COIS 3020H

Assignment 2

Fall 2024

Bachelor Of Computer Science, Trent University

COIS-3020H: Data Structures and Algorithms II

Professor : Sheikh Ahmed Munieb

12th November, 2024

Dikshith Reddy Macherla - Student Id : 0789055

Dev Patel - Student Id : 0824874

Work of Each Group Member:

Question 1a & 3 : Dikshith Reddy

Question 1b & 2 : Dev Patel

Assignment 2

Question 1

A. **The landing reservation requests are:**

45, 50, 55, 65, 75, 85, 87, 95, 97, 105, 115, 118, 121, 125

The steps to solve this problem using AVL Trees

1. **Insert Each Node:**

- For each landing reservation, let’s insert it into the AVL Tree.
- After each insertion, let's check the balance factor of each node to ensure that the tree remains balanced.

2. **Perform Rotations if Needed:**

- If the tree becomes unbalanced (balance factor is not in the range [-1, 1]), let’s perform the appropriate rotation(s):
    - Single Rotation (Left or Right)
    - Double Rotation (Left-Right or Right-Left)

Detailed Steps for Each Insertion

Let's begin with inserting each node, and rotations needed for each if necessary.

Step 1: Insert 45

- 45 is the root as it's the first element. No rotations are needed.

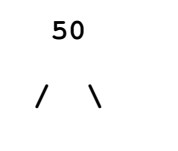
Step 2: Insert 50

- 50 goes to the right of 45. No rotations are needed since the tree is still balanced.

Step 3: Insert 55

- 55 goes to the right of 50.
- This results in an imbalance at node 45 (Right-Right case), so perform a Left Rotation on 45.

After rotation:



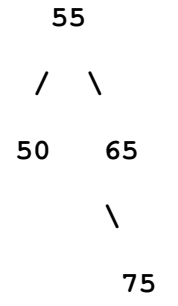
Step 4: Insert 65

- 65 goes to the right of 55.
- No rotations are needed as the tree is balanced.

Step 5: Insert 75

- 75 goes to the right of 65, causing an imbalance at node 50 (Right-Right case), so perform a Left Rotation on 50.

After rotation:



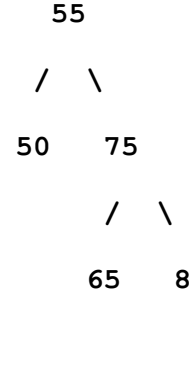
Step 6: Insert 85

- 85 goes to the right of 75.
- No rotations are needed as the tree remains balanced.

Step 7: Insert 87

- 87 goes to the right of 85, causing an imbalance at node 65 (Right-Right case), so perform a Left Rotation on 65.

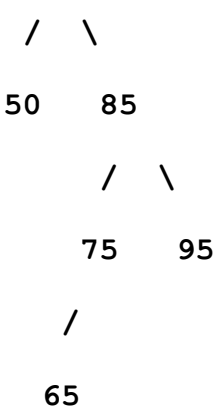
After rotation:



Step 8: Insert 95

- 95 goes to the right of 87.
- This causes an imbalance at node 75, requiring a Left Rotation on 75.

After rotation:



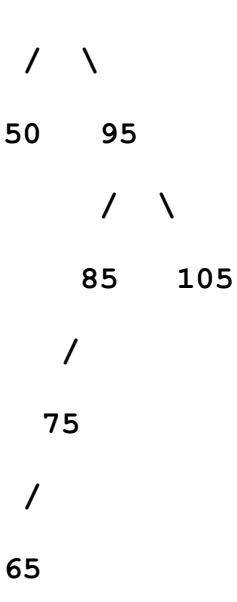
Step 9: Insert 97

- 97 goes to the right of 95.
- No rotations are needed as the tree remains balanced.

Step 10: Insert 105

- 105 goes to the right of 97, causing an imbalance at node 85, so we perform a Left Rotation on 85.

After rotation:



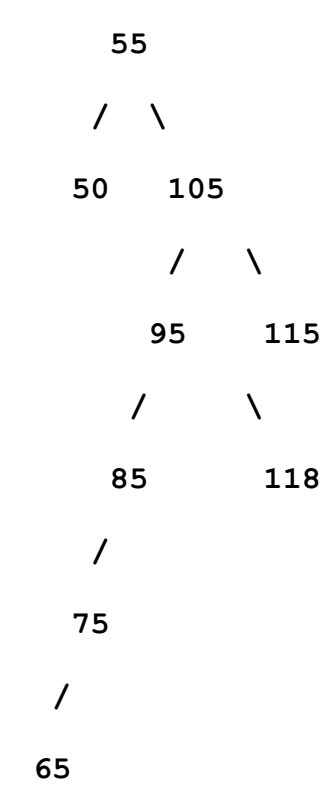
Step 11: Insert 115

- 115 goes to the right of 105.
- No rotations are needed as the tree is balanced.

Step 12: Insert 118

- 118 goes to the right of 115, causing an imbalance at node 95 (Right-Right case), so perform a Left Rotation on 95.

After rotation:



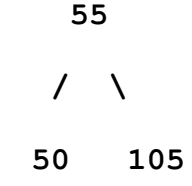
Step 13: Insert 121

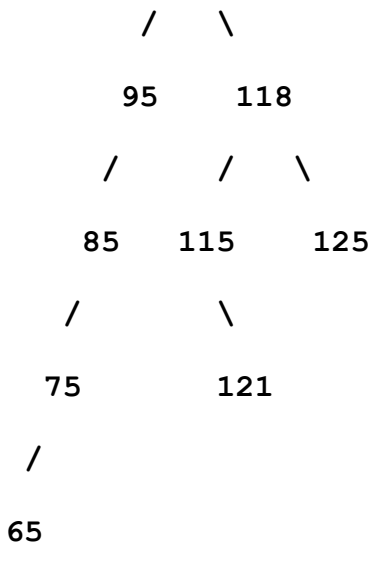
- 121 goes to the right of 118.
- No rotations are needed as the tree remains balanced.

Step 14: Insert 125

- 125 goes to the right of 121, causing an imbalance at node 115 (Right-Right case), so perform a Left Rotation on 115.

Final AVL Tree after all insertions:





B. Steps for Double Hashing

In this scenario, we have the same list of landing reservation requests:

45, 50, 55, 65, 75, 85, 87, 95, 97, 105, 115, 118, 121, 125

1. **Choosing the Hash Function:**
   - We can use a primary hash function $h_1(x)$ and a secondary hash function $h_2(x)$
   - A typical primary hash function for simplicity could be:

     $$h_1(x) = x \bmod m$$
   - A typical secondary hash function is: $h_2(x) = 1 + (x \bmod (m-1))$
   - Here, m is the size of the hash table, which we need to decide based on the number of elements. We have 14 elements, so selecting m as the next prime greater than 14 (say m = 17) is reasonable.
2. **Apply Double Hashing:**
   - For each landing request, compute the initial position using $h_1(x)$
   - If there's a collision (i.e., the slot is occupied), use the secondary hash function $h2(x)$ to determine the next slot by probing in steps of $h2(x)$
3. **Insertion Example:**

- For each landing reservation, calculate the index and attempt insertion, handling collisions with double hashing.

Detailed Insertion Steps

Let's go through a few insertions to demonstrate:

Insert 45:

- $h_1(45) = 45 \bmod 17 = 11$
- (Position 11 is empty, so insert 45 at index 11.

Insert 50:

- $h_1(50) = 50 \bmod 17 = 16$
- Position 16 is empty, so insert 50 at index 16.

Insert 55:

- $h_1(55) = 55 \bmod 17 = 4$
- Position 4 is empty, so insert 55 at index 4.

Insert 65:

- $h_1(65) = 65 \bmod 17 = 14$
- Position 14 is empty, so insert 65 at index 14.

Insert 75:

- $h1_1(75) = 75 \bmod 17 = 7$
- Position 7 is empty, so insert 75 at index 7.

Continue this process for all entries, handling any collisions by using $h_2(x)$ to probe for the next available slot.

Time Complexity

In the **average case**, double hashing provides a time complexity of $O(1)$ for insertion, search, and deletion when the load factor (ratio of filled slots to total slots) is low. This is because double hashing minimizes clustering and spreads elements uniformly across the table.

In the **worst case**, when the table becomes full or nearly full, the time complexity can approach O(n), where n is the number of elements, because we may have to probe multiple slots before finding an open one.

However, with a properly chosen hash table size and low load factor, double hashing can achieve efficient O(1) operations.

Question 2

We need to create a hash table of size S=20 and resolve collisions using three different methods: **Chaining**, **Linear Probing**, and **Double Hashing**. The hash function used is k mod 20.

The keys given are:

45, 50, 55, 65, 75, 85, 87, 95, 97, 105, 115, 118, 121, 125

1. **Chaining**

In chaining, each slot in the hash table points to a linked list of keys that hash to the same index. For each key, compute its hash index and append it to the list at that index.

Calculating Positions for Chaining

Using k mod 20:

45 mod 20 = 5

50 mod 20 = 10

55 mod 20 = 15

65 mod 20 = 5

75 mod 20 = 15

85 mod 20 = 5

87 mod 20 = 7

95 mod 20 = 15

97 mod 20 = 17

105 mod 20 = 5

115 mod 20 = 15

118 mod 20 = 18

121 mod 20 = 1

125 mod 20 = 5

Hash Table with Chaining

The table will look like this:

Index	**Values**
0	
1	121
2	
3	
4	
5	45 → 65 → 85 → 105 → 125
6	
7	87
8	
9	
10	50
11	
12	
13	
14	
15	55 → 75 → 95 → 115
16	
17	97
18	118

19	

2. **Linear Probing**

In linear probing, if a collision occurs, we check the next slot in a sequential manner until an empty slot is found.

Calculating Positions for Linear Probing

Using k mod 20 and handling collisions by moving to the next available slot:

1. 45 mod 20 = 5 → Insert at index 5
2. 50 mod 20 = 10 → Insert at index 10
3. 55 mod 20 = 15 → Insert at index 15
4. 65 mod 20 = 5 (collision) → Move to 6 → Insert at index 6
5. 75 mod 20 = 15 (collision) → Move to 16 → Insert at index 16
6. 85 mod 20 = 5 (collision) → Move to 7 → Insert at index 7
7. 87 mod 20 = 7 (collision) → Move to 8 → Insert at index 8
8. 95 mod 20 = 15 (collision) → Move to 17 → Insert at index 17
9. 97 mod 20 = 17 (collision) → Move to 18 → Insert at index 18
10. 105 mod 20 = 5 (collision) → Move to 9 → Insert at index 9
11. 115 mod 20 = 15 (collision) → Move to 19 → Insert at index 19
12. 118 mod 20 = 18 (collision) → Move to 0 → Insert at index 0
13. 121 mod 20 = 1 → Insert at index 1
14. 125 mod 20 = 5 (collision) → Move to 11 → Insert at index 11

Hash Table with Linear Probing

The table will look like this:

Index	**Values**

0	118
1	121
2	
3	
4	
5	45
6	65
7	85
8	87
9	105
10	50
11	125
12	
13	
14	
15	55
16	75
17	95
18	97
19	115

3. Double Hashing

For double hashing, we use a second hash function to calculate the step size, which helps avoid primary clustering.

Let:

- Primary hash function: $h_1(x) = x \bmod 20$
- Secondary hash function: $h_2(x) = 1 + (x \bmod 19)$

Calculating Positions for Double Hashing

1. 45: $h_1(45) = 5, h_2(45) = 8$ → Insert at index 5
2. 50: $h_1(50) = 10, h_2(50) = 12$ → Insert at index 10
3. 55: $h_1(55) = 15, h_2(55) = 18$ → Insert at index 15
4. 65: $h_1(65) = 5$ (collision) → Move by 8 → Insert at index 13
5. 75: $h_1(75) = 15$ (collision) → Move by 18 → Insert at index 13 (collision) → Move by 18 again → Insert at index 11
6. 85: $h_1(85) = 5$ (collision) → Move by 8 → Insert at index 13 (collision) → Move by 8 again → Insert at index 1
7. 87: $h_1(87) = 7, h_2(87) = 7$ → Insert at index 7
8. 95: $h_1(95) = 15$ (collision) → Move by 18 → Insert at index 13 (collision) → Move by 18 → Insert at index 11 (collision) → Move by 18 → Insert at index 9
9. 97: $h_1(97) = 17, h_2(97) = 3$ → Insert at index 17
10. 105: $h_1(105) = 5$ (collision) → Move by 8 → Insert at index 13 (collision) → Move by 8 → Insert at index 1 (collision) → Move by 8 → Insert at index 9 (collision) → Move by 8 → Insert at index 17 (collision) → Move by 8 → Insert at index 9

Question 3

solving Question 3 using Treaps

Lets perform insertions based on both key values and priority values, As Treaps combine the properties of a Binary Search Tree (BST) and a Heap.

Key Points of Treaps:

1. **BST Property:** Nodes are arranged such that for any node, all values in the left subtree are smaller, and all values in the right subtree are larger.
2. **Heap Property:** Each node has a priority, and each node's priority should be greater than the priorities of its children.

The insertion process will involve:

1. Inserting the node based on the key value (to maintain BST structure).
2. Rotating nodes (if necessary) to maintain the heap property based on priorities.

Given Input

Each item is represented as (key,priority)(key, priority)(key,priority):

[(45,12), (50,3), (55,13), (65,8), (75,5), (85,2), (87,10), (95,4), (97,1), (105,9), (115,6), (118,7), (121,14), (125,11)]

Step-by-Step Process

Let’s go through each insertion, using rotations where necessary.

1. Insert (45, 12)

- Since the treap is empty, let's insert (45, 12) as the root.

2. Insert (50, 3)

- 50 > 45, so let’ insert (50, 3) as the right child of 45.
- No rotations needed because the priority of 50 (3) is less than 45 (12).

3. Insert (55, 13)

- 55 > 45, so let’s move right to 50.
- 55 > 50, so let’s insert (55, 13) as the right child of 50.

- **Rotation Needed**: 55 has a higher priority than 50, so perform a **left rotation** on 50.
    - After rotation, 55 becomes the right child of 45, and 50 is the left child of 55.

4. Insert (65, 8)

- 65 > 45, so let’s move right to 55.
- 65 > 55, so let’s insert (65, 8) as the right child of 55.
- No rotations needed as the priority of 65 (8) is less than 55 (13).

5. Insert (75, 5)

- 75 > 45, so let’s move right to 55.
- 75 > 55, so let’s move right to 65.
- 75 > 65, so let’s insert (75, 5) as the right child of 65.
- No rotations needed because 75 has a lower priority than 65.

6. Insert (85, 2)

- 85 > 45, so let’s move right to 55.
- 85 > 55, so let’s move right to 65.
- 85 > 65, so let’s move right to 75.
- 85 > 75, so let's insert (85, 2) as the right child of 75.
- No rotations needed as the priority of 85 (2) is less than 75 (5).

7. Insert (87, 10)

- 87 > 45, so let’s move right to 55.
- 87 > 55, so let’s move right to 65.
- 87 > 65, so let’s move right to 75.
- 87 > 75, so let’s move right to 85.
- 87 > 85, so let’s insert (87, 10) as the right child of 85.
- **Rotation Needed**: 87 has a higher priority than 85, so perform a **left rotation** on 85.
    - After rotation, 87 becomes the right child of 75, and 85 becomes the left child of 87.

8. Insert (95, 4)

- 95 > 45, so let’s move right to 55.
- 95 > 55, so let’s move right to 65.
- 95 > 65, so let’s move right to 75.
- 95 > 75, so let’s move right to 87.
- 95 > 87, so let’s insert (95, 4) as the right child of 87.
- No rotations needed as the priority of 95 (4) is less than 87 (10).

9. Insert (97, 1)

- 97 > 45, so let’s move right to 55.
- 97 > 55, so let’s move right to 65.
- 97 > 65, so let’s move right to 75.
- 97 > 75, so let’s move right to 87.

- 97 > 87, so let’s move right to 95.
- 97 > 95, so let’s insert (97, 1) as the right child of 95.
- No rotations needed as the priority of 97 (1) is less than 95 (4).

10. Insert (105, 9)

- 105 > 45, so let’s move right to 55.
- 105 > 55, so let’s move right to 65.
- 105 > 65, so let’s move right to 75.
- 105 > 75, so let’s move right to 87.
- 105 > 87, so let’s move right to 95.
- 105 > 95, so let’s move right to 97.
- 105 > 97, so let’s insert (105, 9) as the right child of 97.
- **Rotation Needed**: 105 has a higher priority than 97, so perform a **left rotation** on 97.
    - After rotation, 105 becomes the right child of 95, and 97 becomes the left child of 105.

11. Insert (115, 6)

- 115 > 45, so let’s move right to 55.
- 115 > 55, so let’s move right to 65.
- 115 > 65, so let’s move right to 75.
- 115 > 75, so let’s move right to 87.
- 115 > 87, so let’s move right to 95.

- 115 > 95, so let’s move right to 105.
- 115 > 105, so let’s insert (115, 6) as the right child of 105.
- No rotations needed as the priority of 115 (6) is less than 105 (9).

12. Insert (118, 7)

- 118 > 45, so let’s move right to 55.
- 118 > 55, so let’s move right to 65.
- 118 > 65, so let’s move right to 75.
- 118 > 75, so let’s move right to 87.
- 118 > 87, so let’s move right to 95.
- 118 > 95, so let’s move right to 105.
- 118 > 105, so let’s move right to 115.
- 118 > 115, so let’s insert (118, 7) as the right child of 115.
- **Rotation Needed**: 118 has a higher priority than 115, so perform a **left rotation** on 115.
    - After rotation, 118 becomes the right child of 105, and 115 becomes the left child of 118.

13. Insert (121, 14)

- 121 > 45, so let’s move right to 55.
- 121 > 55, so let’s move right to 65.
- 121 > 65, so let’s move right to 75.
- 121 > 75, so let’s move right to 87.

- 121 > 87, so let’s move right to 95.
- 121 > 95, so let’s move right to 105.
- 121 > 105, so let’s move right to 118.
- 121 > 118, so let’s insert (121, 14) as the right child of 118.
- **Rotation Needed**: 121 has a higher priority than 118, so perform a **left rotation** on 118.
    - After rotation, 121 becomes the right child of 105, and 118 becomes the left child of 121.

14. Insert (125, 11)

- 125 > 45, so let’s move right to 55.
- 125 > 55, so let’s move right to 65.
- 125 > 65, so let’s move right to 75.
- 125 > 75, so let’s move right to 87.
- 125 > 87, so let’s move right to 95.
- 125 > 95, so let’s move right to 105.
- 125 > 105, so let’s move right to 121.
- 125 > 121, so let’s insert (125, 11) as the right child of 121.
- No rotations needed as the priority of 125 (11) is less than 121 (14).