**CSCI 204 Binary Tree Activity 1**

Student name(s)\_\_\_\_\_Suggested Solution\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Find two real life examples in which the binary tree data structure is appropriate. Briefly discuss reasons and why it is best to use the binary tree data structures, not other linear structures such as list, queue, or stack.
2. Given following binary tree we discussed in our lecture notes (Lecture 26: Introduction to Binary Trees)



* 1. Manually “traverse” the tree with the help of a stack, following the algorithm given below. Write out what’s been printed.

Initialize the stack S to be empty

put the root (A in this case) on S

while S is not empty:

 n = S.pop()

 print(n)

 for all children c of n:

 S.push( c)

Assume the right side of the stack is the top of the stack

s= stack()

s.push(A) s = [A]

n = s.pop() prints ‘A’, adds all ‘A’s children ‘B’, ‘C’ to the stack, S = [B, C]

s.is\_empty() == False

n = s.pop() prints ‘C’, adds all C’s children, ‘F’ and ‘G’ to the stack, S = [B, F, G]

s.is\_empty() == False

n = s.pop() prints ‘G’, adds all G’s children, ‘I’ to the stack, S = [B, F, I]

s.is\_empty() == False

n = s.pop() prints ‘I’, adds all I’s children, None, S = [B, F]

s.is\_empty() == False

n = s.pop() prints ‘F’, adds all F’s children, None, S = [B]

s.is\_empty() == False

n = s.pop() prints ‘B’, adds all B’s children ‘D’, ‘E’ to the stack, S = [D, E]

s.is\_empty() == False

n = s.pop() prints ‘E’, adds all E’s children ‘H’ to the stack, S = [D, H]

s.is\_empty() == False

n = s.pop() prints ‘H’, adds all H’s children, None S = [D]

s.is\_empty() == False

n = s.pop() prints ‘D’, adds all D’s children, None S = []

s.is\_empty() == True

the algorithm stops.

* 1. Do the same, but using a queue as the data structure.

Assume the right side of the queue is the end of the queue

s= Queue()

s.enq(A) s = [A]

n = s.deq() prints ‘A’, adds all ‘A’s children ‘B’, ‘C’ to the queue, S = [B, C]

s.is\_empty() == False

n = s.deq() prints ‘B’, adds all B’s children, ‘D’ and ‘E’ to the queue, S = [C, D, E]

s.is\_empty() == False

n = s.deq() prints ‘C’, adds all C’s children, ‘F’ and ‘G’ to the queue, S = [D, E, F, G]

s.is\_empty() == False

n = s.deq() prints ‘D’, adds all D’s children, None, S = [E, F, G]

s.is\_empty() == False

n = s.deq() prints ‘E’, adds all E’s children, H, S = [F, G, H]

s.is\_empty() == False

n = s.deq() prints ‘F’, adds all F’s children, None, to the queue, S = [G, H]

s.is\_empty() == False

n = s.deq() prints ‘G’, adds all G’s children ‘I’ to the queue, S = [H, I]

s.is\_empty() == False

n = s.deq() prints ‘H’, adds all I’s children, None S = [I]

s.is\_empty() == False

n = s.deq() prints ‘I’, adds all I’s children, None S = []

s.is\_empty() == True

the algorithm stops.