

02. Datastrukturer

Bonus:

Forelesning 2

For å unngå grunnleggende kjøretidsfeller er det viktig å kunne organisere og strukturere data fornuftig. Her skal vi se på hvordan enkle strukturer kan implementeres i praksis, og hva vi vinner på å bruke dem i algoritmene våre.

Pensum

- ☐ Innledningen til del III
- ☐ Kap. 10. Elementary data structures: Innl., 10.1 og 10.2
- ☐ Kap. 11. Hash tables: Tom. 11.3.1
- ☐ Kap. 16. Amortized analysis: Innl., 16.1 og s. 460–463 (tom. *at most 3*)

Læringsmål

- [B₁] Forstå *stakker* og *køer*
- [B₂] Forstå *lenkede lister*
- [B₃] Forstå *direkte adressering* og *hashtabeller*
- [B₄] Forstå *konfliktløsning ved kjeding*
- [B₅] Kjenne enkle *hashfunksjoner*
- [B₆] Kjenne *perfekt hashing*; $O(1)$ søk for statiske data
- [B₇] Kunne definere *amortisert analyse*
- [B₈] Forstå *aggregert analyse*
- [B₉] Forstå *dynamiske tabeller*

2

Stack

En Stack er en lineær last-in-first-out (LIFO) datastruktur. En stack fungerer som en stabel hvor elementene som legges på først, kommer ut sist. Relevant funksjonalitet på en Stack i pensum er:

- *StackEmpty* - er stacken tom?
- *StackPush* - legg til et element på toppen av stacken i konstant tid $O(1)$
- *StackPop* - fjern et element fra toppen av stacken i konstant tid $O(1)$

algdatt/Stack.scala at main · matsjla/algdatt

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below. To review, open the file in an editor that reveals hidden Unicode characters. Learn

<https://github.com/matsjla/algdatt/blob/main/src/main/scala/com/supergrecko/dsa/linear/Stack.scala>

matsjla/algdatt

Implementation of all algorithms and data structures from TDT4120 at NTNU in Scala

1 Contributor

0 Issues

0 Stars

0 Forks

Queue

En Queue er en lineær first-in-first-out (FIFO) datastruktur. En Queue fungerer på lik måte som en Stack, bortsett fra at elementene som ble lagt på først, kommer ut først. Relevant funksjonalitet på en Queue fra pensum er:

- *QueueEnqueue* - legg til et element i køen i konstant tid $O(1)$
- *QueueDequeue* - ta det fremste elementet i køen i konstant tid $O(1)$
- *QueueSize* - størrelsen på køen, som regel kun relevant hvis køen ikke backes av en resizable container.

Implementasjonen i pseudokode i pensum tar ikke hensyn til overflow eller underflow av køen. Implementasjon i Scala tar hensyn til dette.

algdatt/Queue.scala at main · matsjla/algdatt

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

<https://github.com/matsjla/algdatt/blob/main/src/main/scala/com/supergrecko/dsa/linear/Queue.scala>

matsjla/algdatt

Implementation of all algorithms and data structures from TDT4120 at NTNU in Scala

1 Contributor

0 Issues

0 Stars

0 Forks

LinkedList

En lenket liste (i pensum, doubly-linked-list) er en liste som består av noder som er allokert tilfeldige steder i minnet som er koblet sammen via pekere. Det tar dermed lineær tid å slå opp på en gitt posisjon, men det er konstant tid for å sette inn eller slette elementer. Relevant funksjonalitet på en LinkedList fra pensum er:

- *ListSearch* - søk igjennom en liste. Beste tilfelle på $O(1)$ hvis første element er søkeelementet, ellers er det gjennomsnittslig og verste tilfelle på $O(n)$.
- *ListPrepend* - legg til en node i frontend av lista i konstant tid $O(1)$.

- *ListInsert* - legg til en ny node x foran en eksisterende node y i konstant tid $O(1)$. Denne funksjonen trenger ikke et liste objekt da den kun operer på to noder.
- *ListDelete* - slett en node x fra lista i konstant tid $O(1)$

algdat/LinkedList.scala at main · matsjla/algdat

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

<https://github.com/matsjla/algdat/blob/main/src/main/scala/com/supergrecko/dsa/linear/LinkedList.scala>

matsjla/algdat

Implementation of all algorithms and data structures from TDT4120 at NTNU in Scala

1 Contributor
0 Issues
0 Stars
0 Forks

HashTable

En hashtabell er en relasjon eller mapping mellom en nøkkel på en verdi. En hashtabell implementeres som regel ved hjelp av en lineær datastruktur av LinkedList-er. Nøkkelen går igjennom en hash-funksjon som bestemmer hvilken bønne verdien skal havne i. En god hash-funksjon vil ha så få kollisjoner som mulig som gjør innsetting til, og henting fra tabellen ta konstant tid.

I tilfelle hvor det er kollisjoner, lagres også noe ekstra data (gjerne nøkkelen eller annen data som kan gjenkjenne nøkkelen) for å løse kollisjoner. Dette resulterer i veldig simple algoritmer for å bruke hashtabellen. Relevant funksjonalitet på et HashTable fra pensum er:

- *ChainedHashSearch* - finn verdien til en nøkkel i hashtabellen. Kjører i beste og gjennomsnittlig tilfelle i konstant tid $O(1)$, men siden det kan være kollisjoner er verste tilfelle $O(n)$.
- *ChainedHashInsert* - sett inn gitt nøkkel og verdi i lista i konstant tid $O(1)$.
- *ChainedHashDelete* - fjern verdien til en nøkkel i hashtabellen. Kjører i beste og gjennomsnittlig tilfelle i konstant tid $O(1)$, men siden det kan være kollisjoner er verste tilfelle $O(n)$.

algdat/HashTable.scala at main · matsjla/algdat

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below. To review, open the file in an editor that reveals hidden Unicode characters. Learn

<https://github.com/matsjla/algdat/blob/main/src/main/scala/com/supergrecko/dsa/linear/HashTable.scala>

matsjla/algdat

Implementation of all algorithms and data structures from TDT4120 at NTNU in Scala

1 Contributor
0 Issues
0 Stars
0 Forks


DynamicTable

En dynamisk tabell er en liste eller et array som har evnen til å gro eller minke i størrelse basert på antall elementer i lista. Det betyr at lista kan bli (i teorien) uendelig stor siden vi kan gro lista når den nærmer seg full. Det er vanlig å bruke en større faktor enn $n + 1$ for å gro tabellen for å unngå mange allokasjoner. Relevant funksjonalitet på et DynamicTable fra pensum er:

- *TableInsert* - legg til et nytt element i den dynamiske lista. Kjører beste og gjennomsnittslig tilfelle i konstant tid $O(1)$, men siden lista må gro innimellom er verste tilfelle $O(n)$.

algsat/DynamicTable.scala at main · matsjla/algsat

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below. To review, open the file in an editor that reveals hidden Unicode characters. Learn more

 <https://github.com/matsjla/algsat/blob/main/src/main/scala/com/supergreco/dsa/linear/DynamicTable.scala>

matsjla/algsat


Implementation of all algorithms and data structures from TDT4120 at NTNU in Scala

1 Contributor

0 Issues

0 Stars

0 Forks



Amortisert arbeid

Kjøretiden for én enkelt operasjon er ikke alltid like informativt. I en dynamisk tabell kan en insert operasjon få lista til å gro og dermed bli $O(n)$, men siden vi får en større grofaktor vil dette som regel ikke skje. Amortisert arbeid er aggregert analyse av et problem. Vi finner totalt arbeid og deler på antall operasjoner.

Average-case kommer dermed av forventet kjøretid for en algoritme, selv om den kan i noen tilfeller være mye tregere (som i *DynamicTableInsert*). Amortisert arbeid er snitt over operasjoner.