Tim Kerschbaumer & Joakim Berntsson
Laboration 2.2
Grupp 6

Resultat

Jämförelser

st	TEXT1	TEXT2	TEXT3	TEXT4	TEXT5
SLC	2 212 067	491 598	17 796 870	1 645 263	1 661 868
BST	74 860	45 966	17 796 870	1 645 263	1 661 868
AVL	52 592	31 186	111 735	40 969	41 109
ST	47 115	13 610	13 775	5 493	17 506

Exekveringstider

ms	TEXT1	TEXT2	TEXT3	TEXT4	TEXT5
SLC	21	6	271	20	21
BST	2	2	310	24	23
AVL	2	2	5	2	3
ST	3	2	3	1	3

Analys

Text1
Osorterad (ej alfabetisk ordning)
Flera lika ord

Text2
Osorterad
Flera lika ord

Text3
Sorterad
Inga lika ord

Text4
Sorterad
Flera lika ord

Text5
Sorterad
Inga lika ord

SortedLinkedCollection
Detta är den långsammaste och med mest jämförelser. Detta på grund av att den underliggande datastrukturen är väldigt simpel. När programmet skall ta ut ett element (eller lägga till ett nytt) måste den, i värsta fall, gå igenom alla element för att hitta det den letar efter.
1 → 2 → 3 → 4 → …

BinarySearchTree
Denna implementation är bättre än SLC, men endast på de två första texterna på grund av att de två första är osorterade (alltså inte i bokstavsordning). Så de tre sista är lika snabba då ordningen i SLC och BST är lika. Vid sorterade texter kommer elementen ligga likadant som i SLC, som en lista.
1
  2
    3
      4
        …

AVL-Tree
AVL träd ger en markant förbättring på alla texter. Detta då den interna datastrukturen ger utöver sortering även balansering som gör att hämtning/tillägg av element är mycket snabbare. Detta åtgärdad den största nackdelen med BST är då åtgärdat med denna struktur.

SplayTree
Denna implementation ger en ändå större förbättring. Här är den största förbättringen att då man hämtar ett element så splayas det uppåt till roten vilket medför att nästföljande sökning, som kommer vara en sökning på ord efter i alfabetet än i förra sökningen, kommer vara högre upp i trädet (och då även snabbare åtkomst).