Uppgift 5

Simon Moradbakti

HT 2024

1 Introduktion

För den här uppgiften skulle man hitta ett sätt att implementera en länkad lista och samtidigt jämföra listan med enkla arrayer. Det kommer även utföras så kallade ”Benchmarks” på dessa för att mäta och se deras tidskomplexitet.

2 Task 1 - Länkad lista och dess benchmarks

I den första tasken skulle vi skapa en länkad lista med hjälp av den givna koden:

```
LinkedList(int n) {
Cell last = null;
for (int i = 0; i < n; i++) {
last = new Cell(i, last);
}
first = last;
}
```

och sedan ta fram tidskomplexiteten. Denna kod ska också anpassas så append funktionen är med. Delar av koden ser ut som följande:

```
public class LinkedLinst {
       public LinkedLinst(int n) {
        Cell last = null;
        for (int i = 0; i < n; i++) {
            last = new Cell(i, last);
        }
        first = last;
    }
    public void append(LinkedLinst other) {
        if (first == null) {
            first = other.first;
            return;
        }
    }
```

När vi kör denna benchmark kommer vi att öka storleken på den första listan medan den andra listan har en fast storlek. Det gör att man får en nästan konstant funktion. I nästa exempel ska vi genomföra en liknande undersökning, fast nu d byter vi så att storleken på den första listan är konstant och den andra listan ökar.

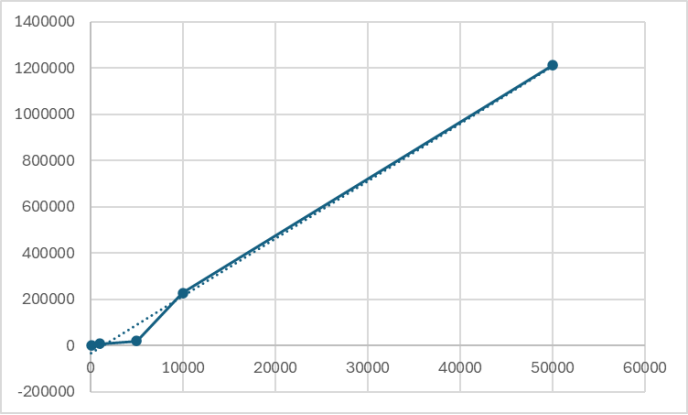

Figure 1: A varierad B fast

Vi ser ganska tydligt, med trendlinjen som referens att detta är en tidskomplexitet av typ O(n). Man kan anta att det tar just O(n) att hitta rätt värde fast O(1) att utföra själva operationen. Om man gör tvärt om, dvs att B är varierad storlek och A är fast fick man fram följande graf på excel:

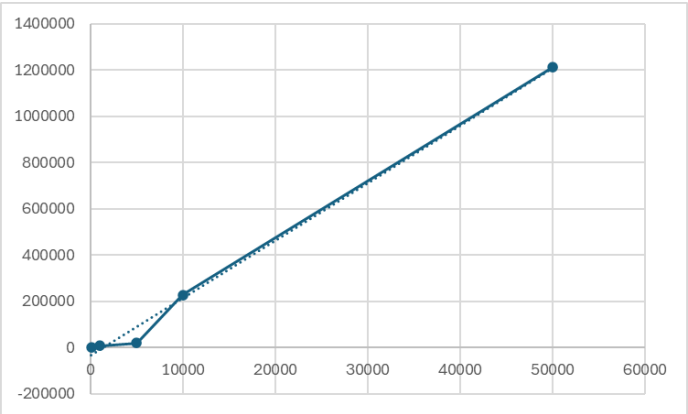

Figure 2: B varierad A fast

Man ser alltså att det blir liknande och samma tidskomplexitet.

3 Task 2 - Jämförelse med array

Här ska samma sak göras fast med arrayer istället. Skribenten valde att koda och se resultatet.

```
public int[] appendArrays(int[] a, int[] b) {
    int[] newArray = new int[a.length + b.length];
    for (int i = 0; i < a.length; i++) {
        newArray[i] = a[i];
    }
    for (int i = 0; i < b.length; i++) {
        newArray[a.length + i] = b[i];
    }
    return newArray;
}
```

Resultatet av den exekverade koden gav liknande resultat:

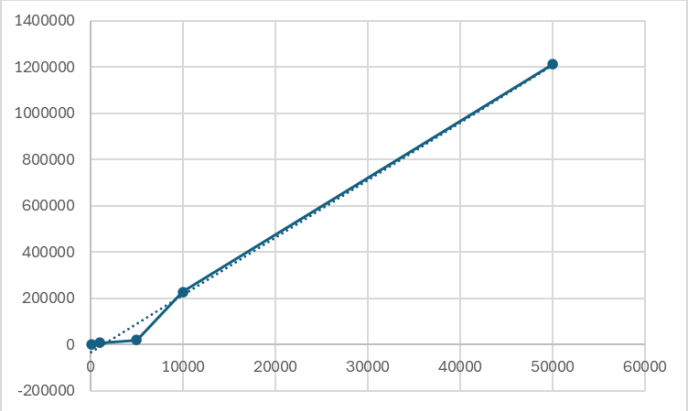

Figure 3: tidskomplexitet med en array

Vi kan alltså se att det blir samma O(n).Den betydelse som det här får är att vi måste använda oss av nya arrayer då vi nu har arrayer av en fixerad storlek. Vi måste också kopiera in våra svar från en array till en annan samt returnera arrayer. Detta påverkar vår benchmark på det sättet att tidskomplexiteteten blir beroende och därför växer linjärt beroende på hur arrayen ser ut. Den nya tidskomplexiteten skulle därför bli O(x+y).

4 Task 3 - Jämförelse med stack

När det gäller att implementera en stack är en länkad lista i principe den vanligaste datatypen man kan välja och den har såklart både för och nackdelar. En av de största fördelarna med en länkad lista att den har dynamisk storlek,

dvs den växer eller krymper efter behov, utan att fixeras av en initial storlek, vilket förhindrar överflödssituationer. Det betyder dessutom att minnesallokeringen endast händer när element läggs till som vilket leder till en mer effektiv användning av minnet. Till skillnad med arrayer som kan kräva att mycket minne måste vara tillgänligt. En länkad lista slipper också omallokering som kan vara en jobbig process då man även har en array-baserad stack, som gör att man måste allokera mer minne och kopiera befintliga element. Å andra sidan har länkade listor även nackdelar, såsom minnesöverhuvud. Detta då precis varje nod i listan kräver extra minne för en pekare till nästa nod som kan leda till högre minnesanvändning speciellt särskilt med ett stort antal noder. Cahen kan också påverka eftersom element i arrayer lagras, vilket gör att dom kan bli lite mer cachevänliga och därför mycket snabbare att komma åt. Medans noder i en länkad lista kan vara lite överallt då dom är spridda i minnet. Det kan resultera i flera cache-missar. Implementeringen av en länkad lista för en stack är också mer komplex eller jobbigare att utföra och kräver därför en noggrann hantering av pekare. Det kan öka risken för fel om det inte görs korrekt. För körningstid är push-operationen i en array-implementering oftast i allafall O(1) om utrymme finns. Finns det inte måste en omallokering ske och då blir den o(n). I en länkad lista är push-operationen alltid O(1) och inget annat utom pop-operationen, detta pga skapande av nya noder.