INFDEV016A – Algoritmiek Practicumopdracht

Algemene informatie

- Je mag de opdracht in *groepen* van twee studenten uitvoeren. Het **verslag** over de opdracht moet *individueel* zijn (code mag hetzelfde zijn, maar antwoorden en de uitleg moeten verschillend zijn).
- Je mag de *programmeertaal* kiezen: Java, C# of Python. Als je gebruik wil maken van een andere taal, bespreek dit dan eerst met je docent.
- Je moet zowel bestaande taal-functionaliteiten gebruiken (zoals JCF in Java) als sommige fundamentele datastructuren en algoritmen implementeren.
- Uiterste datum voor de *inlevering*: toetsweek (dag van de schriftelijke toets).
    - Voor de inlevering, gebruik het "***verslag sjabloon***" (report template). Dat zal straks op de cursus website beschikbaar zijn. Je kan verder instructies over de inlevering in het verslag sjabloon vinden.
- *Voorwaarden*: de practicumopdracht zal meetellen voor het eindcijfer, als het tentamen voldoende is.
- *Eindcijfer*: Het gewogen gemiddelde van het tentamen (10%) en practicumopdracht (90%).

Casestudy omschrijving

Denk na over een fotowinkel die fotoproducten (zoals fotoboeken, fotobekers, fotoshirts, visitekaartjes, kalenders, …) verkoopt. De winkel drukt, terwijl de klant wacht, foto's van USB-sticks af. De winkel slaat informatie over de klanten op in een gegevensbestand. Over elke klant, wordt de volgende informatie opgeslagen:

- Klant ID (uniek en automatisch samengesteld vanuit ander informatievelden)
- Achternaam
- Tussenvoegsel
- Voornaam
- Leeftijd
- Geslacht (M/V)
- Plaats
- Email adres

De winkel heeft ook een gegevensbestand voor bestellingen. Elke bestelling bevat de volgende informatie:

- Klant ID
- Bestelling ID
- Verwerking [Ja/Nee; Nee wanneer aangemaakt; wordt Ja wanneer de verwerking van de bestelling begint]
- Start tijd [zinloos als Verwerking=Nee]
- Duur

- Compleet [Ja/Nee]
- Dadelijk [Ja/Nee; Ja als de klant wacht de bestelling op; Nee anders]

Voorwaarde

Creëer twee klassen, een voor de **Klant**-informatie, een voor de **Bestelling**-informatie.

Scenario 1

*Denk na over het bestellingen-gegevensbestand. Stel dat de winkel maar twee drukkers heeft (een voor de "dadelijk" bestellingen, een voor de anderen, omdat ze langer duren). Elke drukker verwerkt de bestellingen als ze binnen komen. De verwerking van bestellingen volgt een FIFO patroon, dus gebruikt de winkel een **queue**-data-structuur voor elke drukker om de bestellingen op te slaan.*

- **Implementeer een queue datastructuur**.
    - Implementeer de **invoegprocedure** (wanneer een klant een nieuwe bestelling maakt) en de **verwijder procedure** (wanneer een bestelling compleet is).
    - Om de bijgewerkte staat van de bestelling te behouden, implementeer de **update procedure** die checkt de tijdstempel van bestellingen en…:
        - Zet de staat van de bestelling op "Compleet" als de huidige tijdstempel hoger dan de som van "start tijd" en "duur" is
        - Zet de staat van de bestelling op "Verwerking" als er is geen andere bestelling in verwerking staat en deze bestelling de volgende te verwerken is. In dit geval, zet de "start tijd" waarde ook op de huidige tijdstempel.

Scenario 2

Denk na over het klanten-gegevensbestand. Stel dat de winkel een array gebruikt om de klant-gegevens op te slaan. Stel dat de array niet wordt gesorteerd. Wanneer een nieuwe klant zich aanmeldt, wordt zijn informatie opgeslagen aan het einde van de array. De winkel heeft een procedure nodig om de klant-gegevens in gesorteerde volgorde (soms op de leeftijd vel, soms op de achternaam, …) af te drukken.

- Omdat de array niet gesorteerd is, in dit geval, is de beste sorteeralgoritme (vanuit de twee dat wij hebben geleerd) de **merge-sort**. De prestaties van merge-sort zijn onafhankelijk van de gegevens vorm (gesorteerde, deels gesorteerde, niet gesorteerde, …) en ze zijn in elke geval goed.
    - **Implementeer de merge-sort** om de array op het *leeftijd*-veld te sorteren.
- Als wij op zoek zijn naar een klant met een specifieke waarde in een veld (leeftijd, achternaam, …), moeten we in dit geval het **linear search** algoritme te gebruiken omdat de array niet gesorteerd is.
    - **Implementeer de linear search** procedure.
    - Als er meer is dan één klant met dezelfde specifiek veld waarde, welke wordt van de linear search gevonden?

De winkel merkt, na enige tijd, dat ze het klantenbestand voornamelijk op achternaam afdrukken. Ze willen dus de array altijd gesorteerde hebben op achternaam. De invoeging van een nieuwe klant voegt zijn informatie toe naar het einde van de gesorteerde array en moet daarna de array opnieuw sorteren.

- Omdat de array al bijna gesorteerd is, in dit geval is de beste sorteeralgoritme (vanuit de twee die we geleerd hebben) **insertion-sort**. De prestaties van insertion-sort zijn lineair wanneer de data al gesorteerd is.
    - **Implementeer de insertion-sort** om de array op het *Achternaam*-veld te sorteren
    - Hoe kan jij het algoritme veranderen om nog beter prestaties te krijgen, wetende dat alleen de laatste element van de array is niet gesorteerde?
- Als wij op zoek zijn naar een klant met een specifieke achternaam, kunnen we de **binary search** algoritme gebruiken omdat de array is nu gesorteerde naar achternaam.
    - **Implementeer de binary search** procedure.
    - Als er meer dan een klant is met dezelfde achternaam, welke wordt van de binary search gevonden?
        - Veranderen het algoritme om alle klanten met dezelfde specifiek achternaam te vinden.

Scenario 3

Na enige tijd, merkt de winkel dat het gegevensbestand te groot wordt. Dus besluit de winkel om het gegevensbestand periodiek bij te werken, door het verwijderen van oude klanten. Met "oude klanten" bedoelen wij "klanten die geen bestellingen in de laatste maand hebben gedaan". Hierdoor gebeuren invoeging en verwijdering beide vaak.

- Waarom is een array niet meer de beste manier om gegevens op te slaan?
- Om nog betere prestaties te krijgen, in dit geval besluit de winkel een **binary tree** datastructuur te gebruiken (i.p.v. een array).
    - **Implementeer een binary tree** data structuur met de invoeg en verwijder procedures.
        - Opmerking 1: de data structuur moet nu naar *Klant ID* (i.p.v. *Achternaam*) gesorteerde zijn.
        - Opmerking 2: Implementeer en gebruik de *comparator*-methode.
        - Opmerking 3: geen zorgen over *welke* bepaalde klant verwijderd moet worden. Implementeren een functie die, gegeven een *Klant ID*, de bijbehorende klant verwijdert en de datastructuur gesorteerd houdt.
    - Om de klantenlijst gesorteerd naar *Klant ID* af te drukken, nu mogen wij het **in-order traversal** algoritme gebruiken om de data structuur te doorkruisen. **Implementeer dit algoritme**.