TENTAMEN

Imperatief Programmeren

22 augustus 2006

Opgave 1 (Op stage)

Deze opgave gaat over het verdelen van een aantal stagiaires over (een even groot aantal) stageplaatsen. Je kunt natuurlijk de mensen willekeurig over de bedrijven verdelen, maar dat levert waarschijnlijk niet een optimale situatie. Een stage levert het meest op als de stageopdracht aansluit bij de belangstelling van de stagiaire. Omgekeerd zal ook de aanbieder van de stageplaats wensen hebben met betrekking tot de persoonlijkheid en de capaciteiten van de stagiaire.

We stellen ons voor dat de bedrijven de stageopdrachten omschrijven en dat de stagiaires elk een CV opstellen. Op grond van deze informatie bepaalt elke stagiaire een voorkeurslijstje met bedrijven en elk bedrijf bepaalt een voorkeurslijstje van stagiaires. Om dat goed te kunnen modelleren worden zowel de stagiaires als de bedrijven genummerd (vanaf 0). Een voorkeurslijstje wordt gerepresenteerd door een array, waarin de nummers op volgorde van voorkeur staan: op de eerste plaats het bedrijf danwel de stagiaire met de hoogste voorkeur, op de tweede plaats de volgende, enzovoort. Zo'n lijstje is uitputtend: alle nummers komen er (precies één keer) in voor.

Aangezien stagiaires en bedrijven met dezelfde soort gegevens worden gemodelleerd, maken we daar één klasse voor:

```
class Kaart {
  String naam;
  int [] pref;
  int gekoppeldAan;
  int act = 0;

} // Kaart
```

De string `naam` administreert de naam van de stagiaire danwel van het bedrijf; het array `pref` bevat de nummers van bedrijven danwel stagiaires in volgorde van voorkeur; de variabele `gekoppeldAan` administreert aan welk bedrijf de stagiaire danwel aan welke stagiaire het bedrijf is gekoppeld. De rol van de variabele `act` zal pas verderop in de opgave aan de orde komen. Bij de eerste onderdelen heb je deze variabele niet nodig.

In elk van de onderdelen mag je gebruik maken van voorgaande onderdelen, ook als je niet in staat bent geweest deze uit te werken!

lees verder

[3 pt] □ 1. Voorzie de klasse `Kaart` van een constructor. Deze constructor heeft als parameters de naam van de stagiaire/het bedrijf en het lijstje met voorkeuren. De gegevens uit dit lijstje moeten worden gekopieerd in het array `pref`; een enkele toekenning is dus niet voldoende.

[3 pt] □ 2. Voorzie de klasse `Kaart` van een methode `indexOf` die bij een nummer (van bedrijf/stagiaire) de bijbehorende index in het array `pref` oplevert.

[3 pt] □ 3. Implementeer in de klasse `Kaart` de methode

```
boolean prefereert (int a, int b) {
    // a heeft een hogere prioriteit dan b

} // prefereert
```

We veronderstellen nu twee arrays

```
Kaart [] stag;
Kaart [] bedr;
```

met dezelfde lengte, die de gegevens van een aantal stagiaires en een zelfde aantal bedrijven administreren.

Neem aan dat iemand "met de hand" een toedeling van studenten aan bedrijven heeft gemaakt en deze gegevens ook in de arrays heeft verwerkt. Daar kan natuurlijk gemakkelijk iets mis gaan. Bijvoorbeeld: bij een stagiaire staat een bedrijf geadministreerd, maar bij dat bedrijf staat een andere stagiaire genoteerd. We noemen een toedeling *consistent* als dit soort fouten niet voorkomt.

[6 pt] □ 4. Geef een methode `isConsistent` die oplevert of de geadministreerde toedeling consistent is. Je mag daarbij aannemen dat de geadministreerde waarden inderdaad nummers van bedrijven danwel stagiaires zijn.

Neem nu aan dat de toedeling consistent is. Een toedeling is *instabiel* als door het overlopen van een stagiaire naar een ander dan het hem/haar toegewezen bedrijf zowel de stagiaire als het bedrijf er op vooruitgaat. Dat wil zeggen dat er in de toedeling twee koppelingen `(stagA, bedrB)` en `(stagX, bedrY)` voorkomen waarvoor geldt dat `stagA` liever bij `bedrY` had willen komen dan bij bedrijf `bedrB` en dat `bedrY` liever stagiaire `stagA` had willen hebben dan `stagX`.

[8 pt] □ 5. Schrijf een methode `isStabiel` die oplevert of de toedeling stabiel is.

lees verder

In plaats van de koppelingen met de hand te maken, is het natuurlijk veiliger om dit met behulp van een (correct!) programma te doen. De bedoeling van de rest van deze opgave is dat je een programma schrijft dat een stabiele toedeling realiseert. Het zou wat te ver voeren om van je te verlangen dat je dit algoritme zelf bedenkt; dat komt misschien wel ergens in het tweede jaar. We formuleren in natuurlijke taal hoe het algoritme (zo ongeveer) werkt; daarna mag je het in detail uitwerken in java.

Bij elke stagiaire (en bij elk bedrijf) is een lijstje met voorkeuren geadministreerd. Gedurende de uitvoering van het algoritme kunnen deze lijstjes worden ingekort. Daar gebruiken we de variabele `act` voor. Deze variabele wijst in het array `pref` de actuele positie aan. De variabele `act` wordt nooit in waarde verlaagd; door `act` in waarde te verhogen wordt in feite de lijst met voorkeuren ingekort of anders gezegd: er worden bedrijven uit de voorkeurslijstjes van de stagiaire geschrapt. Op deze manier representeren `pref` en `act` samen het actuele lijstje van voorkeuren.

Het algoritme eindigt als iedere stagiaire een stageplaats heeft toegewezen gekregen. Zolang dat niet het geval is, kiest het algoritme een (willekeurige) stagiaire `S` die nog niet aan een bedrijf is gekoppeld. Laat nu `B` het bedrijf zijn dat in het actuele lijstje van `S` bovenaan staat. Als `B` nog niet aan een stagiaire is gekoppeld, dan worden `S` en `B` aan elkaar gekoppeld. Als `B` al wel is gekoppeld, maar dan aan een stagiaire die een lagere voorkeur heeft dan `S`, dan wordt die koppeling ongedaan gemaakt (en uit het lijstje van die stagiaire wordt `B` geschrapt) en `S` en `B` worden aan elkaar gekoppeld. In het overblijvende geval wordt het lijstje van `S` ingekort.

[3 pt] □ 6. Beargumenteer dat het beschreven algoritme eindigt.

[9 pt] □ 7. Geef een methode die bij een array van stagiaires en een even lang array van bedrijven bovenstaand algoritme implementeert.

lees verder

Opgave 2 (Sorteren)

Sorteren blijft een belangrijke operatie in software. De meeste elementaire algoritmen (selection sort, insertion sort, bubble sort) hebben een kwadratische tijdcomplexiteit. Maken we gebruik van merge sort, dan reduceren we de tijdcomplexiteit tot $n \cdot \log n$.

Maar het kan nog efficiënter. Tenminste als we de volgende aanname maken: de waarde waarop de objecten moeten worden gesorteerd heeft maar een beperkt domein. Bijvoorbeeld: voor het maken van een verjaardagskalender van een grote groep mensen is het voldoende om te sorteren op de geboortedatum. De naam en zelfs het geboortejaar kunnen worden genegeerd. Er zijn zo maar 366 verschillende waarden die bij het sorteren een rol spelen.

We bijken in deze opgave objecten van de klasse `Ding`:

```
class Ding {
  int wrd;  // 0 <= wrd < 250

  ...

} // Ding
```

Objecten van de klasse `Ding` hebben een attribuut `wrd` die gehele waarden van en met 0 tot aan 250 kan aannemen. Wat er verder in deze objecten zit is niet van belang.

De bedoeling is om een implementatie te maken voor de methode

```
Ding [] sorteer (Ding [] ding)
```

die de elementen van het array `ding` in gesorteerde volgorde oplevert.

[4 pt] □ 8. Introduceer in de methode `sorteer` een array `freq` en voeg code toe, zodat

`freq[w]` = aantal elementen van `ding` met `wrd` = `w`

[1 pt] □ 9. Introduceer in de methode `sorteer` een array `res` die de elementen van `ding` in gesorteerde volgorde moet gaan bevatten.

[4 pt] □ 10. Introduceer in de methode `sorteer` een array `plaats` en voeg code toe, zodat

`plaats[w]` = de index in `res` waar het eerstvolgende object met `wrd` = `w` moet worden geplaatst.

[6 pt] □ 11. Completeer de implementatie van de methode `sorteer`.

einde

Maximaal aantal punten: 50