Assignment 1 – Design of Algorithms
William Liandri – 728710

Part 6: Graph Representation

Adjacency matrix is a 2D array that contains information which neighbors are reachable from a specified vertex. To make the program works using adjacency matrix, there are some changes that need to be done. For depth first traversal, it needs to iterate through the row of a vertex until the first number '1' is found (indicates that vertex v is the first edge and reachable from u). After that, it will go through the row of vertex v and find the first edge. It keeps doing that until all of the vertices have been visited. However, for depth first traversal, it will iterate through the row of a vertex until the end of the column. After that, it will dequeue the vertex from the queue and go through the row of the first vertex on the queue. It will repeat the process again until all of the vertices have been visited.

Part 7: Design of Algorithms

- **Part 3**:
  My program is using depth first traversal to find a detailed path from vertex u to vertex v. Depth first transversal is an algorithm that uses stack for traversing a graph where it starts from the source id and try to look the first edge. Each time when the program finds the unvisited vertex, it will push the vertex to the stack. However, when the vertex has been visited or it cannot reach the destination, it will pop that vertex from the stack and try to find another vertex by checking the edges of the top vertex on the stack. After it has reached the destination, the program will print the path and find the total distance from u to v. The worst-case complexity of the program is $O(2(V + E))$ where V is the number of vertices and E the number of edges. The first $O(V + E)$ comes from the program iterate through all of the vertices and all of edges that each vertex have before reach the destination and the other $O(V + E)$ comes from the program iterate through the stack to print the path (assume that the path goes through all of the vertices before reach the destination) and iterate through its edges to find the total distance.

- **Part 4**:
  My program for part 4 is similar to part 3 where I am using depth first traversal to find all possible paths and also apply the same process to find the unvisited vertex when the vertex has been visited or it cannot reach the destination is found. The difference is when it has reached the destination; it will print the path and pop the destination from the stack. Then, it will continue to find another path by iterating through all of the neighbors of the vertex on the top of the stack. On my program, after the first path has found, it will store all of the parent edges on an array and use it to check whether all of these parent edges have been visited. If they have been visited, the program will stop the loop. The worst-case complexity of the program is $O(V^2 + E)$ where V is the number of vertices and E is the number of edges that each vertex has. The complexity can be $O(V^2 + E)$ since the $O(V^2)$ when the program pick a vertex, it might to all other vertices to find all of the possible path and the O(E) comes from when the program store the parent edges on the array.

- **Part 5**:
  To solve questions part 5, my program is using Dijkstra's algorithm to find the shortest path from vertex u to v. Dijkstra's algorithm is an algorithm where it searches for a parent of a vertex that can give a minimum distance from a source id. To get the minimum distance for all vertices from a source id, the program will select a vertex that has a minimum distance on the distance array and that vertex is still available on the heap. Then, the program will check if there is a vertex that is reachable by looping through each vertex and iterating each neighbor. If the minimum distance is found, it will update the parent and the distance for that vertex. After that process has finished, it will insert the path from source id to destination id into a stack by finding each parent (except for source id), calculate the total distance and print the path. The worst-case complexity of the program is $O(V^2 + V)$ where $O(V)$ comes from when the program set INT_MAX value for all of the vertices and $O(V^2)$ comes from when the program need to go through all of through all of the vertices that is still available in the heap array and each process, the program will go through all of the vertices and edges on the graph to find the minimal distance.