Analys av ett BSL (Binary Search Tree)

Linus Jansson

1 Höjd av träd vid olika antal noder (osorterad data)

Datan nedan är genererad med min implementation av ett Binary Search Tree. Höjden är tagen med funktionen height(). Värdena är tagna ifrån c++ egna rand() funktion. Nummergeneratorn är seedad med time(NULL), så all data är baserad på tiden.

N	Höjd 1	Höjd 2	Höjd 3	Höjd 4	Höjd 5
100	12	11	11	11	10
1000	22	22	20	21	18
10000	28	29	30	28	27

Table 1: : Höjden av trädet som anges av bst.height()

2 Höjd av träd vid olika antal noder (sorterad data)

Datan nedan är genererad med min implementation av ett Binary Search Tree. Höjden är tagen med funktionen height(). Den sorterade datan är insatt med det största elementet först, och det minsta sist.

N	Höjd 1	Höjd 2	Höjd 3	Höjd 4	Höjd 5
100	99	99	99	99	99
1000	999	999	999	999	999
10000	9999	9999	9999	9999	9999

Table 2: : Höjden av trädet som anges av bst.height()

3 Analys

Höjderna på ett Binary Search Tree är, om man har slumpmässig data, i genomsnitt $2 \ln n$ (Cormen et al,. 2009, Kapitel 12). Detta ger oss att förväntan för våra tre fall (100,1000,10000) är: $\ln 100 = 4.60517018599$, $\ln 1000 = 6.90775527898$ och $\ln 10000 = 9.21034037198$. Detta stämmer inte helt överens med vad som visas i tabell 1, men efter att ha dubbelkollat koden stämmer uträkningarna. Intersant är att de förväntade värdena är ca 3 gånger mindre än det faktiska värdena, vilket visar ett mönster. Detta skulle kunna bero på att antalet tester (5) är för lite, och om fler tester hade körts så hade ett mer korrekt värde visat sig.

För sorterad data så blir höjden $n - 1$ eftersom om all data är i ordning, så kommer den antingen alltid vara större eller mindre än föregående, därför kommer alltid grenen till höger eller alltid till vänster.

Om ett rödsvart träd hade använts för slumpmässig data, så hade höjden minskat, ner mot det teoretiska medelvärdet, eftersom ett rödsvart träd balanserar sig självt, och har aldrig mer än 1 i höjdskillnad mellan 2 vägar från rooten. I det allra flesta fall så hade ett sådant träd varit bättre. I det sorterade datans fall, så hade förbättringen varit ännu större, då sorterad data med 100 element, hade en gren med 999 element, och den andra med 0. Då hade ett rödsvart träd balanserat sig, så att det inte har en skillnad på mer än 1 mellan alla grenar. Efter att trädet hade balanserat sig självt, så hade då höjden för det trädet varit $h <= 2log(n+1)$. I fallet med 1000 element så hade trädet haft en hjöd på $h <= 2log(n+1)$ eller $h <= 2log(1001) \approx 6$. I ett rödsvart träd finns också maximalt $\log 2(n+1)$ svarta noder.

Med en sorterad array som in data, då hade det bästa sättet att sätta in datan varit att lägga in det medianen av arrayn som root, sedan kalla på funktionen rekursivt igen, och göra den undre halvan av arrayn till rootens vänstersida, och den övre delen till rootens högersida. På så sätt får man ett så bra Binary search tree som möjligt.

4 Slutsats

I vanliga fall, så räcker ett vanligt träd, men om datan som ska läggas in, så kan ett rödsvart träd användas för att försäkra att tiden för insert, search, delete, tar mindre tid.

5 Referenslista

1. Cormen, T., Leiserson, C., Rivest, R., Stein, C. (2009). Introduction to algorithms, 3rd edition (3rd ed).