Daily Coding Problem   Blog

Daily Coding Problem #149

Problem

This problem was asked by Goldman Sachs.

Given a list of numbers L, implement a method sum(i, j) which returns the sum from the sublist L[i:j] (including i, excluding j).

For example, given L = [1, 2, 3, 4, 5], sum(1, 3) should return sum([2, 3]), which is 5.

You can assume that you can do some pre-processing. sum() should be optimized over the pre-processing step.

Solution

This problem has many different possible tradeoffs you could make:

- Have no pre-processing step and compute sum on the fly.
- Pre-process the sum of every i, j combination s.t. i < j and store it in a list.
- Create a segment tree representing sums of many segments of the list.

Let's explore the segment tree since it has a good balance of pre-processing time and query time.

Basically, a segment tree's nodes would have these fields:

- start_ind and end_ind which represents the segment in the list this node

represents.

- `val`, which represents the sum of this segment.
- `left`, for the left node.
- `right`, for the right node.

With these fields we can see how we might query for a segment.

- If the node's segment is equivalent to the query, return its `val`
- Otherwise, query for the left node if it encloses the starting index and add it to our result.
- Also query for the right node if it encloses the ending index and add it to our result.

```
result = 0

if left.start_ind <= start_ind <= left.end_ind:
    result += query(left, start_ind, min(end_ind, left.end_ind))

if right.start_ind <= end_ind <= right.end_ind:
    result += query(right, max(start_ind, right.start_ind), end_ind)
```

The full code:

```
class Node:
    def __init__(self, val, start_ind, end_ind, left=None, right=None):
        self.val = val
        self.start_ind = start_ind
        self.end_ind = end_ind
        self.left = left
        self.right = right

    @property
    def interval(self):
        return (self.start_ind, self.end_ind)


def make_segment_tree(lst):
```

```
        return _make_segment_tree(lst, 0, len(lst) - 1)


    def _make_segment_tree(lst, start_ind, end_ind):
        if start_ind == end_ind:
            assert(len(lst) == 1)
            val = lst[0]
            return Node(val, start_ind, end_ind)


        mid = len(lst) // 2


        left = _make_segment_tree(lst[:mid], start_ind, start_ind + mid - 1)
        right = _make_segment_tree(lst[mid:], start_ind + mid, end_ind)


        root_val = left.val + right.val


        return Node(root_val, start_ind, end_ind, left, right)


    def query(node, start_ind, end_ind):
        if node.start_ind == start_ind and node.end_ind == end_ind:
            return node.val


        result = 0
        left = node.left
        right = node.right


        if left.start_ind <= start_ind <= left.end_ind:
            result += query(left, start_ind, min(end_ind, left.end_ind))


        if right.start_ind <= end_ind <= right.end_ind:
            result += query(right, max(start_ind, right.start_ind), end_ind)


        return result
```

This takes O(N log N) time and space during pre-processing while querying takes O(log N) time.

© Daily Coding Problem 2019

Privacy Policy

Terms of Service

Press