תרגיל 4 במבני נתונים

תאריך פרסום: 12.5.19

תאריך הגשה: 2.6.19

מתרגלים אחראים: אשר ותום

מבוא

בתרגיל זה אתם מתבקשים לכתוב תכנית לווידוא תקינות של סיסמאות.

שינוי סיסמא הינה פרוצדורה חשובה (אך מייגעת) שמטרתה להגן על המשתמש מפני תוכנות נוזקה (Malware) למיניהם.

התהליך עצמו של שינוי סיסמא הוא תהליך מורכב אשר לוקח בחשבון מרכיבים שונים כגון היסטוריה של סיסמאות קודמות לכל משתמש ושילוב מתאים של תווים מסוגים שונים בסיסמא.

בעבודה זו אנו נחליט אם לאשר או לדחות סיסמא חדשה באופן הבא:

יהיה לנו מסד נתונים (Database) אשר יכיל בתוכו סיסמאות לא תקינות. מדובר באוסף סיסמאות יחסית פשוטות לניחוש. אם משתמש ינסה לשנות סיסמא לאחת מהסיסמאות הללו לא נאפשר זאת בכלל, אך אם הוא ינסה לשנות לכל סיסמא אחרת נרצה לאפשר זאת בהסתברות גבוהה.

הדרך שבה נחליט אם לאשר או לפסול סיסמא תהיה באמצעות פילטר בלום (Bloom-Filter), שע"י שימוש באוסף של פונקציות גיבוב נחליט אם הסיסמא החדשה נמצאת במסד נתונים ועלינו לדחות אותה, או שזוהי סיסמא תקינה שעלינו לאשר.

- בשלב ראשון תתבקשו לקרוא שלושה קבצים:
  - קובץ בשם hash_functions.txt עם רשימה של פונקציות גיבוב לשימוש בפילטר בלום. כל פונקציה מתוארת באמצעות פרמטרים α וβ עליהם נפרט בהמשך.
  - קובץ בשם bad_passwords.txt אשר יכיל רשימה של הסיסמאות הרעות.
  - קובץ בשם requested_passwords.txt אשר יכיל רשימה של סיסמאות חדשות שהתוכנה מתבקשת לאשר או לדחות.

  יהיה עליכם ללמוד באופן עצמאי על קריאה/כתיבה מקובץ טקסט.

- בשלב השני יהיה עליכם להחליט לאשר או לדחות את הסיסמא החדשה שהמשתמש רוצה.

בהמשך מופיע הסבר מפורט על העבודה. אנחנו ממליצים לקרוא אותו מתחילתו ועד סופו לפחות פעם אחת לפני שאתם ניגשים לכתוב את העבודה.

לעבודה מצורפת תיקייה בשם Assignment_4. בתוך התיקייה תמצאו קבצי קלט ופלט לדוגמא. בנוסף תמצאו תיקיית src ובו מחלקה אחת בשם Runner עם פונקציית ה- main של העבודה שלכם.

פירוט המטלות

לנוחותכם המשימות חולקו למטלות (תתי-משימות).

שימו לב, עליכם לממש בעצמכם את כל המחלקות של המבני-נתונים המוגדרים בהמשך ואינכם רשאים להשתמש במחלקות מוכנות של Java או להיעזר בהן לצורך מימוש המבני נתונים.

מטלה ראשונה – קליטת הפונקציות גיבוב ועדכון הפילטר בלום

1. עברו על הקובץ hash_functions.txt ושמרו את הפונקציות במבנה נתונים כלשהו.
   כל הפונקציות שנשתמש בהם יהיו פונקציות הדומות לאלו שנלמדו בהרצאה מהסוג:
   $h_i = ((\alpha_i k + \beta_i) \bmod p) \bmod m_1$, כאשר עבור כל פונקציה i כך ש $i \in [1, \dots, N]$, נגדיר:
   - β_i וα_i הם מספרים שלמים ייחודיים לפונקציה i, אותם תקראו מהקובץ hash_functions.txt. (בשורה ה-i של הקובץ תמצאו שני מספרים מופרדים ע"י מקף תחתון ("_"), כאשר המספר השמאלי מייצג את α_i והימני את β_i של הפונקציה ה-i).
   - k הוא מפתח כלשהו שהפונקציה i מקבלת כפרמטר. ראו בהמשך איך להמיר סיסמא למפתח.
   - p הוא מספר ראשוני גדול. השתמשו ב p = 15486907.
   - m_1 הוא גודל המערך בפילטר בלום, אשר יינתן גם הוא בתור פרמטר לmain.
2. בשלב זה עליכם לקרוא את הסיסמאות הרעות מהקובץ bad_passwords.txt, ולעדכן בהתאם את המערך של הפילטר בלום.
   להלן התהליך הנדרש:
   א. הגדירו מחלקה בשם BloomFilter שמגדירה את הפילטר בלום של התכנית.
   ב. צרו מערך בינארי בגודל m_1 (שיתקבל כקלט לתכנית) אשר יהיה חלק מהפילטר בלום.

ג. עברו על הסיסמאות הרעות, והכניסו אותם למערך הנ"ל תוך שימוש בכל הפונקציות גיבוב שקיבלתם. עליכם להמיר את הסיסמא לערך מספרי שניתן להכניס לפונקציית גיבוב. לצורך מציאת הערך המספרי שמייצג סיסמא מסוימת, התייחסו אל סיסמא כאל מספר בבסיס 256 והמירו את המספר למפתח, תוך שימוש ב"כלל הורנר" (Horner's rule) אותו ראיתם בהרצאות. גם כאן השתמשו ב p = 15486907 בתור המספר הראשוני.

מטלה שניה – קליטת הסיסמאות החדשות הרצויות ומציאות אחוז השגיאה של הפילטר בלום

בשלב זה נרצה לאשר או לדחות כל סיסמא חדשה שהמשתמש ביקש, ע"י שימוש בפילטר בלום. כמו כן נרצה למצוא את אחוז השגיאה של הפילטר בלום שיצרתם (במקרה של פילטר בלום זהו אחוז הFalse-Positive) ע"י השוואה לתשובות שנקבל מטבלת גיבוב עם שרשור (שגם אותו עליכם ליצור בעצמכם).

להלן התהליך הנדרש:

1. הגדרה של טבלת גיבוב.
   הגדירו מחלקות בשם HashTable, HashList ו- HashListElement שמגדירות טבלת גיבוב עם שרשור, רשימה מקושרת וחוליה של רשימה מקושרת, בהתאמה.
   גודל טבלת הגיבוב m_2 יינתן כפרמטר של הפונקציה main (ראו בהמשך).
   במחלקה HashTable הגדירו מתודה בשם hashFunction שתשמש כפונקציית הגיבוב של הטבלה. אופן המימוש של פונקציית הגיבוב תלויה בכם. אתם יכולים להשתמש בפונקציה הפשוטה אותה ראיתם בתרגול או לחילופין בפונקציה מורכבת יותר.
2. עברו שוב על רשימת הסיסמאות הרעות. לכל סיסמא רעה, צרו חוליה לטבלת גיבוב שהמפתח שלה זה הייצוג המספרי של הסיסמא (המרה של סיסמא לייצוג מספרי מתבצע כמו לעייל).
3. כעת עליכם לקרוא את הסיסמאות הרצויות מהקובץ requested_passwords.txt לפי הסדר המופיע בקובץ. לכל סיסמא רצויה, בדקו אם היא מופיעה ברשימת הסיסמאות האסורות ע"י שימוש בפילטר בלום.
4. לכל סיסמא רצויה בדקו אם היא מופיעה בטבלת הגיבוב שיצרתם.
5. הדפיסו בשורה הראשונה של קובץ הפלט את אחוז השגיאה של הפילטר בלום. אחוז השגיאה יוגדר כמספר הסיסמאות שנדחו ע"י הפילטר בלום בטעות (זאת אומרת שהסיסמאות נדחו למרות שהם לא מופיעות בטבלת הגיבוב שמכיל את הסיסמאות הרעות, ולמרות זאת הפילטר איתר אותם בטעות כסיסמאות רעות), חלקי סך כל הסיסמאות המבוקשות הטובות שלא היו אמורות להידחות (כלומר כל הסיסמאות המבוקשות שלא נמצאות בטבלת הגיבוב).
6. בשורה השנייה של קובץ הפלט הדפיסו את כמות הסיסמאות הרעות מתוך קובץ הסיסמאות המבוקשות שהתכנית שלכם איתרה בעזרת הפילטר בלום (לא כולל איתורים באמצעות טבלת הגיבוב).

ניתן למצוא דוגמא של הפלטים הנדרשים בקובץ הפלט המצורף לעבודה. למרות שזאת רק דוגמא, וודאו שאתם מצליחים לקבל את אותו הפלט בדיוק עבור קבצי הקלט המצורפים לעבודה.

מטלה שלישית – הכנסת המילים הרעות לעץ חיפוש B

אתם מתבקשים כעת לעבור על רשימת המילים הרעות, ולהכניסם לעץ חיפוש B **לפי הסדר** המופיע בקובץ. מבנה העץ B שתיצרו יהיה דומה לעץ שלמדתם בכיתה.

להלן התהליך הנדרש:

1. הגדירו שתי מחלקות BTree ו- BTreeNode המגדירות עץ חיפוש B וקדקוד חיפוש B, בהתאמה, כרגע עם הפעולות של הוספה וחיפוש. בהמשך תדרשו לממש גם את פעולת המחיקה.
   הדרגה t של העץ תינתן כפרמטר של הפונקציה main (ראו בהמשך).

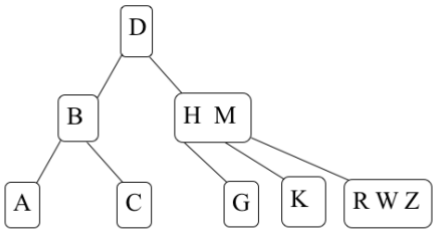
2. צרו אובייקט מסוג BTree, עברו שוב על הקובץ bad_passwords.txt והכניסו את הסיסמאות לאובייקט לBTree. סדר ההכנסה יתבצע לפי סדר השורות בקובץ.

כדי שנוכל לבדוק את העץ שיצרתם יהיה עליכם להדפיס אותו לקובץ.

להלן התהליך הנדרש:

1. הוסיפו למחלקות פעולת סריקה של in-order, היוצרת מחרוזת המורכבת מהמפתחות בעץ לפי הכללים הבאים:
   ד. הפרדה בין מפתחות שונים צריכה להיעשות באמצעות פסיק (",").
   ה. בהדפסת מפתח כלשהו, נדפיס עת המפתח והעומק של הקודקוד שמכיל את המפתח, כאשר נפריד בין המפתח לעומק ע"י מקף תחתון. למשל אם בשורש יש מפתח 'qwerty', אז במקום המתאים בפלט נרשום "...,qwerty_0,...".

דוגמה של הדפסת in-order: מחרוזת של העץ למטה (זהו העץ B אותו ראיתם בתרגול) צריכה להראות כך: "A_2,B_1,C_2,D_0,G_2,H_1,K_2,M_1,R_2,W_2,Z_2".



2. לאחר שסיימתם לבנות את העץ, הדפיסו את המחרוזת שלו בשורה השלישית לקובץ פלט של התכנית. בין הקבצים המצורפים לעבודה, תוכלו למצוא דוגמא לפלט תקין בקובץ הפלט המצורף לעבודה. ודאו שאתם מצליחים לקבל **בדיוק** את אותו הפלט.

הערות:

ו. DFS הינו אלגוריתם מפורסם שניתן לממש באמצעותו סריקת in-order, במידה ותירצו בכך. ניתן לקרוא עוד על DFS כאן.
ז. לצורך מימוש העץ B השתמשו במימוש שראיתם בכיתה, ועדכנו אותו היכן שצריך בהתאם לדרישות הנ"ל. כאשר תממשו את העץ, השתמשו בסדר לקסיקוגרפי של המפתחות, כאשר ערכים גדולים או שווים ילכו ימינה.
ח. לא תקבלו קובץ hash_functions.txt ריק (ובאופן כללי לא תקבלו קבצים ריקים), כך שאין צורך לדאוג למקרה קצה זה.

מטלה רביעית – השוואת זמני ריצה

כעת נרצה לעמוד על ההבדלים בין הזמן הדרוש לביצוע חיפוש בעץ B לבין הזמן של חיפוש בטבלת הגיבוב שיצרתם.

1. עברו שוב על המילים בקובץ requested_passwords.txt ובדקו האם הם מופיעות בעץ B שלכם. שמרו בצד את סך הזמן במילישניות שלקח לחפש את כל המילים. הזמן שתשמרו יהיה מספר כלשהו עם 4 ספרות אחרי הנקודה העשרונית. התעלמו מהספרות שמופיעות לאחר הספרה הרביעית.
2. בצעו שוב את אותו החיפוש של המילים, הפעם בעזרת טבלת הגיבוב.
3. בשורה הרביעית של קובץ הפלט, הדפיסו את שני הזמנים שמצאתם והפרידו בין הזמנים עם מקף תחתון ("_"). ההדפסה תתבצע משמאל לימין, כאשר קודם יופיע הזמן של חיפוש המילים באמצעות העץ. ראו דוגמא בקובץ פלט המצורף לעבודה.
4. הערה – אל תדאגו אם תראו זמני ריצה השונים מאלו המופיעים בפלט המצורף לעבודה. הגיוני שיהיו הבדלים בין מחשבים שונים ואפילו בין ריצות שונות על אותו מחשב. כל עוד מימשתם את יתר החלקים כנדרש, זה בסדר.

מטלה חמישית – מחיקה מעץ חיפוש B

1. עברו **לפי הסדר** על הקובץ delete_keys.txt המצורף לעבודה, ולכל שורה מחקו את המפתח המתאים מהעץ B.
2. בשורה החמישית בקובץ הפלט, הדפיסו את מבנה העץ B המתקבל לאחר כל המחיקות. סדר ההדפסות הוא בדומה להדפסת העץ שראיתם מקודם.

פונקציית ה- main

פונקציית המain שעליכם להריץ רשומה במחלקה **Runner**, שמצורפת לתיקייה שמצורפת לפרויקט. שימו לב, עליכם להשתמש במחלקה Runner **<u>ללא שינויים</u>** (ערעורים לא יתקבלו עבור קוד המתבסס על קובץ Runner אחר מזה שניתן לכם). וודאו שאתם מבינים את הקוד וההערות בmain, זה יעזור לכם כדי להבין מה מצופה ואיך להתחיל.

הפרמטרים של ה- main:

1. ערך m_1- גודל הטבלה של הפילטר בלום.
2. ערך m_2- גודל הטבלת הגיבוב.
3. ערך ה-t של העץ B.

הערות כלליות בהתייחס לכל המימוש.

חוסר הקפדה על הדגשים הבאים יגרור הורדת נקודות.

- **מימוש מבני הנתונים**
כמוזכר לעייל, את כל המבני נתונים הדרושים לעבודה אתם נדרשים לממש לבד (כלומר ללא שימוש במחלקות מוכנות של Java). היוצא מן הכלל היחיד הוא המערך המובנה של Java בו ניתן להשתמש (שכן הוא לא דורש שילוב של ספריות חיצוניות נוספות). הציון של עבודה עם שימוש במבני נתונים הממומשים ע"י ספריות יהיה אפס.
- **זמני ריצה**
הפעולות שהתכנית תבצע בעזרת מבני הנתונים שיצרם צריכים לרוץ בזמנים יעילים (בהתאם לפרמטרים שתקבלו כקלט לתכנית) כמו שלמדתם בכיתה. לא נשווה זמני ריצה ברמת המילישניות, אבל כן נצפה עמידה בזמנים סבירים.
- **קוד נקי (Clean code)**
ישנה חשיבות עילאית שהקוד שאתם מגישים יהיה מסודר ונקי. זה חשוב גם כי הבודק יצטרך לבדוק את הקוד שלכם ולהבין מה כתבתם, אבל בעיקר כי יום אחד תסיימו את הלימודים ותתחילו לעבוד בחברה כלשהי, ופחות או יותר באותו זמן מישהו אחר יצטרך (כנראה) לעבור על הקוד שכתבתם ולתקן באגים. אז בואו נחסוך מאותו בן-אנוש מסכן לגשת אל קוד עם מתודות של 150 שורות או יותר, ונתחיל כבר עכשיו לכתוב קוד קריא ונקי שמאפשר להבין את הפונקציונאליות של המתודות ללא יותר מדי מאמץ. במידה שתרצו פוינטר נחמד לכתיבת קוד נקי, אתם מוזמנים לצפות בסרטונים של הדוד בוב, וספציפית מומלץ לצפות בסרטון הזה בין דקה 52 לדקה 57 כדי להבין איך לנקות את הקוד.
TLDR, הקוד צריך להיות מסודר ונקי עם שמות של מתודות שמסבירות את הפונקציונאליות של כל מתודה. אז תכתבו קוד, תיראו שהקוד מבצע את מה שאתם רוצים, ואח"כ אל תשכחו לנקות אחריכם. זו יכולת חשובה שעדיף לפתח בתחילת הדרך, שכן זה נהיה קשה יותר ככל שצוברים הרגלים רעים. ספציפית לעבודה הזאת, 15 שורות למתודה זה דיי והותר, ולכן 15 שורות זה המקסימום המותר (כאשר פותח/סוגר מסולסל שמופיע לבדו בשורה ייחשב כחלק מהשורה הקודמת. הספירה מתחילה

לאחר הפותח המסולסל הראשון בכל מתודה. שורות של הערות לא ייספרו בתור שורות). נקודות יורדו לעבודות עם מתודות שעוברות את המקסימום.

- יש לוודא שהערכים שמועברים כפרמטר לבנאים, לשיטות ולפונקציות תקינים ואם לא יש לזרוק חריגת זמן ריצה (RuntimeException) עם הודעה מתאימה (אין צורך לדאוג לתקינות הקבצים עצמם).
- וודאו שאתם שומרים על Encapsulation.
- וודאו שאין לכם warnings בקוד. אם יש לכם warnings תטפלו בהם ואל תנסו להסתיר אותם בשיטות שונות.

בבדיקה הידנית נעבור על הקוד באקליפס, אז גם אם קידדתם באמצעות IDE אחר, וודאו שאין לכם אזהרות באקליפס (ייתכן שיהיו הבדלים בין סביבות עבודה שונות).

- הקוד צריך להתקמפל בסביבת Java 1.8.
- הקוד שלכם צריך להיות מסוגל לרוץ מה- command line (ראו הסבר בהמשך).
- וודאו שאין לכם אותיות בעברית או בשפה אחרת חוץ מאנגלית בקוד.
- וודאו שהקבצים שאתם מייצרים תקינים. בפרט וודאו שלפחות עבור קבציי-הקלט הנתונים על ידנו, אתם מצליחים לקבל קבצי פלט אשר **זהים** לקבציי-הפלט הנתונים על ידינו (מלבד בזמני הריצה).
- כמובן שאין להעתיק קוד.

<u>אופן ההגשה:</u>

- יש להגיש קובץ אחד בשם assignment4.zip. קובץ ה-zip שיצרתם יכיל רק את תיקיית ה- src, כאשר ה-src יכיל את כל קבצי התוכנית. לא אמורים להיות לכם תיקיות נוספות.
- כל המחלקות שתגישו צריכות להיות ב Default Package.
- אחרי שהעליתם קובץ, וודאו שהעליתם קובץ תקין שניתן להריץ אותו.

אתם יכולים להיעזר בקבצי הקלט והפלט המצורפים לעבודה, כדי לבדוק את העבודה שלכם.
למשל, אם שלפתם את תיקיית ה- src לתוך תיקייה x כלשהי, בצעו את הדברים הבאים:

- העתיקו את קבצי הקלט שצירפנו לתיקיית x הנ"ל.
- בתוך terminal, נווטו אל התיקייה x.
- הריצו את הפקודות הבאות:

```
javac -d . src/*.java
java Runner 32 32 2
```

- השוו את קובץ הפלט שקיבלתם עם קובץ הקלט המצורף לעבודה. השורות בקובץ הפלט שקיבלתם אמורות להיות **זהות** לאלו בקובץ הפלט שצירפנו לעבודה (מלבד בזמני הריצה).

בהצלחה!
