**NOTE :- Please execute the Assignment\_code.py module, it is the main file(module) to be executed which will call all other modules as and when required.**

**Design Strategy: -**

* The algorithm used for the given problem statement is Dijkstra’s algorithm.
* The given problem statement is to find the shortest distance between 2 places which are a Hospital and an airport which have different routes between them, which is a single-source shortest-path problem. For this we mainly have 2 different algorithms, which are Dijkstra’s algorithm and the Bellman-Ford algorithm.
* Dijkstra’s algorithm fails when there are negative weights for the paths/edges between 2 points/nodes and Bellman-Ford is more suitable for this scenario.
* But, for our given problem statement, it is mentioned that the values represent the distance between 2 points/nodes which cannot be negative, hence we applied the Dijkstra’s algorithm for the problem statement.
* Also, Dijkstra’s algorithm can be used to solve the problem statement considering 2 approaches, one with adjacency matrix and the other with adjacency list to store the nodes and the distance between them.
* We chose the adjacency matrix over the adjacency list for the below reasons: -

1. Space Complexity: In the worst case, we considered the graph to be a dense matrix where there are paths from each node to every other node.

In this case, the adjacency matrix occupies less space compared to the adjacency list which is more suitable when the graph is sparse.

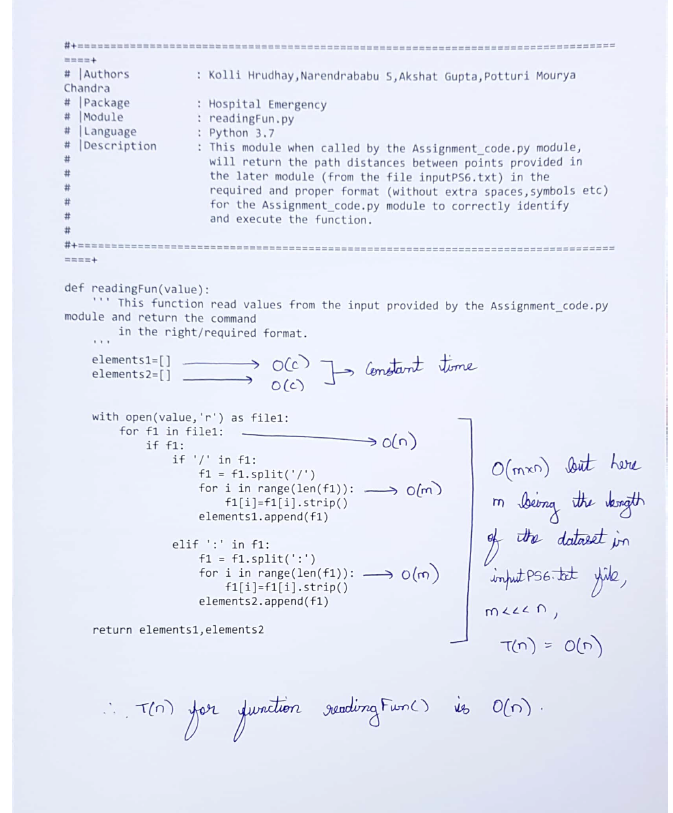
1. Access time: In terms of access time, it takes O(1) to check the distance between two nodes in adjacency matrix, whereas it can take up to O(n) as it must list all the nodes that the node is connected to.
2. Time complexity: The overall time complexity is O(n^2) in case of adjacency matrix whereas it is O(V+E) for the adjacency list where V are the no of node and E are no of edges.

Though, the overall time complexity is higher for the adjacency matrix, in terms of the accessing time, adjacency matrix is much more efficient when finding the relationships in a graph, it is fast to look up and check for presence or absence of a specific edge between any two nodes and also it is more efficient in terms of space when a dense graph is considered.

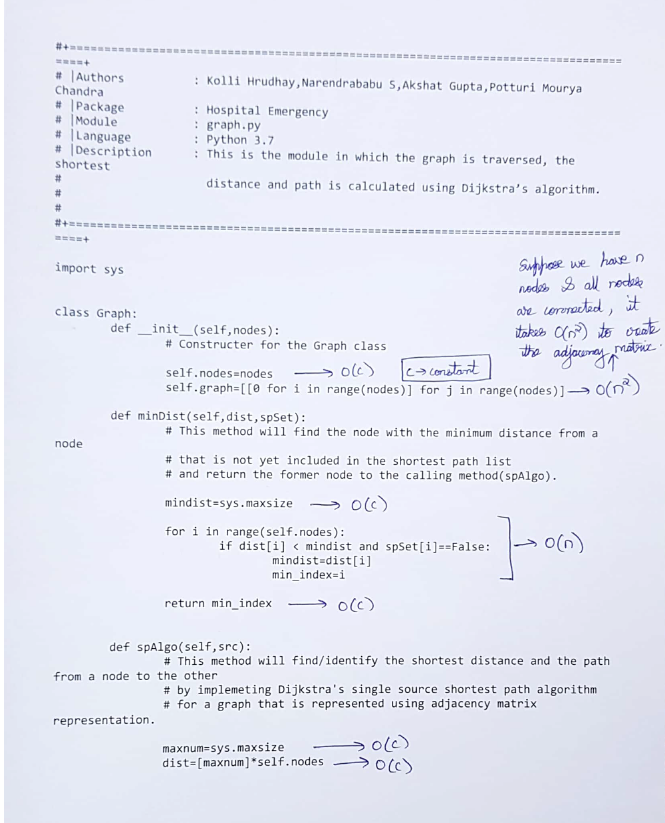
**Time Complexity: -**

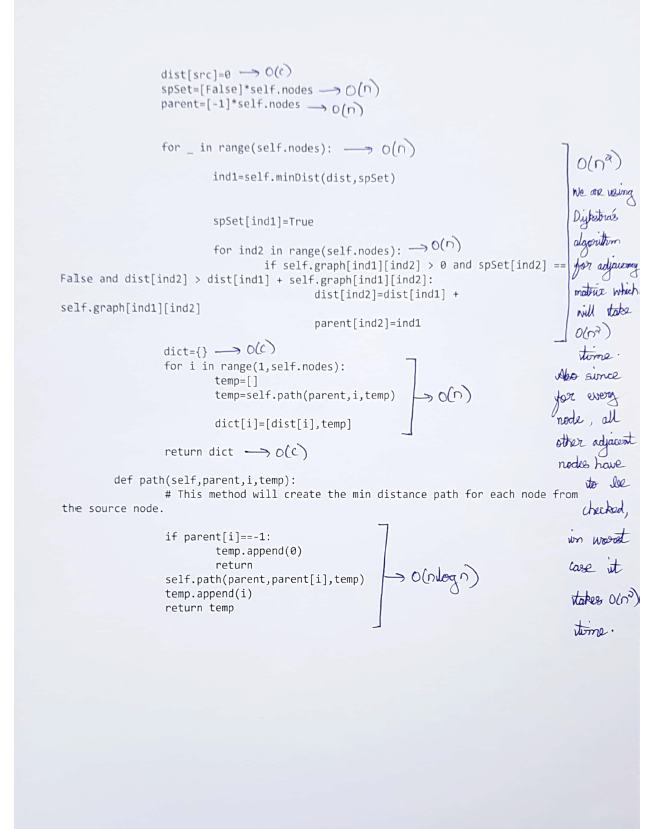
The time complexity of our entire code is **O(n^2)** where n is the no of nodes.

We calculated it as below: -

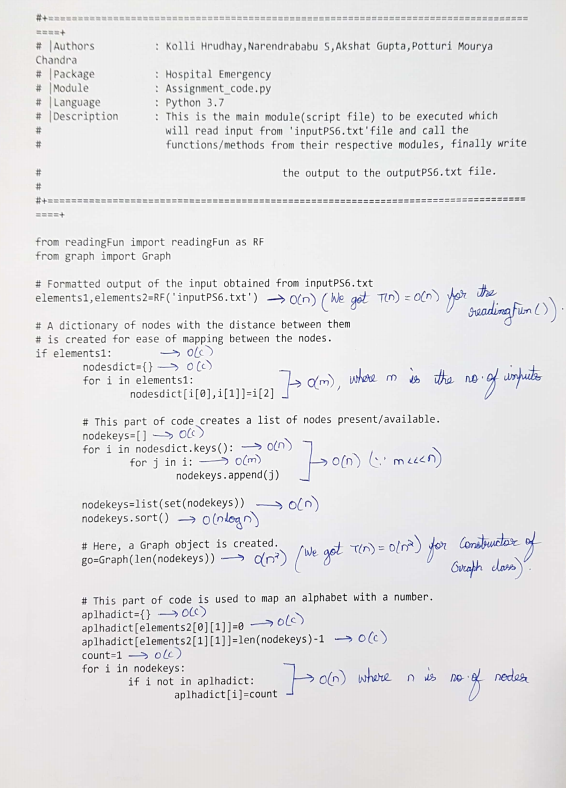
**File name – readingFun.py** 

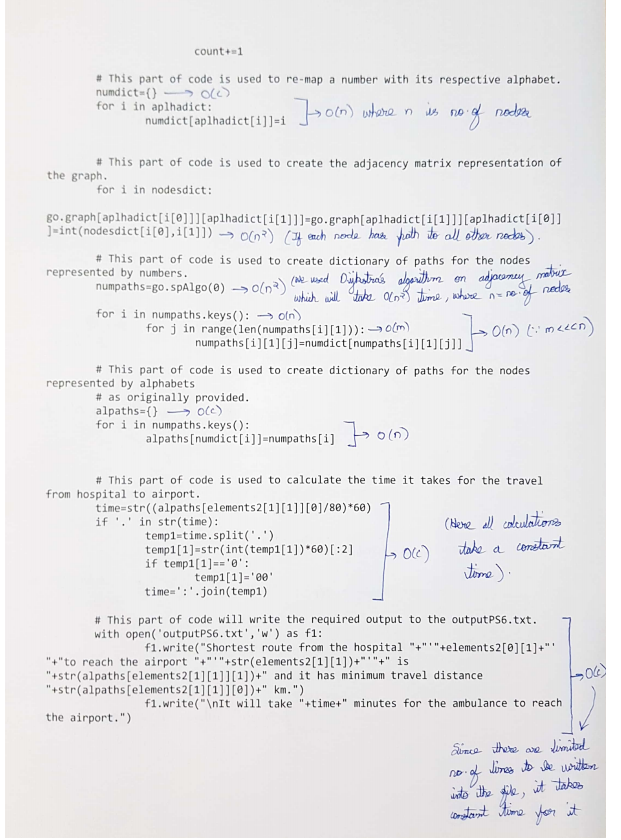
**File name – graph.py**





**File name – Assignment\_code.py**







As we can see from the calculations, the time complexity is O(n^2).

It is an efficient one because it take O(1) ) to check the distance between two nodes in an adjacency matrix and also it is fast to look up and check for presence or absence of a specific edge between any two nodes and also it is more efficient in terms of space when a dense graph is considered.

Though adjacency list can be used for the same, but in case of very dense graph where there are paths/edges from each node to all other nodes, this increases the space complexity. Whereas for very dense graphs, adjacency matrix is more efficient with less space complexity and better access time.

Hence, the algorithm we used is an efficient one to solve the given problem statement.