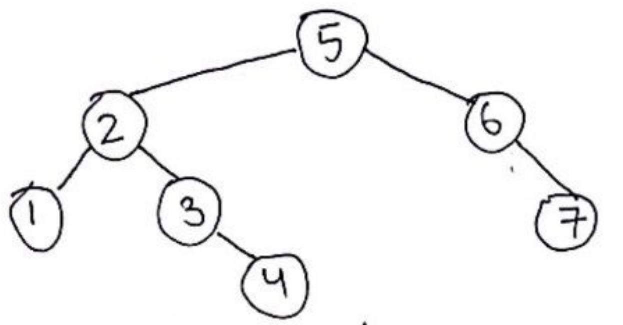
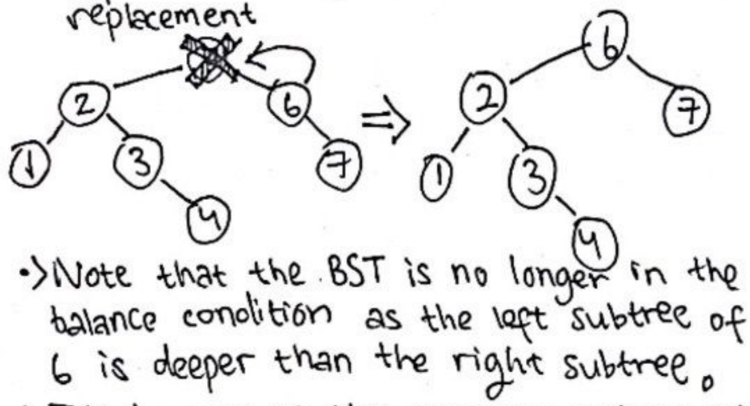
Yohandi (120040025)

CSC3100 Assignment 3

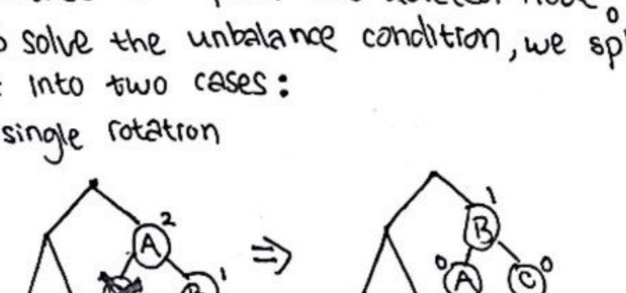
1)



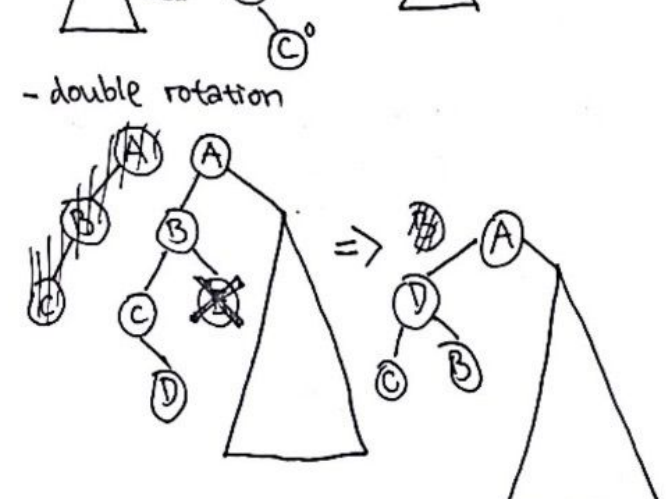
- Define a balance condition as a case where the balance factor of every node is $\in \{-1, 0, 1\}$. Then, the current BST is already in a balance condition.
- Suppose we want to delete node 5 from the BST. Then, based on the successor replacement rule, we will take 6 as a replacement

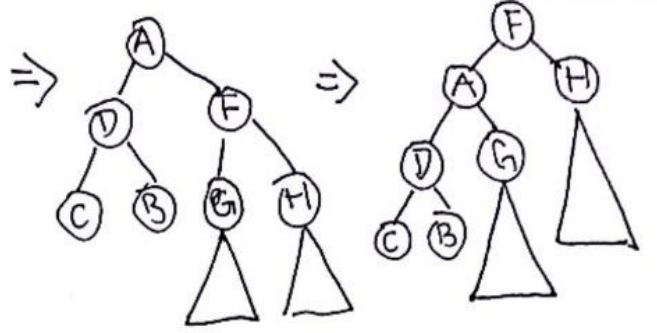


- Note that the BST is no longer in the balance condition as the left subtree of 6 is deeper than the right subtree.
- This happens as the successor replacement rule always uses all nodes on the right subtree to replace the deleted node.
- To solve the unbalance condition, we split it into two cases:
  - single rotation



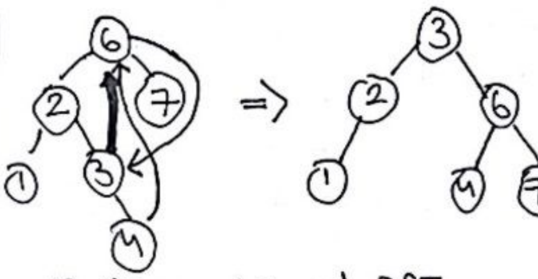
  - double rotation





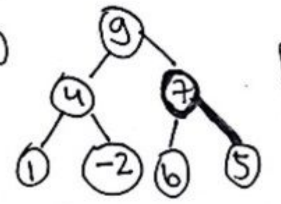
general steps:
→ search node that is going to be deleted
→ delete and replace with successor
→ correct balance factor
→ if imbalance #1, then perform single rotation
→ if imbalance #2, then perform double rotation

- with that, we have:

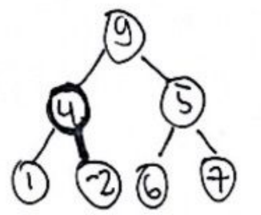


as the new balanced BST

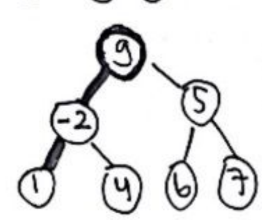
2)



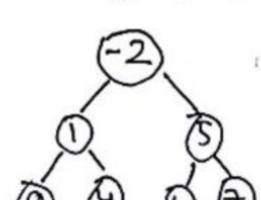
| | | | | | | |
|---|---|---|---|---|---|---|
| 9 | 4 | 7 | 1 | -2 | 6 | 5 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |



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



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



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

pseudocode ↺

```
procedure percolate(arr[], hole):
    child = -1
    tmp = arr[hole]
    while 2·hole ≤ arr size:
        child = 2·hole   // left child
        if child ≠ arr size and arr[child+1] ≥ arr[child]:
            child = child + 1   // right child
        if arr[child] ≥ tmp:
            arr[hole] = arr[child]
        else:
            break
        hole = child
    arr[hole] = tmp
```

```
main:
    arr[] = max-heap representation
    for(i = ⌊arr size / 2⌋; i≥1; i--):
        percolate(arr, i)
```

Complexity Analysis:

•) percolate(n) = $\begin{cases} \log_2(N) & , n=1 \\ \log_2(N)-1 & , n=2,3 \\ \log_2(N)-2 & , n=4,5,6,7 \\ \vdots & \vdots \\ 1 & , n=2^{\log_2(N)-1}, \dots, N \end{cases}$

where N = size of arr

•) $\sum_{i=1}^{\lfloor N/2 \rfloor} \text{percolate}(i) \le \sum_{i=1}^{2^{k-1}} \text{percolate}(i)$, where $2^{k-1} \le N \le 2^k$

$= \sum_{i=1}^{k-1}\left(\sum_{j=1}^{2^{i-1}} \text{percolate}(2^{i-1}+j-1)\right)$

$= \sum_{i=1}^{k-1}\left(\sum_{j=1}^{2^{i-1}} (k-i)\right)$

$\le \sum_{i=1}^{k-1}\left(\frac{2^k}{2^{k-i+1}}(k-i)\right)$

$= \sum_{i=1}^{k-1}\left(\frac{2^k}{2^{i+1}}\cdot i\right)$

$= 2^k \sum_{i=1}^{k-1} \frac{i}{2^{i+1}}$

$= 2^{k-1} \sum_{i=1}^{k-1} \frac{i}{2^i}$

$\le 2^{k-1}\cdot 2$

$= 2^k$

$= N + (2^k - N)$

$\le N + (2^{k-1})$

$= O(N)$ since $N \ge 2^{k-1}$

3) Yes, there is. However, note that, in a case where all keys are randomly distributed, the prime number will not have an expected benefit.
In its application, these prime numbers are used to minimize the number of collisions. ~~due to [illegible] data~~

This is possible as many data in this world exhibit some particular patterns that we may or may not notice.

Example: ~~suppose, we want to h(x) for: x=~~

suppose we want to find h(x) for $x \in \{3, 6, 9, 12, 15, 18\}$, then:

•) for p=6,

$h(3) = 3 \% 6 = 3$
$h(6) = 6 \% 6 = 0$
$h(9) = 9 \% 6 = 3$
$h(12) = 12 \% 6 = 0$
$h(15) = 15 \% 6 = 3$
$h(18) = 18 \% 6 = 0$

we have $h(x) \in \{0, 3\}$

•) for p = 5 (prime number),

$h(3) = 3 \% 5 = 3$
$h(6) = 6 \% 5 = 1$
$h(9) = 9 \% 5 = 4$
$h(12) = 12 \% 5 = 2$
$h(15) = 15 \% 5 = 0$
$h(18) = 18 \% 5 = 3$

we have $h(x) \in \{0, 1, 2, 3, 4\}$

based on the possible values for h(x), we can judge that in our case, prime number serves as a better choice in hash function

For general perspective, we can first notice that prime numbers are unique in a sense that those numbers are not divisible by any other numbers aside from 1 and itself.

Then, multiplying it with any number will make the results quite rare. Because of that reason, for list of keys that we obtain, it is quite rare to have the same modulus result with prime number in a data that has patterns. This also implies the possibilities of less collisions

4) Linear probe:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 34 | 0 | 45 | | | 1001 | 6 | 23 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7 | | | 28 | 12 | 29 | 11 | 30 | 33 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

records:

→ h(6,0) = 6 % 17 = 6
→ h(12,0) = 12 % 17 = 12
→ h(29,0) = 29 % 17 = 12 (occupied)
h(29,1) = 30 % 17 = 13
→ h(28,0) = 28 % 17 = 11
→ h(34,0) = 34 % 17 = 0
→ h(11,0) = 11 % 17 = 11 (occupied)
⋮ (occupied)
h(11,3) = 14 % 17 = 14
→ h(23,0) = 23 % 17 = 6 (occupied)
h(23,1) = 24 % 17 = 7
→ h(7,0) = 7 % 17 = 7 (occupied)
h(7,1) = 8 % 17 = 8
→ h(0,0) = 0 % 17 = 0 (occupied)
h(0,1) = 1 % 17 = 1
→ h(33,0) = 33 % 17 = 16
→ h(30,0) = 30 % 17 = 13 (occupied)
⋮
h(30,2) = 32 % 17 = 15
→ h(45,0) = 45 % 17 = 11 (occupied)
(occupied)
⋮
h(45,8) = 53 % 17 = 2
h(10001,0) = 10001 % 17 = 5

Total hits = 1+3+1+1+1+2+8 = 17

Double hashing:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 29 | 0 | | | | 34 | 6 | 7 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1001 | 23 | 28 | 12 | 11 | 30 | 45 | 33 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

records:

→ h(6,0) = 6
→ h(12,0) = 12
→ h(29,0) = 12 (occupied)
h(29,1) = 0
→ h(28,0) = 11
→ h(34,0) = 0 (occupied)
h(34,1) = 5
→ h(11,0) = 11 (occupied)
h(11,1) = 13
→ h(23,0) = 6 (occupied)
h(23,1) = 10
→ h(7,0) = 7
→ h(0,0) = 0 (occupied)
h(0,1) = 1
→ h(33,0) = 16
→ h(30,0) = 13 (occupied)
h(30,1) = 14
→ h(45,0) = 11 (occupied)
⋮ (occupied)
h(45,4) = 15
→ h(10001,0) = 5 (occupied)
⋮ (occupied)
h(10001,2) = 9

Total hits = 1+1+1+1+1+1+4+2 = 12

Based on the total hits, double hashing method seems to provide a better collision handling technique. Moreover, in such a case, using linear probe means that there are way bigger chances that the collisions are counted repetitively.

However, I still prefer the linear probe method.

Reason: → To reduce the number of collisions when hashing, we can use a data structure Disjoint-set like that allows at most one collision per hashing (It is done by saving the ancestor of which a parent is defined as the next index)

5 ) Deletion on both linear probe and double hashing can be done by simply change the deleted value with something else that marks the place being deleted. For example using -1 to mark.

Note that this simple trick solves the case where there might be two keys locating in the hash position and the first one getting deleted so that it causes confusion. By marking the deleted spot, the search and insert will still assume that the spot is taken; hence, there will be no ambiguogity for two same keys case. This technique is known as tombstone deletion.