# Segment Trees

League of Programmers

ACA, IIT Kanpur

# Outline

# Binary Indexed Tree

## Representation

# Binary Indexed Tree

## Representation

1. Used to represent almost complete binary trees

# Binary Indexed Tree

## Representation

1. Used to represent almost complete binary trees
2. n elements stored in $\log n$ levels

# Binary Indexed Tree

## Representation

1. Used to represent almost complete binary trees
2. n elements stored in $\log n$ levels
3. Stored in array

# Binary Indexed Tree

## Representation

1. Used to represent almost complete binary trees
2. n elements stored in $\log n$ levels
3. Stored in array
4. Indexed root to leaves, (1,...,n) or (0,...,n-1).

# Binary Indexed Tree

## Representation

1. Used to represent almost complete binary trees
2. n elements stored in $\log n$ levels
3. Stored in array
4. Indexed root to leaves, (1,...,n) or (0,...,n-1).
5. Choosing (1,...,n)

# Binary Indexed Tree

## Properties

# Binary Indexed Tree

## Properties

1. Level i starts at index $2^i$.

# Binary Indexed Tree

## Properties

1. Level i starts at index $2^i$.
2. Parent of vertex at $i$ is $\lfloor \frac{i}{2} \rfloor$

# Binary Indexed Tree

## Properties

1. Level i starts at index $2^i$.
2. Parent of vertex at $i$ is $\lfloor \frac{i}{2} \rfloor$
3. Children of vertex at $i$ is $2 * i$ and $2 * i + 1$

# Binary Indexed Tree

## Applications

# Binary Indexed Tree

## Applications

1. Binary Heap

# Binary Indexed Tree

## Applications

1. Binary Heap
2. Segment trees

# Outline

1. Essential Tools
2. Segment Trees
3. Segment trees for Rooted Trees
4. Problems

# Motivation

## Problem Statement

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

# Motivation

## Problem Statement

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

# Motivation

**Problem Statement**

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

1. Find the sum of elements l to r

# Motivation

## 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 (Using Simple array)

# A Simple Problem

## Possible Solutions (Using Simple array)

- 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 (Using Simple array)

- 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 (Using Simple array)

- 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 (Using Simple array)

- 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 (Using Simple array)

- 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

## Possible Solutions (Using Binary Indexed Trees)

# Segment Trees

## Possible Solutions (Using Binary Indexed Trees)

- Representation of the tree

# Segment Trees

## Possible Solutions (Using Binary Indexed Trees)

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

# Segment Trees

## Possible Solutions (Using Binary Indexed Trees)

- Representation of the tree
  - Leaf Nodes are the elements in the array.
  - Each internal node represents all leaf nodes in its subtree.

# Segment Trees

## Possible Solutions (Using Binary Indexed Trees)

- Representation of the tree
  - Leaf Nodes are the elements in the array.
  - Each internal node represents all leaf nodes in its subtree.
  - For this problem sum of all the leaf values in subtree.

# Segment Trees

## Possible Solutions (Using Binary Indexed Trees)

- Representation of the tree
    - Leaf Nodes are the elements in the array.
    - Each internal node represents all leaf nodes in its subtree.
    - For this problem sum of all the leaf values in subtree.
- Size of complete Binary tree with n leaves?

# Segment Trees

## Possible Solutions (Using Binary Indexed Trees)

- Representation of the tree
  - Leaf Nodes are the elements in the array.
  - Each internal node represents all leaf nodes in its subtree.
  - For this problem sum of all the leaf values in subtree.
- Size of complete Binary tree with n leaves?
- 4n, How?

# Segment Trees

## Possible Solutions (Using Binary Indexed Trees)

- Representation of the tree
  - Leaf Nodes are the elements in the array.
  - Each internal node represents all leaf nodes in its subtree.
  - For this problem sum of all the leaf values in subtree.
- Size of complete Binary tree with n leaves?
- 4n, How?
- Levels $\log n$

# Implementation

## Building the tree

## Implementation

### Building the tree

- Size of tree $2^{l+1}$ where { `for(l=0;(1<<l)<n;l++);` }

# Implementation

## Building the tree

- Size of tree $2^{l+1}$ where { `for(l=0;(1«l)<n;l++);` }
- From the bottom up: first write the values of the elements a[i] the corresponding leaves T[(1«l)+i]

# Implementation

## Building the tree

- Size of tree $2^{l+1}$ where { `for(l=0;(1«l)<n;l++);` }
- From the bottom up: first write the values of the elements a[i] the corresponding leaves T[(1«l)+i]
- Calculate the nodes of the previous level as the sum of the two leaves

# Implementation

## Building the tree

- Size of tree $2^{l+1}$ where { `for(l=0;(1«l)<n;l++);` }
- From the bottom up: first write the values of the elements a[i] the corresponding leaves T[(1«l)+i]
- Calculate the nodes of the previous level as the sum of the two leaves
- ```
  for(i=(1«l)-1;i>0;i - -)
  T[i]= sum(T[2*i],T[2*i+1]);
  ```
  ```

# Implementation

## Building the tree

- Size of tree $2^{l+1}$ where { `for(l=0;(1«l)<n;l++);` }
- From the bottom up: first write the values of the elements a[i] the corresponding leaves T[(1«l)+i]
- Calculate the nodes of the previous level as the sum of the two leaves
- ```
  for(i=(1«l)-1;i>0;i - -)
  T[i]= sum(T[2*i],T[2*i+1]);
  ```
- Time to do this?
  ```

# Implementation

## Building the tree

- Size of tree $2^{l+1}$ where { `for(l=0;(1«l)<n;l++);` }
- From the bottom up: first write the values of the elements a[i] the corresponding leaves T[(1«l)+i]
- Calculate the nodes of the previous level as the sum of the two leaves
- ```
  for(i=(1«l)-1;i>0;i - -)
  T[i]= sum(T[2*i],T[2*i+1]);
  ```
- Time to do this?
  ```

# Implementation

## Building the tree

- Size of tree $2^{l+1}$ where { `for(l=0;(1«l)<n;l++);` }
- From the bottom up: first write the values of the elements a[i] the corresponding leaves T[(1«l)+i]
- Calculate the nodes of the previous level as the sum of the two leaves
- `for(i=(1«l)-1;i>0;i - -)`
  `T[i]= sum(T[2*i],T[2*i+1]);`
- 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

## Implementation

### Query

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

# Implementation

## Query

- 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

- 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

# Implementation

## Query

- 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

- 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

# Implementation

## Query

- 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(Recursive)

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(Recursive)

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(Recursive)

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(Recursive)

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

# Implementation

## Query

- Iteratively??

# Implementation

## Query

- Iteratively??
- What is stored at LCA of $l$ and $r$ in tree.

# Implementation

## Query

- Iteratively??
- What is stored at LCA of $l$ and $r$ in tree.
- How do we remove excess sum. In log n time.

# Implementation

## Query

- Iteratively??
- What is stored at LCA of $l$ and $r$ in tree.
- How do we remove excess sum. In log n time.
- Prob: if left is not left and right is not right.

# Implementation

## Query Request(Iterative)

```
 query(l,r,k(levels)) {
Sum=0; l= (1«k)+l; r= (1«k)+r;
while(l!=r) {
    if(l is not left child)
        sum - = T[l];
    if(r is not right child)
        sum - = T[r];
    l/=2,r/=2;}
return sum + T[l];
}
```

# 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.
- Iterative Code

```
 Update(Index i,Change c,k(levels)) {
i= (1«k)+i;
while(i!=0) {
    T[i]+=c;
    i/=2; } }
```

# Implementation

## Variants

# Implementation

## Variants

- Find Maximum/Minimum instead of Sum

# Implementation

## Variants

- Find Maximum/Minimum instead of Sum
- Array of 1's and O's. Find total 1's or 0's in range l to r, Flip a bit.

# Implementation

## Variants

- Find Maximum/Minimum instead of Sum
- Array of 1's and O's. Find total 1's or 0's in range l to r, Flip a bit.
- Remove a part of array. Or exchange two parts of arrays.

# Implementation

## Variants

- Find Maximum/Minimum instead of Sum
- Array of 1's and O's. Find total 1's or 0's in range l to r, Flip a bit.
- Remove a part of array. Or exchange two parts of arrays.
- Add/Subtract $x$ from all values in range $l$ and $r$
  { Keep extra variable S[i] at each node for lazy update.}

# Implementation

## Variants

- Find Maximum/Minimum instead of Sum
- Array of 1's and O's. Find total 1's or 0's in range l to r, Flip a bit.
- Remove a part of array. Or exchange two parts of arrays.
- Add/Subtract $x$ from all values in range $l$ and $r$
  { Keep extra variable S[i] at each node for lazy update.}
- Again add at LCA remove intermediate.

# Implementation

## Variants

- Find Maximum/Minimum instead of Sum
- Array of 1's and O's. Find total 1's or 0's in range l to r, Flip a bit.
- Remove a part of array. Or exchange two parts of arrays.
- Add/Subtract $x$ from all values in range $l$ and $r$
  { Keep extra variable S[i] at each node for lazy update.}
- Again add at LCA remove intermediate.
- Similarily for 1's and 0's flip all the bits in range l to r.

# Implementation

## Add Range(Iterative)

```
 query(l,r,Change c,k(levels)) {
l= (1«k)+l; r= (1«k)+r;k1=0;
while(l!=r) {
if(k!=0)
    T[l]=sum(T[2*l],T[2*l+1]);
    T[r]=sum(T[2*r],T[2*r+1]);
    if(l is not left child)
        S[i-1]-=c;
        T[i-1]-= c*(1«k1);
    if(r is not right child)
        S[i+1]-=c;
        T[i+1]-= c*(1«k1);
    l/=2,r/=2;k1++;}
}
```

# Implementation

## Add Range(Iterative)

```
 if(k!=0) T[l]=sum(T[2*l],T[2*l+1])
S[l]+=c;
T[l]+= c*(1«k1);
while(l!=0) T[l]=sum(T[2*l],T[2*l+1]);
```

# Outline

# Case of Rooted trees

## Problem

Given a rooted tree with edges $E$ and vertices $V$.

# Case of Rooted trees

## Problem

Given a rooted tree with edges $E$ and vertices $V$.

- Weight of vertices or edges.

# Case of Rooted trees

## Problem

Given a rooted tree with edges $E$ and vertices $V$.

- Weight of vertices or edges.
- Query and Updates on Subtree.

# Case of Rooted trees

## Problem

Given a rooted tree with edges $E$ and vertices $V$.

- Weight of vertices or edges.
- Query and Updates on Subtree.
- Query and Updates on paths.

# Case of Rooted trees

## Problem

Given a rooted tree with edges $E$ and vertices $V$.

- Weight of vertices or edges.
- Query and Updates on Subtree.
- Query and Updates on paths.
- We know solution for linear array probs.

# Case of Rooted trees

## Problem

Given a rooted tree with edges $E$ and vertices $V$.

- Weight of vertices or edges.
- Query and Updates on Subtree.
- Query and Updates on paths.
- We know solution for linear array probs.
- Reduce these structures to linear structures.

# Case of Rooted trees

## Query and Updates on Subtree.

# Case of Rooted trees

## Query and Updates on Subtree.

# Case of Rooted trees

**Query and Updates on Subtree.**

Pre Order Numbering

**Query and Updates on Path.**

# Case of Rooted trees

**Query and Updates on Subtree.**

Pre Order Numbering

**Query and Updates on Path.**

# Case of Rooted trees

**Query and Updates on Subtree.**

Pre Order Numbering

**Query and Updates on Path.**

Heavy Light Decomposition

# Heavy Light Decomposition

## Structure

Divide the whole edges into Heavy and Light Edges.

# Heavy Light Decomposition

## Structure

Divide the whole edges into Heavy and Light Edges.

- Edge which hangs more than half of descendants is Heavy.

# Heavy Light Decomposition

## Structure

Divide the whole edges into Heavy and Light Edges.

- Edge which hangs more than half of descendants is Heavy.
- Edge which hangs less than half of descendants is Light.

# Heavy Light Decomposition

## Structure

Divide the whole edges into Heavy and Light Edges.

- Edge which hangs more than half of descendants is Heavy.
- Edge which hangs less than half of descendants is Light.
- Example ?

# Heavy Light Decomposition

## Structure

Divide the whole edges into Heavy and Light Edges.

- Edge which hangs more than half of descendants is Heavy.
- Edge which hangs less than half of descendants is Light.
- Example ?
- Light edges divides the tree into set of heavy paths.

# Heavy Light Decomposition

## Structure

Divide the whole edges into Heavy and Light Edges.

- Edge which hangs more than half of descendants is Heavy.
- Edge which hangs less than half of descendants is Light.
- Example ?
- Light edges divides the tree into set of heavy paths.
- Any path in tree can have how many heavy path intervals??

# Heavy Light Decomposition

## Structure

Divide the whole edges into Heavy and Light Edges.

- Edge which hangs more than half of descendants is Heavy.
- Edge which hangs less than half of descendants is Light.
- Example ?
- Light edges divides the tree into set of heavy paths.
- Any path in tree can have how many heavy path intervals??
- O(log n)

# Heavy Light Decomposition

## Variants

Dynamic Link and Cut.

# Heavy Light Decomposition

## Variants

Dynamic Link and Cut.

# Heavy Light Decomposition

## Variants

Dynamic Link and Cut.
ST Trees

# 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/