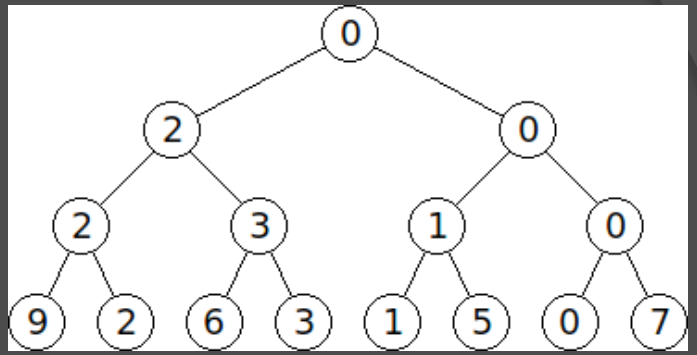
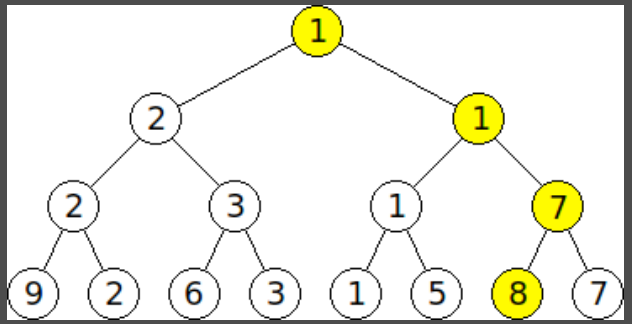
Advanced Computer Contest Preparation
Lecture 23

SEGMENT TREE LAZY PROPAGATION

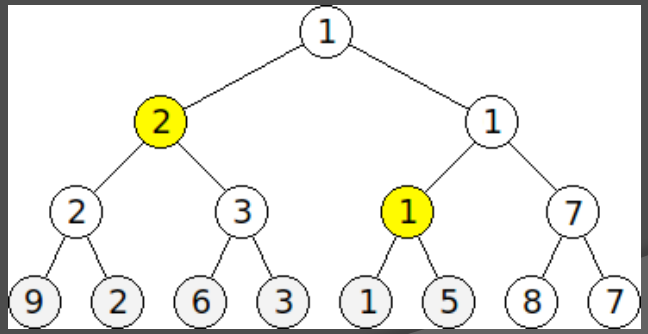
Review

- Segment trees
  - Build
  - Point update
  - Point and range query





Update 7th element



Query [1,6]

Range Updates

- Range update means updating several consecutive elements at once
- Based on what we know, how do we update a range?
  - Call point update on each leaf node individually
    - Time complexity: $O(N \log N)$
  - Improvement: start at the leftmost node and follow a path similar to the one used for `build()` until the rightmost node
    - Time complexity: $O(N)$

Lazy Principle

- "We don't need to do something unless it is absolutely necessary to do so"
- Example:
  - You have an array of N integers
  - 2 operations:
    - Add a constant to all elements
    - Get the x^{th} element
  - When updating, do not update each individual element
  - Instead, keep track of total updates and apply to individual element when query is required

Lazy Propagation

- The application of the lazy principle on segment trees
- We do not update nodes unless it is absolutely necessary to do so
- We mark nodes as lazy, meaning that it should be updated later, only if necessary

Lazy Value

- By setting a node's lazy value, we mark it as lazy
- Each answer in a node should have an associated lazy value
- ***Lazy value is generally what the node's answer should be after an actual update***
- Each lazy value should have an "unset" value, a value specifying that the node is not marked as lazy
  - Examples: 0, $+\infty$, $-\infty$
- This value should never be a possible update value

Lazy Propagation

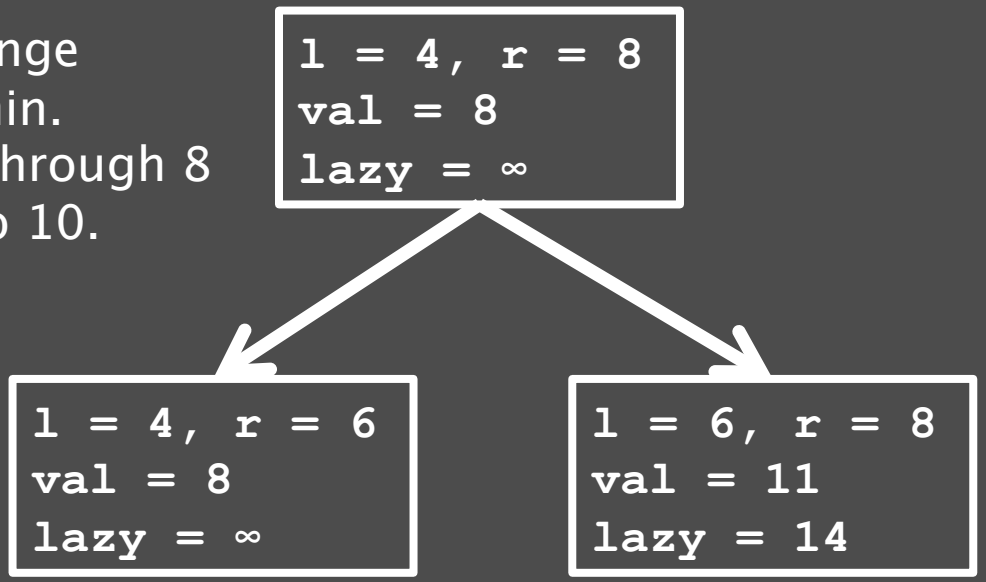
- When is it absolutely necessary to update a node's value using its lazy value?
  - When a node is visited for any reason (update/query)
  - This includes after setting the initial lazy value!

Lazy Propagation

- How do we update a node marked as lazy?
  - Node's values should reflect the answer as if the entire subtree was fully updated
  - Update children's lazy values, if the node has children
  - Method of updating node's values and children's lazy values using the current lazy value depends on the problem
  - Unset lazy values of current node
- We call this method **`push_down()`**

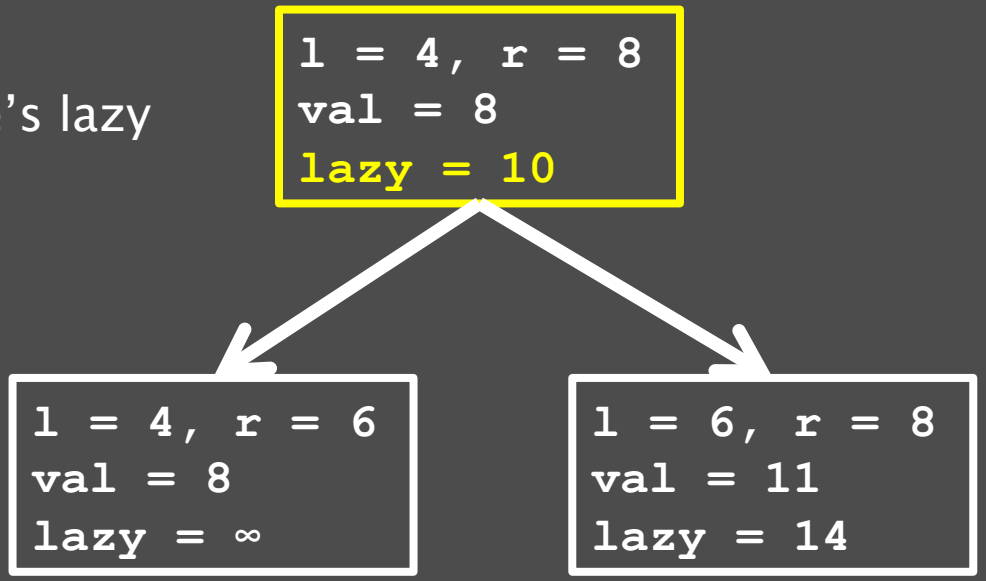
push_down

Example:
Segment tree has range replace and query min.
Update elements 4 through 8 by changing them to 10.



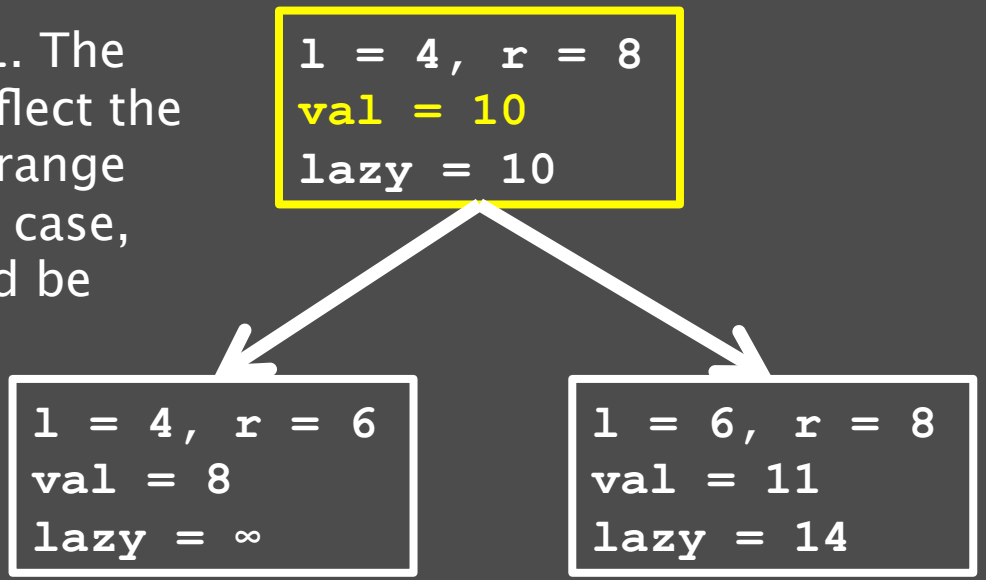
push_down

Update node's lazy value to 10.



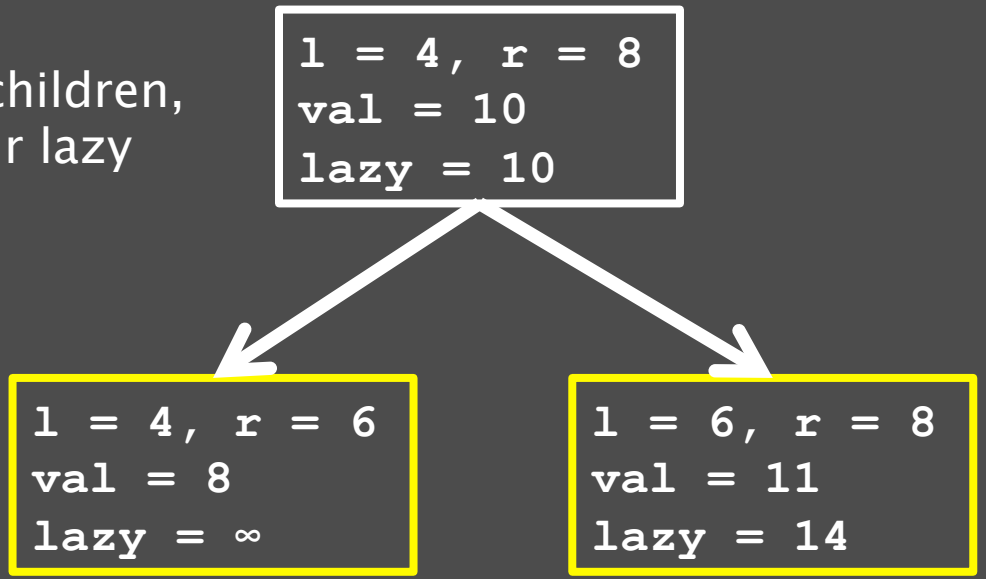
push_down

Begin **push_down** by updating node's **val**. The new value should reflect the answer if the entire range was updated. In this case, every number should be equal to 10.



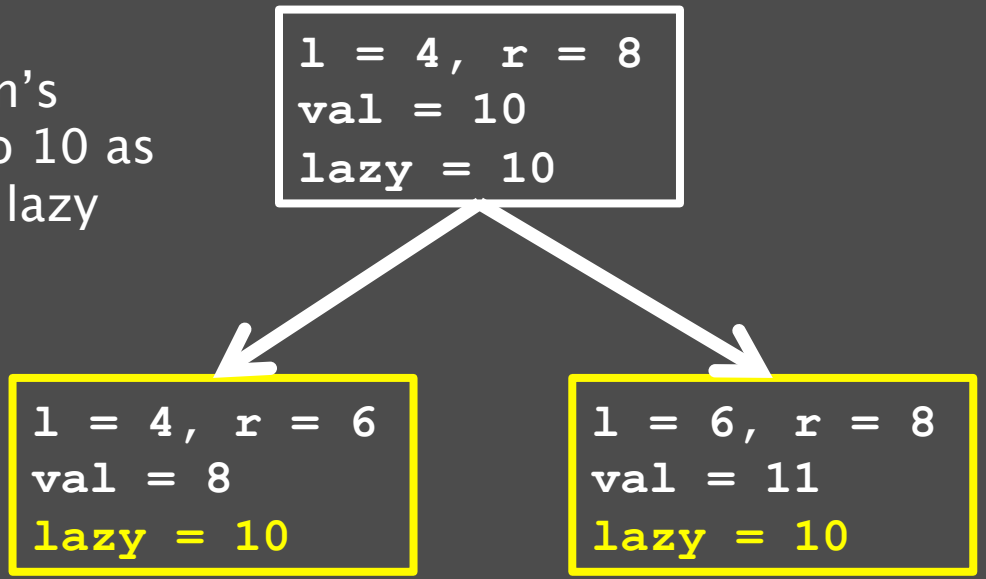
push_down

Since the node has children, we must update their lazy values.



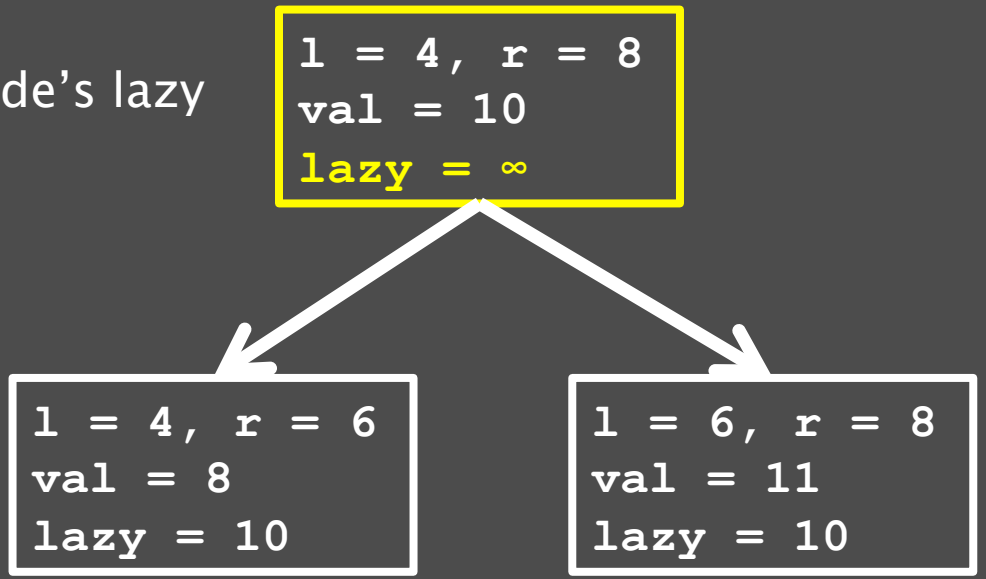
push_down

We want our children's values to be equal to 10 as well, so we set their lazy values.



push_down

Finally, unset the node's lazy value.



`push_down` Functionality Examples

- Replace, min:
  - **`val = lazy`**, replace children lazy
- Add, min:
  - **`val += lazy`**, add to children lazy
- Replace, max:
  - **`val = lazy`**, replace children lazy
- Add, max:
  - **`val += lazy`**, add to children lazy

`push_down` Functionality Examples

- Replace, sum:
  - **`val = lazy*range_size`**, replace children lazy
- Add, sum:
  - **`val += lazy*range_size`**, add to children lazy
- Replace, GCD:
  - **`val = lazy`**, replace children lazy

Lazy Propagation

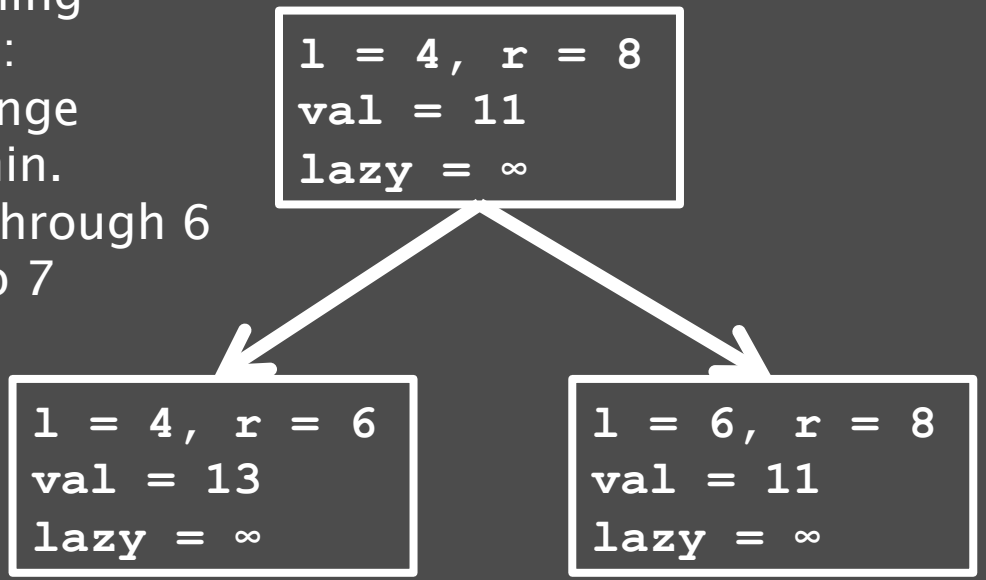
- `push_down()` functionality is not always straightforward
- Example: add value to range, range query GCD
  - Possible to do, but not as simple as the other examples
  - Requires segment tree of a difference array

Lazy Propagation

- Warning: in order for `push_up(nd)` to function as expected, `nd`'s children must be updated
  - Otherwise, `push_up()` will use `nd`'s children's pre-update values
- `push_down` must be called on `nd`'s children
- This is the reason why `push_down` is necessary after setting a node's lazy value – `push_up` will be called on the parent and it will use old values if we did not call `push_down`
- Also an issue if point functions used, since they do not typically visit (and thus `push_down`) both children

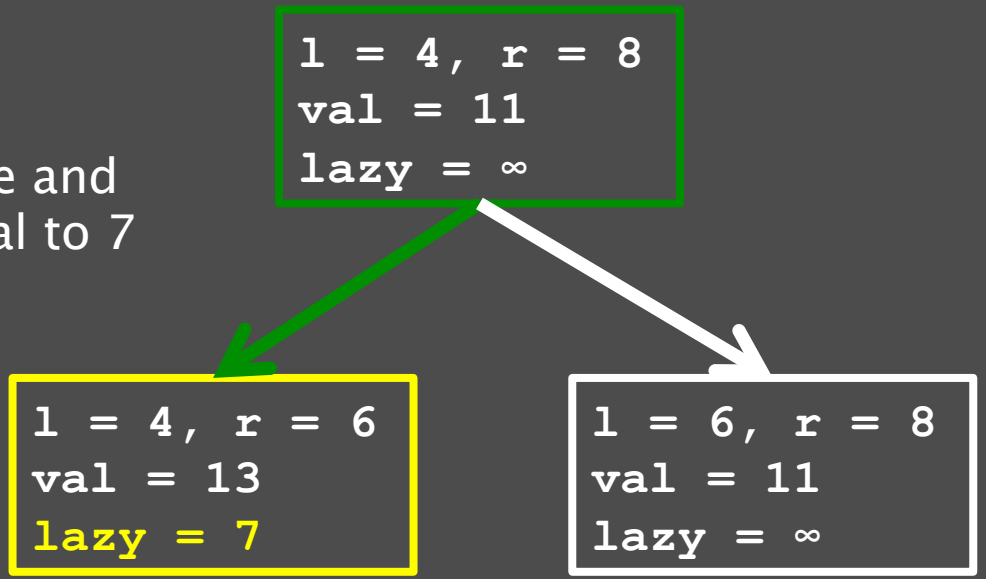
push_up/push_down Issue

Example of not pushing down after updating:
Segment tree has range replace and query min.
Update elements 4 through 6 by changing them to 7



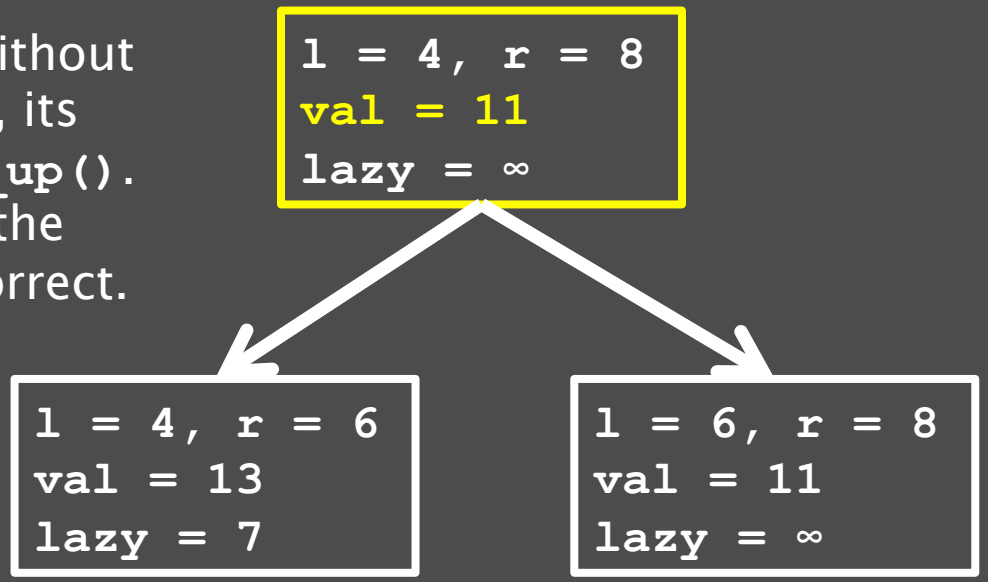
push_up/push_down Issue

Get to the target node and set its lazy value equal to 7



push_up/push_down Issue

If the node returns without calling `push_down()`, its parent will call `push_up()`. Note that the `val` in the parent node is not correct.

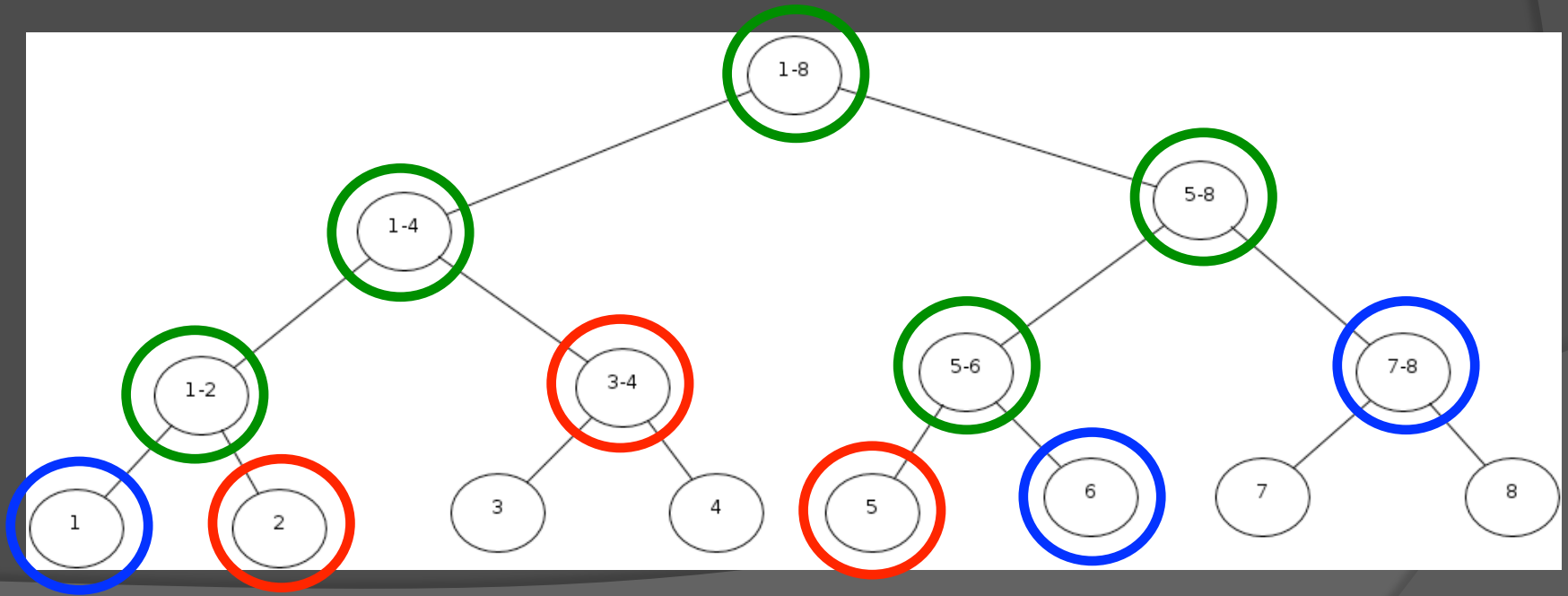


Lazy Propagation

- For each update operation, which nodes should be initially set as lazy?
  - Each of these nodes must be entirely within the update interval
  - Combination of all intervals of nodes chosen must equal the update interval
  - We use the same nodes as if we were doing a range query
  - Like range query, the time complexity must therefore be $O(\log N)$

Range Update

Update: [2,5]
Green nodes specify visited nodes.
Red nodes specify nodes that we mark as lazy
Blue nodes are visited, but ignored



Range Update – Pseudocode (replace, min, elements > 0)

```
void update(int updL, int updR, int val, int idx){
        push_down(idx); //necessary to update the node
        if (tree[idx].l >= updL && tree[idx].r <= updR){
                //interval is entirely within range
                tree[idx].lazy = val; //mark as lazy
                push_down(idx);
                return;
        }
        if (tree[idx].l > updR || tree[idx].r < updL){
                //interval is entire out of range
                return;
        }
        update(updL,updR,val,2*idx); //update left
        update(updL,updR,val,2*idx+1); //update right
        push_up(idx) // update answer
}
```

push_down – Pseudocode (replace, min, elements > 0)

```
void push_down(int idx){
        if (tree[idx].lazy != 0){ //lazy is marked
                tree[idx].val = tree[idx].lazy; //update val
                if (tree[idx].l != tree[idx].r){
                        //update child nodes
                        tree[2*idx].lazy = tree[idx].lazy;
                        tree[2*idx+1].lazy = tree[idx].lazy;
                }
                //unmark lazy
                tree[idx].lazy = 0;
        }
}
```

Query – Pseudocode (replace, min, elements > 0)

```
int query(int queryL, int queryR, int idx){
        push_down(idx); //query is same as before except this line
        if (tree[idx].l >= queryL && tree[idx].r <= queryR){
                //interval is completely within query range
                return tree[idx].val;
        }
        if (tree[idx].l > queryR || tree[idx].r < queryL){
                //interval is completely out of query range
                return INF; //return some answer that will not affect
                //final answer
        }
        int ansL = query(queryL,queryR,2*idx);
        int ansR = query(queryL,queryR,2*idx+1);
        return min(ansL,ansR); //choose the better answer
}
```

THANK YOU!