DV2: Algoritmer och problemlösning

OU2

Theodor Jonsson

2022-01-31

ens18trn thjo0102

Innehållsförteckning

1. Inledning

Hur man lagrar datan i ett kalkylblad? Att spara hela kalkylbladet kan ta upp oändligt av minne, detta går ej. Man kan spara bara den minsta rektangel där cellerna används men detta använder en stor mängd minne för att massor av cellerna som ej har värde blir sparade. Då kommer frågan hur lagrar vi datan i ett kalkylblad utan att ta upp onödig minnesutrymme samt snabbt kunna nå datan. I denna rapport ska vi försöka lösa detta genom ta fram tre stycken kriterier för att bedöma en bra datatyp för detta problem. Sedan ska vi ta fram tre stycken datatyper som är passande. Dessa datatyper kommer bli utvärderad mot dessa kriterier för att till sist ge ett omdöme om vilken datatyp rekommenderas att använda.

2. Kriterier

Detta avsnitt kommer handla om att presentera och förklara de tre kriterier som kommer användas för att utvärdera de representationer

Dessa kriterier är framtagna genom ett seminarium med en grupp av fem datavetenskap studenter. Detta seminarium hölls för att diskutera gemensamt vilka kriterier som bör användas när man ska utvärdera representationer till detta problem.

2.1 Tidseffektiv insättning.

En bra representation har en låg tid för insättning, på grund av att denna operation kommer användas flitigt av användaren

Tidseffektivitet har mätts genom stora ordo detta är ett mått för att se operationens tidseffektivitet genom att bara kolla på dess tillväxthastighet. Om mer information om stora ordo skriver Sehgal (2017) djupare om detta. I denna kriterium kommer vi kolla efter medeltidskomplexiteten samt värstafallstidskomplexiteten detta används för till exempel har sökning inom ett icke komplett binärt sökträd en värstafallstidskomplexitet av O(n) och medeltidskomplexiteten är O(log n).

2.2 Tidseffektiv sökning av värden.

Att söka värden blir en stor del av programmet vilket menar att denna operationer bör vara tidseffektivitet. Stora ordo används här igen för att mäta tidseffektiviteten. Samt kommer också medeltids och värstafallstidskomplexiteten mätas.

2.3 Tidseffektiv Borttagning av värden.

Borttagning av värden i ett kalkylblad är en del som är mycket använt vilket gör att denna operationen bör vara tidseffektiv. I detta kriterium kommer också kolla på både medeltids och värstafallstidskomplexiteten med stora ordo. Till exempel i en Stack kan de element som vi vill ta bort vara längst nere i stacken vilket gör att vi måste gå igenom alla element för att nå detta element vilket gör att borttagningstiden blir O(n)

3. Representationer

I detta avsnitt presenteras samt förklaras de representationer som ska bli utvärderade av de kriterier från förra avsnittet

Dessa representationer kommer exemplet av [A: {(r, c, v)}] där A är alla element, r är rader, c är kolumner och v är värdet. För att enklare förklara dessa kommer vi använda oss av mängden B som visas i figur 1.

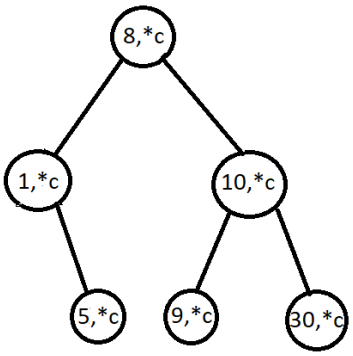
B								
r	8	10	1	30	5	10	1	9
c	3	4	2	15	3	6	3	3
v	e	8	6	a	10	a	7	3

Figur 1. Exempel mängd B med 8 element som innehåller rader, kolumner samt värde.

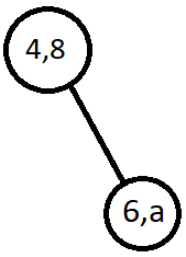
3.1 Binära sökträd nästlad i ett binärt sökträd.

Ett binärt sökträd är en typ av binärt träd där varje nod följer en viss ordning. Denna ordning är alla barn till vänster av noden är mindre än noden och alla barn åt höger av noden är större. Ordningen gör att de blir väldigt effektivt att söka inom ett binärt sökträd. När man söker så börjar man i roten av trädet här finns de tre alternativ: värdet är större än noden då går vi till högra delträdet, värdet är lägre än noden då går vi till de vänstra delträdet eller värdet ligger i den nuvarande noden då har vi hittat värdet. Betances (2018) skriver mer om binära sökträd om mer information om denna datatyp behövs.

I denna representation kommer första trädets noder innehålla rader samt pekare till de nästlade träden. Borttagning kommer använda sig av inorder successor vilket är en borttagnings algoritm vilket Simic (2022) förklarar mer om. Dessa innehåller kolumnerna samt värdet i cellen. Om mängden B antas vara ordnad som skrivet ovan då (8,3,e) blir insatt först och sist (9,3,3) blir de binära sökträdet uppbyggt som Figur 2. Om alla värden i rad 10 ska hittas så kommer de binära sökträdet för rad 10 visas som i Figur 3.



Figur 2. Binärt sökträd som innehåller noder som i sig innehåller rad samt pekare till binärt sökträd som innehåller kolumner.

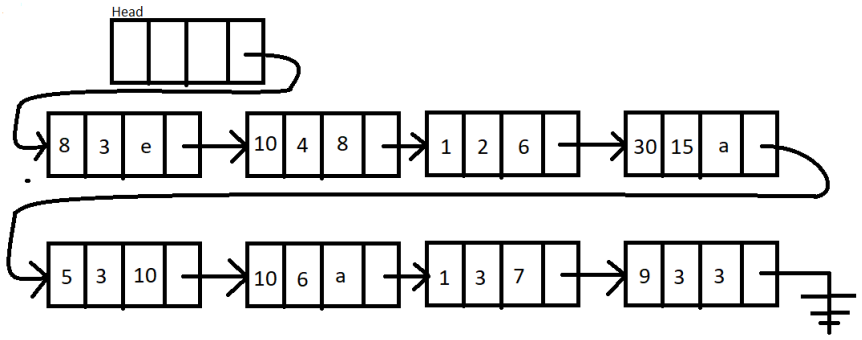


Figur 3. Binärt sökträd som innehåller noder som i sig kolumner samt värdet av cellen. Figur 1 nod 10 pekar på detta träd.

3.2 Riktad Lista av tripplar som innehåller koordinater och värde.

En Riktad lista är en specialisering av Lista. En Riktad lista har riktning man kan bara flytta sig åt ett håll. Listan använder sig av ett ‘Head’ som kommer va samma typ som elementen i listan men dess värden är odefinierade förutom pekaren på nästa element vilket kommer va första elementet i listan. Bopche (2021) redogör mer om datatypen riktad lista.

Denna representation kommer ej vara sorterad. Listan kommer gå efter första elementet som blir insatt blir de sista element i listan samt sista element som blir insatt blir de första elementet i listan. Dessa element kommer innehålla en egendefinierad struktur med fyra variabler rad, kolumn, värde och sist en pekare till nästa element. Om mängden B vilket visas i figur 1 antas vara ordnad att första element är (8, 3, e) samt sista element (9, 3, 3) kan denna representation visas som figur 4.

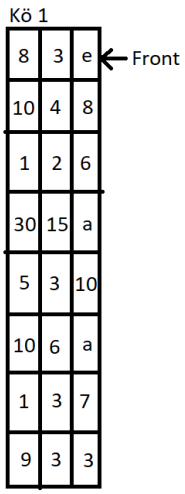


Figur 4. En lista med 8 element där varje element innehåller rad, kolumn, värde samt pekare till nästa element.

3.3 Kö av tripplar.

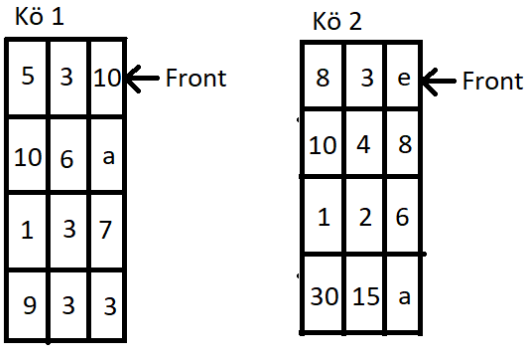
En kö är datatyp vilket sätter in element med en ordning av först in först ut. Datatypen kö är ej så komplicerad datatypen. De enda positionerna i denna datatyp som är viktiga är första och sista. Första positionen används för att både kolla värdet på de element som är på den positionen och för att ta bort de elementet, alltså de går bara att ta bort de element som är först i kön. De sista position används bara för att sätta in element.

Denna datatyp kommer bli konstruerad med hjälp av en dubbellänkad lista. Vanligtvist i en kö om man vill komma åt ett element om de inte är längst fram i kön så måste man ta bort elementet längst fram tills man kommer till de önskade elementet. För att inte ta bort elementen helt så kommer vi använda oss av två stycken köar i denna representation en för att hålla alla element och en för att hålla element som blir borttagna under traversering av kön. Elementen i kön kommer innehålla tre variabler rad, kolumn och värde. I figur 5 ser vi en kö med mängden B vilket visas i figur 1.



Figur 5. En kö med element som innehåller rad, kolumn och värde.

Exempel om vi skulle nå elementet med värde 10 på rad 5 kolumn 3 kan representation se ut som i figur 6.



Figur 6. Kö från figur 4 som traverserat till elementet med rad 5 kolumn 3 och värde 10. Den har sparat de borttagna värden i kö 2.

4. Utvärdering

I detta avsnitt kommer de tre representationer utvärderas genom de kriterier som har tagits fram.

För att sätta in ett element i ett binärt sökträd så måste man söka igenom de trädet för att först hitta rätt plats åt elementet. Inom ett binärt sökträd kan de bli att alla element i trädet ligger längst en gren samt att elementet som ska bli insatt är de element som ska allra längst ner vilket gör att värstatidskomplexiteten för detta är O(n). I ett binärt sökträd som är balanserad gör de att för varje steg i sökningen halveras antal möjliga noder som värdet är i detta ger en medeltidskomplexitet av O(log n).

Tidskomplexitet av insättning i riktad lista av tripplar enligt representationen blir O(1). Detta är för att varje element blir placerad först i listan och inga element behöver blir förflyttade eller traverserade.

Inom en Kö av tripplar blir alltid elementet insatt längst bak i kön vilket ger en tidskomplexitet av O(1)

Sökning inom ett binärt sökträd blir likadan tidskomplexitet som för insättning på grund av att under insättning söks trädet igenom för att hitta rätt plats och för att söka efter ett värde söks trädet igenom för att hitta rätt värde. Värstatidskomplexiteten blir då O(n) och medeltidskomplexiteten blir O(log n).

Medeltidskomplexitet för sökning i en riktad lista blir O(n). Egentligen så i snitt behövs bara hälften av elementen bli kollade innan man hittar rätt men på grund av hur stora ordo räknas så blir detta O(n). Samt värstatidskomplexiteten för sökning i en riktad lista blir också O(n) på grund av att elementet som söks kan ligga längst bak i listan detta gör att man behöver söka igenom varje element för att hitta rätt.

Sökning inom denna Kö av tripplar har en medeltidskomplexitet och värstatidskomplexitet av O(n). Medeltidskomplexiten visas genom att i snitt behövs bara hälften av alla element sökas igenom men på grund av hur representationen är uppbyggd så måste dessa element stoppas in i en till kö sen sättas in igen den ursplungiga kön.

Tidskomplexitet för borttagning i detta binära sökträd blir likt sökning på grund av att vi först behöver hitta rätt element att ta bort sedan hitta rätt element att byta ut de med om elementet har två barn. Detta ger en värstafallstidskomplexitet av O(n) och en medeltidskomplexitet av O(log n)

Borttagning inom Kö av tripplar blir likadant som i sökning först behöver rätt element hittas. Detta gör att vi kan behöva gå igenom hela kön för att

hitta rätt element detta ger en värstafallstidskomplexitet av O(n) samt en medeltidskomplexitet av O(n)

Borttagning i en Riktad lista har samma tidskomplexitet som sökning i en lista på grund av att elementet behöver hittas för att ta bort de. Detta ger en värstafalls av O(n) och en medel av O(n)

I Tabell 1 ser vi de olika representationerna samt deras resultat i de olika kriterierna. Tidskomplexiteten anges i stora ordo där n är antal element i representationen.

	Insättning (Ordo)		Sökning(Ordo)		Borttagning (Ordo)	
	Värsta	Medel	Värsta	Medel	Värsta	Medel
Binära sökträd nästlad i ett binärt sökträd	O(n)	O(log n)	O(n)	O(log n)	O(n)	O(log n)
Riktad lista	O(1)	O(1)	O(n)	O(n)	O(n)	O(n)
Kö av tripplar	O(1)	O(1)	O(n)	O(n)	O(n)	O(n)

Tabell 1. Sammanfattning av utvärdering av representationerna

5. Slutsats

Den bästa representation av representationerna genom dessa tre kriterier är Binära sökträd nästlad i ett binärt sökträd. Detta är för att till exempel medeltidskomplexiteten för alla kriterier för binära sökträd är O(log n). Samt så är värstafallstidskomplexiteten för ett binärt sökträd så osannolikt när antalet element växer. Kriteriet som gjorde skillnaden för binära sökträdet är sökning där de andra har båda O(n) för medel och värstafallstidskomplexitet medans binära sökträdet har O(n) och O(log n).

Vet man dock att kalkylbladet kommer användas med bara en liten mängd element kan riktad lista vara ett bättre alternativ med en snabbare insättning och borttagning.

6. Källförteckning

1. Sehgal, Karuna. 2017. A simplified explanation of the Big O notation. Medium. 27 November.

https://medium.com/karuna-sehgal/a-simplified-explanation-of-the-big-o-notation-82523585e835

(Hämtad 2022-01-27)

2. Betances, Maria. 2018. Data Structures: Binary Search Tree Explained. Medium. 1 Januari

https://medium.com/@mbetances1002/data-structures-binary-search-trees-explained-5a2eeb1a9e8b

(Hämtad 2022-02-08)

3. Bopche, Raj. 2021. Singly linked list Learn about singly linked list. Scaler. 20 September

https://www.scaler.com/topics/data-structures/singly-linked-list/

(Hämtad 2022-02-08)

4. Simic, Milos. 2022. Finding the In-order Successor of a Node.

Baeldung. 1 Januari

https://www.baeldung.com/cs/in-order-node-successor

(Hämtad 2022-02-08)