FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE - CS161
Spring 2020

Programming Assignment 2 - Due 11:55pm Tuesday, April 14

Homework Guidelines:

- Submit your commented LISP program in a file named **hw2.lsp** via **CCLE**.
  NOTE: There will be 10% credits penalty per hour delay.

- Your programs will be evaluated under CLISP interpreter. In order to get any scores, you need to make sure that the following LISP command does not produce any errors in CLISP interpreter:

  > (load "hw2.lsp")

- Your programs should be written in **good style**. In LISP, a comment is any characters following a semicolon (;) on a line. Provide an overall comment explaining your solutions. Furthermore, every function should have a header comment explaining precisely what its arguments are, and what value it returns in terms of its arguments. In addition, you should use meaningful variable names.

- You are restricted to using the following functions, predicates, and operators introduced in class: quote ['], car, cdr [cadadr, etc.], first, second [third, etc.], rest, cons, list, append, length, numberp, stringp, listp, atom, symbolp, oddp, evenp, null, not, and, or, cond, if, equal, defun, let, let*, =, <, >, +, -, *, /. Note: you are <u>not</u> permitted to use setq.

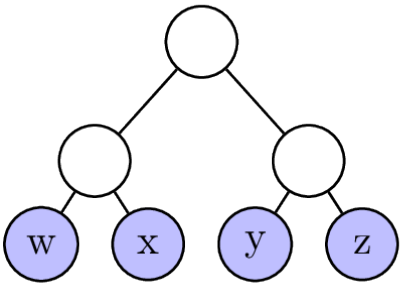
- You may assume that all input to your functions are legal; i.e. you do not need to validate inputs.

- Do not write any additional helper functions for your code unless this is explicitly allowed.

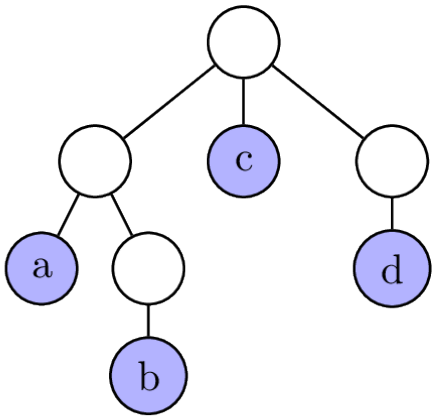
- Your function declarations should look exactly as specified in this assignment. Make sure the functions are spelled correctly, take the correct number of arguments, and those arguments are in the correct order.

- Even if you are not able to implement working versions of these functions, please include a correct skeleton of each. Some of these assignments are auto graded and having missing functions is problematic.

The first three problems are to implement three brute-force search algorithms: breadth-first, depth-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.



((w x) (y z))



((a (b)) c (d))

The fourth problem is to implement a depth-first iterative-deepening solver for the missionary-cannibal problem discussed in class using a code skeleton.

1. 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 returns 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.

2. 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 returns a single, top-level list of the terminal nodes in the order they would be visited by a **right-to-left** depth-first search. For example, (dfs '((A (B)) C (D))) would return (D C B A). Do not use any auxiliary functions.

3. 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 returns 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).

Each of these functions 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 exhibit sufficient test cases to convince yourself and us that your programs work in general. Try at least the examples above, as well as the list (A (B C) (D) (E (F G))).

4. Implement a depth-first solver for the missionary-cannibal problem that was described in class. To implement this solver, we provide you with a code skeleton (hw2_skeleton.lsp). Implement the functions in the code skeleton as described in their associated comments. DO NOT CHANGE THE FUNCTION NAMES OR PARAMETERS. Also do not write any additional functions.