FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE - CS161

Programming Assignment 2 - Due 11:59 p.m., Tuesday, January 22, 2019

DO NOT CONSULT ANY OUTSIDE REFERENCES OR SOURCES FOR THIS PROBLEM SET

The problem is to implement several brute-force search algorithms, including depth-first, breadth-first, and depth-first iterative-deepening. The search trees will be represented as lists in which a leaf node is represented by an atom, and a non-leaf node is represented by a list of its child nodes. For example, the list ((W X) (Y Z)) represents the complete two-level binary tree shown below on the left, and the list ((A (B)) C (D)) represents the tree shown below on the right.

1. Write a single pure LISP function, called DFS, that performs a depth-first search of a tree. The function should take a single argument that is the list representation of the tree, and return a single, top-level list of the terminal nodes in the order they would be visited by a left-to-right depth-first search. For example, (dfs '((A (B)) C (D))) would return (A B C D). Do not use any auxiliary functions.

2. Write a set of pure LISP functions that implement depth-first iterative-deepening. Your top-level function, called DFID, should take two arguments, the list representation of the tree, and an integer representing the maximum depth of the tree, and return a single top-level list of the terminal nodes in the order that they would be visited by a left-to-right depth-first iterative-deepening search. Note that those nodes that are visited in multiple iterations will appear multiple times in the output list. For example, (dfid '((A (B)) C (D)) 3) would return (C A C D A B C D).

3. Write a single pure LISP function, called BFS, that performs a breadth-first search of a tree. The function should take a single argument that is the list representation of the tree, and return a single, top-level list of the terminal nodes in the order they would be visited by a left-to-right breadth-first search. For example, (bfs '((A (B)) C (D))) would return (C A D B). Do not use any auxiliary functions.

All your programs must work for trees of arbitrary depth and branching factor, and hence you may not assume any a priori upper bound on these parameters. Be sure to experiment with sufficient test cases to convince yourself that your programs work in general.