Tamir Krief, Iaian Milton, Blessing Abumere
COSC 336
11/14/2024

100

Assignment 7

Exercise 1: Radix Sort Example

Sorting the following arrays using the Radix Sort algorithm:

Array 1: 34, 9134, 20134, 29134, 4, 134

Initial	Ones	Tens	Hundreds	Thousands	Ten Thousands
34	4	4	4	4	4
9134	34	34	34	34	34
20134	134	134	134	134	134
29134	9134	9134	9134	9134	9134
4	20134	20134	20134	20134	20134
134	29134	29134	29134	29134	29134

Final sorted array: 4, 34, 134, 9134, 20134, 29134

Array 2: 4, 34, 134, 9134, 20134, 29134

This array is already sorted, so each pass will maintain the same order.

Array 3: 29134, 20134, 9134, 134, 34, 4

Initial	Ones	Tens	Hundreds	Thousands	Ten Thousands
29134	29134	29134	4	4	4
20134	20134	34	34	34	34
9134	9134	9134	134	134	134
134	134	134	9134	9134	9134
34	34	20134	20134	20134	20134
4	4	29134	29134	29134	29134

Final sorted array: 4, 34, 134, 9134, 20134, 29134

Exercise 2: O(n) Sorting Algorithm for Positive Integers

Algorithm:

1. Choose base $k = n$
2. Convert all numbers from base 10 to base k
3. Apply Radix Sort to sort the numbers

Example (a):

Input: 45, 98, 3, 82, 132, 71, 72, 143, 91, 28, 7, 45

$n = 12$ (since $12^2 - 1 = 143$)

Base 10	Base 12	Least Significant	Most Significant	Final (Base 10)
45	39	60	03	3
98	82	70	07	7
3	03	B0	24	28
82	6A	03	39	45
132	B0	24	39	45
71	5B	39	5B	71
72	60	39	60	72
143	BB	79	6A	82
91	79	82	79	91
28	24	5B	82	98
7	07	6A	B0	132
45	39	BB	BB	143

Example (b):

Input: 45, 98, 3, 82, 132, 71, 72, 143, 91, 28, 7, 45, 151, 175, 145, 399, 21, 267, 346, 292

$n = 20$ (since $20^2 - 1 = 399$)

Base 10	Base 20	Least Significant	Most Significant	Final (Base 10)
45	25	11	03	3
98	4I	21	07	7
3	03	25	11	21
82	42	25	1I	28
132	6C	03	25	45
71	3B	73	25	45
72	3C	03	3B	71
143	73	42	3C	72
91	4B	82	42	82
28	1I	92	4B	91
7	07	4B	4I	98
45	25	6C	6C	132
151	7B	3B	73	143
175	8F	3C	75	145
145	75	75	7B	151
399	JJ	7B	8F	175
21	11	8F	D7	267
267	D7	D7	EC	292
346	H6	EC	H6	346
292	EC	H6	JJ	399

This algorithm runs in O(n) time because it performs a constant number of passes (2 in this case) over the n elements, regardless of the size of the numbers.

100

Programming Task 7

Programming Task Example 1

Adjacency matrix of vertex0 head → 0 → 1 → 0 → 0
Adjacency matrix of vertex1 head → 0 → 0 → 1 → 0
Adjacency matrix of vertex2 head → 0 → 0 → 0 → 1
Adjacency matrix of vertex3 head → 0 → 0 → 0 → 0

0: → 1
1: → 2
2: → 3
3:

Computed Adjacency Matrix of Example 1:
Adjacency matrix of vertex0 head → 0 → 1 → 1 → 0
Adjacency matrix of vertex1 head → 0 → 0 → 1 → 1
Adjacency matrix of vertex2 head → 0 → 0 → 0 → 1
Adjacency matrix of vertex3 head → 0 → 0 → 0 → 0

Computed Adjacency List of Example 1:
0: → 1 → 2
1: → 2 → 3
2: → 3
3:

input-7-1.txt:

Adjacency matrix of vertex0 head → 0 → 1 → 0
Adjacency matrix of vertex1 head → 0 → 0 → 1
Adjacency matrix of vertex2 head → 0 → 0 → 0

0: → 1
1: → 2
2:

Computed Adjacency Matrix of input-7-1.txt:
Adjacency matrix of vertex0 head → 0 → 1 → 1
Adjacency matrix of vertex1 head → 0 → 0 → 1
Adjacency matrix of vertex2 head → 0 → 0 → 0

Computed Adjacency List of input-7-1.txt:
0: → 1 → 2
1: → 2
2:

input-7-2.txt:

Adjacency matrix of vertex0 head → 0 → 1 → 0 → 0 → 0
Adjacency matrix of vertex1 head → 0 → 0 → 1 → 1 → 0
Adjacency matrix of vertex2 head → 0 → 0 → 0 → 0 → 0
Adjacency matrix of vertex3 head → 0 → 0 → 0 → 0 → 1
Adjacency matrix of vertex4 head → 0 → 0 → 0 → 0 → 0

0: → 1
1: → 2 → 3
2:
3: → 4
4:

Computed Adjacency Matrix of input-7-2.txt:
Adjacency matrix of vertex0 head → 0 → 1 → 1 → 1 → 0
Adjacency matrix of vertex1 head → 0 → 0 → 1 → 1 → 1
Adjacency matrix of vertex2 head → 0 → 0 → 0 → 0 → 0
Adjacency matrix of vertex3 head → 0 → 0 → 0 → 0 → 1
Adjacency matrix of vertex4 head → 0 → 0 → 0 → 0 → 0

Computed Adjacency List of input-7-2.txt:
0: → 1 → 2 → 3
1: → 2 → 3 → 4
2:
3: → 4
4:

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Scanner;
import java.util.ArrayList;

class Assignment7{
    public static void main(String[] args) {
        //using https://www.cs.usfca.edu/~galles/visualization/DFS.html the print_Graph() function
prints what they call an adjacency matrix and not the adjacency list
        // Both adjacency list and adjacency matrix representationsare printed to be safe
        // in This file the only change made to Adj_List_Graph is to make it a static class
        /*the adjacency matrix of the example in Programming Task
        Should Display
        0 1 1 0
        0 0 1 1
        0 0 0 1
        0 0 0 0
         */
        //example given in Programming Task
        Adj_List_Graph Example_1 = create_Graph(new int[][]{
            {0,1,0,0},
            {0,0,1,0},
            {0,0,0,1},
            {0,0,0,0}
        });

        Adj_List_Graph input7_1 = inputFile("input-7-1.txt");
        Adj_List_Graph input7_2 = inputFile("input-7-2.txt");

        System.err.print("\nExample 1: ");
            Example_1.printGraph();
            Example_1.printList();
        System.err.print("\nComputed Adjacency Matrix of Example 1: ");
            Compute_AdjacencyList(Example_1).printGraph();

        System.err.print("\nComputed Adjacency List of Example 1: ");
            Compute_AdjacencyList(Example_1).printList();


        System.err.print("\n\ninput-7-1.txt: ");
          input7_1.printGraph();
          input7_1.printList();
        System.err.print("\nComputed Adjacency Matrix of input-7-1.txt: ");
            Compute_AdjacencyList(input7_1).printGraph();
        System.err.print("\nComputed Adjacency List of input-7-1.txt: ");
            Compute_AdjacencyList(input7_1).printList();

        System.err.print("\n\ninput-7-2.txt: ");
            input7_2.printGraph();
            input7_2.printList();
        System.err.print("\nComputed Adjacency Matrix of input-7-2.txt: ");
            Compute_AdjacencyList(input7_2).printGraph();
        System.err.print("\nComputed Adjacency List of input-7-2.txt: ");
            Compute_AdjacencyList(input7_2).printList();
    }

    /** reads the file and converts it to an {@link Adj_List_Graph} */
    public static Adj_List_Graph inputFile(String filename) {
        try (Scanner console = new Scanner(new FileReader(filename))) {
            //node count
            int N = console.nextInt();

            //birth of graph of size N
            final Adj_List_Graph GRAPH = new Adj_List_Graph(N);
```

```
            //add edges
            for (int u = 0 ; u < N;u++)
                for (int v = 0; v < N; v++)
                    GRAPH.addEdge(u, console.nextInt()); //(u,v) u is the head , v is a single link

            return GRAPH;
        }catch(FileNotFoundException e) {
            System.err.println("File not found: '" + filename + "'");
        }

        return null;
    }

    /** computes the adjacency list of a <b>directed</b> graph
     * @return G2
     */
    public static Adj_List_Graph Compute_AdjacencyList(final Adj_List_Graph G) {
        final int N = G.n;

        final Adj_List_Graph G2 = new Adj_List_Graph(N);

        //copying G
        for (int u = 0; u < N; u++)
            for (int v : G.adj.get(u))
                G2.addEdge(u, v);

        //calculating G2
        int i,j;
        for (int k = 0; k < N*N; k++) {
            i = k / N; //calc row [i][?]
            j = k % N; //calc column [?][j]

            if (G.adj.get(i).get(j) == 1) {  // If there's an edge at [i][j]
                for (int v = 0; v < N; v++)
                    if (G.adj.get(j).get(v) == 1 && G.adj.get(i).get(v) == 0)  //if there's an edge
at [j][v] but not at [i][v] then the path length is atleast 1
                        G2.adj.get(i).set(v,1);  //replaces [i][v] with 1
            }
        }
       return G2;
    }


    /** exists for manual testing */

    public static Adj_List_Graph create_Graph(final int[][] MATRIX){
        Adj_List_Graph GRAPH = new Adj_List_Graph(MATRIX.length);

        final int N = MATRIX.length;

        for (int u = 0; u< N; u++)
            for (int v = 0; v < N; v++)
                GRAPH.addEdge(u, MATRIX[u][v]);//(u,v) u is the head , v is a single link

        return GRAPH;
    }



    public static class Adj_List_Graph{
        int n;  // no of nodes
        ArrayList<ArrayList<Integer> > adj;

        //constructor taking as the single parameter the number of nodes
```

ok!

```
        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>());

        }


        // A utility function to add an edge in an
        // undirected graph; for directed graph remove the second line
        public void addEdge(int u, int v)
        {
          adj.get(u).add(v);
          // adj.get(v).add(u);  //this line should be un-commented, if graph is undirected
        }

        /** A utility function to print the adjacency <strike>list</strike> <b>Matrix</b>
representation of graph */
        //this function was not changed
        public void printGraph()
        {
          for (int i = 0; i < n; i++) {
            System.out.println("\nAdjacency matrix 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();
          }
        }

        /** A utility function to print the adjacency <b>list</b> representation of graph */
        //this function was added
        public void printList(){

          int vertex = 0;

          for (ArrayList <Integer> u : this.adj) {
              System.out.printf("\n%d: ", vertex);

              for (int v = 0; v < u.size(); v++){
                  if (u.get(v) != 0)
                      System.out.print(" -> " + v);

              }
              System.out.println();

              vertex++;
          }
        }

      }

}
```