Laboration 3

Simon Johansson johsim-0,
Henrik Johansson ehioja-0 ,
Harald Andersson harand-0

Implementation

Vår datastruktur är ett AVL-träd med noder som en nyckel och ett värde. Nyckeln är ordet/frasen i språk A. Nodens värde är översättningen till det andra språket. Operationerna som har implementerats är insert och translate (search).

Vid insert så skapas först en ny nod av datan man vill sätta in (ord/fras i de båda språken). Noden kommer få en nyckel som är ordet/frasen i originalspråket samt ett värde som är ordet/frasen i det andra språket (översättningen).
Sedan sätts noden in i trädet genom jämförelse mot nyckeln rekursivt. På vägen upp i rekursionen så uppdateras höjden för noderna som passerats och om en nod är “ur balans” så roteras den nodens subträd.

Sökning sker iterativt så länge noden man försöker jämföra mot existerar. Om nodens nyckel är lika med sökningens nyckel så returneras nodens värde, annars byter man ut noden mot en av noderna som ligger under, den vänstra om sökningens numeriska värde är lägre än nodens, annars den högra.

Analys

Då det är ett AVL-träd så kommer space-complexity:n att vara O(n).

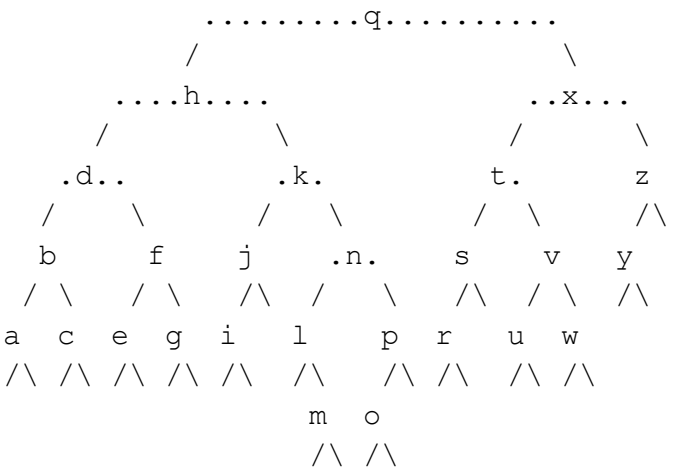
Tidskomplexiteten för insert är O(log n) ty vid varje nod jämförs bara den numeriska nyckeln och höjden på trädet är mindre än 1.44*log_2(n+2)-0.328 då trädet balanseras automatiskt.

Tidskomplexiteten för search ligger även den på O(log n) ty trädet är balanserat och har en maxhöjd på ⌈log2(n) + 2⌉.

Test

Första testet utfördes genom att lägga in bokstäver i en slumpad ordning och sedan använda funktionen för att skriva ut trädet för att kolla så att allt har roterats rätt och höjden är korrekt samt genom att därefter söka igenom trädet och kolla vilka noder som passeras och kolla så att antalet noder passerade inte överskred trädets höjd.

Insättning av alfabetet i följande ordning 'q', 'x', 'z', 'w', 'n', 'j', 'h', 'g', 'r', 'd', 'i', 'y', 'c', 'k', 'f', 'p', 'e', 'v', 'l', 's', 't', 'm', 'b', 'a', 'u', 'o' med samma värde som nyckel (bokstaven) gav utskrift av det slutgiltiga trädet enligt följande:



vilket visar att trädet balanseras och får en maxhöjd av ⌈log2(n) + 2⌉.

En sökning efter ‘m’ samt ‘asdf’ i ovan definierat träd gav följande utskrifter:

```
Balance    Key   Left h     Right h
1          q     5          4
-1         h     3          4
-1         k     2          3
0          n     2          2
-1         l     0          1
0          m     0          0
Translation: m

Balance    Key   Left h     Right h
1          q     5          4
-1         h     3          4
0          d     2          2
0          b     1          1
0          a     0          0
No translation found
```

vilket visar att algoritmen söker igenom trädet i rätt ordning (skrev ut varje nod den passerade).

Det andra testet som utfördes var att lägga in 100000 noder i trädet och därefter kolla trädets höjd så denna inte överskred ⌈log2(n) + 2⌉ samt att sedan göra flera olika sökningar på både saker som fanns i trädet och saker som inte fanns i trädet och räkna antalet steg som krävdes för att hitta noden respektive gå igenom hela trädet utan att hitta något och jämföra antalet steg mot den uppskattade tiden för algoritmen.

Noderna som sattes in i trädet fick nycklar i form av ett nummer från 0 till och med 99999 och samma värde.

Körning gav utskriften:

```
Nodes in tree: 100000
Height of tree: 19
Word or sentence to translate: 1

Translation: 1
Steps: 16

Word or sentence to translate: 0

Translation: 0
Steps: 17

Word or sentence to translate: 99999

Translation: 99999
Steps: 19

Word or sentence to translate: 100000

No translation found
Steps: 18

Word or sentence to translate: 65536

Translation: 65536
Steps: 14

Word or sentence to translate: a

No translation found
Steps: 20

Word or sentence to translate: 037hej

No translation found
Steps: 18
```

Vilket visar att trädets höjd är 19 vilket stämmer överens med $\lceil \log_2(n) + 2 \rceil = \lceil \log_2(100000) + 2 \rceil = \lceil 16.61 + 2 \rceil = 19$ samt att ingen körning tog fler steg än 20 (vilket var vid sökning av

element som inte existerade som tog den längsta vägen genom trädet). Ty log2(n) ≈ 16.61 och maximala antalet steg som en sökning i trädet kan ta är 20 så betyder det att körningstiden för algoritmen är O(log2(n)).