Øving 1 Algoritme og datastrukturer IDATT2101

# Algoritmen

Algoritmen til å finne best mulig fortjeneste i denne øvingen er illustrert i figuren under:

Et bilde som inneholder tekst, skjermbilde, Font, nummer

Automatisk generert beskrivelse

Figur 1: Utsnitt av metoden findBestProfit.

Algoritmen er implementert i metoden *findBestProfit* som først tilordner fire variabler. Deretter kjører den en for-loop gjennom listen over alle aksjeforandringer kalt *listOfStockChanges*. Herfra setter den variabelen *currentProfit* til null for hver iterasjon slik at det er mulig å sammenlikne fortjenestene mellom alle dagene. I neste for-loop kjører algoritmen gjennom alle andre aksjeforandringene i listen etter den verdien som den første for-loopen itererer over. På denne måten kan man separere variablene *i* og *j*, der hvor *i* leter etter investeringsindeksen, og *j* leter etter salgsindeksen. I denne loopen sjekkes det for om den nåværende profitten er større enn den største profitten som er registrert. Hvis den er større, blir variabelen *bestProfit* satt til denne verdien.

# Kompleksitets analyse

Gjennom å analysere algoritmen linje for linje kan vi se tidskompleksiteten:

Tid

0 Dekl.

1 Tilordn.

1 Tilordn.  
1 Tilordn.  
1 Dekl.  
1 Tilordn, n sammenl, n inkr.  
n Tilordn.  
n - 1 Tilordn, n - 1 add, 1/2\* n\* (n + 1) sammenl,

1/2 \* n\*(n + 1) inkr.

1/2 \* n \* (n + 1) add, 1/2 \* n \* (n + 1) Tabelloppsl.  
1/2 \* n \* (n + 1) sammenl.  
? 1/2 \* n \* (n + 1) Tilordn.  
? 1/2 \* n \* (n + 1) Tilordn, 1/2 \* n \* (n + 1) add.  
? 1/2 \* n \* (n + 1) Tilordn, 1/2 \* n \* (n + 1) add.  
0  
0  
0  
1 Return

Totalt:   
Totalt:

public String findBestProfit() {  
 int bestProfit = 0;  
 int investingDay = 0;   
 int sellDay = 0;   
 int currentProfit;   
 for (int i = 0; i < listOfStockChanges.size(); i++) {

currentProfit = 0;   
 for (int j = i + 1; j < listOfStockChanges.size(); j++) {

currentProfit += listOfStockChanges.get(j);   
 if (currentProfit > bestProfit) {   
 bestProfit = currentProfit;   
 investingDay = i + 1;   
 sellDay = j + 1;   
 }   
 }   
 }   
 return "InvestDay: " + investingDay + ", SellDay: "   
 + sellDay + ", Profit: " + bestProfit;

Ved å summere alle disse operasjonene ender vi opp med uttrykket for kjøretiden. Grunnen til at det ofte blir telt i den andre for-loopen, er fordi den den alltid itererer over verdiene som er plassert etter den aktive verdien i tallrekka. F.eks. Du har tre objekter i en liste som skal itereres over første loop. Ved første iterasjon itererer man i den andre loopen to ganger. I den andre iterasjonen itererer man bare en gang i den andre loopen. I den siste iterasjonen itereres det ingen ganger i den andre loopen. Til sammen blir dette 6 iterasjoner totalt. Ved å bruke formelen får man det samme svaret .

# O øvre grense

På forrige side så vi at det summen av alle de operasjonene ble . Ved å la ser vi at det bare er første ledd som blir interessant, siden dette går mye raskere mot uendelig. Bare ved å se på uttrykket for kjøretiden kan det tenkes at :

deler på

velger = 10 og 7 slik at

for alle .

Konstanten bryr vi oss ikke om siden unøyaktigheter gjør slike konstanter uinteressante. Det interessante er Av dette kan vi si at kjøretiden til algoritmen øker kvadratisk med .

# Tidsmålinger

For å bekrefte analysen om at kjøretiden øker kvadratisk med må algoritmen tidsmåles. Dette er gjort ved å teste for med verdiene . For hver gang tidobles, burde tiden hundredobles. Tiden måles med hvor mange millisekunder det går per runde. Under vises resultatet fra tidsmålingene av algoritmen fra min stasjonære og bærbare datamaskin.

Figur 2: Tidsmåling for stasjonær PC.

Figur 3: Tidsmåling for bærbar PC.

Ut ifra tidsmålingene fra tabellene ser det ut til at analysen stemmer. For hver gang tidobles, hundredobles tiden. Likevel i figur 3 ser det ut som noe rart har hendt under tidsmålingene. I overgangen fra til har vi ikke en hundre dobling, men en åtti dobling. Også i overgangen fra til ser vi en sekstifem dobling. En mulig forklaring til dette fenomenet kan være at metoden som genererer de tilfeldige aksjekursforandringene genererte en tallrekke slik at den største fortjenesten ble funnet tidlig. Dermed er det en mulighet for at koden if-setningen i algoritmen ikke ble utført i like stor grad. En annen mulig forklaring er at jeg i retrospekt av målingen la merke til jeg har flere programmer og faner oppe på den stasjonære PC-en enn den bærbare. Det siste kan være at det er en altfor lang tallrekke for den bærbare PC-en, og derfor ikke kan være representative for algoritmen. Totalt brukte den stasjonære og bærbare henholdsvis 4,9 og 8,1 minutter på siste måling ().

Et bilde som inneholder line, Plottdiagram, tekst, diagram

Automatisk generert beskrivelse

Figur 4: Regresjons analyse for tidsmålingen av algoritmen på den stasjonære PC-en. Faktorene til a og b leddet ble så små at det ikke synes på grafen.