Länkade Listor

Mattias Sandberg

21 Sep 2022

Introduktion

Hittills har vi bara arbetat med primitiva datastrukturer och arrays. Det finns mer komplicerade datastrukturer som är länkade med varandra med hjälp av pekare. I denna uppgift kommer vi konstruera simpla länkade datastrukturer och ta reda på dess struktur och inte fokusera på vad det representerar. Denna uppgift kommer att likna den första uppgiften fast med länkade datastrukturer. Nedan kommer en bild från hemsidan "https://www.quora.com/What-is-a-linked-list-in-C-language" som förklarar Länkade listor och pekare:

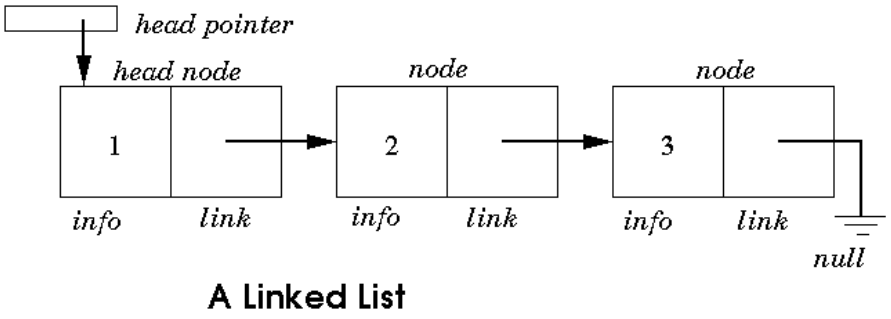

Figure 1: Simpliferad bild över länkade listor och pekare

append operation

I första uppgiften ska jag konstruera en benchmark som mäter exekveringstiden för append operation. Detta ska ge oss en generell uppfattning om hur exekveringstider beter sig beroende på varierande mängder i listor samt fixade listor när länkade listor är inblandade. I denna uppgift så ska vi variera storleken på den första länkade listan (a) och fästa den till en länkad lista som har en bestämd storlek. När jag representerar mina mätvärden så är fokuset på hur programmets exekveringstid varierar med utökningen av lista (a) och inte den exakta exekveringstiden. Därmed kommer jag att

konstruera en graf i Gnuplot med Y-axeln som tidsaxeln och X-axeln som den länkade listan (a). Detta kommer gör att vi får en bra överblick över tidskomplexiteten:

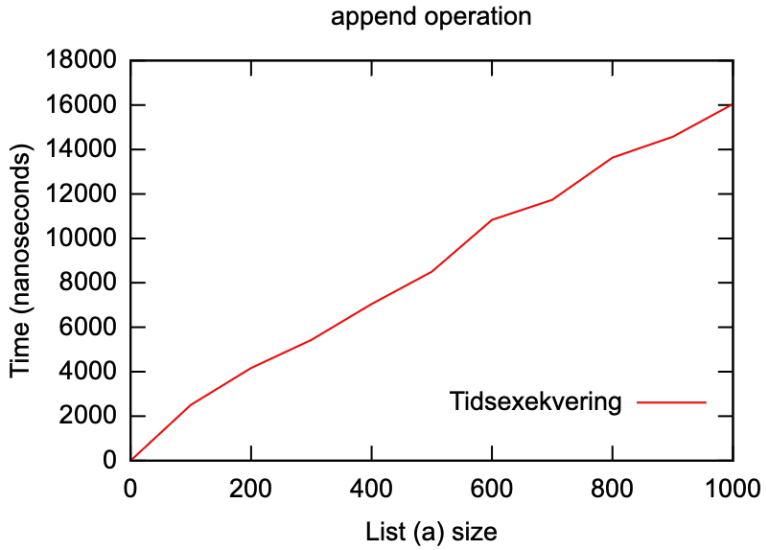

Figure 2: Graf över append operation med länkade listor

Enligt grafen ser vi att tidskomplexiteten är Ordo O(n) vilket var min hypotes då tidskomplexiteten för att hitta rätt position är O(n). Och tidskomplexiteten är O(1) för att genomföra operationen när positionen väll är funnen. Detta betyder att den totala tidskomplexiteten blir O(n) vilket min graf representerar. I andra testomgången gjorde jag samma sak fast jag lät (a) vara en fast storlek och (b) varierande, resultatet som jag fick var att tidskomplexiteten O(1) vilket är rimligt.

I nästa uppgift ska jag benchmarka en länkad lista mot motsvarande operation med hjälp av en array. För att fästa ihop två arrays måste vi först allokera en ny array som är stor nog för att kunna hålla alla värden och sedan kopiera värden från det båda arraysen till den nya. När vi gör dessa operationer så kommer tiden att allokera den nya arrayen behövas räknas in till den slutliga tidskomplexiteten. I denna uppgift ska vi räkna ut hur exekveringstiden förändras med varierande första och andra array och presentera det på ett prydligt sätt samt diskutera resultatet. Jag vill även

jämföra detta mot uppgiften tidigare och se skillnaden i tidskomplexitet och exekveringstid. Nedan kommer en bit av min kod där jag lägger ihop array (a) och array (b) och beräknar exekveringstiden. Därefter kommer grafen över exekveringstiden i förhållande till storlek på arrayerna:

```
  // Fäster länkade listan b till a
for(int n = 0; n < 1000; n++)
{
    t0 = System.nanoTime();
    int[] baarray = new int[p + m];
    // Kopierar över a
    for(int y = 0; y < p; y++)
    {
        baarray[y] = AArray[y];
    }
     // Kopierar över b
    int BIndx = 0;
    for(int y = AArray.length; y < baarray.length; y++)
    {
        baarray[y] = BArray[BIndx];
        BIndx++;
    }
    t1 = System.nanoTime();
    RunTime += (t1-t0);
}
```

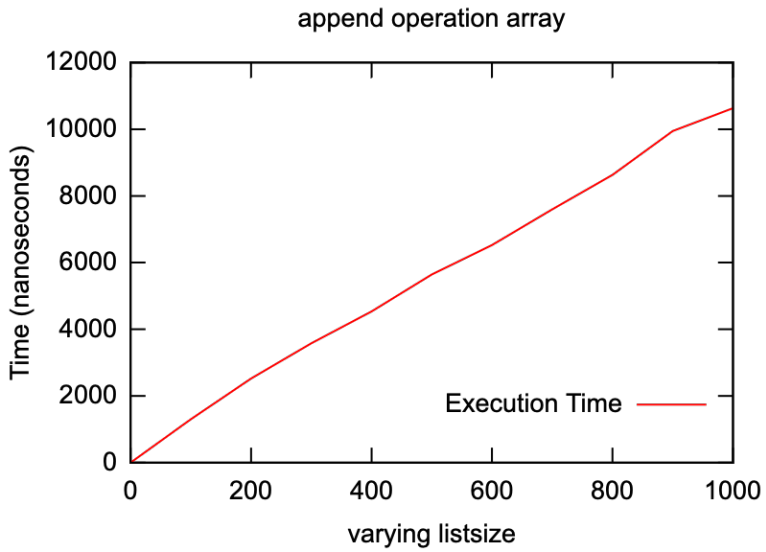

Figure 3: Graf över append operation med länkade listor (arrays)

Exekveringstiden jämfört med det andra programmet är något snabbare vilket vill säga att det är ett effektivare program i det fallet jag testar det i. Däremot så är tidskomplexiteten densamma som i det föregående programmet, dvs Ordo O(n). Detta betyder att beroende på hur man sätter upp testerna och hur programmen exekveras så kan exekveringstiderna bli annorlunda enligt min analys. Det jag menar är att om man appendar "fäster" (b) som är konstant till (a) som varierar så är arrayen långsammare om b är markant större. Detta beror på att vid den länkade listan så stannar loopen efter n element eftersom den enbart länkar vidare till b tillskillnad från arrayen där programmet även måste kopiera b. Detta gör att tidskomplexiteten är O(n) men i mitt fall för mina program så är append operation array något snabbare.

a stack

I en av det föregående uppgifterna som jag har gjort konstruerade jag en dynamisk stack som ändrar storlek beroende på om du pushar eller poppar objekt. Nu ska vi använda våran version av den länkade listan och implementera den dynamiska stack-strukturen med det ordinarie push och pop-operationer. Nedan kommer koden från min dynamiska stack men implementation av länkad lista:

```
          LankadLista stack ;
public void Stack ()
{
    stack = null ;
}
    public void push ( int item )
    {
    stack = new LankadLista ( item , stack );
    }
public int pop () throws Exception
{
if ( stack == null )
{
    throw new Exception ( " Poppa från tom stack " );
}
int x = stack . head ;
stack = stack . tail ;
return x ;
}
```

Utan att följa pdf:en slaviskt så tänkte jag diskutera några för och nackdelar med Länkade listor jämfört med Arrays. Fördelarna med Länkade listor är att dom har en dynamisk storlek jämfört med arrays och att dom har en konstruktion som förenklar insättning och radering. Men givetvis finns det även nackdelar och några av det nackdelar som jag bedömer väsentliga är att random access inte är tillåtet eftersom vi måste finna elementen genom att sekventiellt gå igenom listan från den första noden. Detta gör att binary serach ej blir tillgängligt vilket vi i tidigare uppgifter har sett kan vara mycket effektiv och icke kostsam. Till sist skulle jag även vilja pointera att länkade listor kräver extra minnesutrymme då vi implementerar en pekare som är elementär för varje element i listan. Även fast detta kanske inte drabbar oss i denna uppgift är det en viktig tanke för andra program. Jag ville lyfta dessa saker då jag uppfattade uppgiften som att man skulle lära sig och analysera länkade listor.