Verslag Algdata 2 project 2022

2-3 zoekbomen

Ben De Meurichy

1. Theoretische vragen

2. Implementaties

gewone 2-3 boom

Voor de gewone 2-3 zoekboom heb ik er voor gekozen om over alle mogelijke opties per grootte van de subtree, hierdoor is de code nogal omslachtig en groot maar presteert de boom wel zeer goed.
Ik heb er ook speciaal op gelet dat er zo weinig mogelijk nieuwe objecten worden aangemaakt.
Er is gewerkt met subtrees die recursief naar boven tot in de root balanceren.

De vervangbomen die ik heb gebruikt zijn :
//TODO : teken vervangbomen
grootte 6:
grootte 5:
grootte 4:
grootte 3:
grootte 2:

Bottom Up Semi Splay 2-3 boom

De bottomup semi splay boom is geïmplementeerd aan de hand van een stack om de nodes bij te houden die gesplayed moeten worden.

Doordat toppen met 1 sleutel soms omhoog bewegen door de splay bewerking kan het soms niet mogelijk zijn om een sleutel toe te voegen in een enkele top, omdat deze niet noodzakelijk op deze plaats in de boom thuis hoort (zie voorbeeld).
//TODO teken voorbeeld

Hierdoor zal de boom minder toppen met 2 sleutels bevatten en dus redelijk wat dieper zijn in het algemeen.
Dit is denk ik de reden van het duidelijke verschil in prestatie tussen de semi splay bomen en de gewone 2-3 boom.

De weinige dubbele nodes hadden waarschijnlijk voorkomen kunnen worden door het splayen op sleutels en niet op toppen (zoals nu). Dit had ik eerst geprobeerd om te implementeren maar dit vergrootte de complexiteit van de boom enorm.
De basis hiervan is vrij simpel door gewoon de sleutels van de toppen op het splaypad te sorteren en deze op de juiste plaats te steken maar de kinderen van de vervangbomen zitten dan niet altijd op de juiste plaats na een splay bewerking.

De gemakkelijkste oplossing hiervoor zou het herbalanceren van de kinderen zijn tot aan de bladeren van de boom wat natuurlijk onbegonnen werk is en de boom onbruikbaar maakt. Hierdoor zou de diepte van de boom in het algemeen minder groot zijn maar dan zou er wel gesorteerd worden wat per splay stap nog een extra $\Theta(nlog(n))$ toevoegd.
Hoewel dit maar voor maximaal 6 sleutels zou zijn als er zoals nu voor elke splay bewerking 3 toppen werden gekozen wat dus een kost van $\Theta(k * 6log(6))$ zou toevoegen aan alle splaybewerkingen samen per opzoeking.

Voor de 1000-10000 elementen waarvoor dit was gelukt was deze implementatie toch gemiddeld een kwart sneller dan de huidige.

Top Down Semi Splay 2-3 boom

Deze boom is ongeveer hetzelfde geïmlementeerd als de bottom up boom buiten dan dat de boom al tijdens het zoeken splay bewerkingen uitvoert.
Eerst was de *splay* functie recursief gedefinïeerd tot aan de wortel wat voor de bottom up boom geen probleem was maar voor deze boom wel.

Hierdoor zou de boom bij elke opgeroepen splaybewerking tot aan de wortel splayen waardoor deze een pak trager was dan de andere.
Om dit op te lossen heb ik een stack gebruikt met alle nodige toppen die door *splay()* wordt afgehandeld met een while loop.
Tijdens het zoeken worden er toppen toegevoegd aan de stack en als deze 3 toppen groot is wordt er gesplayed.

De vervangbomen die ik heb gebruikt zijn voor beide semi-splay bomen dezelfde:
//TODO: teken vervangbomen

3. Benchmarks en performantie