Algoritmer och Datasturkturer

Projektrapport

Av

Daniel Westerlund

2020-03-15

# Datastruktur

I valet av datastruktur fanns det tre huvudkandidater som det valdes mellan.

* ArrayList
* Träd (av något slag)
* Hashad struktur

Tidskomplexiteten med ArrayList är O(1) tidskomplexitet på att få lägga till ett element samt att ta bort ett element. Sökning av ett element har tidskomplexiteten O(n).

Binärt träd eller B+ träd, där binärt träd har bäst tidskomplexitet med O(log n) på sökning, borttagning och att lägga till element.

HashMap har värsta scenario med (O)n på sökning, borttagning och att lägga till element, dock är medeltiden O(1).

Med detta i åtanke så valdes ArrayList som val av datastruktur då den är enkel att implementera och har en bra tidskomplexitet. För att underlätta arbetet med datan så skapades en WeatherDataPoint-klass som representerar en mätning (dvs en rad i .csv filen). Attributerna för klassen är:

* Date – datumet då mätningen gjordes
* Time – tiden då mätningen gjordes
* Temp – aktuella temperaturen
* approvedValue – om mätningen var godkänd eller ej

# Algoritmer

Ett av målen med algoritmen var att försöka få ner tidskomplexiteten på sökningar av element. Eftersom datan är sorterad kan man använda en binär sökning av datan för att få ner den till O(log n). Min lösning är dock modiferad binärsökning, då den söker efter ett datum endast och där efter stegar den sig framåt för att hitta sista objektet med det sökta datumet. Detta gör att om man söker efter datumet före det som användaren angav, kommer man att få platsen där det datumet förekommer första gången. Denna metod retunerar ett heltalsvärde med platsen för datumet i ArrayList. Men den amorterade kostnaden är O(log n). Datumsökningen är den enda iterativa lösningen.

Alla lösningar bygger på att man börjar på första objektet och sedan görs rekursiva lösningar tillsdär man hämtar ut den datan man villa ha, beroende på om man det är averageTempeatures, missingValues eller approvedValues. De rekursiva lösningarna körs tills man nått

# Designtekniker

För att ta fram start och slut-punkterna i listan användes divide and conquer i binärsökningen med ett litet inslag av greedy för att få den slutgiltiga platsen. Resterande algoritmer använder greedy-design då den besöker datumet som kommer efter i listan hela tiden, tills man når slutet.

För att söka igenom listan efter start och slutplatsen krävs två sökningar vars tidskomplexitet är O(log n). Värsta scenariot är att man väljer första datumet i listan och slutar vid sista, vilket ger en tidskomplexitet på O(n). 2log(n) + n ger en amorterad tidskomplexitet på O(n).