UNIVERSITEIT GENT

FACULTEIT WETENSCHAPPEN

Systeemprogrammeren

Academiejaar 2018–2019

sysprog@lists.UGent.be

Oefeningenset 4
C++

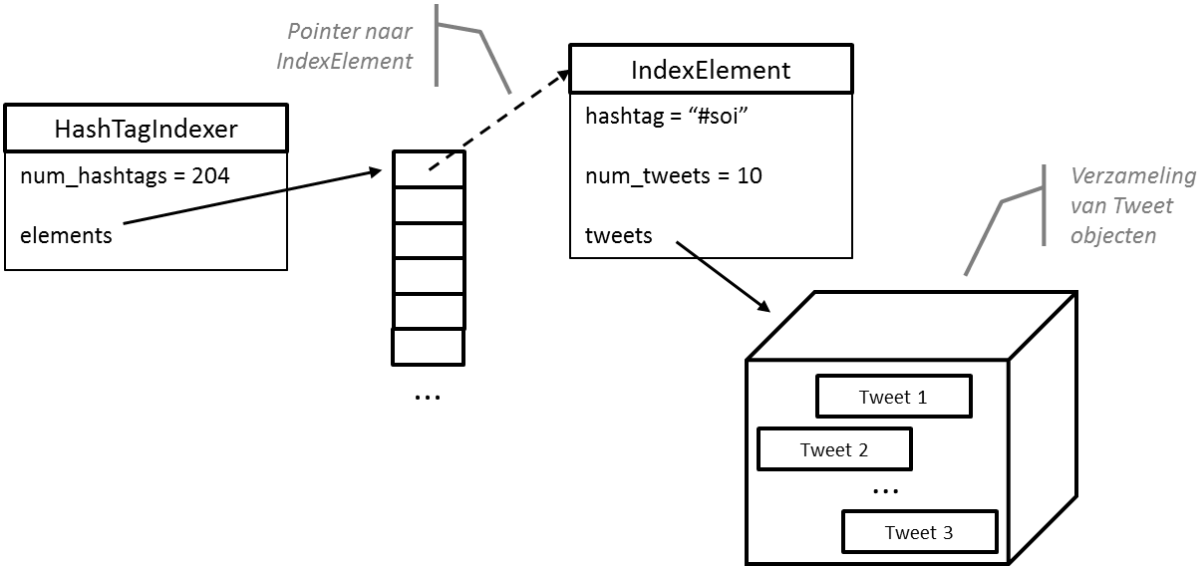
Associatieve array

Situering

Een associatieve array, ook wel een map of een dictionary genoemd, is een datastructuur waarmee door middel van een sleutelobject een ander object gezocht wordt. In tegenstelling tot een gewone array waar de index de vorm van een getal aanneemt, hoeft de index bij een associatieve array geen getal te zijn, maar kan het bijvoorbeeld ook een string zijn. Deze datastructuur laat toe om op een efficiënte manier een waarde op te halen die bij een bepaalde sleutel hoort, vaak ten koste van complexere toevoeg- en verwijder-operaties. In deze opgave zal een associatieve array geïmplementeerd worden om op een efficiënte manier "tweets" op te zoeken op basis van een "hashtag".

Opgave

Het doel van deze opgave is om in de objectgeoriënteerde taal C++ een aantal klassen te implementeren, waaruit een associatieve array opgebouwd wordt die hashtags als sleutel neemt en een verzameling van `Tweet` objecten als waarden bevat. Deze structuur wordt schematisch weergegeven in Figuur 1.



Figuur 1 Schematische weergave van de HashTagIndexer datastructuur

Van deze datastructuur zullen twee afzonderlijke versies worden voorzien. In een eerste versie wordt de verzameling van Tweet objecten voorgesteld als een array waarvan de grootte dynamisch wordt aangepast naarmate tweets worden toegevoegd. In een tweede versie wordt deze verzameling voorgesteld als een gelinkte lijst.

Test-gedreven Software Ontwikkeling

Verder zullen jullie in deze opgave kennis maken met het principe van *Test Driven Development (TDD)*. TDD is een methode uit de software ontwikkeling waarbij eerst tests worden geschreven en daarna pas de eigenlijke code. Op deze manier wordt naar de software gekeken vanuit het perspectief van de gebruiker. De testcases die worden voorzien, zijn namelijk gebaseerd op de requirements, die vanuit het

oogpunt van de gebruiker zijn opgesteld. Zo kan op een eenvoudige manier worden nagegaan of elke methode of functie wel aan de gevraagde interface voldoet.

De ontwikkelcyclus ziet er bij TDD ruwweg als volgt uit. Voor het implementeren van elke methode worden volgende stappen doorlopen:

1. **Test maken:** schrijf een of meerdere tests die de functionaliteit van de te implementeren methode testen. Vergeet hierbij de randgevallen en uitzonderingen niet te testen.

2. **Tests uitvoeren**: voer de tests uit om na te gaan dat deze tests falen. Zoniet bevatten de tests waarschijnlijk een fout.

3. **Code schrijven:** schrijf de code die de nieuwe, te testen functionaliteit bevat.

4. **Tests uitvoeren:** voer de tests opnieuw uit om te verifiëren dat de functionaliteit correct werd geïmplementeerd en geen randgevallen uit het oog verloren werden. Hierbij is het ook belangrijk om na te gaan of alle voorgaande tests nog slagen, dit om te vermijden dat afhankelijkheden tussen verschillende componenten voor onverwachte effecten zorgen.

Schrijven van testen

Gegeven is een bestand `main.cpp` waar reeds wat functionaliteit aanwezig is om de tests uit te voeren. Het vervolg van deze opgave bestaat uit een beschrijving van de verschillende klassen en bijhorende methodes die geïmplementeerd moeten worden. Voor een groot deel van deze methodes werd reeds een test voorzien in `main.cpp`. Voor volgende methodes dient de test echter nog door jullie geschreven te worden, **vooraleer** de eigenlijke methode te implementeren:

```
bool getTweeterTweetTest()

bool getHashtagTweetTest()

bool addTweetIndexElementTest()

bool getTweetIndexElementTest()

bool addTweetHashTagIndexerTest()
```

Elk van deze methodes moet de waarde `true` teruggeven indien de uitvoering van de methode correct was, `false` indien niet. Deze methodes worden opgeroepen vanuit de methode `void runTests()` en het resultaat ervan wordt afgedrukt gebruikmakend van de hulpfunctie `void printTestResult(string testname, bool testresult)`.

Implementeren van klassen

`Tweet`

Schrijf een klasse `Tweet` die volgende informatie van een tweet bijhoudt: een unieke id, de gebruikersnaam van de tweeter, het tijdstip waarop de tweet werd verstuurd, de boodschap, het aantal

bijgevoegde hashtags en de lijst van hashtags. Je mag er bij de implementatie van uitgaan dat er maximaal 5 hashtags bij een tweet horen, zodat een statische array gebruikt kan worden.

Implementeer minimaal deze methodes van de klasse `Tweet`:

- `Tweet(int id, std::string tweeter, time_t date, std::string tweet, std::string* tags, int num_tags)`
  Constructor die een nieuwe tweet initialiseert met de opgegeven variabelen.
- `Tweet()`
  Default constructor die alle numerieke parameters van de klasse instelt op 0 en alle string parameters op de lege string "".
- `int getID() const`
  Geeft de id van de tweet terug.
- `std::string getTweeter() const`
  Geeft de gebruikersnaam van de tweeter terug.
- `time_t getDate() const`
  Geeft het tijdstip terug waarop de tweet werd verstuurd.
- `std::string getTweet() const`
  Geeft de boodschap van de tweet terug.
- `int getNumHashtags() const`
  Geeft weer hoeveel hashtags er bij deze tweet horen.
- `std::string getHashtag(int i) const`
  Geeft de `i`-de hashtag van deze tweet terug (`i` >=0)

`IndexElement` (eerste versie, gebaseerd op arrays)

Schrijf een eerste versie van de klasse `IndexElement` waarbij gebruik gemaakt wordt van arrays. Een object van de klasse `IndexElement` linkt een hashtag aan een verzameling die alle `Tweet` objecten met deze hashtag bevat (zie schematische weergave in Figuur 1). In deze eerste versie worden de `Tweet` objecten bijgehouden in array.

Implementeer minimaal deze methodes van de klasse `IndexElement`:

- `IndexElement(std::string tag)`
  Constructor die een nieuw element initialiseert voor de opgegeven hashtag. Voorzie hierbij initieel ruimte voor 10 `Tweet` objecten. Maak hiervoor gebruik van een macro definitie.
- `IndexElement(const IndexElement& copy)`
  Constructor die een diepe kopie neemt van het meegegeven element.
- `std::string getHashTag() const`
  Geeft de hashtag terug die bij dit element hoort.
- `void addTweet(Tweet tweet)`
  Voegt de opgegeven `Tweet` toe aan de verzameling van tweets van dit element. Als er reeds een tweet met dezelfde id aanwezig is, dient niets te gebeuren. Indien de array reeds vol is, wordt zijn grootte verdubbeld.
- `int getNumTweets() const`
  Geeft weer hoeveel tweets dit element bevat.

- `Tweet getTweet(int i) const`
  Geeft de `i`-de tweet van dit element terug (`i`>=0). Indien dit element niet bestaat, wordt een default `Tweet` object (met alle numerieke parameters van de klasse op 0 en alle string parameters een lege string “”) teruggegeven.

HashTagIndexer

Schrijf een klasse `HashTagIndexer` die de eigenlijke associatieve array implementeert. Deze klasse bevat een array van pointers naar `IndexElement` objecten (zie schematische weergave in Figuur 1). De grootte van deze array is steeds gelijk aan het aantal `IndexElement`en (= het aantal hashtags) dat de `HashTagIndexer` bevat. Bovendien zijn de `IndexElement`en in deze array steeds gesorteerd volgens hashtag, wat efficiënte opzoekingen mogelijk maakt.

Implementeer minimaal deze methodes van de klasse `HashTagIndexer`:

- `HashTagIndexer()`
  Constructor die een nieuwe `HashTagIndexer` initialiseert.
- `HashTagIndexer(const HashTagIndexer& copy)`
  Constructor die een diepe kopie neemt van de opgegeven `HashTagIndexer`.
- `int getNumHashTags() const`
  Geeft weer hoeveel verschillende hashtags (= hoeveel `IndexElement`en) dit object bevat.
- `void addTweet(Tweet tweet)`
  Voegt de opgegeven `Tweet` toe aan de associatieve array voor elk van zijn bijhorende hashtags. Indien voor een bepaalde hashtag nog geen IndexElement aanwezig is, dient een nieuw element te worden toegevoegd. Zorg ervoor dat de lijst met `IndexElement`en ten allen tijde gesorteerd blijft volgens hashtag.

Implementeer bovendien volgende operatoren voor de klasse:

- De +-operator: voegt twee associatieve arrays (`HashTagIndexer`s) samen. Indien een bepaalde tweet in beide elementen voorkwam, wordt slechts 1 kopie bijgehouden.
- De index-operator []: laat toe om op een efficiente manier alle tweets met een bepaalde hashtag op te vragen. Voor een object `indexer` van het type `HashTagIndexer` geeft `indexer["#SOI"]` een pointer naar het `IndexElement` terug dat alle tweets bevat met de hashtag “#SOI”. Maak hierbij gebruik van de gesorteerde eigenschap van de array met `IndexElement`en om deze opzoeking zo efficiënt mogelijk uit te voeren. Indien de `HashTagIndexer` geen `Tweet`s bevat met de opgegeven hashtag wordt `NULL` teruggegeven.
- De output operator <<: drukt de HashTagIndexer af in het volgende formaat:

```
===
HashTag A
===
 * Tweet 1 (dd/mm/yyyy hh:mm:ss, tweeter)
 * Tweet 2 (dd/mm/yyyy hh:mm:ss, tweeter)
 * ...
```

```
===
HashTag B
===
...
```

`LinkedListElement`

Schrijf een klasse `LinkedListElement` die een element in een lineair geschakelde lijst voorstelt. Een object van deze klasse houdt als waarde een `Tweet` object bij, alsook een pointer naar het volgende element in de geschakelde lijst (`NULL` als er geen volgend element is).

Implementeer minimaal deze methodes van de klasse `LinkedListElement`:

- `LinkedListElement(Tweet tweet)`
  Constructor die een nieuw element initialiseert met de opgegeven `Tweet` als waarde.
- `LinkedListElement(const LinkedListElement& copy);`
  Constructor die een diepe kopie neemt van de opgegeven gelinkte lijst. Zorg ervoor dat de pointer naar het volgende element in de lijst ook naar een diepe kopie van dit volgende element wijst (op deze manier wordt bij het maken van een diepe kopie van het anker van de gelinkte lijst, een diepe kopie genomen van de volledige gelinkte lijst).
- `Tweet getTweet() const`
  Geeft de `Tweet` terug die in het `LinkedListElement` wordt bijgehouden als waarde.
- `LinkedListElement* getNext() const`
  Geeft de pointer terug naar het volgende element in de gelinkte lijst.
- `void append(LinkedListElement* node)`
  Voegt het opgegeven element `node` toe nét achter het huidige element.
- `LinkedListElement* removeFromList(LinkedListElement* anchor)`
  Verwijder het huidige element uit de gelinkte lijst met als anker het opgegeven element `anchor`. Zorg ervoor dat de lijst geconnecteerd blijft na het verwijderen. Deze methode geeft een pointer terug naar het anker van de gelinkte lijst na de verwijdering (merk op dat deze gewijzigd zal zijn indien het te verwijderen element overeen komt met de anker van de gelinkte lijst).

`IndexElement` (tweede versie, gebaseerd op een gelinkte lijst)

Schrijf een tweede versie van de klasse IndexElement. Deze versie heeft ***dezelfde functionaliteit en dezelfde methodes als de eerste versie***, maar met een andere interne representatie. In tegenstelling tot de vorige versie zullen de `Tweet` objecten worden bijgehouden in een lineair geschakelde lijst in plaats van een array. Een object van de klasse `IndexElement` zal dus een pointer bijhouden die wijst naar het anker van de geschakelde lijst.

Belangrijk: Voor alle gevraagde klassen dient mimimaal de gevraagde functionaliteit geïmplementeerd te worden. Indien nodig is het toegestaan om hulpmethodes te definiëren. Duplicatie van code dient vermeden te worden.

Tweets inlezen uit een bestand

Tot slot zullen tweets worden ingelezen uit een bestand. Gegeven zijn de bestanden `tweets_lorem.txt` en `tweet_pangram.txt` die willekeurig gegenereerde tweets bevatten in volgend formaat:

```
id;tweeter;unix_timestamp;tweet;hashtag1,hashtag2,...,hashtag5
```

Elke tweet heeft minimaal 1 en maximaal 5 hashtags. Implementeer de methode `HashTagIndexer readFile(std::string file)` die een `HashTagIndexer` aanmaakt, het bestand met naam `file` inleest en de tweets toevoegt aan de indexer. De functionaliteit van deze methode wordt samen met de +-operator, de output operator << en de index operator [] getest in de methode `executeReadFile`.

Opmerking

In de oplossing maken we gebruik van `localtime_s` om met timestamps te werken. Deze functie is niet portabel (op POSIX systemen is er `localtime_r` die gelijkaardige functionaliteit biedt)