Tema 9: Jämförelse av datastrukturer

Amanda Enhörning amen4199 Annie Jeppsson anje3090

7 september 2023

I denna rapport undersöker vi skillnaden på prestanda mellan datastrukturerna treap, röd-svart träd och splayträd. Vi har valt att genomföra mätningarna med hjälp av att undersöka snittet av antalet iterationer som görs vid varje operation/set av operationer, specifikt insert och remove. De datamängder vi bedömde var intressanta att testa var små och stora mängder av sorterad och osorterat data. Vi undersökte även hur ordningen på operationerna insert och remove påverkade prestandan för splayträd och treaps. Vi har valt att dokumentera våra resultat i form av antalet iterationer per värde eftersom den faktiska tiden kan bero på olika faktorer, som datorns hastighet, komplexiteten av operationen och storleken på datat som behandlas. Vi ansåg att iterationer per element därför var lättare att jämföra.

Först tittade vi på hur datastrukturerna hanterade slumpmässig data i små mängder jämfört med stora. Resultaten skilde sig minimalt, vilket var väntat då alla tre datastrukturer hanterar både små och stora datamängder väl. Vi märkte däremot att treaps kunde bli mindre effektiva ju större datamängden blev, detta beror på att desto fler noder som finns i trädet, desto mindre effektiv blir treaps eftersom deras slumpmässigt valda prioriteringsvärde kanske inte fördelas jämt och detta kan leda till ett obalancerat träd.

Sedan tittade vi på sorterat data, och där blev det mer intressant. Vi valde att titta på två olika fall, dels när man sätter in ett litet antal värden och dels när man sätter in ett större antal värden. Vi valde 1-10 respektive 1-1000, sorterat i stigande ordning. För varken det röd-svarta trädet eller treapen blev det någon större skillnad för litet antal värden jämfört med stort antal värden, utan skillnaden ligger främst hos splayträdet.

Att skillnaden inte är så stor, eller i stort sett icke-existerande, beror för det röd-svarta trädet på att då man sätter in ett litet antal värden så förändras höjden på trädet inte på ett märkbart sätt, eftersom nya noder läggs till längst ned i trädet. När man sätter in ett stort antal värden kommer höjden att öka, men ökningen sker logaritmiskt. Det betyder att antalet iterationer fortfarande kommer vara ungefär samma, med en liten ökning iterationer på grund av ökningen av höjden på trädet. Eftersom en treap är ett självbalanserande binärt sökträd kommer höjden på trädet osannolikt att öka märkbart när man sätter in ett litet antal värden. Därför kommer antalet iterationer i stort sett bli proportionellt till höjden av treapen. Sätter man istället in ett stort antal värden kommer trädet växa i storlek och höjden kan öka. Dock kommer troligen antalet iterationer inte öka signifikant eftersom trädet är självbalanserande, och antalet iterationer kommer fortfarande vara proportionella till höjden av trädet. Som man kan se i vår tabell ökade antalet iterationer litegrann med ett större antal värden, men inte signifikant.

Splayträdet får däremot ett mycket större antal iterationer per värde när vi endast sätter in 10 värden jämfört med när vi sätter in 1000 värden. Detta beror på att splayträdet är en självjusterande sökträddatastruktur som använder sig av splay()-metoden, vilken utgör en sekvens av rotationsoperationer, för att flytta nyligen åtkomna noder närmare roten. Detta gör det möjligt för oftare åtkomna noder att snabbare nås i framtiden. Med en stor datamängd ökar antalet noder i trädet, vilket gör det mindre troligt att nya noder läggs till längst ned i trädet utan kommer istället läggas till i en del av trädet som är närmare roten. Ju fler element vi lägger in desto färre iterationer krävs, och därför blir alltså antalet iterationer per värde mindre i vårt fall när vi sätter in 1000 värden jämfört med 10.

Slutligen undersökte vi hur splayträd och treaps påverkas av olika mönster av insättningar och borttag. För att göra detta satte vi in ett stort antal osorterat data i båda datastrukturerna och jämförde sedan prestandan när vi utförde insättningar och borttagningar i olika ordning. Första jämförde vi antalet iterationer per värde när vi satte in 1000 element och sedan tog bort 1000 element från datastrukturerna. För det andra jämförde vi antalet iterationer per värde när vi satte in ett element och sedan tog bort ett slumpmässigt element 1000 gånger. Resultaten visade att Splayträdet presterade sämre när insättningar och borttagningar skedde blandat, medan Treap prestanda påverkades väldigt olika, ibland knappt alls och ibland mer. Att Treaps resultat varierade så mycket i detta scenario var väntat då dess prestanda beror mycket på hur slumpmässig datat är. Ju mer sorterad, desto fler rotationer krävs det och eftersom datat i testet är helt slumpmässigt kommer det att uppstå situationer där den är mer sorterad är optimalt.

Förklaringen till varför Splayträdets prestanda blev så mycket sämre är återigen på grund av att den självbalanserar på så sätt att nyligen använda element är lättare att nå. Om vi sätter in alla värden och sedan plockar bort dem kommer trädet att balansera sig själv på ett optimalt sätt, men om vi blandar insättning och borttag kan det trädet inte balanseras lika bra och detta leder i sin tur till längre söktider.