Pre-lab Part 1

1. if (input file given):

        get input_content from input file

   else:

        get input_content from stdin

   for (i = 0; i not at end of input_content; i++):

        if i != line break:

            input_pair[2] = array of characters before \n

        else:

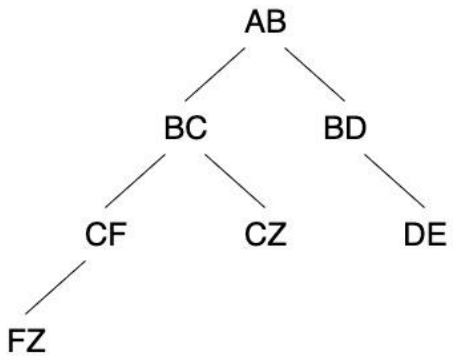
            if (!directed):

                matrix[input_pair[0] – ‘A’][input_pair[1] – ‘A’] = 1

                matrix[input_pair[1] – ‘A’][input_pair[0] – ‘A’] = 1

            else:

                matrix[input_pair[0] – ‘A’][input_pair[1] – ‘A’] = 1

2. 

   Order edges taken: AB, BC, CF, FZ or AB, BC, CZ or AB, BD, DE, backtrack, BC, CF, FZ or AB, BD, DE, backtrack, BC, CZ

3. The worst case complexity is having to go three deep on the tree.

Pre-lab Part 2

// From assignment PDF

```
typedef struct Stack {
	uint32_t *items;
	uint32_t top;
	uint32_t capacity;
} Stack;

// Derived from stack lecture slides
Stack *stack_create(void) {
	minimum = 32
	s = allocate memory based on size of stack
	s -> top = 0
	s -> capacity = minimum
	allocate memory for items based on size of item * capacity
}

void stack_delete(Stack *s) {
	for (*s, *s > 0, (*s)--) {
		free memory at *s
	}
	return void

bool stack_empty(Stack *s) {
	if s->top == 0 {
		return true
	} else {
		return false
	}
}

uint32_t stack_size (Stack *s) {
	return s->top
```

```
}

// Derived from lecture slides
bool stack_push (Stack *s, uint32_t item) {
	if (s->top == s->capacity) {
		increase s-> capacity
		reallocate more memory
	}
	s->items[s->top] = item
	s->top++
	return true
}

// Derived from lecture slides
bool stack_pop (Stack *s, uint32_t *item) {
	if (stack_empty) {
		return false
	} else {
		s->top--
		*item = s->items[s->top]
		return true
	}
}

void stack_print (Stack *s) {
	for (s->top, s->top > 0, s->top--) {
		print(s->items[s->top])
	}
	return void
}
```

Program Design:

Write code for the stack based on the pseudocode above.

```
path = stack_create()
file_edges = stack_create()

Set defaults for code (undirected, input from stdin instead of input file)
directed = false
input = false

Use if statements to check the command line arguments
for (command line arguments) {
        if (command = i) {
                input = true
                infile = optarg
        }
        if (command = d) {
                directed = true
        }
        if (command = u) {
                if (directed) {
                        print error
                        exit
                }
        }
        if (command = m) {
                print matrix
        }

Make matrix
matrix[26][26] = full of zeros
```

```
if (input file given):
        get input_content from input file
else:
        get input_content from stdin
for (i = 0; i not at end of input_content; i++):
        if i != line break:
                input_pair[2] = 2 characters before \n
        else:
                if (!directed):
                        matrix[input_pair[0] – ‘A’][input_pair[1] – ‘A’] = 1
                        matrix[input_pair[1] – ‘A’][input_pair[0] – ‘A’] = 1
                else:
                        matrix[input_pair[0] – ‘A’][input_pair[1] – ‘A’] = 1

Print matrix
void print_matrix(matrix) {
        for (i = 0; i < 26; i++) {
                for (j = 0; j < 26; j++) {
                        print matrix[i][j]
                print line break
        return
}

Print path
void print_path(s->top) {
        set path length to stack size
        curr_path[path_length]
        for (i = path_length; i >= 0; i--) {
                curr_path[i] = path.pop()
        }
        for (i = 0; i < path_length; i++) {
```

```
                print curr_path[i]
        }
}

visited = [26 false]
short_path = 26

// Derived from assignment PDF
void dfs(uint32_t curr_node):
        visited[curr_node] = true
        if curr_node is the exit:
                if stack_size < short_path:
                        short_path = stack_size
                print_path()
        for (i = 0-25):
                if matrix[curr_node][i] == 1:
                        path.push(curr_node)
                        dfs(next_node)
                        path.pop()
                else:
                        curr_node = path.pop()
                        dfs(curr_node)
        return
```