Purpose of the Lab:

This program will find and print the shortest path, length, and total recursive calls from .graph file using graph data structure. This program will find a shortest path(or enable -v to print all paths) through all of the vertices visiting exactly once(direct).

Files:

Graph.c and .h: implementation for graph ADT, contain sets of vertices, for example: <0, 1, 2> , <1, 2, 2>......
Stack.c and .h: implementation stack ADT, helper files, stack should be used in path.c
Path.c and .h: implementation for path ADT,
Vertices.h: define the initial vertices and maximum vertices(26), if vertices in graph greater than 26, then exit the program.
Tsp.c: main function(include inflie, outfile, command options, create graph and path, print message etc.)
Helper.c: content helper functions like dfs(), and other functions that needed in main function

Command line options:

-h: print out help message describing purpose of this program, command line options, and idea of what to print

-v: enable verbose printing

-u: set the graph to be undirected(meaning can go back and forward in 1 path)

-i and -o: input file and output file

Pseudocode:

Graph.c:

```
graph_create:
Pointer g = malloc
G -> vertices = vertices
G -> undirected = undirected
For loop: set all g -> visited[i] = false
Flor loop: set g -> matrix[i][j] = 0

Graph_delete:
Free pointer and set g to null

graph_vertices
Return g-> vertices

graph_add_edge:
If i< vertices and j < vertices,
    If directed, matrix[i][j]=k, if not, add an edge, matrix [j][i]= k
```

Return true
Else return false

graph_has_edge:
If i< vertices and j < vertices and matrix[i][j]!=0, return true, else return false

graph_edge_weight:
If i< vertices and j < vertices and matrix[i][j]!=0, return matrix items, else return 0

graph_visited:
Return visited true or not

graph_mark_visited:
If v < vertices, set visited = ture

graph_mark_unvisited:
If v < vertices, set visited = false

Stack.c
stack_create:
Stack *s = malloc
Then initial top, capacity, and items(calloc)
If can't allocated memory, then free(s) and set s to null

stack_delete:
Delete each items in memory first, free pointer, and set pointer to null

stack_size:
Return s->top

stack_empty:
Check if s-> top true or not

stack_full:
Check capacity true or not

stack_push:
if stack_full return false, if not items[top]=x, top++ and return true

stack_pop:
if stack_empty return false, if not top-- and *x = item[top]
bool stack_peek(Stack *s, uint32_t *x)
If stack_empty = true, then *x = s->items[s->top-1]

stack_copy:

```
dst->top = src->top
dst->capacity = src->capacity
for i =0; i < src->capacity; i++:
        dst->items[i] = src->items[i]
```

Path.c:

path_create:

```
Path *p = = malloc
P -> vertices = stack_create(vertices)
P -> length = 0
```

path_delete:

Delete vertices items first, free pointer, set pointer to null

path_push_vertex:

```
Flag = false
Flag = stack_peek(vertices, &u)
Length += pragh_edge_weight(G,u,v)
```

path_pop_vertex:

```
Flag = false
Flag = stack_pop(vertices, v)
Flag = stack_peek(vertices, u) &&flag;
Length -= graph_edge_weight(G, u, *v)
```

path_vertices:

```
Return stack_size p->vertices
```

path_length

```
Return p->length
```

path_copy:

```
stack_copy(dst->vertices, src-> vertices)
dst-> length = src-> length
```

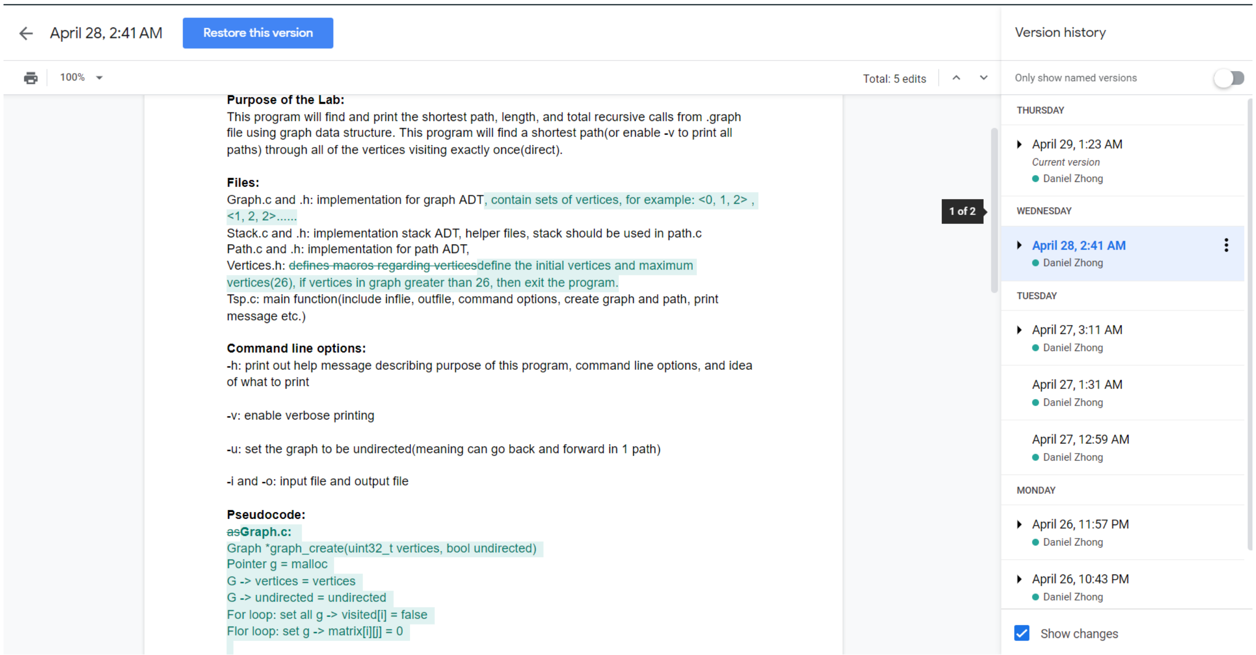
DFS() check pdf pseudocode

Tsp.c:

```
Set verbose and undirected to false
FILE *infp = stdin
FILE *outfp = stdout //default for input and output file

Getop for command options:
-h printf (help message)
-v  verbose = true
-u  undirected = ture
-i if((infp = fopen(optarg,"r")) == NULL), print error message
-o if((outfp = fopen(optarg,"r")) == NULL), print error message

Fscanf (read graph from infile)
If n<=1 or n> 26, then print out error message

Create graph
Read edges<u,v,w> from infile

Record path and shortest path
Use dfs() to find hamiltonian path
Print shortest path
Delete path and graph, fclose infile and outfile
```

Working history on google docs before Thursday:

I prob submitted wrong design.pdf on Thursday(4/29), but this is my working history before Thursday



Purpose of the Lab:
This program will find and print the shortest path, length, and total recursive calls from .graph file using graph data structure. This program will find a shortest path(or enable -v to print all paths) through all of the vertices visiting exactly once(direct).

Files:
Graph.c and .h: implementation for graph ADT, contain sets of vertices, for example: <0, 1, 2> , <1, 2, 2>......
Stack.c and .h: implementation stack ADT, helper files, stack should be used in path.c
Path.c and .h: implementation for path ADT,
Vertices.h: ~~defines macros regarding vertices~~define the initial vertices and maximum vertices(26), if vertices in graph greater than 26, then exit the program.
Tsp.c: main function(include inflie, outfile, command options, create graph and path, print message etc.)

Command line options:
-h: print out help message describing purpose of this program, command line options, and idea of what to print

-v: enable verbose printing

-u: set the graph to be undirected(meaning can go back and forward in 1 path)

-i and -o: input file and output file

Pseudocode:
~~as~~**Graph.c:**

```
Graph *graph_create(uint32_t vertices, bool undirected)
Pointer g = malloc
G -> vertices = vertices
G -> undirected = undirected
For loop: set all g -> visited[i] = false
Flor loop: set g -> matrix[i][j] = 0
```