Segment Trees

League of Programmers

ACA, IIT Kanpur

Outline

1. Segment Trees
2. Problems

A Simple Problem

Problem Statement

We have an array a[0 . . . n-1].

A Simple Problem

Problem Statement

We have an array a[0 . . . n-1].

A Simple Problem

Problem Statement

We have an array a[0 . . . n-1].
We should be able to

1. Find the sum of elements l to r

A Simple Problem

Problem Statement

We have an array a[0 . . . n-1].
We should be able to

1. Find the sum of elements l to r
2. Change in the value of a specified element of the array a[i]=x

A Simple Problem

Possible Solutions

A Simple Problem

Possible Solutions

- Naive one: Go on from l to r and keep on adding and update the element when you get a update request.
  Running time: O(n) to sum and O(1) to update

A Simple Problem

Possible Solutions

- Naive one: Go on from l to r and keep on adding and update the element when you get a update request.
  Running time: O(n) to sum and O(1) to update
- Store sum from start to i at the i^{th} index in an another array.
  Running time: O(1) to return sum, O(n) to update

A Simple Problem

Possible Solutions

- Naive one: Go on from l to r and keep on adding and update the element when you get a update request.
  Running time: O(n) to sum and O(1) to update
- Store sum from start to i at the i^{th} index in an another array.
  Running time: O(1) to return sum, O(n) to update
- This works well if the number of query operations are large and very few updates

A Simple Problem

Possible Solutions

- Naive one: Go on from l to r and keep on adding and update the element when you get a update request.
  Running time: O(n) to sum and O(1) to update
- Store sum from start to i at the i^{th} index in an another array.
  Running time: O(1) to return sum, O(n) to update
- This works well if the number of query operations are large and very few updates
- What if the number of query and updates are equal?

A Simple Problem

Possible Solutions

- Naive one: Go on from l to r and keep on adding and update the element when you get a update request.
  Running time: O(n) to sum and O(1) to update
- Store sum from start to i at the i^{th} index in an another array.
  Running time: O(1) to return sum, O(n) to update
- This works well if the number of query operations are large and very few updates
- What if the number of query and updates are equal?
- Can we perform both the operations in O(log n) time once given the array?

Segment Trees

Segment Trees

- Representation of the tree

Segment Trees

- Representation of the tree
  - Leaf Nodes are the elements in the array.

Segment Trees

- Representation of the tree
  - Leaf Nodes are the elements in the array.
  - Each internal node represents some merging of the leaf nodes

Segment Trees

- Representation of the tree
  - Leaf Nodes are the elements in the array.
  - Each internal node represents some merging of the leaf nodes
- Number each node in the tree level by level

Segment Trees

- Representation of the tree
  - Leaf Nodes are the elements in the array.
  - Each internal node represents some merging of the leaf nodes
- Number each node in the tree level by level
  - Observe that for each node the left child is 2*i and right child is 2*i+1

Segment Trees

- Representation of the tree
  - Leaf Nodes are the elements in the array.
  - Each internal node represents some merging of the leaf nodes
- Number each node in the tree level by level
  - Observe that for each node the left child is 2*i and right child is 2*i+1
  - For each node the parent is i/2

Segment Trees

- Representation of the tree
  - Leaf Nodes are the elements in the array.
  - Each internal node represents some merging of the leaf nodes
- Number each node in the tree level by level
  - Observe that for each node the left child is 2*i and right child is 2*i+1
  - For each node the parent is i/2
  - Just use an array to represent the tree, operate on indices to access parents and children

Segment Trees

- Representation of the tree
  - Leaf Nodes are the elements in the array.
  - Each internal node represents some merging of the leaf nodes
- Number each node in the tree level by level
  - Observe that for each node the left child is 2*i and right child is 2*i+1
  - For each node the parent is i/2
  - Just use an array to represent the tree, operate on indices to access parents and children
- Note: An important feature of the tree segments is that they use linear memory: standard tree segments requires about 4n memory elements to work on an array of size n

Solution

Solution

- We start with a segment [0 . . . n-1]. and every time we divide the current period of two (if it has not yet become a segment of length), and then calling the same procedure on both halves, and for each such segment we store the sum on it.

Solution

- We start with a segment [0 . . . n-1]. and every time we divide the current period of two (if it has not yet become a segment of length), and then calling the same procedure on both halves, and for each such segment we store the sum on it.
- In other words, we calculate and remember somewhere sum of the elements of the array, ie segment a[0 . . . n-1]. Also calculate the amount of the two halves of the array: a$[0 \ldots n/2]$ and a$[n/2+1 \ldots n-1]$. Each of the two halves, in turn, divide in half and count the amount to keep them, then divide in half again, and so on until it reaches the current segment length 1

Solution

- We start with a segment [0 . . . n-1]. and every time we divide the current period of two (if it has not yet become a segment of length), and then calling the same procedure on both halves, and for each such segment we store the sum on it.
- In other words, we calculate and remember somewhere sum of the elements of the array, ie segment a[0 . . . n-1]. Also calculate the amount of the two halves of the array: a$[0 \ldots n/2]$ and a$[n/2+1 \ldots n-1]$. Each of the two halves, in turn, divide in half and count the amount to keep them, then divide in half again, and so on until it reaches the current segment length 1
- The number of vertices in the worst case is estimated at

$$n + n/2 + n/4 + n/8 + \cdots + 1 < 2n$$

Solution

- We start with a segment [0 . . . n-1]. and every time we divide the current period of two (if it has not yet become a segment of length), and then calling the same procedure on both halves, and for each such segment we store the sum on it.
- In other words, we calculate and remember somewhere sum of the elements of the array, ie segment a[0 . . . n-1]. Also calculate the amount of the two halves of the array: a$[0 \dots n/2]$ and a$[n/2+1 \dots n-1]$. Each of the two halves, in turn, divide in half and count the amount to keep them, then divide in half again, and so on until it reaches the current segment length 1
- The number of vertices in the worst case is estimated at

$$n + n/2 + n/4 + n/8 + \cdots + 1 < 2n$$

- The height of the tree is the value of the segments O(log n).

Implementation

Building the tree

Implementation

Building the tree

- From the bottom up: first write the values of the elements a[i] the corresponding leaves of the tree, then on the basis of these values to calculate the nodes of the previous level as the sum of the two leaves, then similarly calculate values for one more level, etc. Convenient to describe the operation recursively.

Implementation

Building the tree

- From the bottom up: first write the values of the elements a[i] the corresponding leaves of the tree, then on the basis of these values to calculate the nodes of the previous level as the sum of the two leaves, then similarly calculate values for one more level, etc. Convenient to describe the operation recursively.
- Time to do this?

Implementation

Building the tree

- From the bottom up: first write the values of the elements a[i] the corresponding leaves of the tree, then on the basis of these values to calculate the nodes of the previous level as the sum of the two leaves, then similarly calculate values for one more level, etc. Convenient to describe the operation recursively.
- Time to do this?

Implementation

Building the tree

- From the bottom up: first write the values of the elements a[i] the corresponding leaves of the tree, then on the basis of these values to calculate the nodes of the previous level as the sum of the two leaves, then similarly calculate values for one more level, etc. Convenient to describe the operation recursively.
- Time to do this?
  O(n) since each node in the tree is modified once and uses only a max of 2 nodes (already computed) for computation.

Implementation

Query Request

Implementation

Query Request

- The input is two numbers l and r. And we have the time $O(logn)$. Calculate the sum of the segment a$[l \dots r]$.

Implementation

Query Request

- The input is two numbers l and r. And we have the time $O(logn)$. Calculate the sum of the segment a[$l \ldots r$] .
- Can be done recursively

Implementation

Query Request

- The input is two numbers l and r. And we have the time $O(logn)$. Calculate the sum of the segment a$[l \ldots r]$.
- Can be done recursively
    - If your range is within the segment completely, return the value at that node

Implementation

Query Request

- The input is two numbers l and r. And we have the time $O(logn)$. Calculate the sum of the segment a$[l \dots r]$.
- Can be done recursively
  - If your range is within the segment completely, return the value at that node
  - If its completely out of range, return 0 or null

Implementation

Query Request

- The input is two numbers l and r. And we have the time $O(logn)$. Calculate the sum of the segment a[$l \ldots r$] .
- Can be done recursively
  - If your range is within the segment completely, return the value at that node
  - If its completely out of range, return 0 or null
  - If its in one of the child, query on that child

Implementation

Query Request

- The input is two numbers l and r. And we have the time $O(logn)$. Calculate the sum of the segment a$[l \dots r]$.
- Can be done recursively
  - If your range is within the segment completely, return the value at that node
  - If its completely out of range, return 0 or null
  - If its in one of the child, query on that child
  - If its in both the child, do query on both of them

Implementation

Query Request

Pseudocode

```
 query(node,l,r) {
  if range of node is within l and r
   return value in node
  else if range of node is completely outside l and r
   return 0
  else
   return
sum(query(left-child,l,r),query(right-child,l,r))
}
```

Implementation

Query Request

Pseudocode

```
 query(node,l,r) {
  if range of node is within l and r
   return value in node
  else if range of node is completely outside l and r
   return 0
  else
   return
sum(query(left-child,l,r),query(right-child,l,r))
}
```

- But this doesn't look O(log n)

Implementation

Query Request

Pseudocode

```
query(node,l,r) {
 if range of node is within l and r
  return value in node
 else if range of node is completely outside l and r
  return 0
 else
  return
sum(query(left-child,l,r),query(right-child,l,r))
}
```

- But this doesn't look O(log n)
- It is.
  At any level of the tree, the maximum number of segments that could call our recursive function when processing a request is 4.

Implementation

Query Request

Pseudocode

```
 query(node,l,r) {
  if range of node is within l and r
   return value in node
  else if range of node is completely outside l and r
   return 0
  else
   return
sum(query(left-child,l,r),query(right-child,l,r))
}
```

- But this doesn't look O(log n)
- It is.
  At any level of the tree, the maximum number of segments that could call our recursive function when processing a request is 4.
- So, only O(log n) running time.

Implementation

Renewal Request

Implementation

Renewal Request

- Given an index i and the value of x. What to do?

Implementation

Renewal Request

- Given an index i and the value of x. What to do?
- Update the nodes in the tree so as to conform to the new value a[i]=x in O(log n).

Implementation

Renewal Request

- Given an index i and the value of x. What to do?
- Update the nodes in the tree so as to conform to the new value a[i]=x in O(log n).
- How many nodes and what nodes will be affected?

Implementation

Renewal Request

- Given an index i and the value of x. What to do?
- Update the nodes in the tree so as to conform to the new value a[i]=x in O(log n).
- How many nodes and what nodes will be affected?

Implementation

Renewal Request

- Given an index i and the value of x. What to do?
- Update the nodes in the tree so as to conform to the new value a[i]=x in O(log n).
- How many nodes and what nodes will be affected?
  The nodes from i^{th} leaf node to the way upto the root of the tree.
- Then it is clear that the update request can be implemented as a recursive function: it sends the current node of the tree lines, and this function performs a recursive call from one of his two sons (the one that contains the position i in its segment), and after that - counts the value of the sum in the current node in the same way as we did in the construction of a tree of segments (ie, the sum of the values for the two sons of the current node).

Implementation

Problem

Implementation

Problem

- You are given an array. You have queries which ask for the maximum element in the range l to r. And you have updates which increase the value of a particular element in the array with some value val.

Implementation

Problem

- You are given an array. You have queries which ask for the maximum element in the range l to r. And you have updates which increase the value of a particular element in the array with some value val.
- But seriously, how to code?

Implementation

Problem

- You are given an array. You have queries which ask for the maximum element in the range l to r. And you have updates which increase the value of a particular element in the array with some value val.
- But seriously, how to code?
- Lets look at a code

Outline

Problems

Links:

1. http://www.spoj.pl/problems/GSS1/
2. http://www.spoj.pl/problems/GSS3/
3. http://www.spoj.pl/problems/HORRIBLE/
4. http://www.spoj.pl/problems/BRCKTS/
5. http://www.spoj.pl/problems/HELPR2D2/
6. http://www.spoj.pl/problems/KFSTD/
7. http://www.spoj.pl/problems/FREQUENT/
8. http://www.spoj.pl/problems/LITE/