Academie voor Engineering en ICT

Opleiding Informatica

Opdrachten

Algoritmen & Datastructuren

Periode 2.1

Inhoud

[Inleiding 3](#_Toc50118245)

[1 Voorbeeld recursie 3](#_Toc50118246)

[2. Opdrachten bij les 3 4](#_Toc50118247)

# Inleiding

De aangeboden stof maak je je eigen door veel te oefenen. Je kunt niet uit je hoofd stampen voor dit vak. Reserveer ruim tijd voor de opgaven. Door veel te oefenen, verkrijg je ook de snelheid om de opgaven af te ronden in de tijd die je krijgt voor het tentamen.

Kopiëren van uitwerkingen van medestudenten of van internet heeft geen zin. Je krijgt namelijk geen beoordeling voor het maken van de opgaven.

Wen jezelf er aan om broncode altijd Engelstalig te maken. Het is een kleine moeite en zo stimuleer je hergebruik.

Het vak ADS vraagt om verschillende vaardigheden. Elk van deze vaardigheden komt aan bod tijdens het tentamen; oefen jezelf daarom in elk van deze vaardigheden:

1. Begrijpen van het *probleem*. Dit komt neer op het heel nauwkeurig lezen van de tekst van een opgave.
2. Bedenken van een *oplossing* voor het probleem. Lastige aspecten:
   1. De oplossing moet efficiënt zijn, oftewel een goede rekencomplexiteit en geheugencomplexiteit hebben. De keuze van de datastructuur is hier van belang. Een hashmap heeft heel vaak de voorkeur, of een tree als er sprake is van sortering.
   2. Als er gevraagd wordt om een recursieve oplossing, dan moet het maar net lukken om de oplossing te zien.
3. Het omzetten van de oplossing naar Engelstalige *broncode* in Java of C#. Lastige aspecten:
   1. Je weet precies wat je wilt doen, maar niet hoe je dat m.b.v. Java of C# moet doen. We laten toe dat je Java en C# API-beschrijving raadpleegt op het internet en we laten ook [www.stackoverflow.com](http://www.stackoverflow.com) toe.
   2. Kan je algoritme ook omgaan met randvoorwaarden, zoals negatieve getallen, een lege array, het getal 0, …?
   3. Bij recursieve algoritmen werkt het vaak handig om static variabelen te gebruiken. Daar kun je namelijk vanaf elke plek bij. Maar static variabelen leiden tot slecht herbruikbare code. Lukt het je om ze te vermijden?

Als je meer wilt oefenen:

* Het boek “Datastructuren in Java” van Gertjan Laan bevat meer opgaven.
* Het boek “Intro to Java Programming” van Liang (te vinden in Xplora) bevat veel opgaven.
* Internet.
* Vraag de docent.

# Voorbeeld recursie

Als docenten krijgen we vaak terug van studenten dat ze het zo lastig vinden om de aanpak te bedenken voor een recursieve opdracht.

Ons antwoord is dan:

*“druk het probleem van grootte n uit in termen van het probleem van grootte n-1. Als dat gelukt is, dan ben je klaar, want je gaat er vanuit dat het probleem van grootte n-1 al is opgelost”*

Misschien wordt het duidelijker door dit uit te voeren op de volgende opdracht  
"schrijf een methode reverse(String s) die een string omgekeerd teruggeeft"  
reverse("KEVIN") geeft dan "NIVEK" terug.

Aanpak:

* Je wilt reverse("KEVIN") berekenen. Het probleem heeft grootte n=5
* We gaan het probleem met grootte n=5 uitdrukken in termen van het probleem met grootte n=4; dat is dus reverse("KEVI"), waarbij je er van uit mag gaan dat reverse("KEVI") al werkt
* Aangezien je er van uit gaat dat reverse("KEVI") werkt, mag je reverse("KEVI") lezen als "IVEK"
* Hoe kan je het probleem reverse("KEVIN") uitdrukken in termen van reverse("KEVI")?
* Oftewel, hoe kan je reverse("KEVIN") uitdrukken in termen van "IVEK"?
* Oftewel, hoe kan je "NIVEK" uitdrukken in termen van "IVEK"?

Oplossing:

* "NIVEK" = "N" + "IVEK"
* oftewel, "NIVEK" = "N" + reverse("KEVI")
* oftewel, reverse("KEVIN") = "N" + reverse("KEVI")

Dan ben je klaar en kun je het omzetten naar code! Natuurlijk niet vergeten om een stopconditie toe te voegen, zodat de recursie stopt en geen stack overflow geeft. De code zou er zo uit kunnen zien:

**public static void** main(String[] args) {  
 System.***out***.println(*reverse*(**"KEVIN"**));  
}

**static** String reverse(String s) {  
 **if**(s.length() > 0) {  
 **char** lastLetter = s.charAt(s.length() - 1);  
 String firstPartOfString = s.substring(0, s.length() - 1);  
  
 **return** lastLetter + reverse(firstPartOfString);  
 } **else** {  
 **return ""**;  
 }  
}

Overigens, kun je ook beginnen met letters aan het begin van de string weg te halen. Dat werkt ook.

Dit is zeker niet de enige aanpak voor een recursief probleem. Als je een methode hebt die voor jou werkt, is dat ook prima.

# Opdrachten bij les 3

**Opdracht 1**Bron: Datastructuren in Java – Gertjan Laan

Schrijf een recursieve methode die aangeeft of een string een palindroom is of niet. Een palindroom is een string die van voor naar achter en van achter naar voor gelezen hetzelfde woord oplevert. Voorbeelden van palindromen zijn:

ABBA

parterretrap

meetsysteem

reinier

bob

Als je leestekens, spaties en hoofdletters negeert :

Jij slaapt, ronkt, knort; ronkt, knort pa als jij?

Baas, neem een racecar, neem een Saab.

En er is ananas, Irene.

Lezen is in, ezel!

**Opdracht 2**Je hebt bij programmeren 1 & 2 al vaak de grootste int in een array gevonden door erover heen te lopen met een for-lus. Schrijf nu een methode maxrec die het grootste getal vindt op een recursieve manier. Je mag hier dus geen for- of while-lus gebruiken!

**Opdracht 3**

Bron: Datastructuren in Java – Gertjan Laan

Schrijf een recursieve implementatie van het binair zoekalgoritme (binary search).