Problem 6.17.a:

Let G be a directed acyclic graph whose nodes labels are represented by a single character. If we string nodes that are connected together it will and combine their labels we will have a word. The goal is to see if there are any palindromes, words that are spelled the same forwards as backwards, in G among the connected nodes. If there is, return the length of the longest palindrome.

Algorithm:

```
#O(n^2)
for node in range(graph) #O(n)
    #O(n)
    -do a depth first search on each node until we reach the sink node
    once we have possible paths for each node to sink node

    #O(n^2)
    -if there are any paths, i.e there is more than one node then:
        -get associated letters in graph_letters with path,
        the letters will be stringed together according to the route
        in each path. So, if nodes 2->3->4 correlated to
        a,b,c. Then we would get the string 'abc'

        #O(n^2)
        -while each word has at least one letter check if word is a
        palindrome. for each loop we will cut the last letter off
        of each word and check for palindrome. The idea behind this
        is that since we are getting words associated to paths in our
        dfs we need to make sure we check for potential palindromes
        in the word. So for example, hannahtgt is not a palindrome
        but if you keep cutting off the end eventually you get
        hannah, which is a palindrome

    #O(n)
    -if there are any palindromes then
        filter out any empty values and then
        return the length of the longest palindrome.

    -else return none
```

Analysis of correctness:

Since we are given a directed acyclic graph we know that there is one source and one sink and that we should be able to reach the sink from any given node in the graph. Therefore if we do a depth first search on each node we will get all possible paths from any given node to the sink.

If we then take the labels for each node and string them together according to the list of paths we find going from each node to the sink we will get a list of words that potentially are palindromes.

We then check each by simply seeing if we get the same word when it's reversed. We do this for each potential word and those shortened by one character until the word is gone. We do this to check subwords in each word for a palindrome. So for example if the word is hannahs and we check to see if it was a palindrome we would find that its not since shannah is not the same as hannahs, but if we drop the s at then we find that the palindrome hannah is in the graph G.

Once we find a palindrome we store it in a list with another found palindrome and return the length of the longest one. As a side note we avoid words that are length one. So, the word ‘a’ would not be counted.

Analysis of runtime:

-Looping through each node in the graph will take n time. (n)

-My depth first search should also take n time since we are recursing down through each node int graph. (n)

-If there are any paths then validating each potential path through G will take n^2. In the my method that checks each path we loop through each path given to the function and then grab its associated letter in our graph containing the node labels. (n^2)

-My while loop that goes until the longest word is cut down to nothing will take n time (n) combined with the two loops inside it, which both are on the same line and run in (n) time. Its final time is (n^2)

-My check to see if a string is a palindrome will take n time as it just loops through the list of words. (n)

-My for loop to cut off the end of each word will also take n time. (n)

-If there are any palindromes my call to remove all empty nodes in my list of palindromes will take n time. (n)

Since my depth first search check, my while loop and my function to get letters are all at the same level that part of the code comes out to (n^2). My loop to remove None from the list is at the same as my main loop and since that just (n) and we add it to my (n^2) loop my run time comes to be (n^2)