Romerico David, Haley Elliott, Julian Halsey, Nkechiyem Molokwu
COSC 336
04/30/2024

Assignment 7

Exercise 1:

Each column sorted in order from:

Ones place →	Tens place →	Hundreds place →	Thousands place →	Ten-Thousands place
1.				
00034	00004	00004	00004	00004
09134	00034	00034	00034	00034
20134 →	09134 →	09134 →	20134 →	00134
29134	20134	20134	00134	09134
00004	29134	29134	09134	20134
00134	00134	00134	29134	29134
2.				
00004	00004	00004	00004	00004
00034	00034	00034	00034	00034
00134 →	00134 →	00134 →	00134 →	00134
09134	09134	09134	09134	09134
20134	20134	20134	20134	20134
29134	29134	29134	29134	29134
3.				
29134	00004	00004	00004	00004
20134	29134	00034	00034	00034
09134 →	20134 →	29134 →	20134 →	00134
00134	09134	20134	00134	09134
00034	00134	09134	29134	20134
00004	00034	00134	09134	29134

Exercise 2:

Algorithm that converts from base 10 to base k = n:

```
public static String[] convertArrayToBase(int[] nums,
   int base) {
    if (base < 2 || base > 36)
        return new String[] { "Invalid base" };
```

```
    String[] results = new String[nums.length];
    for (int i = 0; i < nums.length; i++) {
        int num = nums[i];
        String result = "";
        while (num > 0) {
                int remainder = num % base;
                if (remainder < 10)
                        result = remainder + result;
                else
                        result = (char) ('A' +
                           remainder - 10) + result;
                num /= base;
        }
        results[i] = result.equals("") ? "0" : result;
    }
    return results;
}
```

For this algorithm you would need to use base $k = n$ for it to run in $\Theta(n)$ time. Thus, each number in $\{0, 1, ..., n^2 - 1\}$ can be represented in base n with only two digits.

(a) 45, 98, 3, 82, 132, 71, 72, 143, 91, 28, 7, 45

In this case the base would be 12 as $n = 12$. In base 12 the sequence would be 3 9, 8 2, 0 3, 6 10, 11 0, 5 11, 6 0, 11 11, 7 7, 2 4, 0 7, 3 9

This example sorting would look like:

3 9	11 0	0 3
8 2	6 0	0 7
0 3	8 2	2 4
6 10	0 3	3 9
11 0	2 4	3 9
5 11	7 7	5 11
6 0	0 7	6 0
11 11	3 9	6 10
7 7	3 9	7 7
2 4	6 10	8 2
0 7	5 11	11 0
3 9	11 11	11 11

(b) 45, 98, 3, 82, 132, 71, 72, 143, 91, 28, 7, 45, 151, 175, 145, 399, 21,

267, 346, 292

In this case the base would be 20 as $n = 20$. In base 20 the sequence would be 2 5, 4 18, 0 3, 4 2, 6 12, 3 11, 3 12, 7 3, 4 11, 1 8, 0 7, 2 5, 7 11, 8 15, 7 5, 19 19, 1 1, 13 7, 17 6, 14 12

This example sorting would look like:

2 5	1 1	0 3
4 18	4 2	0 7
0 3	0 3	1 1
4 2	7 3	1 8
6 12	2 5	2 5
3 11	2 5	2 5
3 12	7 5	3 11
7 3	17 6	3 12
4 11	0 7	4 2
1 8	13 7	4 11
0 7	1 8	4 18
2 5	3 11	6 12
7 11	4 11	7 3
8 15	7 11	7 5
7 5	6 12	7 11
19 19	3 12	8 15
1 1	14 12	13 7
13 7	8 15	14 12
17 6	4 18	17 6
14 12	19 19	19 19

Programming Task:

Description of Algorithm:

Our algorithm, to create the squared graph, uses a triple for-loop to test if a node "i" is indirectly connected to another node "k" through one neighbor "j". If so, an edge is added between i and k. This condition is repeated for every node in the graph. A copy constructor was also added to the Adj List Graph helper class so that the G squared graph could be constructed while analyzing the original graph without changing it.

Code:

```
public class Adj_List_Graph {
```

```
int n;
ArrayList<ArrayList<Integer>> adj;

Adj_List_Graph(int no_nodes) {
  n = no_nodes;
  adj = new ArrayList<ArrayList<Integer>>(n);
  for (int i = 0; i < n; i++)
    adj.add(new ArrayList<Integer>());

}

// added copy constructor
public Adj_List_Graph(Adj_List_Graph g) {
  if (g == null)
    throw new IllegalArgumentException("Graph is null
       ");
  n = g.n;
  adj = new ArrayList<>(n);
  for (int i = 0; i < n; i++) {
    adj.add(new ArrayList<>(g.adj.get(i)));
  }
}

public void addEdge(int u, int v) {
  adj.get(u).add(v);
}

public void printGraph() {
  for (int i = 0; i < n; i++) {
    System.out.println("\nAdjacency list of vertex" +
        i);
    System.out.print("head");
    for (int j = 0; j < adj.get(i).size(); j++) {
      System.out.print(" -> " + adj.get(i).get(j));
    }
    System.out.println();
  }
}
```

```
}

public class Assign7 {
    public static void main(String[] args) {
        int[] input1 = { 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
           0, 1, 0, 0, 0, 0 };
        int[] input2 = fileReader("input-7-1.txt");
        int[] input3 = fileReader("input-7-2.txt");

        // Getting G
        Adj_List_Graph A0 = createGraph(input1);
        Adj_List_Graph B0 = createGraph(input2);
        Adj_List_Graph C0 = createGraph(input3);

        // Getting G^2
        Adj_List_Graph A = createGraphSquared(A0);
        Adj_List_Graph B = createGraphSquared(B0);
        Adj_List_Graph C = createGraphSquared(C0);

        System.out.print("Adjacency List of A^2:");
        A.printGraph();
        System.out.println("-------------------------");
        System.out.print("Adjacency List of B^2:");
        B.printGraph();
        System.out.println("-------------------------");
        System.out.print("Adjacency List of C^2:");
        C.printGraph();
    }

    public static int[] fileReader(String fileName) {
        try {
            File file = new File(fileName);
            Scanner fileReader = new Scanner(file);
            int size = 0;
            if (fileReader.hasNextInt())
                size = (int) Math.pow(fileReader.
                   nextInt(), 2);
            int[] list = new int[size];
```

```
            int i = 0;
            while (fileReader.hasNextInt())
                list[i++] = fileReader.nextInt();
            fileReader.close();
            return list;
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.
               getMessage());
        }
        return null;
    }

    public static Adj_List_Graph createGraph(int[]
       input) {
        Adj_List_Graph g = new Adj_List_Graph((int)
           Math.sqrt(input.length));
        for (int i = 0; i < g.n; i++)
            for (int j = 0; j < g.n; j++)
                if (input[i * g.n + j] == 1)
                    g.addEdge(i, j);
        return g;
    }

    public static Adj_List_Graph createGraphSquared(
       Adj_List_Graph g) {
        Adj_List_Graph g2 = new Adj_List_Graph(g);
        for (int i = 0; i < g.n; i++)
            for (int j = 0; j < g.n; j++)
                for (int k = 0; k < g.n; k++)
                    if (g.adj.get(i).contains(j) && g.
                       adj.get(j).contains(k))
                        g2.addEdge(i, k);
        return g2;
    }
}
```

Results Table: Programming Task 1

Data Set #	Adjacency List G^2
7.1	Adjacency list of vertex0: head -> 1 -> 2 Adjacency list of vertex1: head -> 2 Adjacency list of vertex2: head
7.2	Adjacency list of vertex0: head -> 1 -> 2 -> 3 Adjacency list of vertex1: head -> 2 -> 3 -> 4 Adjacency list of vertex2: head Adjacency list of vertex3: head -> 4 Adjacency list of vertex4: head