*Solution for Assignment 3:*

COMP-352

by

Shadi Jiha (40131284)

Concordia University

Department of Computer Science and Software engineering

18 June 2021

**Question 1:**

1. Content of the array of 13 elements (each index represents a linked list):

[[65], [105], [28, 223], [185, 120], [225, 69], [70, 122, 18, 44], null, [85, 111, 59], [177], [256], [10, 49, 140], [245, 180], [12]]

1. 11 collisions

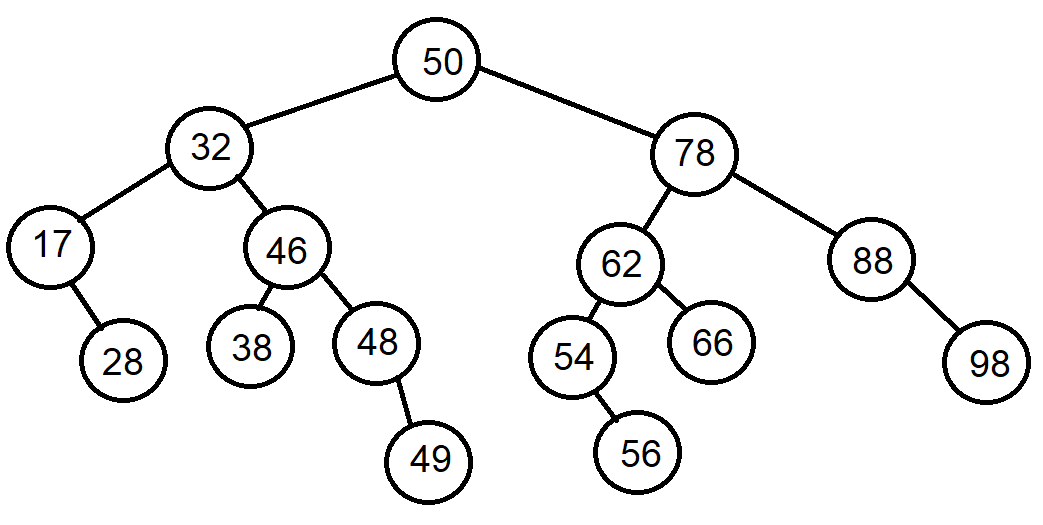
**Question 2:**

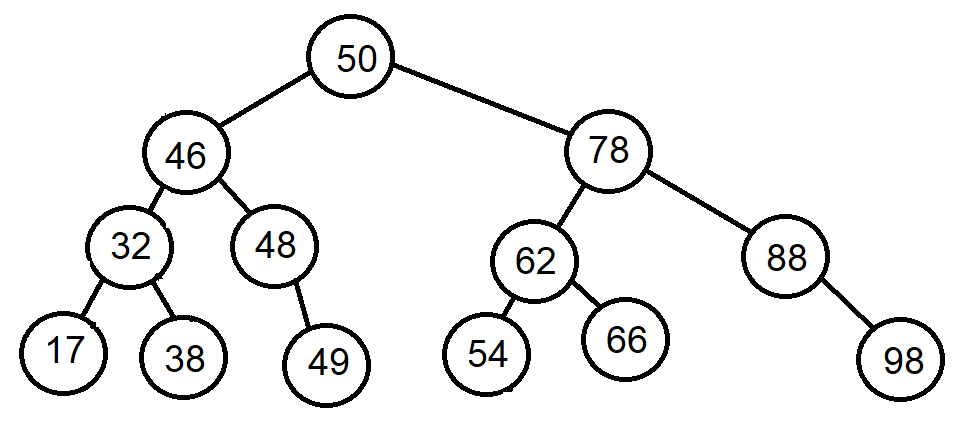
1. [19, 58, null, null, 62, 24, null, null, null, 28, null, 36, null, 47, null, null, null, 17, 37]
2. 3
3. 6 total collisions
4. is the load factor

**Question 3:**

1. [null, 36, 58, null, null, null, null, null, null, 47, null, null, null, null, null, null, null, 17, 37]
2. 2.
3. 3 collisions

**Question 4:**

1. After inserting 56
2. After removing 28 to the initial tree



**Question 5:**

1. Merge sort

First the array is divided as follows:

[4, 12, 19, 26, 47, 53, 63, 74, 89] [8, 15, 17, 50, 71, 82, 87, 93]

[12, 19, 47, 74, 89] [4, 26, 53, 63]

[8, 15, 71, 93] [17, 50, 82, 87]

[12, 47, 74] [19, 89]

[50, 87] [17, 82] [4, 63] [26, 53] [8, 93] [15, 71]

[12, 47] [74]

[12] [47] [19] [89] [26] [53] [71] [15] [87] [50] [17] [82] [4] [63] [8] [93]

Now the array is merged and sorted.

[4, 63]

[8, 93]

[12, 47]

[19, 89]

[26, 53]

[15, 71]

[50, 87]

[17, 82]

[12, 47, 74]

[4, 26, 53, 63]

[8, 15, 71, 93]

[17, 50, 82, 87]

[12, 19, 47, 74, 89]

[8, 15, 17, 50, 71, 82, 87, 93]

[4, 12, 19, 26, 47, 53, 63, 74, 89]

[4, 8, 12, 15, 17, 19, 26, 47, 50, 53, 63, 71, 74, 82, 87, 89, 93]

1. Quick sort:

Pivot: 82. Start: 0. End: 16

[**12, 47, 74, 19, 4, 63, 26, 53, 8, 71, 15, 50, 17, 82, 89, 93, 87**]

Pivot: 93. Start: 0. End: 15

[**12, 47, 74, 19, 4, 63, 26, 53, 8, 71, 15, 50, 17, 82, 89, 93**, 87]

Pivot: 89. Start: 0. End: 14

**[12, 47, 74, 19, 4, 63, 26, 53, 8, 71, 15, 50, 17, 82, 89**, 93, 87]

Pivot: 82. Start: 0. End: 13

[**12, 47, 74, 19, 4, 63, 26, 53, 8, 71, 15, 50, 17, 82**, 89, 93, 87]

Pivot: 17. Start: 0. End: 12

[**12, 4, 8, 15, 17, 63, 26, 53, 74, 71, 19, 50, 47**, 82, 89, 93, 87]

Pivot: 50. Start: 0. End: 11

[**12, 4, 8, 15, 17, 26, 19, 50, 74, 71, 63, 53**, 47, 82, 89, 93, 87]

Pivot: 63. Start: 0. End: 10

[**12, 4, 8, 15, 17, 26, 19, 50, 63, 71, 74**, 53, 47, 82, 89, 93, 87]

Pivot: 71. Start: 0. End: 9

[**12, 4, 8, 15, 17, 26, 19, 50, 63, 71**, 74, 53, 47, 82, 89, 93, 87]

Pivot: 63. Start: 0. End: 8

[**12, 4, 8, 15, 17, 26, 19, 50, 63**, 71, 74, 53, 47, 82, 89, 93, 87]

Pivot: 50. Start: 0. End: 7

[**12, 4, 8, 15, 17, 26, 19, 50**, 63, 71, 74, 53, 47, 82, 89, 93, 87]

Pivot: 19. Start: 0. End: 6

[**12, 4, 8, 15, 17, 19, 26**, 50, 63, 71, 74, 53, 47, 82, 89, 93, 87]

Pivot: 19. Start: 0. End: 5

[**12, 4, 8, 15, 17, 19**, 26, 50, 63, 71, 74, 53, 47, 82, 89, 93, 87]

Pivot: 17. Start: 0. End: 4

[**12, 4, 8, 15, 17**, 19, 26, 50, 63, 71, 74, 53, 47, 82, 89, 93, 87]

Pivot: 15. Start: 0. End: 3

[**12, 4, 8, 15,** 17, 19, 26, 50, 63, 71, 74, 53, 47, 82, 89, 93, 87]

Pivot: 8. Start: 0. End: 2

[**4, 8, 12**, 15, 17, 19, 26, 50, 63, 71, 74, 53, 47, 82, 89, 93, 87]

Pivot: 8. Start: 0. End: 1

[**4, 8,** 12, 15, 17, 19, 26, 50, 63, 71, 74, 53, 47, 82, 89, 93, 87]

Final:

[4, 8, 12, 15, 17, 19, 26, 47, 50, 53, 63, 71, 74, 82, 87, 89, 93]

1. Bucket sort

We put data in 10 buckets. Because we know the range, we can use the following “hash” function to distribute the values:

[4, 8]

[12, 19, 15, 17]

[26]

[]

[47]

[53, 50]

[63]

[74, 71]

[89, 87, 82]

[93]

Now we merge the buckets after sorting each one individually and we get the sorted array.

1. Radix sort

Original array:

[12, 47, 74, 19, 89, 4, 63, 26, 53, 8, 93, 71, 15, 87, 50, 17, 82]

Using the first digit

[50, 71, 12, 82, 63, 53, 93, 74, 4, 15, 26, 47, 87, 17, 8, 19, 89]

Using the second digit

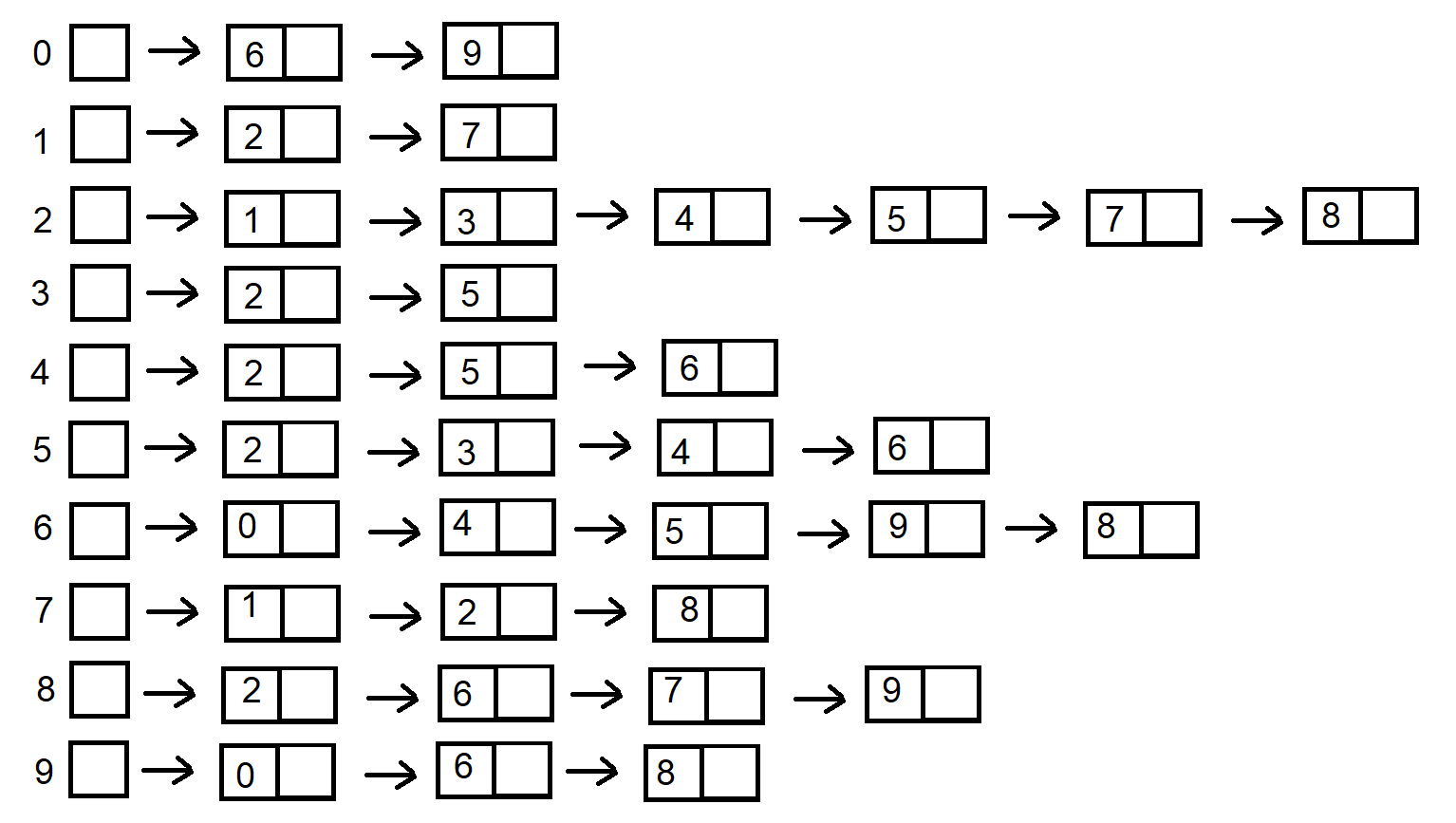
[4, 8, 12, 15, 17, 19, 26, 47, 50, 53, 63, 71, 74, 82, 87, 89, 93]

**Question 6:**

1. Matrix representation (empty squares are 0):

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  | 1 |  |  | 1 |
| 1 |  |  | 1 |  |  |  |  | 1 |  |  |
| 2 |  | 1 |  | 1 | 1 | 1 |  | 1 | 1 |  |
| 3 |  |  | 1 |  |  | 1 |  |  |  |  |
| 4 |  |  | 1 |  |  | 1 | 1 |  |  |  |
| 5 |  |  | 1 | 1 | 1 |  | 1 |  |  |  |
| 6 | 1 |  |  |  | 1 | 1 |  |  | 1 | 1 |
| 7 |  | 1 | 1 |  |  |  |  |  | 1 |  |
| 8 |  |  | 1 |  |  |  | 1 | 1 |  | 1 |
| 9 | 1 |  |  |  |  |  | 1 |  | 1 |  |

1. List representation (with linkedlist):



1. Using breadth-first tree starting at 0:

Trace (This is the content of the queue):

[0]

[6, 9]

[9, 4, 5, 8]

[4, 5, 8]

[5, 8, 2]

[8, 2, 3]

[2, 3, 7]

[3, 7, 1]

[7, 1]

[1]

Output: 0 6 9 4 5 8 2 3 7 1

1. Using depth-first search tree (also starting at 0):

The number to the left is the head of the stack **that is not visited** (the element that will be popped in the current iteration). The elements inside [] are the reset of the stack.

[]

0 [6]

9 [6, 0, 6]

8 [6, 0, 6, 2, 6]

7 [6, 0, 6, 2, 6, 1]

2 [6, 0, 6, 2, 6, 1, 1, 3, 4]

5 [6, 0, 6, 2, 6, 1, 1, 3, 4, 2, 3, 4]

6 [6, 0, 6, 2, 6, 1, 1, 3, 4, 2, 3, 4, 0]

4 [6, 0, 6, 2, 6, 1, 1, 3, 4, 2]

3 [6, 0, 6, 2, 6, 1]

1

Output: 0 9 8 7 2 5 6 4 3 1

**Question 7:**

In order to get these answers, I have implemented the Dijkstra’s Algorithm’s pseudo code in java and ran it on the provided graph.

|  |  |  |
| --- | --- | --- |
| Source | Destination | Distance |
| H | A | 2 |
| H | G | 2 |
| H | I | 1 |
| H | B | 8 |
| H | F | 3 |
| H | E | 6 |
| H | J | 3 |
| H | D | 6 |
| H | C | 11 |

Here is the code I used:

public static Map<WUGraph.Vertex, Integer> dijkstraAlgorithm(WUGraph graph, WUGraph.Vertex vertex) {  
  
 Map<WUGraph.Vertex, Integer> D = new HashMap<>();  
 for (var v : graph.getVertices()) {  
 D.put(v, Integer.MAX\_VALUE);  
 }  
 D.put(vertex, 0);  
  
 PriorityQueue<WUGraph.Vertex> Q = new PriorityQueue<>(((o1, o2) -> o1.getLabel().compareTo(o2.getLabel())));

while (!Q.isEmpty()) {  
 var u = Q.poll();  
 for (var v : u.getAdjVertecies(Q)) {  
 if (D.get(u) + u.getWeightTo(v) < D.get(v)) {  
 D.put(v, D.get(u) + u.getWeightTo(v));  
 }  
 }  
 }  
  
 return D;  
}