Datastructures & Algorithms: Exercise 4

David Sermoneta

May 14, 2024

Exercise 1.a)

The array `C` after line `4` looks like the following

```
C = [1,3,6].
```

After each step of the `FOR`-loop in line `5`, we get the following arrays.

```
j = 6) B = [-,-,-,-,-,2]
       C = [1,3,5]
------------------------
j = 5) B = [0,-,-,-,-,2]
       C = [0,3,5]
------------------------
j = 4) B = [0,-,-,-,2,2]
       C = [0,3,4]
------------------------
j = 3) B = [0,-,1,-,2,2]
       C = [0,2,4]
------------------------
j = 2) B = [0,1,1,-,2,2]
       C = [0,1,4]
------------------------
j = 1) B = [0,1,1,2,2,2]
       C = [0,1,3].
```

Exercise 1.b)

The array `A = [2,2,2,2]` gives the desired result. Since then at line `4` `C` will be set to

```
C = [0,0,4]
```

since the amount of elements smaller than `0` and `1` is `0` for both. Each time the `FOR`-loop runs then, it will decrement `C[2]` by 1. This will happen `4` times, and so, at the final iteration, we get that $C = [0, 0, 0]$.

Exercise 2a)

Upon first calling `BinarySearch(A, 135, 1, 32)`, a few things are happening: Firstly, in line `1.` we check that `1 > 32`, which isn't true, and so go over to line `2.` initialising `mid = (first + last)//2 = 16` and now we check that `A[16] = 203 == 135` in line `3.`, which is false, and so we go to line `4.`, where we check if `A[mid]` is larger, which it is, and so we call `BinarySearch(A, 135, 1, 15)`, and repeat the process. Here, again line `1.` isn't satisfied, so we set `mid = 8`, check that `A[8] = 119 == 135`, which isn't true, further it is not larger, so now we call `BinarySearch(A, 135, 9, 15)`. We keep ignoring line `1.`, we set `mid = 12`, check that `A[12] = 175 == 135`, not true, so we check if it is larger, this is true, so we get to call `BinarySearch(A, 135, 9, 11)`, where we still ignore line `1.`, so we set `mid = 10`, check if `A[10] = 130 == 135`, this doesn't hold, so we check if it is bigger -it is not- so we call `BinarySearch(A, 135, 11, 11)`. Again, line `1.` is ignored, so we set `mid = 11`, check if `A[11] = 150 == 135`, it isn't, it is however larger, so we call `BinarySearch(A, 135, 11, 10)`, where line `1.` is finally satisfied, and so we get the return value:

```
"A does not contain 135"
```

and we are done.

Exercise 2.b)

The optimal choice for jump width `m` is $\lfloor\sqrt{n}\rfloor$, that is, $\lfloor\sqrt{32}\rfloor$ =`m=5`. Since we want to find the key `135`. At `i = 1` and `m = 5` and jump to `i*m + 1 = 6`, we get that `L[6] = 108 < 135`, so we increment `i` by one, and now check `L[i*m + 1] = L[2*5 + 1] = L[11] = 150`. Since `150 > 135`, we know that `135` must be between index `6` and `10`, if it's in the list at all. So we apply linear search on the sublist `L[6:10]` and get that `L[7] = 111 < 135`, so we increment by one and get `L[8] = 119 < 135`, again, increment, and we get the same for all indices until at last `L[10] = 130 < 135`, which means that `135` is nowhere in the ordered list since it is smaller than `150` and bigger than `130`, which both are in `L[10]` and `L[11]` respectively.

Exercise 3:

```
1  INORDER-TREE-WALK(x):
2      stack = []
3      WHILE True:
4          IF x != NIL:
5              stack.append(x)
6
7              x = x.left
8
9          ELIF: stack != []
10             x = stack.pop()
11             print(x.key())
12             x = x.right
13         ELSE:
14         break
```

The way this algorithm works, is that at first, it will just insert elements from top to bottom left into the stack, until we hit the left most leaf l_1, which will mean it is the first element we want to print. Since the `WHILE` loop wil run once more when this occur, now with `x = NIL`, and the left most leaf being at the top of the stack, this will lead us to set `x =` l_1, print `x`, all in accordance to the inorder tree traversal. After that it will set `x` equal to l_1's right child, and if it exists, everything will go as per usual: We will find the left most leaf of the subtree rooted at l_1`.right`, and that will correspond to the second element being printed via the same argument just given.

If l_1`.right = NIL`, then the `ELIF` statement will just run again making the next `x` equal to the parent of l_1. That element will be printed, following an inorder traversal once again, and then setting the next `x` equal to its right child, where the argument for getting at the outermost leaf of the tree rooted at that node will be satisfied, exhausting all the cases we can think of.

Exercise 4a)

I interpret "remove key 2" as meaning to remove the elements 3,6 and 3 respectively for each list, and not just the element 2. (The course has been a bit unclear about when what notation means what).

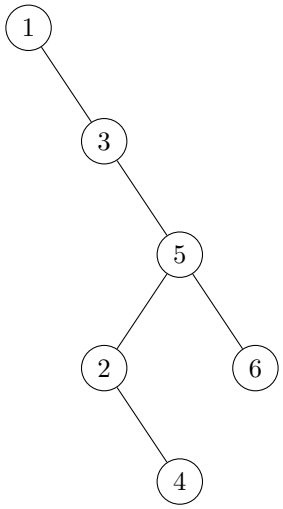


Figure 1: Binary search tree built from list a).

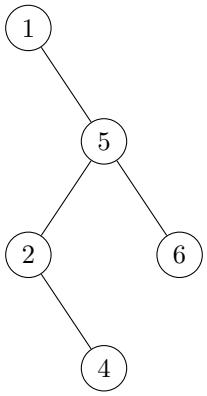


Figure 2: Binary search tree built from list a) after removal of key 2 (element 3).

Exercise 4b)

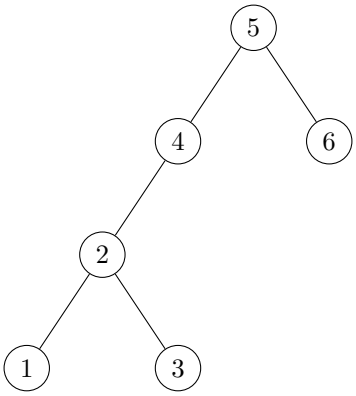


Figure 3: Binary search tree built from list b).

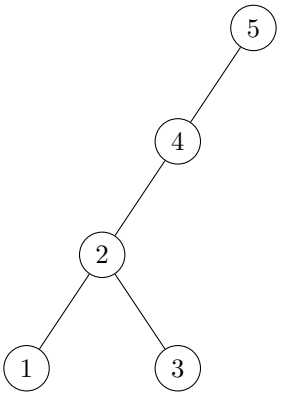


Figure 4: Binary search tree built from list b) after removal of key 2 (element 6).

Exercise 4c)

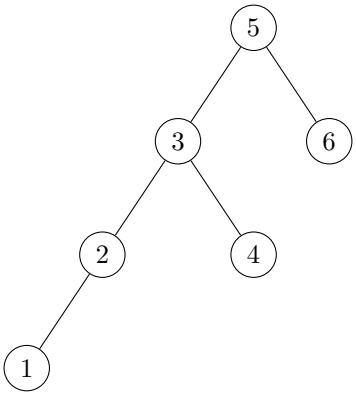


Figure 5: Binary search tree built from list c).

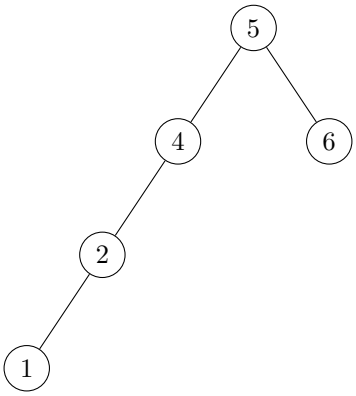
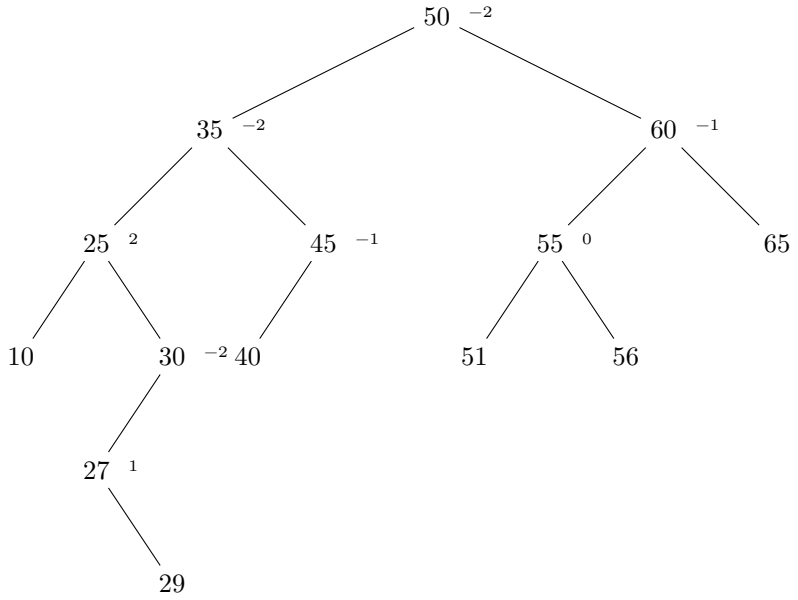
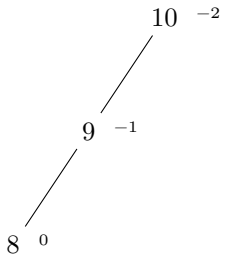


Figure 6: Binary search tree built from list c) after removal of key 2 (element 3).

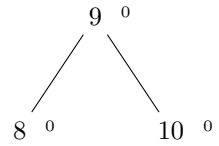
Exercise 5:



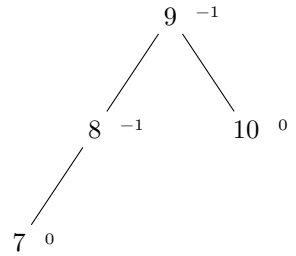
Exercise 6.a)



(a) Tree before having to do `RightRot(10)`

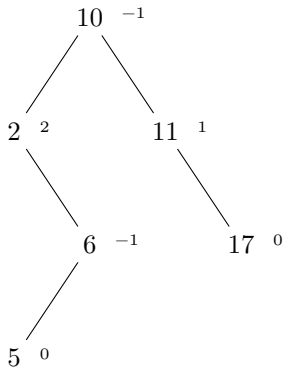


(b) Tree after right rotation at vertex `9`, this is sufficient to lower the BF(10) to 0.

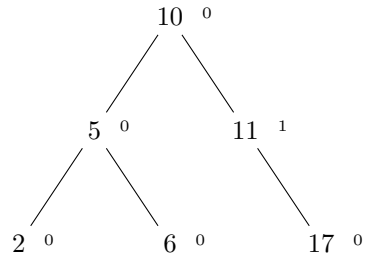


(c) Final Tree!

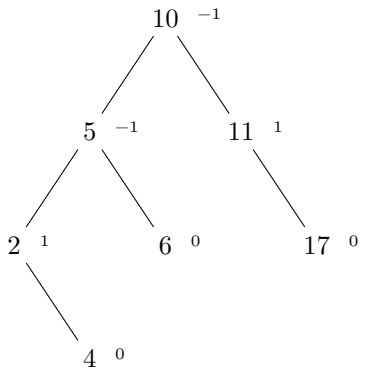
Exercise 6.b)



(a) Tree right after adding vertex 5, giving us an unbalanced tree, with a righ-left case. And so we will need two rotations, one right at 6 and one left afterwards at 2.



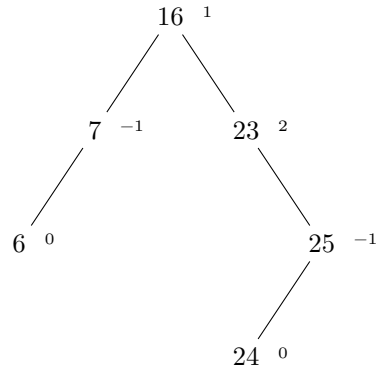
(b) This is the tree after the necessary right and left rotations at the previously specified vertices.



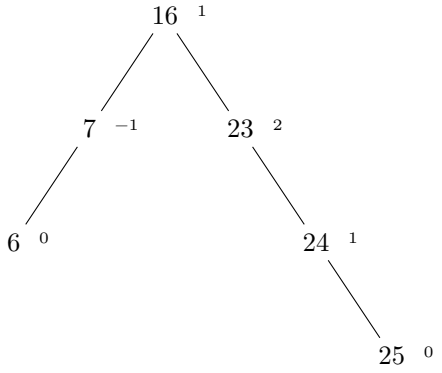
(c) Final tree!

Exercise 7:

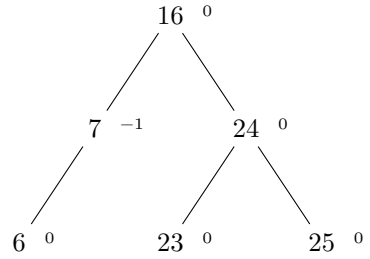
Removing the key `15` from `T` involves first searching for the smallest key in the right subtree rooted at `15.right`, replace `15` with that key, and then delete `15` at the new position. This only involves looking for the leaf `16`, as that satisfies the conditions, setting it as the new root of the tree, and this gives us the following tree.



(a) Tree after deleting key `15`. Now we need to balance the subtree rooted at `23`, which we do by applying a double rotation, first a right one at `25`, and then a left one at `23`.

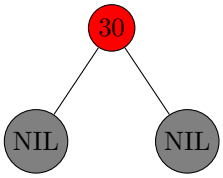


(b) Tree after right rotation at `25`.

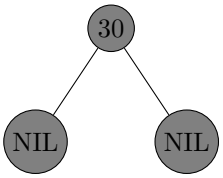


(c) Tree after left rotation at `23`.

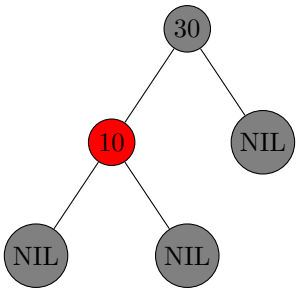
Exercise 8:



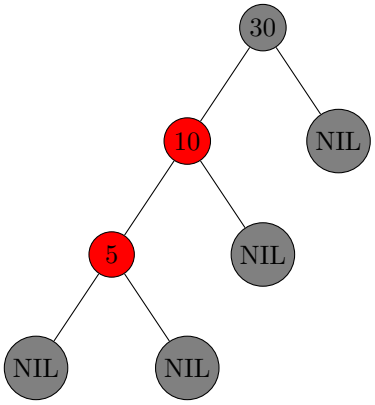
(a) Tree after insertion of `30`.



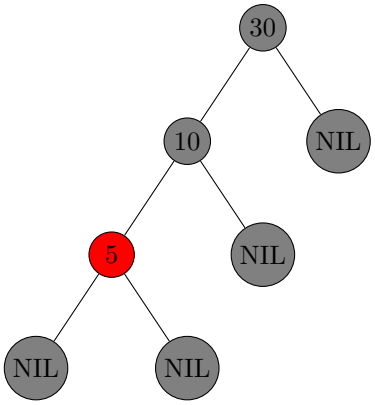
(b) after recoloring to preserve RB property 1.b



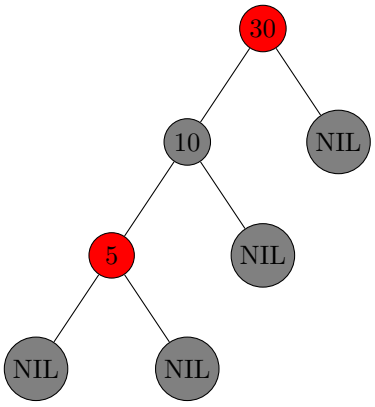
(c) RB property satisfied when inserting `10`



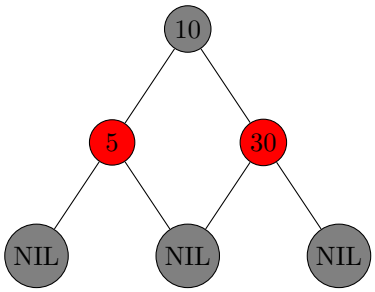
(a) After insertion of `5`.



(b) Recoloring according to line 12, we now have that RB property is not satisfied, as all paths from any node do not contain equal black nodes.



(a) We have changed color of `30` via line 13.



(b) We have done a right rotation on `30`, getting our final tree, satisfying the RB property!

Exercise 9:

index	0	1	2	3	4	5	6
key			**9**				
i			0				

index	0	1	2	3	4	5	6
key			9			**12**	
i			0			0	

index	0	1	2	3	4	5	6
key	**5**		9			12	
i	1		0			0	

index	0	1	2	3	4	5	6
key	5		9	**3**		12	
i	1		0	0		0	

index	0	1	2	3	4	5	6
key	5		9	3	**7**	12	
i	1		0	0	2	0	

Exercise 10:

Proof. A little bird whispers in my ear and tells me to read the first code as hexadecimal ascii representations of characters. That is, the integer `54` should be interpreted as the character **T**, as that is the hexadecimal ascii representation of the character **T**. Following the bird's advice, we get the string

`T h e SP a n s w e r SP is : SP 4 2`,

which in natural language becomes

The answer is: 42.

Just like in exercise 1, we want to translate the number 240743138 to its representation in base 128, where then each coefficient will translate to an ascii character in decimal form. That is, we want to find the coefficients a_k to the equation

$$240743138 = \sum_{k=0}^{N} a_k \cdot 128^k.$$

By using the euclidean algorithm, we get that each iteration r_i will give us the coefficient a_i. For example, since

$$240743138 = 98,$$

we know that $\mathfrak{a}_0 = 98$. Then we can take the difference of these two, and divide it by 128, to get the integer

$$\frac{1240743138 - 98}{128} = 880805.$$

Repeating this process until it terminates (that the euclidean algorithm terminates is omitted in this assignent) yields the coefficients

$$\begin{aligned}\mathfrak{a}_0 &= 98\\ \mathfrak{a}_1 &= 101\\ \mathfrak{a}_2 &= 101\\ \mathfrak{a}_3 &= 114,\end{aligned}$$

after which the program terminates. Translating these into their decimal ascii counterparts gives us the characters

$\mathfrak{a}_0$	$\mathfrak{a}_1$	$\mathfrak{a}_2$	$\mathfrak{a}_3$
98	101	101	114
$\mathfrak{b}$	$\mathfrak{e}$	$\mathfrak{e}$	$\mathfrak{r}$

And so the holy recipe of liquid bread is **beer**. □