Segment Trees

Diego Perez-Torres, Max Mueller, Hayden St. Germain, Alexander Lang

What is a Segment Tree?

A data structure used to answer range queries over an array effectively, while also being flexible enough to modify it if need be.

Queries using the entire segment tree require O(logn) time.

Some simple uses are finding the sum, minimum and maximum of the array of values.

The underlying data structure of a segment tree is a binary tree.

History of the Segmentation Tree

The Segmentation tree was first invented in 1977 by John Bentley to be able solve the Klee's Rectangle Problem more efficiently.

What is the Klee's Rectangle Problem?

Klee's Rectangle Problem is a way to determine how efficient the measure of a union of rectangular ranges can be computed

Also be thought as:

Given a set of n axis-parallel boxes in *d*-dimensional space, compute the volume of the union of the boxes

Real World Application

A real world application of the segment tree is the Wayback Machine

> The Wayback Machine is a way for a user to look back at what a website used to look like

The websites and information is stored in segment trees which allows a user to go back to a point in a websites history

Other Applications

Using Segment Trees

- Solve Min/Max & Sum Queries and Range Update Queries
- Static and Dynamic RMQ (Range Minimum Query)
- Storing segments in an arbitrary manner

Tasks done with Segment Trees

- List all pairs of intersecting rectangles from a list in a plane
- Used in pattern recognition and image processing
- Geographic information systems
- Computational geometry

But why use a Segment tree?

Sum?

Minimum?

Maximum?

Simple so far, no?

Sum: 10

Minimum: 1

Maximum: 4

What about this range of values?

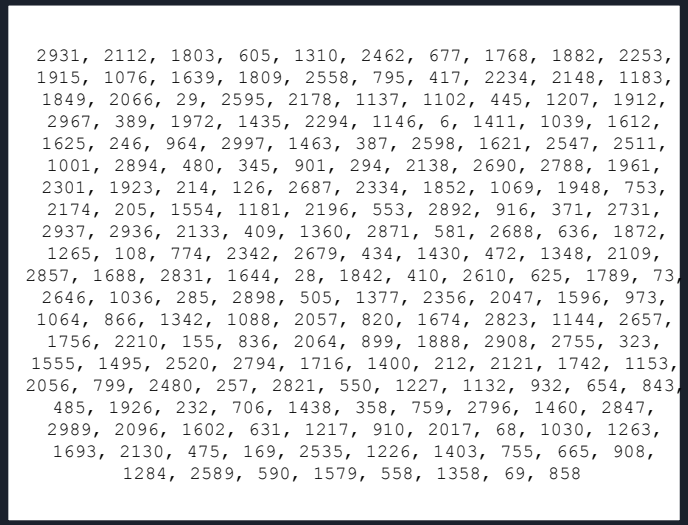
2931, 2112, 1803, 605, 1310, 2462, 677, 1768, 1882, 2253,
1915, 1076, 1639, 1809, 2558, 795, 417, 2234, 2148, 1183,
1849, 2066, 29, 2595, 2178, 1137, 1102, 445, 1207, 1912,
2967, 389, 1972, 1435, 2294, 1146, 6, 1411, 1039, 1612,
1625, 246, 964, 2997, 1463, 387, 2598, 1621, 2547, 2511,
1001, 2894, 480, 345, 901, 294, 2138, 2690, 2788, 1961,
2301, 1923, 214, 126, 2687, 2334, 1852, 1069, 1948, 753,
2174, 205, 1554, 1181, 2196, 553, 2892, 916, 371, 2731,
2937, 2936, 2133, 409, 1360, 2871, 581, 2688, 636, 1872,
1265, 108, 774, 2342, 2679, 434, 1430, 472, 1348, 2109,
2857, 1688, 2831, 1644, 28, 1842, 410, 2610, 625, 1789, 73,
2646, 1036, 285, 2898, 505, 1377, 2356, 2047, 1596, 973,
1064, 866, 1342, 1088, 2057, 820, 1674, 2823, 1144, 2657,
1756, 2210, 155, 836, 2064, 899, 1888, 2908, 2755, 323,
1555, 1495, 2520, 2794, 1716, 1400, 212, 2121, 1742, 1153,
2056, 799, 2480, 257, 2821, 550, 1227, 1132, 932, 654, 843,
485, 1926, 232, 706, 1438, 358, 759, 2796, 1460, 2847,
2989, 2096, 1602, 631, 1217, 910, 2017, 68, 1030, 1263,
1693, 2130, 475, 169, 2535, 1226, 1403, 755, 665, 908,
1284, 2589, 590, 1579, 558, 1358, 69, 858

Sum?

Minimum?

Maximum?

Tedious labor, even for a machine.


2931, 2112, 1803, 605, 1310, 2462, 677, 1768, 1882, 2253,
1915, 1076, 1639, 1809, 2558, 795, 417, 2234, 2148, 1183,
1849, 2066, 29, 2595, 2178, 1137, 1102, 445, 1207, 1912,
2967, 389, 1972, 1435, 2294, 1146, 6, 1411, 1039, 1612,
1625, 246, 964, 2997, 1463, 387, 2598, 1621, 2547, 2511,
1001, 2894, 480, 345, 901, 294, 2138, 2690, 2788, 1961,
2301, 1923, 214, 126, 2687, 2334, 1852, 1069, 1948, 753,
2174, 205, 1554, 1181, 2196, 553, 2892, 916, 371, 2731,
2937, 2936, 2133, 409, 1360, 2871, 581, 2688, 636, 1872,
1265, 108, 774, 2342, 2679, 434, 1430, 472, 1348, 2109,
2857, 1688, 2831, 1644, 28, 1842, 410, 2610, 625, 1789, 73,
2646, 1036, 285, 2898, 505, 1377, 2356, 2047, 1596, 973,
1064, 866, 1342, 1088, 2057, 820, 1674, 2823, 1144, 2657,
1756, 2210, 155, 836, 2064, 899, 1888, 2908, 2755, 323,
1555, 1495, 2520, 2794, 1716, 1400, 212, 2121, 1742, 1153,
2056, 799, 2480, 257, 2821, 550, 1227, 1132, 932, 654, 843,
485, 1926, 232, 706, 1438, 358, 759, 2796, 1460, 2847,
2989, 2096, 1602, 631, 1217, 910, 2017, 68, 1030, 1263,
1693, 2130, 475, 169, 2535, 1226, 1403, 755, 665, 908,
1284, 2589, 590, 1579, 558, 1358, 69, 858

Sum: 297,761

Minimum: 6

Maximum: 2,997

Underlying structure

In this implementation of a Segment Tree, we use a 1D vector to mimic a node-tree.

But how does it work?

Creating a Segment Tree

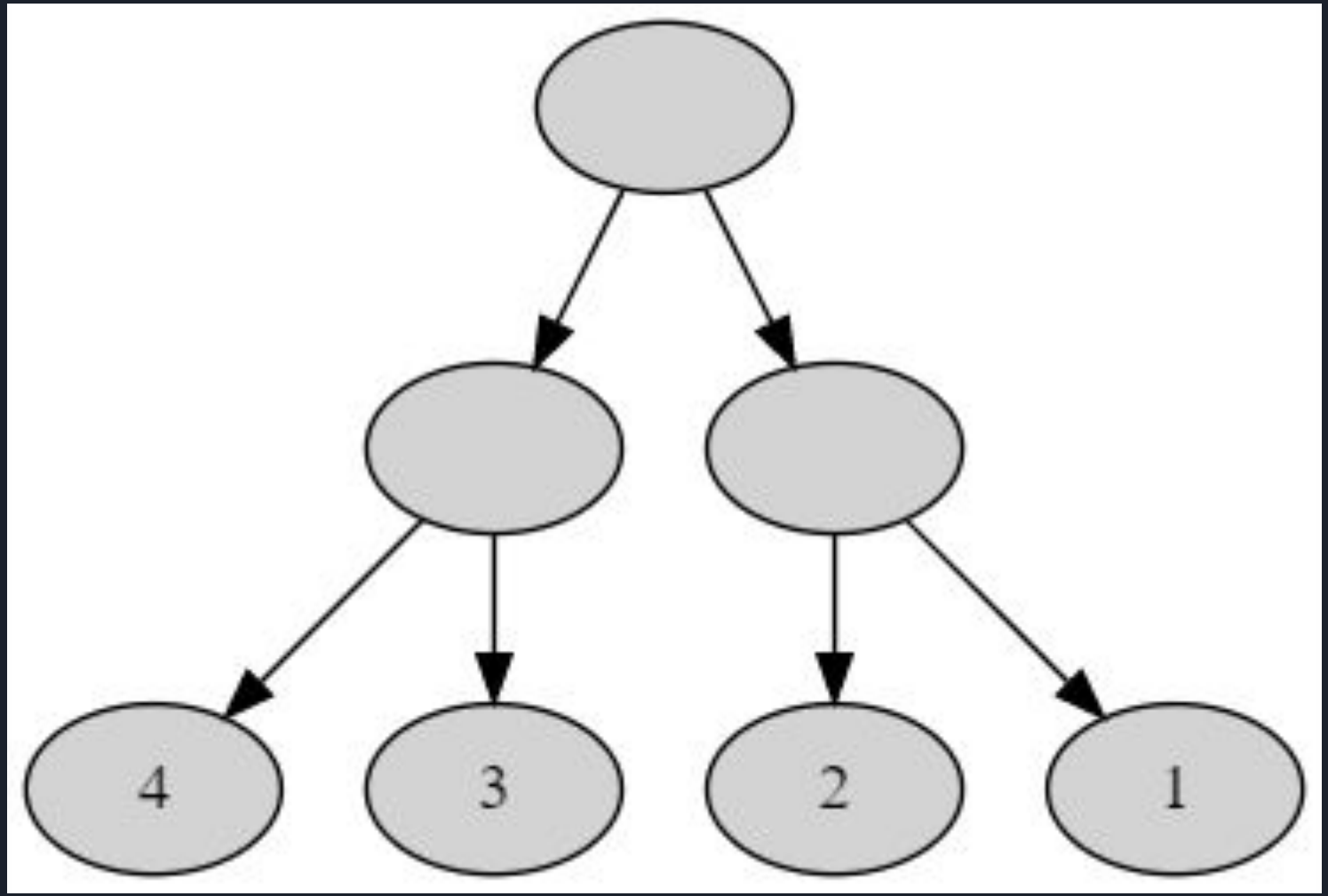
Pseudo Code:

Create an object that takes in the vector of data and a string to determine if you are looking for either the sum, min or max.

Initialize n = size of the vector and create a copy of vector for later modification if need be

Create a second vector the size of 2n-1 filled from the end to front with the values of the first vector

The first n-1 values should always be empty or 0

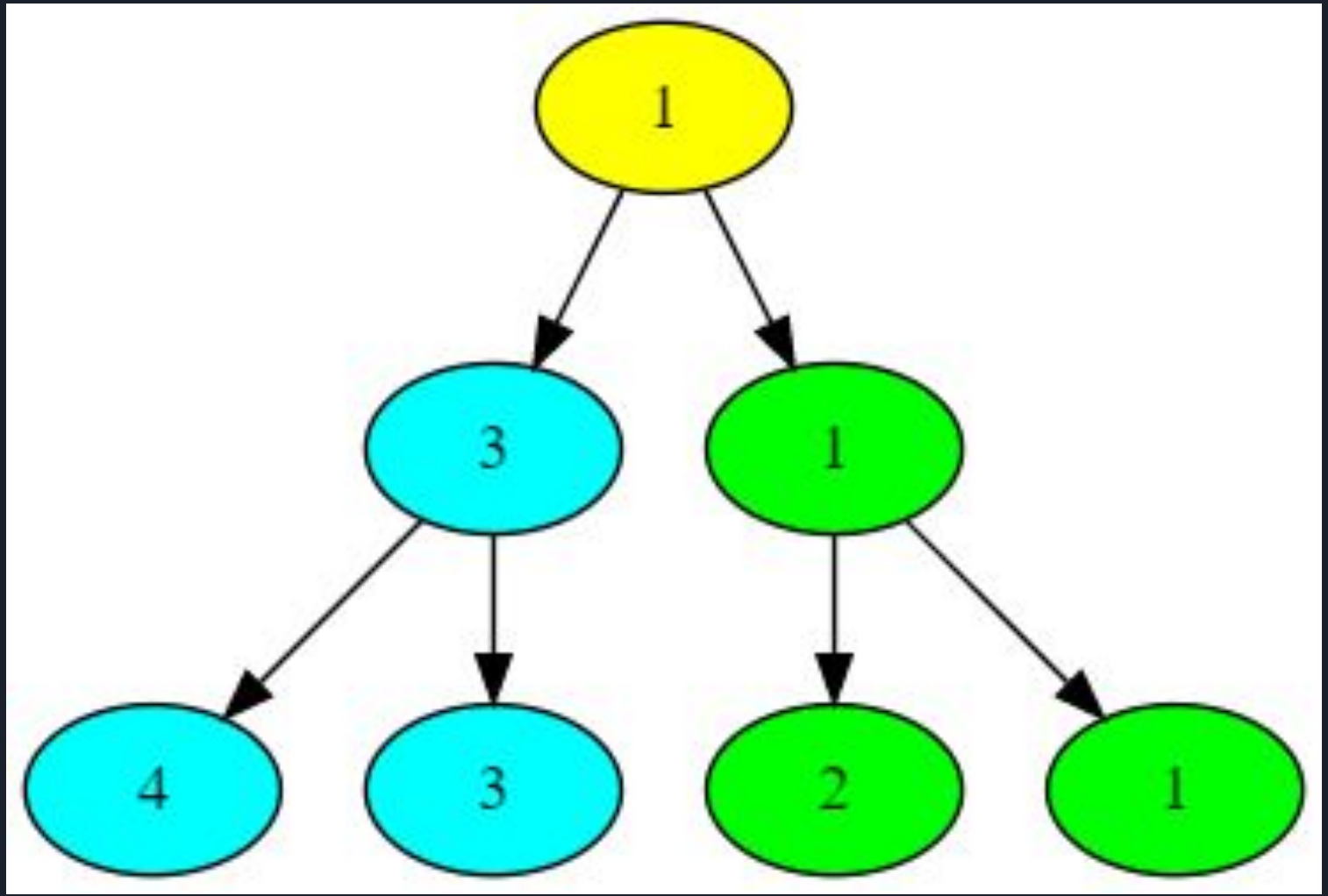

4
3
2
1

Filling in the rest of the Segment Tree

Check what the string is, and then calculate the min, max or sum using a for loop

For loop loops through the second vector, initializing index = n-1 and stopping when index > 0

Calculates the min, max, or sum by comparing/adding the values at index n*2 and n*2 +1


1
3
1
4
3
2
1

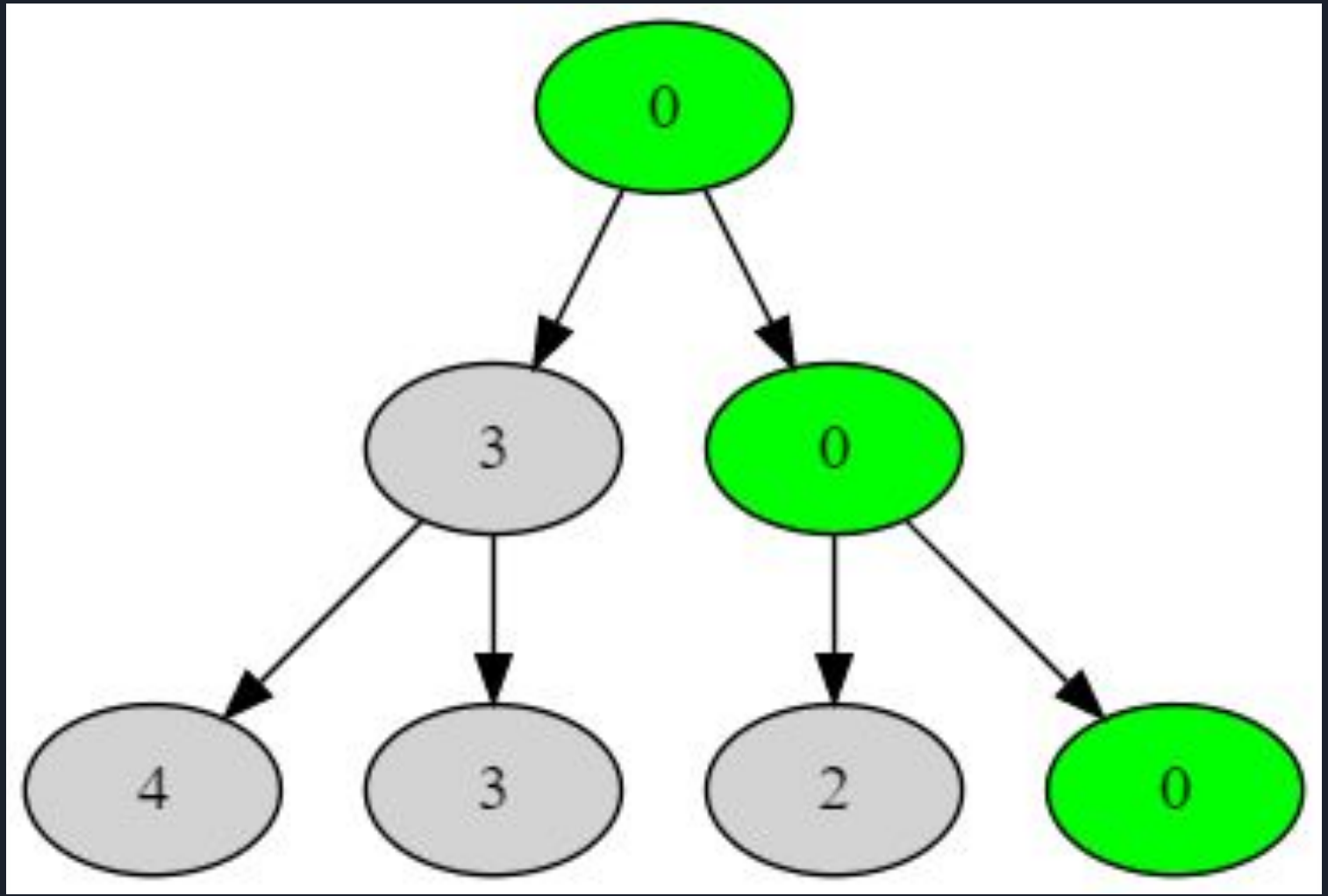
Updating the values in the vector

Pseudo Code:

Create a function that takes in the index of which value you want to update, and the value itself

- Update the value at the index of the original vector
- Follow the same steps as when creating the completed segment tree with the updated value

Let’s say I want to update the value at index 3, replacing the value with 0.


0
3
0
4
3
2
0

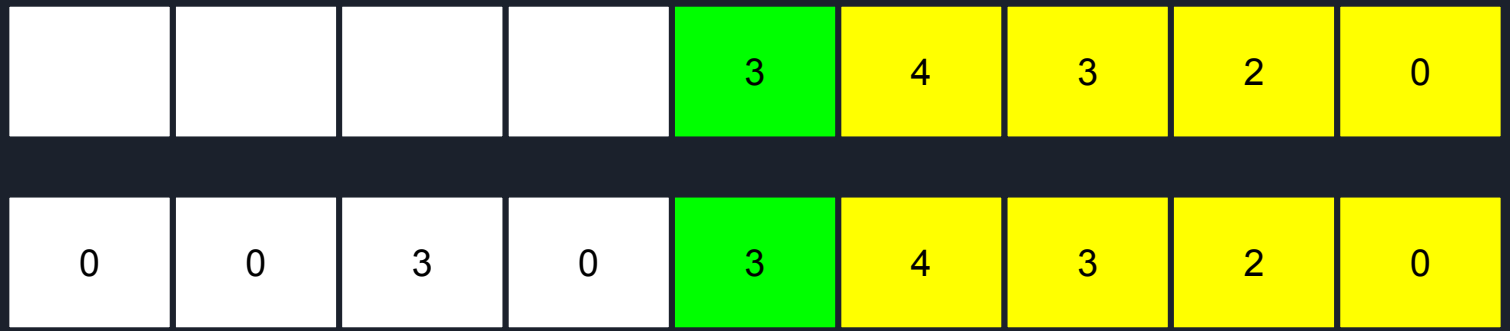
Inserting a value to the vector

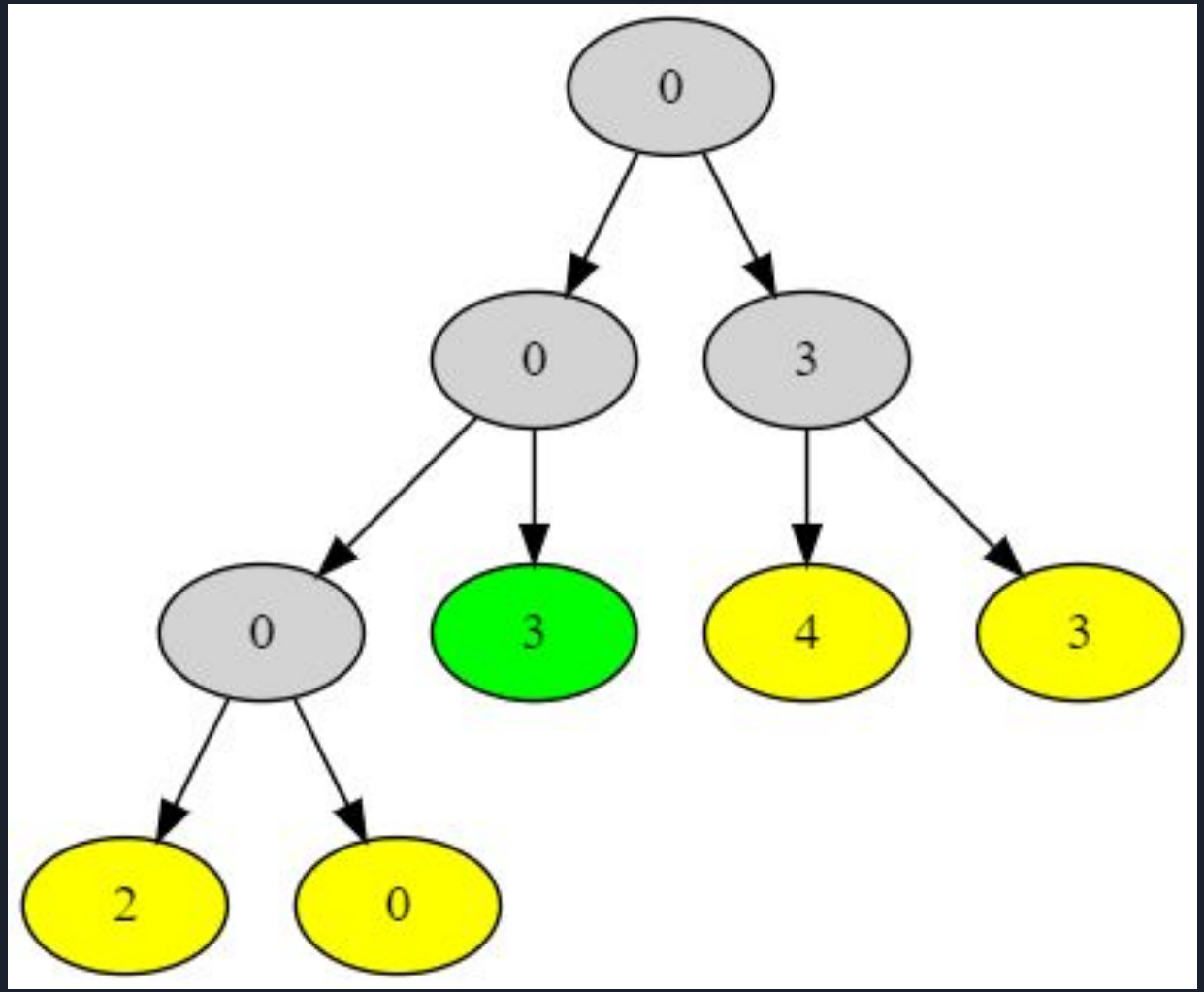
Pseudo Code:

Create a function that takes in the index of which value you want to insert, and the value itself

- Insert the value into the original vector, pushing all the values after it to the metaphorical left
- Update the value of n with the size of the newly modified vector
- Follow the same steps as when creating the completed segment tree with the updated value

Let’s say I want to insert the value 3 at index 0.


3 4 3 2 0
0 0 3 0 3 4 3 2 0


0
0
3
0
3
4
3
2
0

Searching for a value in the Vector

Pseudo Code:

Create a function that takes in the value you are looking for

- Create an empty vector that will store in the indexes in which the value is found
- Use a for loop that checks the entire vector for the value, and then stores into into the new vector
- If the vector is empty, print out an error message
- Else, use another for loop to loop through the new vector, printing out where it was found
    - Print out how many times it was found

Output should look like this:

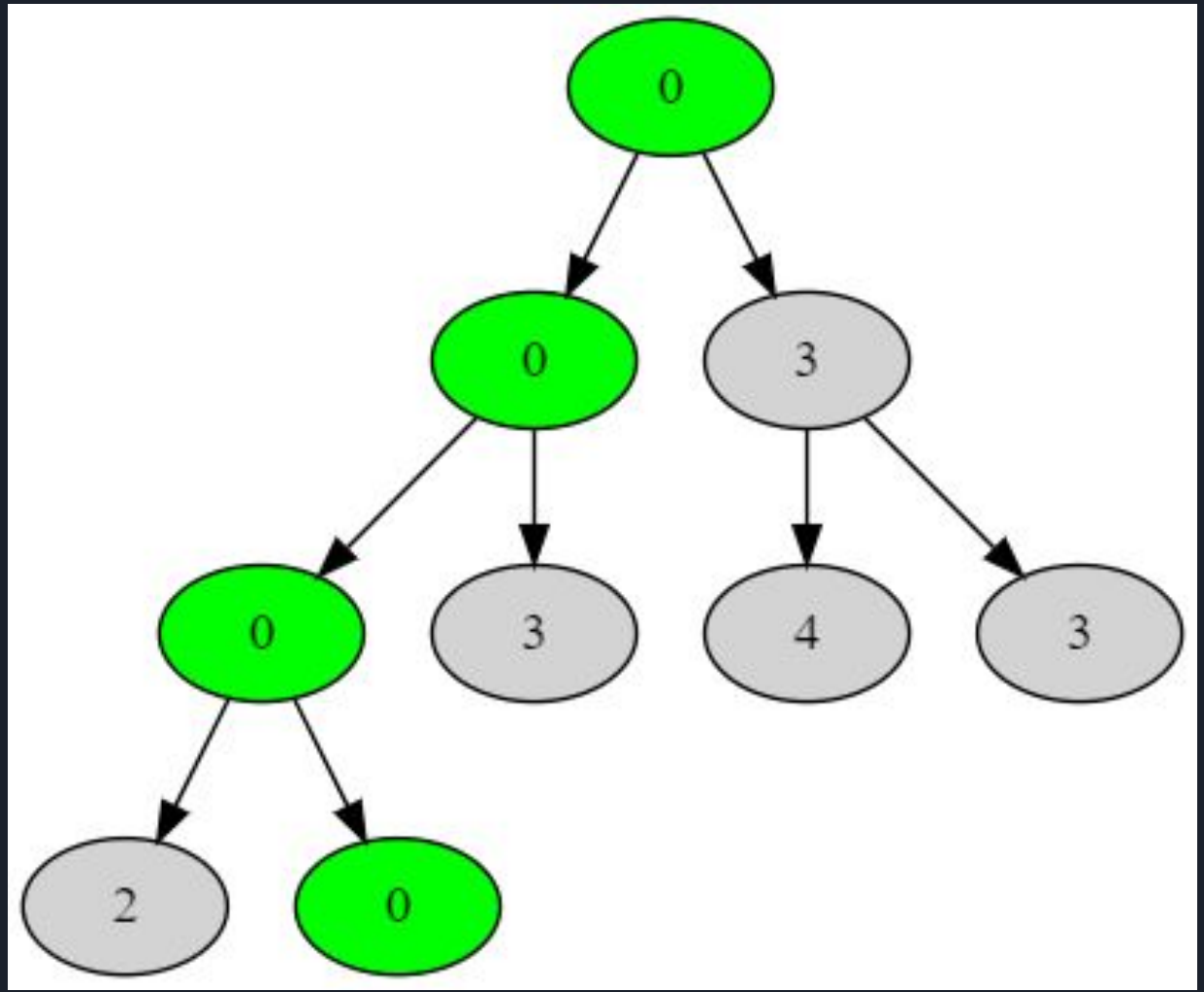
0 was found at: 1

0 was found at: 2

0 was found at: 4

0 was found at: 9

Found 4 time(s).


0
0
3
0
3
4
3
2
0

Viewing the entire Segment Tree

Pseudo Code:

Create a function that takes in a string that can be converted into a txt file, ex: “output.txt”

Initialize the variable “current” to 2

Use ofstream to input the current index with a label of its stored value in a for loop

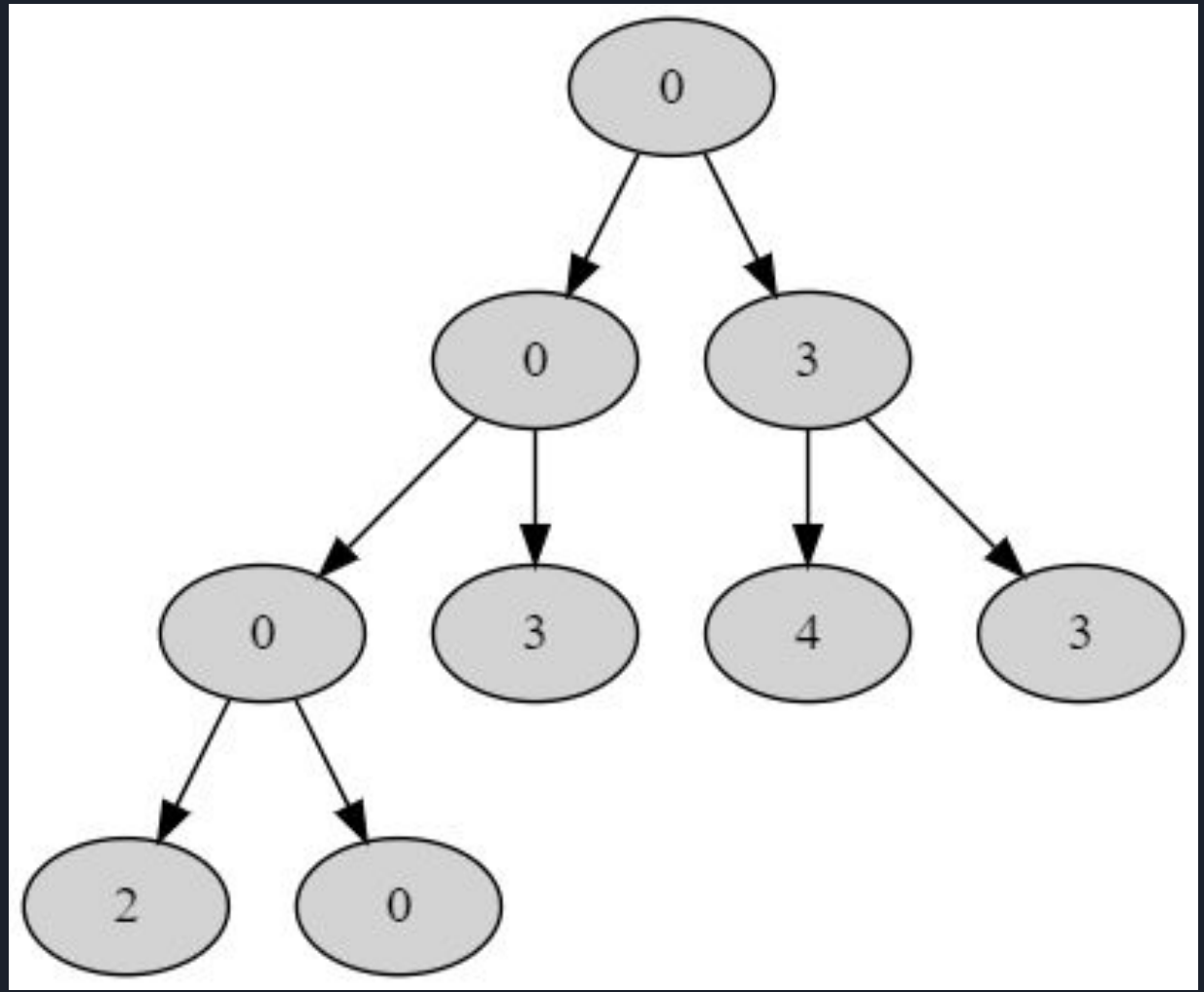
Print out the current index and its value

If current < n:

input current index with two pointers to current and current +1

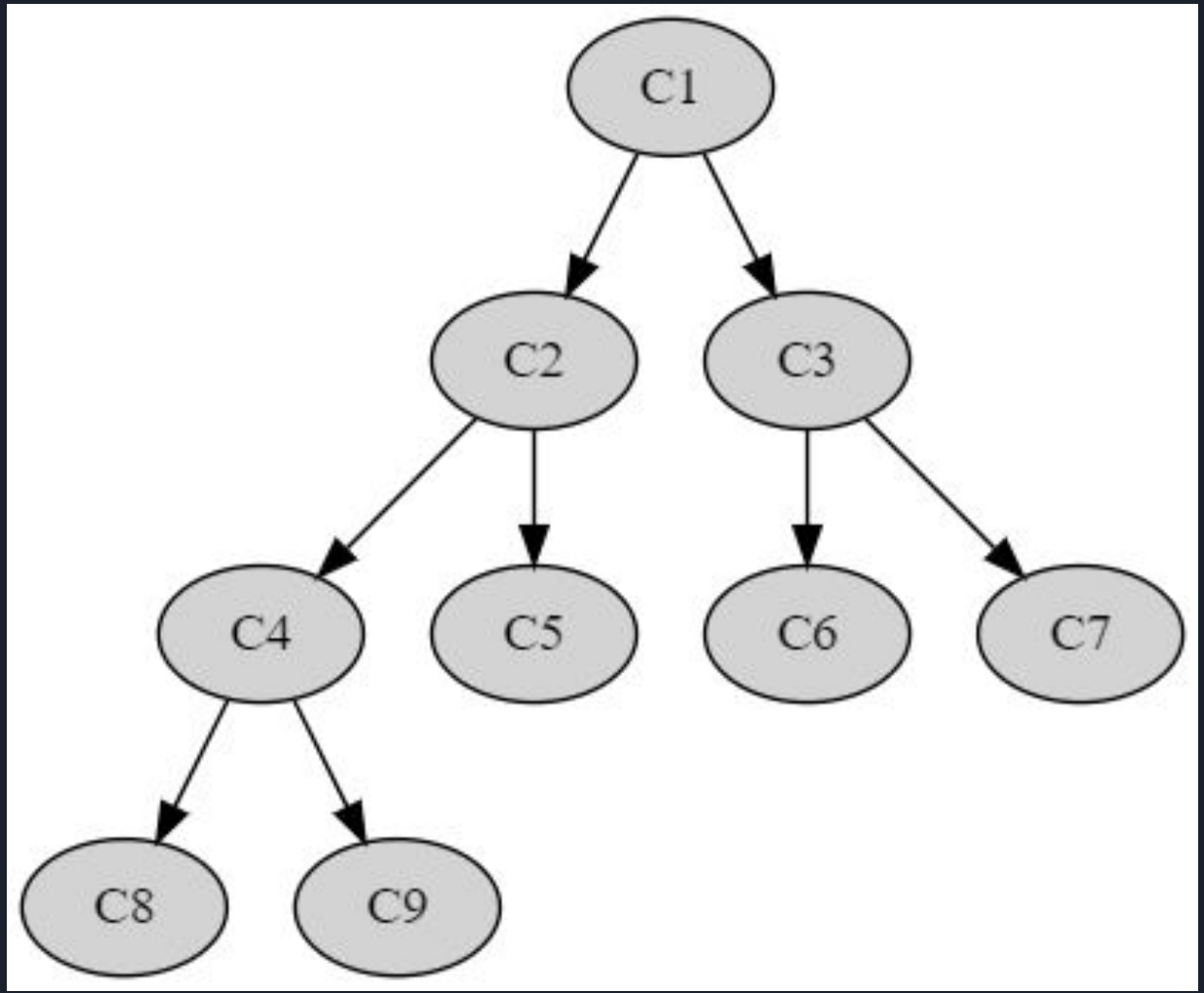
Current += 2

I use https://dreampuf.github.io/GraphvizOnline/ to view my output



C1: 0
C2: 0
C3: 3
C4: 0
C5: 3
C6: 4
C7: 3
C8: 2
C9: 0

C1 [label=0]
C1 -> {C2, C3}
C2 [label=0]
C2 -> {C4, C5}
C3 [label=3]
C3 -> {C6, C7}
C4 [label=0]
C4 -> {C8, C9}
C5 [label=3]
C6 [label=4]
C7 [label=3]
C8 [label=2]
C9 [label=0]



2^0

2^1

2^2

2^n starting at 0 or when index = n-1

Sources:

https://en.wikipedia.org/wiki/Segment_tree

https://cp-algorithms.com/data_structures/segment_tree.html

https://www.youtube.com/watch?v=Oq2E2yGadnU&t=2s

https://www.geeksforgeeks.org/segment-tree-efficient-implementation/?ref=lbp

https://en.wikipedia.org/wiki/Segment_tree#cite_note-Schwarzkopf4-7

https://en.wikipedia.org/wiki/Klee%27s_measure_problem