Laboration 2, analys av datastrukturer, TDA416

Grupp 7: Erik Öhrn, Paula Eriksson Imable

2015-02-20

Kort om varje datastruktur:

SLC

Sorted Linked Collection är en länkad lista som även är ordnad. När ett objekt sätts in i listan stegar listan igenom de objekten som finns och jämför med detta. När den hittar var objektet platsar sätter den in det där. Detta leder till att objekt som är större (enligt dess egen sortering, alltså typens inbyggda sortering enligt java) kommer ta relativt lång tid att sätta in, då listan kommer jämföra alla objekt i listan för att sedan sätta in det längst bak. När objekt hämtas stegas listan igenom för att hitta rätt objekt. Detta gör att objekt med låga värden (eller tidigt alfabetiskt, om det är ord, etc) kommer gå snabbt att hämta, medan listan måste stega igenom många objekt för att hämta de större. Den linjära sökningen innebär att både insättning och sökning får komplexiteten O(n).

BST

Det binära sökträdet sätter in ett objekt även det sorterat, men ser på varje nod om dess objekt är större eller mindre än (eller lika stort som) objektet som skickas in. Beroende på svar kommer nästa sak upprepas med antingen dess barn till höger eller till vänster, och fortsätter så till dess att rätt plats hittas varvid det sätts in. Höjden regleras inte, vilket gör att trädet i värsta fall kan bete sig som en lista, med enbart en barnnod på nod och alla följande värden antingen till höger eller till vänster om noden. Då det inte är balanserat kommer det i värsta fall få komplexiteten O(n), både för insättning och sökning. I medelvärde och för slumpmässig data kommer det få O(log(n)) som komplextet för båda. Detta eftersom trädet då i genomsnitt halveras efter varje sökning.

AVL

AVL-trädet är självbalanserande. Data sätts in på samma sätt som i BST, men höjden regleras efter varje insättning och borttagning med hjälp av rotationer om höjdskillnaden mellan två subträd överstiger 1. Detta gör att exemplet ovan, när trädet beter sig som en lista, undviks. Således är komplexiteten O(log(n)) för insättning och uttagning.

Splay

I vårt splayträd splayas ett objekt enbart när det söks, eller det närmaste objektet om det inte hittas. Normalt sätt splayas ett objekt även vid insättning och borttagning, men enligt labbhänsvisningarna skulle detta inte göras i vår implementation. Splayträdet fungerar som ett binärt sökträdet, med skillnaden att det senaste sökta objektet flyttas till toppen genom en serie rotationer. Hittas inte objektet kommer det senaste sökta flyttas till toppen. Det resulterande trädet ordnar sig därefter. Detta leder till att de objekt som söks efter ofta

hamnar högt upp i trädet. Då det inte är balanserat finns möjligheten att en sökning efter ett värde kan ta O(n), om trädet fått liknande liststruktur som BST riskerar få. Detta brukar dock ordna ut sig och undantaget enstaka sökningar opererar splayträdet med komplexiteten O(log(n)).

Textanalys

TEXT1

Text1 är en till synes vanlig text, om än skriven på något ålderdomlig svenska. Den är inte sorterad på något sätt och innehåller upprepningar. Den består av 3468 ord.

SLC

I den sorterade listan kommer ett ord, när det sätts in, sorteras. Om ett ord är väldigt långt ned, exempelvis "vaccin" måste därför hela eller nästan hela listan gås igenom vid hämtning. Det blir många jämförelser vid denna text eftersom den är osorterad från början och innehåller många olika ord (varav många återkommer flera gånger och är alfabetsikt stora, som "zz", "vaccin", "under" och "till").

BST

Då texten är någorlunda slumpmässig och inte har så många upprepningar för varje ord kommer trädet bli någorlunda jämt. Hämtningen går snabbt eftersom den delar antalet noder i ungefär två vid varje jämförelse.

AVL

AVL-trädet liknar det binära sökträdet och har ungefär likvärdig prestanda. Trädet balanseras dock när element sätts in, vilket gör att sökningarna kommer operera i O(log(n)) i tid. Att hämta objekt kommer därför här göras på något färre jämförelser eftersom antalet noder halveras vid varje hämtning.

Splay

I denna text finns inte så många upprepningar, men väl några. Dessutom kommer de stora orden komma närmare och närmare toppen eftersom programmet hämtar orden i bokstavsordning. Detta gör att splayträdet, relativt till de andra, visar på god prestanda.

TEXT2

Samma text som början av TEXT1 förutom att alla ord nu upprepas och därmed står fem gånger i rad. Texten består av 3480 ord.

SLC

SLC kommer fungera på liknande sätt som tidigare, men då bara det första av orden vid upprepning behöver returneras minskas antalet jämförelser avsevärt. Av samma anledning kommer även insättning gå snabbare.

BST

Av samma anledningar kommer insättning och hämtning från BST gå på färre jämförelser.

AVL

Även AVL-trädet gagnas av detta.

Splay

Splayträdet är det som får mest ut av förändringen till många upprepningar. Då trädets struktur går ut på att ta ofta sökta element till toppen kommer antalet jämförelser minska drastiskt när texten innehåller många upprepningar.

TEXT3

En ordlista som är sorterad i bokstavsordning, utan upprepningar. Den består av 3445 ord.

SLC

Denna typ av text får SLC stora problem med. Då den redan är sorterad kommer varje efterföljande ord sättas in längst bak i listan. Detta gör, vilket vi skrev i introduktionen för SLC, att den måste jämföra med alla andra objekt som redan finns i den. Detta gör denna datastruktur mycket långsam vid denna typ av text.

BST

Även BST kommer här fungera som en lista, som en form av extremfallet som nämndes i introduktionen för datastrukturerna. Prestandan blir då lika dålig som för SLC. Vilket vi antydde i introduktionen kommer även här, liksom för SLC, insättning och hämtning gå på O(n) i komplexitet.

AVL

AVL-trädet kommer klara av texten bättre eftersom det är självbalanserat. Däremot kommer det bli många rotationer när datan som kommer in är sorterad, vilket gör att insättningen tar tid jämfört med tidigare.

Splay

Splayträdet påverkas inte negativt av att indatan är sorterad, tvärtom visar den på bra prestanda. Då orden hämtas i bokstavsordning kommer nästa ord i ordningen alltid finnas nära på grund av splayningnen. Detta då nästa ord kommer vara den senaste hämtningens barn och därmed bara vara ett steg från roten. Detta leder till mycket snabba söktider.

TEXT4

En ordlista i bokstavsordning, samt att varje upprepas och står fem gånger i rad. Texten består av 3445 ord.

SLC

Liksom vid Text2 kommer SLC gagnas av att det finns upprepningar, men fortfarande tar det lång tid med tanke på att orden kommer i bokstavsordning.

BST

BST uppvisar samma antal jämförelser som SLC, vilket den även gjorde i den tidigare texten där det var bokstavsordning. Detta för att den strukturmässigt fortfarande liknar en lista.

AVL

Liksom i skillnaden mellan Text1 och Text2 gagnas AVL även här av att det finns upprepningar. Antalet jämförelser minskar avsevärt.

Splay

Splay visar här på mycket bra prestanda. Dels gagnas det av att datan kommer sorterad, och dels av att det finns upprepningar. Det går därför mycket snabbt att hämta de olika orden och på väldigt få jämförelser.

TEXT5

En ordlista i bokstavsordning, där själva listan upprepas fyra gånger och därmed står fem gånger. Den består av 3445 ord.

SLC

Insättningen går här på ungefär lika många operationer som för Text 4, då texten har liknande struktur.

BST

I likhet med de tidigare texterna 3 och 4 bildar strukturen formen av en lista, fast med upprepningar. Således blir antalet jämförelser även här lika med de hos BST..

AVL

Inte heller AVL-trädet påverkas nämnvärt av den nya textens struktur

Splay

Då texten inte har varje nästa element som ska hämtas direkt efter det som nyss hämtats kommer inte splayträdet visa på samma goda prestanda som för text 4. Däremot kommer det gå snabbt eftersom orden i sig kommer läggas i alfabetisk ordning samt att upprepningarna gagnar hämtningarna.

Sammanfattning

Sammanfattningsvis kan man se att SLC coh BST påverkas mycket negativt av sorterad data, speciellt utan dubletter. Splayträdet är som snabbast när element som hämtas är på varandra påföljande samt när texten innehåller många upprepningar eller orden ligger nära varandra. Vid helt slumpmässig data ligger splay och AVL nära varandra prestandamässigt, men AVL halkar efter vid helt sorterad data. Intressant är att trots Splay bara gör en bråkdel av jämförelserna jämfört med AVL-trädet i text 3 är det bara ungefär dubbelt gånger så snabbt. Detta beror troligtvis på att splayningarna tar mycket tid. Likadant kan man i de övriga testerna se att AVL inte är mycket långsammare än Splayträdet, även om det gör många fler jämförelser.

Tabeller

	Text1	Text2	Text3	Text4	Text5
Sorted Linked Collection	2212067	491598	17796870	1645263	1661868
Binary Search Tree	74860	45966	17796870	1645263	1661868
AVL tree	52592	31186	111735	40969	41109
Splay tree	47115	13610	13775	5493	17506

Table 1: Antal jämförelser

	Text1	Text2	Text3	Text4	Text5
Sorted Linked Collection	28	9	359	17	17
Binary Search Tree	2	2	350	21	21
AVL tree	2	2	4	1	4
Splay tree	2	1	2	1	3

Table 2: Tid i millisekunder

Referenser

De källor, utöver kurslitteratur etc, som använts är

- http://en.wikipedia.org/wiki/Splay_tree
- http://lcm.csa.iisc.ernet.in/dsa/node93.html
- https://www.cs.usfca.edu/~galles/visualization/SplayTree.html