Verslag Project Algoritmen en Datastructuren

Verslag door Dries Huybens, informatica ba2 2022-2023

Inleiding

In dit project hebben we een normale 2-3 boom, een 2-3 boom met semi-splay die via de Bottom-Up manier werkt en een 2-3 boom met semi-splay die via de Top-Down manier werkt moeten maken.

In dit verslag ga ik bespreken welke van de drie het efficientst is per scenario, we zullen het toevoegen, verwijderen en opzoeken van een element bespreken. We gaan dit zowel doen op een normale manier waarop de elementen steeds in dezelfde volgorde verwijderd worden als ze werden toegevoegd, en wanneer ze op een willekeurige andere volgorde worden verwijderd.

Implementatie

2-3 Boom

Mijn vervangenbomen van mijn normale 2-3 boom zijn hieronder te zien.

//afbeelding invoegen 2-3 bomen

Top-down- & Bottom-up Semi-Splay 2-3 Boom

Vervang bomen

De vervangenbomen die ik voor mijn semi splay gebruikt heb zijn te zien in de onderstaande afbeeldingen. Ik heb zoveel mogelijk geprobeerd op toppen te splitsen en niet op sleutels. Ik verwacht dat op sleutels splitsen efficienter was geweest aangezien je dan meer toppen zou samenvoegen en dus effiecienter gebruikt van de 2-3 eigenschappen maakt, maar dit leken mij al heel snel heel veel situaties die ik ging moeten coderen.

//afbeelding invoegen splay bomen

Bottom-up Semi-Splay 2-3 Boom

De implementatie voor mijn Bottom-up boom werkt als volgt, ik geef een node mee aan mijn splay methode die dan vervolgens steeds drie nodes gaat samennemen en door een vervangboom gaat vergangen. Hierna splay ik nog eens op de node daarboven zodat die ook door een vervangboom vervangen word. Dit blijf ik doen tot ik aan een kind van de wortel of wortel zit.

Top-down Semi-Splay 2-3 Boom

Mijn Top-down boom begint bij de wortel en geef ik een waarde mee die ik zoek. Hij gaat steeds een vervangboom maken van de postities waar het mogelijk zou kunnen zijn waar de gezochte waarde zit. Hij gaat steeds zo een stap naar beneden tot hij bij de gezohte waarde terechtkomt. Ik zoek eerst mijn node op waar ik de waarde aan zou toevoegen of uit zou verwijderen. Het zou efficienter zijn als ik terwijl ik splay die

waarde uiteindelijk toevoeg/verwijder. Dit zou er voor zorgen dat ik een keer minder het pad zou moeten doorlopen.

Benchmarks

Het toevoegen van elementen

Het testen van het toevoegen van elementen heb ik gedaan door 50 maal een boom op te bouwen met eerst 10.000 elementen dan 20.000, 30.000...50 000 elementen.

Zoals u hieronder op de grafiek kan zien is de gewone 2-3 boom veruit het efficienst. Dit komt natuurlijk doordat we hier niet moeten splayen waar uitendelijk steeds wel een aantal operaties voor nodig zijn. Het is ook opmerkelijk dat Top-down net iets trager is dan Bottom-up, dit komt door het feit dat ik eigenlijk twee keer zoek naar in node in mijn Top-down implementatie.

 grafiek betreffende toevoeg-operatie

Het verwijderen van elementen

Het testen van het toevoegen van elementen heb ik weer gedaan door 50 maal een boom op te bouwen met eerst 10.000 elementen dan 20.000, 30.000...50 000 elementen. Deze keer gaan we ook vergelijken of er een verschil is als we de elementen in dezelfde volgorde verwijderen als we ze hebben toegevoegd.

Net zoals bij de toevoeg operatie is de 2-3 boom opnieuw het efficientst. Dit komt alweer door dezelfde reden, er worden geen splay operaties uitgevoerd. De Bottom-Up manier is net weer iets meer performant dan de Top-Down manier. We zien wel geen verschil in verhouding als we verwijderen in dezelfde volgorde als we de elementen hebben toegevoegd tegenover ze verwijderen in een willekeurige volgorde.

![grafiek betreffende normale-verwijder-operatie](/Users/drieshuybens/ad2_project/extra/bijlagen verslag/verwijder(normaal).png);

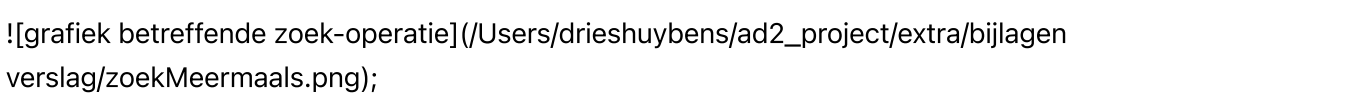
![grafiek betreffende zipf-verwijder-operatie](/Users/drieshuybens/ad2_project/extra/bijlagen verslag/verwijder(zipf).png);

Het zoeken van elementen

Het testen van het zoeken is op exact dezelfde manier gebeurt als de testen hiervoor. Er was geen verschil tussen de verhoudingen van snelheid als we in willekeurige volgorde zochten of in dezelfde manier als we de elementen toegevoegd hadden. Top-down is weeral net iets trager dan Bottom-up door het feit dat we hier alweer een extra zoek operatie uitvoeren. Het verhaal blijft ook weer hetzelfde met het feit dat de 2-3 Boom het meest performant is dankzij het feit dat deze boom geen splay operaties moet uitvoeren.

![grafiek betreffende zoek-operatie](/Users/drieshuybens/ad2_project/extra/bijlagen verslag/zoek.png);

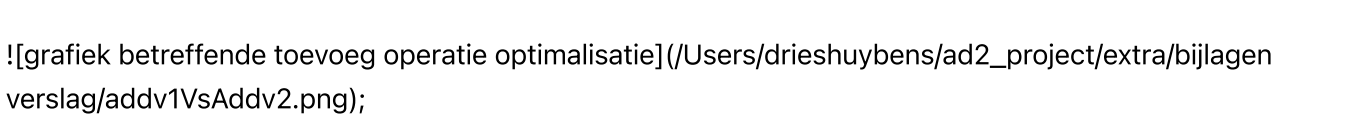
Ik wou ook graag nog eens zien of deze semi-splay methode effectief efficienter zou worden als we hetzelfde elementen meerdere keren achter elkaar zouden opzoeken. In deze test heb ik 100 keer na elkaar hetzelfde element opgezocht en hier blinkt de efficientie van de Semi-Splay toch echt uit, vooral op de Bottom-up implementatie die geen extra zoekoperatie moet doen zoals de Top-down implementatie. De Bottom-up is hier toch wel wat sneller dan de normale 2-3 Boom terwijl de Top-down implementatie ongeveer gelijk staat met de normale 2-3 Boom.



Optimalisatie ideen

Zoals bij de implementatie van mijn Top-down boom al vermeld was had ik een idee om deze sneller te maken waarvoor ik helaas geen tijd meer had om het effectief te implementeneren. Inplaats van eerst de node die ik wil splayen te zoeken zou ik ook kunnen zoeken terwijl ik al aan het splayen ben vanaf de wortel, dit zou 1 zoekoperatie minder zijn waardoor deze implementatie mij toch meer performant lijkt

Mijn tweede idee dat ik wel effectief heb kunnen uitproberen betreft de manier waarop ik elementen toevoeg. Inplaats van telkens een nieuw blad toe te voegen kunnen we ook efficienter gebruik maken van de 2-3 eigenschappen van onze boom door te kijken of we de nieuwe waarde kunnen toevoegen aan een al bestaand blad. Zoals je op de bijhorende grafiek kan zien is dit in het begin best wat efficienter aangezien we minder toppen intotaal hebben. Hierdoor moeten we minder diep zoeken naar het volgende blad als we opnieuw een nieuw element willen toevoegen. Uitendelijk word dit even efficient omdat mijn splay methode de toppen samengevoegd heeft.



Theorethische oefeningen

//todo