Øving 4 Algoritme og datastrukturer IDATT2101

# *Deloppgave 1 (Josephus problem)*

# Oppgave 4.3–5 s. 91

For å løse Josephus problem, der hvor 41 soldater stiller seg i en sirkel og hver tredje man i sirkelen blir drept av soldaten før, har jeg valgt følgende løsning:

Et bilde som inneholder tekst, skjermbilde

Automatisk generert beskrivelse

Figur 1: Utsnitt av klassen JosephusProblem.

Dette gir igjen følgende resultat:  
Et bilde som inneholder tekst, Font, skjermbilde

Automatisk generert beskrivelse

Dermed vet vi at hvis Josephus har lyst til å være sistemann som lever, må han stille seg i posisjon .

# Kompleksitet

For å finne kompleksiteten som en funksjon av som en funksjon av antall personer/soldater og lengden på intervallet mellom dem, kan vi utføre følgende tilnærming:

*Josephus løsning:* Big-O

)

**public static int** JosephusSolution**(int** n**, int** m**) {** JosephusProblem jp = **new** JosephusProblem();  
 jp.createSoldiersInCircle(n); Soldier currentSoldier = jp.head;  
 **int** soldierCount = 0;  
 **while** (jp.head.nextSoldier != jp.head) {  
 soldierCount++; **if** (soldierCount % m == 0) {  
 currentSoldier = jp.remove(currentSoldier);  
} **else** { currentSoldier = currentSoldier.nextSoldier;  
}} **return** jp.head.findPosition();  
}

Metoden *createSoldiersInCricle* som danner soldater kjører før hovedløkka og har derfor en øvre grense på . Videre møter vi på enkelte operasjoner med som vi ikke er så interessert i. Når vi kommer til while loopen, som kjører så lenge helt til *hode*-soldaten sin neste soldat ikke er seg selv, får vi en kompleksitet på siden vi alltid fjerner hver -te . Videre møter vi på metoden *remove*, som minsker sirkelen ved å fjerne den -te soldaten. Denne metoden i seg selv har en tidskompleksitet, men den kalles bare hver -te gang i while loopen. Derfor blir tidskompleksiteten til denne operasjonen (i while loopen) totalt . Videre får vi en tidskompleksitet på for de gangene ellers-setningen inntreffer. Dette er da alltid mindre enn de sistnevnte kompleksitetene vi har nevnt hvis vi antar at og er positive heltall. Da har vi at de største tidskompleksitetene er og . Siden like godt kan være større enn kan vi ikke kvitte oss med sistnevnte kompleksitet. Vi får da at tidskompleksiteten for øvre grense må være .

# *Deloppgave 2 (matche parenteser, klammer og krøllparenteser)*

# Oppgave 5-2 s. 109

For å lage en algoritme som sjekker om et program har riktig antall og nøsting av (), [] og {}, har jeg implementert denne algoritmen:

Et bilde som inneholder tekst, skjermbilde, Font

Automatisk generert beskrivelse

Figur 2: Utsnitt av klassen MatchingParenthesesAndBrackets. Her vises metoden isParenthesisAndBracketsMatched som finner ut om parenteser og etc. er riktig matchet.

I denne metoden brukes data strukturen stakk for å lagre på tegnene (, [ og {. Dermed kan vi enkelt sjekke om at de omvendte tegnene kommer i riktig rekkefølge. Etter å ha plassert tegnene (, [ eller { i stakken, sjekkes det for når de omvendte tegnene kommer. Hvis vi får et tegn av ), ] eller }, og stakken er tom, må det altså være en feil. Ellers sjekkes det om at toppen av stakken har det omvendte tegnet til ), ] eller }. For å sikre tellefeil, må stakken være tom slik at alle tegnene stakken har blitt poppet ut like mange ganger som programmet registrerer omvendte tegn.

NB: metoden ignorerer ikke parenteser inni kommentarer eller streng-elementer.

# Fillesning

For at det skal være mulig å lese fra en fil, har jeg implementert denne metoden: Et bilde som inneholder tekst, skjermbilde, Font

Automatisk generert beskrivelse

Figur 3: Filleser metoden som returnerer fil-dataen om til en String.

Metoden bruker BufferReader klassen med try-with-reasource for å håndtere fillesningen. Dermed kan vi kontrollere kildekoden for de tidligere øvingsfilene om at parentesene, klammene og krøllparentesene er riktig plassert og matchet.

Et bilde som inneholder tekst, skjermbilde, Font

Automatisk generert beskrivelse

Figur 4: Utsnitt fra run-terminalen når programmet kjøres. Se kildekode for å se hvordan dette er gjort.