

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: שעה וחצי, יש לענות על 4 מתוך 4 שאלות, בגוף השאלון בלבד. חומר סגור.

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

בקיאות

שאלה 1 (24 נק'): מנגנון לסיווג (קלסיפיקציה) של טקסטים פועל באופן הבא. בשלב הלמידה אנו סורקים ה-M-I-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:

- קוד ליניארי, פשוט
- קוד ליניארי, פשוט

חסרונות:

- לא יעיל, לא יעיל
- לא יעיל, לא יעיל

ב. כל אובייקט Classification יישמר בקובץ אחר

יתרונות:

- קוד ליניארי, פשוט
- קוד ליניארי, פשוט

חסרונות:

- לא יעיל, לא יעיל
- לא יעיל, לא יעיל

מס' מח': 103

מטלה: 1.1

שנת: תשע"ז סמסטר: 2 מועד: 1

קורס: 89211-02 תכנות מתקדם 2



00100021388200000112034

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

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

יתרונות:

- גודל הנתונים גדל עם הזמן (מאגר נתונים) - מאגר נתונים גדל עם הזמן
- אנון, יצר ויז' יצא, אף חזק, מאגר נתונים - מאגר נתונים גדל עם הזמן

חסרונות:

- מאגר נתונים, מאגר נתונים - מאגר נתונים גדל עם הזמן
- קוד רגיל - מאגר נתונים - מאגר נתונים גדל עם הזמן

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

יתרונות:

- גודל הנתונים גדל עם הזמן, מאגר נתונים - מאגר נתונים גדל עם הזמן
- מאגר נתונים - מאגר נתונים - מאגר נתונים גדל עם הזמן

חסרונות:

- קוד רגיל - מאגר נתונים - מאגר נתונים גדל עם הזמן
- מאגר נתונים - מאגר נתונים - מאגר נתונים גדל עם הזמן

שאלה 2: (12 נק')

הקיפו בעיגול את התשובות הנכונות:

- א. פתרון memcached מספק סקאלביליות 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

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

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 `fluent | pipes and filters 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`. תוצר 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.

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

```

public class Stream<T>{

    public interface Predicate<E>{
        public boolean run(E input)// 5 points
    }

    BlockingQueue<T> buffer;
    volatile boolean stop;

    Stream<T> filterStream; // 2 points
    Thread thread; // 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
        filterStream = new Stream<>();
        thread = new Thread(() -> {
            try { while (!stop) {
                while (true) { T streamObject = buffer.take(); // blocking
                if (p.run(streamObject)) {
                    filterStream.push(streamObject);
                }
            }
        }
        catch (Exception e) { // to something is interrupted }
        }
        return filterStream;
    }

    public BlockingQueue<T> getBuffer(){return buffer;}

```


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

public void endOfStream() { // 5 points

```

    Stop = true; // Stop the FilterThread
    BlockingQueue.removeAll(); // remove all elements
    if (FilterStream != null) // if not null
        FilterStream.closeStream(); // close the stream
}

```

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 במקביל 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.

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

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

```
private double currentPrice;
private Bidder currentBidder;
```

```
private volatile boolean stop; // 2 points
```

```
// registers a bidder to an auction
```

```
public void registerTheAuction( Bidder b ) { // 2 points
```

```

    observersList.add(b);
}

```

```
// receives a bidding request by some bidder
```

```
public 5/20/201 void acceptBid(Bidder b, double price){ // 8 points
```

```

if ( Price > currentPrice ) && (!sell) {
    currentPrice = Price;
    currentBidder = b;
    notifyObservers(currentPrice);
}

```

}

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

Current Price = initial Price ;

Verify Observers (in current prices): $\frac{1}{1000000} \times 1000000 = 1$

}

```
public Bidder endAuction(){
    stop=true;
    return currentBidder;
}
```

1

```

} // end of class Auction

```

// ~~... of~~
 35 ~~...~~
 Update
 notify in
 1. What's IP
 (U)

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

```
public interface Policy { // 3 points
```

```
    public double run(double x);
}
```

```
public class Bidder implements Observer { // 1 points
```

```
    Policy bidPolicy; // 2 points
    double currentBid;
    Auctioneer auctioneer;
    String name;
```

```
// 5 points
```

```
public Bidder(Auctioneer a, String name, Policy bidPol) {
```

```
    this.auctioneer = a;
```

```
    this.name name = name; // this
```

```
    this.bidPolicy = bidPol;
```

```
    this.auctioneer registerToAuction(this);
```

```
    currentBid = 0;
}
```

```
@Override
```

```
public void update(String arg0, Object arg1) { // total of 7 points
```

```
    this currentBid = (double) arg1;
```

```
    this currentBid = bidPolicy.run(currentBid);
```

```
    auctioneer.acceptBid(this, currentBid);
```

(-3)

```
    }
}
```

