LA3main.py:

```
import the oop file


def main():  # Main function
    pathfinder_list, boolean_list, rows, columns = read_data_from_file(Filename)
    passage_seeker = A PathFinder object
    passage_seeker.find_longest_path_length()  # Finds the longest path for the object
    pf_str = format string for display
    print(pf.str.format(rows, cols, longest_path))  # prints the display


# Use open(), readline() and/or readlines() functions to read the following from
# the input file:
# - length and width of the map;
# - the characters to be stored in the map.
# Create the map using the data read from the input file.
# Note: you may need to use strip and split functions.
# The next part is OPTIONAL but might be helpful to use.
# Declare and initialize a Boolean list with similar dimensions to the map; this
# list can be used to keep track of the A's in the input file that have already
# been counted in the path of A's being 'discovered' in the program.
# Parameter to function: input file name.
def read_data_from_file(filename):
    pathfinder_list =  Creates an empty list
    bool_pf_list = Creates an empty list

    with open(filename, 'r') as input_file:  # Opens file and starts reading from it
        first_row = the first line in the file
        m_cols, n_rows = the number of columns and rows given by the first line of the file
        m_cols = turns m_cols into a integer data types
        n_rows = turns n_rows into a integer data types
        for n in range(n_rows):  # iterates through the file for each row
            row_val = Reads in a single line
            row_val = strips unwanted char values off the string
            col_list =Turns string into a list
            Appends col_list to pathfinder_list making a 2d list
            Creates a 2d (bool_pf_list) list of exacts size with all false values

    return pathfinder_list, bool_pf_list, n_rows, m_cols


main()
```

oop.py:

```
# This method determines which of the four values passed to it is the maximum.
# The four values passed represent the path lengths for the four paths of recursive calls from a
specific character
# in the 2D list.
# Function parameters:
# a: The length returned from position [i-1, j].
# b: The length returned from position [i+1, j].
# c: The length returned from position [i, j-1].
# d: The length returned from position [i, j+1].
# Function return: Returns the Maximum of all lengths passed to it.
# '''
def find_max(a, b, c=0, d=0):
    max_val = finds max value out of the 2-4 parameters given
    return max_val


class PathFinder:
    def __init__(self, path_list, bool_list, rows, cols):  # Initializes data attributes needed for the
                                                            # class
        self.__path_list = path_list
        self.__bool_list = bool_list
        self.__rows = rows
        self.__cols = cols
        self.__max_path = 0

    # Resets the boolean list to all false values
    def reset_bool_list(self):
        for n in range(self.__rows):
            for m in range(self.__cols):
                self.__bool_list[n][m] = False

    # Iterate through all the positions in the map (2-dimensional list);
    # at each position, call the recursive method findPathLengthRecursive(), and at
    # each position, update the maximum number of A's in the path so far.
    # Function return: The maximum number of A's in the path.
    # '''
    def find_longest_path_length(self):
        for n in range(self.__rows):
            for m in range(self.__cols):
                self.reset_bool_list()
                position_val = self.find_path_length_recursive(n, m)
                if self.__max_path < position_val:
                    self.__max_path = sets the longest path

    # This method uses recursion to check the cells to the left, right, above and
    # below the current cell to determine if any of these is an 'A' that hasn't yet
    # been counted as part of the longest path of A's.
    # NOTE: Each 'A' in the path should be counted only once.
    # Function parameters:
    # n: The current row.
    # m: The current column.
    # Function return: Return either zero or the current length signifying the number of connected
A's so far.
    # '''
    def find_path_length_recursive(self, n, m):
        if (self.__path_list[n][m] == 'A') and (not self.__bool_list[n][m]):

            self.__bool_list[n][m] = set element to ‘True’ value so recursive statement knows not to
            count the same ‘A’

            # This algorithm uses recursive statements to check every possible path and uses if
            # statements to keep the statements in valid range
            if n == 0 and m == 0:  # if element is the top left one
                path_length = 1 + find_max(self.find_path_length_recursive(n + 1, m),
                                           self.find_path_length_recursive(n, m + 1))
            elif (n == 0) and m == (self.__cols - 1):  # if element is the top right one
                path_length = 1 + find_max(self.find_path_length_recursive(n + 1, m),
                                           self.find_path_length_recursive(n, m - 1))
            elif n == (self.__rows - 1) and m == 0:  # if element is the bottom left one
                path_length = 1 + find_max(self.find_path_length_recursive(n - 1, m),
                                           self.find_path_length_recursive(n, m + 1))
            elif n == (self.__rows - 1) and m == (self.__cols - 1):  # if element is the bottom right one
                path_length = 1 + find_max(self.find_path_length_recursive(n - 1, m),
                                           self.find_path_length_recursive(n, m - 1))
            elif n == 0:  # if element is in the top row
                path_length = 1 + find_max(self.find_path_length_recursive(n, m - 1),
                                           self.find_path_length_recursive(n + 1, m),
                                           self.find_path_length_recursive(n, m + 1))
            elif m == 0:  # if element is in the first column
                path_length = 1 + find_max(self.find_path_length_recursive(n + 1, m),
                                           self.find_path_length_recursive(n - 1, m),
                                           self.find_path_length_recursive(n, m + 1))
            elif m == (self.__cols - 1):  # if element is in the last column
                path_length = 1 + find_max(self.find_path_length_recursive(n + 1, m),
                                           self.find_path_length_recursive(n - 1, m),
                                           self.find_path_length_recursive(n, m - 1))
            elif n == (self.__rows - 1):  # if element is in the bottom row
                path_length = 1 + find_max(self.find_path_length_recursive(n - 1, m),
                                           self.find_path_length_recursive(n, m - 1),
                                           self.find_path_length_recursive(n, m + 1))
            else:
                path_length = 1 + find_max(self.find_path_length_recursive(n + 1, m),
                                           self.find_path_length_recursive(n - 1, m),
                                           self.find_path_length_recursive(n, m - 1),
                                           self.find_path_length_recursive(n, m + 1))
            return path_length
        else:
            # Base Statement returns 0 if no more A’s to be counted
            return 0

    # Returns max_path data attribute to caller
    def get_max_path(self):
        return self.__max_path
```