ALDA VT12: Avancerade datastrukturer

Leon Hennings leonh
Kamyar Sajjadi kamy-saj

1 mars 2012

Innehåll

1 Treap

Treap är en relativ enkel datastruktur. Den är både lätt att förstå och lätt att implementera Datastrukturen är ett binärt sökträd där varje nod består av ett data, ett vänster barn och ett höger barn, samt en prioritet. I binära sökträd gäller det att alla noder i vänstra subträdet av en nod har element som är mindre än aktuella noden och alla noder i högra subträdet har element som är större än aktuella noden. Noderna får sin prioritet slumpmässigt av en random generator. Prioriteten måste följa heap-ordern, det innebär att varje barns prioritet måste vara likamed eller lägre än sin förälders. Prioriteten används för att det binära sökträdet ska balanseras. Den nod med högst prioritet är den nod som är roten. När en nod med högre prioritet sätts in i treapen så görs rotationer för att heap-ordern ska följas och detta balanserar i sin tur trädet till viss del.

Insättning i en treap är väldigt lik insättning i ett binärt sökträd. Det rekursiva anropet börjar vid roten och gör jämförelser ner mot löven i trädet tills den rätta platsen där noden ska placeras hittas. Den nya noden placeras på denna plats och får en prioritet x av random generatorn. När vi börjar gå ur rekursionen så kontrolleras det om heap-ordningen följs. Om någon nod har högre prioritet än sin förälder så kommer det ske en rotation för att heap-ordningen ska följas. Detta leder till att trädet balanseras upp.

När man gör borttag av noder så måste man försäkra sig om att trädet är binärt efter bortag och att heap-ordern följs. Detta kan göras lättast om man endast tar bort noder som är lövnoder, dvs noder som ej har några barn. Vid borttag av en nod söks noden upp i trädet genom rekursiva anrop, med start från roten. Om noden är ett löv så kan den enkelt tas bort genom att ändra så att föräldern pekar på null. Som vi nämnde tidigare så är detta det lättaste fallet för att samtliga kriterier ska följas. Därför är det bra om vi kan hamna i detta tillstånd när vi ska göra borttag av noder med barn. Detta görs genom att sätta nodens prioritet till oändligheten och sedan rotera den neråt tills den blir en lövnod, och då göra borttaget.

Sökning i treaps utförs på samma sätt som sökningar i binära sökträd. Vi börjar med att kolla om roten är likamed det sökta värdet, om det ej är fallet så går vi vidare och kollar om det sökta värdet är mindre än den aktuella noden. Skulle värdet vara mindre så går vi vidare till det vänstra subträdets root och kollar om det är likamed det sökta värdet. Samma sak kommer ske om det sökta värdet är större än den aktuella noden. Denna process upprepas tills värdet vi söker efter hittas och true returneras. Skulle det sökta värdet ej finnas med i trädet returneras false.

Vi har svårt att komma fram till vettiga användningsområden för treap. Som Weiss nämner i boken så är det relativt lätt att implementera denna datastruktur. I förhållande till sin enkelhet får man en rimlig tidskomplexitet vid insättning och borttag. Vi tycker det är bättre att använda sig av treaps än vanliga binära sökträd då de är relativ lätta att implementera och man får bättre tidskomplexitet av att använda sig av dessa. Treaps kan vara bra att använda om vi har statistik på hur ofta ett element kommer att användas. Då kan man använda statistiken som prioritet och man får ett balanserat träd baserat deras användning.

2 Splayträd

Splayträd är en typ av binära sökträd som har en amorterad tidskomplexitet för M operationer O(M log N). Detta innebär att även om vissa operationer i värsta fall kan ta O(N) så jämnas det ut för flera operationer så att genomsnittet bli O(log n). Detta åstadkoms genom att det för alla enkla operationer, insättning, borttag och sökning görs rotationer av trädet så att den sökta noden blir den nya rotnoden. I fallet vid borttagning blir det istället den borttagna nodens förälder som blir ny rotnod. Uppflyttningen av en nod kallas en splay operation.

Splayningen av trädet kan göras nerifrån och upp när den noden hittats eller placerats. Detta kräver att man antingen gör det rekursivt och lagrar sökvägen på stacken eller att man har dubbellänkade noder som pekar på sina föräldrar. Båda dessa alternativen använder mycket extra minne och ett tredje sätt finns där man gör splayningen uppifrån och ner, så kallade Top-Down Splayträd.

För top-down splayning används istället två noder, vänster V och höger H, som håller noderna som är lägre respektive högre än den som ska bli nya rotnoden. Efter splay operationen är klar sätts noden i V med högsta värdet till rotnodens vänstra subträd och noden i H med lägst värde pekar till rotens högra subträd. Därefter sätts rotnodens vänsterbarn till V och dess högerbarn till H.

Rotationerna som görs kallas zig, zig-zig och zig-zag. Zig rotationen byter plats på en förälder och ett av dess barn. Den används bara när botten av trädet nåts, och subträdets höjd är udda. Zig-Zig inverterar ordningen på ett subträd, exempelvis en förälder X, med vänsterbarn Y och dess vänstra barn Z så att Z är förälder, Y dess högerbarn och X blir Ys högra barn. Zig-Zag görs när förälderns vänsterbarn har ett högerbarn och vice versa. Subträdrotens barnbarn sätts till subträdets rot. Den föredetta föräldern och dess förälder blir dess högra respektive vänstra barn. Detta motsvarar att man först gör en zig rotation med föräldern åt ena hållet och sen en zig rotation med roten åt andra hållet.

Splayträd funkar bra när fler element har identiska nycklar, till skillnad från andra balanserade sökträd. Noderna behåller ordningen de har för alla operationer och det är möjligt att i sin implementation specifiera om sökmetoden ska hämta elementet längst till höger eller längst till vänster av de med identisk nyckel.

Splayträd är bäst att använda i situationer där samma element behöver hämtas flera gånger. Eftersom ett nyligen insatt eller hämtat element blir rotnod kommer nästa access av den göras på konstant tid. Ett exempel på när detta skulle kunna användas är ifall man i ett sorts kassasystem ska interagera med kunder och kunden först i kön kan ha flera ärenden som ska utföras. Då flyttas den aktuella kunden upp i toppen av trädet och för behandlning av kundens nästa ärende tar hämtning av kundens data konstant tid.

3 Poolfrågor

Veckans fråga behandlar splayträd.

3.1 Fråga 1

Om du har ett balanserat splayträd som innehåller elementen 1 till 10, hur kommer trädet se ut ifall du söker igenom det sekventiellt efter elementen 1 till 10? För högre betyg ska du utifrån ditt svar resonera kring ifall det är lämpligt att iterera över ett splayträd med hänsyn till antal steg som behövs för accessa varje nod. Jämför effektiviteten med att iterera över en länkad lista.

3.2 Fråga 2

Vad är skillnaden mellan ett bottom-up och ett top-down splayträd? För högre betyg ska du resonera kring vilket som är mest effektivit och varför.