Beginselen van Programmeren: Oefenzitting 7

Recursie (vervolg) en intuïtieve complexiteit

Een recursief algoritme is een algoritme dat:

- controleert of het probleem triviaal oplosbaar is;
  - indien wel: dan wordt de triviale oplossing gegeven.
  - indien niet:
    - dan wordt het probleem *wat* kleiner gemaakt,
    - wordt er recursief een oplossing gezocht voor het kleinere probleem,
    - en wordt de oplossing voor het originele probleem bepaald uit de oplossing voor het kleinere probleem.

Op die manier wordt een probleem telkens kleiner, tot het triviale geval bereikt wordt.

Oefeningen

(De eerste twee oefeningen zijn de laatste uit vorige oefenzitting.)

1. Schrijf een recursief binair zoekalgoritme. Dit is een algoritme dat nagaat of een gegeven getal voorkomt in een gesorteerde lijst getallen.
2. Werk onderstaande code verder uit voor het mergeSort sorteeralgoritme. Dit algoritme werkt via het verdeel-en-heers principe, waarbij de te sorteren lijst eerst in twee gesplitst wordt en beide delen afzonderlijk gesorteerd worden. Daarna worden beide gesorteerde deelrijen efficiënt samengevoegd.

```
##  Mergesort: sorteert een lijst via mergesort
#   @param lijst: de te sorteren lijst
#   @return: een nieuwe lijst met de elementen van de gegeven lijst,
#            gesorteerd
def mergesort(lijst):
    # Indien triviaal: geef triviale oplossing
    if len(lijst) == 1:
        return lijst
    else:
        # splits in twee helften
        linkerhelft = ...
        rechterhelft = ...
        # sorteer beide delen recursief
        linksGesorteerd = mergesort(linkerhelft)
        rechtsGesorteerd = mergesort(rechterhelft)
        # voeg beide gesorteerde delen samen:
        gesorteerd = [...]

        # geef het resultaat terug
        return gesorteerd
```

3. Bij binair zoeken (oefening 1) verwachten we een tijdscomplexiteit die logaritmisch verloopt (*O(log n)*). Gebruik de code die op Toledo staat en observeer dat dit algoritme een andere tijdscomplexiteit heeft dan verwacht. (Welke?)
   Identificeer het probleem, en implementeer een nieuwe versie van het zoekalgoritme dat wél logaritmisch verloopt.
   a. (**UOVT**) Implementeer een niet-recursieve versie van binair zoeken (dus een iteratieve versie).

4. Schrijf een recursieve functie die gegeven een lijst van woorden (strings) alle mogelijke zinnen genereert die uit exact deze woorden bestaan.

   Input:

   ```
   ["ik", "programmeer", "graag"]
   ```

   Output:

   ```
   ["ik programmeer graag", "ik graag programmeer", "programmeer ik
   graag", "programmeer graag ik", "graag ik programmeer", "graag
   programmeer ik"]
   ```

5. Schrijf een recursieve functie 'geneerZinnen' met twee argumenten. Het eerste argument is een dictionary met als keys strings en met als value hun functie in een zin.

   Bijvoorbeeld:

   ```
   {"de tafel" : "substantief", "de appel" : "substantief", "staat" :
   "werkwoord", "ligt" : "werkwoord", "op" : "voorzetsel", "naast" :
   "voorzetsel"}
   ```

   Het tweede argument is een zinskelet dat een zin definieert als een lijst van zin-functies.

   Bijvoorbeeld:

   ```
   ["substantief", "werkwoord", "voorzetsel", "substantief"]
   ```

   De functie 'genereerZinnen' geeft nu alle zinnen terug die voldoen aan het doorgegeven zinskelet, uitgaande van de woordenschat gedefinieerd in het doorgegeven dictionary.

   Voorbeeld:

   ```
   woordenDict = {"de tafel" : "substantief", "de appel" : "substantief",
   "staat" : "werkwoord", "ligt" : "werkwoord", "op" : "voorzetsel",
   "naast" : "voorzetsel"}

   zinSkelet = ["substantief", "werkwoord", "voorzetsel", "substantief"]

   resultaat = ['de appel staat op de appel.', 'de tafel staat op de
   appel.', 'de appel ligt op de appel.', 'de tafel ligt op de appel.', 'de
   appel staat naast de appel.', 'de tafel staat naast de appel.', 'de
   appel ligt naast de appel.', 'de tafel ligt naast de appel.', 'de appel
   staat op de tafel .', 'de tafel staat op de tafel .', 'de appel ligt op
   de tafel .', 'de tafel ligt op de tafel .', 'de appel staat naast de
   tafel .', 'de tafel staat naast de tafel .', 'de appel ligt naast de
   tafel .', 'de tafel ligt naast de tafel .']
   ```

   Uitbreiding: zorg ervoor dat elk element uit de dictionary maximaal 1 keer gebruikt wordt. (Dus 'de appel staat op de appel mag niet meer gegenereerd worden.)