Algorithms III

Trees

David Esparza Alba

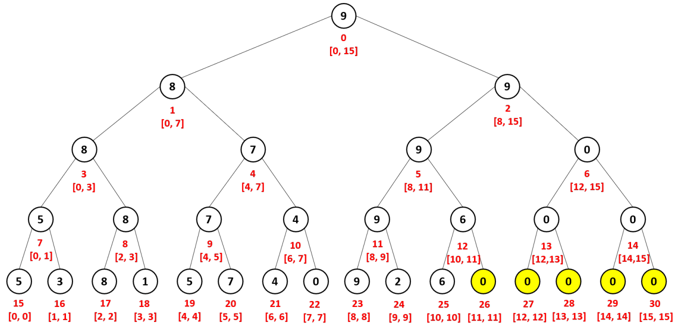
March 19, 2021

Universidad Panamericana

Outline

Segment Tree

Segment Tree - Build



Segment Tree Example

- Find the number of levels in the tree
- Place the original data in the leaf nodes and add dummy nodes to complete the binary tree.
- Compute internal nodes from leaf nodes.
- Points

Segment Tree - Search

1. Check if the query interval intersects with the node's interval, if it does not, then return a value that doesn't affect the result.
2. Update the query interval, e.a. If query interval is [3,9] and node's interval is [0,7], update the query interval to [3,7].
3. If the query interval is equal to the node's interval, then return the maximum value of that segment, otherwise return the greatest value between the maximum of the left sub-tree and the maximum of the right sub-tree.

RMQ

RMQ - Creation

Given an array X of n elements and two positions or indices of the array, the *RMQ* finds the minimum element in X between those two positions.

The *RMQ* builds a table M in $O(n \log n)$ time, and using that table each query can be answered in $*O(1)$ time.

Each row of M represents a starting position in the array, and each column represents a power of two, in such a way that $M_{i,j}$ contains the index of the minimum element between elements $X[i], X[i+1], \ldots, X[i+2^j-1]$.

RMQ - Creation

for $X = [2, 4, 3, 1, 6, 7, 8, 9, 1, 7]$ the table will look as follows:

0	0	3	3
1	2	3	3
2	3	3	3
3	3	3	-
4	4	4	-
5	5	8	-
6	6	8	-
7	8	-	-
8	8	-	-
9	-	-	-

RMQ - Table Creation

```
for (i = 0; i < n; i++) {
  M[i][0] = i;
}

for (j = 1; 1 << j <= n; j++) {
  for (i = 0; i + (1 << j) - 1 < n; i++) {
    if (X[M[i][j - 1]] < X[M[i + (1 << (j - 1))][j - 1]]) {
      M[i][j] = M[i][j - 1];
    } else {
      M[i][j] = M[i + (1 << (j - 1))][j - 1];
    }
  }
}
```

RMQ - Queries

RMQ (Answer a Query)

```
ans = 0;
k = (long)floor(log(double(j - i + 1)) / log(2.0));
if (X[M[i][k]] <= X[M[j - (1 << k) + 1][k]]) {
    ans = M[i][k];
} else {
    ans = M[j - (1 << k) + 1][k];
}
printf("%d\n", ans);
```

RMQ - Queries

The time complexity depends on the log function. O(1)?

BIT

BIT - What it is?

BIT allows us to solve the problem of the sum of an interval when this is not static.

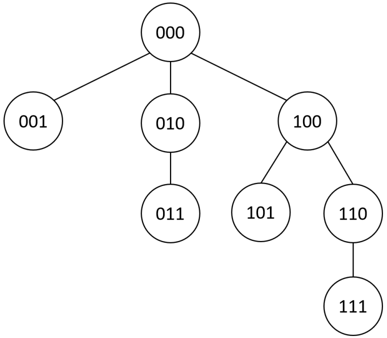
The idea behind BIT consists in storing the accumulates from index 0 to index i using a binary representation of the indexes.

BIT - How it works?

x=10101000	x represented as eight-based bit
-x=01011000	Negative x represented as Two’s-Complement
x&(-x)=	10101000
	&01011000
	00001000

Bitwise operation to get the least significant bit

BIT - Tree



BIT - Creation and Updates

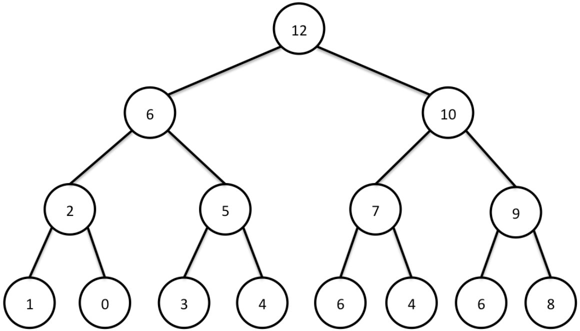
```
void updateBIT(int i, int v) {
  while (i <= N) {
    T[i] += v;
    i += (i & -i);
  }
}

void createBIT() {
  for (int i = 1; i <= N; i++) {
    updateBIT(i, X[i]);
  }
}
```

BIT - Queries

```
int queryBIT(int i) {
  int res = 0;
  while (i > 0) {
    res += T[i];
    i -= (i & -i);
  }
  return res;
}
```

Heap

Heap - What is it?



Valid Heap

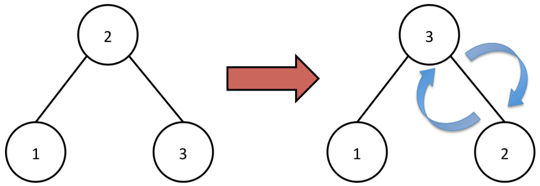
A heap is binary tree where the value associated to node k is larger or equal than the value associated to its children.

Heap - How to build it?

There can be multiple valid heaps for the same data

1. As we read numbers we can place the elements in the heap in a valid position
2. We can store the values in a tree structure (no heap), and then starting from the leafs to the root place the elements in the correct position until we have a heap.

Heap - How to build it?



Valid Heap

To place an element in the correct position just swap it with the largest of its children, keep doing that until the children are smaller or we reach a leaf node.

Heap - What it is used for?

1. The root represents the largest element, so heaps are frequently used to retrieve the largest or smallest (min-heap) element of some given data.
2. Retrieving the largest/smallest element takes $O(1)$.
3. Heaps are not designed to search elements, their main advantage is to get the maximum/minumum in $O(1)$.
4. Heaps works pretty good to find the k largest/smallest elements of some data.

Heap - Applications

1. Heap sort.
2. Get the median of some data that is being updated frequently.
3. Find the k largest/smallest elements. Whenever you hear “Find the k largest... ” your heap sensor should beep.