

03. Splitt og hersk

Forelesning 3

Rekursiv dekomponering er kanskje den viktigste ideen i hele faget, og designmetoden *splitt og hersk* er en grunnleggende utgave av det: Del instansen i mindre biter, løs problemet rekursivt for disse, og kombiner løsningene.

Pensum

- ☐ Innledningen til del II
- ☐ Kap. 2. Getting started: 2.3
- ☐ Kap. 4. D&C: Innl. og 4.3–4.5
- ☐ Kap. 7. Quicksort
- ☐ Oppgaver 2.3-6 og 4.5-3 med løsning (binærsøk)
- ☐ Appendiks B og C i pensumheftet

Læringsmål

- [C₁] Forstå *divide-and-conquer* (*splitt og hersk*)
- [C₂] Forstå BISECT og BISECT' (se appendiks C i pensumheftet)
- [C₃] Forstå MERGE-SORT
- [C₄] Forstå QUICKSORT og RANDOMIZED-QUICKSORT
- [C₅] Kunne løse rekurrenser med *substitusjon*, *rekursjonstrær* og *masterteoremet*
- [C₆] Kunne løse rekurrenser med *iterasjonsmetoden* (se appendiks B i pensumheftet)

1

Splitt og hersk

Splitt og hersk er et paradigme for å bryte ned et kompleks problem inn i mindre underproblemer og løse underproblemene for seg selv. Deretter setter vi sammen delløsningene for å lage en induktivt riktig løsning.

Rekurrenser

Rekurrenser er rekursive likninger, gjerne på formen $f(n) = f(n - 1) + c$. Problemet med slike definisjoner er at de er rekursive som betyr at vi kan prøve å løse de, men utfordringen er å komme seg til bunns og deretter nøste seg opp igjen.

$$f(\boxed{n}) = f(\boxed{n}/2) + 1$$

$$f(\boxed{\boxed{n}/2}) = f(\boxed{\boxed{n}/2}/2) + 1$$

$$f(\boxed{\boxed{\boxed{n}/2}/2}) = f(\boxed{\boxed{\boxed{n}/2}/2}/2) + 1$$

$$f(\boxed{\boxed{\boxed{\boxed{n}/2}/2}/2}) = f(\boxed{\boxed{\boxed{\boxed{n}/2}/2}/2}/2) + 1$$

Rekurrenser har som regel et grunntilfelle som gjør det enklere å nå bunnen.

$$f(n) = \begin{cases} n = 1 & 1 \\ n \neq 1 & f(n-1) + 1 \end{cases}$$

Rekurrenser er nyttige fordi de kan beskrive kompleksiteten eller kjøretiden til rekursive algoritmer. Vi kan løse rekurrenser med:

- Iterasjonsmetoden: gjenntatt ekspandering av rekurrensen til vi får en sum vi kan regne ut
- Rekursjonstrær: ekspander rekurrensen og lag et tre man kan summere på. TODO
- Masterteoremet: en enkel måte å løse rekurrenser av formen $T(n) = a \cdot T(\frac{n}{b}) + f(n)$. Mer om masterteoremet kommer senere i notatet.


Til slutt verifiserer vi løsningen vår med induksjonsbevis.

Binary Search

Binærsøk er en divide & conquer søkealgoritme som opererer på en sortert liste. Den velger elementet i midten av lista og sammenligner det med søkeelementet, og kaller seg selv rekursivt på enten høyre eller venstreside av lista. Grunntilfellet er en tom liste hvor vi kan konkludere med at elementet ikke eksisterer i lista.

- Best case: $O(1)$ hvis søkeelementet er i midten av lista
- Worst case: $O(\lg n)$ siden rekursjonstreet har høyde $\lg n$
- Average case: $O(\lg n)$

algdat/BinarySearch.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 more about bidirectional Unicode characters



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


Merge Sort

Merge Sort er en divide & conquer sorteringsalgoritme. Den drives av subprosedyren *Merge* som tar inn en liste, splitter det i to og sorterer sublistene. Siden merge kalles etter de rekursive kallene til *MergeSort* vil det via induksjon bevises at halvsidene av lista er sortert før de merges. Grunntilfellet er en tom liste hvor det ikke finnes mer arbeid å gjøre.

- Best case: $O(n \lg n)$
- Worst case: $O(n \lg n)$
- Average case: $O(n \lg n)$

Kompleksiteten blir $O(n \lg n)$ fordi *Merge* har en kompleksitet på n , ganget med høyden på rekursjonstreet som er $\lg n$.

algdat/MergeSort.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 more about bidirectional Unicode characters



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

Quick Sort

Quick Sort er en divide & conquer sorteringsalgoritme. Den drives av subprosedyren *Partition* som velger et pivotelement som plasseres på riktig plass i den sorterte lista, samtidig som den sørger for at alle elementer til venstre er lavere, og alle til høyre er

høyere. Siden partition kalles før det rekursive kallet til *QuickSort* partisjonerer alle breakpoints i lista og dermed kan vi ved induksjon bevise at lista er sortert. Grunntilfellet er tom liste hvor det ikke finnes mer arbeid å gjøre.

- Best case: $O(n \lg n)$
- Worst case: $O(n^2)$ hvis lista var sortert fra før av og pivot velges som siste element
- Average case: $O(n \lg n)$

For å unngå tilfellet hvor dårlig valg av pivot oppstår finnes en alternativ måte å velge pivot på; randomisert Quick Sort. Her vil verste tilfelle bli $O(n \lg n)$ når $n \rightarrow \infty$ ettersom det blir tilnærmet null sjanse for at verste tilfellet oppstår.

algdat/QuickSort.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 more about bidirectional Unicode characters.

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

matsjla/algdat

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

1 Contributor
0 Issues
0 Stars
0 Forks

algdat/RandomizedQuickSort.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 more about bidirectional Unicode characters.

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

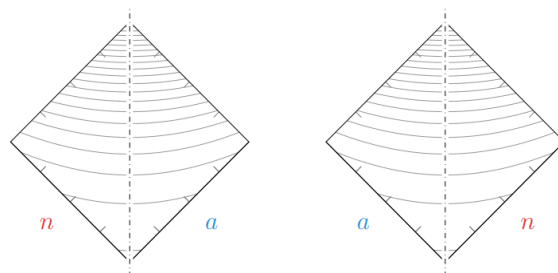
matsjla/algdat

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

1 Contributor
0 Issues
0 Stars
0 Forks

Masterteoremet

Masterteoremet er en veldig enkel metode for å løse rekurrenser på formen $T(n) = a \cdot T(\frac{n}{b}) + f(n)$. Her gjelder det bare å vite at $a^{\lg n} = n^{\lg a}$. Ved å se på hvordan logaritmen skisses på grafen er det enkelt å se at disse to er like.



Vi kan bruke iterasjonsmetoden for å finne et mønster i den rekursive funksjonen. Vi får dermed en løsning på formen $a^? T(\frac{n}{b^?})$. Her setter vi inn $? = \lg_b n$.

$T(n) = f(n)$		$a^{\lg_b n} \cdot T(\frac{n}{b^{\lg_b n}})$
$+ af(n/b)$	(1)	$a^{\lg_b n} \cdot T(\frac{n}{b})$
$+ a^2 f(n/b^2)$	(2)	$a^{\lg_b n} \cdot 1$
$+ a^3 f(n/b^3)$	(3)	$a^{\lg_b n}$
\vdots	\vdots	$n^{\lg_b a}$
$+ a^? T(n/b^?)$	(?)	

Til slutt sammenligner vi resultatet med den drivende funksjonen $f(n)$.

$$f(n) = O(n^{\log_b a - \epsilon}) \implies T(n) = \Theta(n^{\log_b a})$$

$$f(n) = \Theta(n^{\log_b a}) \implies T(n) = \Theta(n^{\log_b a} \lg n)$$

$$f(n) = \Omega(n^{\log_b a + \epsilon}) \implies T(n) = \Theta(f(n))$$