Dubbellänkad lista

Adalet Adiljan

Fall 2023

Introduktion

I denna uppgift har vi fått möjligheten att arbeta med primitiva datastrukturer och arrayer. Denna gång kommer vi att medföra en bättre och effektivare variant av en länkad lista. Med hjälp av en ytterliggare referens har vi en möjlighet att skapa en bredare uppsättning operationer som kommer ge oss en oumbärligt verktyg för vissa tillämpningar. I vår resultatavsnitt redogörs effektivteten av speciella metoder mellan en enkelriktad- och dubbelriktad lista.

Dubbellänkad lista

En dubbelriktad lista är en variant av en länkad datastruktur som används för att lagra och hantera listor av element. I varje nod har vi två referenspekare vars den ena, döpt till ”next” - pekar framåt åt höger, och den andra ”prev”- pekar åt vänster till det föregående elementet. Den dubbelriktade egenskapen är fördelaktig för att utföra borttagning av element som råkar befinna sig i mitten av listan eller möjliggöra traversering baklänges. Den enkelriktade listan är begränsad till detta då vi måste jobba oss från att traversera från början till slut oavsett.

Metod

Dubbellänkade listans struktur

I vår dubbelriktade lista har vi lagt till ett ytterliggare fält och referenspekare, där vi nu innehar en ”prev”- pekare som pekar på den föregående noden. Listans hjälpmetoder kommer nu att skilja sig i en viss grad från vår länkade lista. Huvudsakliga skillnader ligger således i hur listorna hanterar sina referenser och navigering mellan noderna. I vår dubbelriktade implementation har vi fördelen att enkelt gå bakåt i listan, vilket kommer att effektivisera bl.a. vår ”unlink” operation. Med hjälp av vår ”prev” - och next-pekare möjliggör detta oss att skapa en effektiv ”unlink” metod som

kopplar bort en nod med den givna datan. Detta då funktionen kommer oavsett vad att behöva traversera igenom från början och framåt i syfte för att hitta och avlägsna den eftersökta cellen. Pga. dess dubbla referens kommer detta att kräva mer minnesuttrymme per nod samt innebära en komplexare kod för att hantera och uppdatera de två riktningarna medan den enkelriktade listan är mer platsbesparande.

```
// Unlink för vår double linkedlist:
public void unlink(Node node) {
    (:) // kollar om listan är tom
    }
    if(node.prev != null){
    node.prev.next = node.next;
    } else {
        head = node.next;
    }
    if(node.next != null){
        node.next.prev = node.prev;
    }
    listsize--;
    (:)
```

Avlägsning och insättning

Syftet med vår bench metod är att utföra prestandamätningen hos de respektive listorna enligt denna stegvisa ordning:

1. Lägga till: En cell läggs till i listan genom att skapa en ny ”Node” med den angivna datan associerat till den noden. Om listan är tom, sätts både head och tail till den nya noden. Om listan innehåller minst ett element sätts den nya nodens ’next’ till att peka på head och head’s ”prev” sätts till att peka på den nya noden. Därefter uppdateras vår head-pekare för att peka på den nya noden och storleken på listan inkrementeras till +1 för varje gång.

2. Avlägsna cellen: Beroende på den angivna datan, kan vi traversera från både höger och vänster för att hitta det eftersökta elementet. Om cellen har en föregående ”prev” nod, ändras nodens ”next” -pekare för att peka på nästa nod efter den nuvarande cellen. Råkar cellen ha en efterföljande ”next” nod ändras noden och prev att peka på den föregående ”prev” för den nuvarande cellen. Efter att ”head”-pekaren har uppdaterats, dekrementerar vi listan med -1.

3. infoga cellen i början: Infogningen sker i allra första början av listan. Om listan inte är tom kommer prev-pekaren för den befintliga noden

att peka på den nya noden och nodens next att peka på head. Råkar listan inte vara tom, sätts den nya noden som listans både ”head” och ”tail”, därpå som listans storlek och dessa pekare uppdateras.

De förbestämda cellerna genererades med slumpvalda heltalsindex mellan 0-1000, vilka lagras i en array som vi kommer åt för att välja ut celler från vår ”cellArray” för att utföra vår benchmark. Därefter startas tiden och fortsätter tills vår loop har slutförts. Efter att ha stegvis utfört operationerna i vår for-loop, upprepas sekvensen ”k”-gånger, vilket vi har ställt in för 1000 omgångar varpå som tiden avslutas. Den slutgiltiga resultatet av den konverterade tiden till ms skrivs sedan ut och upprepar samma procedur för stegvis ökande lisstorlek.

Resultat

Tabell

storlek	**enkelriktad**	**dubbelriktad**
100	2.7	0.15
200	0.41	0.15
400	0.76	0.05
800	1.6	0.05
1600	3.2	0.05
3200	9.9	0.05
6400	15.4	0.05
12800	29.8	0.06
25600	65.5	0.17
51200	131.1	0.82
102400	257.8	0.77
204800	515.9	0.22
409600	1053.1	0.47

Table 1: Tiden är mätt i millisekunder. Första kolumnnen motsvarar listans storlek i stigande ordning medan den andra och tredje kolumnen visar körtiden för de respektive implementationerna.

Vi ser att körtiden skiljer sig betydligt mellan dessa två särskilt för ju längre listan blir. Dubbelriktade implementationen påvisar då högre effektivitet jämfört med den enkelriktade listan, med en konstant tidskomplexitet på

$$O(1)$$

där n motsvarar antalet noder i listan. Ju större listan blir börjar den

enkelriktade listan att närma sig en komplexitet som efterliknar en

$$O(n)$$

istället, vilket. Unlink-metoden för den dubbelriktade medför konstant komplexitet tillskillnad från den enkelriktade eftersom att vi navigerar listan med båda pekare, vilket gör att vi direkt kan hoppa till den nod som eftersöks. När det gäller den länkade listan så efterliknar komplexiteten en linjär sökning, eftersom att vi med vår okända föregående pekare, behöver börja om från listan för att hitta den eftersökta noden.

Implementationen av insertFirst och add-funktion har samma syfte, dvs. att lägga till elementet i början vilket båda medför en konstant tidskomplexitet. Sammantaget påvisar både unlink- och insertFirst metoden för dubbellistan att ha en konstant komplexitet i dess operation. Trots att körtiden påvisar att den dubbla listan sammantaget är mer effektiv än den simpla listan kan den dock vara ett bättre alternativ om vi vill minimera minnesanvädningen och förenkla implementationen. För att erhålla en större flexibilitet kan dubbelriktade listor vara bättre alternativ men det kommer att dock kosta minnesanvändningen och komplexare operationer.