Jimmy Åkesson   
Olle Olsson

Linus Forsberg

*Vår kod för att iterera igenom en textfil och ta bort alla char ’e’:*

import java.io.IOException;

import java.io.UnsupportedEncodingException;

import java.nio.file.Files;

import java.nio.file.Paths;

public class Algoritm\_Lab1 {

public static String filterOutChar(String in, char charToGetRidOf) {

String out = "";

for (int i = 0; i < in.length(); i++) {

char c = in.charAt(i);

if (c != charToGetRidOf) {

out += c;

}

}

return out;

}

public static void main(String[] args) throws IOException {

String s = " ";

byte[] encoded = Files.readAllBytes(Paths.get("text/bible-en.txt"));

s = new String(encoded, "UTF8");

for (int i = 1000; i < s.length(); i = i \* 2) {

String str = s.substring(0, i);

long before = System.currentTimeMillis();

String t = filterOutChar(str, 'e');

long after = System.currentTimeMillis();

System.out.print(i + " chars");

System.out.println(" iterated in time " + (after - before) / 1000.0);

}

//Test av kontroll N

String str = s.substring(0, 200000);

**long** before = System.*currentTimeMillis*();

String t = *filterOutChar*(str, 'e');

**long** after = System.*currentTimeMillis*();

System.***out***.print(200000 + " chars");

System.***out***.println(" iterated in time " + (after - before) / 1000.0);

}

}

**Hypotes:**

Tidskomplexitet: T(N) = tC \* N  
t = tid   
c = sökning efter charToGetRidOf   
N = längden på sträng (in.length)

Linjär - eftersom den itererar genom en hel sträng. Ju längre strängen är desto längre tid tar det.

T(2N)/T(N) kallas för *’Growth Ratio’*

*Test med Bibel-dokument , N = antal chars, T(N) = tid för iteration genom chars*

|  |  |  |  |
| --- | --- | --- | --- |
| N | T(N) | T(2N) / T(N) | c |
| 1000 | 0,001 | 4,0 | 0,0000000010 |
| 2000 | 0,004 | 3,3 | 0,0000000010 |
| 4000 | 0,013 | 1,9 | 0,0000000008 |
| 8000 | 0,025 | 3,6 | 0,0000000004 |
| 16000 | 0,091 | 4,2 | 0,0000000004 |
| 32000 | 0,38 | 3,4 | 0,0000000004 |
| 64000 | 1,306 | 4,0 | 0,0000000003 |
| 128000 | 5,221 | 4,6 | 0,0000000003 |
| 256000 | 23,815 | 4,0 | 0,0000000004 |
| 512000 | 95,142 | 4,7 | 0,0000000004 |
| 1024000 | 450,121 | 4,9 | 0,0000000004 |
| 2048000 | 2227,72 |  | 0,0000000005 |
| Test N |  |  |  |
| 200000 | 14,746 |  | 0,0000000004 |

*Graf som visar ökningen i sekunder förhållande till Chars algoritmen går genom*

Baserat på testet ser vi att det inte är en linjär tidskomplexitet

**Uträkning för formel T(N) = c \* Nb för att se vilken tidskomplexitet koden har:**  
T(N) = c \* NbT(2N) = c \* (2N)b

T(2N) = c \* 2b \* NbT(2N) = 2b \* c \* Nb

T(2N) = 2b \* T(N)

T(2N)/T(N) = 2b \* T(N)/T(N)

T(N) = 2b

T(N) = 4  
  
4 = 2b

b = 2 *1 = linjär, 2 = kvadrat, 3 = kubisk*

T(N) = c \* N2

Tidkomplexiteten är kvadratisk och inte linjär. Detta beror på att String inte går att ändra utan kräver att man kopierar den gamla strängen till en ny sträng varje gång strängen blir en char längre (alla gånger en char i strängen inte är ’e’)

Test av formel baserat på testet och ett kontroll N:

Tabell

23,815 = c \* 256 0002

23,815 / 256 0002 = c

c = 0,0000000004

Tabell

0,38 = c \* 32 0002

0,38 / 32 0002 = c

c = 0,0000000004

Valt nytt N

14,746 = c 200 0002

14,746 / 200 0002 = c

c = 0,0000000004