Graph.c

Graph_vertices is essentially how many vertices there is. Therefore I would return the value inputted.

```
graph_vertices(*G){
    return G->vertices;
}
```

graph_ add_edge checks if vertices are in bounds so < 26. Then assigns the weight k to the matrix using G-> matrices [I][j]. return bool

```
graph_add_edge(*G, i, j, k){
    if(i,j < 26){
        G->matrix[i][j] = k;
        return true;
        if(G-> undirected){
            G->matrix[j][i] = k;
            return ther
        }
        }else{
            return false
    }
```

graph_has_edge just checks if a matrix has a weight k or not(returns bool)

```
if(i,j < 26){
    k= G->matrix[i][j]
    if(k>0){
        return true
    }else{
        return false
    }
return false
```

graph_edge_weight does what graph_has_edge just returns the actual value though rather than a bool

```
if(graph_has_edge)== true){
    return e = G->matrix(i)[j];
}else{
    return false
```

graph_visited checks if the v value is in the array and returns a bool

```
return G->visited[v]; * visited is already a bool
```

graph_mark_unvisited marks that v value as false & visited

visited

```
if(v<26){
    G->visited[v] = true
}
```

unvisited

```
if(v<26){
    G->visited[v] = false
}
```

stack.c

stack_empty returns a bool checking if the stack is empty. Im going make use of the top var in the struct to check.

```
if top == 0 {
  return true;
}
```

stack_full; alot like stack_empty I will use the top var but I will compare it to the capacity var in the struct. Returns bool

```
if top >= 26
return true
}
```

stack_size; will use top var and return what is in the var

```
return top;
```

stack_push; will have to use top var to see where the stack is, check if is full, if not continue, then push onto items then return true.

```
if (stack_full) {
   return False
} else {
   *top = x
   rtrn true
```

stack_pop; will use top and stack_empty. First check if stack empty, then remove *x at items/ top

```
if (stack_empty) {
   return false
} else {
   *x = s->items[s->top]      //Set to zero after pop
   return true
```

Stack_peek; checks top value but doesnt alter

```
if (stack_empty){
   return False
} else {
   *x = s->items[s->top]
   return true
```

path.c

path_create ; initializes everything

Stack vertices =(Stack *) malloc (size of (vertices))

if (vertices) {

p->length =0;

}

return vertices

path_delete ; set vertices to null

free (*p)

*p = null

path_push_vertex ; returns a bool. I will use stack push & Graph to get edge. I also need to increase the path

x=stack_peek

make sure would make it false stack_push(v)

k= graph_edge_weight (*G, x, v)

length += k;

return true

path_pop ; take top one off (backtracking) & subtracts the length

Stack_pop(v)

x=Stack_peek(x)

k= graph_edge_weight (*G, x, v)

length -= k

return true

path_length returns length

return = p-> length

path_vertices ; returns num vertices so top of stack

return = p-> vertices [top]