עבודת בית מספר 4 – בודק איות

תאריך הגשה: 08.01.2021 בשעה 23:59

הוראות מקדימות

1. קראו היטב את השאלות.
2. **ניתן להגיש את העבודה בזוגות.**
3. הגשת העבודה תהיה דרך אתר הקורס במודל.
4. יש לכווץ את כל קבצי העבודה לקובץ zip. (קובץ zip יכיל את כל קבצי c. המרכיבים את הפרוייקט).
5. שם הקובץ שיוגש למערכת ההגשה יהיה מורכב מת.ז של המגיש/ים. לדוגמה:

   עבור הגשה ביחיד – 111111111.zip

   עבור הגשה בזוג – 111111111_222222222.zip
6. **אפשר להשתמש בקבצים המצורפים למטלה.**
7. במקרה של הגשה בזוגות, רק אחד מבני הזוג יגיש את העבודה במודל.
8. חובה לתעד את הקוד.

10. איחור במועד ההגשה יגרור הורדה של ציון, 5 נק' לכל יום איחור או חלק ממנו. בכל מקרה לא יהיה ניתן להגיש מעבר ל-3 ימי איחור ממועד ההגשה המקורי. במקרים חריגים בלבד יש לפנות למרצה כדי לקבל אישור על הגשה באיחור.
11. שאלות לגבי העבודה יש לשאול בפורום באתר הקורס ("מודל") או בשעות קבלה של המתרגל האחראי **בלבד**. *אין לשלוח שאלות במייל למרצה.*
12. להזכירם: יש לשמור על הגינות אקדמית.

הקדמה

בעבודה זו תממשו טבלת גיבוב (hash table) ותראו דוגמה של פעולת הגיבוב בפעולה.
בחלק הראשון תממשו ותעריכו את האיכות של שלוש טכניקות לגיבוב של מחרוזות קצרות. מה שתגלו תוך כדי השוואה הוא שלבחירה של פונקציית גיבוב ישנה השפעה קריטית על הביצועים של טבלת גיבוב.
בהמשך, תעזרו בטבלת הגיבוב שלכם כדי לממש בודק איות למילים באנגלית.

שימו לב, תרגיל זה הינו תרגיל <u>תכנותי</u> אחרון בקורס. היקף התרגיל רחב יותר מתרגילי תכנות קודמים, ולכן מומלץ להתחיל לחשוב על המימוש ולעבוד עליו כבר מהיום הראשון.

בודק איות

אם בקובץ הקלט תופיע המילה soun, בודק איות טוב ידווח שהמילה אינה מאויתת נכון, כיוון שהיא אינה מופיעה במילון. בנוסף, הבודק יציע את החלופות הבאות: "soon", "soul", "soup", "sun", "son" ו-"sound" כיוון שכולן דומות למילה soun. (בודקי איות אפילו יותר חכמים מזה יחפשו מילה שמתאימה יותר מבחינת משמעות של המשפט כולו, אבל זה מעבר למה שנדרש בעבודה זו.)

משימת בודק האיות מבוססת על חיפוש מילים ברשימת מילים מאוד ארוכה. חיפוש *יעיל* הינו קריטי לביצועים של בודק האיות, מכיוון שהחיפוש יתבצע לא רק עבור המילים בטקסט הקלט אלא גם עבור מאות המילים אותן הוא יציע כחלופה עבור המילים הלא מאויתות (כפי שנראה בהמשך). כפי שתראו בעבודה זו, פונקציית גיבוב גרועה תהפוך את בודק האיות לאיטי ולא שימושי.

חלק 1 – מימוש פונקציית גיבוב

בחלק זה תממשו פונקציות גיבוב שונות שלכל אחת מהן יש רמת פיזור שונה.
שימו לב: ניתן להניח כי הקלט עבור חלק זה הוא מחרוזת שונה מ-`null`.

פונקציית גיבוב קבועה

עליכם לממש את הפונקציה `constantStringHashFunction`.
ממשו את הפונקציה:

```
int constantStringHashFunction(char* str)
```

המחזירה 3 עבור כל קלט. כלומר, עבור כל מחרוזת קלט אפשרית, פונקציית גיבוב מחזירה ערך 3.

שאלה למחשבה: מה הבעיה בפונקציית גיבוב שכזו?

פונקציית גיבוב צוברת

עליכם לממש את הפונקציה `accumulateStringHashFunction`.
ממשו את הפונקציה:

```
int accumulateStringHashFunction(char* str)
```

המחזירה את סכום התווים של המחרוזת (בייצוג ה-ASCII שלהם).
עבור מחרוזת ריקה הפונקציה תחזיר 0.

לדוגמה, עבור המחרוזת "`My hash`", הפונקציה תחזיר 650, שכן סכום ייצוג התווים של "`My hash`" הינו:

$$077 + 121 + 032 + 104 + 097 + 115 + 104 = 650$$

שאלה למחשבה: מה הבעיה בפונקציית גיבוב שכזו?

פונקציית גיבוב משופרת

עליכם לממש את הפונקציה `improvedHashFunction`.
ממשו את הפונקציה:

```
int improvedHashFunction(char* str)
```

אשר מחזירה עבור המחרוזת `str` מאורך `n` תווים ערך הגיבוב שמחושב לפי הנוסחה הבאה:

$$\text{str}[0] * 31^{n-1} + \text{str}[1] * 31^{n-2} + \ldots + \text{str}[n-1] = \sum_{i=0}^{n-1} \text{str}[i] * 31^{n-1-i}$$

עבור מחרוזת ריקה הפונקציה תחזיר 0.

לדוגמה, עבור המחרוזת "`My hash`", הפונקציה תחזיר 1179785502-.
לפי החישוב:

$$077 * 31^6 + 121 * 31^5 + 032 * 31^4 + 104 * 31^3 + 097 * 31^2 + 115 * 31^1 + 104 * 31^0$$

מדוע קיבלנו ערך שלילי?

מאחר והתוצאה כל כך גדולה (ולא ניתן לייצג אותה בעזרת משתנה int) אז נגרם integer overflow ולכן אנחנו מקבלים תוצאה שלילית. ניתן לקרוא יותר על הנושא בלינק הבא:
https://en.wikipedia.org/wiki/Integer_overflow

שאלה למחשבה: מה הבעיה בפונקציית גיבוב שכזו?

חלק 2 – מימוש טבלת גיבוב

בחלק זה תממשו `HashTable` המייצגת טבלת גיבוב של מחרוזות. פתרון בהתנגשויות יעשה ב**שיטת Hashing with chaining**.
יש למשש את הפונקציות המפורטות מטה ומצויינות בקובץ HashTable.h.
אתם רשאים לממש פונקציות עזר נוספות.

פונקציות למימוש

אתחול הטבלה

עליכם לממש את הפונקציה:

```
HashTable* initTable(int tableSize, int hashFunction)
```

המקבלת את גודל הטבלה ומספר פונקציית hash (1 - עבור פונקציית הגיבוב הקבועה, 2 - עבור פונקציית הגיבוב הצוברת, 3 - עבור פונקציית הגיבוב המשופרת), ויוצרת טבלה ריקה (ללא איברים) מגודל `tableSize` ופונקציית הגיבוב, שבעזרתה נקבעים המפתחות של האיברים תתאים למספר ה - `hashFunction`.
ניתן להניח כי `tableSize` הוא מספר חיובי וש-`hashFunction` הוא מספר בטווח 1-3.

גיבוב

עליכם לממש את הפונקציה:

```
int* hash(char* str, HashTable* ht)
```

המקבלת מחרוזת לגיבוב וטבלת גיבוב בה רוצים לשמור את המחרוזת.
הפונקציה מחשבת ומחזירה index שבו נאחסן את המחרוזת.
כדי לחשב את ה-index של מחרוזת `str` בטבלה נבצע:

$$index = |hashFunction(str)|(mod\ \ tableSize)$$

כאשר hashFunction מתייחס לפונקציית הגיבוב המתאימה לHashTable שהתקבלה.
משמע: הערך המוחלט של `hashFunction (str)` מודולו גודל הטבלה שהתקבלה.
חישוב זה יבטיח כי המפתחות יהיו בטווח שבין 0 ל-`tableSize-1`.

שאלה למחשבה: תחת חישוב זה, מה יהיה גודל הטבלה האופטימלי?

הכנסה

עליכם לממש את הפונקציה:

```
int insert(HashTahble* ht, char* str)
```

המקבלת טבלה ומחרוזת, ומכניסה את המחרוזת לטבלה.
במידה והקלט לא תקין (`null`) או במידה והמחרוזת כבר נמצאת בטבלה יש להחזיר 0 מבלי לשנות את הטבלה.
אחרת, המחרוזת תוכנס לתא המתאים בטבלה (לפי פונקציית הגיבוב) ויוחזר 1.
כאמור, ניהול הטבלה לפי hashing with chaining (ברשימה המקושרת נוסיף את המילה לתחילת הרשימה).

הוצאה

עליכם לממש את הפונקציה:

```
int deleteElement(HashTable* ht, char* str)
```

המקבלת מחרוזת וטבלה ומוחקת את המחרוזת מהטבלה.
במידה והקלט אינו תקין (`null`) או במידה והמחרוזת לא קיימת בטבלה יש להחזיר 0 מבלי לשנות את הטבלה.
אחרת, יש למחוק את המחרוזת מהטבלה ולהחזיר 1.

חיפוש

עליכם לממש את הפונקציה:

```
int search(HashTable* ht, char* str)
```

המקבלת מחרוזת וטבלה ובודקת האם המחרוזת קיימת בטבלה.
במידה והקלט אינו תקין (`null`) או במידה והמחרוזת לא קיימת בטבלה יש להחזיר 0.
אחרת, יש להחזיר 1.

חלק 3 – קריאה מקובץ

בחלק זה תממשו פונקציה לקריאת מילים מקובץ ושמירתם בטבלת הגיבוב שמימשתם עד כה.
המימוש של חלק זה יהיה בקובץ Driver.c. ההכרזות על הפונקציות כבר קיימות בקובץ.

מבנה קובץ המילון

נתון הקובץ `dictionary.txt` המכיל 3000 מילים בשפה האנגלית.
כל מילה מופיעה בשורה נפרדת ומורכבת רק מאותיות קטנות באנגלית (ללא אותיות גדולות או תווים מיוחדים).
אין בקובץ כפילויות של מילים.

parser

עליכם לממש את הפונקציה:

```
int parseWordsToTable(char* filePath, HashTable* ht)
```

המקבלת טבלת גיבוב ומחרוזת המכילה את ה-path של הקובץ (הכתובת של הקובץ במחשב). לדוגמה:
`"C:/Users/Unicorn/Documents/DS/Assignment4/dictionary.txt"`
שימו לב לכיוון ה- `'/'`!

הפונקציה פותחת את הקובץ לקריאה, קוראת את המילים שורה אחר שורה, מאכסנת אותן ב-`table` <u>וסוגרת את הקובץ</u>.
במידה והקלט אינו תקין (`null`) או שקיימת בעיה בקריאת הקובץ, יש להחזיר `0`.
ניתן להניח כי במידה והפונקציה החזירה `0`, המשתמש לא יעשה שימוש ב-`hash table`.
במידה והקריאה והאחסון הצליחו יש להחזיר `1`. במקרה זה הטבלה תכיל את כל המילים שנקראו מהקובץ.
בשום מצב הפונקציה לא צריכה לזרוק חריגה או לקרוס!

חלק 4 – בודק מילה

בחלק זה תממשו את ה `WordSpellingChecker` המאפשר לבצע בדיקה האם מילה קיימת במילון ומציע תיקון למילים שאויתו באופן שגוי לפי אחת מהשיטות הבאות:

1. פיצול המילה לצמד מילים על ידי הוספת רווח בין כל שתי אותיות סמוכות במילה. אם שתי המילים שנוצרו מופיעות במילון, הצמד יוצע כחלופה.
2. החלפת כל אחת מהאותיות במילה בכל אחת מהאותיות `'a'` עד `'z'`.
3. מחיקת כל אחת מהאותיות במילה.
4. הכנסת אות אחת, בין `'a'` ל- `'z'`, בין כל שתי אותיות סמוכות, לפני האות הראשונה ואחרי האות האחרונה.
5. החלפת כל שתי אותיות סמוכות במילה.

בחלק זה ניתן להניח שכל המילים מורכבות מאותיות קטנות באנגלית וללא רווחים או תווים מיוחדים אחרים.

הופעת מילה במילון

עליכם לממש את הפונקציה:

```
int isWordInDictionary(HashTable* dictionaryTable, char* word)
```

המקבלת מילה ומילון ועונה האם המילה קיימת במילון.
אם הקלט לא תקין (`null`) או שהמילה לא מופיעה במילון על הפונקציה להחזיר `0`. אם המילה מופיעה במילון, הפונקציה תחזיר `1`.

לדוגמה, עבור קובץ ה- dictionary.txt המצורף לעבודה השורה:

```
isWordInDictionary(dictionaryTable, "beer")
```

תחזיר 1.

עבור השורה:

```
isWordInDictionary(dictionaryTable, "beel")
```

יוחזר 0.

בדיקות מילה בשיטות שונות

עליכם לממש את הפונקציות להצעת חלופות למילה שאויתה באופן שגוי.
אלא אם צוין אחרת, אם הקלט לא תקין (null) או לא ניתן לבצע את הפעולה, הפונקציה תחזיר רשימה ריקה.
במידה ולא קיימות הצעות עבור הקלט, הפונקציה תחזיר רשימה ריקה.

שימו לב! הקלט של חמשת הפונקציות הבאות עשוי להיות מילה שקיימת במילון.
בכל מקרה רשימת הפלט <u>לא תכיל את המילה המקורית</u>.

בדיקה 1 – פיצול מילים

עליכם לממש את הפונקציה:

```
LinkedList* addSpaceCheck(HashTable* dictionaryTable, char* word)
```

המקבלת מילון ומילה ומחזירה רשימה של הצעות לתיקון האיות.
הפונקציה לוקחת את המילה ועבור כל זוג אותיות סמוכות היא מוסיפה רווח (' ').
אם שתי המילים שנוצרו בעקבות הוספת הרווח מופיעות במילון, הפונקציה תצרף את המחרוזת שמכילה את שתי המילים הללו לרשימה (עם רווח ביניהן). כל מחרוזת שכזו תהיה מהצורה "<word1> <word2>" (המילה הראשונה, רווח, המילה השנייה) כאשר <word1> וגם <word2> קיימות במילון.

לדוגמה, עבור השורה:

```
addSpaceCheck(dictionaryTable, "idealer")
```

יוחזר אובייקט מסוג LinkedList* המכיל את שתי המחרוזות:

```
"i dealer"
"ideal er"
```

בדיקה 2 – החלפת אות

עליכם לממש את הפונקציה:

```
LinkedList* replaceCharacterCheck(HashTable* dictionaryTable, char*
word)
```

המקבלת מילון ומילה ומחזירה רשימה של הצעות לתיקון האיות.
הפונקציה מחליפה כל תו במילה ומציבה במקומה כל אחד מהתווים בין 'a' ל-'z'. אם המילה החדשה (כלומר, מילה אחרי החלפת אות) מופיעה במילון אז היא תתווסף לרשימת ההצעות.

לדוגמה, עבור השורה:

```
replaceCharacterCheck(dictionaryTable, "bake")
```

יוחזר אובייקט מסוג LinkedList* המכיל את שמונה המחרוזות:

```
"cake"
"lake"
```

"make"
"sake"
"take"
"wake"
"bike"
"base"

<u>שימו לב</u>: המחרוזת "bake" לא תופיע בפלט.

בדיקה 3 – מחיקת אות

עליכם לממש את הפונקציה:

```
LinkedList* deleteCharacterCheck(HashTable* dictionaryTable, char*
word)
```

המקבלת מילון ומילה ומחזירה רשימה של הצעות לתיקון האיות.
הפונקציה מוחקת כל תו במילה ואם המילה החדשה מופיעה במילון אז היא תתווסף לרשימת ההצעות.

לדוגמה, עבור השורה:

```
deleteCharacterCheck(dictionaryTable, "contesxt")
```

יוחזר אובייקט מסוג `LinkedList*` המכיל את שתי המחרוזות:

"context"
"contest"

בדיקה 4 – הוספת אות

עליכם לממש את הפונקציה:

```
LinkedList* addCharacterCheck(HashTable* dictionaryTable, char* word)
```

המקבלת מילון ומילה ומחזירה רשימה של הצעות לתיקון האיות.
בין כל שתי אותיות סמוכות, לפני ואחרי המילה, הפונקציה מוסיפה כל אחד מהתווים בין 'a' ל-'z'. אם המילה החדשה מופיעה במילון אז היא תתווסף לרשימת ההצעות.
במידה והמילה היא מחרוזת ריקה הפונקציה תחזיר רשימה ריקה.

לדוגמה, עבור השורה:

```
addCharacterCheck(dictionaryTable,"son")
```

יוחזר אובייקט מסוג `LinkedList*` המכיל את שלושת המחרוזות:

"soon"
"soon"
"song"

שאלה למחשבה: למה soon מופיעה פעמיים?

למרות שמילה "son" מופיעה במילון היא <u>לא</u> תופיע בפלט.

בדיקה 5 – החלפת אותיות שכנות

עליכם לממש את הפונקציה:

```
LinkedList* switchAdjacentCharacterCheck(HashTable* dictionaryTable,
char* word)
```

המקבלת מילון ומילה ומחזירה רשימה של הצעות לתיקון האיות.

עבור כל שתי אותיות סמוכות הפונקציה מבצעת החלפה ואם המילה החדשה מופיעה במילון אז היא תתווסף לרשימת ההצעות. ז"א לכל $0 \leq i < word.length$ הפונקציה תחליף בין התו ה-i והתו ה-$i + 1$.

לדוגמה, עבור השורה:

```
switchAdjacentCharacterCheck(dictionaryTable, "ksis")
```

יוחזר אובייקט מסוג `LinkedList*` המכיל את המחרוזות:

```
"kiss"
```

חלופות איות

עליכם לממש את הפונקציה:

```
LinkedList* getWordSuggestions(HashTable* dictionaryTable, char* word)
```

המקבלת מילון ומילה ומחזירה רשימה של הצעות לתיקון האיות.

רשימת ההצעות מורכבת ממילים לפי כל אחת מחמשת השיטות שתוארו למעלה. הרשימה לא תכיל כפילויות או את המילה עצמה, כלומר יש לאחד כל הצעות ולמחוק כפילויות. במידה והקלט `null` או מחרוזת ריקה (`""`), הפונקציה תחזיר רשימה ריקה.

לדוגמה, עבור השורה:

```
getWordSuggestions(dictionaryTable, "soun")
```

יוחזר אובייקט מסוג `LinkedList*` המכיל את המחרוזות:

```
"soon"
"soul"
"soup"
"sun"
"son"
"sound"
```

SpellingSuggestion

המבנה `SpellingSuggestion` נועד לאכסן מילה שגויה ואת רשימת ההצעות שלה ומצביע להצעה הבאה.

המבנה נמצא בקובץ WordSpellingChecker.h.

חלק 5 – בודק איות

את החלק זה תממשו בקובץ `Driver.c` ותמצאו טעויות איות בקטע טקסט. לכל מילה שאויתה באופן שגוי יוצגו מילים חלופות.

בודק איות

עליכם לממש את הפונקציה:

```
SpellingSuggestion* spellingCheck(char* text)
```

המקבלת מחרוזת מילים ומחזירה רשימה של מילים שגויות והצעות עבורן.

ניתן להניח שמחרוזת הקלט מורכבת אך ורק מאותיות קטנות ורווחים (ללא סימנים מיוחדים נוספים). בין כל שתי מילים במחרוזת יש תו רווח (ללא רווח בהתחלה ובסוף).

לכל מילה במחרוזת הקלט, הפונקציה בודקת האם המילה נמצאת במילון ובמידה ואינה, הפונקציה תחשב עבור המילה רשימה של הצעות. לאחר מכן הפונקציה תייצר רשימה מסוג `SpellingSuggestion*` עם המילים השגויות ורשימת ההצעות. כמו כן:

- **הפלט לא יכיל כפילויות**. משמע, אם המילה `"eevee"` זוהתה כשגויה אך כבר קיים ברשימת הפלט אובייקט `SpellingSuggestion` שבשדה ה-`originalWord` שלו יש את הערך `"eevee"` אז הפונקציה לא תוסיף אובייקט נוסף לפלט.
- במידה ולא נמצאו המלצות למילה שגויה, האובייקט `SpellingSuggestion` שלה ייווצר עם שדה `suggestionList` המכיל רשימה ריקה.
- אין חשיבות לסדר המילים.
- הפלט יכיל רק מילים שלא קיימות במילון (לא מאויתות נכון).
- במידה והקלט לא תקין (`null` או מחרוזת ריקה) הפונקציה תחזיר רשימה ריקה.
- במידה לא נמצאו מילים לא תקינות הפונקציה תחזיר מחרוזת ריקה.

לדוגמה, עבור המחרוזת:

```
“iam afraid youare about to become teh immexdiate pst president of teh
eing alive club ha ha glados”
```

הפונקציה `spellingCheck` תחזיר רשימה עם 8 איברים מסוג `SpellingSuggestion`.
הדפסה של הקלט (שעליכם לממש) תציג למסך את הטקסט הבא:

```
[The word "iam" was misspelled. Did you mean:
      i am
      am
      aim
, The word "youare" was misspelled. Did you mean:
      you are
, The word "teh" was misspelled. Did you mean:
      tea
      ten
      the
, The word "immexdiate" was misspelled. Did you mean:
      immediate
, The word "pst" was misspelled. Did you mean:
      pet
      pot
      put
      past
      post
, The word "eing" was misspelled. Did you mean:
      king
      ring
      sing
      wing
      being
, The word "ha" was misspelled. Did you mean:
      he
      hi
      a
      hat
      ah
, The word "glados" was misspelled. No suggestions found for this
word.
```

בהצלחה!