Homework 5

CS 677 - Analysis of Algorithm

Fall Semester - 2019

Dzung Bui

Due date: October 03, 2019

1. Exercise 6.3-2 - *20 points*

Why do we want the loop index i in line 2 of BUILD_MAX_HEAP to decrease form *[A.length/2]* to 1 rather than increase form 1 to *[A.length/2]*

Answer

The algorithm running from *[A.length/2]* to 1 because we would like to build Max Heap from the lowest level in the tree - the leaves. We start sorting the array by comparing the leaves with their parents, and then go up to higher level. As doing that, we can ensure that all procedures just are done in one loop.

2. Exercise 8.2-3 - *20 points*

Suppose that we were to rewrite the for loop header in line 10 of the COUNTING - SORT as

10 *for j = 1 to A.length*

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

Figure 1: Code of Counting Sort

Show that the algorithm still works properly. Is the modified algorithm stable?

Answer

Assuming the input array A is:

i1	i2	i3	i4	i5	i6	i7	i8
2	5	3	0	2	3	0	3

After building array C, which should be like the following:

v0	v1	v2	v3	v4	v5
2	2	4	7	7	8

As we change the line 10 by the new code: *for j = 1 to A.length*, the algorithm still work as showing here:

j = 1: Array B:

i1	i2	i3	i4	i5	i6	i7	i8
-	-	-	2	-	-	-	-

update on C:

v0	v1	v2	v3	v4	v5
2	2	3	7	7	8

j = 2: Array B:

i1	i2	i3	i4	i5	i6	i7	i8
-	-	-	2	-	-	-	5

update on C:

v0	v1	v2	v3	v4	v5
2	2	3	7	7	7

j = 3: Array B:

i1	i2	i3	i4	i5	i6	i7	i8
-	-	-	2	-	-	3	5

update on C:

v0	v1	v2	v3	v4	v5
2	2	3	6	7	7

j = 4: Array B:

i1	i2	i3	i4	i5	i6	i7	i8
-	0	-	2	-	-	3	5

update on C:

v0	v1	v2	v3	v4	v5
1	2	3	6	7	7

j = 5: Array B:

i1	i2	i3	i4	i5	i6	i7	i8
-	0	2	2	-	-	3	5

update on C:

v0	v1	v2	v3	v4	v5
1	2	2	6	7	7

j = 6: Array B:

i1	i2	i3	i4	i5	i6	i7	i8
-	0	2	2	-	3	3	5

update on C:

v0	v1	v2	v3	v4	v5
1	2	2	5	7	7

j = 7: Array B:

i1	i2	i3	i4	i5	i6	i7	i8
0	0	2	2	-	3	3	5

update on C:

v0	v1	v2	v3	v4	v5
0	2	2	5	7	7

j = 8: Array B:

i1	i2	i3	i4	i5	i6	i7	i8
0	0	2	2	3	3	3	5

update on C:

v0	v1	v2	v3	v4	v5
0	2	2	4	7	7

Thus, the algorithm is still running well. However, it is not stable as the previous algorithm. As sorting index of array B increases, the order of putting the same values into B following from large index to smaller one.

3. Implement in C/C++ - *30 Points*

Calculate the the running time of the algorithm

Answer: The running time is of the algorithm is $\Theta(n/2)$

4.

a. Exercise 13.1-2

In the style of Figure 13.1(a), draw the complete binary search tree of height 3 on the keys {1, 2, ... 15}. Add the NIL leaves and color the nodes in three different ways such that the black-heights of the resulting red-black trees are 2, 3, and 4.

Answer

The complete binary tree with height 3:

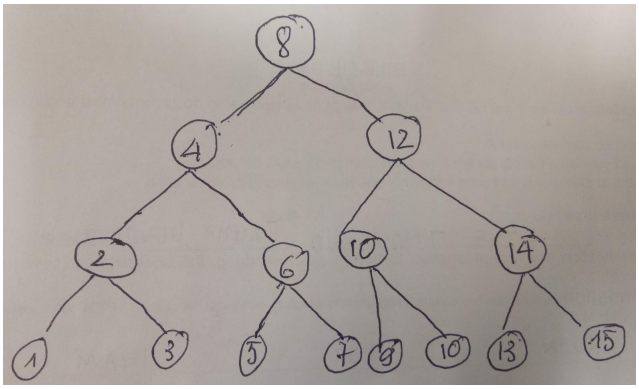

Figure 2: Complete binary tree with height 3 for key {1,2, ..., 15}

The complete binary tree with height 4:

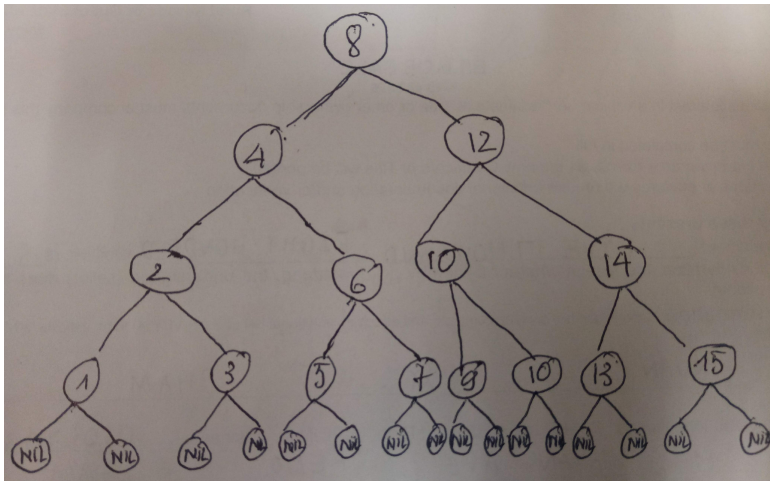

Figure 3: Black Red tree with height 4 for key {1,2, ..., 15}

The complete binary tree with height 2:

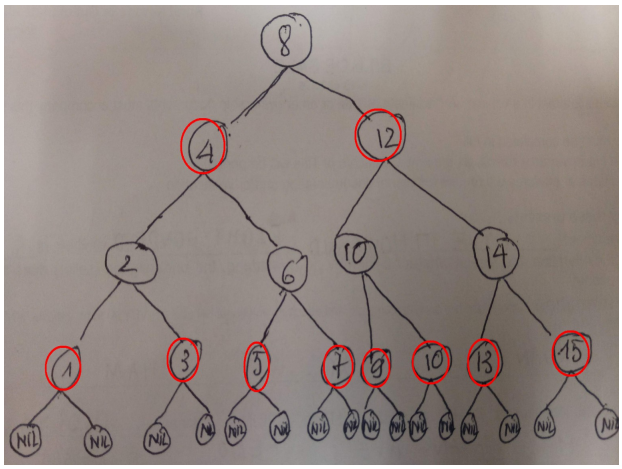

Figure 4: Black Red tree with height 2 for key {1,2, ..., 15}

The complete binary tree with height 3:

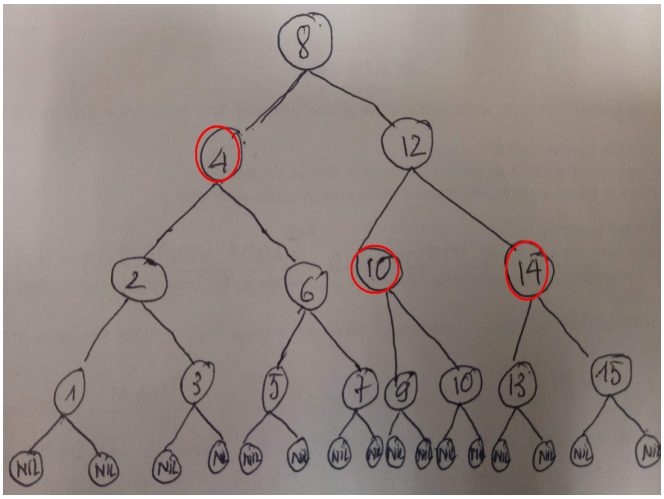

Figure 5: Black Red tree with height 3 for key {1,2, ..., 15}

b. Exercise 13.1-3

Draw the red-black tree that results after TREE -INSERT is called on the tree in Figure 13.1 with key 36. If the inserted node is colored red, is the resulting tree a red-black tree? What if it is colored black?

Answer

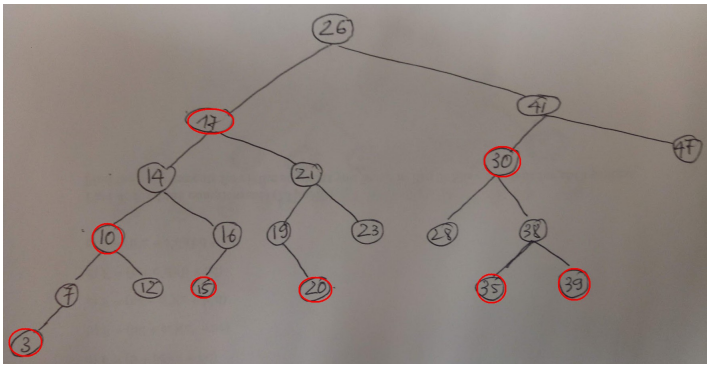

Figure 6: Figure 13.1 from the textbook

After adding item "36", to the tree, it will be added to right brand of node "35". If the adding node is red, the new black-red tree will like the diagram following.

It is not a black-red tree if the adding node "36" is black.

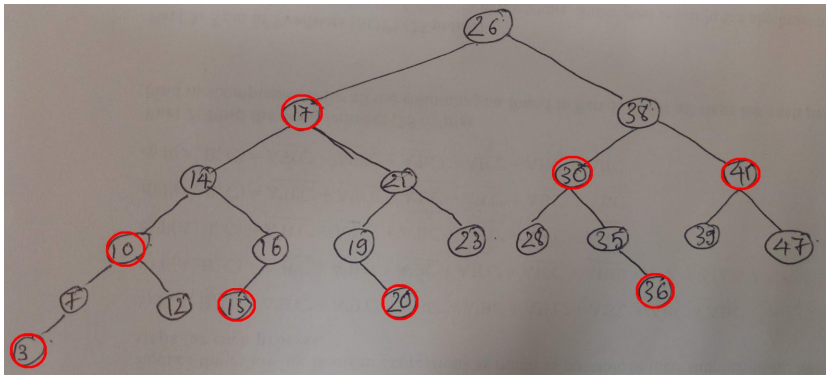

Figure 7: Black Red tree with new node "36"

5. Exercise 8.2-4

Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range [a .. b] in $O(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time

Answer

We can use one part of the counting-sort algorithm to solve this problem:

query_Number(A, a, b, k)

// Pre-processing data ——-

Let *C[0 .. k]* be a new array

for i = 0 to k

C[i] = 0

for j = 0 to *A.length*

C[A[j]] = C[A[j]] + 1

// calculate the number is in range[a .. b] –

number = 0

for k = a to b

number = number + C[k]

return number

6.

a) Find the smallest and the largest number of keys that a heap of height h can have.

For *h=1*, the minimum and maximum should both be 1.

For other case, to get the maximum item, the lowest level should be full. At level [i], the number of nodes are 2^{h-1}. So the total number of nodes are:

$$\sum_{i=1}^{h} 2^{i-1} = 2^h - 1 \tag{1}$$

For the minimum, the last level should contain a single node. It means the tree is full up to a height of *h-1*. So the total number of nodes are:

$$\sum_{i=1}^{h-1} 2^{i-1} + 1 = 2^{h-1} - 1 + 1 = 2^{h-1} \tag{2}$$

b) Prove that the height of a heap with n nodes is $[log_2 n]$

- The number of nodes n of a complete binary tree satisfies:

$$2^h \leq n \leq (2^{h+1} - 1) \tag{3}$$

- Take the log_2 to get h:

$$lg_2(n+1) - 1 \leq h \leq lg_2 n \tag{4}$$

- Since, h is integer, $h = round(lg_2(n+1) - 1) = round(lg_2 n)$