AI 1 Homework 2 Report

Olivia Guess, Melissa Tully

Implementation

To implement each search algorithm, we created several helper functions and classes. We created a node class that describes each node of a search tree or graph, include parents/children, vacuum location, dirt locations, cost, depth, state, and next possible actions. insert_node and new_rooms_node creates a node from a given parent node and from a specified action. A Tree class was also created to store the root node, which describes the initial environment.

Overall Program

We created classes for nodes, a minimum queue and trees for our implementation. Each node has many fields indicating its state, including the vacuum and dirt locations, the current cost, possible actions, parent node, depth, and children. The state information for the vacuum and dirty squares are simply kept as ordered pairs as a list type.

The function new_rooms_node is really important to making a new node. It determines the possible actions the vacuum can take from the given state.

Iterative Deepening Tree Search

This search algorithm has three different functions associated with it - the main function, iterative_deepening_tree_search, and two helper functions, idts_expand and idts_sol.

The main function follows the pseudocode closely. It uses various variables to keep track of the necessary statistics.
Idts_expand generates new nodes based on the possible actions of the current state node and adds them to the fringe in smallest coordinate order.
Idts_sol prints out the statics for the search algorithm once the goal is found for a given instance.

Uniform Cost Tree Search

For the uniform cost tree search algorithm, a minimum queue was created for the fringe. Every insertion places that node into sorted order from least to greatest. The initial node is placed into the queue first and is then popped off and expanded. It will expand each possible action of that node and add those nodes to the fringe. This is done repeatedly until the goal state is reached (all rooms are clean), until the time has passed an hour, or there are no more items in the fringe (meaning a solution was not found).

Uniform Cost Graph Search

For this algorithm, a minimum queue is used for the fringe and an array is used to store closed nodes. This means that a state (a pair of vacuum location and dirt locations) cannot be visited twice. So once a node has been expanded, it will be added to the closed nodes. Then, when looking to expand a node, the algorithm will check to see if the state has already been visited or not. If it has not, then the node will be added to the fringe. The minimum queue is used to find uniform cost, so that the next expanded node is the node with the least cost.

Programming Language, Hardware

Language: Python v3.9
Hardware used was MacBook Pro and a Dell laptop

Both instances of Iterative Deepening Tree Search were run on a Dell laptop with an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz

All 4 Uniform Cost searches were done on the MacBook Pro

Results

Instance 1

Result	**Uniform Cost Tree Search**	**Uniform Cost Graph Search**	**Iterative Deepening Tree Search**
1st 5 Exp. Nodes	Start(no action), Suck, Down, Up, Right	Start, Suck, Down, Up, Right	Initial state, Initial state, Up, Left, Suck
No. Nodes Expanded	75494	76213	507937
No. Nodes Generated	1441976	1454723	1185779
CPU Execution TIme (seconds)	4974.9	3600.0	363.3
Solution	Start, Up, Suck, Right, Right, Down, Suck, Down, Right, Suck	Start, Up, Suck, Right, Right, Down, Suck, Down, Right, Suck	Start, Up, Suck, Right, Right, Down, Suck, Right, Down, Suck
No. of Moves	9	9	9
Cost of Solution	6.7	6.7	6.7

Instance 2

Result	Uniform Cost Tree Search	Uniform Cost Graph Search	Iterative Deepening Tree Search
1st 5 Exp. Nodes	Start, Suck, Down, Up, Right	None, left, up, right, right	None, Up, Left, Suck, Right, Down
No. Nodes Expanded	58781	148386	2332722
No. Nodes Generated	1116384	2893528	5159508
CPU Execution TIme (seconds)	3959.2	3600.0	36
Solution	Start, Left, Up, Suck, Up, Right, Right, Down, Down, Suck, Right, Up, Suck	Start, Left, Up, Suck, Up, Right, Right, Down, Down, Suck, Right, Up, Suck	Start, Right, Suck, Up, Right, Suck, Up, Left, Left, Suck, Left, Down, Suck
No. of Moves	12	12	13
Cost of Solution	9.3	9.3	9.5