Øving 3: Rekursive Algoritmer

Godkjenning

Denne øvingen skal godkjennes av studentassistent i øvingstimene. Denne øvingen kan gjøres i grupper på opptil to studenter.

Læringsmål

Du skal lære hvordan å lage og analysere en rekursiv algoritme. Du skal lære om forskjellen mellom iterative og rekursive algoritmer.

Oppgave 1: Rekursjon vs. Iterasjon

a) Lag en rekursiv funksjon som sjekker om et ord er et palindrom, det vil si om det blir likt om det leses baklengs. Ordet «regninger» er et eksempel på et palindrom.
    a. Algoritme: Et ord er et palindrom hvis første og siste bokstav er like og resten av ordet (hvor du fjerner første og siste bokstav) også er et palindrom. Et ord på 0 eller 1 bokstaver er et palindrom.

b) Analyser algoritmen for å finne ut hva kjøretida til algoritmen er i O-notasjon. Problemstørrelsen er lengden til strengen.

c) Lag en iterativ funksjon som sjekker om ordet er et palindrom.
    a. Algoritme: Sjekk første og siste bokstav, andre og nest siste bokstav, og så videre helt til de to referansene møtes på midten.

d) Analyser algoritmen for å finne ut hva kjøretida til algoritmen er i O-notasjon. Problemstørrelsen er lengden til strengen.

Oppgave 2: Hanois Tårn

Hanoi's tårn er et gammelt spill som man kan løse med en rekursiv algoritme. Både Wikipedia, læreboka «Problem Solving with Algorithms and Data Structures using Python» samt læreboka dere hadde i DAT110 inneholder eksempel-implementasjoner av denne.

Du skal skrive en variant av denne implementasjonen som skriver ut tilstanden til de tre stablene av skiver etter hvert flytt. Husk at de ulike rekursive kallene vil referere til ulike stabler, så for å skrive ut en status som er sammenliknbar mellom kall så må metoden som skriver ut stablene ha sine egne referanser til stablene. En måte å gjøre det på er ved å definere en klasse Hanoi, la de tre stablene med skiver være instansvariabler, la utskriftsmetoden være en instansmetode, og la selve den rekursive metoden også være en instansmetode.

Kjøretida til Hanois tårn er $O(2^n)$ hvor n er antall skiver i stabelen. Analyser algoritmen deres for å finne ut hvorfor.

Oppgave 3: Fraktal

Skriv en rekursiv funksjon som tegner et Koch snøfnugg. (https://en.wikipedia.org/wiki/Koch_snowflake) Funksjonen skal ta lengden til sidene og antall nivåer med rekursjon som parameter. Du kan bruke Turtle Graphics til dette på samme måte som demonstrert i for eksempel 2d_tre.py.

Oppgave 4, frivillig: Knapsack problemet

Knapsack problemet kan formuleres som følger: Du spiller et dataspill hvor du har drept noen monstre og funnet en del verdigjenstander. Du har en ryggsekk som kan ta opptil n kilo med gjenstander. Hver gjenstand har en vekt og en verdi. Vekta til en gjenstand er et heltall antall kilo. Det er flere gjenstander enn du har plass til i ryggsekken. Finn ut hvilke av gjenstandene du skal ta med for å få mest mulig penger for dem.

Programmer en algoritme for å løse dette problemet i Python og analyser kjøretida dens. Knapsack problemet har en svært ineffektiv brute force løsning og en litt mer effektiv versjon som bruker dynamisk programmering.

Beskrivelse av dynamisk programmering algoritmen:

Gitt n gjenstander og en ryggsekk som er w stor. Lag en n*w matrise, hvor denne teksten bruker indeksene i og j for å angi enkeltceller i matrisen. Hver celle inneholder den maksimale verdien du kan få med deg ved å ta med deg de første i gjenstandene i en sekk som er j stor. For hver celle (i, j) har du to valg:

1: Ikke ta med den i-ende gjenstanden og behold maksimalverdien fra den i-1 ende cella

2: Ta med den i-ende gjenstanden. Da sjekker du løsningen for cella med indekser i-1 og j-(vekta til den i-ende gjenstanden). Løsningen er å ta med gjenstandene som er med i denne cella pluss den i-ende gjenstanden.