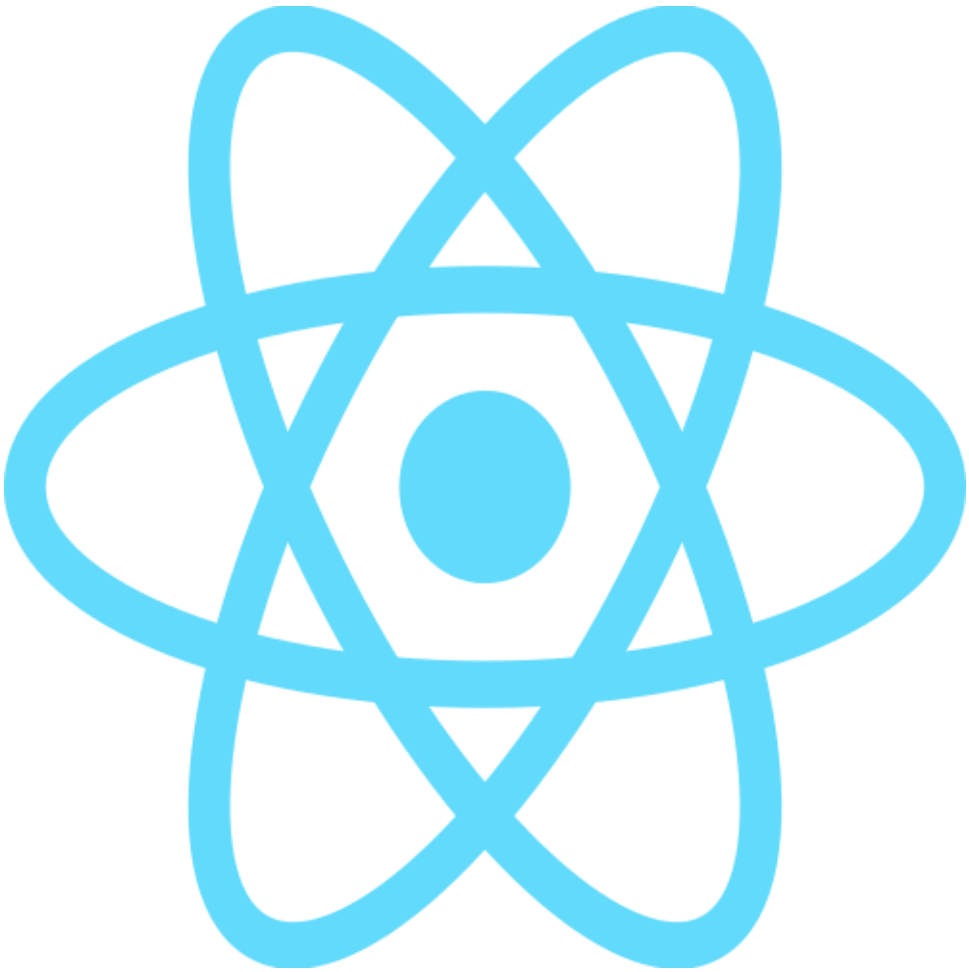
ABDELLANI Mohamed

Follow 10 Followers About

How does the diff algorithm work in React JS?

A ABDELLANI Mohamed Oct 9, 2019 · 3 min read



React JS logo, source: https://reactjs.org/

Introduction

React is a JavaScript frontend framework created by Facebook. It helps developers to write more organized code to build user interfaces, by letting the use of a descriptive language (JSX) and supporting the concept of components and composition.

React is also fast, and the reason behind its performance is the use of virtual DOMs. When the user interacts with the UI, React tries to figure out the most efficient way to

update the UI. In this article, we'll get an idea about how things work inside.

Virtual DOM :

Based on the application code, React builds a tree of element that describes how the written component should be rendered. The nodes of this tree are stored as plain objects called elements.

After a user interaction that results in a state or props changes, React will generate a new tree of element.

Reconciliation:

Before updating the user interface, React uses a reconciliation algorithm to compare the new tree with the most recent tree to find out the most efficient way to update the user interface. The user interface is not necessary a browser interface but can be android/IOS application (React native or React IOS).

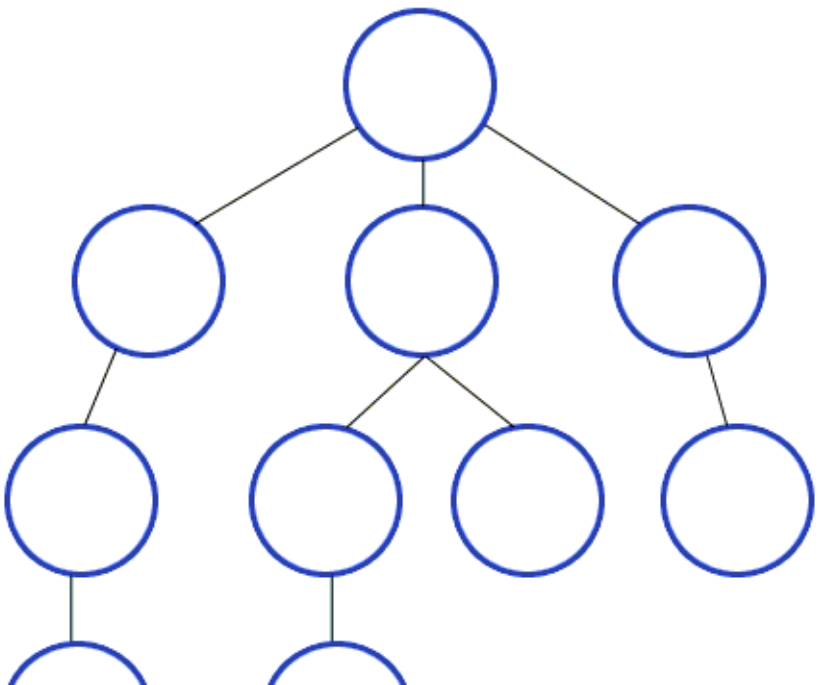
The problem at this level is that the latest algorithms for solving this kind of problems have a complexity of $O(n^3)$.

React uses heuristics to reduce the time of processing and to solve the problem in a linear time $O(n)$.

From the official documentation, there are two assumptions :

1- Different elements will produce different trees.

React parses the tree using Breadth-first search (BFS). For a node of tree, if the element type is changed, for example from 'section' to 'div'. React will destroy all the sub-tree under that element and will reconstruct it from scratch.



BFS, source: https://codeaccepted.wordpress.com/2014/04/08/depth-and-breadth-first-search/

2- The developer can hint at which child elements may be stable across different renders with a key prop.

This means by adding keys to children, React will be able to track changes. For example, given a component that contains the following code:

```
<ul>

<li>item 1</li>

<li>item 2</li>

</ul>
```

if in the future, a new element is inserted as follow :

```
<ul>

<li>item 0</li>

<li>item 1</li>

<li>item 2</li>

</ul>
```

React will not be able to figure out that the two last elements are the same as in the original list, and will update the interface in an inefficient way. This problem can be solved by adding keys to every child element as follow:

```
<ul>

<li key='1'>item 1</li>

<li key='2'>item 2</li>

</ul>
```

Conclusion:

This is just a brief summary of the React internals. if you want to have a more in-depth view of the subject, please consult the last two links in the references.

References :

https://reactjs.org/blog/2015/12/18/react-components-elements-and-instances.html

https://reactjs.org/docs/reconciliation.html

https://github.com/acdlite/react-fiber-architecture

https://indepth.dev/inside-fiber-in-depth-overview-of-the-new-reconciliation-algorithm-in-react/

JavaScript React Reconciliation Reactjs Virtual Dom