Datatyp jämförelse för syftet kalkylprogram

Vincent Johansson
Umeå universitet
dv14vjn@cs.umu.se

DV2: Algoritmer och problemlösning

Innehållsförteckning

Inledning	1
Kriterier	1
Representationer	1
Utvärdering	3
Slutsats	4
Källförteckning	5

Inledning

I denna rapport kommer tre olika datatyper som kan användas vid implementation av ett kalkylprogram att jämföras. Rad och kolumn kommer att representeras av heltal och cellerna ska kunna innehålla olika datatyper vilket benämns som en heterogen datatyp. Rapporten kommer först beröra de kriterier som tagits fram i syfte att objektivt kunna jämföra det olika datatyperna och gå in i detalj hur kriterierna mäts samt varför just dessa kriterier valts. När dessa förtydligats kommer de tre utvalda datatyperna att presenteras och förklaras innan en objektiv utvärdering där de jämförs utifrån kriterierna. Avslutningsvis presenteras slutsatsen som dragits utifrån utvärderingen, och ett förslag av datatyp för kalkylprogrammet väljs ut.

Kriterier

Under ett gruppmöte togs olika förslag av kriterier fram och utifrån dessa har de tre som ansågs bäst matcha syftet med denna rapport har valts ut.

- Tidskomplexitet vid insättning, hur lång tid tar det att skapa ett nytt element i den valda datatypen? Mäts genom att jämföra tidsåtgång för insättningsoperationen efter funktionsanrop.
- Tidskomplexitet vid sökning, hur lång tid tar det att söka efter ett specifikt element i den valda datatypen? Mäts genom att jämföra tidsåtgång för sökningsoperationen efter funktionsanrop.
- Rumskomplexitet, hur påverkas minnesbehovet av antalet element i kalkylbladet? Mäts genom att kontrollera minnesanvändningen för ett element i kalkylbladet.

Representationer

- Oordnad länkad lista

Definitionen av en länkad lista är en linjär följd av noder/celler (Janlert, 2000). Dessa noder kan bestå av olika mängder fält, i detta fall vid implementation av ett kalkylblad kommer noderna behöva bestå av fyra fält. Ett fält vardera för att lagra vilken rad och kolumn för värdet, ett fält för själva värdet och sedan ett adressfält som kommer lagra minnesadressen till nästkommande nod i den länkade listan. En fördel med denna typen av implementation är att insättning alltid kommer att vara $\Theta(1)$, medan en nackdel är en länkad lista enbart kan traverseras från start till slut vilket innebär att vid sökning kan i värsta fall alla noder i listan behövas gå igenom innan rätt nod hittas. Se bild 1 för en förklaring i bild av hur en länkad lista fungerar. Notera dock att listan i bilden är ordnad, men för enkelhetensskull kan vi förklara det genom att värdena lagts till i ordning direkt.

Om nästa värde som läggs till skulle vara i rad 2 och kolumn 2, hade nästa nod bestått av [1 , 1, värde, adress]. Se bild 1 för en visuell representation av en länkad lista.

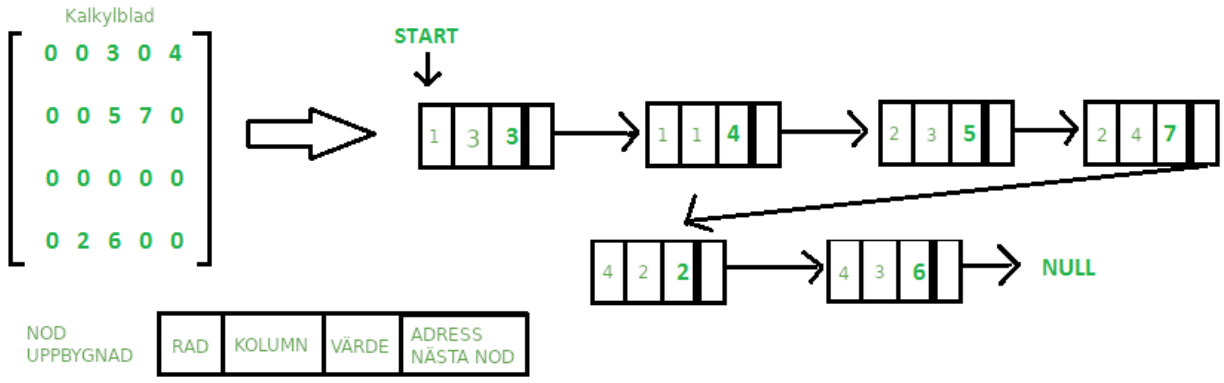
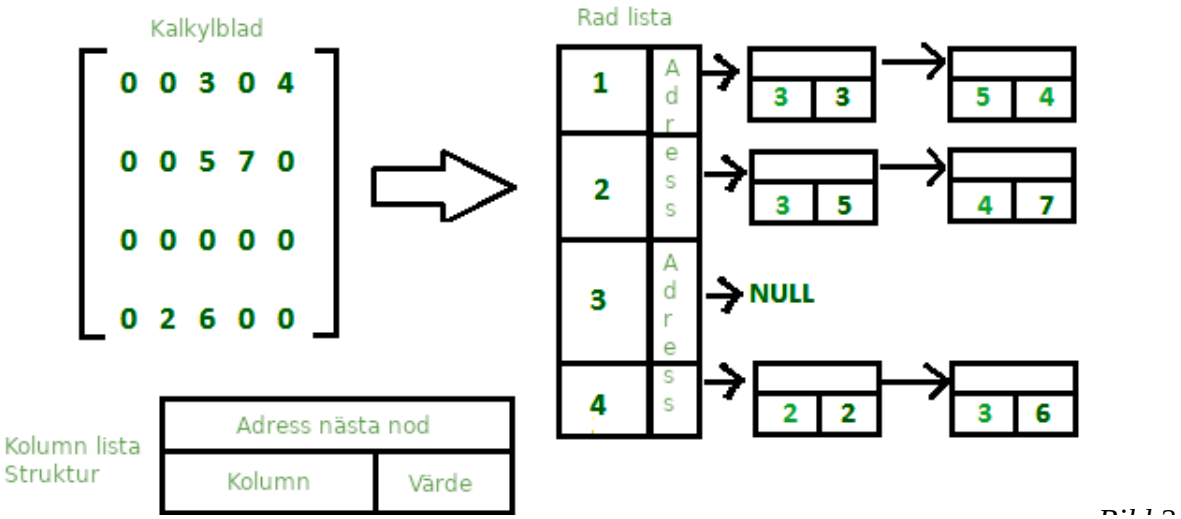


Bild 1: Visuell representation av en länkad lista.

- Lista av listor

Vid denna typ av implementation skapas först en ordnad länkad lista som representerar raderna i kalkylbladet. Denna lista kommer bestå av noder med tre fält, två adressfält och ett fält som specificerar vilken rad den noden hanterar. Ett av adressfälten pekar till nästa nod i listan som kommer represtera nästa rad i kalkylbladet, det andra adressfältet pekar till första elementet i en annan lista som lagrar vilket värde som är sparat i vilken kolumn på raden (GeeksforGeeks, 2021). Kolumn listorna är även dessa ordnade länkade listor och består av tre fält. Ett lagrar värdet, ett lagrar vilken kolumn i kalkylbladet värdet är sparat på och det sista fältet är adressfältet som pekar till nästa nod i listan. Se bild 2 för en visuell representation av hur en lista av listor fungerar.



- Ordnad dubbellänkad lista

En dubellänkad lista fungerar snarlikt den länkade listan som tidigare togs upp, skillnaden är att varje nod innehåller ett extra adressfält (GeeksforGeeks, 2021) samt att listan hålls sorterad till

skillnad från oordnad där rad och kolumnindex ligger i den ordning de lagts till i listan. Det extra adressfältet pekar till föregående nod i listan. Fördelen med denna typ av lista är att det är enklare att göra en ny insättning mitt i listan, för att hålla listan ordnad. Man kan även traversera genom listan både bakåt och framåt vilket underlättar sökning av värden i listan. Det kan även underlätta vid insättning, om man redan befinner sig i en nod någonstans i listan kan man kontrollera om värdet som ska sättas in ska vara på en rad före eller efter den nuvarande nodens rad, och då traversera i den riktning där det nya värdet ska sättas in. Antingen tillbaka i listan för lägre radnummer, eller framåt för högre radnummer. Se bild 3 för en visuell representation av hur en dubellänkad lista fungerar.

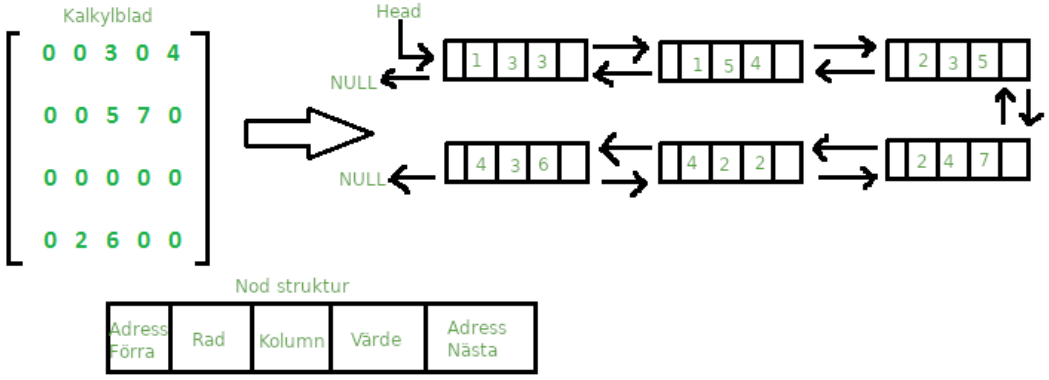


Bild 3: Visuell representation av ordnad dubbellänkad lista.

Utvärdering

I denna del av rapporten kommer vi utvärdera representationerna med hjälp av de tre utvalda kriterierna, tidsåtgång vid insättning och sökning och rumskomplexitet för varje representation. I figur 1 kan vi se en jämförelse mellan de olika representationerna.

Figur 1: Tids- och rumskomplexitet för de olika representationerna

Representation	Insättning	Sökning	Rumskomplexitet
Oordnad länkad lista	O(1)	O(n)	O(n)
Lista av listor	O(n+m)	O(n+m)	O(n)
Ordnad dubbellänkad lista	O(n)	O(n)	O(n)

För den oordnade listan kommer insättningen alltid att utföras med en operation, och ger därför O(1). Detta pga att den är oordnad och alltid sätter in nästa värde i slutet på listan. Med lista av listor, blir det istället O(n+m) där n står för antalet noder som finns i rad listan och m för antalet i kolumnlistan. Eftersom listorna är ordnade, måste man gå igenom rad listan tills man hittar rätt rad

och sedan kolumnlistan tills man är på rätt kolumn innan insättning kan utföras. För ordnad dubbellänkad lista blir tidskomplexiteten O(n), då man i värsta fall kan behöva gå igenom alla noder i listan för att hitta till rätt plats att sätta in det nya värdet på för att listan ska fortsätta vara ordnad. Alla tre representationer får en rumskomplexitet på O(n), men det är värt att ha i åtanke att lista av listor skapar minst två noder då det är två listor som måste skapas. En för rad och en för kolumn.

Slutsats

Den representation jag hade valt är oordnad länkad lista, därför att den har snabbast insättning medan den inte heller är sämre än de andra två i tids- och rumskomplexitet. Detta är förutsatt att det är mest insättning som kommer utföras i kalkylprogrammet, om sökningshastighet prioriteras skulle den ordnade dubbellänkade listan vara att föredra eftersom det går att utföra sökningen från vilken nod som helst i listan. Om det är ett lägre värde man söker går det att bara söka igenom den delen av listan istället för att gå igenom listan från start till slut för att hitta rätt. Även lista av listor skulle kunna vara snabbare då man först söker upp rätt radnummer och därefter rätt kolumn. Det är i slutändan utefter hur programmet är tänkt att användas som ett val av representation får göras, är det snabb insättning som ska prioriteras eller snabb sökning? Hur mycket minne får programmet använda? Jag väljer den oordnade listan pga den snabba insättningen, men kanske någon av de andra två representationerna hade fungerat bättre om man har prioritering på något annat än snabb insättning.

Källförteckning

Janlert, Lars-Erik & Wiberg, Torbjörn (2000). *Datatyper och algoritmer*. 2., [rev.] uppl. Lund: Studentlitteratur

GeeksforGeeks. 2021. https://www.geeksforgeeks.org/sparse-matrix-representation/ (Hämtad 2021-07-20)

GeeksforGeeks. 2021. https://www.geeksforgeeks.org/sparse-matrix-representations-using-list-lists-dictionary-keys/ (Hämtad 2021-07-20)