Design Document

Description

The purpose of this program is to explore depth-first search through the Traveling Salesman Problem. Through the use of three abstract data structures- graphs, paths, and stacks- this problem uses backtracking to optimize the shortest path that visits every city exactly once and returns to the starting city. This program also supports file inputs and outputs to account for large input sizes.

Structure

- **DFS**
- The depth-first search function has six parameters.
    - The Graph *G has a pointer to the adjacency matrix, which shows us which cities are connected, and also keeps track of which cities we have visited.
    - V is the city that the DFS call is currently at.
    - The Path *cur is the current path for the DFS call.
    - The Path *shortest keeps track of the shortest Hamiltonian path we have found so far. Whenever *cur is a valid Hamiltonian path, it is compared to *shortest.
    - Cities is the list of cities.
    - Outfile is where we print things.

```
func DFS(Graph *G, uint32_t v, Path *cur, Path *shortest, char *cities[],
FILE *outfile) {
-     label v as visited
-     add v to Path *cur
-     if cur visited every city:
-           if edge exists from cur to start:
-                 add start to Path *cur
-           if length of cur < length of shortest:
-                 shortest = cur
-                 print cur to file
-           continue
-     for all edges {v,w} in G->matrix[v]:
-           if w is not visited:
-                 DFS(G, W, cur, shortest, cities, outfile)
-     label v as unvisited
}
```

Inputs and Outputs

- Inputs will be read from a file if a file is provided by the user. If not, the input will be from stdin.

```
FILE *infile = stdin;
if(file provided) {
      *infile = fopen("example.txt", "r");
}
```

- Outputs work similarly. If a file is provided, then the output will go there. If not, then the output will be sent to stdout.

```
FILE *outfile = stdout;
if(file provided) {
      *outfile = fopen("output.txt", "w");
}
```

- The number of cities will be read using fscanf().

```
uint32_t n = 0;
fscanf(infile, "%" SCNu32 "\n", &n);
```

- The cities will be read using fgets().

```
for(uint32_t i=0; i<n; i++){
      fgets(cities[i], 1024, infile);
}
```

- Finally, the edges will be read using fscanf() until EOF is reached.

```
uint32_t city1=0, city2=0, weight=0;
while(fscanf(infile, "%" SCNu32 " %" SCNu32 " %" SCNu32 "\n", &city1,
&city2, &weight) != EOF) {
      graph_add_edge(G, city1, city2, weight);
}
```

This chart shows the expected inputs and outputs of this program.