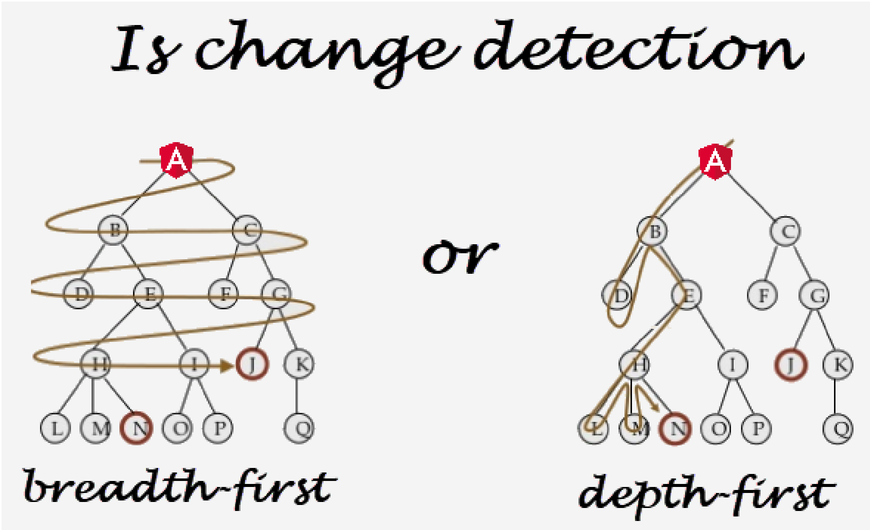
He who thinks change detection is depth-first and he who thinks it's breadth-first are both usually right

Max Koretskyi aka Wizard Follow
Dec 19, 2017 · 4 min read

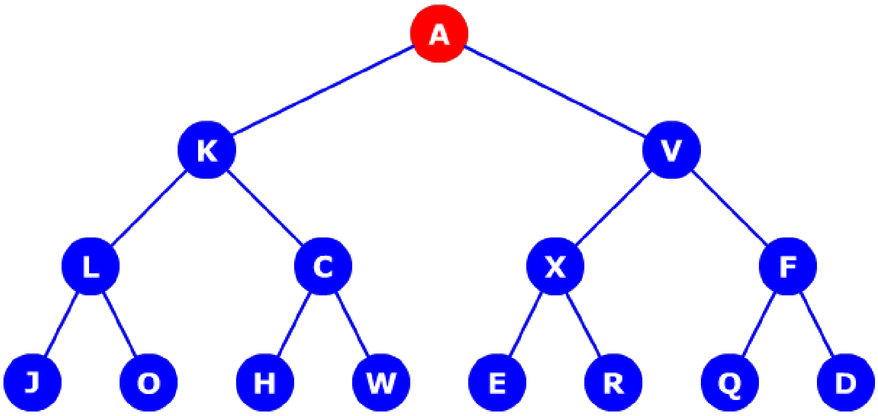


· · ·

I was once asked if change detection in Angular is depth or breadth first. This basically means whether Angular first checks siblings of the current component (breadth-first) or its children (depth-first). I hadn't given any prior thought to this question so I just went with my gut and the knowledge of internals. I declared that it was depth-first. Later, to check my assertion, I created a tree of components and put some logging logic inside the `ngDoCheck` hook:

```
@Component({
    selector: 'r-comp',
    template: `{{addRender()}}`
})
export class RComponent {
    ngDoCheck() {
        // holds all calls in order and is logged to console
        calls.ngDoCheck.push('R');
    }
```

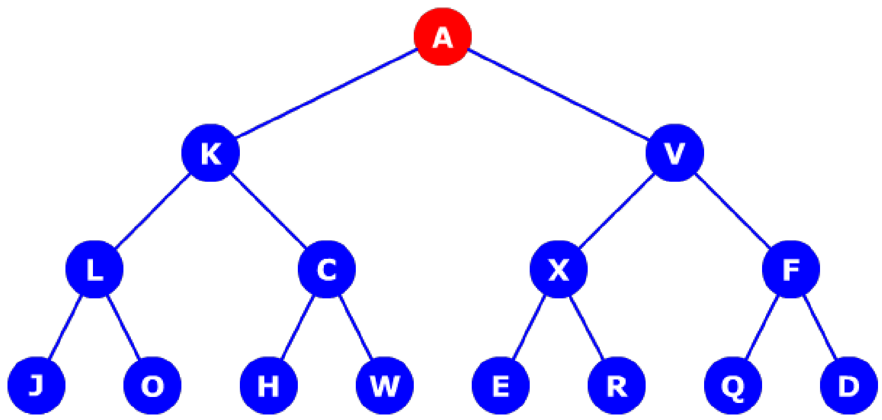
And to my surprise, it turned out that some siblings were checked first as depicted on the diagram below:



So here you see that Angular checks `K` and then `V`, `L` and then `C` and so on. So was I wrong and it's really a breadth-first algorithm? Well, not exactly. First thing to notice in the above representation is that it's not a proper breadth-first algorithm. The conventional implementation of the algorithm checks **all siblings on the same level**, whereas in the diagram above as you can see the algorithm indeed checks `L` and `C` sibling components, but instead of checking `X` and `F` it goes down to `J` and `O`. Also, the implementation of the breadth-first graph traversal algorithm is well defined but I couldn't find it in the sources. So I decided to run another experiment and put a logging logic in a custom function called when change detection evaluates expressions:

```
@Component({
    selector: 'r-comp',
    template: `{{addRender()}}`
})
export class RComponent {
    addRender() {
        calls.render.push('R');
    }
}
```

And this time I got different results:



It's a proper depth-first graph traversal algorithm. So what's going on here? It's actually pretty simple, let's see.

> I work as a developer advocate at **ag-Grid**. If you're curious to learn about data grids or looking for the ultimate Angular data grid solution, give it a try with the guide "**Get started with Angular grid in 5 minutes**". I'm happy to answer any questions you may have. **And follow me to stay tuned!**

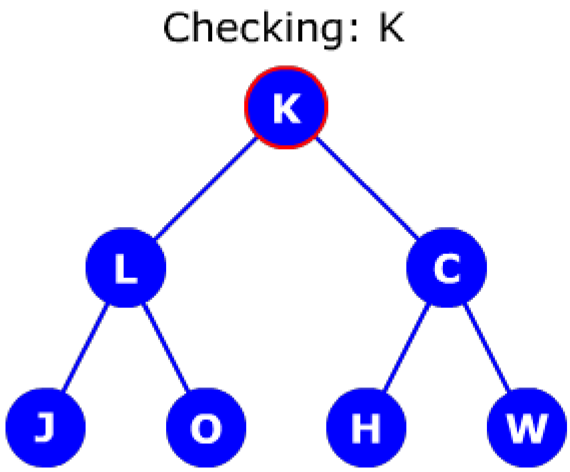
. . .

Change detection operations

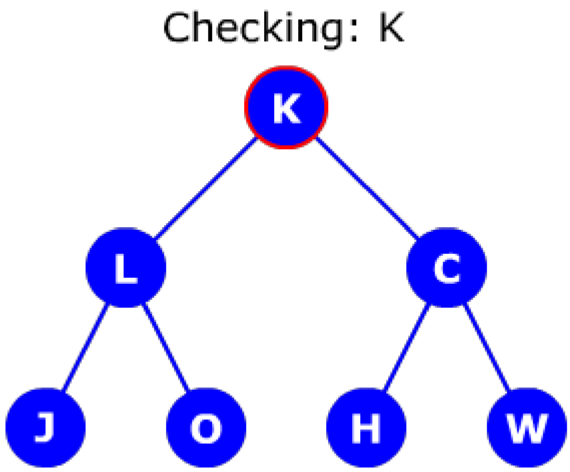
To understand the difference in behavior we need to take a look at the operations performed by change detection mechanism when checking a component. If you've read my other articles on change detection you probably know that the key operations performed by change detection are the following:

- update child components properties
- call `NgOnChanges` and `NgDoCheck` lifecycle hooks **on child components**
- update DOM **on the current component**
- **run change detection for child components**

I highlighted one interesting specifics above—when Angular checks the current component it calls lifecycle hooks **on child components**, but renders DOM for **the current component**. And that's a very important distinction. This is precisely the reason that makes it seem as if the algorithm runs breadth-first if we put logging into NgDoCheck hook. When Angular checks a current component it calls lifecycle hooks for all its child components which are siblings. Suppose Angular checks `K` component now and calls `NgDoCheck` lifecycle hook on `L` and `C`. So, we get the following:



Looks like breadth-first algorithm. However, remember that Angular still in the process of checking `K` component. So after completing all operations for the `K` component it doesn't proceed to checking the sibling `V` component, as it would with the breadth-first implementation. Instead, it goes on to check `L` component, which is a child of `K`. This is the depth-first implementation of change detection algorithm. And as we now know it will call `ngDoCheck` on `J` and `O` components and this is exactly what happens:



So, after all, my gut didn't let me down. Change detection mechanism is implemented as depth-first internally, but involves calling `ngDoCheck` lifecycle hooks on sibling components first. By the way, I already described this logic in depth in the If you think `ngDoCheck` means your component is being checked—read this article.

. . .

Stackblitz demo

Here you can see the demo with logging logic in different places.

· · ·

Thanks for reading! If you liked this article, hit that clap button below 👏. It means a lot to me and it helps other people see the story.

For more insights follow me on Twitter and on Medium.