Pre-lab Part 1

1.

```
while (fscan %c%c\n != E0F){
        check if its alphabet;
        check if its uppercase;
        make it into its respective number depending on the options
        matrix[row][col] = 1;
        if undirected{
                matrix [col][row] = 1;
        }
}
```

2. A -> B -> C -> F -> Z
   A -> B -> C -> Z
   A -> B -> D -> E

3. The worst case complexity of the post order traversal tree would be it going through the whole tree and not finding an answer, and that happening until the end. In general we would say that the worst case complexity would be finding it in 2^n, n being the number of inputs. Using the assignment, for example, the worst case complexity would going be to go down the whole alphabet for you to find the solution, or really late into the algorithm. For the example above, the worst case would be that we found the z paths at the bottom, after we looked at all the other paths.

Pre-lab Part 2

1.

```
Stack create{
        stack malloc size of stack
        capacity = minimum
        stack->top = 0;
        stack->items = calloc(minimum, size of items);
        return stack;
}
Stack delete {
        free (stack->items);
}
Stack empty {
        check if top == 0;
}
Stack size {
        return top of stack = size;
}
Stack push {
        checks if stack top = capacity
                if it is, add more stack space
        stack->items top = item
        increment
}
Stack pop {
        check if its empty first
        if not, take one out from top and decrement top
}
```
Stack print: prints from index 0 to top

MAIN:
- uses switch case for all the options
- default for *fp is stdin, unless user inputs file, default is undirected as well
- make a boolean array to correspond with each junction to mark if it has been visited or not
- after calling dfs, need to clear the stack
- print number of paths and shortest path

read_file:
- this is to read the file given, whether it is user inputted or given file
- parameters is a file pointer and the matrix
- scan line by line with scanf, checking if it is uppercase and in the alphabet, since you need to subtract different values to make the char to int, depending on case and if it is in alphabet
- once converted, do matrix[row][col]
- if user wants undirected, do matrix[col][row] too
- pseudocode is in pre-lab part 1

print_matrix:
- takes matrix as parameter
- prints first row of letters using for loop
```
        for ( i = 0; i < vertices; I++){
                print i + upcase and turn to char
        }
```
- loop that prints out each row separately
```
        for ( i = 0; i < vertices; i++){
                print row number
                loop that print column values of that row
        }
```

dfs
- takes a stack and a junction as a int (A = 0)
- once dfs is called with junction, it checks if it is the end or not
- change the visited status to true and push the junction to the stack, indicating we are checking this path
- the current junction needs to check every column if it has a 1
```
        for ( i = 0; i < vertices; i++){
                junction is row, i is column
                if matrix value has 1 and it has not been visited
                        call dfs with the column, that becomes the new junction to search
                        once dfs is done with that junction, pop from stack to get off that
path
```

```
          }
```
- after the for loop is done, we need to change the visited status of the junction to false so that when we go on a new path, we can visit that specific junction