סגור

89-211-01-02

תכנות מתקדם 2, מרצה: ד"ר אליהו חלסצ'י, מתרגל: מר רועי יהושע

תכנות מתקדם 2: 89-211 – מועד א' תשע"ז

זמן המבחן: **שעתיים וחצי**, יש לענות על 4 מתוך 4 שאלות, **בגוף השאלון בלבד**. חומר **סגור**.

שתי שאלות בקיאות + שתי שאלות עיצוב קוד ותכנות (java).

בקיאות

שאלה1 (24 נק'): מנגנון לסיווג (קלסיפיקציה) של טקסטים פועל באופן הבא. בשלב הלמידה אנו סורקים ה-מ-ו-ן טקסטים שסווגו מראש, ובודקים מהן המילים הנפוצות ביותר השייכות לסיווג ה i ושאינן נפוצות בשאר הסיווגים. כך לכל סיווג משויכת רשימה של מילים שמאפיינת דווקא אותו. למשל כל המילים הנפוצות ביותר שגברים כותבים אך נשים כמעט ולא משתמשות בהן. רשימה כזו נקראת features.

בהינתן טקסט חדש, אנו עוברים על רשימת המילים של כל סיווג ומודדים את שכיחותן של מילים אלה בטקסט החדש. נסווג את הטקסט החדש על פי רשימת המילים שזכתה לשכיחות הגבוהה ביותר.

מידי פעם נרצה לרענן את תהליך הלמידה ולעדכן את features שלמדנו.

לצורך שמירת המילים הנפוצות ביותר עבור כל סיווג, הגדרנו מחלקה בשם Classification המכילה רשימה של מילים. לאחר שלב הלמידה יש לאכסן את המופעים של Classification בדיסק. עבור כל אחת מהשיטות הבאות נמקו בקצרה את היתרונות והחסרונות של כל שיטה (3 נק' ליתרונות, 3 נק' לחסרונות).

א. פשוט נשמור אובייקט מסוג List<Classification> בדיסק (זה הרי serializable ולכן זה אפשרי)

יתרונות:

כל האובייקטים יישמרו קרוב אחד לשני בדיסק ולכן כשנצטרך אליהם מעיקרון הוקאלות ב- cache נחסוך בגישה לדיסק.

חסרונות:

אם נכתב בקובץ אחד וקרוב בדגות הקלות נשכחל את התשובה שלנו נצטרך לבדוק אותה לאחר מהתחלה!

ב. כל אובייקט Classification יישמר בקובץ אחר

יתרונות:

אם קובץ מסוים נמחק אובדן רק אובייקט אחד, ואין צורך לשחזר מגיבוי את כל האובייקטים.

חסרונות:

כל אובייקט יישמר בקובץ אחר אז כשנרצה לעדכן על רשימת המילים של כל סיווג נצטרך לגשת הרבה פעמים לדיסק! (נניח כשנרצה לרענן)

מס' מח': 110

שנת: תשע"ז סמסטר: 2 מועד: 1 מטלה: 1.1

קורס: 89211-02 תכנות מתקדם 2

0010002138820000015785 4

ג. נשתמש במסד נתונים. נמפה כל אובייקט Classification לשורה בטבלה. לטבלה יהיו N עמודות המשמשות לאכסון ה features של כל סיווג. שיטת schema on write.

יתרונות:

אם יהיה לנו מסד נתונים מאוד גדול ניתן להוסיף אותו לשרת חיצוני בלי בעיה.

חסרונות:

קשה לחיצוי, כל שאילתא לוקחת זמן רב כי צריך להתקדם למסד הנתונים כדי השאילתא צריכה להיבדק קשה SQL.

ד. נשתמש במסד נתונים בסגנון no SQL. שיטת schema on read.

יתרונות:

נקבל גמישות נתונים (יחסית) כי נתון מסד קשה schema on read.

חסרונות:

כאשר נרצה לעדכן את ה- features שנמדדו עדיה בקבוה, ובמקום אחר לבדוק בכלל שמלאים.

שאלה 2: (12 נק')

הקיפו בעיגול את התשובות הנכונות:

(א) פתרון memcacheD מספק סקלאבליות ליניארית.
(ב) סביבת Android Runtime חוסכת ב RAM ביחס לגרסאות קודמות.
(ג) Service Provider רושם את עצמו אצל ה broker באמצעות פרוטוקול SOAP
(ד) ל REST יש תמיכה בשליחת מידע רק בפורמט XML

שאלה 3 (31 נק'):

הביטו ב main הבא, המגדיר את המטרות שעליכם להשיג.

```
BlockingQueue<Point> result;
// define the stream
Stream<Point> s=new Stream<>();
result = s.filter(p->p.x>=0).filter(p->p.y<=0).getBuffer();
// the stream is still empty.

// printing thread
final boolean[] stop={false};
new Thread(()->{
      try {
            while(!stop[0])
                  while(!result.isEmpty())
                        System.out.println(result.take());
      } catch (InterruptedException e) {}
}).start();

// a demo of a slow stream-generation
Random r=new Random();
for(int i=0;i<500;i++){
      s.push(new Point(-100+r.nextInt(201),-100+r.nextInt(201)));
      Thread.sleep(50);
}
// stopping the stream(s)
s.endOfStream();

// stopping the printing thread
stop[0]=true;

// result: as the new points are generated,
//         only points with x>=0 & y<=0 are printed
```

ב main לעיל אנו מייצרים מופע של Stream<Point> המאפשר ארכיטקטורת pipes and filters ו fluent programming. באמצעות ביטוי למדה המתודה filter מאפשרת להעביר הלאה את כל הנקודות עם x לא שלילי, ומאלה להשאיר רק את הנקודות עם y לא חיובי. התוצאה תישמר ב result. אולם, בינתיים לכאורה לא קורה דבר, שכן ה stream ריק ממידע.

כעת אנו מגדירים ת'רד אנונימי שפשוט מדפיס את התוכן של result, ככל שיתקבלו לתוכו אובייקטים. הוא חי ברקע.

לאחר מכן אנו מייצרים 500 נקודות אקראיות עם ערכי x,y בין 100- ל 100, ומכנסים אותן ל stream. תוך כדי הכנסתן (ולא רק לאחר שמסתיים הקלט) הן יעברו סינון בהתאם להגדרות לעיל; "השורדים" יכנסו ל result, ויודפסו ע"י הת'רד שהגדרנו.

הפקודה endOfStream מורה על סיום הקלט הנכנס ל stream וכל משאב שצרכנו ישוחרר.

עליכם להשלים את הקוד של המחלקה Stream<T> כך שנוכל להפעיל את המתודות filter ו endOfStream בהצלחה ולקבל את התוצאה הרצויה להפעלה דומה לזו שב main לעיל.

```
public class Stream<T>{

    public interface Predicate<E>{
        boolean apply(E e);          // 5 points
    }

    BlockingQueue<T> buffer;
    volatile boolean stop;

    Thread activeThread;        // 2 points

    Stream<T> survivers;        // 2 points

    public Stream() {
        buffer=new LinkedBlockingQueue<T>();
        stop=false;
    }

    public void push(T t){buffer.add(t);}

    public Stream<T> filter(Predicate<T> p){   // total of 16 points
    survivers=new Stream<>();
    activeThread=new Thread(()->{
        try{
            while(!stop){
                T t = buffer.take();
                if (p.apply(t)){
                    survivers.push(t);
                }
            }
        }catch (InterruptedException e){}}
    }
    activeThread.start();
    return survivers;
    }
    public BlockingQueue<T> getBuffer(){return buffer;}
```

```
public void endOfStream() { // 6 points
    stop=true;
    activeThread.interrupt();
    if(survivers!=null){
        survivers.endOfStream();
    }
}
}
```

שאלה 4 (33 נק'):

ברצוננו ליצור תשתית מחלקות עבור מכירה פומבית. במכירה פומבית יש שני סוגים של שחקנים – המוכר Auctioneer, והקונה Bidder. כל הקונים מכירים את המוכר. המכירה מתחילה מאיזשהו מחיר התחלתי וכל קונה <u>במקביל</u> מעלה הצעות מחיר ע"פ מדיניות כלשהי משלו. המוכר מכריז לכל הקונים על המחיר העדכני, והם ממשיכים להציע מחירים חדשים. לאחר זמן מה המכירה מסתיימת והקונה שהציע את המחיר הגדול ביותר זוכה.

להלן דוגמת קוד להפעלת התשתית שברצוננו ליצור:

```
Auctioneer a=new Auctioneer();
Bidder b1=new Bidder(a, "b1", (x)->x+10);
Bidder b2=new Bidder(a, "b2", (x)->(x<=90 ? x+10 : 100));
Bidder b3=new Bidder(a, "b3", (x)->x+5);

a.startAuction(50);

Thread.sleep(50); // after some time

Bidder winner = a.endAuction();
System.out.println(winner.name+" "+ winner.currentBid); // b1
```

- יצרנו מופע של Auctioneer
- b1 הוא Bidder שהמדיניות שלו היא בהינתן המחיר x העלה את המחיר ל x+10
  - הגדרנו פונקציה מאד פשוטה, היא תעלה את המחיר ללא הגבלה, גם אם b1 הוא זה שקבע את המחיר הקודם...
- b2 נוהג באופן דומה עד לתקרה של 100, ואילו b3 תמיד יעלה ב 5.
- התחלנו את המכירה במחיר התחלתי של 50.
- לאחר זמן מה עצרנו את המכירה וקבלנו את ה Bidder שזכה.
- ע"פ המדיניות שהזרקנו b1 כמעט תמיד יוצא מנצח.

כדי לממש תשתית זו יש צורך בשתי תבניות עיצוב חשובות. עליכם להשלים את הקוד בטופס המבחן במקומות המתאימים בהתאם לתבניות העיצוב, הקוד והדרישות לעיל.

```
public class Auctioneer extends Observable { // 1 points

        private double currentPrice;
        private Bidder currentBidder;

        private static boolean stop; // 2 points (-2)

        // registers a bidder to an auction
        public void registerTheAuction(Bidder b){  // 2 points

                addObserver(b);
        }

        // receives a bidding request by some bidder

        public synchronized void acceptBid(Bidder b,double price){ // 8 points

                if(!stop){
                    if(price > currentPrice){
                        currentPrice = price;
                        currentBid = b;
                        notifyObservers(currentPrice);
                    }
                }
        }

        public void startAuction(double initialPrice){ // 2 points

                stop = false;
                currentPrice = initialPrice;
                notifyObservers(currentPrice);
        }

        public Bidder endAuction(){
                stop=true;
                return currentBidder;
        }
}// end of class Auction
```

```
public interface Policy{ // 3 points
        double getMyPrice(double currentPrice);
}

public class Bidder implement Observer { // 1 points
        Policy p; // 2 points
        double currentBid;
        Auctioneer auctioneer;
        String name;

        // 5 points
        public Bidder(Auctioneer a, String n, Policy policy) {
                p=policy;
                name=n;
                auctioneer=a;
                a.registerTheAuction(this);
        }

        @Override
        public void update(Observable arg0, Object arg1) { // total of 7 points
                double currentPrice=(double)arg1;
                currentBid=p.getMyPrice(currentPrice);
                auctioneer.acceptBid(this, currentBid);

                (-3) יתר

        }
}
```