**Project #2**:

3-Link Disjoint Shortest Paths

**Mankirat Gulati**

ID: 111161128

CSE 346

**Source Code**

This program was written in **Python 3** and consists of three files, util.py, graph.py, and main.py.

* Soft copy of program: http://github.com/devhid/cse346/tree/master/project2-solution

*util.py*

A screen shot of a computer

Description automatically generated

*main.py*

A screenshot of a cell phone

Description automatically generated

*graph.py*

*A screenshot of a cell phone

Description automatically generated*

**Code Explanation & Output**

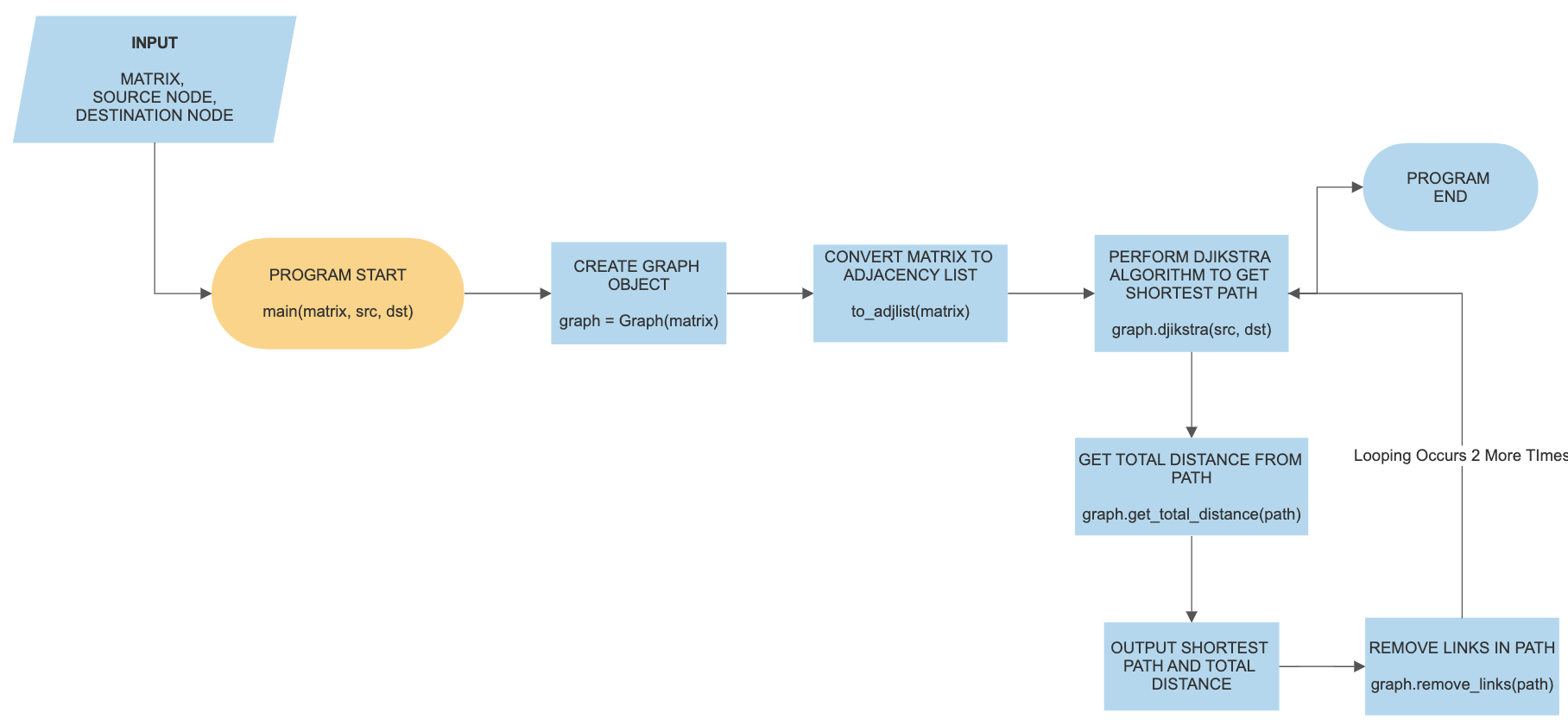
* The to\_adjlist() function in util.py converts an adjacency matrix into an adjacency list. This makes it easier to work with the data since we have a concrete concept of vertices and edges.
* The Graph class in graph.py is a wrapper for the adjacency list and has functions to remove or add edges (add\_edge() and remove\_edge(), perform a djikstra algorithm from a source node to a destination node (djikstra()), remove links in a path (remove\_links()), and calculate the distance in a path (get\_total\_distance()).
* The main() function in main.py outputs the 3-link disjoint shortest paths of the test case provided in the assignment. It takes in a matrix, the source node, and destination node as its input.

*The output of the program is here below.*

A screenshot of a cell phone

Description automatically generated

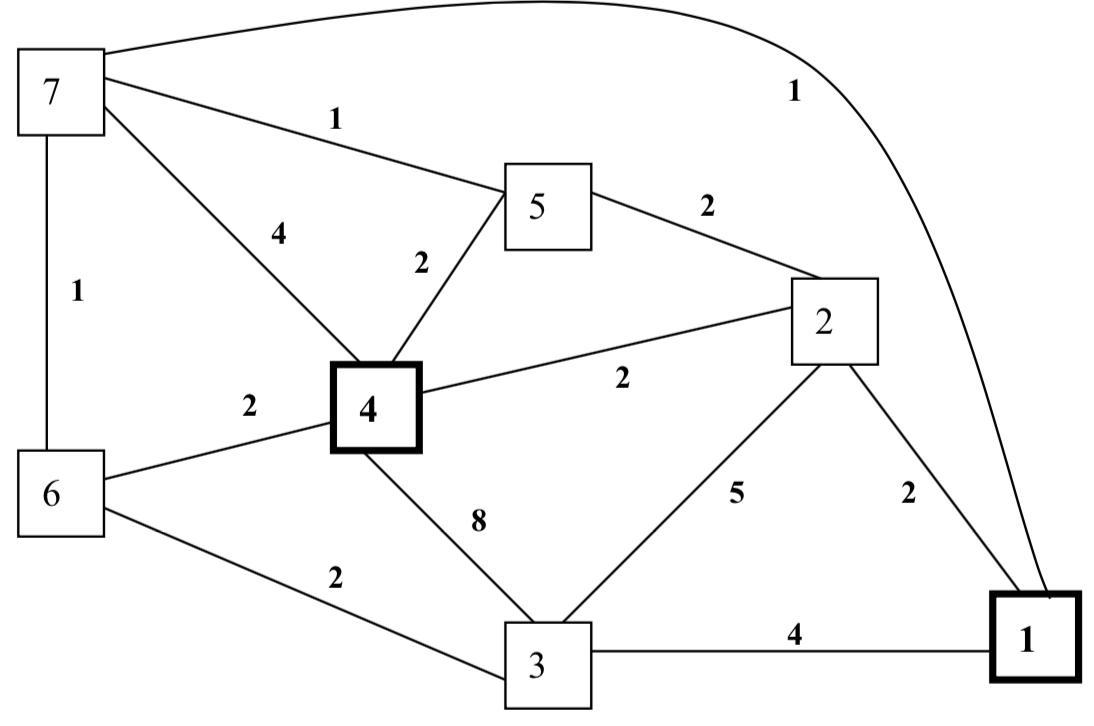
**Flow Chart**



**Visualizations**

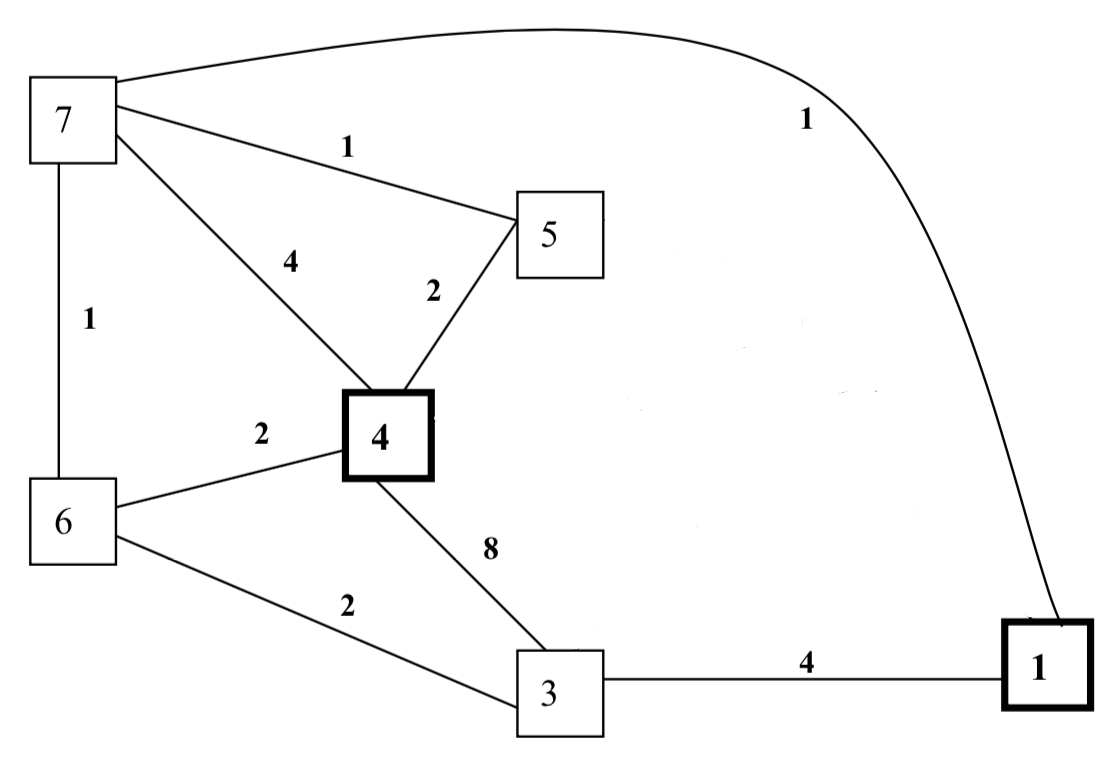
1st Link

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Node | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1** | 0 | 2 | 4 | 0 | 0 | 0 | 1 |
| **2** | 2 | 0 | 5 | 2 | 2 | 0 | 0 |
| **3** | 4 | 5 | 0 | 8 | 0 | 2 | 0 |
| **4** | 0 | 2 | 8 | 0 | 2 | 2 | 4 |
| **5** | 0 | 2 | 0 | 2 | 0 | 0 | 1 |
| **6** | 0 | 0 | 2 | 2 | 0 | 0 | 1 |
| **7** | 1 | 0 | 0 | 4 | 1 | 1 | 0 |



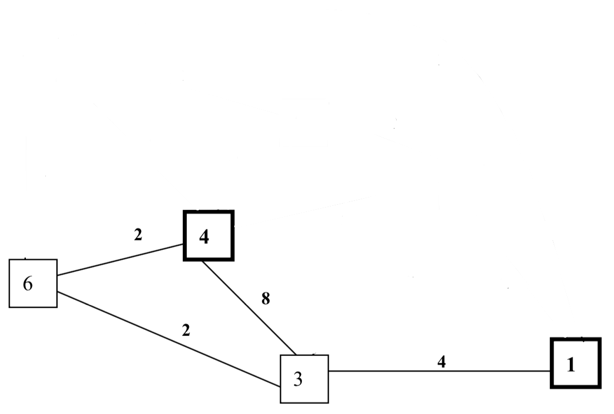
2nd Link

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Node | **1** | **3** | **4** | **5** | **6** | **7** |
| **1** | 0 | 4 | 0 | 0 | 0 | 1 |
| **3** | 4 | 0 | 8 | 0 | 2 | 0 |
| **4** | 0 | 8 | 0 | 2 | 2 | 4 |
| **5** | 0 | 0 | 2 | 0 | 0 | 1 |
| **6** | 0 | 2 | 2 | 0 | 0 | 1 |
| **7** | 1 | 0 | 4 | 1 | 1 | 0 |



3rd Link

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Node | **1** | **3** | **4** | **6** |
| **1** | 0 | 4 | 0 | 0 |
| **3** | 4 | 0 | 8 | 2 |
| **4** | 0 | 8 | 0 | 2 |
| **6** | 0 | 2 | 2 | 0 |



**Routing Tables**

1st Link

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

2nd Link

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

3rd Link

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Conclusion**

It becomes evident that as we remove more and more links, the total distance of the shortest path increases. This is because the paths that were originally the shortest paths cannot be used anymore so an alternate route has to be found. This alternate route might have heavier costs resulting in a greater distance. In this specific test case, the distances that found at each link (1, 2, 3) were 4, 4, and 8, respectively. Removing the first link did not change the shortest distance of 4 but removing the second link gave us a distance of 8, which is double of what was found previously.