Algoritmen en Datastructuren II: Verwachtingen varianten RZ-bomen

Tom Naessens
Tom.Naessens@UGent.be

28 oktober 2011

Hoofdstuk 1

Verwachtingen

1.1 Bespreking basismethoden

Opzoeken

Als we opzoeken in een RZ-boom beginnen we bij de wortel en vergelijken we de waarde van wortel met de waarde die we in de boom zoeken. Is de waarde die we zoeken kleiner, doen we hetzelfde bij het linkerkind van de wortel, anders vergelijken we de gezochte waarde met de waarde van het rechterkind van de wortel. Dit blijven we doen tot op het moment dat of de waarde die we zoeken gelijk is aan de waarde van een top, of dat we naar links of naar rechts moeten, maar daar geen top is. In het eerste geval bestaat de top in de boom, in het tweede geval bestaat de top niet. De complexiteit, zoals beschreven in de cursus hiervan is $O(log(n))$ waar n staat voor de verzameling van sleutels die we toevoegen.
Het opzoeken in een inwendige en uitwendige RZ-boom is ongeveer gelijk aan het vorig algoritme. Er zijn 2 verschillen:
Volgens de definitie van een bladzoekboom zitten alle sleutels in de bladeren. We zullen dus in alle gevallen moeten toppen vergelijken tot we in een blad terecht komen. Dit is ons eerste verschil. Het tweede verschil is, dat, in vergelijking met een gewone RZ-boom, we altijd $n-1$ meer toppen hebben voor een sleutelverzameling van n waarden. Dit komt omdat we altijd een top aan een blad toevoegen, en dan herbalanceren. In een inwendige en uitwendige RZ-boom moeten we altijd, behalve in de wortel, een extra top toevoegen. In totaal zullen er dus altijd $n + (n-1)$ toppen in de boom zitten. Voor het vervolg van dit verslag zal ik de variabele n gebruiken voor het aantal 'echte' sleutels, en m (dat gelijk is aan $n-1$) om het aantal inwendige toppen aan te duiden die geen sleutels zijn. Zoals gezegd bij het eerste verschil moeten we altijd toppen vergelijken tot we helemaal onderaan de boom (met een maximum diepte van $2 * log(n+1+m)$) komen, dus zal de complexiteit hiervan gelijk zijn aan $log(n+m)$. Deze $log(n+m)$ is dus zowel een ondergrens als een bovengrens als het gemiddelde geval voor zowel de inwendige als de uitwendige RZ-boom.

Toevoegen

Het toevoegen in een normale RZ-boom bestaat uit twee delen. Eerst zoeken we de top waaraan de nieuwe top moet worden toegevoegd en daarna, indien nodig, herbalanceren we de boom zodat de boom zodat de boom terug voldoet aan de kleureigenschappen. Beide acties hebben een kost van $O(log(n))$.
Bij een inwendige en een uitwendige RZ-boom is het algoritme licht verschillend: we zoeken de top(b) waaraan de nieuwe top(a) zou moeten toegevoegd worden, daarna halen we deze top er uit en voegen we een nieuwe top(c) toe, waarvan de waarde tussen de waarde van top(a) en

top(b) ligt. Daarna herbalanceren we de boom indien nodig vanaf de top die we zelf hebben toegevoegd, top(b) dus.
Bij een uitwendige RZ-boom hebben de bladeren, de 'echte' sleutels dus, geen kleur. Dit zorgt ervoor dat we vrijer zijn in het kiezen van de kleur van de top die we 'tussen 2 sleutels steken'. Hierdoor verwacht ik dat we tijdens het herbalanceren, in vergelijking met een inwendige RZ-boom, in sommige gevallen vroeger mogen stoppen.
Als we hier kijken naar de verschillende kosten vinden we dat de kost voor het zoeken van de top waaraan we de nieuwe top toevoegen gelijk is aan de maximale diepte van de boom, maximum $2 * log(n + 1 + m)$. Daarna moeten we nog herbalanceren. Zoals beschreven op p. 23 van de cursus komen we met elke herbalancering 1 stap dichter bij de wortel. Aangezien we bij de top die we zelf hebben toegevoegd herbalanceren beginnen met herbalanceren, en niet vanaf een blad, moeten we dus maximaal over $2 * log(n + 1 + m - 1) = 2 * log(n + m)$ toppen balanceren. We kunnen dus besluiten dat de complexiteit hiervan voor zowel een inwendige als uitwendige RZ-boom gelijk is aan $O(log(n + m))$.

Verwijderen

De complexiteit van het verwijderen bij een standaard RZ-boom is, zoals beschreven in de cursus, gelijk aan $O(log(n))$.
Het algoritme om te verwijderen in de varianten van een RZ-boom komt in grote lijnen overeen met het algoritme dat hierboven, bij het toevoegen dus, beschreven staat. Het enige verschil is dat we in plaats van een sleutel toevoegen, hier een sleutel gaan verwijderen. Hierbij kunnen we ook één extra top, die geen element is van de sleutelverzameling, verwijderen. Na het verwijderen van deze twee toppen moeten we eventueel de kleuring en/of de diepte van de boom herstellen op dezelfde manier als bij een standaard RZ-boom. Aangezien we bij elke herbalancering één stap dichter bij de wortel komen, en de maximale diepte van een boom $2 * log(n + 1 + m)$ is, zal dit dus een complexiteit hebben van $O(log(n + m))$.

1.2 Datagebruik

Als we het datagebruik voor een gewone RZ-boom bekijken hebben we geen overhead: voor elke sleutel hebben we 1 top in onze boom. In vergelijking met een inwendige en een uitwendige RZ-boom, waar we altijd $n - 1$ meer toppen nodig hebben voor een sleutelverzameling van n hebben we dus een overhead van $n - 1$ toppen, die allemaal geheugen innemen. Het totale ingenomen geheugen van een inwendige RZ-boom is dus $n + m = 2n - 1$.
In een uitwendige RZ-boom houden de echte sleutels echter geen kleur bij, dus zullen deze iets minder geheugen innemen dan inwendige RZ-bomen, maar nog steeds meer dan gewone RZ-bomen. Dit zal ongeveer gelijk zijn aan $n + m - n * c = 2n - 1 - n * c$, waarbij de c staat voor het geheugen dat wordt ingenomen door de kleureigenschap.

1.3 Overzicht

Om een verzameling van n sleutels toe te voegen is de complexiteit gelijk aan: *(Hier staat de m (= $n-1$ niet in functie van n geschreven om duidelijk te maken dat er wel degelijk een verschil zit in de complexiteit voor eenzelfde sleutelverzameling n.)*

	RZ-Boom	Inwendige RZ-Boom	Uitwendige RZ-boom
Opzoeken	$O(log(n))$	$O(log(n+m))$	$O(log(n+m))$
Toevoegen	$O(log(n))$	$O(log(n+m))$	$O(log(n+m))$
Verwijderen	$O(log(n))$	$O(log(n+m))$	$O(log(n+m))$
Datagebruik	n	$n+m$	$n+m-n*c$

1.4 Besluit

Als we de tabel hierboven bekijken is het duidelijk dat de standaard RZ-boom de beste keuze is op alle vlakken, daarom zal deze in de praktijk ook het best zijn om te gebruiken. Voor een aantal randgevallen kan het wel zijn dat de inwendige of uitwendige RZ-boom beter is dan een standaard RZ-boom. Dit is echter wel verwaarloosbaar aangezien dit maar voor heel weinig gevallen geldt. Als voorbeeld nemen we een uitwendige RZ-boom met één top. Deze top heeft geen kleur. Als we 1 top toevoegen bij een lege standaard RZ-boom is deze automatisch zwart gekleurd, wat er voor zorgt dat een uitwendige RZ-boom met 1 sleutel een verwaarloosbaar beetje minder geheugen gebruikt dan een standaard RZ-boom.

Als we de inwendige RZ-boom met de uitwendige vergelijken merken we op het eerste zicht niet echt veel verschil in complexiteit van de basisbewerkingen. Wat wel een verschil uitmaakt is de kleuring. Dit zorgt er ten eerste voor dat we iets minder data moeten gebruiken, aangezien de 'echte' sleutels geen kleuren hoeven bij te houden. Dit zal echter wel verwaarloosbaar zijn voor een groot aantal toppen. Ten tweede, zoals besproken bij het toevoegen, zorgt dit ervoor dat we hier vrijer zijn bij het kleuren van de toppen die we er zelf tussen steken, waardoor we over het algemeen efficiënter de toppen kunnen kleuren, en zo eventueel vroeger mogen stoppen. Om deze twee redenen verwacht ik dat een uitwendige in het algemeen beter zal presteren dan een inwendige RZ-boom.
Als algemeen besluit verwacht ik dat een standaard RZ-boom het best zal functioneren, daarna de uitwendige RZ-boom en dan de inwendige RZ-boom.