323-35 Project 3: Radix-Sort for String Java

Student: Matthew Smyth

Project Due Date: 09/17/2020

Algorithm Steps:
Step 0: inFile open the input file
outFile1 open outFile1 // for the output of sorted data
outFile2 open outFile2 // for observations
use constructor to create two hash tables of LLQueue where each hashTable[i][j] linked list queue with
a dummy node, and let the head and tail point to the dummy node.
Step 1: firstReading (inFile) // see algorithm below
Step 2: close inFile
Step 3: inFile open the input file // open the file second time
step 4: S loadStack (inFile) // see algorithm below
Step 5: printStack (S, outFile2) // Print caption!!! Say what you are printing
Step 6: currentPosition longestStringLength -1 // Sort from right to left of the paddedData.
currentTable 0
Step 7: moveStack (currentPosition, currentTable) // move all nodes on the stack to

// the first hash table. See the algorithm below

Step 8: - currentPosition - -
- nextTable mod (currentTable + 1, 2)
- currentQueue 0
Step 9: // moving nodes from currentTable to nextTable, process queues in the current table sequentially.
node <-- deleteHead (hashTable[currentTable][currentQueue])
chr <-- getChar (node, currentPosition) // i.e., the character at the currentPosition of node's data
hashIndex <-- (int) chr or atoi (chr) // cast chr from asci to integer
addTail (hashTable[nextTable][hashIndex], node) //
// add the node at the tail of the queue at hashTable[nextTable][hashIndex]
Step 10: repeat steps 9 until the currentQueue is empty // finish moving all node in currentQueue.
Step 11: currentQueue ++ // process the next queue in the current hashTable
Step 12: repeat step 9 to step 11 while currentQueue < tableSize
// finish moving all queues from the current table.
Step 13: printTable((hashTable[nextTable], outFile2) // to outFile2
Step 14: currentTable nextTable
Step 15: repeat step 8 to step 14 while currentPosition >= 0
Step 16: PrintSortedData (hashTable[nextTable], outFile1)
Step 17: close all files

****
V. firstReading (inFile)
****
Step 0: longestStringLength 0
Step 1: data read a word from inFile
Step 2: If length of data > longestStringLength
longestStringLength length of data
Step 3: repeat step 1 to step 2 until inFile is empty

4

****
VI. (LLStack) loadStack (inFile)
****
Step 0: S Use LLStack constructor to establish a LLStack
Step 1: S.top null // initialize top points to null
Step 2: data read a string from inFile
// YOU MUST READ ONE STRING At A TIME!! -2pt otherwise
Step 3: paddedData padString (data)
Step 4: newNode create a new listNode for paddedData
Step 5: push (newNode) <-- push newNode onto the top of the stack

newNode.next top
top newNode

step 6: repeat step 2 – step 5 until inFile is empty
step 7: return S
****
VII. moveStack (S, currentPosition, currentTable)
****
Step 1: node <-- pop from the top of the stack S
// move each listNode from stack to hashTable[0]
step 2: chr <-- getChar (node, currentPosition) // i.e., the character at the currentPosition of node's data
step 3: hashIndex <-- (int) chr // cast chr from asci to integer
step 4: addTail (hashTable[currentTable][hashIndex], node)
// add the node at the tail of the queue at hashTable[currentTable][hashIndex]
Step 5: repeat step 1 to step 4 until stack is empty

Source Code

```
import java.io.FileReader;
import java.io.FileWriter;
import java.util.Scanner;

public class Main {

	public static void main(String[] args) {
		try {
			Scanner inFile = new Scanner(new FileReader(args[0]));
			RSort hash = new RSort();
			hash.firstReading(inFile);
			inFile.close();
			inFile = new Scanner(new FileReader(args[0]));
			RSort.LLStack S = hash.loadStack(inFile);
			hash.moveStack(S);
			FileWriter outFile2 = new FileWriter(args[2]);
			while (hash.currentPosition > 0) {
				hash.currentPosition--;
				hash.nextTable = (hash.currentTable + 1) % 2;
				hash.currentQueue = 0;
				hash.deleteHead();
				hash.printTable(outFile2);
				outFile2.write("\n");
				hash.currentTable = hash.nextTable;
			}
			outFile2.close();
			hash.printSortedData(args[1]);

		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}

import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class RSort {
	public class listNode {
		String data;
```

```
	listNode next;

	public listNode(String data) {
		this.data = data;
	}

	public void printNode() {
		System.out.print("(" + data + "," + next.data + ") -> ");
	}
}

public class LLStack {
	listNode top;

	public LLStack() {
		this.top = new listNode("dummyNode");
	}

	public void push(String newNode) {
		listNode temp = new listNode(newNode);
		temp.next = top;
		top = temp;
	}

	public listNode pop() {
		if (this.isEmpty()) {
			System.out.println("empty!");
			return null;
		}
		listNode temp = top;
		top = top.next;
		temp.next = null;
		return temp;
	}

	public boolean isEmpty() {
		return top.next == null;
	}

	public void printTop() {
		System.out.println(top.data);
	}
}
```

```
public class LLQueue {
	listNode head, tail;

	public LLQueue() {
		listNode temp = new listNode("dummyNode");
		this.head = temp;
		this.tail = temp;
	}

	public void insertQ(String newNode) {
		listNode temp = new listNode(newNode);
		tail.next = temp;
		tail = temp;
	}

	public listNode deleteQ() {
		if (this.isEmpty()) {
			System.out.println("empty!");
			return null;
		}
		listNode temp = head.next;
		if (head.next == tail) {
			tail = head;
			head.next = null;
		} else {
			head.next = head.next.next;
		}
		return temp;
	}

	public boolean isEmpty() {
		return head == tail;
	}

	public void printQueue(int whichTable, int index, FileWriter output) {
		try {
			output.write("Table [" + whichTable + "][" + index + "]: ");
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		listNode temp = hashTable[whichTable][index].head;
		String data;
		String data1;
```

```
                while (temp != null) {
                        data = temp.data;
                        if (temp.next != null) {
                                data1 = temp.next.data;
                        } else {
                                data1 = null;
                        }
                        try {
                                output.write("(" + data + ", " + data1 + ") -> ");
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                        temp = temp.next;
                }
                try {
                        output.write("NULL\n");
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
}

LLQueue[][] hashTable;
String data;
int currentTable = 0;
int nextTable = 1;
int longestStringLength = 0;
int currentPosition;
int currentQueue = 0;

public RSort() {
        hashTable = new LLQueue[2][256];
        for (int i = 0; i < 2; i++) {
                for (int j = 0; j < 256; j++) {
                        hashTable[i][j] = new LLQueue();
                }
        }
}

public void firstReading(Scanner inFile) {
        String data;
        while (inFile.hasNext()) {
```

```
                data = inFile.next();
                if (longestStringLength < data.length()) {
                        longestStringLength = data.length();
                }
        }
}

public LLStack loadStack(Scanner inFile) {
        LLStack S = new LLStack();
        String data;
        while (inFile.hasNext()) {
                data = inFile.next();
                String paddedData = padString(data);
                listNode newNode = new listNode(paddedData);
                newNode.next = S.top;
                S.top = newNode;
        }
        currentPosition = longestStringLength - 1;
        return S;
}

public void moveStack(LLStack S) {
        while (!S.isEmpty()) {
                listNode node = S.pop();
                char chr = getChar(node);
                int hashIndex = (int) chr;
                addTail(hashIndex, node);
        }
}

private char getChar(listNode node) {
        return node.data.charAt(currentPosition);
}

private String padString(String data2) {
        int difference = 0;
        difference = longestStringLength - data2.length();
        for (int i = 0; i < difference; i++) {
                data2 = data2 + " ";
        }
        return data2;
}

private void addTail(int hashIndex, listNode node) {
```

```
		node.next = null;
		hashTable[nextTable][hashIndex].tail.next = node;
		hashTable[nextTable][hashIndex].tail = node;
	}

	public void printTable(FileWriter outFile2) {
		for (int i = 0; i < 256; i++) {
			if (!hashTable[nextTable][i].isEmpty()) {
				hashTable[nextTable][i].printQueue(nextTable, i, outFile2);

			}

		}
	}

	public void printSortedData(String outFile1) {
		String sorted = "";
		LLQueue Q;
		for (int i = 0; i < 256; i++) {
			if (!hashTable[nextTable][i].isEmpty()) {
				Q = hashTable[nextTable][i];
				listNode temp = Q.head.next;
				while (temp != null) {
					sorted = sorted + " " + temp.data;
					temp = temp.next;
				}
			}
		}
		FileWriter output;
		try {
			output = new FileWriter(outFile1);
			output.write(sorted);
			output.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void deleteHead() {
		for (int i = 0; i < 256; i++) {
			while (!hashTable[currentTable][i].isEmpty()) {
				listNode node = hashTable[currentTable][i].deleteQ();
				char chr = getChar(node);
				int hashIndex = (int) chr;
```

```
                    addTail(hashIndex, node);
                }
            }
        }
}
```

Output1

```
 AAbb   AbCdEfG Acc    BbAa   Bdd    CcaabB  Hijk   XyZzz  ZZZZ   aA     acc    bggff
cCaAbb  jlJkL  xyyk   zxcccc
```

Output2

```
Table [1][32]: (dummyNode, bggff  ) -> (bggff  , Bdd    ) -> (Bdd    , acc    ) -> (acc    , Acc    ) ->
(Acc    , jlJkL  ) -> (jlJkL  , Hijk   ) -> (Hijk   , ZZZZ   ) -> (ZZZZ   , xyyk   ) -> (xyyk   , XyZzz  ) ->
(XyZzz  , zxcccc ) -> (zxcccc , AAbb   ) -> (AAbb   , aA     ) -> (aA     , BbAa   ) -> (BbAa   ,
CcaabB ) -> (CcaabB , cCaAbb ) -> (cCaAbb , null) -> NULL
Table [1][71]: (dummyNode, AbCdEfG) -> (AbCdEfG, null) -> NULL

Table [0][32]: (dummyNode, Bdd    ) -> (Bdd    , acc    ) -> (acc    , Acc    ) -> (Acc    , Hijk   ) ->
(Hijk   , ZZZZ   ) -> (ZZZZ   , xyyk   ) -> (xyyk   , AAbb   ) -> (AAbb   , aA     ) -> (aA     , BbAa   )
-> (BbAa   , null) -> NULL
Table [0][69]: (dummyNode, AbCdEfG) -> (AbCdEfG, null) -> NULL
Table [0][76]: (dummyNode, jlJkL  ) -> (jlJkL  , null) -> NULL
```

Table [0][98]: (dummyNode, CcaabB) -> (CcaabB , cCaAbb) -> (cCaAbb , null) -> NULL
Table [0][99]: (dummyNode, zxcccc) -> (zxcccc , null) -> NULL
Table [0][102]: (dummyNode, bggff) -> (bggff , null) -> NULL
Table [0][122]: (dummyNode, XyZzz) -> (XyZzz , null) -> NULL

Table [1][32]: (dummyNode, Bdd   ) -> (Bdd   , acc   ) -> (acc   , Acc   ) -> (Acc   , aA   ) -> (aA   , null) -> NULL
Table [1][65]: (dummyNode, cCaAbb) -> (cCaAbb , null) -> NULL
Table [1][90]: (dummyNode, ZZZZ   ) -> (ZZZZ   , null) -> NULL
Table [1][97]: (dummyNode, BbAa   ) -> (BbAa   , CcaabB) -> (CcaabB , null) -> NULL
Table [1][98]: (dummyNode, AAbb   ) -> (AAbb   , null) -> NULL
Table [1][99]: (dummyNode, zxcccc) -> (zxcccc , null) -> NULL
Table [1][100]: (dummyNode, AbCdEfG) -> (AbCdEfG, null) -> NULL
Table [1][102]: (dummyNode, bggff) -> (bggff , null) -> NULL
Table [1][107]: (dummyNode, Hijk   ) -> (Hijk   , xyyk   ) -> (xyyk   , jIJkL) -> (jIJkL , null) -> NULL
Table [1][122]: (dummyNode, XyZzz) -> (XyZzz , null) -> NULL

Table [0][32]: (dummyNode, aA   ) -> (aA   , null) -> NULL
Table [0][65]: (dummyNode, BbAa   ) -> (BbAa   , null) -> NULL
Table [0][67]: (dummyNode, AbCdEfG) -> (AbCdEfG, null) -> NULL
Table [0][74]: (dummyNode, jIJkL) -> (jIJkL , null) -> NULL
Table [0][90]: (dummyNode, ZZZZ   ) -> (ZZZZ   , XyZzz) -> (XyZzz , null) -> NULL
Table [0][97]: (dummyNode, cCaAbb) -> (cCaAbb , CcaabB) -> (CcaabB , null) -> NULL
Table [0][98]: (dummyNode, AAbb   ) -> (AAbb   , null) -> NULL
Table [0][99]: (dummyNode, acc   ) -> (acc   , Acc   ) -> (Acc   , zxcccc) -> (zxcccc , null) -> NULL
Table [0][100]: (dummyNode, Bdd   ) -> (Bdd   , null) -> NULL
Table [0][103]: (dummyNode, bggff) -> (bggff , null) -> NULL
Table [0][106]: (dummyNode, Hijk   ) -> (Hijk   , null) -> NULL
Table [0][121]: (dummyNode, xyyk   ) -> (xyyk   , null) -> NULL

Table [1][65]: (dummyNode, aA   ) -> (aA   , AAbb   ) -> (AAbb   , null) -> NULL
Table [1][67]: (dummyNode, cCaAbb) -> (cCaAbb , null) -> NULL
Table [1][73]: (dummyNode, jIJkL) -> (jIJkL , null) -> NULL
Table [1][90]: (dummyNode, ZZZZ   ) -> (ZZZZ   , null) -> NULL
Table [1][98]: (dummyNode, BbAa   ) -> (BbAa   , AbCdEfG) -> (AbCdEfG, null) -> NULL
Table [1][99]: (dummyNode, CcaabB) -> (CcaabB , acc   ) -> (acc   , Acc   ) -> (Acc   , null) -> NULL
Table [1][100]: (dummyNode, Bdd   ) -> (Bdd   , null) -> NULL
Table [1][103]: (dummyNode, bggff) -> (bggff , null) -> NULL
Table [1][105]: (dummyNode, Hijk   ) -> (Hijk   , null) -> NULL
Table [1][120]: (dummyNode, zxcccc) -> (zxcccc , null) -> NULL
Table [1][121]: (dummyNode, XyZzz) -> (XyZzz , xyyk   ) -> (xyyk   , null) -> NULL

Table [0][65]: (dummyNode, AAbb) -> (AAbb , AbCdEfG) -> (AbCdEfG, Acc) -> (Acc , null) -> NULL
Table [0][66]: (dummyNode, BbAa) -> (BbAa , Bdd) -> (Bdd , null) -> NULL
Table [0][67]: (dummyNode, CcaabB) -> (CcaabB , null) -> NULL
Table [0][72]: (dummyNode, Hijk) -> (Hijk , null) -> NULL
Table [0][88]: (dummyNode, XyZzz) -> (XyZzz , null) -> NULL
Table [0][90]: (dummyNode, ZZZZ) -> (ZZZZ , null) -> NULL
Table [0][97]: (dummyNode, aA) -> (aA , acc) -> (acc , null) -> NULL
Table [0][98]: (dummyNode, bggff) -> (bggff , null) -> NULL
Table [0][99]: (dummyNode, cCaAbb) -> (cCaAbb , null) -> NULL
Table [0][106]: (dummyNode, jlJkL) -> (jlJkL , null) -> NULL
Table [0][120]: (dummyNode, xyyk) -> (xyyk , null) -> NULL
Table [0][122]: (dummyNode, zxcccc) -> (zxcccc , null) -> NULL

2output1

Aaron Abeesh Alexis Amreen Angel Archimed Asher Bijaya Bret Calvin Carlos Christopher Christos Edwin Eunhee Frederick Gurnoor Hammad Hyungbin Jamil Jason Jason Jephter Jianhui Jiawei Jonathan Jordon Joshua Juan Justin Kenley Kevin Krzysztof Lei Lei Martin Matthew Matthew Michael Murgray Nahian Naiem Nectario Rajendra Rashad Robert Roberto Rupert Russell Ryan Shadman Shahan Shelley Steven Talha Tenzin Thomas Umair Vincenzo Weiting Wilber Wong Ye Yuhang Yulin Zaynab

2output2

Table [1][32]: (dummyNode, Eunhee) -> (Eunhee , Krzysztof) -> (Krzysztof , Gurnoor) -> (Gurnoor , Shadman) -> (Shadman , Talha) -> (Talha , Edwin) -> (Edwin , Angel) -> (Angel , Shahan) -> (Shahan , Jason) -> (Jason , Alexis) -> (Alexis , Ryan) -> (Ryan , Jephter) -> (Jephter , Vincenzo) -> (Vincenzo , Shelley) -> (Shelley , Thomas) -> (Thomas , Jamil) -> (Jamil , Amreen) -> (Amreen , Rajendra) -> (Rajendra , Jordon) -> (Jordon , Murgray) -> (Murgray , Lei) -> (Lei , Joshua) -> (Joshua , Rashad) -> (Rashad , Umair) -> (Umair , Abeesh) -> (Abeesh , Aaron) -> (Aaron , Tenzin) -> (Tenzin , Archimed) -> (Archimed , Wilber) -> (Wilber , Asher) -> (Asher , Bijaya) -> (Bijaya , Carlos) -> (Carlos , Roberto) -> (Roberto , Christos) -> (Christos , Jonathan) -> (Jonathan , Martin) -> (Martin , Steven) -> (Steven , Lei) -> (Lei , Nectario) -> (Nectario , Zaynab) -> (Zaynab , Matthew) -> (Matthew , Calvin) -> (Calvin , Wong) -> (Wong , Bret) -> (Bret , Kenley) -> (Kenley , Jason) -> (Jason , Juan) -> (Juan , Justin) -> (Justin , Russell) -> (Russell , Yulin) -> (Yulin , Weiting) -> (Weiting , Rupert) -> (Rupert , Ye) -> (Ye , Jiawei) -> (Jiawei , Michael) -> (Michael , Frederick) -> (Frederick , Naiem) -> (Naiem , Matthew) -> (Matthew , Kevin) -> (Kevin , Yuhang) -> (Yuhang , Nahian) -> (Nahian , Jianhui) -> (Jianhui , Hammad) -> (Hammad , Robert) -> (Robert , Hyungbin) -> (Hyungbin , null) -> NULL
Table [1][114]: (dummyNode, Christopher) -> (Christopher, null) -> NULL

Table [0][32]: (dummyNode, Eunhee) -> (Eunhee , Gurnoor) -> (Gurnoor , Shadman) -> (Shadman , Talha) -> (Talha , Edwin) -> (Edwin , Angel) -> (Angel , Shahan) -> (Shahan , Jason) -> (Jason , Alexis) -> (Alexis , Ryan) -> (Ryan , Jephter) -> (Jephter , Vincenzo) -> (Vincenzo , Shelley) -> (Shelley , Thomas) -> (Thomas , Jamil) -> (Jamil , Amreen) -> (Amreen , Rajendra) -> (Rajendra , Jordon) -> (Jordon , Murgray) -> (Murgray , Lei) -> (Lei , Joshua) -> (Joshua , Rashad) -> (Rashad , Umair) -> (Umair , Abeesh) -> (Abeesh , Aaron) -> (Aaron , Tenzin) -> (Tenzin , Archimed) -> (Archimed , Wilber) -> (Wilber , Asher) -> (Asher , Bijaya) -> (Bijaya , Carlos) -> (Carlos , Roberto) -> (Roberto , Christos) -> (Christos , Jonathan) -> (Jonathan , Martin) -> (Martin , Steven) -> (Steven , Lei) -> (Lei , Nectario) -> (Nectario , Zaynab) -> (Zaynab , Matthew) -> (Matthew , Calvin) -> (Calvin , Wong) -> (Wong , Bret) -> (Bret , Kenley) -> (Kenley , Jason) -> (Jason , Juan) -> (Juan , Justin) -> (Justin , Russell) -> (Russell , Yulin) -> (Yulin , Weiting) -> (Weiting , Rupert) -> (Rupert , Ye) -> (Ye , Jiawei) -> (Jiawei , Michael) -> (Michael , Naiem) -> (Naiem , Matthew) -> (Matthew , Kevin) -> (Kevin , Yuhang) -> (Yuhang , Nahian) -> (Nahian , Jianhui) -> (Jianhui , Hammad) -> (Hammad , Robert) -> (Robert , Hyungbin) -> (Hyungbin , null) -> NULL
Table [0][102]: (dummyNode, Krzysztof) -> (Krzysztof , null) -> NULL

Table [0][104]: (dummyNode, Christopher) -> (Christopher, null) -> NULL
Table [0][107]: (dummyNode, Frederick) -> (Frederick , null) -> NULL

Table [1][32]: (dummyNode, Eunhee) -> (Eunhee , Gurnoor) -> (Gurnoor , Shadman) -> (Shadman , Talha) -> (Talha , Edwin) -> (Edwin , Angel) -> (Angel , Shahan) -> (Shahan , Jason) -> (Jason , Alexis) -> (Alexis , Ryan) -> (Ryan , Jephter) -> (Jephter , Shelley) -> (Shelley , Thomas) -> (Thomas , Jamil) -> (Jamil , Amreen) -> (Amreen , Jordon) -> (Jordon , Murgray) -> (Murgray , Lei) -> (Lei , Joshua) -> (Joshua , Rashad) -> (Rashad , Umair) -> (Umair , Abeesh) -> (Abeesh , Aaron) -> (Aaron , Tenzin) -> (Tenzin , Wilber) -> (Wilber , Asher) -> (Asher , Bijaya) -> (Bijaya , Carlos) -> (Carlos , Roberto) -> (Roberto , Martin) -> (Martin , Steven) -> (Steven , Lei) -> (Lei , Zaynab) -> (Zaynab , Matthew) -> (Matthew , Calvin) -> (Calvin , Wong) -> (Wong , Bret) -> (Bret , Kenley) -> (Kenley , Jason) -> (Jason , Juan) -> (Juan , Justin) -> (Justin , Russell) -> (Russell , Yulin) -> (Yulin , Weiting) -> (Weiting , Rupert) -> (Rupert , Ye) -> (Ye , Jiawei) -> (Jiawei , Michael) -> (Michael , Naiem) -> (Naiem , Matthew) -> (Matthew , Kevin) -> (Kevin , Yuhang) -> (Yuhang , Nahian) -> (Nahian , Jianhui) -> (Jianhui , Hammad) -> (Hammad , Robert) -> (Robert , null) -> NULL
Table [1][97]: (dummyNode, Rajendra) -> (Rajendra , null) -> NULL
Table [1][99]: (dummyNode, Frederick) -> (Frederick , null) -> NULL
Table [1][100]: (dummyNode, Archimed) -> (Archimed , null) -> NULL
Table [1][110]: (dummyNode, Jonathan) -> (Jonathan , Hyungbin) -> (Hyungbin , null) -> NULL
Table [1][111]: (dummyNode, Vincenzo) -> (Vincenzo , Nectario) -> (Nectario , Krzysztof) -> (Krzysztof , null) -> NULL
Table [1][112]: (dummyNode, Christopher) -> (Christopher, null) -> NULL
Table [1][115]: (dummyNode, Christos) -> (Christos , null) -> NULL

Table [0][32]: (dummyNode, Eunhee) -> (Eunhee , Talha) -> (Talha , Edwin) -> (Edwin , Angel) -> (Angel , Shahan) -> (Shahan , Jason) -> (Jason , Alexis) -> (Alexis , Ryan) -> (Ryan , Thomas) -> (Thomas , Jamil) -> (Jamil , Amreen) -> (Amreen , Jordon) -> (Jordon , Lei) -> (Lei , Joshua) -> (Joshua , Rashad) -> (Rashad , Umair) -> (Umair , Abeesh) -> (Abeesh , Aaron) -> (Aaron , Tenzin) -> (Tenzin , Wilber) -> (Wilber , Asher) -> (Asher , Bijaya) -> (Bijaya , Carlos) -> (Carlos , Martin) -> (Martin , Steven) -> (Steven , Lei) -> (Lei , Zaynab) -> (Zaynab , Calvin) -> (Calvin , Wong) -> (Wong , Bret) -> (Bret , Kenley) -> (Kenley , Jason) -> (Jason , Juan) -> (Juan , Justin) -> (Justin , Yulin) -> (Yulin , Rupert) -> (Rupert , Ye) -> (Ye , Jiawei) -> (Jiawei , Naiem) -> (Naiem , Kevin) -> (Kevin , Yuhang) -> (Yuhang , Nahian) -> (Nahian , Hammad) -> (Hammad , Robert) -> (Robert , null) -> NULL
Table [0][97]: (dummyNode, Jonathan) -> (Jonathan , null) -> NULL
Table [0][101]: (dummyNode, Archimed) -> (Archimed , null) -> NULL

Table [0][103]: (dummyNode, Weiting) -> (Weiting , null) -> NULL
Table [0][105]: (dummyNode, Jianhui) -> (Jianhui , Frederick) -> (Frederick , Hyungbin) -> (Hyungbin , Nectario) -> (Nectario , null) -> NULL
Table [0][108]: (dummyNode, Russell) -> (Russell , Michael) -> (Michael , null) -> NULL
Table [0][110]: (dummyNode, Shadman) -> (Shadman , null) -> NULL
Table [0][111]: (dummyNode, Roberto) -> (Roberto , Christopher) -> (Christopher, Christos) -> (Christos , null) -> NULL
Table [0][114]: (dummyNode, Gurnoor) -> (Gurnoor , Jephter) -> (Jephter , Rajendra) -> (Rajendra , null) -> NULL
Table [0][116]: (dummyNode, Krzysztof) -> (Krzysztof , null) -> NULL
Table [0][119]: (dummyNode, Matthew) -> (Matthew , Matthew) -> (Matthew , null) -> NULL
Table [0][121]: (dummyNode, Shelley) -> (Shelley , Murgray) -> (Murgray , null) -> NULL
Table [0][122]: (dummyNode, Vincenzo) -> (Vincenzo , null) -> NULL

Table [1][32]: (dummyNode, Talha) -> (Talha , Edwin) -> (Edwin , Angel) -> (Angel , Jason) -> (Jason , Ryan) -> (Ryan , Jamil) -> (Jamil , Lei) -> (Lei , Umair) -> (Umair , Aaron) -> (Aaron , Asher) -> (Asher , Lei) -> (Lei , Wong) -> (Wong , Bret) -> (Bret , Jason) -> (Jason , Juan) -> (Juan , Yulin) -> (Yulin , Ye) -> (Ye , Naiem) -> (Naiem , Kevin) -> (Kevin , null) -> NULL
Table [1][97]: (dummyNode, Joshua) -> (Joshua , Bijaya) -> (Bijaya , Shadman) -> (Shadman , Murgray) -> (Murgray , null) -> NULL
Table [1][98]: (dummyNode, Zaynab) -> (Zaynab , Hyungbin) -> (Hyungbin , null) -> NULL
Table [1][100]: (dummyNode, Rashad) -> (Rashad , Hammad) -> (Hammad , Rajendra) -> (Rajendra , null) -> NULL
Table [1][101]: (dummyNode, Eunhee) -> (Eunhee , Michael) -> (Michael , Jephter) -> (Jephter , Matthew) -> (Matthew , Matthew) -> (Matthew , Shelley) -> (Shelley , null) -> NULL
Table [1][103]: (dummyNode, Yuhang) -> (Yuhang , null) -> NULL
Table [1][104]: (dummyNode, Abeesh) -> (Abeesh , Jonathan) -> (Jonathan , null) -> NULL
Table [1][105]: (dummyNode, Jiawei) -> (Jiawei , null) -> NULL
Table [1][108]: (dummyNode, Russell) -> (Russell , null) -> NULL
Table [1][109]: (dummyNode, Archimed) -> (Archimed , null) -> NULL
Table [1][110]: (dummyNode, Shahan) -> (Shahan , Amreen) -> (Amreen , Jordon) -> (Jordon , Tenzin) -> (Tenzin , Martin) -> (Martin , Steven) -> (Steven , Calvin) -> (Calvin , Justin) -> (Justin , Nahian) -> (Nahian , Weiting) -> (Weiting , Vincenzo) -> (Vincenzo , null) -> NULL
Table [1][111]: (dummyNode, Gurnoor) -> (Gurnoor , null) -> NULL
Table [1][114]: (dummyNode, Wilber) -> (Wilber , Frederick) -> (Frederick , Nectario) -> (Nectario , null) -> NULL

Table [1][115]: (dummyNode, Alexis) -> (Alexis , Thomas) -> (Thomas , Carlos) -> (Carlos , null) -> NULL
Table [1][116]: (dummyNode, Rupert) -> (Rupert , Robert) -> (Robert , Roberto) -> (Roberto , Christopher) -> (Christopher, Christos) -> (Christos , null) -> NULL
Table [1][117]: (dummyNode, Jianhui) -> (Jianhui , null) -> NULL
Table [1][121]: (dummyNode, Kenley) -> (Kenley , null) -> NULL
Table [1][122]: (dummyNode, Krzysztof) -> (Krzysztof , null) -> NULL

Table [0][32]: (dummyNode, Ryan) -> (Ryan , Lei) -> (Lei , Lei) -> (Lei , Wong) -> (Wong , Bret) -> (Bret , Juan) -> (Juan , Ye) -> (Ye , null) -> NULL
Table [0][97]: (dummyNode, Talha) -> (Talha , Zaynab) -> (Zaynab , Rashad) -> (Rashad , Hammad) -> (Hammad , Michael) -> (Michael , Shahan) -> (Shahan , Nahian) -> (Nahian , Nectario) -> (Nectario , Thomas) -> (Thomas , null) -> NULL
Table [0][101]: (dummyNode, Eunhee) -> (Eunhee , Jiawei) -> (Jiawei , Russell) -> (Russell , Amreen) -> (Amreen , Steven) -> (Steven , Vincenzo) -> (Vincenzo , Wilber) -> (Wilber , Frederick) -> (Frederick , Kenley) -> (Kenley , null) -> NULL
Table [0][103]: (dummyNode, Hyungbin) -> (Hyungbin , null) -> NULL
Table [0][104]: (dummyNode, Matthew) -> (Matthew , Matthew) -> (Matthew , Jianhui) -> (Jianhui , null) -> NULL
Table [0][105]: (dummyNode, Archimed) -> (Archimed , Tenzin) -> (Tenzin , Martin) -> (Martin , Calvin) -> (Calvin , Justin) -> (Justin , Weiting) -> (Weiting , Alexis) -> (Alexis , null) -> NULL
Table [0][108]: (dummyNode, Angel) -> (Angel , Jamil) -> (Jamil , Shelley) -> (Shelley , null) -> NULL
Table [0][109]: (dummyNode, Naiem) -> (Naiem , Shadman) -> (Shadman , null) -> NULL
Table [0][110]: (dummyNode, Edwin) -> (Edwin , Jason) -> (Jason , Aaron) -> (Aaron , Jason) -> (Jason , Yulin) -> (Yulin , Kevin) -> (Kevin , Rajendra) -> (Rajendra , Yuhang) -> (Yuhang , null) -> NULL
Table [0][111]: (dummyNode, Jordon) -> (Jordon , Gurnoor) -> (Gurnoor , Carlos) -> (Carlos , null) -> NULL
Table [0][114]: (dummyNode, Umair) -> (Umair , Asher) -> (Asher , Murgray) -> (Murgray , Rupert) -> (Rupert , Robert) -> (Robert , Roberto) -> (Roberto , null) -> NULL
Table [0][115]: (dummyNode, Abeesh) -> (Abeesh , Christopher) -> (Christopher, Christos) -> (Christos , Krzysztof) -> (Krzysztof , null) -> NULL
Table [0][116]: (dummyNode, Jephter) -> (Jephter , Jonathan) -> (Jonathan , null) -> NULL
Table [0][117]: (dummyNode, Joshua) -> (Joshua , null) -> NULL
Table [0][121]: (dummyNode, Bijaya) -> (Bijaya , null) -> NULL

Table [1][32]: (dummyNode, Lei) -> (Lei , Lei) -> (Lei , Ye) -> (Ye , null) -> NULL
Table [1][97]: (dummyNode, Yuhang) -> (Yuhang , Jonathan) -> (Jonathan , Bijaya) -> (Bijaya , null) -> NULL
Table [1][98]: (dummyNode, Wilber) -> (Wilber , null) -> NULL
Table [1][99]: (dummyNode, Vincenzo) -> (Vincenzo , null) -> NULL
Table [1][100]: (dummyNode, Frederick) -> (Frederick , Shadman) -> (Shadman , Jordon) -> (Jordon , null) -> NULL
Table [1][101]: (dummyNode, Amreen) -> (Amreen , Angel) -> (Angel , Naiem) -> (Naiem , Rajendra) -> (Rajendra , Asher) -> (Asher , Rupert) -> (Rupert , Robert) -> (Robert , Roberto) -> (Roberto , Abeesh) -> (Abeesh , null) -> NULL
Table [1][103]: (dummyNode, Wong) -> (Wong , Murgray) -> (Murgray , null) -> NULL
Table [1][104]: (dummyNode, Talha) -> (Talha , Rashad) -> (Rashad , Michael) -> (Michael , Shahan) -> (Shahan , Eunhee) -> (Eunhee , Archimed) -> (Archimed , Jephter) -> (Jephter , Joshua) -> (Joshua , null) -> NULL
Table [1][105]: (dummyNode, Nahian) -> (Nahian , Jamil) -> (Jamil , Edwin) -> (Edwin , Yulin) -> (Yulin , Kevin) -> (Kevin , Umair) -> (Umair , Christopher) -> (Christopher, Christos) -> (Christos , null) -> NULL
Table [1][108]: (dummyNode, Kenley) -> (Kenley , Shelley) -> (Shelley , Carlos) -> (Carlos , null) -> NULL
Table [1][109]: (dummyNode, Hammad) -> (Hammad , Thomas) -> (Thomas , null) -> NULL
Table [1][110]: (dummyNode, Ryan) -> (Ryan , Juan) -> (Juan , Zaynab) -> (Zaynab , Hyungbin) -> (Hyungbin , Jianhui) -> (Jianhui , Gurnoor) -> (Gurnoor , null) -> NULL
Table [1][111]: (dummyNode, Jason) -> (Jason , Aaron) -> (Aaron , Jason) -> (Jason , null) -> NULL
Table [1][115]: (dummyNode, Russell) -> (Russell , null) -> NULL
Table [1][116]: (dummyNode, Bret) -> (Bret , Nectario) -> (Nectario , Matthew) -> (Matthew , Matthew) -> (Matthew , Martin) -> (Martin , Justin) -> (Justin , Weiting) -> (Weiting , null) -> NULL
Table [1][118]: (dummyNode, Steven) -> (Steven , Calvin) -> (Calvin , null) -> NULL
Table [1][119]: (dummyNode, Jiawei) -> (Jiawei , null) -> NULL
Table [1][120]: (dummyNode, Alexis) -> (Alexis , null) -> NULL
Table [1][121]: (dummyNode, Krzysztof) -> (Krzysztof , null) -> NULL
Table [1][122]: (dummyNode, Tenzin) -> (Tenzin , null) -> NULL

Table [0][32]: (dummyNode, Ye) -> (Ye , null) -> NULL
Table [0][97]: (dummyNode, Shadman) -> (Shadman , Shahan) -> (Shahan , Umair) -> (Umair , Ryan) -> (Ryan , Juan) -> (Juan , Jianhui) -> (Jianhui , Jiawei) -> (Jiawei , null) -> NULL
Table [0][98]: (dummyNode, Robert) -> (Robert , Roberto) -> (Roberto , null) -> NULL

Table [0][99]: (dummyNode, Michael) -> (Michael , Archimed) -> (Archimed , Nectario) -> (Nectario , null) -> NULL
Table [0][101]: (dummyNode, Frederick) -> (Frederick , Abeesh) -> (Abeesh , Shelley) -> (Shelley , Bret) -> (Bret , Steven) -> (Steven , Alexis) -> (Alexis , null) -> NULL
Table [0][103]: (dummyNode, Angel) -> (Angel , null) -> NULL
Table [0][104]: (dummyNode, Yuhang) -> (Yuhang , Asher) -> (Asher , Nahian) -> (Nahian , null) -> NULL
Table [0][105]: (dummyNode, Lei) -> (Lei , Lei) -> (Lei , Naiem) -> (Naiem , Weiting) -> (Weiting , null) -> NULL
Table [0][106]: (dummyNode, Bijaya) -> (Bijaya , Rajendra) -> (Rajendra , null) -> NULL
Table [0][108]: (dummyNode, Wilber) -> (Wilber , Talha) -> (Talha , Yulin) -> (Yulin , Calvin) -> (Calvin , null) -> NULL
Table [0][109]: (dummyNode, Jamil) -> (Jamil , Hammad) -> (Hammad , null) -> NULL
Table [0][110]: (dummyNode, Jonathan) -> (Jonathan , Vincenzo) -> (Vincenzo , Wong) -> (Wong , Eunhee) -> (Eunhee , Kenley) -> (Kenley , Tenzin) -> (Tenzin , null) -> NULL
Table [0][111]: (dummyNode, Thomas) -> (Thomas , null) -> NULL
Table [0][112]: (dummyNode, Rupert) -> (Rupert , Jephter) -> (Jephter , null) -> NULL
Table [0][114]: (dummyNode, Jordon) -> (Jordon , Amreen) -> (Amreen , Murgray) -> (Murgray , Christopher) -> (Christopher, Christos) -> (Christos , Carlos) -> (Carlos , Gurnoor) -> (Gurnoor , Aaron) -> (Aaron , Martin) -> (Martin , null) -> NULL
Table [0][115]: (dummyNode, Rashad) -> (Rashad , Joshua) -> (Joshua , Jason) -> (Jason , Jason) -> (Jason , Russell) -> (Russell , Justin) -> (Justin , null) -> NULL
Table [0][116]: (dummyNode, Matthew) -> (Matthew , Matthew) -> (Matthew , null) -> NULL
Table [0][117]: (dummyNode, Hyungbin) -> (Hyungbin , null) -> NULL
Table [0][118]: (dummyNode, Kevin) -> (Kevin , null) -> NULL
Table [0][119]: (dummyNode, Edwin) -> (Edwin , null) -> NULL
Table [0][121]: (dummyNode, Zaynab) -> (Zaynab , null) -> NULL
Table [0][122]: (dummyNode, Krzysztof) -> (Krzysztof , null) -> NULL

Table [1][97]: (dummyNode, Nahian) -> (Nahian , Naiem) -> (Naiem , Rajendra) -> (Rajendra , Talha) -> (Talha , Calvin) -> (Calvin , Jamil) -> (Jamil , Hammad) -> (Hammad , Carlos) -> (Carlos , Aaron) -> (Aaron , Martin) -> (Martin , Rashad) -> (Rashad , Jason) -> (Jason , Jason) -> (Jason , Matthew) -> (Matthew , Matthew) -> (Matthew , Zaynab) -> (Zaynab , null) -> NULL
Table [1][98]: (dummyNode, Abeesh) -> (Abeesh , null) -> NULL
Table [1][100]: (dummyNode, Edwin) -> (Edwin , null) -> NULL

Table [1][101]: (dummyNode, Ye) -> (Ye , Nectario) -> (Nectario , Lei) -> (Lei , Lei) -> (Lei , Weiting) -> (Weiting , Kenley) -> (Kenley , Tenzin) -> (Tenzin , Jephter) -> (Jephter , Kevin) -> (Kevin , null) -> NULL
Table [1][104]: (dummyNode, Shadman) -> (Shadman , Shahan) -> (Shahan , Shelley) -> (Shelley , Thomas) -> (Thomas , Christopher) -> (Christopher, Christos) -> (Christos , null) -> NULL
Table [1][105]: (dummyNode, Jianhui) -> (Jianhui , Jiawei) -> (Jiawei , Michael) -> (Michael , Bijaya) -> (Bijaya , Wilber) -> (Wilber , Vincenzo) -> (Vincenzo , null) -> NULL
Table [1][108]: (dummyNode, Alexis) -> (Alexis , null) -> NULL
Table [1][109]: (dummyNode, Umair) -> (Umair , Amreen) -> (Amreen , null) -> NULL
Table [1][110]: (dummyNode, Angel) -> (Angel , null) -> NULL
Table [1][111]: (dummyNode, Robert) -> (Robert , Roberto) -> (Roberto , Jonathan) -> (Jonathan , Wong) -> (Wong , Jordon) -> (Jordon , Joshua) -> (Joshua , null) -> NULL
Table [1][114]: (dummyNode, Archimed) -> (Archimed , Frederick) -> (Frederick , Bret) -> (Bret , Krzysztof) -> (Krzysztof , null) -> NULL
Table [1][115]: (dummyNode, Asher) -> (Asher , null) -> NULL
Table [1][116]: (dummyNode, Steven) -> (Steven , null) -> NULL
Table [1][117]: (dummyNode, Juan) -> (Juan , Yuhang) -> (Yuhang , Yulin) -> (Yulin , Eunhee) -> (Eunhee , Rupert) -> (Rupert , Murgray) -> (Murgray , Gurnoor) -> (Gurnoor , Russell) -> (Russell , Justin) -> (Justin , null) -> NULL
Table [1][121]: (dummyNode, Ryan) -> (Ryan , Hyungbin) -> (Hyungbin , null) -> NULL

Table [0][65]: (dummyNode, Aaron) -> (Aaron , Abeesh) -> (Abeesh , Alexis) -> (Alexis , Amreen) -> (Amreen , Angel) -> (Angel , Archimed) -> (Archimed , Asher) -> (Asher , null) -> NULL
Table [0][66]: (dummyNode, Bijaya) -> (Bijaya , Bret) -> (Bret , null) -> NULL
Table [0][67]: (dummyNode, Calvin) -> (Calvin , Carlos) -> (Carlos , Christopher) -> (Christopher, Christos) -> (Christos , null) -> NULL
Table [0][69]: (dummyNode, Edwin) -> (Edwin , Eunhee) -> (Eunhee , null) -> NULL
Table [0][70]: (dummyNode, Frederick) -> (Frederick , null) -> NULL
Table [0][71]: (dummyNode, Gurnoor) -> (Gurnoor , null) -> NULL
Table [0][72]: (dummyNode, Hammad) -> (Hammad , Hyungbin) -> (Hyungbin , null) -> NULL
Table [0][74]: (dummyNode, Jamil) -> (Jamil , Jason) -> (Jason , Jason) -> (Jason , Jephter) -> (Jephter , Jianhui) -> (Jianhui , Jiawei) -> (Jiawei , Jonathan) -> (Jonathan , Jordon) -> (Jordon , Joshua) -> (Joshua , Juan) -> (Juan , Justin) -> (Justin , null) -> NULL
Table [0][75]: (dummyNode, Kenley) -> (Kenley , Kevin) -> (Kevin , Krzysztof) -> (Krzysztof , null) -> NULL
Table [0][76]: (dummyNode, Lei) -> (Lei , Lei) -> (Lei , null) -> NULL

Table [0][77]: (dummyNode, Martin) -> (Martin , Matthew) -> (Matthew , Matthew) -> (Matthew , Michael) -> (Michael , Murgray) -> (Murgray , null) -> NULL
Table [0][78]: (dummyNode, Nahian) -> (Nahian , Naiem) -> (Naiem , Nectario) -> (Nectario , null) -> NULL
Table [0][82]: (dummyNode, Rajendra) -> (Rajendra , Rashad) -> (Rashad , Robert) -> (Robert , Roberto) -> (Roberto , Rupert) -> (Rupert , Russell) -> (Russell , Ryan) -> (Ryan , null) -> NULL
Table [0][83]: (dummyNode, Shadman) -> (Shadman , Shahan) -> (Shahan , Shelley) -> (Shelley , Steven) -> (Steven , null) -> NULL
Table [0][84]: (dummyNode, Talha) -> (Talha , Tenzin) -> (Tenzin , Thomas) -> (Thomas , null) -> NULL
Table [0][85]: (dummyNode, Umair) -> (Umair , null) -> NULL
Table [0][86]: (dummyNode, Vincenzo) -> (Vincenzo , null) -> NULL
Table [0][87]: (dummyNode, Weiting) -> (Weiting , Wilber) -> (Wilber , Wong) -> (Wong , null) -> NULL
Table [0][89]: (dummyNode, Ye) -> (Ye , Yuhang) -> (Yuhang , Yulin) -> (Yulin , null) -> NULL
Table [0][90]: (dummyNode, Zaynab) -> (Zaynab , null) -> NULL