Øving 4

I denne øvingen har jeg kun gjort deloppgave 1 og 2.

# Hva er det programmet printer ut?

Programmet kjører en demo av koden jeg har laget. Den printer ut følgende informasjon i denne rekkefølgen:

* En test av koden som løser Josephus problem. En med 10 soldater og intervall lik 4, og en med 40 soldater og intervall lik 3.
* Måleserie for funksjonen som løser Josephus problem, som viser kompleksiteten med hensyn til forholdet mellom de to variablene.
* En test av koden som validerer parentes-syntaksen i en kildefil.

# Kompleksitetsanalyse av Josephus problem

Her er funksjonen jeg har skrevet for å løse problemet.

A screen shot of a computer program

Description automatically generated

De mest krevende linjene i denne operasjonen er linje 276, 280, og løkken på linje 288. På linje 276 allokerer jeg nok plass til å holde på alle soldatene, som er en operasjon. På linje 280 tilegner jeg hver node riktig verdi, som også er en operasjon. Det tyngste arbeidet ligger i løkken på linje 288. For hver soldat må listen traversere ‘interval’ antall noder fremover. I tillegg må vi ikke glemme at listen må de-allokeres også, som krever at vi itererer over alle nodene i listen. Disse delene gir oss en samlet kompleksitet på .

## Om ring\_buffer-klassen

Den bruker en egen klasse («ring\_buffer») som er laget for å holde på en sirkulær liste. Den var et forsøk på å lage en klasse som likner på en iterator-klasse hvor man kan bruke ++ for å flytte ‘avleserhodet’ frem en posisjon, og bruke \* foran variabelnavnet for å referere til den underliggende variabelen. I etterkant ser jeg at det var en litt dårlig idé, siden operasjonen forårsaker klassen til å returnere kopi av seg selv som kan ta lang tid å lage. Det må gjøres slik, siden det er funksjonaliteten man forventer fra et slikt uttrykk: «gi meg variabelen state *før* du inkrementerer». Derimot fungerer det helt fint å bruke -operasjonen. Den returnerer kun en referanse til seg selv.

Det som er kult, er at den kun holder på adressen til maks to noder samtidig: den nåværende noden, og den forrige. Resten ligger usynlig allokert i minne, slik at den eneste måten man kan finne seg frem til ønsket node er ved å traversere fremover i den sirkulære listen. Hver gang man legger til en node, allokeres mer minne med ‘new’-nøkkelordet. Hver gang man fjerner en node, de-allokeres den med ‘delete’. Klassen skal være ganske sikret mot minne-lekking, ved at de som bruker klassen aldri får direkte tilgang til nodene. I tillegg går de-konstruktøren gjennom hele listen og sletter alle nodene fra minne.

# Måleresultater

|  |  |  |
| --- | --- | --- |
| **Antall soldater** | **Intervall** | **Tid per funksjonskall (s)** |
| 10 | 100 | 0.00000245 |
| 100 | 100 | 0.00001857 |
| 1000 | 100 | 0.00033904 |
| 10000 | 100 | 0.00600515 |
| 100000 | 100 | 0.11857271 |
| 10 | 10 | 0.00000204 |
| 100 | 100 | 0.00001786 |
| 1000 | 1000 | 0.00056832 |
| 10000 | 10000 | 0.07924900 |
| 100 | 100 | 0.00001682 |
| 100 | 1000 | 0.00001793 |
| 100 | 10000 | 0.00001780 |
| 100 | 100000 | 0.00001831 |
| 100 | 1000000 | 0.00001786 |
| 100 | 10000000 | 0.00001791 |
| 100 | 100000000 | 0.00001796 |
| 10000 | 10 | 0.00230410 |
| 10000 | 100 | 0.00567280 |
| 10000 | 1000 | 0.04222510 |
| 10000 | 10000 | 0.08195890 |
| 10000 | 100000 | 0.10749760 |
| 10000 | 1000000 | 0.11103740 |
| 10000 | 10000000 | 0.10904310 |
| 10000 | 100000000 | 0.10711020 |

En interessant egenskap ved disse dataene er at økningen i intervall bytter mellom konstant og økende kompleksitet. Grunnen til dette er at jeg gjør en modulo-operasjon i traverserings-funksjonen, for å kutte vekk unødvendige løkke-traverseringer. Dette fører til at traverseringsfunksjonen har kompleksitet lik for alle verdier større enn antallet av noder. Det vises mest tydelig i det siste test-settet, der antallet soldater er konstant og intervallet øker forbi antall soldater. Økningen i tid når et tak med en gang .

# Konklusjon

Fra det vi har sett så langt, kan vi konkludere med at kompleksiteten til funksjonen avhenger av forholdet mellom antall soldater og intervallet.

Hvis intervallet er større enn eller lik antallet soldater , gjelder kompleksiteten .

Hvis antallet soldater er større enn intervallet , gjelder kompleksiteten .