Windesheim

Practicum Algoritmen & Datastructuren (96056)

Vakcode	:	ICT.SE.AD.V19 (ICT.SE.AD.V18,V16) (t1)
Datum	:	donderdag 31 oktober 2019
Tijd	:	08.30 - 11.30 uur

Klas:	Lokaal:	Aantal:
ICTSE3a t/m ICTSE3d, herkansers	T4.03, T4.09, T4.19, T4.18	145

Opgesteld door	:	Ernst Bolt, Arjen Jansen
Docenten	:	Ernst Bolt, Arjen Jansen, Eltjo Voorhoeve, Tijn Witsenburg
Gecontroleerd door	:	Eltjo Voorhoeve
Rekenmachine	:	alleen Casio FX-82**, Texas TI-30**, HP-10S**
Literatuur	:	Data Structures & Problem Solving Using Java – Mark Allen Weiss
Overige hulpmiddelen	:	Zie instructie
Opgaven inleveren :	:	ja

CONTROLEER VOORAF DE VOLGENDE GEGEVENS:

Dit tentamen bevat:

3 opgaven

8 genummerde pagina's

Waarschuw de surveillant als één van deze aantallen niet klopt!

Studentnummer	Naam	Klas	Cijfer
Tijd van inleveren:			

Instructies

- Niet toegestaan is het gebruik van telefoons, social media, forums, dropbox en alles wat je in contact met andere personen kan brengen. Als dit toch gebeurt, is er sprake van fraude en wordt dit aangegeven bij de examencommissie.
- Het is niet toegestaan andere hulpmiddelen te gebruiken dan het boek, de sheets, je huiswerk en materiaal over de gebruikte programmeertaal. Alle uitgewerkte opgaven moeten je eigen werk zijn.
- Programmeer alle functionaliteit zo efficiënt mogelijk. Overbodige of onnodig complexe code kan leiden tot puntenaftrek.
- In totaal zijn voor de vragen 90 punten te halen. Het eindcijfer wordt verkregen door de behaalde punten te delen door 10 en te vermeerderen met 1. Het laagst te behalen cijfer is een 1.

Voorbereiding

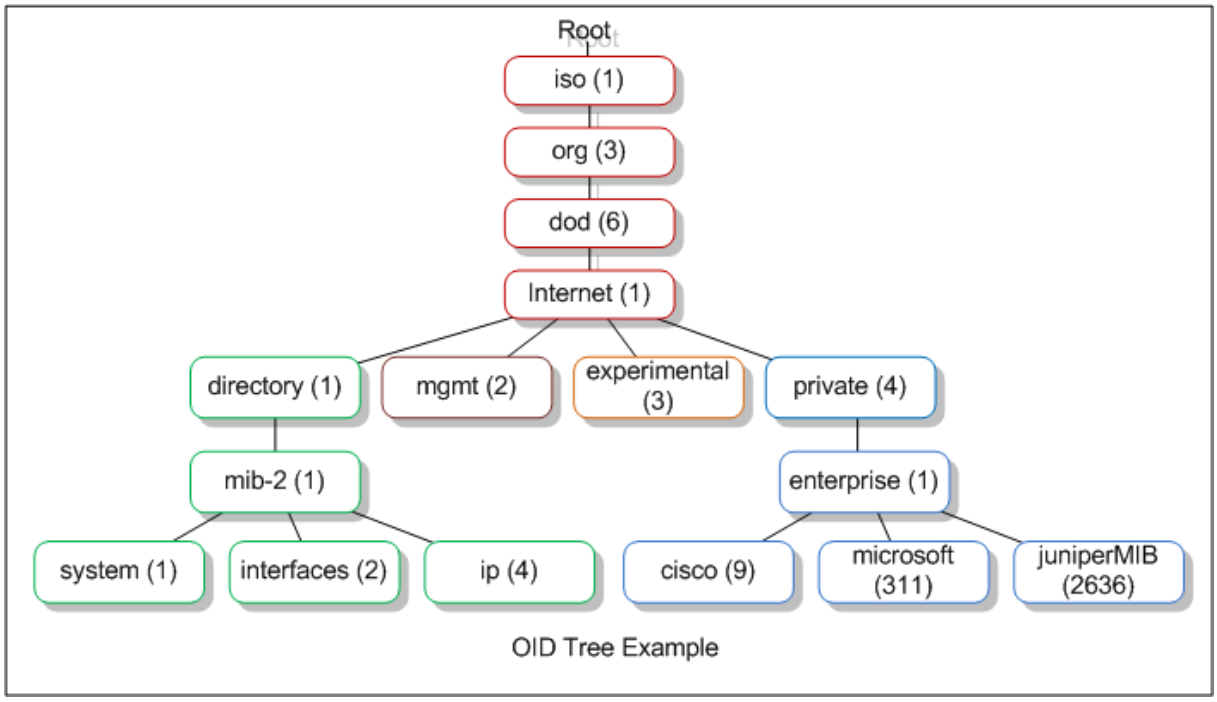
- Download het template project van de ELO (module ICT.SE.AD.V19, map Toetspracticum).
- Zorg er voor dat je je exact houdt aan de naamgeving van klassen, methodes, interfaces e.d. zodat het project eenvoudig te unit-testen is. De template projecten bevatten ook een aantal voorbeeld unit-tests. Deze tests mag je niet aanpassen, maar je mag er wel nieuwe tests bijschrijven. Je mag ook testcode schrijven in `Program.cs`.
- De gegeven unittesten zijn niet toereikend. Voor het nakijken worden meer unittests gebruikt dan gegeven, dus test je oplossing!
- Voor dit practicum dien je de programmeertaal C# te gebruiken.

Extra aanwijzingen m.b.t. het proefpracticum

- Dit zijn 2 (van de oorspronkelijk 3) opgaven van het practicum in 2019/2020.
- Pak de RAR uit in een subfolder van ~/src/Exam. Vanuit de solution worden verwijzingen gebruikt naar ~/src/datastructures, dus let goed op waar je de RAR uitpakt.
- Pas niet je werkende versies van je datastructuren aan! Het is aan te raden eerst een kopie te maken van je codebase. Zo begin je aan het echte practicum met een “schone” datastructuur.

Opgave 2 (Bomen, 20 punten)

In network management wordt gebruik gemaakt van objecten die geïdentificeerd worden met een zogenaamde ASN.1 notatie. Deze notatie bestaat uit een reeks van getallen, gescheiden door een punt (‘.’). Dit zijn de OID’s van het object. In onderstaand figuur is object Internet te identificeren met 1.3.6.1 en microsoft met 1.3.6.1.4.1.311. Een visuele representatie hiervan kan volgens onderstaande boom worden weergegeven.

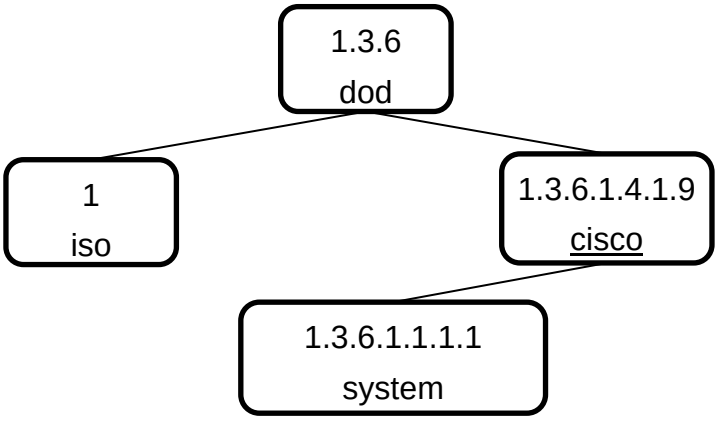


OID Tree Example

De figuur hierboven is slechts een voorbeeld van een representatie. Wij gaan een representatie maken met behulp van een BinarySearchTree. We noemen dit een `MIBTree`. Iedere node (`MIBNode`) in deze `MIBTree` heeft als key het volledige OID van het object. Als extra attribute wordt op iedere `MIBNode` ook de naam van het object opgenomen. Omdat een object binnen de ASN.1 notatie meer dan twee kinderen kan hebben en de volgorde van toevoegen van objecten aan de `MIBTree` zijn positie hierin bepalen zal de `MIBTree` er anders uitzien dan de boom structuur als hierboven weergegeven.

Zo zal de `MIBTree` na achtereenvolgens toevoegen van:

[1.3.6,dod] ,[1.3.6.1.4.1.9,cisco], [1,iso] ,[1.3.6.1.1.1.1,system] er als volgt uitzien:



Voor de implementatie van de `MIBTree` gaan we gebruik maken van de `BinarySearchTree` uit het huiswerk. In de solution is de `MIBTree` en `MIBNode` opgenomen. Vul deze aan met je eigen `BinarySearchTree` implementatie.

De constructor van de `MIBTree` bouwt een boom. Deze mag niet gewijzigd worden. De boom die hierin wordt opgebouwd is de volledige boom uit het eerste plaatje “OID Tree example”. De nodes in deze boom zijn tijdens de opbouw in willekeurige volgorde aangeboden. Deze boom wordt ook in de onderstaande unit tests gebruikt.

a) [10 punten] Implementeer de volgende method in `MIBTree`:

```
public MIBNode FindNode(string oid)
```

Deze method zoekt in de `MIBTree` naar de node met het meegegeven oid. De gevonden node wordt teruggegeven. Is de node niet in de boom opgenomen dan wordt `null` teruggegeven.

De volgende unit tests zijn bijgeleverd:

Testnaam	*Input*	*Verwacht resultaat*
`Ex2a_FindNodeSuccess`	“1.3.6.1”	{“1.3.6.1”,”internet”}
`Ex2a_FindNodeFail`	“1.3.6.2”	`Null`

b) [10 punten] Implementeer de volgende methode in `MIBTree`:

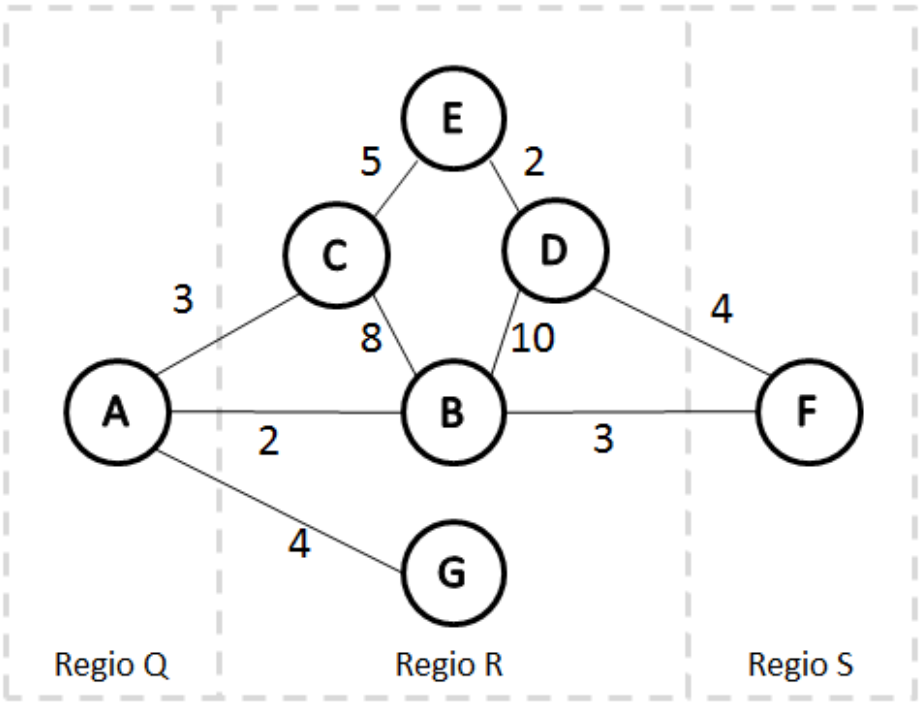
```
public bool AllNodesAvailable(string oid)
```

Deze methode geeft `true` terug als de node met het meegegeven OID en al zijn parent nodes in de `MIBTree` voorkomen, anders wordt false teruggegeven.
Als bijvoorbeeld het OID “1.3.6.1.1” wordt meegegeven dan is het resultaat `true` als de nodes met OID “1”, “1.3”, “1.3.6”, “1.3.6.1”, en “1.3.6.1.1” allen in de `MIBTree` voorkomen.
Deze method **moet recursief** geïmplementeerd worden, anders worden er geen punten toegekend! Je mag hiervoor hulp methodes met aanvullende argumenten gebruiken. Ook mag je gebruik maken van string operaties zoals `Split()`.
De volgende unit tests zijn bijgeleverd:

Testnaam	*Input*	*Verwachte output*
`Ex2b_AllNodesAvailableSuccess`	“1.3.6.1.4.1.311”	`True`
`Ex2b_AllNodesAvailableFail`	“1.3.6.1.4.1.312”	`False`

Opgave 3 (Grafen, 35 punten)

Gegeven de volgende gewogen ongerichte graaf, waarbij elke vertex in een regio gelegen is:



a) [10 punten] Breid de klasse Graph uit met de volgende method:

```
public string AllPaths()
```

Wanneer Dijkstra op bovenstaande graaf is uitgevoerd, dan is het resultaat van de method `AllPaths`:

```
A; B<-A; C<-A; D<-F<-B<-A; E<-C<-A; F<-B<-A; G<-A;
```

Voor elke vertex wordt het pad vanaf de startvertex naar deze vertex beschreven. De vertices worden gescheiden door "<-" en aan het eind van elk pad staat een ";".
Je hoeft geen rekening te houden met whitespaces. Je zou dus ook kunnen kiezen voor het volgende, beter leesbaar, resultaat:

```
A;
B<-A;
C<-A;
D<-F<-B<-A;
E<-C<-A;
F<-B<-A;
G<-A;
```

TIP: doe dit niet recursief.

In deze opgave implementeer je het kortste pad algoritme op basis van het Dijkstra algoritme, waarbij aanvullend geldt dat vanuit een regio **niet teruggekeerd mag** worden naar een reeds bezochte regio. Het pad mag wel door **meerdere vertices in één regio** lopen.

b) [5 punten] Voer het volgende uit:

- Implementeer een nieuwe constructor van `Vertex`:

```
public Vertex(string name, string regio)
```

- Breid de Graph klasse uit met de method:

```
AddVertex(string name, string regio)
```

  Deze method voegt een nieuwe vertex toe aan de graaf.

c) [5 punten] Om een niet gerichte graaf te maken, moet op dit moment in de klasse `Graph` elke vertex dubbel worden toegekend:

```
g.AddEdge("A", "B", 2);
g.AddEdge("B", "A", 2);
g.AddEdge("A", "C", 3);
g.AddEdge("C", "A", 3);
```

Breid de klasse `Graph` uit met de volgende method:

```
public void AddUndirectedEdge(string source,
                              string dest, double cost)
```

Deze method dient een ongerichte edge toe te voegen aan een vertex.

d) [15 punten] Breid de klasse Vertex uit met een hashmap (`HashSet`) van strings, zodat de reeds bezochte regio's van een vertex kunnen worden opgeslagen (denk ook aan de method `Reset`).
Breid de method `Dijkstra` in `Graph` uit. Zorg dat voor je een vertex toevoegt op de priority queue, wordt gecontroleerd of deze vertex in een regio staat die in het verleden al is bezocht. Is dit niet het geval en hebben we een kortere afstand, zorg dan (naast het updaten van `prev` en `distance`) dat de hashmap van de nieuwe vertex wordt aangepast.
Wanneer je het algoritme runt, zul je zien dat het resultaat van `AllPaths` verandert:

```
A;
B<-A;
C<-A;
D<-E<-C<-A;
E<-C<-A;
F<-B<-A;
G<-A;
```

De kortste route van A naar D was eerst A->B->F->D, maar nu A->C->E->D.

Inleveren

- Maak 1 RAR -bestand met daarin je oplossingen
- Noem dit RAR -bestand <studentnr>-<naam>.rar (bijvoorbeeld “s1234567-Piet-de-Vries”)
- Upload het RAR -bestand op elo.windesheim.nl, module ICT.SE.AD.V19

Einde Tentamen

Vergeet niet je naam, klas en studentnummer op iedere pagina te vermelden

---