מבני נתונים – תרגיל 4

B-Trees & Hashing

תאריך פרסום: 14.5.2017 תאריך הגשה: 8.6.2017

מרצה ומתרגלים אחראים:
פרופ' שלומי דולב, מוחמד גנאים, מתן איל

הקדמה

אימות קבצים הינו תהליך של בדיקת שלמות של קבצים.
שלימותו של קובץ עלולה להיפגע כתוצאה מהשחתה שמקורה בכשלים של מערכות האחסון, שגיאות ברשתות התקשורת שדרכן עבר הקובץ, באגים בתוכנה וטעויות אנוש.
בנוסף, השחתת קבצים עשויה להיות חלק ממתקפה של גורמים עוינים - שבמסגרתה מושתל בקובץ קוד זדוני כגון וירוסים, תולעים וסוסים טרויאנים.

שיטה אחת לאימות קבצים היא השוואה בין שני עותקים של אותו קובץ ביט-אחר-ביט, מבין חסרונות השיטה הזו אפשר למנות את זמן החישוב הארוך והצורך לגישה אל שני הקבצים.

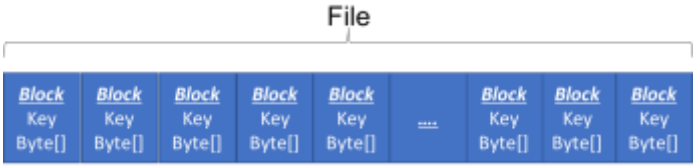
שיטה יותר נפוצה היא שימוש בפונקצית גיבוב קריפטוגרפית חד-כיוונית, שבה האימות נעשה ע"י השוואות הפלטים של פונקציית הגיבוב - "החתימות" - של שני העותקים, שהן הרבה יותר קטנות מהקבצים שרוצים להשוות. אם החתימות נמצאו שוות, אפשר לקבוע ששני הקבצים זהים.
אך כתוצאה אינהרנטית משימוש בפונקציות גיבוב – הקביעה שהקבצים זהים תהיה *בהסתברות גבוהה*, אך לא ודאית.

לדוגמא, ארגון Eclipse Foundation העומד מאחורי תוכנת ה-Eclipse מפיץ את קובץ ההתקנה דרך עשרות שרתים הפזורים ברחבי העולם. בעמוד ההורדה של Eclipse לצד הלינק להורדת קובץ ההתקנה שגודלו 300 מגה-בייט, מופיעה גם חתימה שגודלה אך ורק 64 בייט, שנוצרה כתוצאה מהפעלת פונקציית גיבוב קריפטוגרפית בשם SHA512 על הקובץ.

חתימה זו, מסופקת ע"י מפיצי הקובץ על מנת לאפשר למשתמשים לאמת את שלימותו ואמינותו של עותק קובץ ההתקנה שנמצא בידיהם לפני התקנת התוכנה. נציין שלפעמים מאפשרים להוריד את החתימה ממספר מקורות שונים (להוריד את הקובץ מספר פעמים ממקורות שונים זה הרבה יותר יקר) ובכך למנוע מגורמים עוינים להציג אתר זדוני עם קובץ נגוע וחתימה מתאימה לו.
על המשתמש החפץ באימות, להפעיל את פונקציית הגיבוב SHA512 על העותק שלו, ולהשוות את חתימתו של העותק לחתימתה שמופיעה באתר (שנוצרה ע"י הפעלת SHA512 על הקובץ המקורי).

בתרגיל זה נתמקד בקבצים שמאוחסנים ב B-Tree וביצירת חתימות עבורם.

בייט אחד (Byte) מורכב מרצף של שמונה ביטים (Bits).
קובץ הינו רצף ארוך של בייטים, וכל קובץ מחולק לבלוקים רבים. בלוק מורכב ממערך של בייטים ומפתח ייחודי.

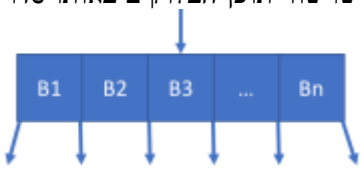


הבלוקים של הקובץ ישמרו בצמתי ה-B-Tree, מסודרים לפי המפתח של כל בלוק.
שמירת הבלוקים של קובץ ב B-Tree מאפשרת חיפוש בלוקים לפי מפתח, וכן עריכת שינויים בתוכן הקובץ ביעילות - ע"י מחיקת והכנסת בלוקים לעץ.
בנוסף, יצירת חתימה לקובץ המאוחסן ב B-Tree, תאפשר אימות של קבצים בשיטה שהוסברה לעיל.

חתימה ל B-Tree: *(בהשראת Merkle Trees)*
החתימה תסתמך על פונקציית הגיבוב SHA1 , המקבלת קלט מאורך שרירותי ומוציאה פלט מאורך קבוע -20 בייטים.

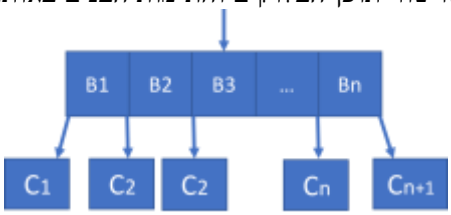
חתימה ל B-Tree הינה בעצמה עץ - נקרא לו ***Merkle-B-Tree*** או בקצרה MBT :
עבור כל צומת ב B-Tree קיים צומת מקביל אליו ב-MBT , כך שה-MBT מכיל אותו מספר של צמתים ואותה "צורה" של עץ B-Tree (אך מבנה הצמתים ותוכנם שונה לחלוטין).
צומת ב-MBT מכיל את החתימה של צומת B המקביל (באופן שמוגדר למטה), ומצביעים לצמתי החתימה המקבילים לבנים של צומת ה B.

- *חתימה של עלה B עם n מפתחות:*
היא פשוט הפלט של SHA1 על שרשור תוכן הבלוקים באותו סדר שבו הם מאוחסנים.



$$hash(LeafeNode) := SHA1(B_1 \cdot B_2 \cdot B_3 \cdot \dots \dots \dots \cdot B_n)$$

- *חתימה של צומת B עם n מפתחות:*
במקרה הזה החתימה נגזרת מתוכן הבלוקים וגם מחתימות הבנים.
היא הפלט של SHA1 על שרשור תוכן הבלוקים וחתימות הבנים באותו סדר שבו הם מאוחסנים.



$$hash(Node) = SHA1(hash(C_1) \cdot B_1 \cdot hash(C_2) \cdot B_2 \cdot hash(C_3) \cdot \dots \cdot B_n \cdot hash(C_{n+1}))$$

חלק א' – תיאורטי (25 נק')

שאלה 1 (5 נק')

נתון B-Tree עם דרגה מינימלית t, המכיל n צמתים. כל הצמתים, כולל השורש, מלאים.
בצמתי העץ מאוחסנים בלוקים של קובץ, כך שגודל כל בלוק הוא D.
חשבו את היחס (כביטוי של n,t,D) בין המקום הדרוש לאחסון ה B-Tree והמקום הדרוש לאחסון ה Merkle-B-Tree הנגזר ממנו.
הניחו שגודל מצביע לצומת זה בייט אחד, ושבעלים קיימים מצביעים אך הם מצביעים ל null.
ותזכרו שהפלט של SHA1 הוא 20 בייטים.

שאלה 2 (15 נק')

נתונים B-Tree ולצידו ה MBT שנגזר ממנו.
המטרה היא לשמור חתימה עדכנית עבור ה B-Tree בכל רגע נתון.
האם עדכון ה B-Tree ע"י הכנסת בלוק חדש או מחיקת בלוק קיים, מחייב חישוב מחדש לכל צמתי ה MBT?
אם כן - הוכיחו.
אם לא - נסחו מחדש את אלגוריתמי ההכנסה והמחיקה של B-Tree (ותוסיפו שדות לצמתי העצים במידת הצורך),
כך שיכללו עדכון שיטתי ויעיל ל MBT, המצריך חישוב מחדש רק לחלק מצומצם מצמתי ה MBT.
חשבו את סיבוכיות הזמן והמקום של האלגוריתמים שניסחתם מחדש.

שאלה 3 (5 נק')

עיינו בפסאודו קוד של SHA1, שהיא פונקציית גיבוב קריפטוגרפית חד-כיוונית.
הקוד מתחיל באתחול 5 משתנים:

```
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0
```

מה הסיבה שמתכנני SHA1 בחרו לאתחל את המשתנים לערכים קבועים ובריש גלי, ולא בערכים אקראיים המוגרלים מחדש בכל הפעלה של הפונקציה?

את הפתרון של החלק התאורטי יש להגיש מוקלד בקובץ PartA.pdf

חלק ב' - מעשי (75 נק')

ממשו B-Tree המיועד לאחסון בלוקים לפי מפתח, שיתמוך בשאילתות הבאות:

- אתחול עץ ריק.
- חיפוש בלוק לפי מפתח.
- מחיקת בלוק לפי מפתח.
- הכנסת בלוק חדש.
- יצירת חתימה לB-Tree בדמות Merkle-B-Tree.

בשונה מחלק א' של התרגיל, **בחלק זה לא צריך לשמור על חתימה עדכנית בכל רגע נתון**. ולכן מספיק לממש את ההכנסה והמחיקה כפי שנלמדו בכיתה.

הנחיות:

- שאילתות החיפוש, המחיקה והכנסה יש לממש ***בדיוק*** כפי שהוגדרו בהרצאה.
  שאילתת יצירת החתימה יש לבצע בסיבוכיות זמן ומקום לינארית בגודל העץ.

- לצורך מימוש התרגיל הוגדרו המחלקות הבאות:
  1. Block: בלוק של קובץ.
  2. BNode: צומת ב B-Tree, מממש ממשק BNodeInterface.
  3. BTree: העץ, מממש ממשק BTreeInterface.
  4. MerkleBNode: צומת בעץ Merkle-B-Tree.

- עברו על הקוד לפני תחילת הפתרון, הנחיות והבהרות נוספות מופיעות בהערות.

- למחלקות מסוימות, קיימים בנאים מרובים לכל מחלקה, הבינו את ההבדל בין הבנאים ואת המטרה של כל אחד.

- ב HashUtils תמצאו קוד מוכן לחישוב הפונקציה SHA1.

- אין להגיש ואין לערוך אף שינוי בקוד של הקבצים שמופיעה בראשם ההערה :

```
/// DO NOT CHANGE
/// DO NOT SUBMIT
```

- שני הקבצים היחידים המיועדים לעריכה ולהגשה הם BNode.java ו BTree.java ,
  אך גם בקבצים האלו אין לעשות שינויים בקוד שמופיע בין שתי ההערות :

```
// ////////////////////BEGIN DO NOT CHANGE ////////////////////
אין לשנות את הקוד שמופיע פה
// ////////////////////DO NOT CHANGE END////////////////////
```

- בבדיקות האוטומטיות נבדוק (בין היתר) את התנהגות השאילתות על פני כל המקרים, התתי-מקרים והתרחישים שראיתם בהרצאות.
  בקוד קיימים בנאים ייעודיים לבדיקות, אתם מוזמנים להשתמש בהם לבדיקת נכונות הקוד שלכם: צרו אובייקט עם ערכים כרצונכם, והפעילו עליו שאילתה והשוו את תוצאתה לאובייקט אחר שמייצג את התוצאה שאתם מצפים לה.
  במחלקה Block תמצאו מתודה ליצירת בלוקים עם תוכן אקראי.

- מצופה מכם לכתוב קוד יעיל, קריא, מסודר ומתועד.
  בתחתית מסמך זה תמצאו הצעת חלוקה למתודות עבור המחלקה BNode.
  כמו כן, למי שלא מכיר, כדאי לעבור על הדוקומנטציה של מתודות ה set, add ו subList של ArrayList.

- על הקוד להתקמפל ולרוץ בסביבת Java 1.7 ו Eclipse.
כל המחלקות שתגישו צריכות להיות ב Default Package.
למעט ArrayList, בקוד המיועד להגשה **אין** להשתמש במבני נתונים גניריים ובספריות נוספות הזמינות ב Java.
קוד שלא מתקמפל ו/או מסתמך על ספריות אחרות - יקבל ציון 0.

- יש להגיש את העבודה בקובץ אחד שיקרא assignment4.zip ויכיל:
  - תיקיית src ובה קבצי ה Java (רק אלו המיועדים להגשה).
  - קובץ PartA.pdf שבו הפתרון לחלק א'.

בהצלחה

הצעת חלוקה למתודות עבור המחלקה BNode:

זו רק הצעה, אין דרישה ללכת לפי חלוקה זו.

```
/**
* Splits the child node at childIndex into 2 nodes.
* @param childIndex
*/
public void splitChild(int childIndex);

/**
* True iff the child node at childIndx-1 exists and has more than t-1 blocks.
* @param childIndx
* @return
*/
private boolean childHasNonMinimalLeftSibling(int childIndx);

/**
* True iff the child node at childIndx+1 exists and has more than t-1 blocks.
* @param childIndx
* @return
*/
private boolean childHasNonMinimalRightSibling(int childIndx);

/**
* Verifies the child node at childIndx has at least t blocks.<br>
* If necessary a shift or merge is performed.
*
* @param childIndxxs
*/
private void shiftOrMergeChildIfNeeded(int childIndx);

/**
* Add additional block to the child node at childIndx, by shifting from left sibling.
* @param childIndx
*/
private void shiftFromLeftSibling(int childIndx);

/**
* Add additional block to the child node at childIndx, by shifting from right sibling.
* @param childIndx
*/
private void shiftFromRightSibling(int childIndx);

/**
* Merges the child node at childIndx with its left or right sibling.
* @param childIndx
*/
private void mergeChildWithSibling(int childIndx);

/**
* Merges the child node at childIndx with its left sibling.<br>
* The left sibling node is removed.
* @param childIndx
*/
private void mergeWithLeftSibling(int childIndx);

/**
* Merges the child node at childIndx with its right sibling.<br>
* The right sibling node is removed.
* @param childIndx
*/
private void mergeWithRightSibling(int childIndx);
```

```
/**
* Finds and returns the block with the min key in the subtree.
* @return min key block
*/
private Block getMinKeyBlock();

/**
* Finds and returns the block with the max key in the subtree.
* @return max key block
*/
private Block getMaxKeyBlock();
```