 70
Final Step Count: 70

Process finished with exit code 0
```

6.



```
Run: main x
Run number 100:
#####
Score this run: 124
Average Score: 99.87

Avg Step Count: 18
Max Step Count: 32
Min Step Count: 5
#####
```

Conclusion

The iterative deepening search method is an extremely useful tool when looking to utilize the best aspects of a depth-first and breadth-first search. In the case of our search agent, IDS allowed us to efficiently find our goal state in each instance and minimize path cost in the process.