Section 7: Parallel Prefix_Sum

Casey Xing, Preston Jiang

University of Washington

November 9, 2017

Problem: Scan

Scan

Suppose we have an array `a`:

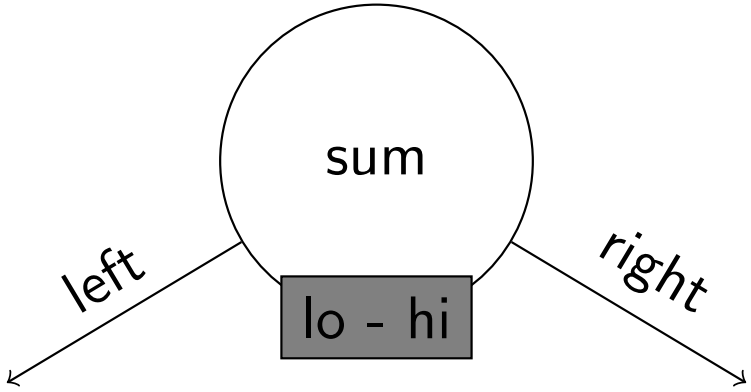
0	1	2	3	4	5
`a[0]`	`a[1]`	`a[2]`	`a[3]`	`a[4]`	`a[5]`

Return an array `b` where `b[i] = a[0] + a[1] + ... + a[i]`

0	1	3	6	10	15
`b[0]`	`b[1]`	`b[2]`	`b[3]`	`b[4]`	`b[5]`

We solve this in parallel by building a tree!

PST: Prefix Sum Tree

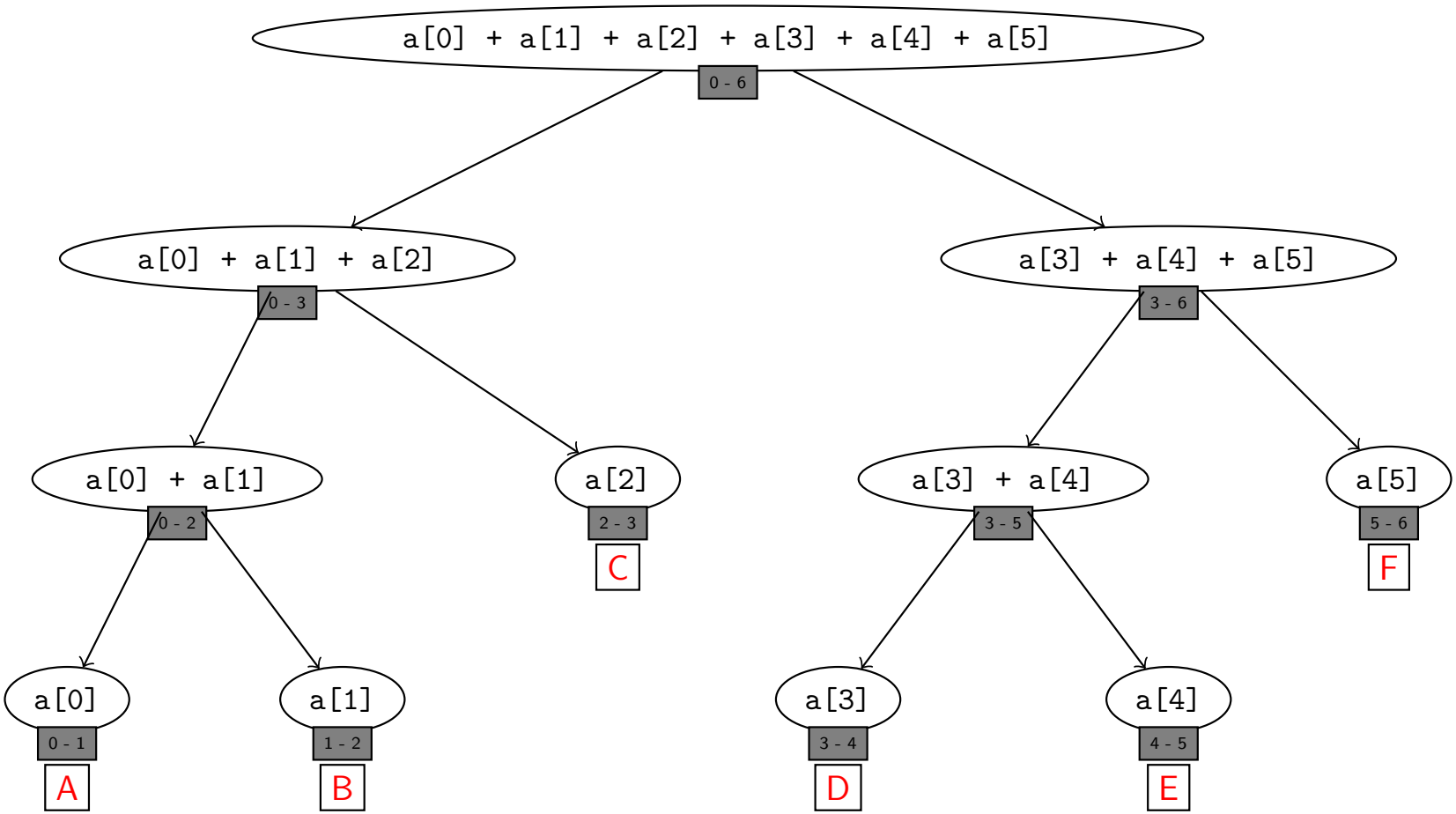


PSTNode

```
PSTNode {
  int lo; // include
  int hi; // exclude
  // sum(arr[lo], arr[lo + 1], ..., arr[hi - 1])
  int sum;
  PSTNode left, right;
}
```

Build the tree (parse input)

```
PSTNode buildTree(int[] arr, int lo, int hi) {
  if (hi - lo == 1) {
    // sum is simply arr[lo]
    return new PSTNode(lo, hi, arr[lo])
  } else {
    mid = lo + (hi - lo) / 2;
    PSTNode left = buildTree(arr, lo, mid);
    PSTNode right = buildTree(arr, mid, hi);
    return new PSTNode(lo, hi, left.sum + right.sum,
        left, right);
  }
}
```


a[0] + a[1] + a[2] + a[3] + a[4] + a[5]
0 - 6
a[0] + a[1] + a[2]
0 - 3
a[3] + a[4] + a[5]
3 - 6
a[0] + a[1]
0 - 2
a[2]
2 - 3
C
a[3] + a[4]
3 - 5
a[5]
5 - 6
F
a[0]
0 - 1
A
a[1]
1 - 2
B
a[3]
3 - 4
D
a[4]
4 - 5
E

Build output

We want one output per leaf!

- A: I'm good!
- B: I need `a[0]`
- C: I need `a[0] + a[1]`
- D: I need `a[0] + a[1] + a[2]`
- E: I need `a[0] + a[1] + a[2] + a[3]`
- F: I need `a[0] + a[1] + a[2] + a[3] + a[4]`

Build output

We want one output per leaf!

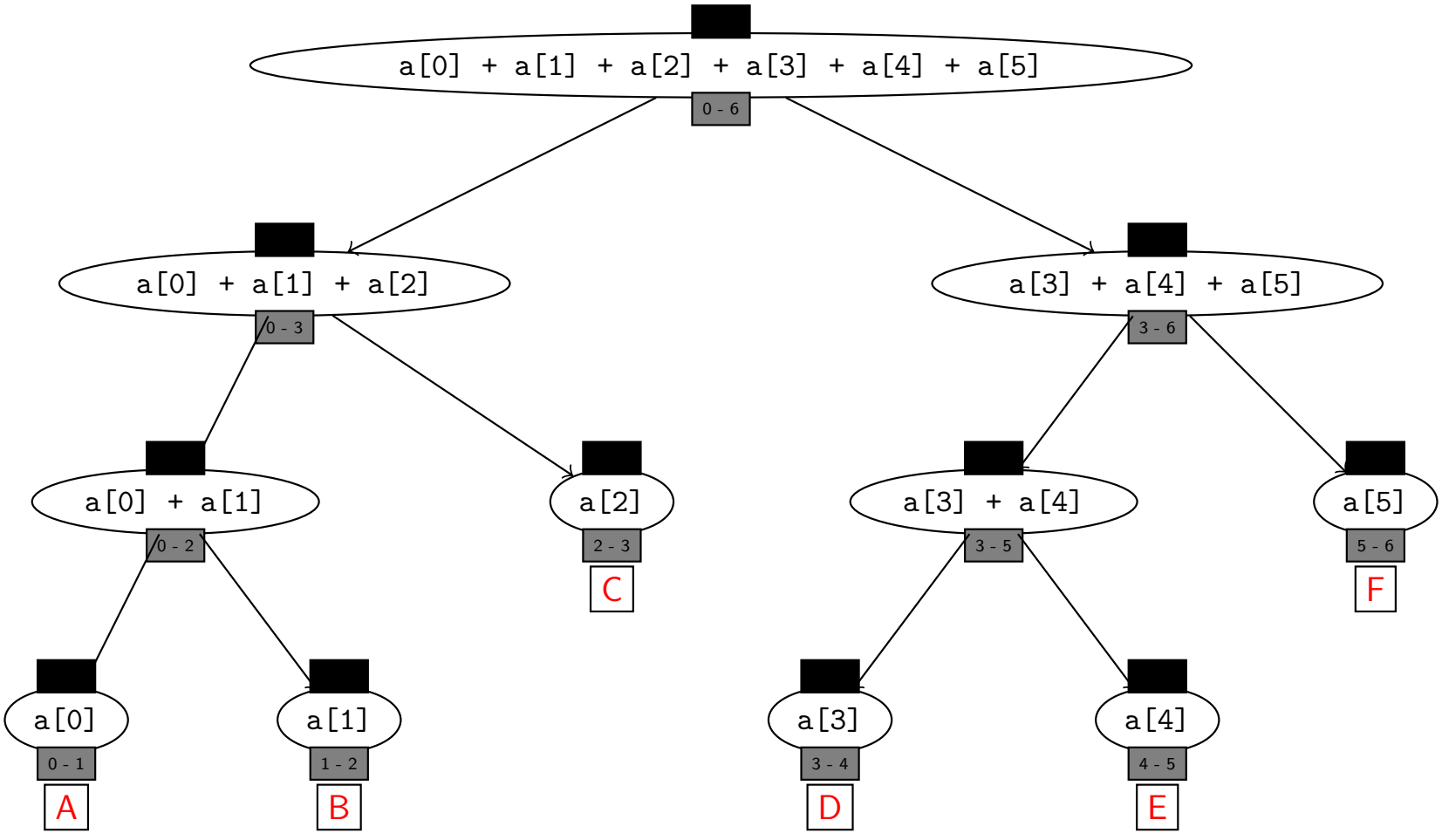
- A: I'm good!
- B: I need `a[0]`
- C: I need `a[0] + a[1]`
- D: I need `a[0] + a[1] + a[2]`
- E: I need `a[0] + a[1] + a[2] + a[3]`
- F: I need `a[0] + a[1] + a[2] + a[3] + a[4]`

This is why we need FromLeft

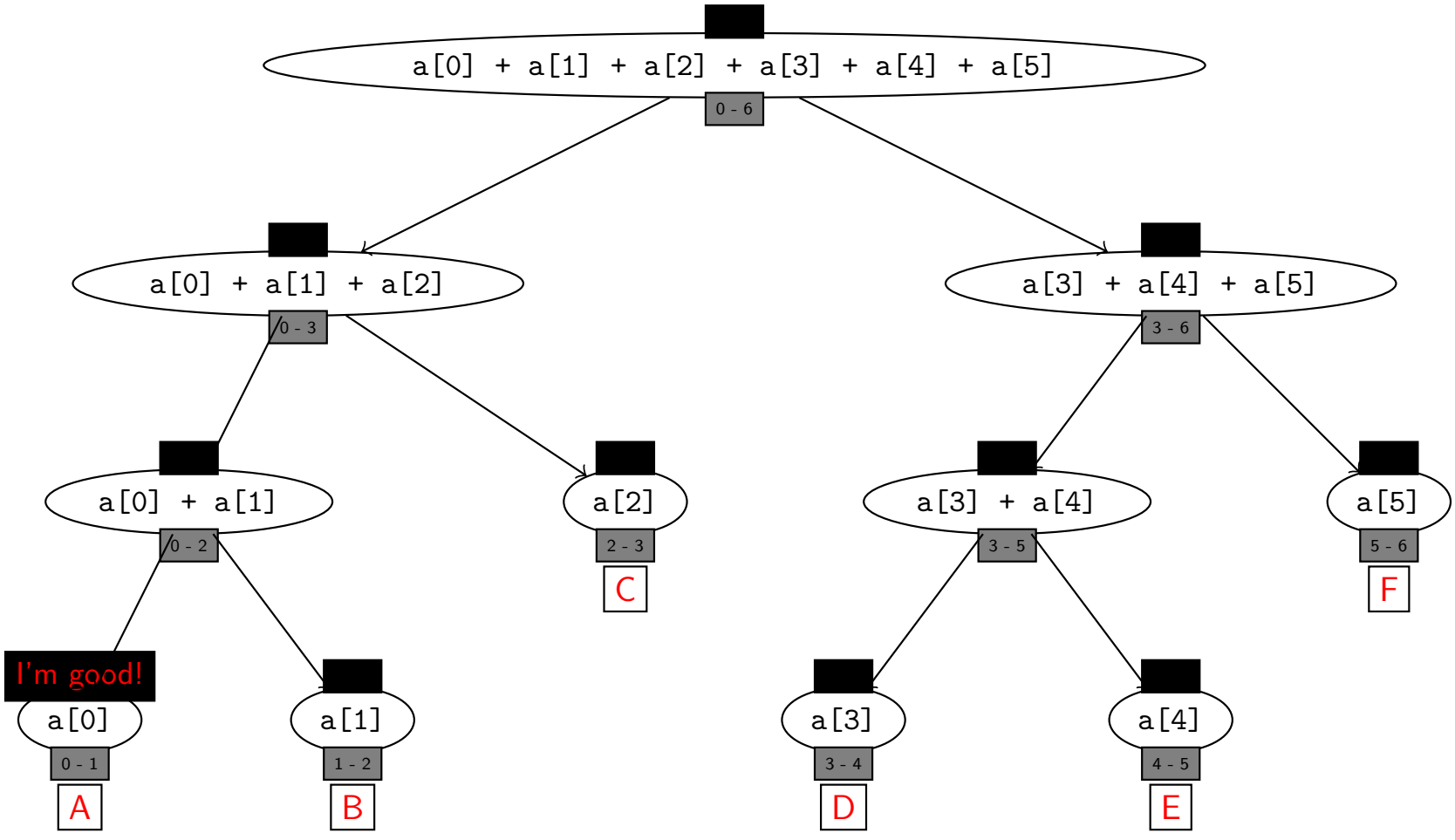
How do we store FromLeft?

- For left node, inherit from parent node
- For right node, inherit parent node + your left neighbor

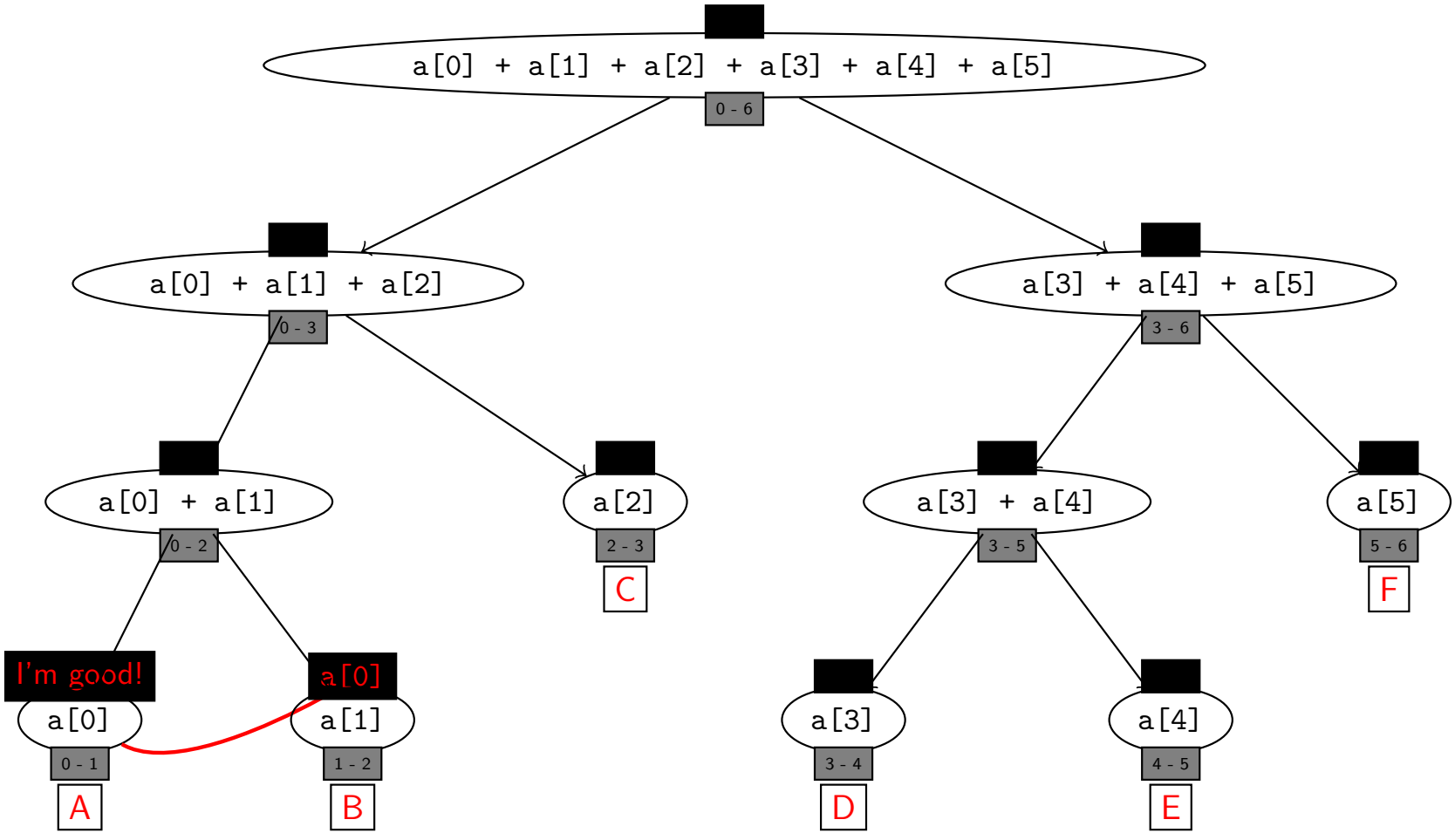
How do we store FromLeft?



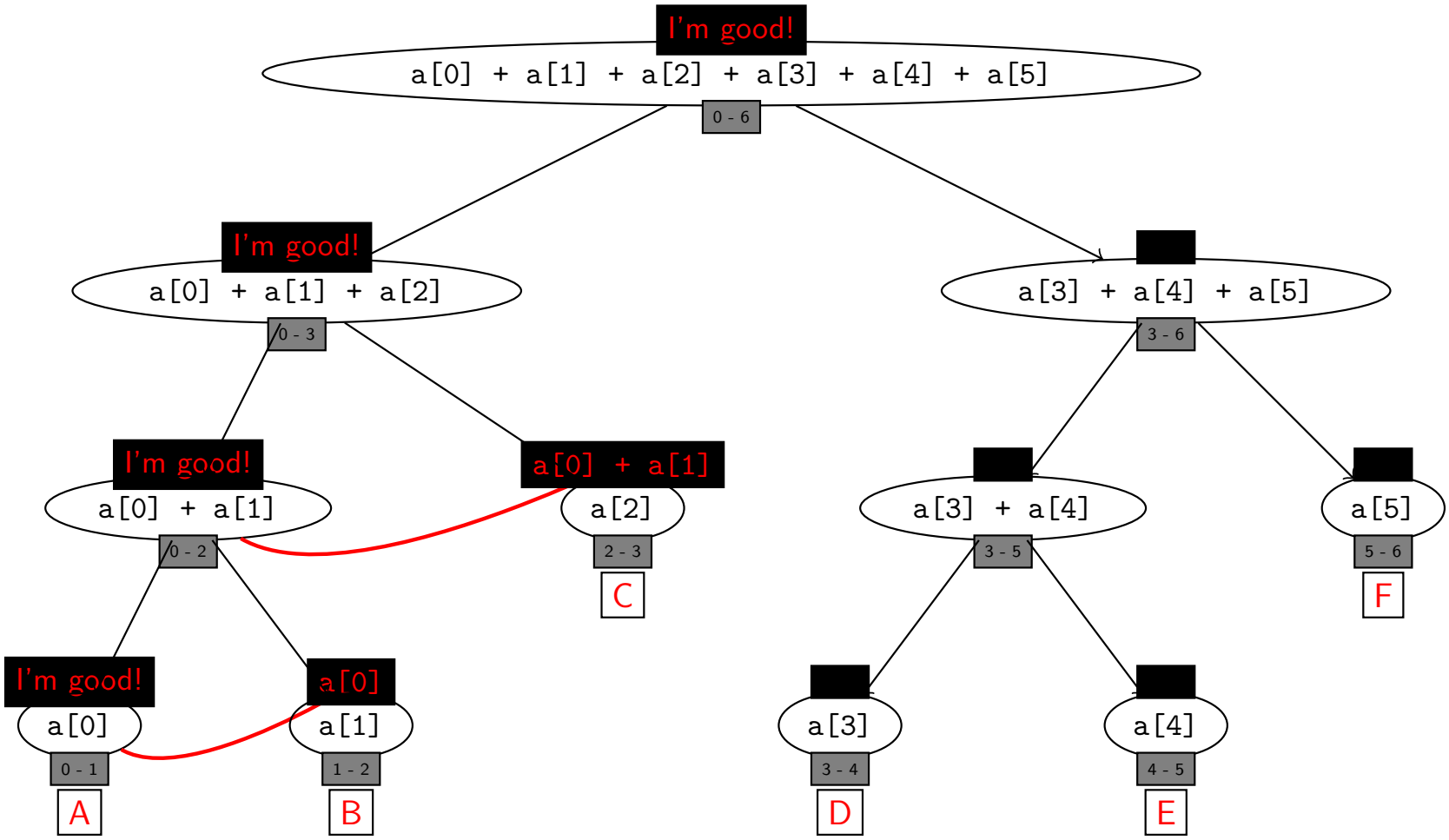
How do we store FromLeft?



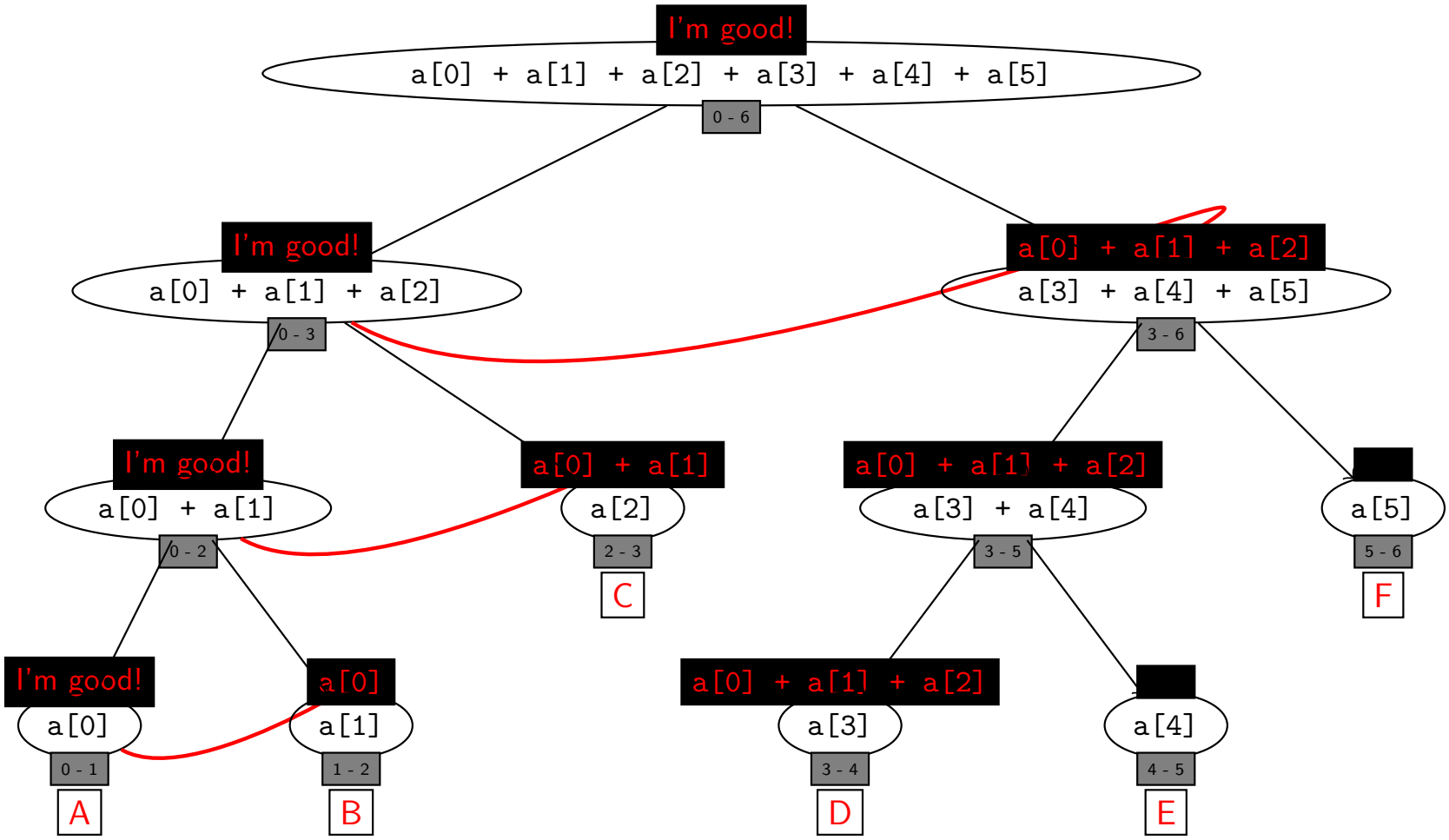
How do we store FromLeft?



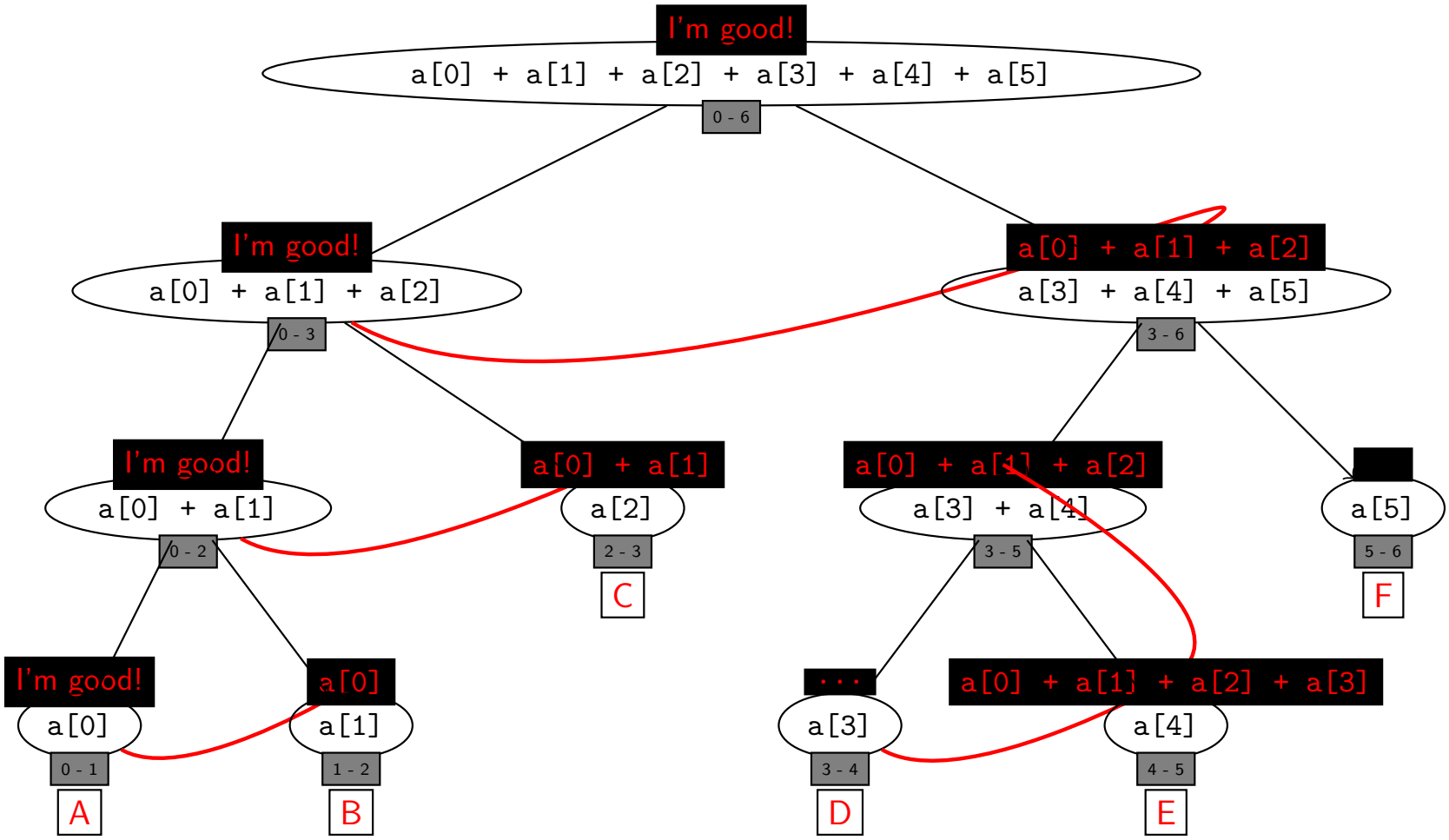
How do we store FromLeft?



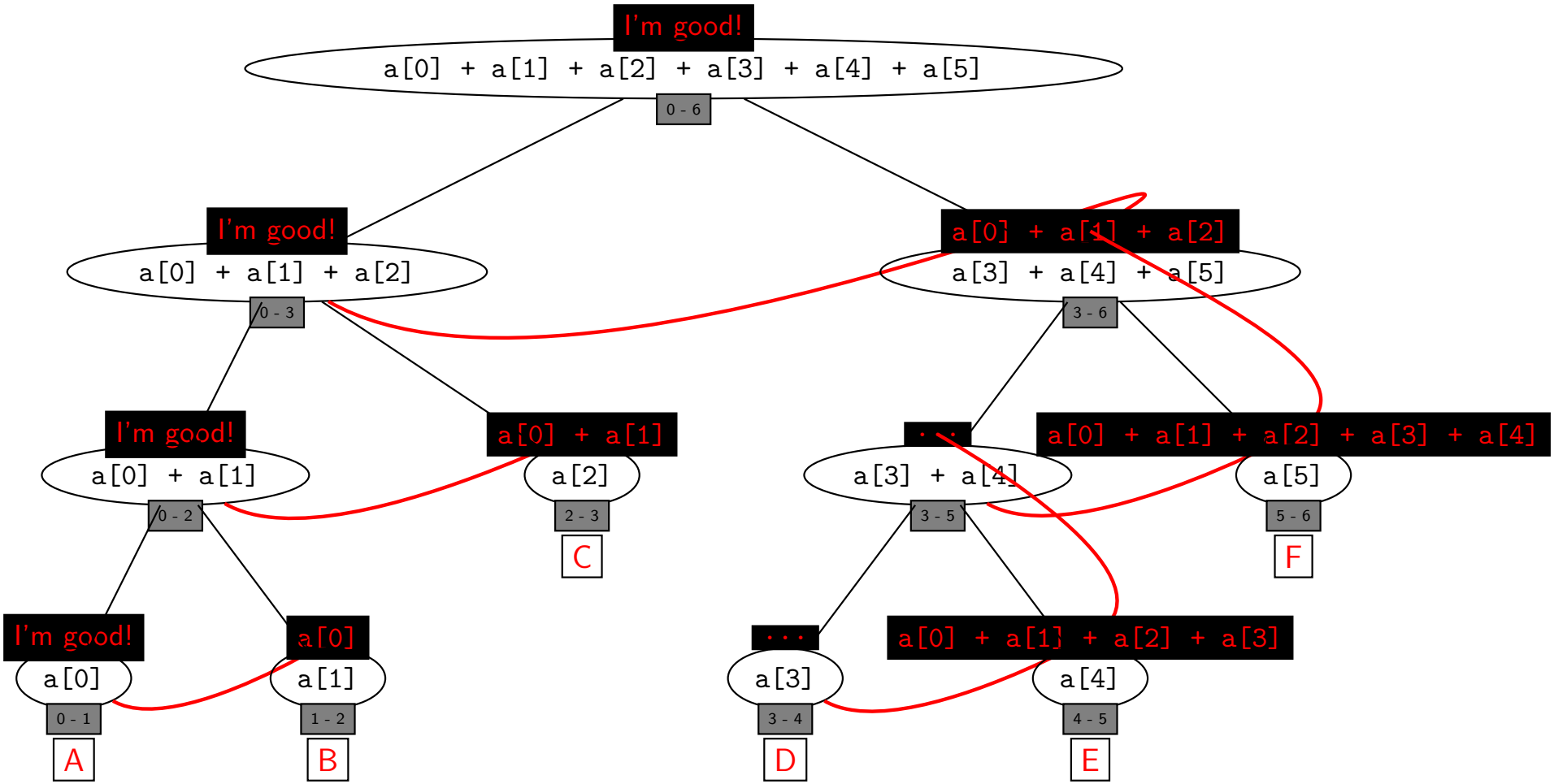
How do we store FromLeft?



How do we store FromLeft?



How do we store FromLeft?



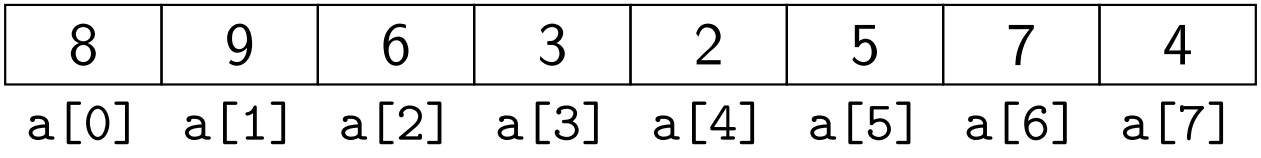
Now that we are done...

For every leaf node, PSTNode.sum + FromLeft is the result!

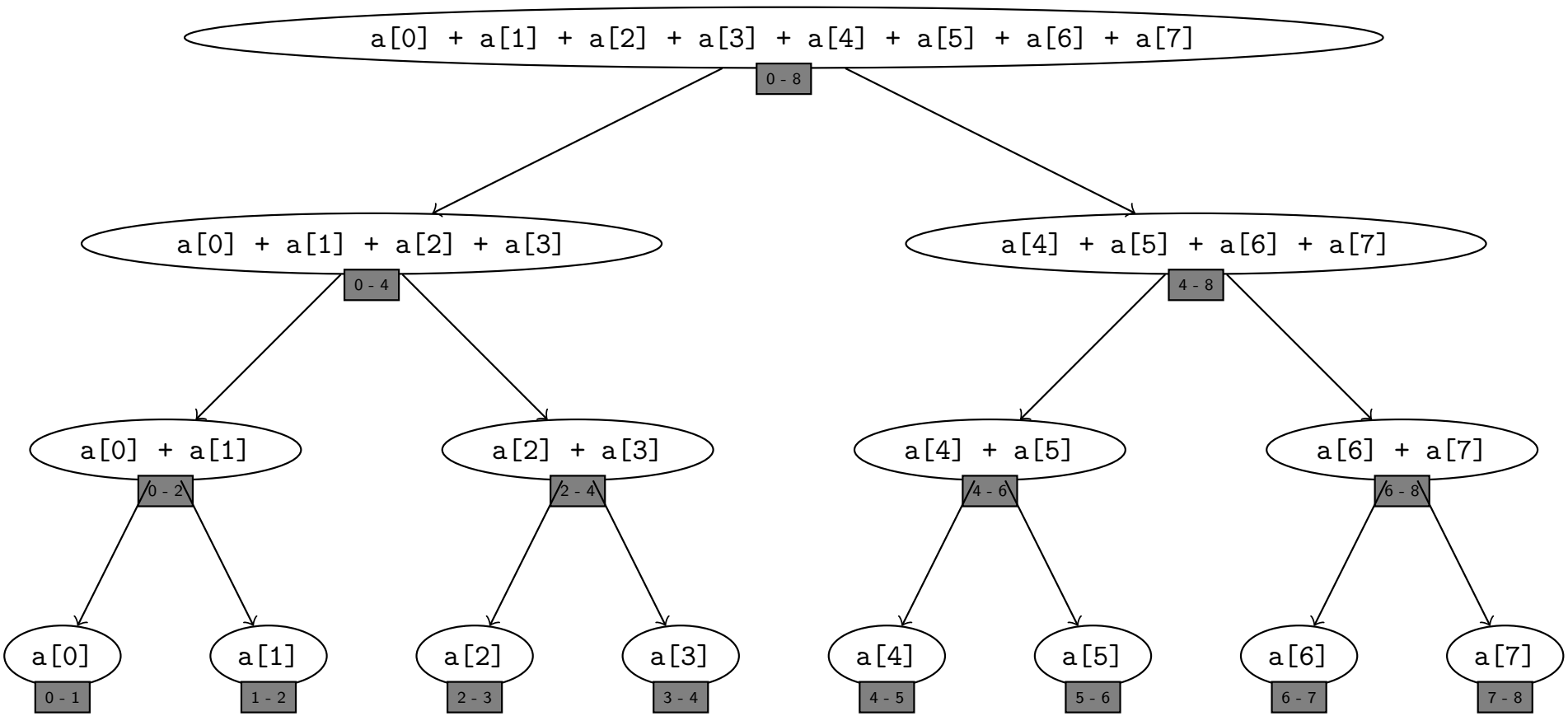
Code here, if you are interested

```
// our output
int[] output;
// prefix sum
void prefix_sum(int[] output, PSTNode curr_node, int
    fromLeft) {
  if (curr_node.left == null && curr_node.right ==
      null) { // leaf
    output[curr_node.lo] = fromLeft + curr_node.sum;
  } else {
    prefix_sum(output, curr_node.left, fromLeft);
    prefix_sum(output, curr_node.right, fromLeft +
        curr_node.left.sum);
  }
}
```

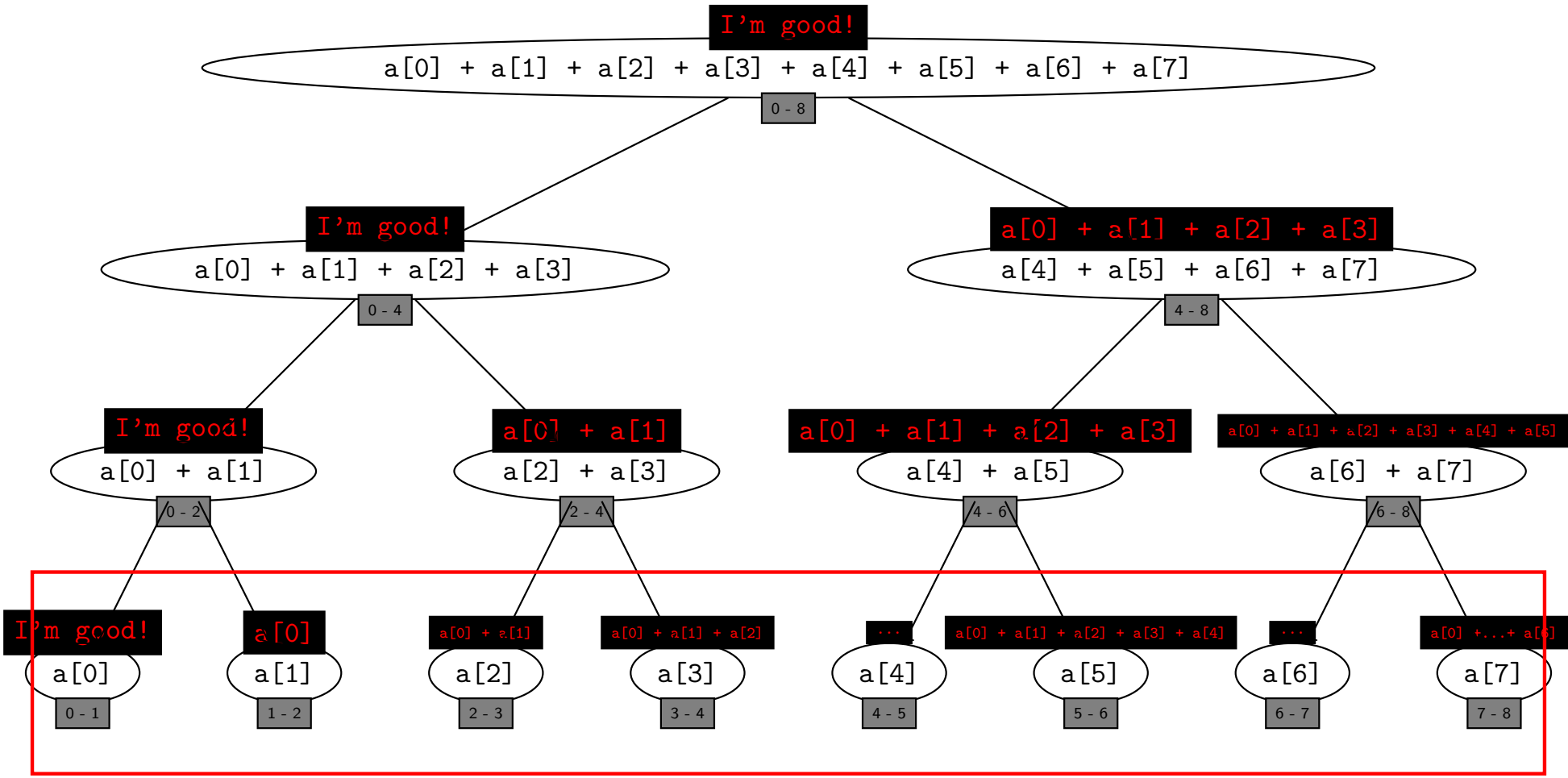
Handout solution 1: prefix sum



Step1: Build Tree



Step 2: Build FromLeft



Handout solution 2: prefix FindMin

What should PSTNode have now?

Handout solution 2: prefix FindMin

What should PSTNode have now?

`lo, hi, min, left, right`

Handout solution 2: prefix FindMin

What should PSTNode have now?

`lo, hi, min, left, right`

How should buildTree work now?

Handout solution 2: prefix FindMin

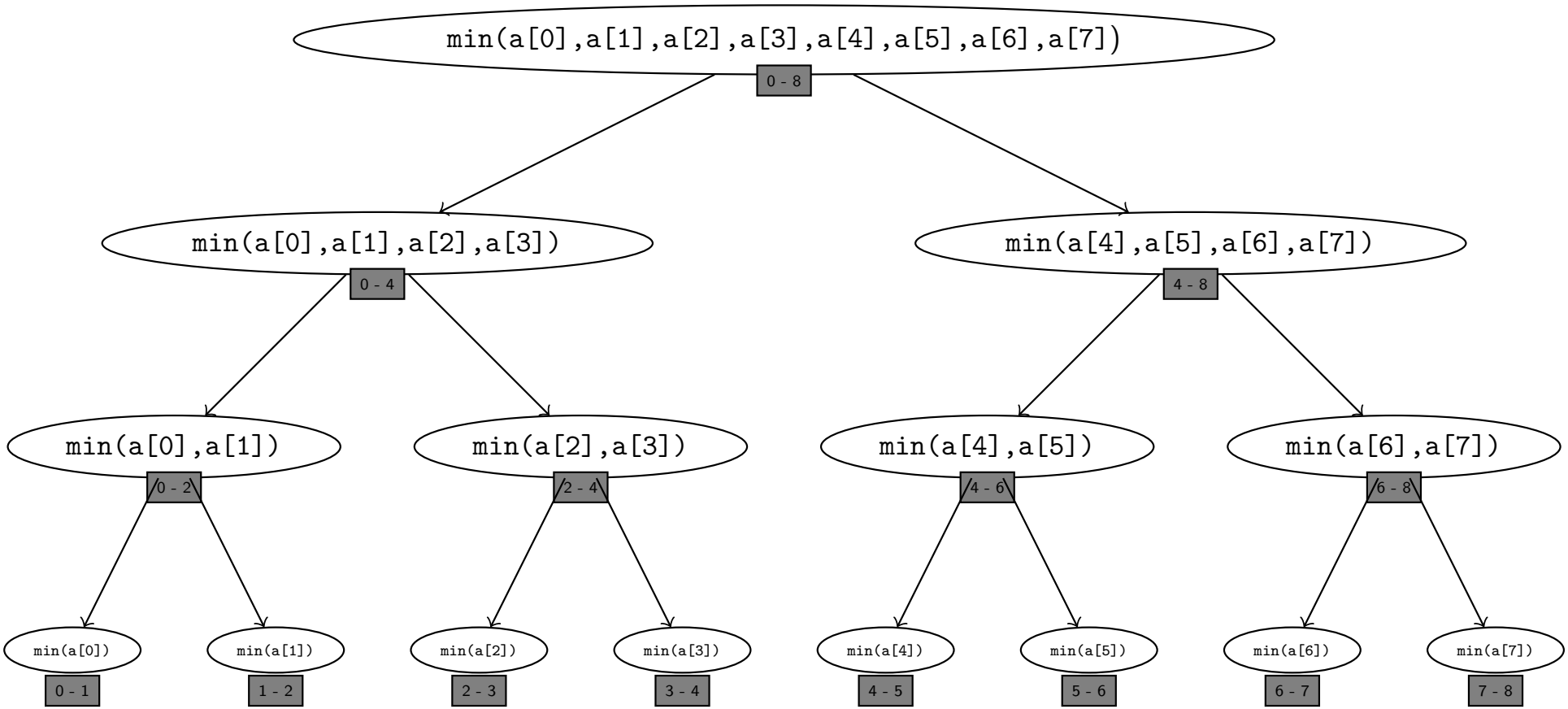
What should PSTNode have now?

`lo, hi, min, left, right`

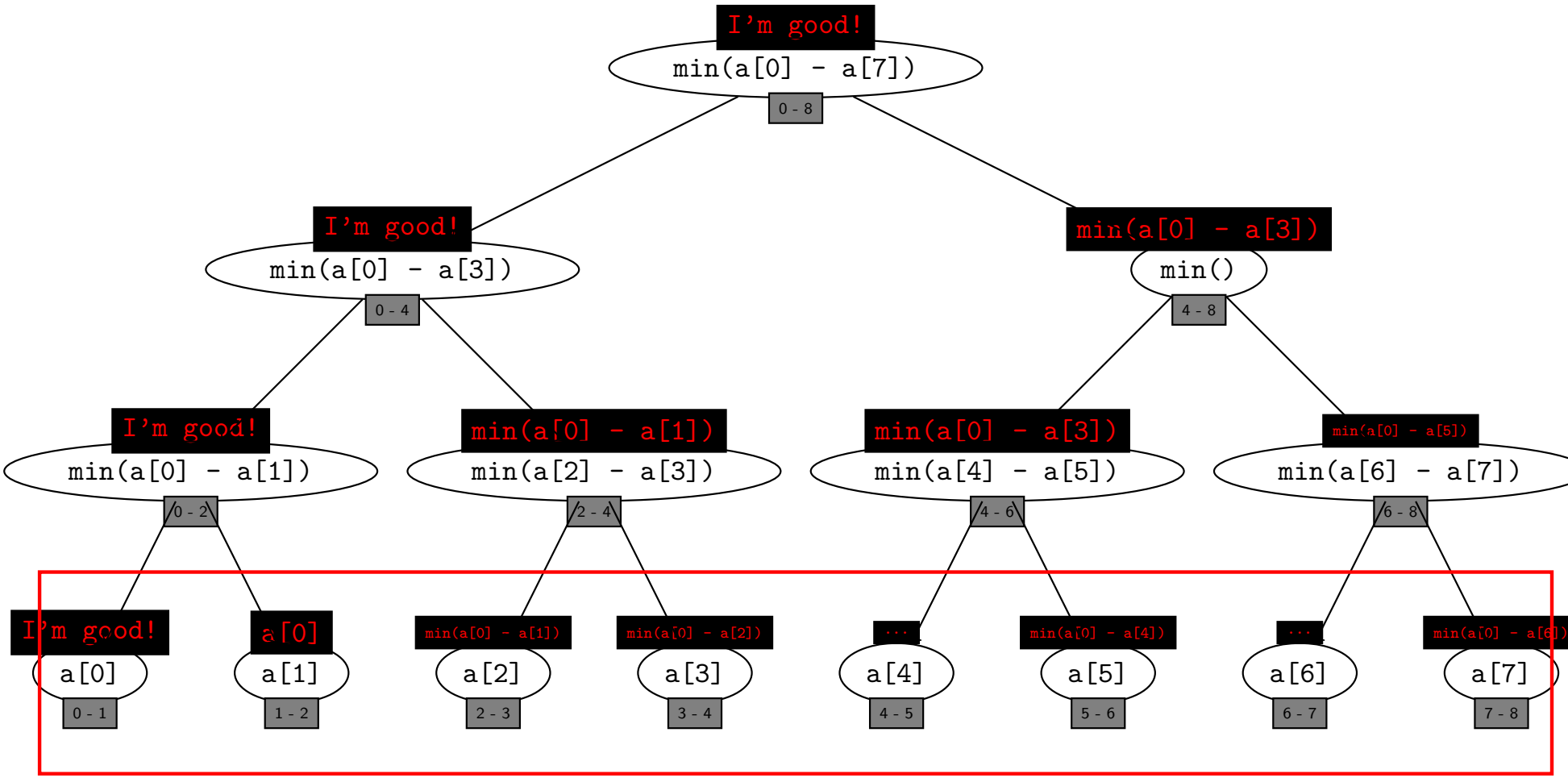
How should buildTree work now?

```
PSTNode buildTree(int[] arr, int lo, int hi) {
  if (hi - lo == 1) {
    // sum is simply arr[lo]
    return new PSTNode(lo, hi, arr[lo])
  } else {
    mid = lo + (hi - lo) / 2;
    PSTNode left = buildTree(arr, lo, mid);
    PSTNode right = buildTree(arr, mid, hi);
    // changed here!
    return new PSTNode(lo, hi, min(left.min, right.min),
        left, right);
  }
}
```

Step 1: Build Tree



Step 2: Build FromLeft



Finally, new code for findMin

```
// our output
int[] output;
// findMin
void findMin(int[] output, PSTNode curr_node, int
    fromLeft) {
  if (curr_node.left == null && curr_node.right ==
      null) { // leaf
    output[curr_node.lo] = min(fromLeft, curr_node.min);
  } else {
    findMin(output, curr_node.left, fromLeft);
    findMin(output, curr_node.right,
        min(fromLeft, curr_node.left.min));
  }
}
```