Quiz 5b

1. (2 points) Which would be faster, depth first search or breadth first search or either, for finding the way through a maze if a path is picked on random whenever there is a fork (on average).

Depth first search is faster. Breadth first search would start searching through the paths once a fork is reached. Depth first search searches through one whole path until it reaches a dead end. Breadth first search would end up going through a larger portion of the maze before finding the end. In retrospect, this question was worded very poorly however, and if they explain their answer, give them partial or full depending on how compelling their argument is.

1. (2 point) Consider the following procedure for binary trees.

(define (mystery tree)

(if (equal? tree the-empty-tree)

the-empty-tree

(make-tree (max (entry tree)

(if (equal? (right-branch tree) the-empty-tree)

(entry tree)

(entry (right-branch tree)))

(if (equal? (left-branch tree) the-empty-tree)

(entry tree)

(entry (left-branch tree))))

(mystery (left-branch tree))

(mystery (right-branch tree)))))

What would the output tree look like if performed on the tree below?

0 2

/ \ / \

1 2 4 6

/ \ / \ / \ / \

3 4 5 6 8 4 5 6

/ \ / \

7 8 10 8

/ \ / \

9 10 9 10

1. (1 point) Consider the binary tree that stores anything smaller than its nodes on the left side and anything larger than its nodes on its right side. As a number is added to it, that number moves along the correct side and becomes the leaf of the node closest to the root while following the rule of the tree. Here’s what the tree would look like if we inserted the numbers in a set in this order: 5, 2, 6, 15, 9, 10, 20

5

/ \

2 6

\

15

/ \

9 20

\

10

Now what would it look like in this order: 10, 6, 9, 2, 20, 5, 15

10

/ \

6 20

/ \ /

2 9 15

\

5

1. (2 points) Suppose we had a tree with nodes that kept a record of its depth from the root as its value. Design a procedure using filter, enumerate-tree, accumulate, and/or map that finds the depth of the leaf farthest from the root. You can also use recursion but it will take you longer to write. Hint: max might be useful depending on your implementation.

0

/ \

1 1

/

2

/ \

3 3

(define (deepest-leaf tree)

(accumulate max (enumerate-tree tree))

OR

(define (deepest-leaf tree)

(find-max (enumerate-tree tree) 0))

(define (find-max lst start)

(cond ((empty? lst) start)

((> (first lst) start) (find-max (bf lst) (first lst)))

(else (find-max (bf lst) start)))

They can either recursively go through the tree to find the max or use enumerate tree to find a list of leaves and write a simple max function.

1. (3 points) Define a procedure that searches a tree for a number and returns the depth of the node with that number. Root is at depth 0 and assume that there are no doubles in the tree. Return #f if the number is not in the tree.

1 depth = 0

/ | \

3 7 5 depth = 1

/ | \

2 4 8 depth = 2

Using the procedure on a tree like this returns 2.

(define (depth-of n tree)

(define (depth-of-helper n tree d)

(cond ((equal? n (datum tree)) d)

((null? (children tree)) #f)

(else (forest-depth-of n (children tree)(+ d 1)))))

(define (forest-depth-of n forest d)

(if (null? (cdr forest))

(depth-of-helper n (car forest) d)

(or (depth-of-helper n (car forest) d)

(forest-depth-of n (cdr forest) d))))

(depth-of-helper n tree 0))

Given the time constraints, the code doesn’t have to be fully functional, just look for these 3 points as a rough guideline

Mutual recursion is +1 point

Having the “or” is +1 point

If the structure or logic looks right in general +1 point