DSA-Hw4

Problem 0 - Proper References

- Problem 1-4: https://www.geeksforgeeks.org/find-the-largest-subarray-with-0-sum/

Problem 1 - Hash Table and Disjoint Set

1.a

keys to be inserted \ index	0	1	2	3	4	5	6
42	42	-	-	-	-	11	-
11	42	-	-	-	-	11	25
25	42	-	-	-	-	11	25
1	42	-	-	1	-	11	25
56	42	56	-	1	-	11	25
70	42	56	70	1	-	11	25
19	42	56	70	1	19	11	25

1.b

keys to be inserted \ index	0	1	2	3	4	5	6
42	42	-	-	-	-	11	-
11	42	-	-	-	-	11	-
25	42	-	25	-	-	11	-
1	42	-	25	1	-	11	-
56	42	56	25	1	-	11	-
70	42	56	25	1	-	11	70
19	42	56	25	1	19	11	70

2.

將房間編號視為二進制的位數， light ON 時為 1， light OFF 時為 0，將light pattern 以17位數的二進位制來表示。 如範例 light pattern {1, 3, 5, 7, 17}可表示為:

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1

轉換後 index 為 65621。 由於組成 light pattern 的整數代表在二進位表示中該位數為1或-1，因此lighht pattern 中的整數之間沒有排列關係， 且該方法可表示的最大位數為17位，因此可表示的最大值為131071，不會超過上限133333。

3.

HashMap 用來存 {cumSum: index}，(array index 從 1 開始)。
step1: Insert(0,0) step2: 從index = 1 開始到 index = arrray length，計算累加到 i-th element 的累加值 cumSum， 若 (cumSum - E) 為 HashMap 中存在的 key，則更新 $maxLen = max(maxLen, (i - Get(cumSum - E)))$，若 cumSum 不是HashMap 中存在的 key，則 Insert(cumSum, i)

pseudo code:

```
FindMaxLen(array a):
    HashMap = empty hash map
    HashMap.Insert(0, 0)
    MaxLen = 0
    cumSum = 0
    for i = 1 to a.length:
        cumSum = cumSum + a[i]
        if cumSum not in HashMap:
            Insert(cumSum, i)
        if (cumSum - E) in HashMap:
            MaxLen = max(MaxLen, i - HashMap.Get(cumSum - E))
    return MaxLen
```

time complexity:
因 Insert, Get 的 time complexcity 都為 $O(1)$，因此 $FindMaxLen$ 的 time complexcity 為 $O(n)$ HashMap額外使用的空間最大為 $O(n)$，因此 space complexcity 為 $O(n)$

4.

Ans: $M - N$

5.

由於此題未說明 `insert(value)` 的實做內容，因此假設題目的 insert 函數同課堂講義有實做 heapify 來維持 heap 特性。 MERGE_HEAP(heap a, heap b) 實做將 heap b 的node 逐個使用 insert function 加入 heap a，由於假設 insert 有實做 heapify，因此 heap b 所有 node 加入 heap a 後，heap a 仍是 heap。對於 binary tree 中的任何一個 node， insert 進另一個 heap 的次數不會超過數的高度 $log(N)$，insert 的time complexcity 為 $log(N)$，最糟的狀況下，所有 binary tree 中所有 node 都曾insert 進另一個 heap，time complexcity 為 $O(Nlog^2N)$

pseudo code:

```
Traverse(node n, heap a):
    if n is not  NIL:
        a.insert(n.val)
        Traverse(n->left, a)
        Traverse(n->right, a)

MERGE_HEAP(heap a, heap b):
   Traverse(b, a)
   return a
```

6.

Problem 2 - Red-Black Tree

1.

每個 left node 可以與其 parent 做 right rotate，每個 right node 可以與其 parent 做 left rotate，只有 root 沒有 parent 可以做 rotation。一個有 n nodes 的 complete binary tree 有 $\lfloor \frac{(n-1)}{2} \rfloor$ 個 right node，$\lceil \frac{(n-1)}{2} \rceil$ 個 left node。

left rotation = $\lfloor \frac{(n-1)}{2} \rfloor$
right rotation = $\lceil \frac{(n-1)}{2} \rceil$

2.

human alogorithm:
從 binary tree 的 root 開始，若目前所在 node 沒有 left node，便將所在 node 指到right node; 若目前所在 node 有 left node，就做 right rotation， 原本的 left child node 在 right rotate 後便會成為該 node 的 parent node，將目前所在的 node 指到該 node 的 parent node，重複上述步驟直到目前所在 node 沒有 left node。當目前所在 node 為 leaf 時終止。 由於在一次 right rotation 後 right-going chain 就會只少增加一個 node，因此最多只會呼叫 n-1 次 right rotate，因此 time complexcity 為 $O(n)$

pseudo code:

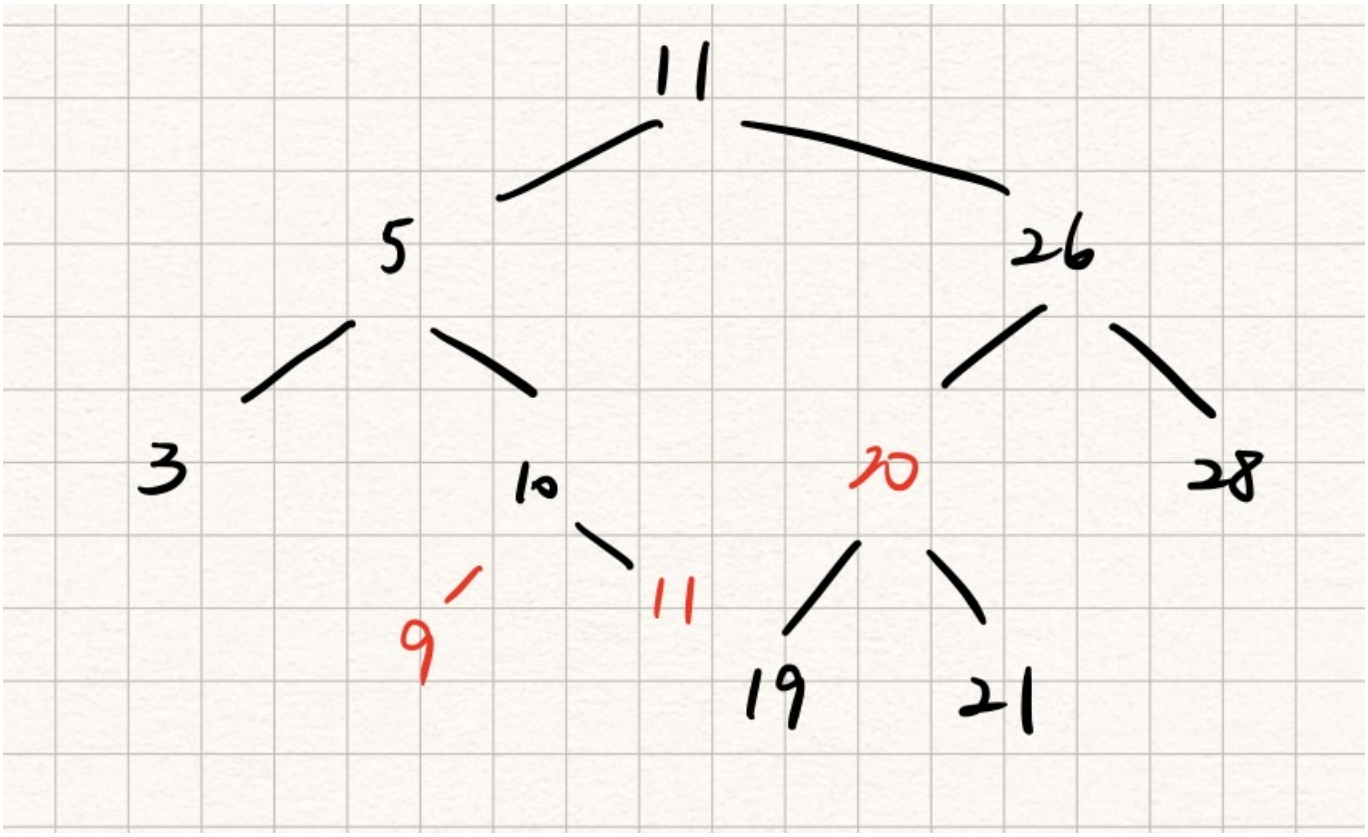
```
Turn-Right-Going-China(Tree T):
    current = T.root
    while (current is not leaf):
        while (current->left is not NIL):
            RIGHT-ROTATE(current)
            current = current->p
```

```
            current = current->right
        return T
```

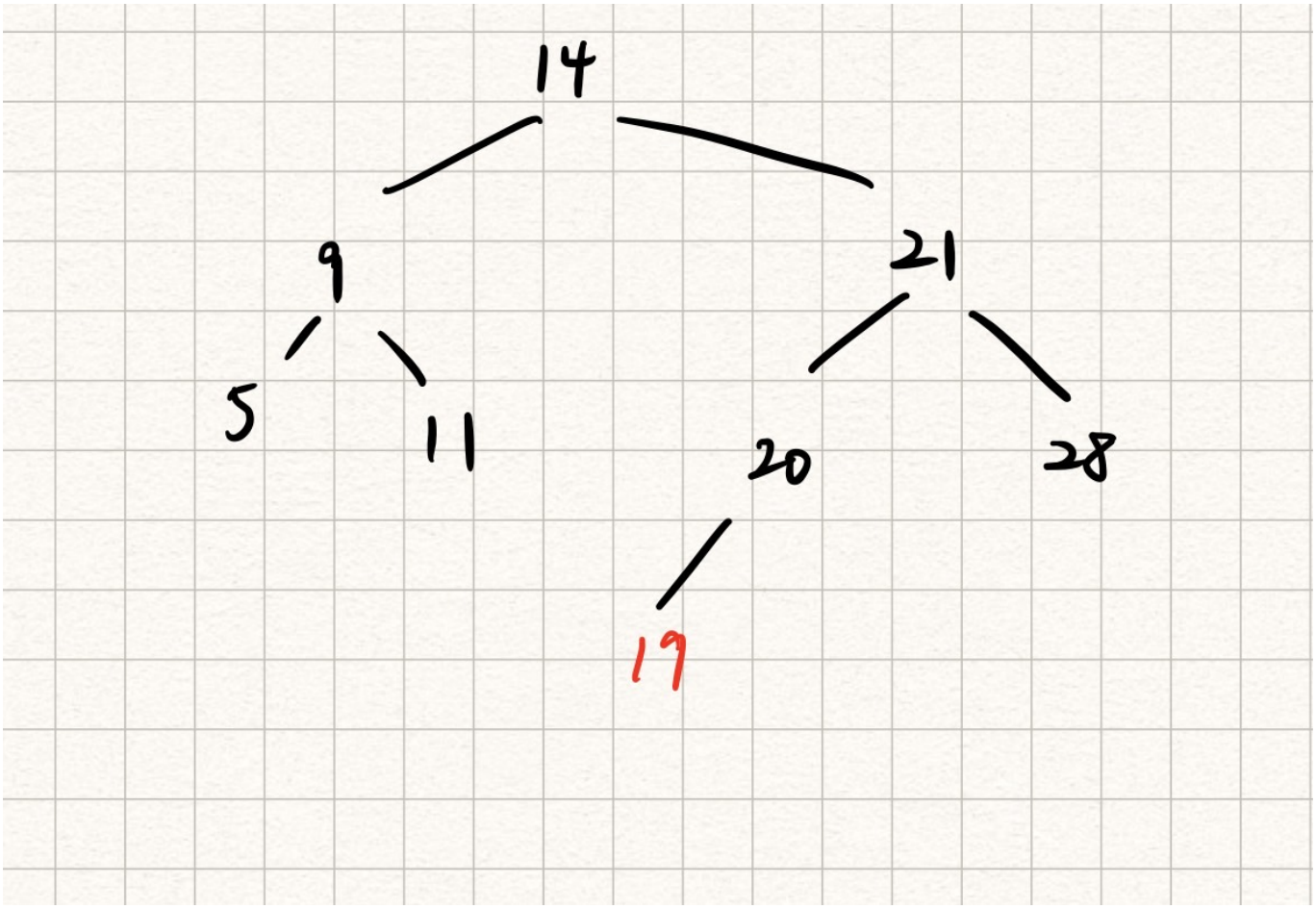
3.

The state is not true. 當一棵 red black tree 只有 black node 時，無論對任何一點做 RIGHT_ROTAE(T,y) or LEFT_ROTATE(T,y) 都不會影響紅黑樹的性質。

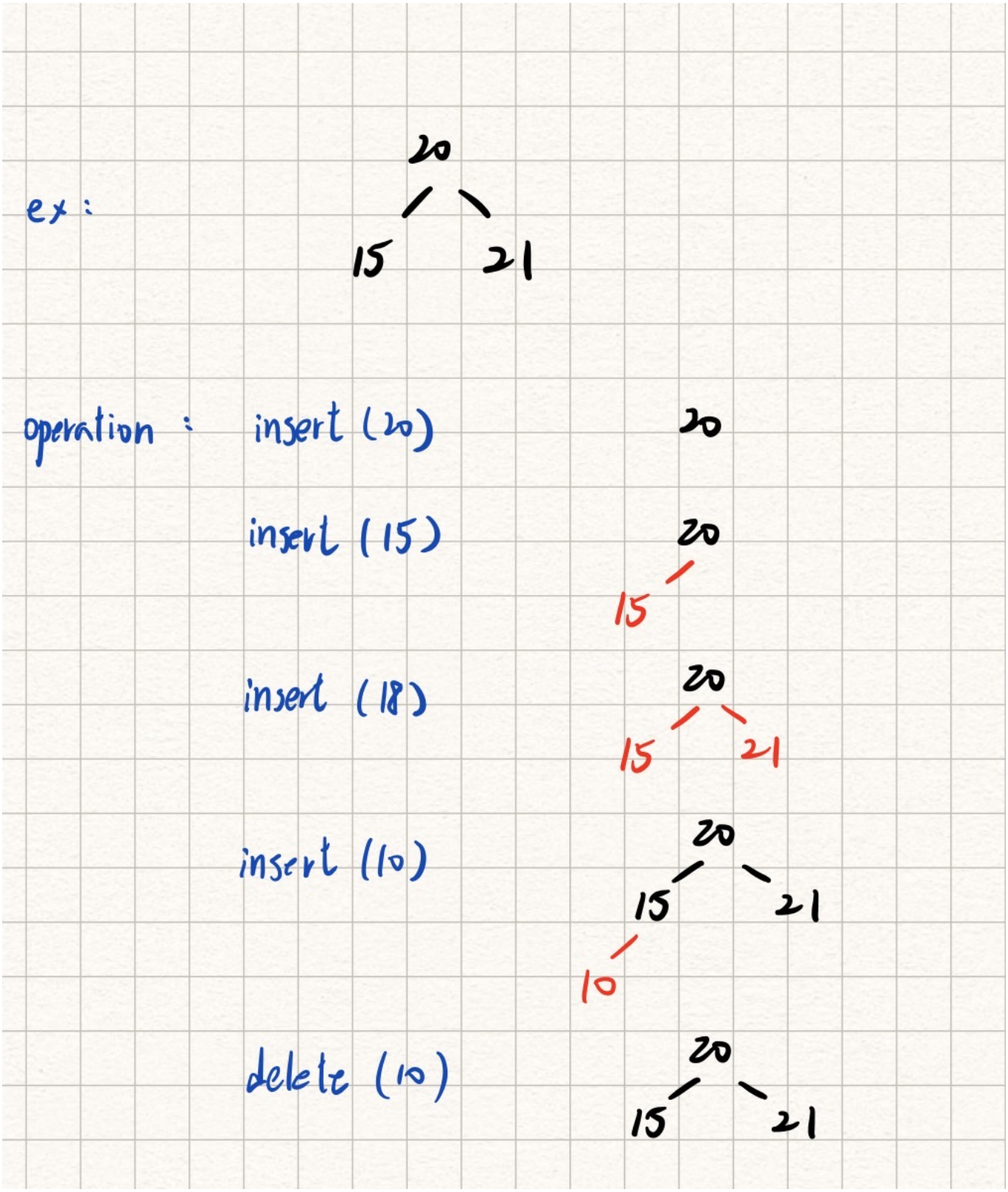
4.



5.



6.



因為除了 root node 之外的 insertion 都無法直接插入黑色的 node，否則會破壞到每一個 leaf 經過的 black node 要一樣多的規則，若要使 RB tree 只有 black node，一定要刪除 red node，因此一定的用到 *DELETE* opertation。