ICA

INFORMATICA
COMMUNICATIE
ACADEMIE

studiehandleiding

APP

titel:	Algoritmes, Programmeertalen en Paradigma's
studiejaar:	2018-2019
opleiding:	HBO-ICT
auteurs:	Michel Portier, Bas Lijnse
versie:	2019.2.2

HAN

Inhoudsopgave

1. De Course

Tot nu toe heb je geleerd te ontwikkelen en ontwerpen met aangereikte talen, vooral gericht op het object-georiënteerde, imperatieve paradigma (Java). Er zijn echter veel meer talen, die gebaseerd zijn op verschillende paradigma's en in de praktijk zul je jezelf deze moeten aanleren, vaak zonder dat je daar les in krijgt. Ook zul je, als je wat dieper binnen een taal kijkt, te maken krijgen met (standaard)datastructuren en -algoritmen. Om goede beslissingen te kunnen nemen heb je daarom kennis nodig van hoe algoritmen, datastructuren en talen onder de motorkap werken. Zo zijn er verschillende types van lijsten waarin je gegevens kunt opslaan, en zijn er talen die vooral goed zijn in specifieke situaties. Ook zul je soms je eigen taal moeten kunnen maken en door software laten begrijpen. In dit vak wordt het fundament gelegd om in de beroepspraktijk niet alleen beter te kunnen programmeren, maar ook goed te begrijpen hoe programmeertalen eigenlijk werken.

2. Praktische Informatie

2.1 Studiepunten, studiebelasting, docententeam

Deze OWE heeft een omvang van 7,5 studiepunten (sp). Dit komt neer op een studiebelasting van 20 uur per week. Bij deze course zijn de volgende docenten betrokken: Dennis Breuker en Sander Maijers.

Alle informatie over competenties, criteria en toetsen is letterlijk overgenomen uit het opleidingsstatuut 2018-2019. Aan deze studiehandleiding kunnen geen rechten worden ontleend.

2.2 Inhoud en onderdelen

Deze OWE bestaat uit drie thema's, te weten:

- Week 1 t/m 4: Algoritmen en datastructuren
- Week 5 t/m 6: Programmeertalen
- Week 1 t/m 8: Paradigma's

Het eerste thema wordt afgesloten met een praktijktoets en een schriftelijke toets. De andere thema's worden afgesloten met een praktijktoets. Voor iedere les bestaat er literatuur, slides, en huiswerk.

2.3 Competenties

In deze course werken we aan het leren van de volgende competenties:
APP-9. De student kan niet-triviale programma's schrijven in een functionele enof logische programmeertaal.
APP-7. De student kan een eenvoudige programmeertaal met behulp van standaardtechnieken implementeren.
APP-1. De student kan (standaard) algoritmen en datastructuren toepassen voor een gegeven programmeer-probleem.
APP-6. De student kent de theoretische aspecten van programmeertaalontwerp en kan deze lezen en beschrijven met standaardnotaties.
APP-8. De student is bekend met meerdere programmeerparadigma's: imperatief, object-georienteerd, functioneel en logisch, en kan in programma's bewust gebruik maken van de mogelijkheden van een paradigma.
APP-3. De student kan gegeven een specifieke context passende keuzes voor algoritmes en datastructuren maken.
APP-4. De student kent de werking van, en kan redeneren over de toepassing van (standaard) algoritmen en datastructuren.
APP-5. De student kan redeneren over de complexiteit van algoritmes en kent complexiteitsklassen en grote O-notatie.
APP-2. De student kan (standaard) algoritmen en datastructuren implementeren en uitbreiden, zonder gebruik te maken van standaardbibliotheken.

2.4 Beoordelingsdimensies

Competentie	Beoordelingscriteria	Tentamen
APP-9	• Legt voor zelf geschreven code uit hoe deze gebruik maakt van de kerneigenschappen van een paradigma. • Legt de werking van zijn in het vorige punt genoemde programma in eigen woorden uit. • Schrijft een niet-triviaal programma in een functionele en of logische programmeertaal.	PO_Paradigma
APP-7	• Realiseert een codegenerator voor een abstracte virtuele machine of source-to-source compilatie. • Realiseert een evaluator op basis van een zelf gedefinieerde abstracte syntaxboom • Realiseert een eenvoudige (type)checker op basis van een zelf gedefinieerde abstracte syntaxboom. • Gebruikt een parsergenerator om op basis van een grammatica een parser en lexer te realiseren.	B_Progtaal
APP-1	• Schrijft een programma dat een standaard of uitgebreid algoritme of datastructuur toepast, gegeven een probleemstelling.	PO_Algoritmes
APP-6	• Beschrijft het ontwerp van een eenvoudige compiler enof interpreter met correct gebruik van terminologie. • Legt de grammatica van een eenvoudige (programmeer)taal formeel vast in een standaardnotatie. Bijvoorbeeld (E)BNF of vergelijkbaar.	B_Progtaal
APP-8	• Kan de relatie tussen programmeertaalfeatures en standaardparadigma's uitleggen. • Kan voor een gegeven programmeertaal beargumenteerd het paradigma waarbinnen die taal valt bepalen.	PO_Paradigma
APP-3	• Beargumenteerd op basis van verwacht dataaccess of een datastructuur een goede keus is. • Beargumenteerd op basis van verwachte input of een algoritme een goede keus is. • Kiest het beste algoritme uit meerdere alternatieven gegeven eisen aan geheugengebruik of rekentijd.	S_Algoritmes
APP-4	• Past algoritmes toe op een concrete input, en laat eventuele tussenstappen zien. • Herkent voor welke problemen gegeven algoritmen oplossingen zijn en kan hier een beredeneerde keuze voor maken. • Legt de essentie of werking van de (standaard) algoritmen en datastructuren in eigen woorden uit al dan niet gegeven een probleemstelling.	S_Algoritmes

Competentie	Beoordelingscriteria	Tentamen
APP-5	• Geeft een op basis van complexiteit beargumenteerde keuze tussen verschillende algoritmes. • Bepaalt de complexiteit in termen van tijd of geheugen voor een gegeven algoritme. • Redeneert over de complexiteit van een algoritme voor een specifieke input.	S_Algoritmes
APP-2	• Implementeert eigen datastructuren, eventueel met een aanpassing of uitbreiding, gegeven een probleemstelling.	PO_Algoritmes

2.5 Tentamens

Code tentamen	B_Progtaal
Toetsvorm	Beroepsproduct
Deeltentamen	Beroepsproduct: implementatie van een programmeertaal
Aantal examinatoren	1
Beoordeling	Individueel cijfer 1 t/m 10
Minimaal resultaat	5,5
Weging	1
Periode afname	Zie toetsrooster in i-SAS

Code tentamen	PO_Paradigma
Toetsvorm	Verslag
Deeltentamen	Programmeeropdracht, verslag in blogvorm
Aantal examinatoren	1
Beoordeling	Voldoende - Onvoldoende (individueel)
Minimaal resultaat	Voldoende
Weging	N.v.t.
Periode afname	Zie toetsrooster in i-SAS

Code tentamen	PO_Algoritmes
Toetsvorm	Performance Assessment
Deeltentamen	Programmeeropdracht, uit te voeren binnen 2 uur
Aantal examinatoren	1
Beoordeling	Individueel cijfer 1 t/m 10
Minimaal resultaat	5,5
Weging	1
Periode afname	Zie toetsrooster in i-SAS

Code tentamen	S_Algoritmes
Toetsvorm	Schriftelijk tentamen
Deeltentamen	Schriftelijk tentamen, open vragen, gesloten boek
Aantal examinatoren	1
Beoordeling	Individueel cijfer 1 t/m 10
Minimaal resultaat	5,5

Weging	1
Periode afname	Zie toetsrooster in i-SAS

2.6 Inleveren en Archivering

2.6.1 PO_Algoritmes

Dit is een performance-assessment en vindt plaats in week 5. Je krijgt ter plaatse een opdracht en die moet je binnen twee uur uitprogrammeren. Je mag (moet!) al het gemaakte huiswerk en literatuur gebruiken. Sterker nog: je hebt het gemaakte programmeerhuiswerk nodig om de toets in twee uur af te kunnen krijgen. Het derde uur wordt gebruikt door de docent om het cijfer te bepalen. Tot slot lever je de toets in op iSAS. Deze toets vindt plaats tijdens een van de normale lesblokken.

2.6.2 S_Algoritmes

Dit is een schriftelijk tentamen met open vragen. De duur is 90 minuten. Het tentamen is gesloten boek.

2.6.3 B_Progtaal

Deze programmeer- en schrijfopdracht maak je in je eigen tijd en lever je in op iSAS op donderdag in week 6. Mogelijk is er een mondeling assessment.

2.6.4 PO_Paradigma

Deze programmeeropdracht maak je in je eigen tijd en lever je in op iSAS op donderdag in week 8. Mogelijk is er een mondeling assessment.

NB. De genoemde weken gelden voor de voltijduitvoering. Voor de deeltijduitvoering gelden afwijkende inlevertijden.

2.7 Middelen

Literatuur

- Weiss, M. (2010). Data structures & problem solving using Java. Boston, Mass. London: Pearson Education. (ISBN-13 978-0-321-54622-7) of de nieuwere versie: Weiss, M. (2013). Data Structures and Problem Solving Using Java: Pearson International Edition. (9781292025766)
- Tate, B. (2010). Seven languages in seven weeks : a pragmatic guide to learning programming languages. Raleigh, N.C. Farnham: Pragmatic Bookshelf O'Reilly distributor. (ISBN-13 978-1-93435-659-3)

Software

- Algoritmes: Java of C# met IDE naar keuze
- Programmeertalen: Java en Antlr
- Paradigma's: Open source implementaties van de talen: Ruby, IO, Prolog, Scala, Erlang, Clojure, Haskell

3. Inhoud en Planning

In de course zullen uiteraard meerdere onderwerpen behandeld worden die samen de beoogde competenties uit hoofdstuk 2.3 afdekken. Een typische les bestaat meestal uit terugblikken op eerdere onderwerpen en de introductie van een nieuw onderwerp.

3.1 Onderwerpen: Algoritmes

3.1.1 Algoritme-analyse

Je kunt programma's met dezelfde functionele eisen op verschilende manieren schrijven. Verschillende oplossingen hebben wel andere eigenschappen. Hoe snel je programma zal zijn voor verschillende soorten invoer hangt af van keuzes in datastructuren en algoritmen die je maakt. In deze les kijken we hoe je programma's systematisch kunt analyseren om van te voren iets te kunnen zeggen over het effect van dit soort keuzes.

Leerdoelen

- Weten wat algoritmen en datastructuren zijn
- Kunnen redeneren over complexiteit van algoritmes
- Kunnen bepalen van Big Oh ordes

3.1.2 Recursie

Een techniek die je in de programma's die je tot nu toe geschreven hebt in de opleiding niet zo vaak tegen komt, maar wel vaak in algoritmen is recursie. Dit is het schrijven van functies die zichzelf herhaaldelijk aanroepen. In deze les kijken we hoe dit werkt en hoe je zet kunt toepassen.

Leerdoelen

- Het begrip "recursie"kunnen uitleggen
- Een recursieve functie kunnen schrijven
- De beperkingen van recursie kennen

3.1.3 Sorteeralgoritmes

Sorteren van data is iets dat vaak moet gebeuren, en afhankelijk van hoe je het aanpakt veel rekentijd kan kosten. In deze les kijken we naar verschillende sorteeralgoritmen en hun eigenschappen.

Leerdoelen

- Het belang van sorteren uit kunnen leggen
- Insertionsort kunnen toepassen en uitprogrammeren
- Mergesort kunnen toepassen en uitprogrammeren
- Quicksort kunnen toepassen en uitprogrammeren

3.1.4 Generics en Collections API

Bij het ontwerpen van eigen datastructuren voor specifieke algoritmen, wil je kunnen abstraheren van het type data dat je in de datastructuur wil opslaan. In Java (en C#) gebruik je daarvoor het ”Generics”mechanisme. In deze les kijken we hoe dit werkt en hoe je het gebruikt in je programma’s.

Leerdoelen

- Java Generics kunnen toepassen
- Uitleggen welke problemen generics oplossen
- Het verschil tussen boxed/unboxed data kunnen uitleggen
- De Comparable interface kennen

3.1.5 Lists, Stacks en Queues

Drie “klassieke” datastructuren die je in heel veel software tegen komt zijn lists, stacks en queues. In deze les bekijken we van deze datastructuren hoe ze in elkaar zitten, hoe je ze gebruikt en wat hun eigenschappen zijn.

Leerdoelen

- De list structuur kennen en kunnen redeneren over de complixiteit van verschillende implementaties
- De stack structuur kennen en kunnen redeneren over de complexiteit van verschillende implementaties
- De queue structuur kennen en kunnen redeneren over de complexiteit van verschillende implementaties

3.1.6 Hashing

Snel gegevens terug kunnen vinden is in heel veel applicaties belangrijk. Een techniek die je daarvoor kunt gebruiken is hashing. In deze les kijken we hoe dat werkt, en hoe je het toe kunt passen.

Leerdoelen

- Het kernidee achter een hastable kunnen uitleggen
- Hash functies en hastables kunnen uitprogrammeren
- Weten wat separate chaining, lineair probing en quadratic probing zijn en ze kunnen toepassen

3.1.7 Grafen en paden

Netwerkstructuren, formeel grafen, kom je ook in veel informaticatoepassingen tegen. Het kunnen vinden van (korte) paden in zo’n graaf hoort daar ook bij. In deze les kijken we naar deze datastructuur en standaard-algoritmen voor het vinden van kortste paden.

Leerdoelen

- Weten wat een graaf is hoe hij gerepresenteerd wordt
- Grafen kunnen uitprogrammeren
- Kortste paden kunnen bepalen in ongewogen en gewogen grafen

3.1.8 Bomen

Bomen zijn een speciaal geval van grafen die overal waar hiërarchieën een rol spelen gebruikt worden. Ook worden ze in zoekproblemen veel gebruikt om gegevens te ordenen. In deze les kijken we hoe je boomstructuren kunt maken en hoe je ze kunt gebruiken.

Leerdoelen

- Weten wat bomen zijn en hoe ze gerepresenteerd worden
- Boomstructuren kunnen uitprogrammeren
- Tree-traversels in pre- en post-order kunnen uitvoeren

3.1.9 Binary-, Search- en AVL-bomen

Voor het efficient terugvinden van gegevens gebruiken we speciale boomstructuren en algoritmen die dat soort bomen bewerken. In deze les kijken we naar deze specifieke datastructuren en algoritmen.

Leerdoelen

- Weten wat BST's zijn en de eigenschappen ervan kunnen uitleggen
- Operaties op BST's kunnen uitvoeren
- BST's kunnen (her)balanceren

3.2 Onderwerpen: Programmeertalen

3.2.1 Taalontwerp

Tot nu toe heb je in de opleiding een aantal programmeertalen leren gebruiken. Een implementatie van een programmeertaal is zelf ook software waar een gestructureerd ontwerp aan ten grondslag ligt. In deze les kijken we naar wat er komt kijken bij het ontwerp van een programmeertaal. Ook kijken we naar standaardmethoden om de grammaticaregels van programmeertalen te beschrijven.

Leerdoelen

- Bewustworden van wat een programmeertaal is
- Weten hoe je een programmeertaal beschrijft: syntax, semantiek en pragmatiek
- Werken met contextvrije grammatica's in (E)BNF

3.2.2 Taalimplementatie

Het ontwerpen van een programmeertaal alleen is niet voldoende. Zonder implementatie heb je er in de praktijk niets aan. Ook hier zijn standaardpatronen die je in de meeste taalimplementaties tegen komt. In deze les kijken we naar de standaard manieren waarop je een programmeertaal kunt implementeren.

Leerdoelen

- Weten hoe je een (programmeer)taal implementeert
- Weten wat de standaardonderdelen van een compiler (en interpreter zijn)

3.2.3 Lexing en Parsing

Een van de standaardstappen in de implementatie van een programmeertaal is het parseren (ontleden) van de invoer volgens de grammatica van de taal. In deze les kijken we hoe we gegeven een informeel taalontwerp met grammatica de stap de eerste stap van een taalimplementatie realiseren. Het maken van een lexer en parser op die platte programmatekst omzetten naar een boomstructuur.

Leerdoelen

- Weten wat een parse tree is
- Weten wat een lexer doet
- Weten wat een parser doet
- Het kunnen gebruiken van de Antlr parser generator

3.2.4 Evaluatie en Checking

Simpelweg parseren volgens de regels van een grammatica geeft weliswaar een boomstructuur, maar wel één waarin nog syntactische informatie staat die we na het parseren eigenlijk niet meer nodig hebben. In deze les gaan we die boom daarom tranformeren naar een abstractere representatie van de programmeertaal, een zogenaamde AST. Deze datastructuur kunnen we vervolgens gebruiken om te evalueren of om te analyseren om programmeerfouten te vinden.

Leerdoelen

- Weten wat het verschil tussen een parse tree en abstract syntax tree (AST) is.
- Weten welke mogelijkheden Antlr biedt om parse trees te traverseren
- Weten hoe je een programma (als AST) kunt evalueren
- Weten wat de rol is van checking in een compiler

3.2.5 Transformatie en Optimalisatie

Tussen het parsen van de invoer en het uiteindelijk genereren van uitvoer wordt een programma in een compiler vaak een aantal keer getransformeerd. In deze les kijken we hoe, wanneer en waarom dit soort transformaties plaatsvinden.

Leerdoelen

- Weten wat programmatransformatie is
- Transformatie kunnen gebruiken om codegeneratie te vereenvoudigen
- Transformatie gebruiken voor optimalisatie

3.2.6 Codegeneratie

In compilers is het uiteindelijke doel het produceren van vertaalde code in uitvoerbestanden. In deze les kijken we naar deze laatste stap in het proces.

Leerdoelen

- Weten wat de rol van codegeneratie is in een compiler

3.3 Onderwerpen: Paradigma's

3.3.1 Software-craftmanship

Tijdens je opleiding wordt je gaandeweg zelfstandig vakman. In dit verdiepende semester staan we deze les weer een keer stil bij deze rol als professioneel vakman. Craftmanship is meer dan alleen Clean code. In deze les sta je onder andere stil bij wat vakmanschap als ontwikkelaar voor jouw betekent.

Leerdoelen

- De rol van developer als vakman kennen
- Zelf een nieuwe programmeertaal kunnen ontdekken met methode van Tate

3.3.2 Standaardparadigma's

Hoewel er heel veel software-paradigma's te onderkennen zijn, zijn er maar een handvol grote stromingen. In deze les kijken we naar de vier veel voorkomende standaardparadigma's.

Leerdoelen

- Het begrip paradigma kunnen uitleggen in de context van programmeertalen
- De kerneigenschappen kunnen noemen van de belangrijke paradigma's:
  - Imperatief
  - Object-geörienteerd
  - Logisch
  - Functioneel

3.4 Planning

Onderstaand schema geeft een overzicht van wanneer de onderwerpen zoals beschreven in de vorige secties zijn verdeeld over de onderwijsweken. Dit schema is slechts een voorlopige planning, geen leidend rooster. Er kan per klas van afgeweken worden.

Les / OW	1	2	3
3-1	Algoritme-analyse	Recursie	Sorteeralgoritmen
3-2	Generics en Collections	Lists, Stacks en Queues	
3-3	Hashing	Grafen en Paden	Bomen
3-4	Binary-, Search- en AVL-bomen	Standaard paradigma's	
Voorjaarsvakantie			
3-5	[Assessment algoritmen]	Taalontwerp	Taalimplementatie, Lexing en Parsing
3-6	Evaluatie en Checking	Codegeneratie, Transformatie en Optimalisatie	
3-7	[Gastspreker]	Software-craftmanship	
3-8			

4. Opdrachten

In dit hoofdstuk staan de praktijkoefeningen en opdrachten. Deze kunnen tijdens de course nog aangepast en aangevuld worden. Ook kan het zijn dat sommige opdrachten niet gemaakt hoeven te worden. Volg hierin altijd de aanwijzigingen van de docent.

4.1 Algoritmes

Deze opdrachten horen bij de lessen in het thema Algoritmen. Bij dit thema gebruiken we het boek van Weiss, de “4th NEW INTERNATIONAL EDITION”. Wanneer in dit deel naar opdrachten uit “het boek” verwezen wordt dan is dat dus dit boek. De vorige druk van dit boek, de “4th INTERNATIONAL EDITION”, is ook nog in omloop. Deze versie heeft helaas een andere nummering dan de huidige editie. Bij alle opdrachten staat schuingedrukt aangegeven wat de equivalente opdracht uit het oude boek is.

De programmeeropgaven welke zijn aangeduid als “(verplicht voor PO_Algoritmes) ” moet je meenemen aan het begin van PO_Algoritmes. Deze heb je nodig om het assessment binnen twee uur af te kunnen krijgen. Opgaven gemarkeerd met “(extra nuttig voor PO_Algoritmes) ” zijn opgaven die bij uitstek belangrijk zijn als oefening voor PO_Algoritmes.

Praktijkopdrachten gemarkeerd met (W) zijn met toestemming overgenomen uit de huiswerkopgaven van Hogeschool Windesheim.

4.1.1 Algoritme-analyse

Bestuderen

- Powerpoint slides uit de les
- Hoofdstuk 5. Focus je op de Big-Oh notatie en sla zoveel mogelijk van de wiskunde over (tenzij je dit begrijpt en interessant vindt natuurlijk).

Opdrachten uit het boek

- 5.4 [In oud boek: 5.6]
- 5.6 [In oud boek: 5.5]
- 5.14a,c,d [In oud boek: 5.12a,c,d (gebruik 0,5 ipv 0,4)]
- 5.15a,c,d [In oud boek: 5.16a,c,d]
- 5.20 [In oud boek: 5.26]

Overige opdrachten

1. Zoek op Internet naar het Traveling Salesman Problem (TSP) en leg in eigen woorden uit wat dit probleem inhoudt.
2. Wat is de runtime-complexiteit van het TSP? Leg in je eigen woorden uit.

Praktijk

Bij deze les zijn geen praktische opdrachten.

4.1.2 Recursie

Bestuderen

- Powerpoint slides uit de les
- Hoofdstuk 7.1-7.3. Hoofdstuk 7.2 is erg wiskundig en wordt niet getoetst op het tentamen

Opdrachten uit het boek

- 7.1 [In oud boek: 7.4]

Praktijk

1. Programmeer de faculteitsfunctie recursief en niet-recursief uit (`5! = 5x4x3x2x1`)
2. Programmeer de somfunctie recursief en niet-recursief uit (`som(5) = 5+4+3+2+1`)
3. Maak opdracht 7.23 [In oud boek: 7.26] uit het boek
4. Maak opdracht 7.49 [In oud boek: 7.44] uit het boek
5. (W) Maak een fractal genaamd de H-Tree:
   `http://en.wikipedia.org/wiki/H_tree`
   
6. Meer oefening nodig? (optioneel/aanbevolen)
   `http://codingbat.com/java/Recursion-1`

4.1.3 Sorteeralgoritmes

Bestuderen

- Powerpoint slides uit de les
- Heel hoofdstuk 8 met uitzondering van ShellSort

Opdrachten uit het boek

- 8.1a,c,d,e (vergeet de cutoff bij e) [In oud boek: 8.3a,b,d,e]
- 8.4 (Zonder ShellSort) [In oud boek: 8.14]
- 8.5 (Zonder ShellSort) [In oud boek: 8.15]
- 8.6 (Zonder ShellSort)

Overige opdrachten

1. Sorteer de volgende getallenreeksen met behulp van de geleerde algoritmen. Laat alle tussenstappen duidelijk zien:

```
2,4,6,8,10,12,14
8,7,6,5,4,3,2,1
0, 1, 4, 9, 0, 3, 5, 2, 7, 0
```

Praktijk

1. (verplicht voor PO_Algoritmes) Programmeer alle sorteeralgoritmen uit de les uit. Zorg dat deze werken op arrays van het type int. Maak unittests om de werking te testen. Zorg voor allerhande testgevallen (lege array, omgekeerde input, alle waarden gelijk etc). Maak een interface die je voor alle sorteeralgoritmen gebruikt, en gebruik deze bij het unittesten. Zo kun je je testcases hergebruiken.
2. Ga op zoek naar een nog niet behandelde sorteervariant en programmeer deze uit.

4.1.4 Generics en Collections API

Bestuderen

- Hoofdstuk 4.6 en 4.7

Opdrachten uit het boek

- 4.4 [In oud boek: 4.6]
- 4.11 [In oud boek: 4.10]

Praktijk

1. Maak opdracht 4.24 [In oud boek: 4.26] uit het boek
2. Maak opdracht 4.29 [In oud boek: 4.32] uit het boek. Gebruik de Generic versie.
3. Maak opdracht 4.33 [In oud boek: 4.22] uit het boek. Gebruik de `Shape` class uit Figure 4.14.
4. (verplicht voor PO_Algoritmes) Programmeer alle sorteeralgoritmen uit de vorige les uit, maar maak ze dit keer Generic voor objecten die de `Comparable <T>` interface implementeren. Kun je de unittests uit de vorige les opnieuw gebruiken?

4.1.5 Lists, Stacks en Queues

Bestuderen

- Hoofdstuk 6: 6.1 t/m 6.6
- Hoofdstuk 15 [In oud boek: 16]
- Hoofdstuk 16 [In oud boek: 17]
- Hoofdstuk 11: t/m 11.1.1

Opdrachten uit het boek

- 15.1 [In oud boek: 16.1]

Praktijk

1. (verplicht voor PO_Algoritmes) Implementeer zelf een `ArrayList` voor type `int`. Implementeer de volgende methodes: `add(int value)` (voegt value toe aan einde vande lijst), `get (int index)` en `set (int index, int value)`. Maak unittests. Zorg voor verdubbeling als de `ArrayList` volloopt. Optioneel: maak een generic `ArrayList`
2. (verplicht voor PO_Algoritmes) (W) Implementeer een generic singly-linked-list klasse `HANLinkedList<T>`. Maak gebruik van een header node. Maak unittests. Verplichte methodes: `addFirst(T value)` (voegt value toe aan begin van lijst), `removeFirst()`, `insert (int index, T value)`, `delete (int index)`, `get (int index)`
3. (verplicht voor PO_Algoritmes) (W) Maak een klasse `HANStack<T>`, met methodes `pop`, `top`, `push` en `getSize` die gebruik maakt van de `HANLinkedList<T>`
4. (verplicht voor PO_Algoritmes) Lees hoofdstuk 11.1.1, maak een Java programma die op basis van de zelfgemaakte stack een balanced-symbol-checker voor haakjes implementeert.
5. (verplicht voor PO_Algoritmes) Maak oefening 15.7 [In oud boek: 16.7]
6. (verplicht voor PO_Algoritmes) Maak voor de datastructuren in 1-4 een zinvolle implementatie voor de `toString()` methode.
7. (extra nuttig voor PO_Algoritmes) Lees `https://nl.wikipedia.org/wiki/Torens_van_Hanoi` en implementeer de oplossing recursief. Maak gebruik van de `HANStack` uit de voorgaande opgave. De recursieve oplossing vind je overal met een goede search, maar het gaat er nu juist om dat je gebruik maakt van je eigen datastructuur.
8. (extra nuttig voor PO_Algoritmes) Maak een nieuw datatype genaamd een `Quack`. Deze heeft alle eigenschappen van zowel een `Queue` als een `Stack`. Bedenk zelf welke onderliggende datastructuur je gebruikt en probeer tot een zo optimaal mogelijke implementatie te komen.

4.1.6 Hashing

Bestuderen

- Hoofdstuk 19: t/m 19.4 [In oud boek: 20: t/m 20.4]

Opdrachten uit het boek

- 19.2 [In oud boek: 20.3]
- 19.3 [In oud boek: 20.4]
- 19.5 [In oud boek: 20.1]
- 19.6 [In oud boek: 20.2]

4.1.7 Grafen en paden

Bestuderen

- Hoofdstuk 13.1 t/m 13.3 [In oud boek: 14.1 t/m 14.3]

Opdrachten uit het boek

Bij dit onderwerp zijn geen opdrachten uit het boek.

Overige opdrachten

1. Bekijk figuur 13.1 [In oud boek: 14.1]. Zoek de unweighted shortest-paths van V0, V1, V2, V3 naar alle andere nodes (je mag dus alle wegingen op 1 zetten)
2. Bekijk figuur 13.1 [In oud boek: 14.1]. gebruik Dijkstra om van V0, V1, V2, V3 naar alle andere nodes het korste pad te vinden

Praktijk

1. (verplicht voor PO_Algoritmes) Implementeer een `Graph` met `Node`s en `Edge`s zoals beschreven in het boek. Implementeer de methode `toString()` zodanig dat deze een lijst van `Node`s met bijbehorende edges print (je mag net als in het boek de `java.util.HashMap` klasse gebruiken).
2. (verplicht voor PO_Algoritmes) Implementeer Dijkstra en Unweighted shortest-path. Gebruik eigen datastructuren, alleen de `java.util.PriorityQueue` mag je gebruiken.
3. (verplicht voor PO_Algoritmes) (W) (extra nuttig voor PO_Algoritmes) Breid de `Graph`-klasse uit met de volgende methode: `bool isConnected()` De methode geeft terug of een ongerichte graaf wel of niet verbonden (zie voor definitie `http://mathworld.wolfram.com/ConnectedGraph.html`) is. Maak testcode aan voor zowel een verbonden als een niet-verbonden graaf.

4.1.8 Bomen

Bestuderen

- Hoofdstuk 17 [In oud boek: 18]
- Hoofdstuk 11.2.1
- 11.2.2

Opdrachten uit het boek

- 17.1 [In oud boek: 18.1] (totdat je het snapt)
- 17.2 [In oud boek: 18.2]

Overige opdrachten

1. Zie figuur 17.11 [In oud boek: 18.1] Geef van de volgende sommen de postfix expressie en teken de binaire boom:

```
1 * 2 * 3
4 * 6 + 5
4 * ( 6 + 5)
```

Praktijk

1. (verplicht voor PO_Algoritmes) Maak een implementatie van van de First-Child-Next-Sibling Tree. Gebruik generics, zodat het basistype is: Tree<T>. Zorg ook voor handige constructor(s) en methode(s) voor toevoegen en verwijderen van kinderen (bijvoorbeeld `addChild/insert` en `removeChild/remove`).
2. Implementeer de BinaryTree<T>. Zorg ook hier voor handige constructor(s) en methode(s) (bijvoorbeeld `setLeft` en `setRight`).
3. Maak 17.9 [In oud boek: 18.11]

4. Implementeer de sommen bij overige oefeningen in een binaire boom, en schrijf code welke de sommen ook uitrekent.

5. Zorg voor een nette printfunctie voor alle type trees.

6. De voorgaande oefeningen (3,4,5) heb je waarschijnlijk opgenomen als methoden in de betreffende Tree-klassen, al dan niet static. Het is natuurlijk veel mooier wanneer we dit generiek kunnen oplossen. Wat we willen is een interface die we kunnen implementeren om berekeningen en/of transformaties op een boom uit te voeren. We hoeven dan alleen de interface aan te roepen om het resultaat terug te krijgen.

   Hier vind je de generic interface voor de BinaryTree:

   ```
   public interface ApplyBinaryTree<T,U> {
      U apply(BinaryTree<T> bt);
      U apply(BinaryNode<T> node);
   }
   ```

   De Type Generator `T` staat voor het generieke datatype binnen de `BinaryTree` en `BinaryNode`; de Type Generator `U` staat voor het resultaat dat wordt teruggegeven. In het geval van opgave 3 is dat bijvoorbeeld een `Integer`. Ook zie je dat de methode `apply` overloaded is; zo werkt de Interface voor `BinaryTree` en `BinaryNode` op dezelfde manier. Het maakt dus niet uit of je `apply` met een `BinaryNode` of `BinaryTree` aanroept, Java zorgt ervoor dat de juiste methode geselecteerd wordt.

   (a) maak de Interface `ApplyBinaryTree<T,U>` aan;

   (b) maak een dergelijke interface voor de `Tree<T>` klasse.

7. (extra nuttig voor PO_Algoritmes) Implementeer de methode uit opgave 3 welke het aantal leafs telt met behulp van de interface `ApplyBinaryTree<T,U>` (je mag code uit de eerder gemaakte methode hergebruiken, maar je mag de eerder gemaakte methode niet aanroepen)

8. (extra nuttig voor PO_Algoritmes) Maak opgave 4 met de `ApplyBinaryTree<T,U>` interface. je mag code uit de eerder gemaakte methode hergebruiken, maar je mag de eerder gemaakte methode niet aanroepen)

9. (extra nuttig voor PO_Algoritmes) Maak met behulp van de interface uit 6b een implementatie welke een `Tree<Integer>` transformeert naar een `Tree<Integer>` waarbij iedere waarde met 1 is opgehoogd.

4.1.9 Binary-,Search- en AVL-bomen

Bestuderen

- Hoofdstuk 18.1 t/m 18.4 [In oud boek: 19.1 t/m 19.4]

Opdrachten uit het boek

- 18.1 [In oud boek: 19.4]
- 18.3 [In oud boek: 19.1]
- 18.4 [In oud boek: 19.2]

Overige opdrachten

1. Maak een AVL boom met de volgende waarden door ze achter elkaar toe te voegen:

   `10,1,15,5,2,3,20,6`

Praktijk

1. (verplicht voor PO_Algoritmes) Maak een BinarySearchTree waarin de values ints zijn.
2. (verplicht voor PO_Algoritmes) Maak 18.15 [In oud boek: 19.16]
3. Maak methodes `remove(int x)`, `insert (int x)`, bestudeer goed de implementaties in het boek.

4.2 Programmeertalen

Deze opdrachten horen bij de lessen in het thema Programmeertalen. Bij dit onderdeel gebruiken we geen boek. En zijn vooral praktijkgerichte opdrachten.

4.2.1 Taalontwerp

Bestuderen

- Slides uit de les

Kennis

1. Wat bedoelen we met de syntax van een programmeertaal?
2. Wat bedoelen we met de semantiek van een programmeertaal?
3. Wat bedoelen we met de pragmatiek van een programmeertaal?

Opdrachten

1. Voor bijna iedere programmeertaal is op internet wel te vinden wat de grammatica van die taal is. Hoe die grammatica precies vastgelegd is wel nog wel eens verschillen, maar vaak wordt een vorm van BNF gebruikt.
   (a) Ga voor een programmeertaal naar keuze op zoek naar de beschrijving van de grammatica van die taal.
   (b) Zoek uit op welke manier deze grammatica beschreven is. Is het BNF of iets dat er op lijkt
   (c) Indien het antwoord op de vorige vraag niet BNF was, hoe wijkt deze manier van beschrijven af van BNF?
2. Stel een grammatica in BNF of EBNF op voor reals. Voorbeelden die moeten matchen zijn ”3.14”, ”42”, -0.34”, ”3.4e3”, ”1.2E-10”

4.2.2 Taalimplementatie

Bestuderen

- Slides uit de les

Kennis

1. Wat zijn de standaardonderdelen van een compiler? Beschrijf in eigen woorden wat elk onderdeel doet.
2. Waarin verschilt een compiler van een interpreter?

Opdrachten

Bij dit onderwerp zijn geen praktijkopdrachten.

4.2.3 Lexing en Parsing

Bestuderen

- Slides uit de les
- ”Getting started with Antlr v4” op `www.antlr.org`
- De Antlr documentatie op `https://github.com/antlr/antlr4/blob/master/doc/index.md`

Kennis

1. Wat is de toegevoegde waarde van een lexer als extra stap voor een parser?
2. Gegeven de volgende Antlr lexer regels:

```
NAME: [A-Za-z]+;
SEP: [,\.];
WS: [ \t\n]+ -> skip;
```

   (a) In welke tokens wordt de input “Wat leuk, een lexer.“ opgesplitst?
   (b) Wat betekent `-> skip` in de `WS` regel?

3. Hoe herken je wat lexer-regels en wat parser-regels zijn in een Antlr grammatica file?
4. Gegeven de volgende Antlr regels:

```
loop: FORKEYWORD OPENPAR assignment condition increment CLOSEPAR
        OPENBRACE body CLOSEBRACE;

assignment: IDENTIFIER ASSIGNOP INTEGER SEMI;
condition: IDENTIFIER comparator INTEGER SEMI;
comparator: LT | GT;
increment: IDENTIFIER INCOP;
body: statement*;
statement: call | assignment;
call: IDENTIFIER OPENPAR args CLOSEPAR SEMI;
args: INTEGER (COMMA INTEGER)*;

LT: '<';
GT: '>';
ASSIGNOP: '=';
INCOP: '++';
OPENPAR: '(';
CLOSEPAR: ')';
OPENBRACE: '{';
CLOSEBRACE: '}';
COMMA: ',';
SEMI: ';';

FORKEYWORD: 'for';
IDENTIFIER: [a-z]+;
INTEGER: [0-9]+;
WS: [ \t\n]+ -> skip;
```

   Teken de parse tree die je krijgt als je de volgende input parseert:

```
for(i = 0; i < 100; i++) {
       print(100);
}
```

Opdrachten

1. Download en installeer Antlr 4 en test je installatie met het “configuratie” voorbeeld uit de slides.
2. Maak een Antlr grammatica `Expressions.g4` waarmee je eenvoudige sommen kunt parseren. Deze expressies bestaan uit gehele getallen, vermenigvuldigen en optellen. Ook kan mag er whitespace voorkomen tussen de getallen en operatoren. Voorbeelden die de grammatica moet accepteren zijn:
   - $3 + 4$
   - $2 + 3 * 5$
   - $43 * 3 + 52$
   - $1 + 22 + 3 + 44 * 5 * 65 + 7$

   (a) Definieer de Antlr lexer-regels om de sommen op te splitsen in tokens.

   (b) Definieer de Antlr parser-regels om de expressies te parseren. Definieer in ieder geval de regel `expression`. Wanneer nodig mag je natuurlijk meer regels toevoegen.

   (c) Respecteert je grammatica uit de vorige opgave de gebruikelijke rekenregels (vermenigvuldigen boven optellen)? Zo niet, pas je grammatica aan zodat hij dit wel doet?

4.2.4 Evaluatie en Checking

Bestuderen

- Slides uit de les

Kennis

1. Wat is de rol van de checker in de compiler pipeline?
2. Wat is het verschil tussen een parse tree en een abstract syntax tree?

Opdrachten

1. In de opgaven van 4.2.3 heb je een Antlr grammatica opgesteld voor eenvoudige sommen.

   (a) Maak een aantal klassen waarmee je ASTs van deze sommen kunt representeren. Bijvoorbeeld een abstracte `Expression` klasse met concrete subklasses voor een `Number`, `Addition` en `Multiplication`.

   (b) Maak een Antlr listener klasse waarmee je de gesparseerde sommen omzet naar de bijbehorende AST’s.

2. Implementeer een `Evaluator` klasse met een methode `int eval(Expression ast)` die de sommen uitrekent.

4.2.5 Transformatie en Optimalisatie

Bestuderen

- Slides uit de les

Kennis

1. Welke redenen zijn er om een AST te transformeren in een compiler?
2. Wat doet een inlining transformatie?

4.2.6 Codegeneratie

Bestuderen

- Slides uit de les

Kennis

1. Wat is de rol van de codegenerator in de compiler pipeline?

4.3 Paradigma's

Deze opdrachten horen bij de lessen in het thema Programmeertalen. Bij dit onderdeel gebruiken we het boek van Tate. Wanneer in dit deel naar opdrachten uit "het boek" verwezen wordt dan is dat nu dus dit boek.