Data Structure

Assignment no.3

תאריך הגשה: **11/09/16, 23:59 p.m.**

את העבודה יש להגיש בזוגות במערכת ההגשות submission system.
על העבודה להיות מוגשות כקובץ pdf יחיד בלבד.

- הנכם נדרשים לנסח **תשובות ברורות**
- עליכם לנסח תחילה את האלגוריתם בפסאדו קוד
- לאחר מכן להסביר את נכונותו
- ולבסוף לנתח את זמן הריצה
- לקובץ שלכם יש לקרוא ב ass2 ושני ת.ז. לדוגמא: ass3_305871458_5487558987.pdf

שאלות לגבי העבודה ניתן לשאול בפורום של העבודה הנמצא באתר של הקורס.

נושאי העבודה :

- תכנון ומימוש מבנה הנתונים של hash table.
- תכנון מבנה נתונים לבעיה נתונה, מימוש המבנה ב-Java ובדיקתו.
- שאלה תאורטית על hash table

חלק א' – מימוש טבלאות גיבוב :

עלכם לממש טבלת גיבוב על פי שלושת המקרים שלמדנו בתרגול:

1. שיטת השרשור – chaining.
2. שיטת Linear probing.
3. שיטת Double hashing.

השיטות שעליכם לממש (כולן בזמן O(1) בממוצע):

1. **void insert(Object key ,Object data)**
   הפונקציה מכניסה את data לתוך הטבלה לפי המפתח key.
   ניתן להניח שלא מכניסים מפתח שכבר קיים בטבלה.
   אם הטבלה מלאה, לא מתבצעת הוספה.
2. **delete(Object key)**
   הפונקציה מוחקת את האובייקט עם המפתח key.
   (במידה ואין אובייקט כזה, הפונ' אינה עושה דבר.
3. **boolean isEmpty()**
   הפונקציה מחזירה האם הטבלה ריקה.

4. **Object find(Object key)**
הפונקציה מקבלת מפתח key ומחזירה את הערך המתאים מהטבלה.
במידה ולא קיים ערך המתאים למפתח key, הפונקציה תחזיר null.

סיפקנו לכם את המחלקה האבסטרקטית **HashTableBGU** אשר עליכם לרשת במחלקה **HTChaining** ואת המחלקה האבסטרקטית **OpenAddressingBGU** היורשת את HashTableBGU ועליכם לרשת במחלקות **HTLinearProbing**, **HTDoubleHashing** ולממש את השיטות האבסטקטיות.

כמו כן, סיפקנו לכם את הממשק **Function** אשר מייצג hash function ו – step function ואת המחלקות **Function1**,**Function2** כדוגמא לפונקציות שראינו בתרגול.

המחלקה **HashObject** מסופקת לכם גם היא. מחלקה זו מייצגת אובייקט המוכנס לטבלת הhash. במחלקה יש את השדות key ו-data המייצגים את המפתח שלפיו נכנס הdata לטבלה.

חישבו מהם השדות אותם עליכם להוסיף לשדות הקיימים במחלקת האב ועל האתחול השונה של השדות.

שימו לב! אין לשנות את המחלקות HashTableBGU, OpenAddressingBGU, HashObject ואת הממשק Function

לעזרתכם יצרנו את המחלקה Test אותה אתם יכולים להריץ ולראות את תוצאות הבדיקות המריצות את הדוגמא מהתרגול. רצוי להוסיף בדיקות נוספות על מנת לוודא את נכונות הקוד שלכם.

אין להשתמש במחלקות של JAVA <u>פרט למחלקה List</u> (והמחלקות היורשות אותה).

חלק ב' – מימוש פתרון לבעיה – צי רכבים:

עליכם לתכנן טיפוס נתונים שיבצע את הפעולות הנדרשות להלן בזמנים הנקובים. כמו כן עליכם ליישם את מבנה הנתונים שלכם בתוכנית JAVA ולהריץ על קלטים נתונים.

<u>הבעיה</u>: צי הרכב של חברה כולל לכל היותר Y מכוניות שמחולקות ל N+1 קבוצות לכל היותר. הקבוצה ה- i מכילה את כל המכוניות שעברו בדיוק i טיפולים.

הפעולות הדרושות הן:

1. Insert(x) - הכנס מכונית חדשה, שמספר הרישוי שלה הוא x, ושעדיין לא עברה אף טיפול, למבנה הנתונים.
2. Treat( ) - טפל באחת מהמכוניות מבין אלה שעברו את המספר הקטן ביותר של טיפולים (אם יש יותר ממכונית אחת כזאת, ניתן לבחור מכונית באופן שרירותי מקבוצה זאת).
3. Times(x) - שאילתא המחזירה את מספר הפעמים שמכונית שמספר הרישוי שלה הוא x טופלה.

יש לממש את מבני הנתונים במחלקה **GarageSim**, כך ששאילתא 2 תיענה בזמן O(1) במקרה הגרוע ביותר, ושאילתות 1, 3 בזמן ממוצע O(1).

<u>הערות</u>

- אין לשנות את כותרות הפונקציות המסופקות במחלקה GarageSim.
- ניתן להוסיף שדות ופונקציות עזר למחלקה GarageSim וכן מחלקות נוספות.
- לצורך התוכנית נניח כי N ידוע מראש.

- כמו כן ניתן להניח כי מספר מכוניות לא יעלה על 10000 (זהו לא קבוע, כלומר מעבר על כל המכוניות הוא לא O(1).
- עליכם לטפל גם במקרים הבאים:

  Treat() כשהמבנה ריק.

  Times(x) כשאין מכונית עם מספר רישוי x.

  Insert(x) כשכבר יש מכונית עם שמספר הרישוי שלה x (מדפיסים הודעה שהמכונית קיימת)

- **אין להשתמש במחלקות מ- java.util או להשתמש בקוד המצוי באינטרנט.**
- **אתם יכולים להשתמש באחד המבנים שמימשתם בסעיף א'.**

<u>דוגמה</u>

קלט	פלט אפשרי
Treat	no car to treat
Insert 512	car 512 is inserted
Insert 1	car 1 is inserted
Treat	car 1 is moved to treatment 1
Treat	car 512 is moved to treatment 1
Treat	car 1 is moved to treatment 2
Times 1	car 1 passed 2 treatments
Insert 14	car 14 is inserted
Times 14	car 14 passed 0 treatments
Times 30	there is no car 30
Treat	car 14 is moved to treatment 1
Treat	car 512 is moved to treatment 2
Treat	car 14 is moved to treatment 2
Treat	car 1 is moved to treatment 3
Insert 14	car 14 is already in

Times 14 car 14 passed 2 treatments

Times 1 car 1 passed 3 treatments

חלק ג' – סיכום התרגיל המעשי:

א. תארו בקצרה את השוני במימושים בין שלושת המחלקות היורשות את HashTableBGU שמימשתם בחלק א'.

ב. תארו בקצרה את המימוש שלכם לבעיה הנתונה בחלק ב' – במילים, אין צורך לכתוב את הקוד מחדש.
לדוגמא – "הגדרתי מבנה נתונים של מחסנית משורשרת, במחסנית שמרתי את שמות השחקנים והשופטים. בנוסף שמרתי בעץ AVL את תוצאות המשחקים."

ג. הסבירו את זמן הריצה של כל אחת מן הפעולות שמימשתם בחלק ב' – הסבירו כיצד אתם עומדים בזמן הריצה לפי המימוש שלכם. גם במקרה זה עליכם להסביר במילים ולא בחלקים מהקוד.
לדוגמא – "את תוצאות המשחקים שמרתי בעץ AVL לפי תאריך המשחק ולכן שליפה של משחק אחד היא ב$O(\log n)$.

ד. הריצו את על שלושת המימושים שלכם מחלק א' קלט רנדומלי לפי הפרמטרים הבאים (הכנסה וחיפוש בלבד):

גודל הטבלה	מספר הערכים הנכנסים	מספר הערכים לחיפוש	טווח הערכים
10	10	20	0-200
100	100	200	0-200
1,000	1,000	2000	0-2000
10,000	1,000	20,000	0-20,000
10,000	5,000	20,000	0-20,000
10,000	10,000	20,000	0-20,000

מלאו טבלה בדומה לזו (טבלה אחת לכל שיטה):

גודל הטבלה	מספר הערכים	מספר הערכים לחיפוש	טווח הערכים	זמן הריצה המקסימלי	ממוצע לפעולה בודדת
10	10	20	0-200		
100	100	200	0-200		
1,000	1,000	2000	0-2,000		
10,000	1,000	20,000	0-20,000		
10,000	5,000	20,000	0-20,000		
10,000	10,000	20,000	0-20,000		

עליכם לספור במהלך ריצת התוכנית את זמן הריצה של כל אחת מפעולות ההכנסה ופעולות החיפוש (ללא מחיקה). שמרו את הזמן המקסימלי וחשבו לאורך הדרך את זמן הממוצע.
במידה וקיים איבר כלשהו בזמן ההכנסה, הגרילו מספר אחר.
שימו לב כי זמן הריצה הינו מספר התאים אותם בדקנו עד אשר נכנס הערך לטבלה (כולל) או שהערך נמצא/לא נמצא בהתאם לפעולה.

חלק ד' – שאלה תאורטית:

להלן פסאודו קוד של אלגוריתם נאיבי לחיפוש מחרוזת p בתוך מחרוזת s.

חיפוש-תת-מחרוזת-נאיבי (מחרוזת s[1..n], מחרוזת p[1..m]):

1. לכל i בין 1 לבין n-m+1:
    a. לכל j מ-1 עד m:
        i. אם s[i+j-1]!=p[j]
            1. צא מהלולאה הפנימית (כלומר עבור ל-1)
    b. החזר "p היא תת מחרוזת של s שמתחילה באינדקס i"
2. החזר "p היא לא תת מחרוזת של s"

נסמן את האורך של s בn ואת אורך p ב-m. מתקיים m<n.

א. נתחו את זמן הריצה של האלגוריתם.

בסעיפים ב' וג' ננסה לייעל את האלגוריתם באמצעות גיבוב.

ב. עבור מפתחות שהנם תת-מחרוזות של s באורך m, נניח (לצורכי השאלה) שזמן ריצת חישוב פונקציות גיבוב הוא *O(m)*, מכיוון שצריך לקרוא את m התווים ולבצע *O(m)* פעולות אריתמטיות של *O(1)*. אולם אין אנו מניחים זאת אם הפונקציה יכולה להיעזר בקלט נוסף פרט לתת-המחרוזת.
נאמר שפונקצית גיבוב היא פונקצית גיבוב מתגלגלת, אם בהינתן אינדקס *i* ואורך תת המחרוזת *m* בלבד ניתן לחשב את פונקצית הגיבוב של תת המחרוזת s[i...i+m] בזמן *O(m)*, אולם אם הקלט הוא האינדקס *i*, אורך תת המחרוזת היא *m* ובנוסף, ידוע גם ערך הגיבוב של תת המחרוזת s[i-1,..,i+m-1] *(תת המחרוזת הקודמת שחושבה)* אזי ניתן לחשב את פונקציית הגיבוב של תת המחרוזת s[i..i+m] בזמן *O(1)*.

הראו דוגמא לפונקציית גיבוב מתגלגלת שאינה מנוונת .כלומר, פונקציה שערך ההחזרה שלה תלוי בכל *m* התווים בתת המחרוזת.
פונקציה מנוונת לדוגמא: h(s[i...i+m])=ASCIIVAL(s[i+m]) כאשר *ASCIIVAL* הוא הערך ה*ASCII* של התו (ראו https://en.wikipedia.org/wiki/ASCII).

הפונקציה לא צריכה להיות אחידה.
הסבירו מדוע היא מתגלגלת.

ג. השלימו את הפסאודו קוד, כדי לקבל אלגוריתם שזמן הריצה הממוצע שלו O(n). נתון ש-h היא פונקציית גיבוב אחידה שמגבבת מחרוזת באורך m בזמן O(m) ושהיא גם פונקצית גיבוב מתגלגלת.

חיפוש-תת-מחרוזת-חכם (מחרוזת s[1..n], מחרוזת p[1..m]):

1. hs := h(s[1..m]) ,hp:=h(p[1..m])
2. לכל i בין 1 לבין n-m+1:

    ...השלימו את החסר (נחוצות שורות פסאדו קוד אחרות, כולן בתוך הלולאה)...

3. החזר "p היא לא תת מחרוזת של s".

חלק ה' – בונוס:

עליכם לכתוב חידה הקשורה לחומר הנלמד בקורס מבנה נתונים עד כה ולצרף את הפתרון לחידה הכולל הסבר מדוייק וברור, צרפו לכך נימוק מה אהבתם בחידה.
החידות יזכו אתכם בלכל היותר 5 נקודות בונוס ואינכם חייבים לענות על חלק זה.
החידות המקוריות והמעניינות ביותר יזכו ל5 נק' בונוס ויעלו לאתר הקורס (אנונימיות מובטחת).

הערות חשובות ודרישות הגשה:

1. חלקים ג'-ה' יוגשו בקובץ PDF אחד עם הכותרות "חלק ג' " , "חלק ד' ", "חלק ה' ".
2. ניתן להניח שהקלט בבדיקות יהיה תקין. לא תקבלו מתכנית הבדיקה שלנו ערך null או טקסט ריק כפרמטר לאף אחת מהפונקציות בממשק (אלא אם ביקשנו לטפל במקרי קצה).
3. בין הקבצים, תקבלו גם קובץ Test.java שישמש אתכם לבדיקה. לאחר שתסיימו את התכנית אתם יכולים להריץ את פונקציית ה- main כדי לדעת אם התכנית עובדת כמו שצריך. **הבדיקות אינן מכסות את כל המקרים ומומלץ להוסיף בדיקות משלכם**.
4. את העבודה יש להגיש ל Submission system.
5. עליכם להגיש קובץ zip בשם assignment3.zip המכיל בתוכו:
   a. תיקיית src ובה קבצי הג'אווה של העבודה,
      ללא הקבצים: Test.java, HashTableBGU.java, OpenAddressingBGU.java, HashObject.java, Function.java, Function1.java, Function2.java
   b. מסמך PDF כפי הנדרש, בשם Assignment3.pdf
6. סביבת העבודה בה תיבדקנה העבודות הינה JavaSE-1.7/8
7. עליכם לדאוג כי עבודותיכם יתקמפלו וירוצו בסביבת eclipse תחת גרסאות Java הנזכרות לעיל.
8. עבודות שלא יתקמפלו – יקבלו ציון 0.
9. **עבודותיכם יבדקו באמצעות כלי בדיקה אוטומטים הבודקים קורלציה בין עבודות. נא לא להעתיק! להזכירכם, המחלקה רואה בחומרה רבה העתקות.**
10. נרצה לראות קוד מתועד, מתוכנן היטב ויעיל שמייצג הבנה.

בהצלחה!