1. Order visited in Breadth-First Traversal:

A-B-E-C-D-F-G-K-J-H

2. Order visited in Depth-First Traversal:

A-B-C-D-J-H-K-E-G-F

3. No, a Depth-First Search (DFS) is not guaranteed to find the optimal shortest path. A DFS may miss a 'shortcut'. In the following example, a DFS from A to H will find the path A-B-D-F-G-H, rather than A-C-H:

A

/ \

B C

/ \ |

D E -------H

\ / |

F-----------G

4. a. 1

/ \

2 3

/ \ / \

4 5 6 7

/ \ / \ / \ / \

8 9 10 11 12 13 14 15

4.b. BFS: 1-2-3-4-5-6-7-8-9-10-11

DFS: 1-2-4-8-9-10-11

Iterative: 1-1-2-3-1-2-4-5-3-6-7-1-2-4-8-9-5-10-11

5.a. Variables: # missionaries on starting side, # cannibals on starting side, boat position

Number of state spaces = 4 \* 4 \* 2 = 32 (not all are valid)

Number of valid state spaces: 26 (so cannibals do not outnumber missionaries while 'unattended')

5.b.

currentState = (Int numMissionaries, Int numCannibals, boolean boat)

//values represent starting side of river: ex. (3,3,1) is beginning state

def successorFunction(currentState):

list possibleSuccessorStates

if (boat) //boat is present, so people leaving

newStateA = (numMissionaries -2, numCannibals, !boat)

newStateB = (numMissionaries - 1, numCannibals -1, !boat)

newStateC= (numMissionaries -1, numCannibals, !boat)

newStateD= (numMissionaries, numCannibals -1, !boat)

newStateE = (numMissionaries, numCannibals - 2, !boat)

else if (!boat) //boat is absent, so ppl coming

newStateA = (numMissionaries +2, numCannibals, !boat)

newStateB = (numMissionaries +1, numCannibals +1, !boat)

newStateC= (numMissionaries +1, numCannibals, !boat)

newStateD= (numMissionaries, numCannibals +1, !boat)

newStateE = (numMissionaries, numCannibals +2, !boat)

if (isValid(newStateA,B,C,D,E)==true: //check each possible newState

possibleSuccessorStates.append(newStateA,B,C,D,E) //add to list if valid

return possibleSuccessorList

def isValid(possibleState) //determines if possibleState is valid by comparing numCannibals, numMissionaries, and boat position

if (boat)

if (numMissionaries ==3 || numMissionaries == 0) return true

if (numMissionaries == 2 & numCannibals >=2) return true

if (numMissionaries == 1 & numCannibals >=1) return true

if(!boat)

if (numMissionaries == 3 || numMissionaries == 0) return true

if(numMissionaries == 2 & numCannibals <=2) return true

if(numMissionaries == 1 & numCannibals <=1) return true

else return false

notes for 5.b:

each state is made up of 3 components, number of missionaries on the starting bank, number of cannibals on the statring bank, and a boolean indicating the presence or absence of the boat on the starting bank.

Each possible successor state is arrived at by one of ten possible operations:

cannibals +-2; missionaries +-2; cannibals and missionaries +-1; cannibals +-1; and missionaries +-1. Every successor state changes the boat boolean.

Valid states are those where the cannibals do not outnumber missionaries on the side without the boat. If the boat is present, cannibals may outnumber missionaries on that side

5.c. starting condition = (3,3,1) (3 missionaries, 3 cannibals, 1 boat on starting side)

(2,2,0) (3,1,0) (3,2,0)

(3,3,1) (3,2,1) (2,3,1) (3,3,1) (3,2,1) (3,3,1)

\*valid states only

6. A DFS from A to H will find the path A-B-D-F-G-H, rather than A-C-H:

A

/ \

B C

/ \ |

D E -------H

\ / |

F-----------G