Possible Errors/Constraints: I have chosen N=5 bits. So, the maximum number of possible nodes is 2^n.

Running Environment: The code is written in python. The visual display of the code is dependent in ***Tkinter*** module which is part of a standard python module and is available in IDE such as Spyder, PyCharm, etc.

How to execute the code: simply execute the code in any python complier. The GUI will appear and you can use it to perform addition, deletion and lookup operation.

Visual Layout:

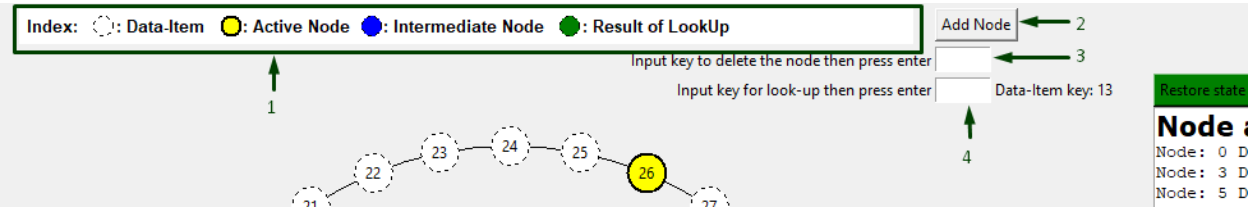

Fig 1: Visual of the program – 1

Index	Function
1	Indicates the different types of nodes and data-items used to represent a chord architecture
2	Button to add a node to the chord architecture. It adds a node and indicates the new node with yellow color.
3	Input field for the ***id*** of the node to be deleted. After the ***id*** has been typed and enter key has been pressed, the node will be removed from the chord architecture and its corresponding data-items will be distributed between its successor and predecessor.
4	Input field for the ***key*** of the data-items to be looked for. After the key has been typed and enter has been pressed, the node associated with the data-item containing the key is located. A path is drawn starting from the node with minimum ***id*** to that node.

Table 1: Shows the function of indexes that are pointed in above figure

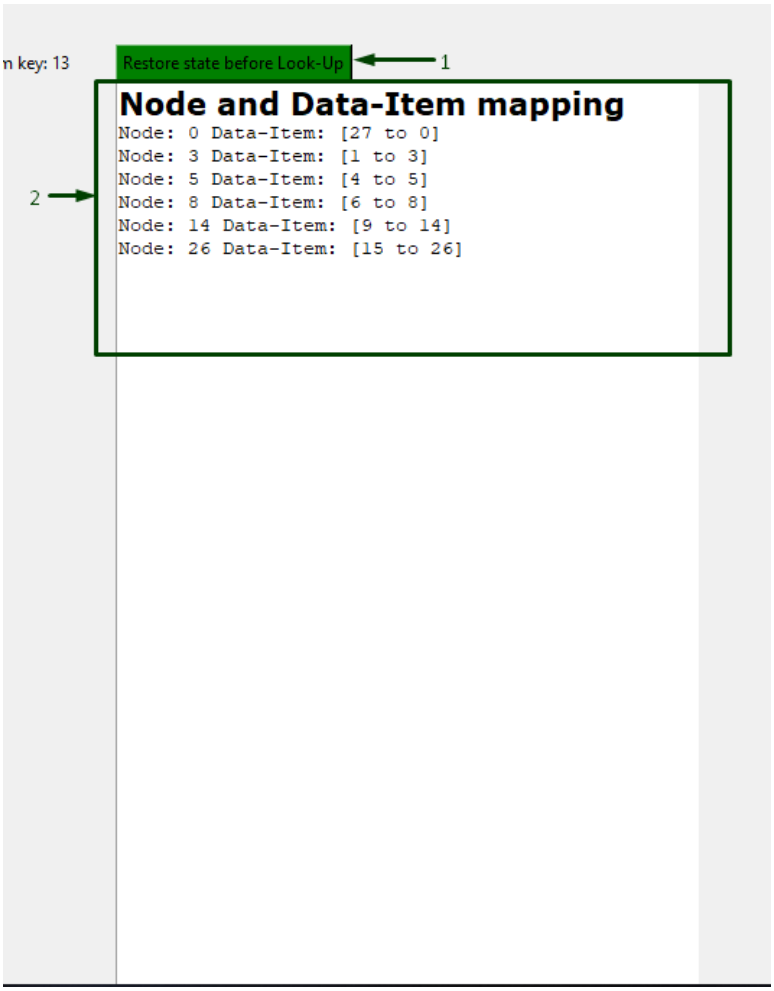

Fig 2: Visual of the program - 2

Index	Function
1	Button to restore the state of the chord architecture to the state before lookup was performed. It just removes the path created by the lookup operation.
2	An area showing the ***id*** of current nodes and the ***key*** of data-items associated with it.

Table 2: Shows the function of indexes that are pointed in above figure

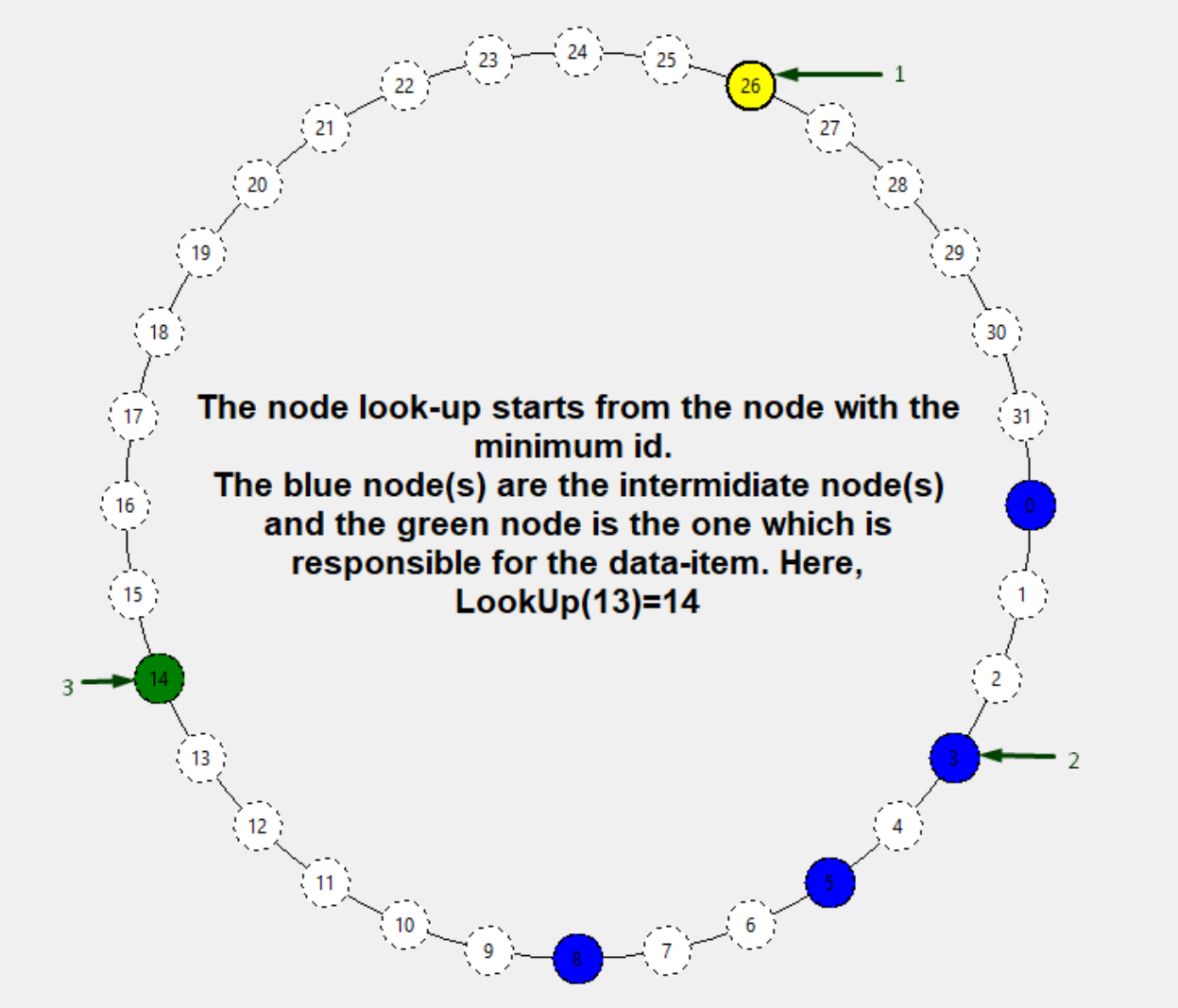

Fig: Visual of the program – 3

Index	Function
1	An active node which is not used during current look up operation.
2	An intermediate node which was used to locate the address of its successor node during the lookup
3	The node which is associated with the ***key*** of the data-items we are looking for.

Table 1: Shows the function of indexes that are pointed in above figure

Operations:

1. **Add Node:**
   Input: click add button.
   Output: A node will be added with yellow color indication on the node with random ***id*** generation. In addition, on the right-hand side of the chord, changes related to the nodes responsible for the data items will be shown.
2. **Delete Node:**
   Input: ***ID*** of the node to delete it. The ID should be integer and only 1 ID can be provided at a time. After providing the ID hit Enter (↵).
   Output: The node will be removed from the list of active nodes. In addition, on the right-hand side of the chord, changes related to the nodes responsible for the data items will be shown.
3. **Look-Up:**
   Input: ***key*** for the data-item, then press Enter (↵). The key should be integer and only 1 key can be provided at a time.
   Output: It draws a path from the node with minimum id to the node responsible for the data-item with the key. It colors blue for intermediate nodes and colors green for the node responsible for data-item with the key.
4. **Restore state before look-up:**
   Input: Press “Restore state before Look Up” button
   Output: Removes the color highlight of the node showing path to the node responsible for the key of a data-item and restores the color to the node before performing look up operation.