Länkade Listor

Viktor Björkén

September 2022

1 Introduktion

I denna uppgift ska vi jobba med olika ihoplänkade strukturer. Vi kommer börja med den metod som handlar om länkade listor. Vi kommer att göra benchmarks för att se hur tidskomplexitet skiller sig åt när vi kör olika funktioner inom ämnet för länkade listor samt stackar. Vi kommer dessutom att göra linknande benchmarks fast med arrayer istället för länkade listor och jämföra dessas tidskomplexitet.

2 ”append” funktionen

I denna uppgift handlar det mäta tiden över en append funktion. Vi kommer att anvädna olika implemetaioner och metoder och jämföra dessa med varandra och dra våra slutsatser. vi kommer att böra med att använda oss utav länkade listor och sedan gå vidare till arrayer. Vi är inte ute efter exakta mätvärden utan mer hur tiden förendras beroede på storleken på listan eller arrayen som vi ändrar.

2.1 Länkade listor

I denna deluppgift ska vi med hjälp av tidigare kunskap benchmarka våran append funktion. Detta är en kodsnipp som vi fått i uppgiften där vi endast behöver kompletera andra delar för att köra.

```
    public class LinkedList {
int head;
LinkedList tail;

public LinkedList(int item, LinkedList list ) {
    head = item;
    tail = list;
}
}
```

Vi börjar med att skapa våran länkade lista och implementerar sedan våran ”append” funktion.

```
    public void append(LinkedList b) {
    LinkedList nxt = this;
    while (nxt.tail != null) {
        nxt = nxt.tail;
    }
    nxt.tail = b;
}
```

När vi kör denna benchmark kommer vi att öka våra värden på den första listan med en konstan storlek på den andra. Detta ger oss en nästan konstant funktion. I nästa exempel ska vi göra en liknande undersökning bara att denna gång kommer vi att swicha så att storkleken på den första listan är konstant medans den andra ökar. Då får vi föjande. Vi kan anta det tar O(n), att hitta rätt värde och sedan O(1) för att uföra operationen, Därför får vi en tidskomplexitet på O(n).

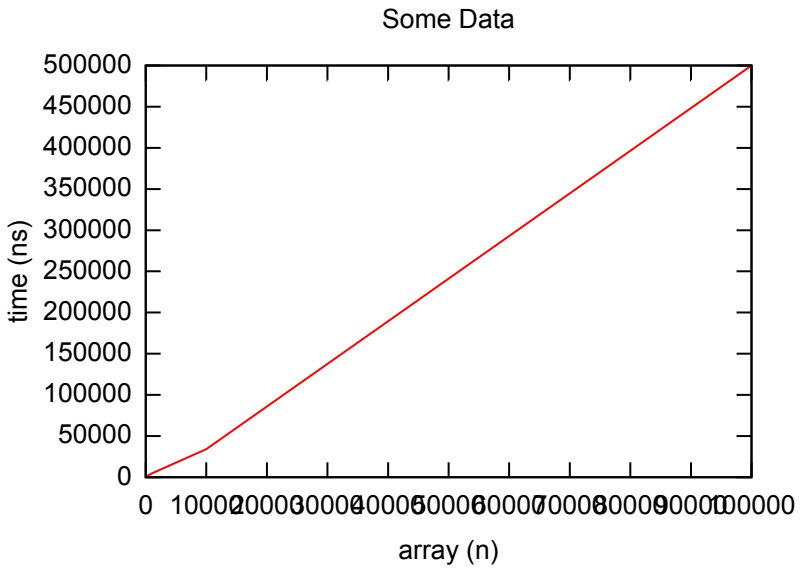

Figure 1: Länkad lista med konstant B

Här ser vi samma tidskomplexitet

$$O(n)$$

2.2 Array

I denna del uppgift ska vi istället för att använda länkade listor anävnda samma opeationer men med arrayer istället. Detta har lite annat funktionalitet men kommer att kunna utföra samma operation som tidigare, vi kommer sedan att jämför de olika tidskomplexiteterna som vi får i de olika delarna.

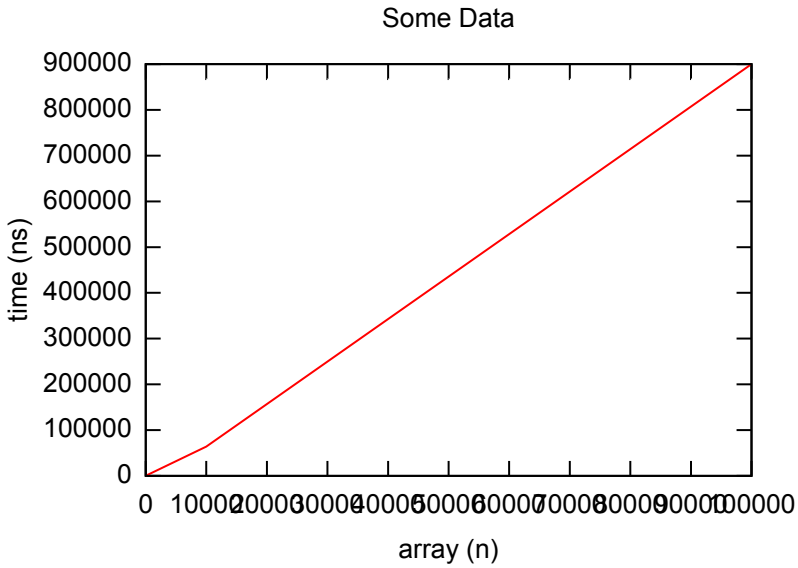

Figure 2: Länkad lista med konstant B

När vi gör benchmark på denna så får vi samma tidskomplexitet dvs

$$O(n)$$

```
int[] array = new int[i + k];
for(int x = 0; x < i; x++){
    array[x] = a[x];
}
int y = 0;
for(int x = a.length; x < array.length; x++){
    array[x] = b[y];
    y++;
}
```

Detta är mycket intressant då vi kanske hade förväntat oss en annan tidskomplexitet med arrayer till skillnad från länkade listor. Dock så är den array

baserad funktionen är långsammare än den länkade listor, men de båda är O(n).

Detta kan bero bland annat på att när vi kör den länkade list append funktionen behöver vi endast skapa en länk mellan första elementet och lista b, när vi istället måste gå genom n antal element innan vi lägger på lista b får vi

$$O(n)$$

3 Stacken

Nu ska vi som i en tidigare uppgift använda oss utav en stack, Vi kan skriva en följande stack på följande sätt. Stackens datastruktur kommer att inneha en push och en pop funktion som kommer göra det möjligt att se skillanden mellan en länkad lista implention samt en array impplention.

```
        LinkedList stack ;
public void Stack () {
    stack = null ;
}

    public void push ( int item ) {
    stack = new LinkedList ( item , stack );
}
public int pop () throws Exception {
if ( stack == null ) {
    throw new Exception ( " pop from empty stack " );
}
int ret = stack . head ;
stack = stack . tail ;
return ret ;
}
```

Där vi har en push och en pop funktion som vi kan använda oss utav med våra länkade listor.

Skillanden i exekveringstid mellan array implentationen och den länkade listan, kan bland annat ses med tanke på att de olika cache localityn i de olika metoderna. I array implentationen kommer vi att ha bättre cache locality vilket kan göra att man får bättre exekveringstid. Detta är något som vi intr direkt såg i min rapport men det kan ha att göra med andra faktorer såsom fel i koden, som gjort att datorn memoriserat olika arrays operation som gör att vi får snabbare tid än vad vi borde. Dock borde detta göra så att det blir snabbare att stoppa in nya värden via den länkade listan.