Coding the Virtual DOM

...

Using Data Structures & Algorithms

Chapter 1

Introducing Data Structures & Algorithms

Data Structures & Algorithms

The study of how to leverage, optimally, CRUD operations as a means to solve a problem, or algorithms.

It’s in code, so it must be an algorithm

Wrong.

An algorithm outlines a sequence of instructions needed to derive a solution to a given problem and...

Let’s take a look...

```
function ten() {
    let a = 5;
    let b = 5;
    let c = a + b;
    return c;
}
```

Finite: 5 steps to ten

```
1.  function ten()
2.      let a = 5;
3.      let b = 5;
4.      let c = a + b;
5.      return c;
```

Inputs: 5 and 5 makes ten

```
1.  function ten()
2.      let a = 5;
3.      let b = 5;
4.      let c = a + b;
5.      return c;
```

Output: ten

```
1.  function ten()
2.      let a = 5;
3.      let b = 5;
4.      let c = a + b;
5.      return c;
```

Feasible and independent: JS, C++, C...?

```
1.  function ten()
2.      let a = 5;
3.      let b = 5;
4.      let c = a + b;
5.      return c;
```

That said...

1. Finite
2. Inputs
3. Outputs
4. Feasible
5. Independent
6. ?

No. 6

An algorithm is one of potentially many approaches to solving a given problem.

No. 6 makes ten, also...and so does No. 6...and so on...

```
function ten() {
    let c = 5 + 5;
    return c;
}
```

Why “should” No. 6 be important?

For starters...performance.

Two metrics to quantify the performance of an algorithm, namely time and memory – also known as the complexities of an an algorithm.

Time complexity

How long the algorithm will take to derive the solution:

- data structure
- instructions – loops, arithmetic operations, comparisons

Memory complexity

How much memory would the algorithm need to execute:

- variables and what goes inside – local or external inputs

JS data structures

Primitive

- number, string, boolean, null, undefined, symbol

Non-primitive

- Array and RegExp

Objects

Why objects?

Objects allow storage of varying data structures by use of references, or keys.

The DOM structure is fundamentally designed with objects in mind.

Chapter 2

Introducing the DOM

What is the Document Object Model?

Encapsulates data using an object-oriented approach to describing the structure of a web page and provide an interface, or API, for you to CRUD state information.

What does DSA have to do with the DOM?

The DOM API implements complex algorithms, using an understanding of DSA to create a performant interface for developers to work with.

Understanding the complexities

Fundamentally, CRUD operations require us to first find what we're looking for and then, to either create, read, update or delete data at the given location. But again, there are many ways to find what we're looking for...

Chapter 3

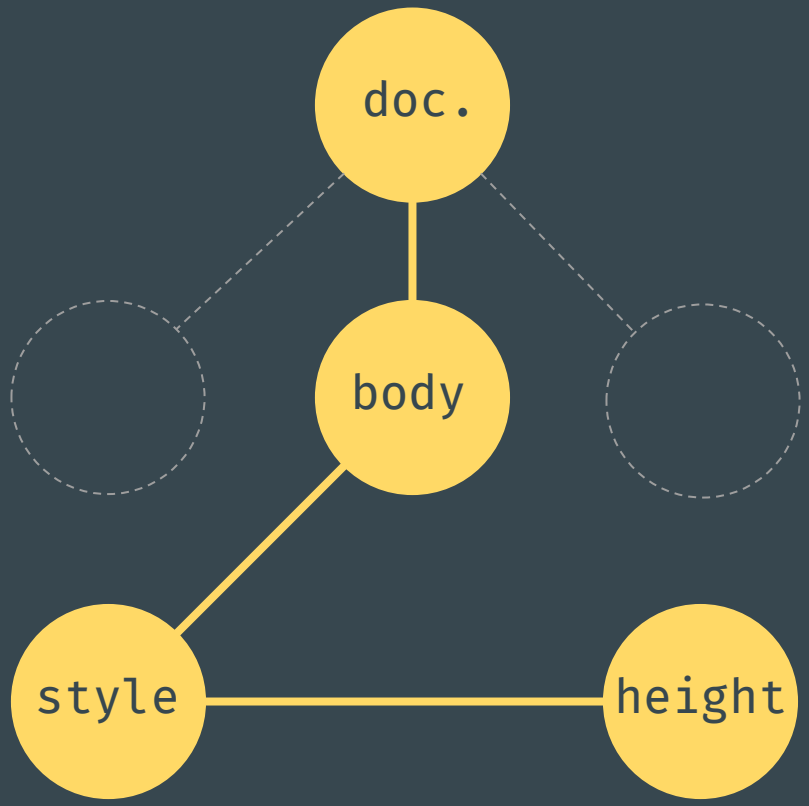
Traversing the DOM

Interacting with the DOM

What is the path to the height inline style of the body?

```
<body style="height:35px">...</body>
```

```
document.
```

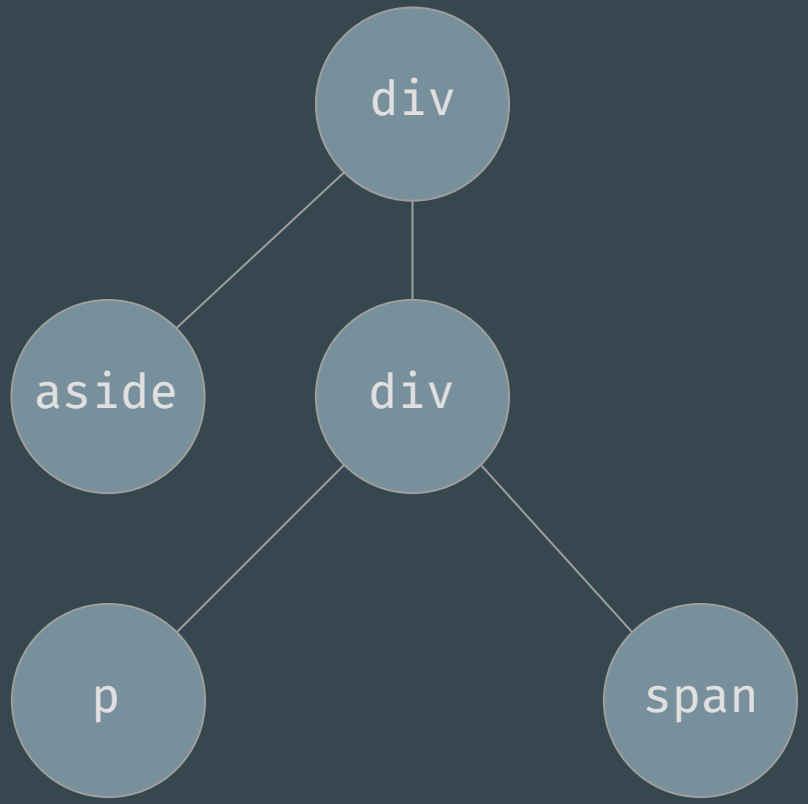


Interacting with the DOM

What is the path to the height inline style of the body?

```
<body style="height:35px">...</body>
```

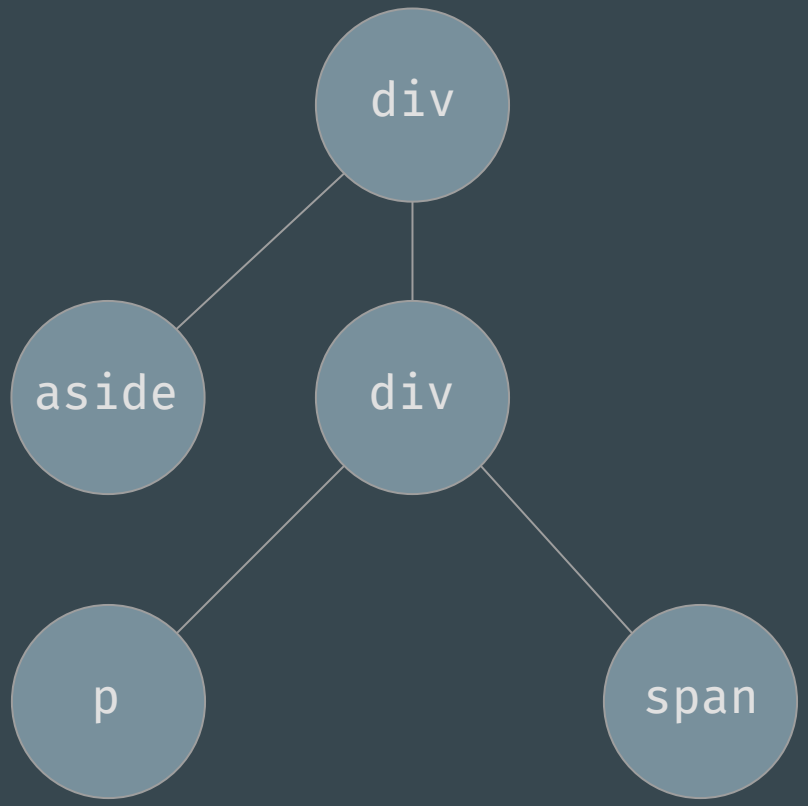
```
document.body.style.height
```



Designing a search algorithm

What is the path to p?

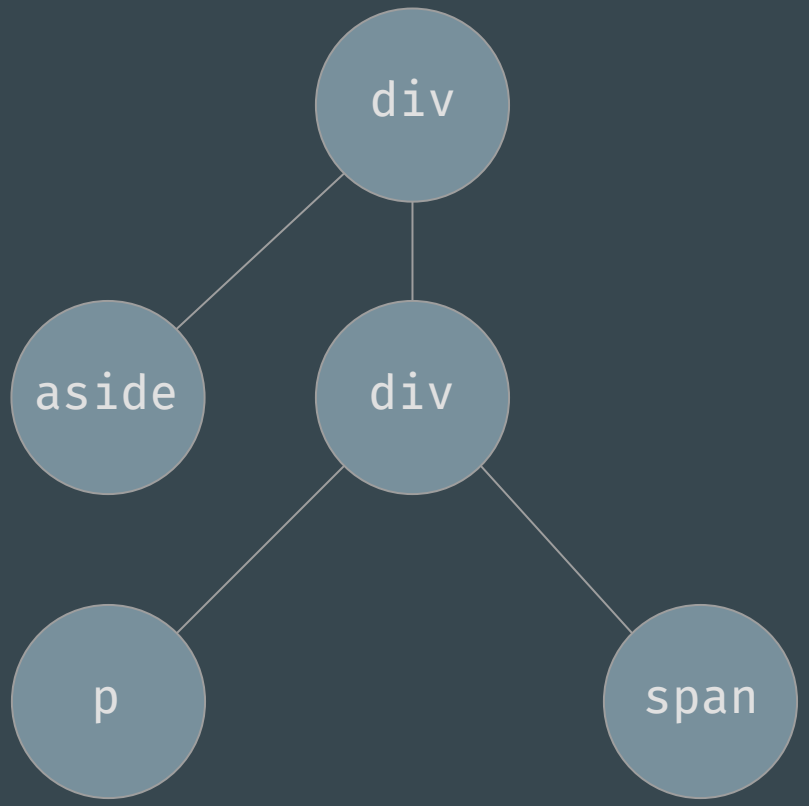
```
<div>
    <aside>5</aside>
    <div>
        <p>[ 1, 2 ]</p>
        <span>10</span>
    </div>
</div>
```



Designing a search algorithm

What is the path to p?

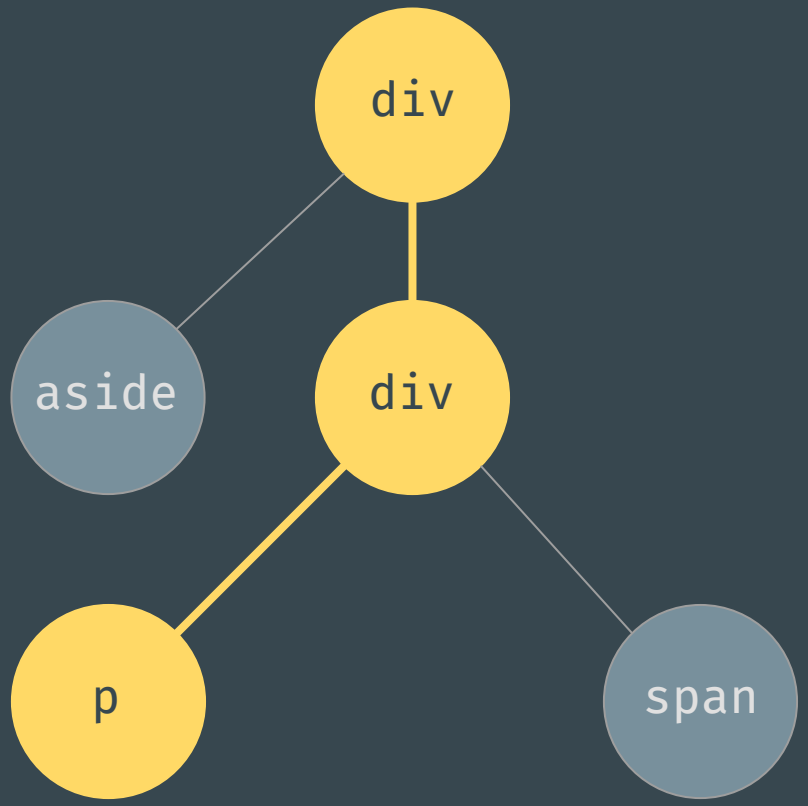
```
let div = {
    aside: 5,
    div: {
        p: [ 1, 2 ],
        span: 10
    }
}
```



Designing a search algorithm

Design an algorithm to parse the path to p and return it's value?

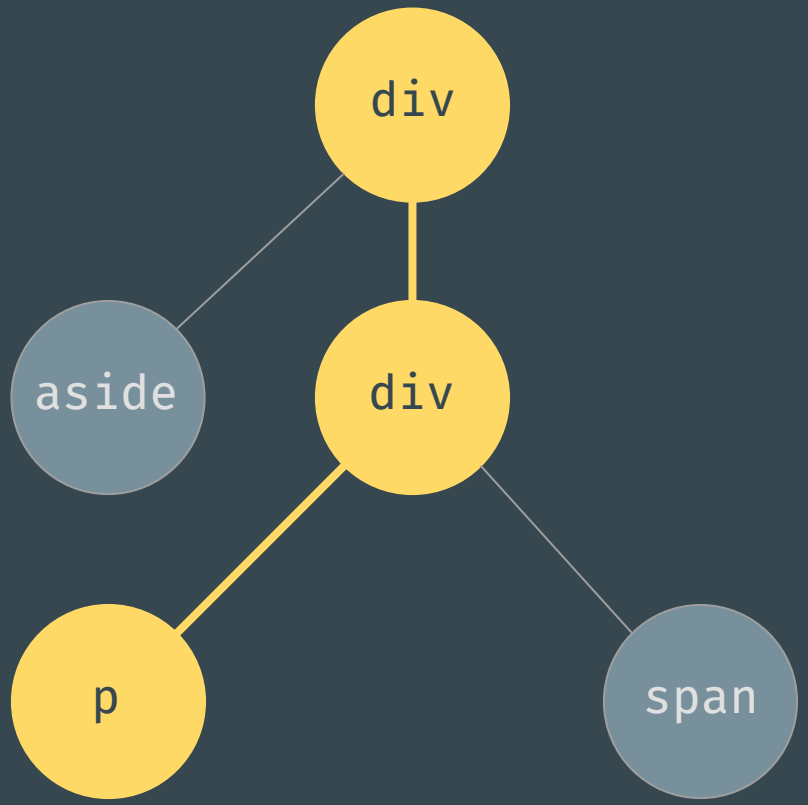
```
let div = {
    aside: 5,
    div: {
        p: [ 1, 2 ],
        span: 10
...
```



Designing a search algorithm

Design an algorithm to parse the path to d and return it's value?

```
let div = {
    aside: 5,
    div: {
        p: [ 1, 2 ],
        span: 10
...
```



Exploring trees

All are nodes

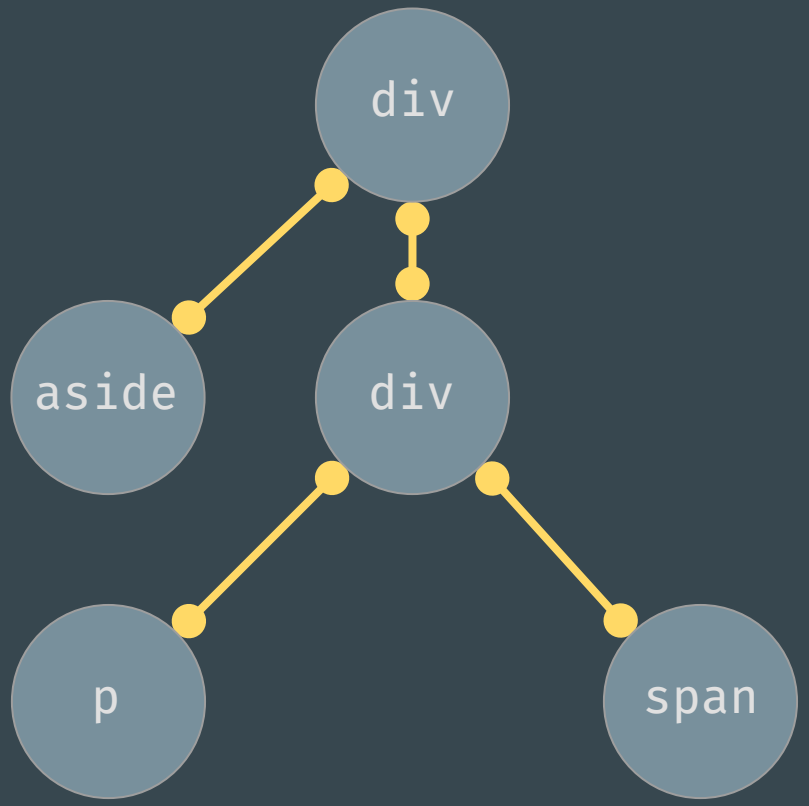
- div - root node
- aside, p and span - leaf nodes
- div, div and p - path
- p - goal

Ways of traversing trees

Depth-first search

Breadth-first search

There are many more....

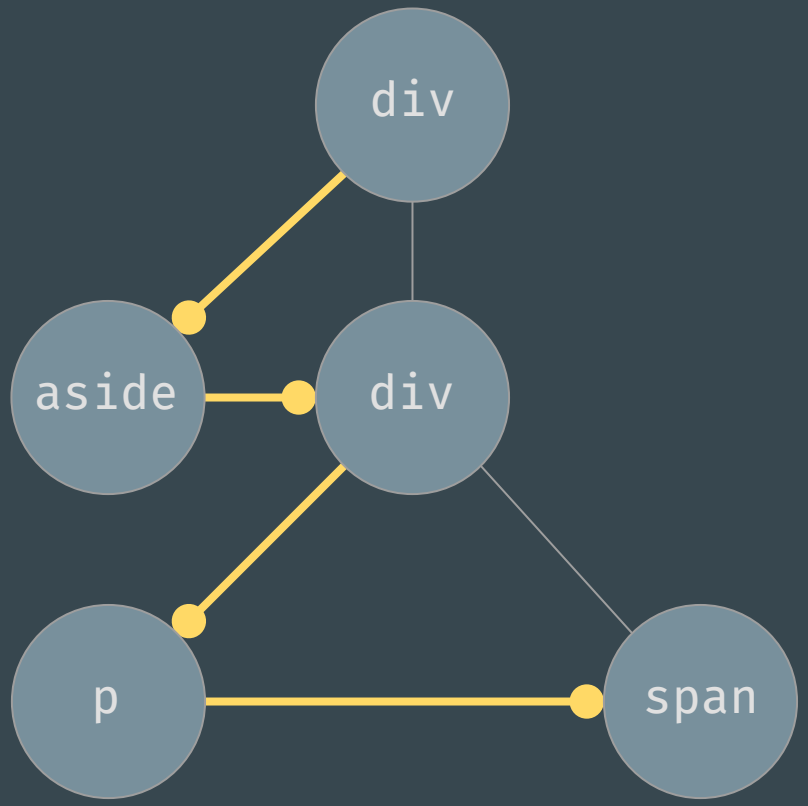


Depth-first search

Takes a top-down-horizontal approach.

Works best when the tree is shallow and the goal is as far left as possible.

Design an algorithm to find d using DFS.



Breadth-first search

Takes a top-horizontal-down approach.

Works best when breadth is small e.g. binary search tree, at most two child nodes per node.

Design an algorithm to find d using BFS.

Chapter 4 - coming up next

Building the DOM API