Segment Tree

Matheus Artur, Luís Alberto Cabús, Nicolas Leão, Fábio Vinícius

https://github.com/projetosufal/data-structures-project

Outline

Matheus Artur, Luís Alberto Cabús, Nicolas Leão, Fábio Vinícius

Stock Exchange

The problem

- Data from thousands of companies worldwide, active for decades
- Usage of multiple operations requiring a range/interval of data in any capacity
- Recources are not inifinite, must be optimized as best as possible

O(n) operations? No!

Segment tree

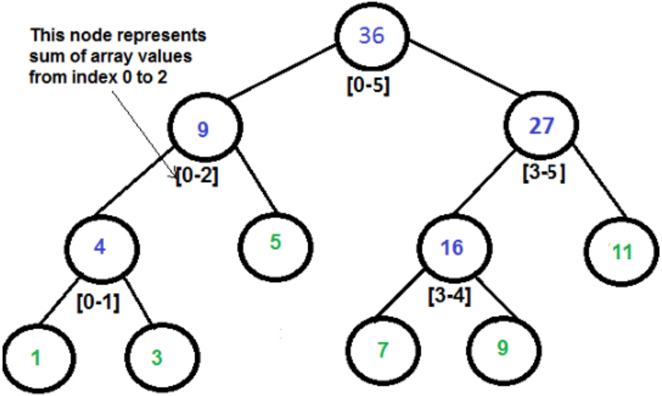
How it is structured?

- The Segtree is a binary tree that's represented from a array, where each node represents a unique interval or segment of the tree and stores a specific value.
- The value, is usually represented by *maximum*, *minimum* or *sum* of the segment.

Intervals of a A[6]

tree[0]	=	A[0:5]
tree[1]	=	A[0:2]
tree[2]	=	A[3:5]
tree[3]	=	A[0:1]
tree[4]	=	A[2:2]
tree[5]	=	A[3:4]
tree[6]	=	A[5:5]
tree[7]	=	A[0:0]
tree[8]	=	A[1:1]
tree[9]	=	NULL
tree[10]	=	NULL
tree[11]	=	A[3:3]
tree[12]	=	A[4:4]

As tree



Segment Tree for input array {1, 3, 5, 7, 9,11}

Operations

Building a tree

- (n log n) storage
- but only (2*n - 1) actual nodes

Query - range search

- O(log n)

Updating a tree

- O(log n)
- can modify any [l:r] section, than it will propagate updating dependencies

Building a Segtree

```
void
buildtree(int (*f)(int l_num, int r_num), int *v, int *tree,
  int *t_size, int node, int min, int max)
{
  int mid;

  if(min == max)
    tree[node] = v[min];

  else
    {
      mid = (min+max)/2;

      buildtree((*f), v, tree, t_size, 2*node + 1, min , mid);
      buildtree((*f), v, tree, t_size, 2*node + 2, mid + 1 , max);
      tree[node] = (*f)(tree[2*node +1], tree[2*node + 2]);
    }
}
```

Matheus Artur, Luís Alberto Cabús, Nicolas Leão, Fábio Vinícius

Query

```
int
query(int (*f)(int l_num, int r_num), int *tree,
      int node, int min, int max, int l, int r)
{
  if(r < min || max < l)
    return 0;

  if(l <= min && max <= r)
    return tree[node];

  int mid, l_bipod, r_bipod;
  mid = (min+max)/2;

  l_bipod = query((*f), tree, 2*node + 1, min , mid, l , r);
  r_bipod = query((*f), tree, 2*node + 2, mid + 1 , max, l, r);
  return((*f)(l_bipod, r_bipod));
}
```

Update

```
void
updatetree(int (*f)(int l_num, int r_num), int *tree,
   int node, int min, int max, int l, int r, int val)
{
  int mid;
  if(min > max || min > r || max < l)
    return ;

  if(min == max)
    {
      tree[node] = val;
      return;
    }

  mid = (min+max)/2;

  updatetree((*f), tree, 2*node + 1, min , mid, l, r, val);
  updatetree((*f), tree, 2*node + 2, mid + 1 , max, l, r, val);

  tree[node] = (*f)(tree[2*node +1], tree[2*node + 2]);
}
```

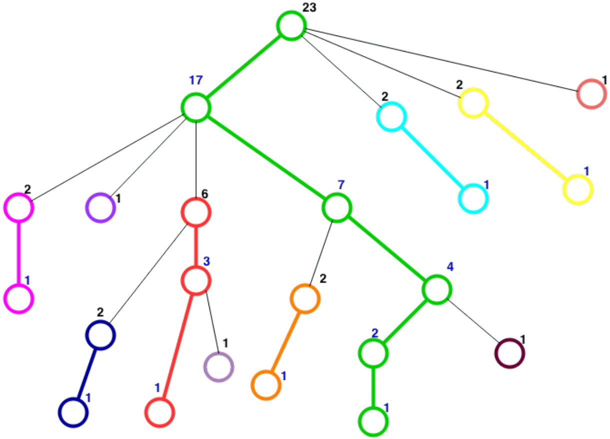
Segtree as a base for advanced algorithms

As seen, segtree is a flexible algorithm, such power can be used as a base for even more advanced algorithms, such as Heavy Light Decomposition

- HLD takes segtree to another new level, transforming linear paths of the graph into multiple segtrees

Matheus Artur, Luís Alberto Cabús, Nicolas Leão, Fábio Vinícius

Heavy Light Decomposition - a Segtree in graph



Each Chain is represented with different color.
Thin Black lines represent the connecting edges. They connect 2 chains.

Conclusion

the algorithm is better suited to solve problems of:

- analysis of big chunks of data
- reorganization and update of said data

And these properties are highly applicable in fields of stock market, weather analysis and competitive programming alike