Софиийски университет "Св. Климент Охридски"
Факултет по математика и информатика

Проект

по

Мрежово програмиране

спец. Информатика, 3 курс, летен семестър,

учебна година 2018/19

Тема 70

Клиент-сървър приложение ползващо TCP протокола за търсене на зависимост в поведение на потребител чрез често използвани затворени елементи с помощта на алгоритъма AprioriClose.

16 май 2019 г.

Изготвил:

Иво Алексеев Стратев

София

Факултетен номер:
45342

Група: 3

Съдържание

1 Тема (Задание) на проекта

Реализирайте клиент и сървър приложения чрез TCP протокола за търсене на зависимост на поведение на потребител чрез реализирането на често използвани затворени елементи с помощта на алгоритъма AprioriClose.

2 Увод

Целта на проекта е да намери зависимости в поведението на потребител-/ли чрез често използвани затворени елементи с помощта на алгоритъма AprioriClose.
Честоизползвани елементи са множества от елементи, които се срещат поне определен брой пъти в данните.
Честоизползвани затворени елементи са затворени елементи, за които няма строго подмножество, което да се среща точно толкова пъти колкото тях.
Данните съдържат много на брой характеристики (реализацията разпознава 30), които определят поведението на потребителите и които биха ни интересували.
За това в текущата реализация има възможност да се избират, кои характеристики да бъдат използвани при изпълнението на алгоритъма, както и възможност за филтрация на входните данни.
Входните данни представляват таблица от събития на потребители, най-вероятно от дейности в учебна система (вероятно *moodle*), състояща се от следните колони:

- **Time** - Време на събитието във формат: ден/месец/година, час:минути.
- **Event context** - Контекст на събитието.
- **Component** - Компонент свързан със събитието.
- **Event name** - Вид на събитието.
- **Description** - Подробно описание.
- **Origin** - Източник на събитието. В примерните данни това е "web"или "cli".
- **IP address** - ip адрес.

2.1 Разпознавани характеристики

Характеристиките, които могат да бъда избирани и по които могат да бъдат филтрирани данните са:

- **Activity** - Активност свързана с дейност по модул на курс. В текущата реализация това е стойност от следния списък:
  - *resource*
  - *forum*
  - *label*
  - *page*
- **Affected user ID** - Идентификатор на потребител засегнат от действие на друг потребител.
- **Component** - Компонент свързан със събитието. Извилича се непосредствено от входните данни.
- **Course ID** - Идентификатор на курса свързан със събитието.
- **Course module ID** - Идентификатор на модула на курса, свързан със събитието.
- **Date** - Дата на събитието във формат: ден/месец/година.
- **Date and Time** - Време на събитието във формат: ден/месец/година, час:минути. Извилича се непосредствено от входните данни.
- **Day** - Ден от времето на събитието.
- **Description** - Пълно описание на събитието. Извилича се непосредствено от входните данни.
- **Discussion ID** - Идентификатор на дискусията свързана със събитието.
- **Enrolment ID** - Идентификатор на записването, свързан със събитието.
- **Enrolment method** - Метод на записването.
- **Event** - Име на събитието. Пример от тестовите данни: *Предаване на курсови работи по Програмни езици.*

- **Event context** - Контекст на събитието. Извилича се непосредствено от входните данни.
- **Event context kind** - Вид на контекстта. Във входните данни това е например: *File*, *Forum*, *Page*, *Other* и други.
- **Event ID** - Идентификатор на събитието, свързан със събитието от системата.
- **Event name** - Вид на събитието. Извилича се непосредствено от входните данни.
- **Event User ID** - Идентификатор на потребителя извършил събитието.
- **Hour** - Час от времето на събитието.
- **IP address** - ip адрес. Извилича се непосредствено от входните данни.
- **Item ID** - Идентификатор на обекта, свързан със събитието.
- **Minutes** - Минути от времето на събитието.
- **Module** - Модул на събитието.
- **Month** - Месец от времето на събитието.
- **Origin** - Източник на събитието. Извилича се непосредствено от входните данни.
- **Post ID** - Идентификатор на поста.
- **Resource name** - Име на ресурса свързан със събитието.
- **Role ID** - Идентификатор на роля.
- **Section number** - Номер на секция.
- **Time** - Време на събитието във формата час:минути.
- **Viewing mode** - Режим на преглед, свързан със събитието.
- **Year** - година от времето на събитието.

2.2 Колона на извличане на характеристиките

Списък на характеристиките по колони, от които се извличат:

- Time
  - Day
  - Month
  - Year
  - Hour
  - Minutes
  - Date
  - Time
  - Date and Time
- Event context
  - Event context
  - Event context kind
  - Resource name
- Component
  - Component
- Event name
  - Event name
- Description
  - Description
  - Event user ID
  - Affected user ID
  - Course ID
  - Course module ID
  - Discussion ID
  - Role ID
  - Item ID

  - Enrolment ID
  - Event ID
  - Post ID
  - Section number
  - Activity
  - Enrolment method
  - Event
  - Viewing mode
  - Module
- Origin
  - Origin
- IP address
  - IP address

За всеки ред от таблицата повечето характеристики нямат стойност. За това следните примери целят на да създат представа как се извличат характеристики в зависимост от стойностите в колоните.

2.3 Примери за извличане на характеристики от редове във входните данни

2.3.1 Пример 1

Входни данни:

- **Time** 2/01/18, 13:09
- **Event context** Course: Програмни езици
- **Component** System
- **Event name** Course viewed
- **Description** The user with id '6546' viewed the course with id '49'.
- **Origin** web
- **IP address** 77.70.17.159

Извлечени характеристики:

- **Day** 2
- **Month** 1
- **Year** 18
- **Hour** 13
- **Minutes** 09
- **Date** 2/1/18
- **Time** 13:09
- **Date and Time** 2/1/18, 13:09
- **Event context** Course: Програмни езици
- **Event context kind** Course
- **Resource name** Програмни езици
- **Component** System
- **Event name** Course viewed
- **Description** The user with id '6546' viewed the course with id '49'.
- **Event user ID** 6546
- **Course ID** 49
- **Origin** web
- **IP address** 77.70.17.159

2.3.2 Пример 2

Входни данни:

- **Time** 2/01/18, 12:17
- **Event context** Forum: Новинарски форум
- **Component** Forum

- **Event name** Course module viewed
- **Description** The user with id ’6216’ viewed the ’forum’ activity with course module id ’81’.
- **Origin** web
- **IP address** 130.204.135.146

Извлечени характеристики:

- **Day** 2
- **Month** 1
- **Year** 18
- **Hour** 12
- **Minutes** 17
- **Date** 2/1/18
- **Time** 12:17
- **Date and Time** 2/1/18, 12:17
- **Event context** Forum: Новинарски форум
- **Event context kind** Forum
- **Resource name** Новинарски форум
- **Component** Forum
- **Event name** Course module viewed
- **Description** The user with id ’6216’ viewed the ’forum’ activity with course module id ’81’.
- **Event user ID** 6216
- **Course module ID** 81
- **Activity** forum
- **Origin** web
- **IP address** 130.204.135.146

2.3.3 Пример 3

Входни данни:

- **Time** 17/01/17, 20:50
- **Event** Forum: Новинарски форум
- **Component** Forum
- **Event name** Some content has been posted
- **Description** The user with id '2436' has posted content in the forum post with id '250' in the discussion '209' located in the forum with course module id '81'.
- **Origin** web
- **IP address** 130.204.21.61

Извлечени характеристики:

- **Day** 17
- **Month** 1
- **Year** 17
- **Hour** 20
- **Minutes** 50
- **Date** 17/1/17
- **Time** 20:50
- **Date and Time** 17/1/17, 20:50
- **Event context** Forum: Новинарски форум
- **Event context kind** Forum
- **Resource name** Новинарски форум
- **Component** Forum
- **Event name** Some content has been posted

- **Description** The user with id ’2436’ has posted content in the forum post with id ’250’ in the discussion ’209’ located in the forum with course module id ’81’.
- **Event user ID** 2436
- **Course module ID** 81
- **Discussion ID** 209
- **Post ID** 250
- **Origin** web
- **IP address** 130.204.21.61

2.3.4 Пример 4

Входни данни:

- **Time** 1/10/16, 19:27
- **Event Course**: Програмни езици
- **Component** Forum
- **Event name** User report viewed
- **Description** The user with id ’4773’ has viewed the user report for the user with id ’4735’ in the course with id ’49’ with viewing mode ’posts’.
- **Origin** web
- **IP address** 84.242.168.87

Извлечени характеристики:

- **Day** 1
- **Month** 10
- **Year** 16
- **Hour** 19
- **Minutes** 27

- **Date** 1/10/16
- **Time** 19:27
- **Date and Time** 1/10/16, 19:27
- **Event context** Course: Програмни езици
- **Event context kind** Програмни езици
- **Resource name** Програмни езици
- **Component** Forum
- **Event name** User report viewed
- **Description** The user with id '4773' has viewed the user report for the user with id '4735' in the course with id '49' with viewing mode 'posts'.
- **Event user ID** 4773
- **Affected user ID** 4773
- **Course ID** 49
- **Viewing mode** posts
- **Origin** web
- **IP address** 84.242.168.87

Както може да се забележи от горния пример има събития, в които потребителя, който предизвиква събитието и този, който бива афектиран по някакъв начин е един и същ (идентификаторите им съвпадат). Понеже използваният алгоритъм работи с множества, то не бихме могли да разберем, че в горното събитие участват два потребителя, ако не пазим различна информация за идентификатора на потребителя предизвикал събитието и този, който бива афектиран.

3 Описание на решението на проблема

Реализирано е клиент-сървър приложение. Клиента приема като параметри, командни аргументи, в този ред:

- път до входния файл
- дробно число (наричано *minSupp*) оказващо какъв процент минимум на участие трябва да има едно множество за да бъде чест елемент.
- път до файла, в който да запише резултата

Характеристиките от, които се интересуваме се прочитат от входния поток на клиента - ред по ред.
В кода на клиента може да се избира филтър, който да ограничава редовете от таблицата, входния файл. Клиент прочита веднъж входния файл, филтрирайки редовете.
За всяка характеристика се създава и попълва хеш карта със стойностите на характеристиката. За ключ ползва стойността на характеристиката, а за стойността - стойността на вътрешен броя при първото срещане на съответната стойност на характеристика.
След което постепенно променя стойностите в хеш таблците, така че да запълнят лявата част на интервала от стойности на типа *Integer*, започвайки от $Integer.MIN\_VALUE$ и увеличвайки с размера на всяка хеш таблица. Тоест извършва интервално хеширане на стойностите на характеристиките.
Изпраща до сървъра стойността на дробното число *minSupp*. След което прочита втори път филтрираните редовре, извличайки избраните характеристики, създавайки масиви от цели числа, с хешираните стойности на характеристиките, които изпраща до сървъра.
Сървъра получава числото *minSupp* и масивите от цели числа, като междувременно ги записва във временен файл. Когато сървъра получи последния масив от цели числа, съответстваш на ред от характеристики изпълнява алгоритъма AprioriClose и връща резултата до клиента като отговор на неговата заявка.
Клиента от своя странна създава и попълва изходна таблица с колони избраните характеристки и стойности - декодираните чрез обратни хеш таблици, редове от числа, последвани от броя на срещанията, върнати от сървъра.
Сървъра изпълнява подобрена версия на алгоритъма AprioriClose, наречена Apriori_TIDClose, която ползва използва кеширано вертикално

представяне на данни с цел по-малко прочитания на входния файл с цел бързодействие. Реализацията е от SPMF open-source data mining library.

4 Описание на използвания алгоритъм

4.1 Описание на алгоритъма AprioriClose

4.1.1 Вход на алгоритъма

Алгоритъма приема дробно число наричано *minSupp*, което оказва какъв процент от срещанията трябва да има едно множество, за да е чест елемемнт и текстови файл от цели числа, в който всеки ред представлява една транзакция от базата данни. Числата във всеки ред трябва да са сортирани и без повторения.

4.1.2 Алгоритъм

Apriori и съответно AprioriClose работят по нива (последователно намират 1, 2, 3, и тн елементните множества с необходимите свойства). Алгоритъма започва с пресмятането на броя срещания на синглетона (едно елементното множество) на всяко число от файла. Премахва всички синглетони с подкрепа (брой срещания) по-малък от *minSupp*. Множеството от 1-елементните генератори са синглетоните, които не са били премахнати. След, което преминава към итеративна фаза.
Ако множеството на генериращите i-елементни множества не е празно, то намира кандидат множеството от генериращи $(i+1)$-елементни множества. Като за целта понеже редовете са сортирани, то се опитва да съедини два i-елементни генератора, за да образува кандидант $(i+1)$-елементен генератор, само ако първите $(i-1)$ елемента на двете множества съвпадат. Тоест вземат се всички елементи на един i-елементен генератор и последния елемент на друг i-елементен генератор, за който останалите елементи съвпадат с тези на другия. След като са намерени всички кандидати от тях се премахват тези, които имат i-елементно подмножество, което не е i-елементен генератор (Заради свойството, че едно множество от елементи, не е често, ако има подмножество от елементи, което не е често.) От останалите кандидати се премахват тези, които не са чести. Накрая се премахват тези, които които се срещат толкова на брой пъти колкото и някое тяхно i-елементно подмножество, защото търсим затворени чести елементи.
Иначе ако се налага се пресмята затварянето на предпоследните две нива генератори, които не са затворени.

4.1.3 Изход на алгоритъма

Изхода на алгоритъма е файл, в който всеки ред представлява затворено често множество от елементи, последвано от броя на срещанията в базата данни.

4.2 Модификацията Apriori_TIDClose

Използва се кеширано вертикално представяне на базата данни. Тоест вместо да описваме всяка транзакция с елементите ѝ. Описваме всеки елемент с транзакциите, в които се включва. Като ползата от това е, че ако знаем транзакциите, в които участват две множества X и Y, то за да намерим транзакциите, в които участва $X \cup Y$ трябва да намерим сечението на транзакциите за X и транзакците за Y. Или $tid(X \cup Y) = tid(X) \cap tid(Y)$. Тоест необходим е само един прочит на базата данни.

5 Описание на четенето на входните данни

Класа client.read.Reader се грижи да изчете входния файл ред по ред да раздели стойностите по петте колони на входния файл. За всеки ред по отделно се връща нова инстанция на класа client.read.EventData, като подава извлечените данни на статичния метод fromData на client.read.EventData, който се грижи от данните да бъдат запазени само минимум информация, която да е достатъчно за извличане на всяка една характеристика.

5.1 Класа client.read.EventData

Член данните на класа client.read.EventData са :

- private final client.read.date_time.DateTime dateTime
- private final client.read.event_context.EventContext eventContext
- private final client.read.event.Event event
- private final String origin
- private final String ipAddress

и имплемнтира интерфейсите:

- client.read.date_time.DateTimeValues
- client.read.event_context.EventContextValue

- client.read.event.EventValues

Като имплементацията на методите на client.read.date_time.DateTimeValues делегира с извиквания до съответните методи на член данната си dateTime. Имплементацията на методите на client.read.event_context.EventContextValue делегира с извиквания до съответните методи на член данната си eventContext. Имплементацията на методите на client.read.event.EventValue делегира с извиквания до съответните методи на член данната си event.

5.2 Класа client.read.date_time.DateTime

Класа client.read.date_time.DateTime се грижи са разчитането на времето на събитие от входния файл, като пази единсвено обект от тип java.time.LocalDateTime, който използва за имплементацията на client.read.date_time.DateTimeValues.

5.3 Йерархията на класа client.read.event_context.EventContext

Класа client.read.event_context.EventContext е абстрактен, имплемнтира client.read.event_context.EventContextValue и се грижи за създаването и връщането на подходящ свой наследник при обработката на стойността от колоната Event context от входния файл.

5.3.1 Наследника client.read.event_context.OtherEventContext

Наследникът client.read.event_context.OtherEventContext е финален клас, който връща следните стойности на характеристиките извличани от Event context колоната.

- Event context: Other
- Event context kind: Other

5.3.2 Подйерархията client.read.event_context.EventContextWithResourceName

Класът client.read.event_context.EventContextWithResourceName е абстрактен, но финализира имплементацията client.read.event_context.EventContextValue. Като приема параметри, чрез конструктура си EventContextWithResourceName(String kind, String name). За Event context генерира символен низ от kind и name, който служи за възтановяване на стойността от колоната Event context на входния файл. За Event context kind връша стойността член данната си kind, а за Resource name връша стойността член данната си name.

Преки и финални наследни са:

- client.read.event_context.CourseEventContext
- client.read.event_context.FileEventContext
- client.read.event_context.ForumEventContext
- client.read.event_context.PageEventContext

Като всеки от тях пази статичен константен низ public static final String kindValue = "Forum"Приема String name като аргумент на конструктура си и вика super(kindValue, forumName); в конструктура си.

5.4 Йерархията на класа client.read.event.Event

В рамките на йерархията на класа client.read.event.Event се случва прочитането и запазването на минималното количество информация, чрез която може да се възтанови информацията от колоните Component, Event name и Description, както и да бъдат извлечени стойности на характеристиките от съответните колони.
Всеки абстрактен клас в тази йерархия, включително класа client.read.event.Event имат статичен публичен метод:
public static Event fromData(String component, String eventName, String description) throws DescriptionMatchFailException, който се грижи за създаване на правилния наследник и валидиране на стойността от колоната Description от съответната подйерархия. Създаването става като се сравнят стойностите на component и/или eventName със съответните на финалните наследници или стойност в корена на подйерархията, най-често това се отнася до component.
Всеки финален наследни използва обект от тип java.util.regex.Pattern, създаден чрез подходящ регулярен израз описваш формата на стойността на колоната Description, след което ако израза съвпадне, тоест е валиден се извличат съответната информация от него, предимно стойности на идентификатори, която се запазва, за да могат да се извличат стойности на характеристики или да се възтанови стойността в полето Description например.
Проекта съдържа unit тестове, които покриват кода от йерархията на класа client.read.event.Event.

6 Описание на обработката на входните данни

6.1 Йерархията на шаблония клас client.consume.extract.Extractor

Шаблония клас client.consume.extract.Extractor има само преки финализиращи наследници, всеки от който служи за връщане на стойност на някоя от характеристиките. Той е шаблонен, защото стойностите са или Integer или String. Извличането на стойност става чрез имплементация на абстрактния метод:
public abstract V extract(EventData eventData). Всеки наследник претоварва този метод чрез връщане на резултат от викане на метод на аргумента eventData.

6.2 Филтриране на данните

Филтрирането на данните, ограничаването на редовете от входния файл се реализира от класа client.consume.filter.FilterReader, който наследява класа client.read.Reader и имплементира интерфейса Closeable.
Класа пази константна референция към обект от тип client.consume.Condition и чрез него ограничава резултати от викането на метода read на своя родител само до тези, които изпълняват условието на обекта condition. Тоест condition.isTrue(eventData) за резултата от super.read() е истина.

6.3 Йерархията на класа client.consume.filter.condition.Condition

Йерархията е следната:

- **True** - Връща винаги истина. Използва се за да не се приложи реална филтрация на входа.
- **Not** - Приема обект от тип Condition и връща негацията на резултата от isTrue.
- **And** - Приема два обекта от тип Condition и връша конюнкцията от резултата на викането на isTrue за двата.
- **Or** - Приема два обекта от тип Condition и връша дизюнкцията от резултата на викането на isTrue за двата.
- **ValueCondition** - Шаблонен абстрактен клас, приемаш обект от тип client.consume.extract.Extractor<V>.

- **HasValue** - Връща истина, ако резултата от extractor.extract не е null.
- **IsNull** - Връща истина, ако резултата от extractor.extract е null.
- **InRange** - Връща истина, ако числовата стойност е в даден затворен интервал от цели числа. Използва двойчно търсене.
- **ValueInList** - проверява дали стойност е в масив. Има член данна от тип java.util.TreeSet<V> за това проверката дали дадена стойност е в подадения като аргумент масив е логаритмична.
- **Like** Сравнява дали извлечената стойност удовлетворява даден регулярен израз. Конструктура приема инстанция на client.consume.extract.Extractor<String>, но може да се създава инстанция за client.consume.extract.Extractor<Integer> чрез шаблония метод fromIntegerExtractor. Обекта от тип client.consume.extract.Extractor<Integer> бива обвързан в инстанция на вътрешен скрит клас, който вика .toString() на резултата от extract. Вътрешния клас наследява client.consume.extract.Extractor<String>.
- **Relation** - Шаблонен абстрактен клас, който служи за имплемнтация на релация над даден тип.
  * **EqualToValue** - Сравнява да ли извлечената стойност съвпада с подадената, използва equals сравнение.
  * **NotEqualToValue** - Сравнява да ли извлечената стойност е различна от подадената, използва негацията на equals сравнение.
  * **GreaterThanOrEqualToValue** - Сравнява дали извлечената стойност е по-голяма или равна на подадената. Типа V трябва да наследява Comparable<V>.
  * **GreaterThanValue** - Сравнява дали извлечената стойност е по-голяма на подадената. Типа V трябва да наследява Comparable<V>.
  * **LessThanOrEqualToValue** - Сравнява дали извлечената стойност е по-малка или равна на подадената. Типа V трябва да наследява Comparable<V>.
  * **LessThanValue** - Сравнява дали извлечената стойност е по-малка на подадената. Типа V трябва да наследява Comparable<V>.

Проекта съдържа unit тестове, които покриват кода от йерархията на класа client.consume.filter.condition.Condition.

6.4 Хеширане на извлечените данни

7 Описание на връзката клиент-сървър

7.1 Описание на взаимодействието клиент-сървър

Сървъра слуша за свързвания. Всяко свързване се обработва в отделна нишка.
Клиента се свързва със сървъра. Изпраща му стойността на параметъра minSupp.
Сървъра прочита стойността на параметъра minSupp.
След което лиента прочита, филтрира, изивлича избраните характеристики, хешира ги и ги праща до сървъра. Като приключи с четенето затваря изходния канал на свързания сокет, което белижи край изпращането на данни от клиента към сървъра.
Сървъра прочита всеки един получен ред, който представлява ред от транзакционна база данни и го запазва във временен файл. Като свърши с четенето затваря своя входен поток.
Прочита и кешира в паметта цялата транзакционна база данни, представена вертикално и изпълнява алгоритъма Apriori_TIDClose за да намери чести затворени елементи и ги записва в същия временен файл.
След, което прочита времения файл и праща неговото съдържание до клиента и затваря своя изходен поток.
Клиента прочита отговора от сървъра, като обръща числата, който образуват чести затворените елементи обратно в стойности на характеристики чрез разхеширане и запазва резултата в изходния файл указан при стартиране на клиента.

7.2 Описание на кода на клиента

Файла client/Client.java е свързващия файл за клиента. Той свързва логиката по четене, обработка, изпращане на данните и приемане на отговора на сървъра. Класът client.Client има следната структура:

- private static <V> void sendValue(BufferedWriter writer, V value) throws IOException
- private static void sendChar(BufferedWriter writer, char c) throws IOException

- private static void sendMappedValues(BufferedWriter writer, int[] mappedValues) throws IOException
- private static void writeHeader(BufferedWriter writer, Column[] columns) throws IOException, IllegalArgumentException
- private final Socket socket
- private int counter
- private void readAndSendData(ValuesMapper reader, BufferedWriter writer)
- private void storeData(BufferedWriter writer, Object[] objects, int support) throws IOException
- private void storeLine(BufferedWriter writer, String line, ReverseValuesMap reverseMap) throws IOException
- private void receiveData(BufferedReader reader, BufferedWriter writer, ReverseValuesMap reverseMap) throws IOException
- public Client() throws UnknownHostException, IOException
- public ReverseValuesMap sendData(String filePath, double minSupportPercentage, Condition condition, Column[] columns)
  throws IOException, FileNotFoundException, IllegalArgumentException, DateTimeParseException, EventContextParseException, DescriptionMatchFailException
- public void receiveResult(String outputFile, ReverseValuesMap reverseMap) throws IOException
- @Override public void close() throws IOException

Член данните на класа са final java.net.Socket socket и int counter. Като чрез socket се установява връзка със сървъра, през него се прещат данните и приема резултата. counter се използва за да може да се пресметна често на среща на често затворените елементи спрямо броя на филтрираните данни, не целите.
В конструктура на класа се създава нов обект от тип java.net.Socket като се подават хоста на сървъра и порта, на който той слуша и брояча counter се инициализира с 0.
В предадената версия хоста е 127.0.0.1, който е локалния IP адрес на машината, но в реален случай би бил реален хост на сървъра. Понеже се очаква, че за да може да се свърже клиента със сървъра то той знае

кой е сървъра, знае хоста и порта. Разбира се разликата хоста и порта да бъдат приемани като входни параметри от клиентското приложение е минимална и за това те са хард коднати за улеснение при изпълнение на клиентския код.
client.Client имплементира интерфейса Closeable като имплементацията на метода close делегира извикване на close на socket.

7.2.1 Прочитане, хеширане и изпращане на входните данни до сървъра

Метода sendData приема път до входния файл, стойността на параметъра minSupp на Apriori_TIDClose, филтър, по който да се филтрират входните данни и масив от характеристиките, които да бъдат извлечени. В него се създава инстанция на ValuesMapper, която да хешира стойностите на характеристиките, в конструктура ѝ се създават и попълват хеш картите, чрез първо прочитане на файла. Инстанцията се използва за да се чете ред от входния файл, от който да бъдат извлечени и хеширани стойностите на желаните характеристики.
Създава се и инстанция на BufferedWriter, на която се подава инстанция на OutputStreamWriter, на която се подава изходния поток на сокета, чрез извикването на socket.getOutputStream()).
До сървъра се изпраща стойността на параметъра minSupp. Добавя се нов ред и след, което се изпращат хешираните входни данни до сървъра чрез извикване на скрития метод readAndSend на client.Clinet.
Затварят се инстанцията на ValuesMapper и изходния канал на сокета.
Като резултат се връща инстанция на класа ReverseValuesMap, която да бъде подадена на receiveResult за да могат да се разхешират стойностите на характеристиките.
sendValue е шаблонен, скрит статичен метод на класа, чрез който може да се изпрати стойност от произволен тип до сървъра.
sendChar вътрешно вика sendValue за да изпрати символ до сървъра.
В метода readAndSendData итеративно се прочита входния файл ред по ред и се хешира резултата от извлечените характеристики, чрез викането на метода readMapped на подадената инстанция от тип ValuesMapper. Увеличава се брояча, който брой събитията от файла. Чрез викането на метода sendMappedValues хешираните стойности се пращат до сървъра.
Метода sendMappedValues е скрит статичен метод, който приема референция към BufferedWriter, който е свързан с изходния поток на сокета и масив от цели числа, който представлява хеширани, сортирани, без повторения стойностите на желаните характеристики. Само тези, които имат стойност за съответното събитие. Масива реално представлява

една транзакция, от елементи, която се праща до сървъра.

7.2.2 Приемане, прочитане, разхеширане и запазване на резултата от сървъра

Метода, чрез който се приемат, прочитат, разхешират и запазват данните от резултата от сървъра е receiveResult. В него се създават инстанция на BufferedReader, като се подава инстанция от InputStreamReader, на която се подава входния поток на сокета, чрез извикването на socket.getInputStream(). Създава се инстанция и на BufferedWriter, на която се подава инстанция на FileWriter, на която се подава пътя до изходния файл.
Първо се записва заглавния ред с колони в изходния файл, след което се приемат, обработват и записват данните, резултат от сървъра. Накрая се затварят инстанцията на BufferedWriter и входния поток на сокета.
Загалвния ред, който се записва като първи в изходния файл представлява списък от имената на колоните - желаните характериситики, заедно с още две колни - Support, която има стойност стойността от изпълнението на алгоритъма - броя на срещания на множеството от елементи и Support %, която е процента на срещане на множеството елементи спрямо броя на филтрираните събития, всяко от което представлява една транзакция от данни, описващи събитие на потребител. Записването става чрез викането на вътрешния статичен метод writeHeader.
Метода receiveData се използва за приемането и обработването на получените чести затворени елементи, резултат на алгоритъма. В него се прочита ред по ред входния поток на сокета, след което за прочетения ред се вика метода storeLine, който да обработи и запише приетото затворено множество от чести елементи. Накрая се вика метода flush на BufferedWriter, асоциран с изходния файл за да е сигурно записването на на обаботените резултати, ако се пазаят в буфера на инстанцията.
Метода storeLine отделя от получения ред, честите елементи и тяхната подкрепа (support) - брой срещания, разхешира честите елементи за да получи масив от стойност на характеристики и вика storeData за да запази данните в изходния файл.
Метода storeData записва стойностите на характеристиките с разделители в изходния файл. След това към тях добавя разделени броя на срещанията на получения масив от стойности на характеристики и процента на срещания, който бива изчислен по формулата: $\frac{support}{counter}$. Накрая добавя нов ред в изходния файл.

7.3 Описание на кода на сървъра

Сървъра е реализиран в отделен пакет в отделен проект. Проекта на сървърва включва кода на сървъра и част от кода на библиотеката SPMF open-source data mining library. Часта е само тази, която е необходима за работата на класа Apriori_TIDClose.
Спазени са всички лицензиони права на библиотеката, кода на сървъра е с лиценз GPL-3 и заедно с кода на клиента и документацията ще бъдат публикувани и публично достъпни след приключване на последната защита на студент от курса.
Сървърния пакет (проект) цели да реализира алгоритъм като услуга, като в случая това е Apriori_TIDClose.
Сървъра е реализиран като независима единица от клиента, той просто приема входни данни за алгоритъма - числото minSupp и числа описващи всяка транзакция. Всеки ред е отделна транзакция, в която числата са цели, сортирани, без повторения и разделени от един символ за шпация.
Кодът на сървъра включва три файла:

- server/Main.java
- server/Server.java
- server/ConnectionHandler.java

7.3.1 server/Main.java

Класът server.Main реализира само един метод и това е main метода: public static void main(String[] args). В него се създава инстанция на класа server.Server и се стартират нишки изпълняващи метода run на server.ConnectionHandler, при приемането на нова връзка.

7.3.2 server/Server.java

Класът server.Server има единствена член данна: private final ServerSocket serverSocket. В конструктура му се създава нова инстанция на java.net.ServerSocket, на която се подава единствен аргумент - порта, на който да слуша сървъра.
Понеже класът държи референция към ServerSocket, който имплемтира интерфейса Closeable, то и класа server.Server го имплемнтира като в него изпълнява serverSocket.close();
Класът има още един метод: public ConnectionHandler acceptConnection(int id) throws IOException, в който се изпълнява единствено:

return new ConnectionHandler(serverSocket.accept(), id), с което се създава нова инстанция на server.ConnectionHandler, като понеже се вика serverSocket.accept(), то нишката (main нишката), която изпълнява този код бива блокирана докато не се получи нова връзка.

7.3.3 server/ConnectionHandler.java

Класът има следната структура:

- private final Socket socket
- private final File tempFile
- private final int id
- private double readRequest() throws IOException
- private void runAlgorithm(double minSupport) throws IOException
- private void sendResponce() throws IOException
- public ConnectionHandler(Socket socket, int id) throws IllegalArgumentException, IOException
- @Override public void run()

Член данната socket е референция към сокета създаден чрез викането на accept на ServerSocket. tempFile е референция към временнен файл, който се ползва за записване на получените данни и за резултата на алгоритъма. id пък е номера на нишката, с цел уникалност на създадения временен файл и извеждане на съобщения при: стартиране, приключване и възникване на грешка на нишката.
В конструктура се инициализират член данните и се извежда съобщение, че нишката започва работа.
Метода run прави последователни извиквания на:

- final double minSupport = readRequest();
- runAlgorithm(minSupport);
- sendResponce();
- socket.close();

Метода readRequest прочита входния параметър *minSupp*, който е резултата от извикването на метода. След което чете по един ред от входния поток, който записва във времения файл, докато клиента не затвори своя изходен поток (тоест не приключи с изпращането на данните си).
Метода runAlgorithm приема входния параметър *minSupp*, прочита от временния файл транзакционната база данни и изпълнява алгоритъма Apriori_TIDClose, като записва резултата в tempFile.
Метода sendResponce чете ред по ред резултата от алгоритъма в tempFile и го праща по изходния поток на socket, след което затваря изходния поток на socket.

8 Няколко примера за резултати от работа на приложението и коментари към намерените зависимости

9 Ръководство за инсталация

Сваля се кода на проекта. Кода е разделен на два отделни maven проекта, които могат да бъдат импортирани в повечето Java IDE-та. От там съответния потребител трябва да стартира първо main метода на класа server.Main, след което main метода на класа client.Main, като трябва да си конфигурира подаването на командни аргументи от IDE-то.
Втори начин е на компилира server.Main и client.Main и да изпълни main методите им чрез стартирането на java виртуална машина с тях.