Populärt om populariteter

Josefin Bodell

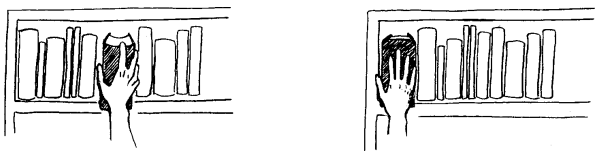
Hur forskar man i matematik? Bland icke-matematiker florerar en mängd olika föreställningar – allt från att matematiken redan är ”färdig”, plus och minus är ju redan kända, och att forskare således räknar ”som vanligt” men med enormt stora tal och många decimaler, till att det bara bollas med bokstäver i komplicerade ekvationer. Sysslar matematiker med något obegripligt? Här är ett exempel från ett pågående arbete som vi hoppas visar på andra sidor av forskningsarbetet.

För en tid sedan skulle jag berätta om min forskning för en journalist. För att om möjligt undvika att skrämma bort både journalisten och de tilltänkta läsarna, koncentrerade jag mig på att beskriva själva grundproblemet, de grundläggande idéerna och ge en uppfattning om hur frågeställningarna kan se ut utan att gå in på komplicerade formler och de mer ”tekniska” problemen. En nog så viktig del av forskning är ju att ställa vettiga frågor, att formulera problem som man sedan kan ge sig i kast med att lösa.

Problemområdet

Jag forskar i sannolikhetsteori. Problemområdet kring vilket avhandlingen kretsar kommer däremot från datorernas värld. Det handlar om att i matematiska, sannolikhetsteoretiska termer beskriva ”hur bra” en viss typ av metoder är för att organisera datorns minne.

Tänk dig att vi har ett antal böcker i en bokhylla. Varje bok symboliserar en informationsenhet och bokhyllan motsvarar själva datorminnet. När vi vill hämta en viss bok i den här hyllan, börjar vi alltid leta efter den längst upp till vänster, d v s vi jämför titeln på den bok vi letar efter med titeln på denna ”första” bok i hyllan. Om titlarna inte överensstämmer går vi vidare till nästa bok i hyllan, jämför titlar, och arbetar oss på detta sätt successivt fram till den sökta boken.



Move-to-front.

Nu vill man förstås hitta det man letar efter så fort som möjligt. En idé är därför att utnyttja att böckerna är olika populära – vi kanske vill ha tag på ordlistan oftare än Shakespeares samlade verk (eller tvärtom). Om vi placerar böckerna i fallande popularitetsordning så att den populäraste boken står först, följd av den näst mest populära, etc, kommer söktiden bli så kort som möjligt eftersom den bok vi oftast vill ha står mest lättillgängligt, medan vi måste leta längre efter böcker vi sällan söker.

Men om vi inte på förhand vet hur populära de olika böckerna är? Eller om deras popularitet ändras med tiden? Hur ska vi då bäst ordna dem? Vi vill på något sätt få böckerna att ”automatiskt” ordna sig i fallande popularitetsordning. En lösning är följande strategi:

Josefin Bodell *är tekn. lic. och doktorand vid matematiska institutionen, avdelningen för matematisk statistik vid KTH, Stockholm.*

Varje gång vi har letat upp en bok plockar vi ut den och läser den. När vi sedan ska ställa tillbaka boken i hyllan ställer vi den först istället för i den position vi hittade den.

Efter en tid kommer nu populära böcker att befinna sig långt fram medan de impopulära samlas längre bort eftersom de sällan söks. Den här metoden kallas för *move-to-front* (flytta det sökta längst fram).

Söktiden

Hur länge måste man då leta för att hitta rätt bok i en sådan bokhylla? Varje jämförelse mellan titeln på den sökta boken och en bok i hyllan tar en konstant tid så vi kan mäta söktiden i det antal jämförelser som krävs för att hitta rätt. Söktiden ges alltså av det antal böcker som står före den sökta boken plus ett – vi måste ju dessutom jämföra med den ”rätta” boken för att kunna avgöra när vi hittat den.

Om vi kände till exakt hur böckerna stod från början och när (i vilken ordning) de anropades skulle vi i princip kunna tala om precis hur bokhyllan såg ut vid varje given tidpunkt och därmed hur lång söktid vi skulle behöva för att hitta var och en av böckerna. I praktiken är detta dock inget lockande företag även om man hade tillgång till den erforderliga informationen. Dessutom får vi ingen upplysning om hur bra metoden är i allmänhet – ändrar vi följden av anrop måste vi räkna om alltihop igen. Nej, vi vill ha ett mer övergripande grepp om prestanda, så vi tar till sannolikhetsteorin och blandar in slumpen i skeendet.

Varje gång vi gör en sökning låter vi slumpen bestämma vilken bok vi letar efter. Slumpen får dock inte härja alldeles fritt, utan vi styr den så att varje bok väljs med motsvarande popularitet. I varje sökning är sannolikheten att boken Bok söks precis lika med boken Boks popularitet som vi symboliserar med en bokstav (p_{Bok}).

Med den här modellen visar det sig att vi kan besvara en rad frågor och ge en tydlig bild av hur *move-to-front*-metoden presterar uttryckt endast i böckernas allmänna popularitet.

– Hur lång blir söktiden i genomsnitt?

Innan vi gör en sökning är detta värde vår bästa gissning av hur länge vi kommer att få leta. Vi förväntar oss att söktiden kommer hålla sig kring detta värde. Men varje sökning tar ju inte lika lång tid.

– Hur mycket varierar söktiden kring medelvärdet?

– Vad är sannolikheten att söktiden blir precis lika med en viss tid?

– Vad är sannolikheten att man behöver leta kortare än en viss tid, d v s att den sökta boken kommer att befinna sig bland de x första?

Den senare typen av fråga ställer sig datorkonstruktörer idag. När man konstruerar datorer för vissa ändamål vill man ibland placera ett litet, snabbt (och dyrt) buffertminne, en s k cache, mellan processor och primärminne. Problemet är att välja rätt cachestorlek. Man vill att cachen ska vara så liten som möjligt samtidigt som de oftast använda enheterna ska få plats. Det man inte hittar i cachen måste sökas i det ”vanliga” primärminnet.

I början beror böckernas ordning i bokhyllan mycket på hur de stod från början, men efter en tid har hyllan ”glömt” hur den startade – *move-to-front*-algoritmen bestämmer ensam hur det ser ut. De matematiska uttrycken i svaren på frågorna ovan blir då enklare och i någon mening mer generella (eftersom det nu inte längre beror på hur vi började).

– Hur ser det ut efter lång tid?

”Lång tid” i matematiska termer betyder i allmänhet att man låtit tiden gå mot oändligheten. I praktiken vill man dock inte vänta så länge utan vill åtminstone approximativt kunna använda resultaten tidigare. Hur länge måste man vänta innan det går bra?

– Hur lång tid är ”lång tid”?

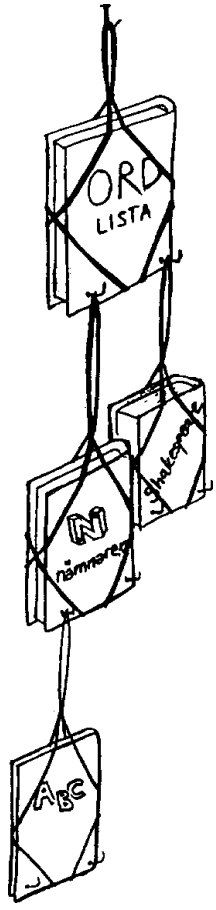
Liksom tidigare beror svaret på hur de olika populariteterna ser ut.

Ytterligare en parameter som vi kan låta gå mot oändligheten är antalet böcker.

– Vad händer när antalet böcker blir stort?

Alfabetsordning

Hittills har vi bara använt böckernas olika popularitet för att sortera dem men det finns ju dessutom en känd ordning mellan dem – alfabetsordningen. För att ta vara på även denna information bygger vi om vår hylla till en mer mobil-liknande struktur: Första boken hänger vi närmast taket och monterar fast två krokar, en i bokens vardera nedre hörn (se figur 1). Dessutom lägger vi på minnet vad boken har för titel. Nästa bok hänger vi sedan i vänster krok om dess titel kommer före i alfabetsordning (eller är en kopia av den första). Annars hänger vi den i höger krok. Därefter monteras krokar även på denna bok och vi upprepar samma förfarande tills alla böcker hänger i mobilen. Vår bokhylla har nu antagit formen av ett binärt sökträd.



Figur 1

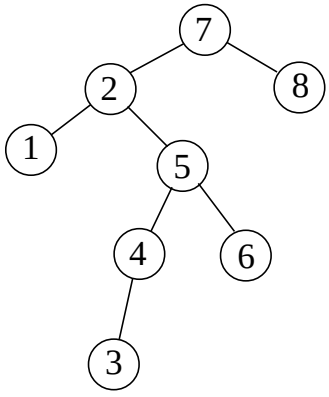
Om trädet är balanserat, d v s om det bara finns fria krokar närmast golvet, och vi söker enligt samma princip som tidigare – börja med boken närmast taket, jämför successivt titel för titel tills vi är framme – kommer vi nu kunna utesluta hälften av de kvarvarande kandidaterna i varje jämförelsesteg. Säg t ex att vi söker den bok vars titel kommer allra först i alfabetsordning. Först jämför vi med boken närmast taket. Eftersom titeln vi söker kommer före i alfabetsordning vet vi att den måste hänga någonstans under vänster krok. Vi kan därmed utesluta alla som hänger under första bokens högerkrok. På samma sätt blir det i alla följande jämförelser tills vi hittat det vi sökte.

Som tidigare vill vi nu att populära böcker automatiskt ska hamna i början av strukturen, vilket nu betyder ”nära taket” (vi börjar ju alltid vårt sökande med att jämföra med boken närmast taket). I analogi med *move-to-front* flyttar vi därför den bok vi sökt rätt på överst i trädet. Nu kallas metoden *move-to-root* (flytta /det sökta/ till roten) eftersom boken närmast taket, den trädet ”växer” och förgrenar sig från, kallas trädets rot.

I *move-to-front*-fallet förblev den inbördes ordningen mellan böckerna oförändrad bortsett från att den sökta boken flyttades främst. Nu blir det betydligt mer komplicerat. Om vi bara flyttade den sökta boken till roten skulle bokstavsordningen i trädet förstöras och därmed skulle hela vitsen med binära sökträdsstrukturen gå förlorad. Varje gång något flyttas till roten måste vi således möblera om stora delar av trädet.

Det gäller alltså först och främst att genomskåda dels hur trädet lämpligen bör möbleras om och dels vad det innebär för de olika böckernas positioner så att man kan hålla reda på vad söktiden blir för den bok slumpen bestämmer att vi ska söka.

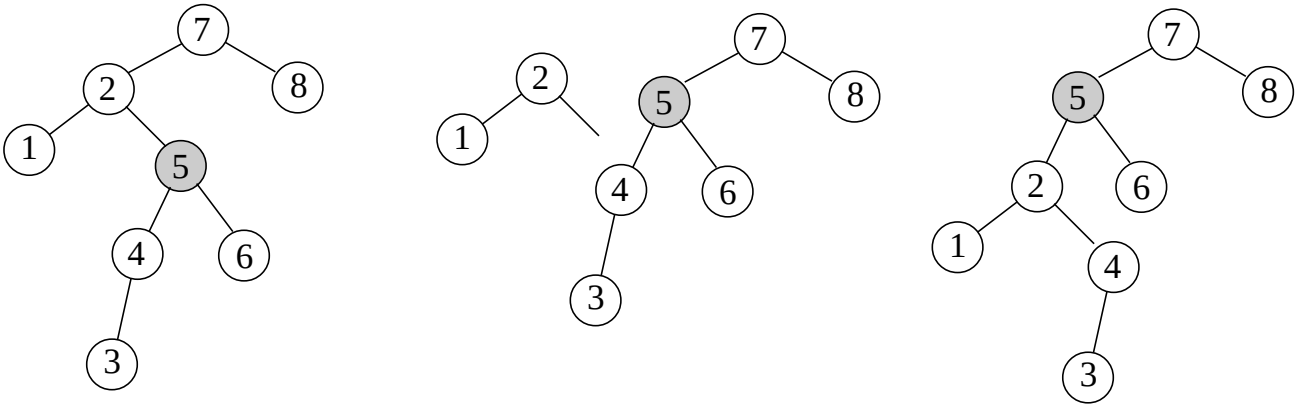
Antag för enkelhets skull att vi har böckerna 1,2,...,8 i följande binära sökträd:



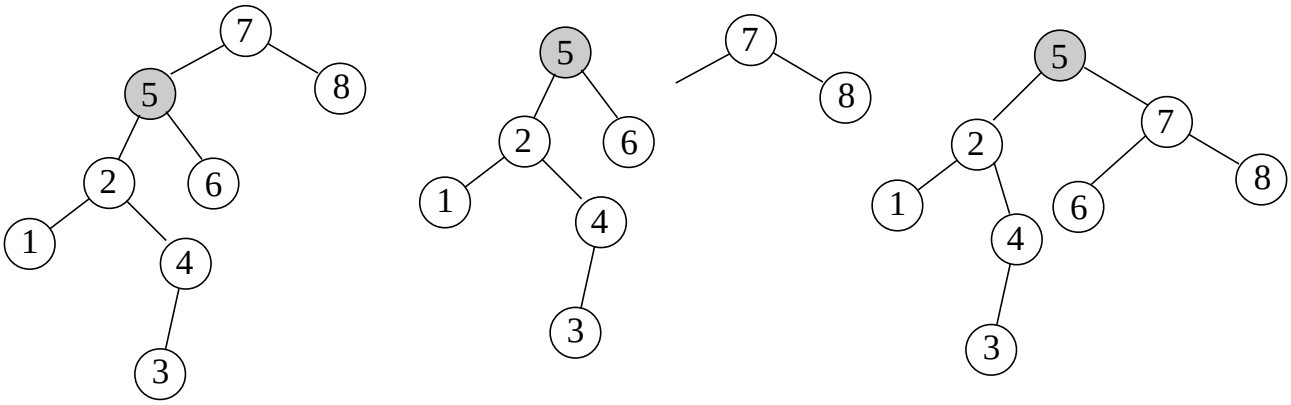
Figur 2

Detta är bara ett av 1430 olika sätt att ordna dessa böcker i ett binärt sökträd.

Antag vidare att vi just anropat ”5” och nu ska flytta ”5” till trädets rot. Tricket består i att genomföra förflyttningen stegvis.



Figur 3



Figur 4

Börja med att göra ”5” till rot i närmast större delträd, dvs flytta ”5” till ”2”:s nuvarande position som i figur 3. Haka loss ”2” för att göra plats åt ”5”, dra i ”5” och fäst den i ”7”:s vänsterkrok. Nu måste ”2” in i trädet igen så att vi inte tappar bort den. Eftersom $2 < 5$ hänger vi ”2” i ”5”:s vänsterkrok. Det innebär att vi måste haka loss ”4” för att göra plats. Men $4 > 2$ så ”4” passar precis i ”2”:s lediga högerkrok (i vilken ”5” hängde innan vi började möblera om).

Nu är ”5” en nivå närmare roten och ordningen mellan böckerna är fortfarande intakt. Upprepa samma procedur för att flytta upp ”5” ytterligare en nivå, se figur 4. Flytta tillfälligt på ”7”, dra upp ”5”, fäst ”7” i ”5”:s högerkrok eftersom $7 > 5$ och ”6” i ”7”:s vänsterkrok eftersom $6 < 7$. ”5” är nu rot till hela det (fortfarande) binära sökträdet och vi har genomfört en *move-to-root*-operation.

Observera att de böcker som inte omnämnts vid omflyttningarna bara följer med sina ”förfäder”.

Söktiden mäts även här i det antal jämförelser vi måste göra för att finna den bok vi vill ha. Med samma terminologi som i släktträd ges nu söktiden av antalet förfäder, de böcker vi passerar på vägen ner från roten till den sökta boken, plus ett (för den sista jämförelsen som talar om att vi är framme). Det räcker alltså att hålla reda på vilka förfäder varje bok har och hur dessa släktförhållanden ändras när en viss bok flyttas till roten.

De frågor vi vill besvara är väsentligen desamma som tidigare: Hur lång är söktiden i genomsnitt? Hur mycket varierar den kring detta medelvärde? Hur ser sannolikhetsfördelningen ut? Vad händer efter lång tid? Vad händer när antalet böcker blir stort (går mot oändligheten)? I det här fallet blir sannolikhetsfördelningen för söktiden oerhört komplicerad så frågan vad som händer om vi har väldigt många böcker är fortfarande till stor del obesvarad utom i det speciella fall att alla böcker är lika populära.

Ytterligare en fråga man kan ställa sig i trädfallet är:

– Vad händer i värsta fall och när inträffar detta?

Det bästa som kan hända är förstås både nu och i fallet med den vanliga bokhyllan att vi alltid vill ha en och samma bok. Den kommer då att stå främst hela tiden och vi behöver inte leta alls. Värsta fallet, dvs då vi får den maximala söktiden, inträffar i den vanliga hyllan om alla böcker är lika populära och vi alltså inte kan ”sortera bort” några impopulära som vi kan ställa längst bort.

Länge trodde man att samma sak gällde för träd men om man verkligen provar visar det sig att det blir ännu värre (längre söktid) om de första och de sista böckerna (i alfabetsordning) är något mer populära än de övriga. Det är av avgörande betydelse att det är just det första och de sista som är mest populära. När de bildar rot tvingas nämligen alla andra böcker att hänga under endast den ena av krokarna för att bokstavsordningen ska stämma och då har man inte längre någon glädje av trädstrukturen. Värsta-fallsuppförandet förklaras av att trädet kommer att växla mellan extremt obalanserade lägen.

Hittills har vi bara numeriska resultat om det här fenomenet. Exakt vad som händer och hur man beskriver (och bevisar) det analytiskt är en av de frågor som återstår att besvara.

Referenser

Allen, B. och Munro, I. (1978). Self-organizing binary search trees. *Journal of the Association of Computer Machinery. Vol. 25*, s. 526-535.

Bodell, J. (1995). *Aspects of the search time in binary search trees modified according to the move-to-root heuristic.* Tekn. lic. avhandling, Matematiska institutionen, KTH.

McCabe, J. (1965). On serial files with relocatable records. *Operations Research. Vol 13*, s 609 – 618.

Knuth, D. (1973). *The Art of Computer Programming, Vol. 3*: Sorting and Searching. Addison-Wesley, Reading, MA.