# Registry

For registry I used Dictionary. I used two dictionaries, One will save all of the persons on a node and other will save the serialized key against a person\_id. I used dictionaries because It we can easily access the persons that are on a particular node by their node\_id. Other dictionary save the person serialized key(in string format) at particular person\_id..

# Communication network

For communication network I have used adjacency list in which we use vertex as a key and its neighboring vertex in a list against that vertex. I used a dictionary for the storage of the nodes but in adjacency list only nodes ids and its cost were used. The reason for using adjacency list is that we can easily store undirected graphs in it. I used undirected graph because all of the nodes should be reachable from other nodes and undirected graph is good for that situation.

# Encoding

For encoding a key associated with a person is used. Basically a key is a list of 36 chars ordered in a way that at every index there is a corresponding combination of dots and slashes in other list. For encoding I just traverse the string and against every chars I placed a combination of dots and slashes separated by space. For combination of dots and slashes I used a Enum class, in which I saved the combinations of dots and slashes.

# Decoding

For decoding same key was used. I split by space the string and check for the key’s char at the corresponding index of the combination of dots and slashes. Like if a corresponds to “…” then it finds the index of “…” in enum then just return the char at the same index in the key.

# Passing

So if I needed a communication between the node and the person, I used registry. Like if I needed the serialized key while decoding the message, I used the registry deserialized it then read all of the messages. Registry is the linking point between the nodes and the persons. Because node doesn’t have a property of persons that’s why I had to use the registry.

# Key (generation, serialization, deserialization)

For key generation as I explained earlier I used a enum of combination of dots and slashes. Firstly, I sorted the chars according to their frequency in the training text then the chars with 0 frequency were added. In this way we have a combination sorted in descending order corresponding to each character.

For serialization I just converted the the list into string as we can only store numbers or string in the registry. For example if [“a”, “v”, “d”, ……] is the order I converted it as “avd..” and when I needed to deserialize it I just converted it back to a list. Because we just need to access its index to decode it.

# Algorithm used:

## DFS

For traversing the graph depth first search was used. Breadth first search could also be used but I used depth first search because it is easier to understand and implement. I used dfs for checking if all of the nodes are reachable and for finding the cheapest path for the forwarding. I used recursive function to implement it. First I am checking if a particular node is already visited? If it is then I am returning it if not then I am visiting the neighboring nodes. Then at the end I am checking of count of the visiting nodes and the nodes in the network are equal? If they are then network is called to be reachable.

## Dijkstra Algorithm

Dijkstra Algorithm was used to find the cheapest path from a node to its destination. It not only tells the cheapest path for destination but for every other vertex, that’s why I used it. It is also using recursive function same as in dfs algorithm but the only difference is that we can visit the neighboring visited nodes if they are already visited. Only parent is permanently added to the visiting list. After that I just checked if the path is cheap then the last path if it is then it is updated in the dictionary also the last vertex is also used in the dictionary. If we need to find the cheapest path from that dictionary, we just need to traverse the dictionary.