מערכת לאנליזת קודים מתקני שגיאות

האוניברסיטה הפתוחה, פרויקט מתקדם במדעי המחשב (22997), סמסטר 2015א

בהנחיית: ד"ר אלעזר בירנבוים, ד"ר אמיר ספיר

נדב וולך, ת.ז. 040582157

2014-2015

תוכן העניינים

<u>פרטים אישיים:</u>

שם: נדב וולך ;

ת.ז.: 040582157 ;

כתובת: ת.ד. 229, קיבוץ ברור חיל. מיקוד: 7915200 ;

טלפון : 054-5911243 ;

דואר אלקטרוני: nvolloch@yahoo.com ;

<u>פרטים על הפרויקט:</u>

שם הפרויקט : מערכת לאנליזת קודים מתקני שגיאות.

שמות המנחים : ד"ר אלעזר בירנבוים, ד"ר אמיר ספיר

מטרות הפרויקט

מטרת הפרויקט הינה מימוש של חיפוש וניתוח קודים מתקני שגיאות בגדלים מסוימים. פרויקט זה נלווה למחקר שנעשה על –ידי ועל ידי המנחים בשנים האחרונות, ובו עסקו סמינר שעשיתי בנושא, תזה שאני כותב בימים אלו ומאמר שהתקבל ופורסם.

מטרות המערכת שנבנתה הינן:

א. חיפוש בסגנון backtracking של קודים בגדלים אופטימליים או קרובים לאופטימליים.

ב. חיפוש עם היוריסטיקה מסוימת - המתאימה לסכמת הניפוח (כמו שמוסבר ברקע לעבודה) על ידי מספר מלות הקוד ש'כדאי' לקחת מכל משטח (הסבר על משטח בחלק הבא – רקע עם התייחסות למקורות).

ג. שילוב בין שתי הגישות באי' ובי', כאשר הקווים המנחים לחיפוש הם לפי הקונפיגורציה הרצויה בבי' (לפי סיומות המחרוזות), ובכל שלב של "לקיחת" מילות קוד ממשטח יתבצע חיפוש בסגנון backtracking שמיועד לבצע אופטימיזציה לתוצאות.

הסבר נוסף ומפורט על האלגוריתמים ומימושם מובא בנספח א'.

רקע עם התייחסות למקורות

בשנים האחרונות ישנם מחקרים רבים העוסקים בקודים על גרפים, מה שכנראה התחיל ב[1], והמשיך במאמרים נוספים (כגון [7], [3], [5], [9]). באופן טיפוסי בכאלו עבודות, גרף מסוים משמש כבסיס, וקדקודיו מתויגים במחרוזות על פי כלל כלשהו. תת-קבוצה של קדקודי הגרף נבחרת כקוד – המחרוזות של קדקודי תת-הקבוצה הן מילות הקוד. בדרך כלל (באופן רצוי) קבוצה זו של קדקודים מהווה קבוצה שלטת מושלמת (כלומר, קבוצה U של קדקודים כך שכל קדקוד בגרף, או שהוא שייך ל-U, או שהוא מחובר בקשת לקדקוד יחיד ב-U). אם מתרחשת שגיאה, מוצאים את הקדקוד של ההודעה השגויה שהתקבלה, ומתקנים אותה להיות המחרוזת שנמצאת בקדקוד היחיד של הקוד שקשור בקשת לקדקוד של ההודעה השגויה שהתקבלה.

ניתן לראות כי ב[4] מוצג 1-ECC מושלם (כלומר, קוד המאפשר תיקון של כל שגיאה אחת שקרתה). קוד זה בנוי על גרף הקונפיגורציות של מגדלי הנוי- בו אנו עוסקים. בנוסף, ב[6] ישנה הרחבה של [4] עבור גרפים בתצורת שרפינסקי (גרפים בתצורה פרקטלית).

הבעיה במקרים מסוג אלו שתוארו לעיל היא חוסר התאמה בין מרחקי המינג של המחרוזות למרחקים בין הקדקודים המייצגים אותן בגרף. ייתכנו מחרוזות שמרחק המינג ביניהן הוא 1, והן לא נמצאות בקדקודים שכנים בגרף. עבור בעיה זו בנינו, ב[8] ובעבודה לאחריו אלגוריתם "ניפוח" לגרף קונפיגורציות הנוי, הפותר בעיה זו. בכל גרף שעובר ניפוח –המחרוזות, כמובן, הופכות למחרוזות באורך גדול יותר, הכוללות את המחרוזת המקורית ואת הניפוח כסיפא. את הגרף החדש אנו יוצרים כשכפולים (כמו הפרקטלים של תצורת שרפינסרקי) של הגרף הקודם (ללא הניפוח). כל שכפול ייקרא "משטח", כאשר לכל אחד מהמשטחים המרכיבים את הגרף המנופח יהיה ניפוח שונה. עבור עבודה זו קיים הצורך להמשיך ולמצוא תוצאות נוספות המתאימות לסכמת ניפוח זו, מה שבא פרויקט תכנותי זה לעשות.

בנוסף, ואף חשוב מכך, ב[2] מוצגים כמה חסמים לגודלי קוד אפשריים עבור אורכי מחרוזות שונים ומרחקי המינג שונים בהתאמה (בין מילות הקוד), כאשר ב[8] ובאמצעות המערכת, הגענו לתוצאות עבור כמה גדלים. פרויקט זה מיועד לבחינת קודים שונים שהוצגו באתר המלווה את [2] (http://www.win.tue.nl/~aeb/codes/ternary-1.html) ולטובת יצירת קודים חדשים עבור גדלים נוספים.

תיאור הפרויקט

הפרויקט הינו מערכת הכתובה בשפת Java (על בסיס ספריית javax.swing)אשר בהינתן (על ידי ממשק משתמש ידידותי) נתונים על אורכי המחרוזות, גודל האלפבית, מרחק המינג רצוי ונתונים נוספים (אופציונאלי), מבצעת כמה דברים :

א. מוצאת קודים מקסימליים לפי הנתונים.

ב. כתהליך מקדים ומסייע לסעיף א'- ספרית עזר שכוללת פרוצדורות עבור :

1. חישוב מרחק הקוד.
2. קביעה האם שני קודים הם זהים, כלומר האם הם מכילים את אותן מילים בדיוק, רק בסדר הצגה שונה, והאם הם מקרה פרטי של שקילות, כלומר האם ניתן להגיע לזהות ע"י אותה תמורה של כל ביט בנפרד בכל המילים.
3. חישוב מדדים מסוימים עבור הקוד : מרחק ממוצע, פונקציית התפלגות המרחק עבור הקוד ועבור "תתי-קודים".
4. קביעה האם קוד נתון הוא ליניארי.
5. איתור מילים שאינן מכוסות על ידי אף מילת קוד.

ופרוצדורות נוספות.

חשיבות הפרויקט

חשיבותו של הפרויקט מתבטאת בעזרתו למחקר אליו הוא נלווה (עליו הוסבר ברקע), ולמחקרים קודמים בנושא. הפרויקט אמור לספק תוצאות ממשיות של מציאת קודים עבור ערכים ספציפיים שנחקרים, ועליהם ישנה התמקדות (אלפבית טרינארי). בנוסף, יש חשיבות בעצם יישום האלגוריתמים המובאים במחקר רק באופן תיאורטי. רובד משמעותי נוסף הוא עזרה בשיפור אלגוריתמים אלו על ידי מציאות תוצאות נוספות בדרכים שונות, ועל ידי כך עזרה במציאת טעויות אפשריות, או, לחילופין, כיווני מחשבה ומחקר חדשים.

שיטות לביצוע הפרויקט

בפרויקט התבצעה התייחסות לתוצאות ב[2] עבור בדיקת קודים שונים (מפורשים) שנמצאים באתר המלווה למאמר (אליו יש קישור בחלק הרקע), ויצירה של קוד לפי האלגוריתם מ[8] עבור התאמתו לתצורת גרפים חדשה שהוצעה במחקר- כאשר כל קודקוד מהקבוצה השלטת בגרף מהווה מילת קוד והקודקודים הסמוכים לו מייצגים את המילים האחרות (אלו שלא בקוד) ומתכנסים לתיקון השגיאות למילת הקוד. כלי העזר בשימוש בפרויקט הם סביבת העבודה Eclipse IDE והקוד עצמו נכתב בשפת Java. הרכיבים בהם נעשה שימוש לממשק המשתמש הפשוט הם בעיקרם רכיבי JOptionPane, כאשר ממשק המשתמש כתוב בעיקרו ברכיבי ספריית javax.swing.

דגש על כך שבפרויקט הגרף אינו מפורש מבחינה ויזואלית (הדמיה ויצירת הקודקודים והצלעות), כיוון שדרישה זו מגבילה את היכולת לעבד נתונים בגדלים משמעותיים יותר, אלא ההתייחסות לנתונים הינה כחלקים בגרף, לא כמבנה נתונים, אלא כקוד מתמטי (אוסף של מילות קוד והתייחסות למילים שאינן בקוד).

הסבר על המערכת והקבצים הנלווים

הפרויקט נמצא בקובץ זיפ – בכדי להפעילו עלינו לחלץ את קובץ הזיפ לתיקייה רגילה (בגלל בעיית העברה של קבצי זיפ בעבר, רצוי על שולחן העבודה).

שנית, יש ללחוץ לחיצה כפולה על קובץ run בתיקייה, בשלב זה המערכת עולה.

כעת- ישנם שני כפתורים למעלה, שטח טקסטואלי גדול במרכז החלון, וארבעה כפתורים למטה (תמונה מס׳ 1 המובאת בעמוד הבא).

העלאת קובץ טקסט המייצג קוד- דרך הכפתור למטה (שמאלי), כאשר אחרי לחיצה זו אנו מתבקשים להזין את אורך המחרוזת (בשמות קבצי הדוגמה הנלווים אורך זה מצוין כ-n) ואת גודל האלפבית (3 ברוב קבצי הדוגמה הנלווים, יש אחד עם 6- בו רשום alpha6... בשם הקובץ).

אחרי הזנת גודל מחרוזת ואלפבית- ישנו חלון המבקש לדעת אם הקוד רשום בצורה דצימלית (באתר של Brouwer- רוב הקודים המפורשים מופיעים בצורה כזו, קישור לאתר: http://www.win.tue.nl/~aeb/codes/ternary-1.html)- אם כן, צריך להזין את אורך המחרוזת הדצימלית (3 בקבצים שלנו) והתכנית מבצעת המרה אוטומטית לאלפבית הרצוי.

אחרי פעולות אלו מופיע הקוד בחלון והפרטים הבאים: גודלו, והאם ליניארי או לא.

כעת, ניתן ללחוץ על כפתור המחשב מרחק ומרחק ממוצע ולקבל פרמטרים אלו. כמו כן, ניתן ללחוץ על כפתור המילים שלא מכוסות ע״י 1-ECC - מה שייתן את כל המילים האלה, מספרן והיחס בין מספרן לכלל המילים.

בנוסף- ניתן לבצע (כפתור ימני למטה) העלאה של קובץ קוד נוסף- כאשר יש להקפיד שהוא מאותו סדר גודל (אלפבית ואורך מחרוזת) ומתבצעת בדיקה האם שני הקודים זהים, אם לא - התכנית מחזירה את המילים בהן נבדלים הקודים.

בנוסף – בכפתור הימני למעלה ניתן ליצור קוד חדש לפי הפרמטרים הרצויים לנו (אורך מחרוזת, גודל אלפבית וכולי) באלגוריתם backtrack שיוצר קודים מתקני שגיאות.

תמונה 1: מסך פתיחה של המערכת

לגבי קבצי הטקסט המצורפים ותסריטי שימוש לבדיקות שניתן לעשות עמם:

קובץ n5- אלפבית טרינארי, גודל מחרוזת 5, גודל קוד 18 .

קובץ n5decimalSmaller-17 - אלפבית טרינארי, גודל מחרוזת 5, גודל קוד 17- מופיע באופן דצימלי, כלומר צריך לבצע המרה, ניתן להשתמש בו להשוואה עם n5.

קובץ n6- אלפבית טרינארי, גודל מחרוזת 6, גודל קוד 38 (קוד שבוצעה לו המרה מהאתר של Brouwer).

קובץ n6decimal - אלפבית טרינארי, גודל מחרוזת 6, גודל קוד 38 - אותו קוד בדיוק כמו n6, רק צריך המרה- ניתן להשוותו לn6 ולראות שהם שווים.

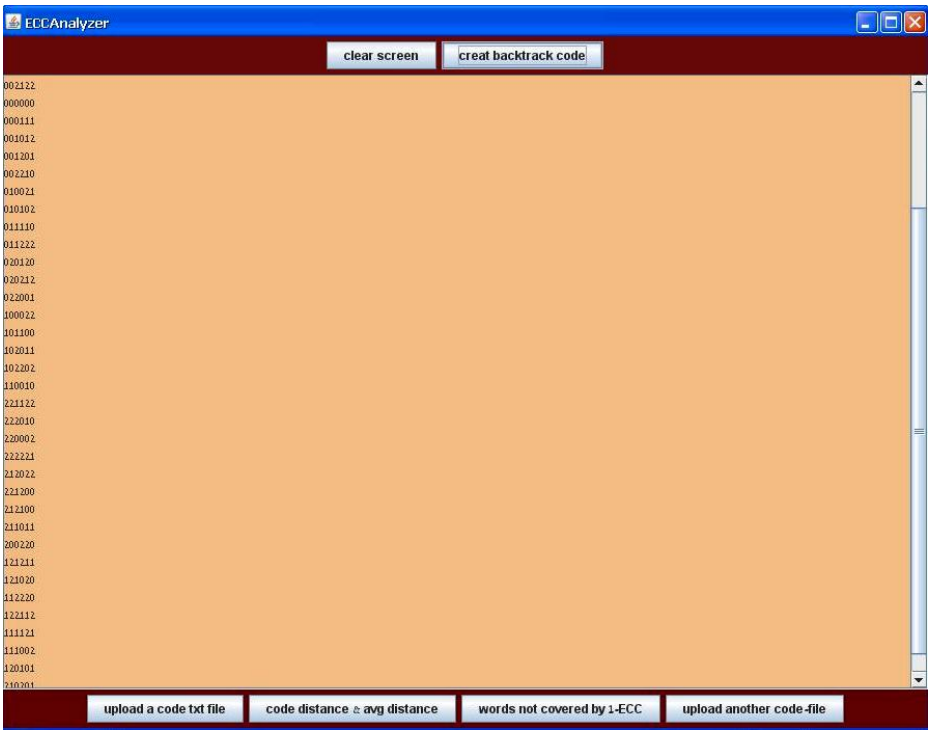
קובץ n12d7- אלפבית טרינארי, גודל מחרוזת 12, גודל קוד 60 (קוד שבוצעה לו המרה מהאתר של Brouwer).

קובץ n14d9- אלפבית טרינארי, גודל מחרוזת 14, גודל קוד 36 (קוד שבוצעה לו המרה מהאתר של Brouwer).

קובץ alpha6n5 - אלפבית בבסיס 6, גודל מחרוזת 5, גודל קוד 144.

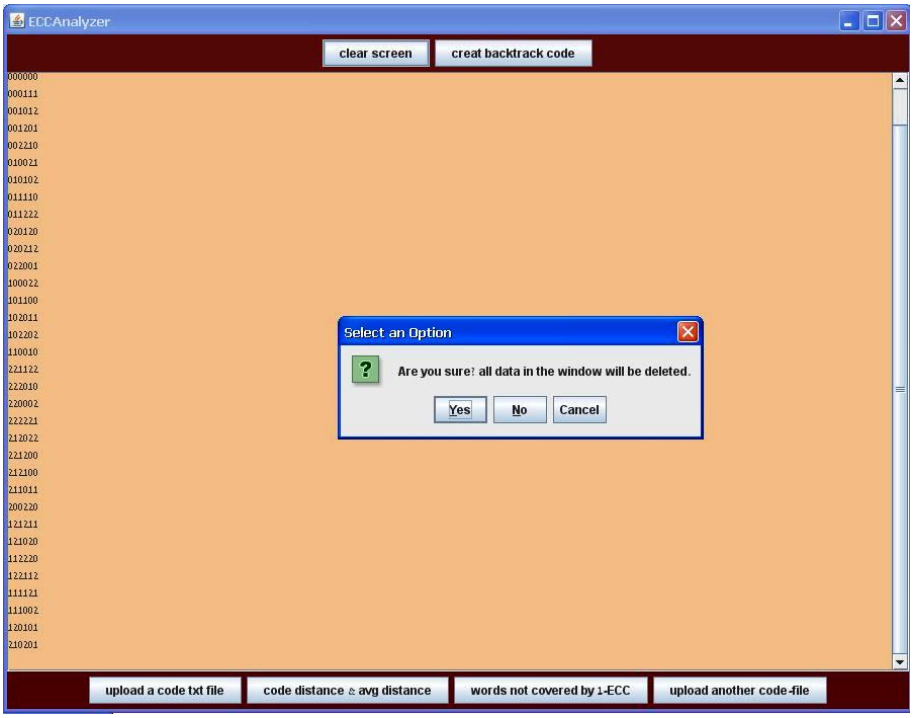
הדגמות שימוש בכל חלקי המערכת

א. יצירת קוד backtrack עם מרחק המינג 3, אלפבית טרנארי, n=6 (גודל קוד 35, כפתור ימני למעלה):



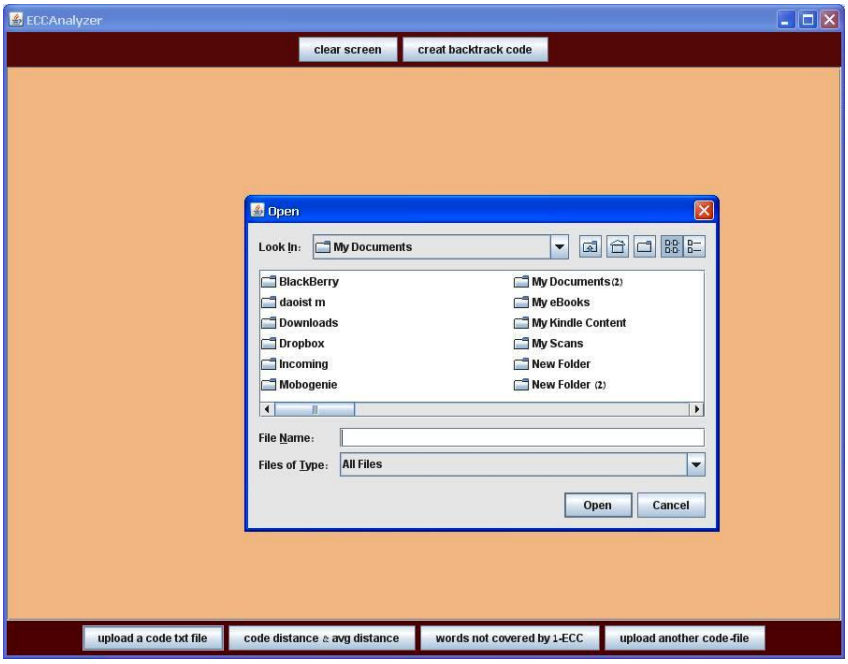
תמונה 2: יצירת קוד backtrack

ב. ניקוי מסך (כפתור שמאלי למעלה):



תמונה 3: ניקוי מסך

ג. העלאת קובץ טקסט (כפתור שמאלי למטה):

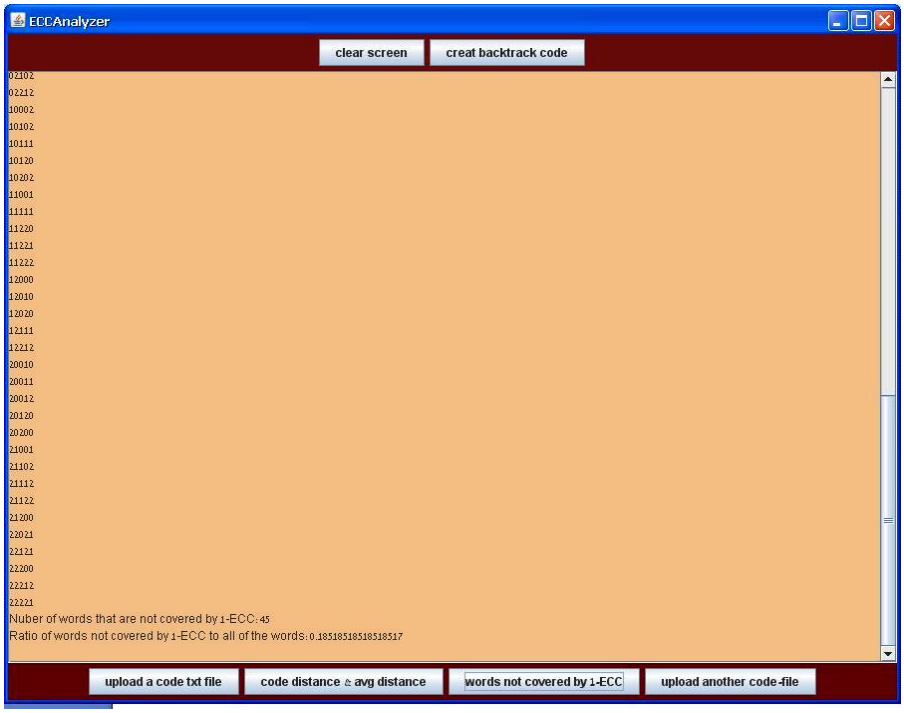


תמונה 4: העלאת קובץ טקסט

ד. חישוב מרחק ומרחק ממוצע של קוד (כפתור שני משמאל למטה):

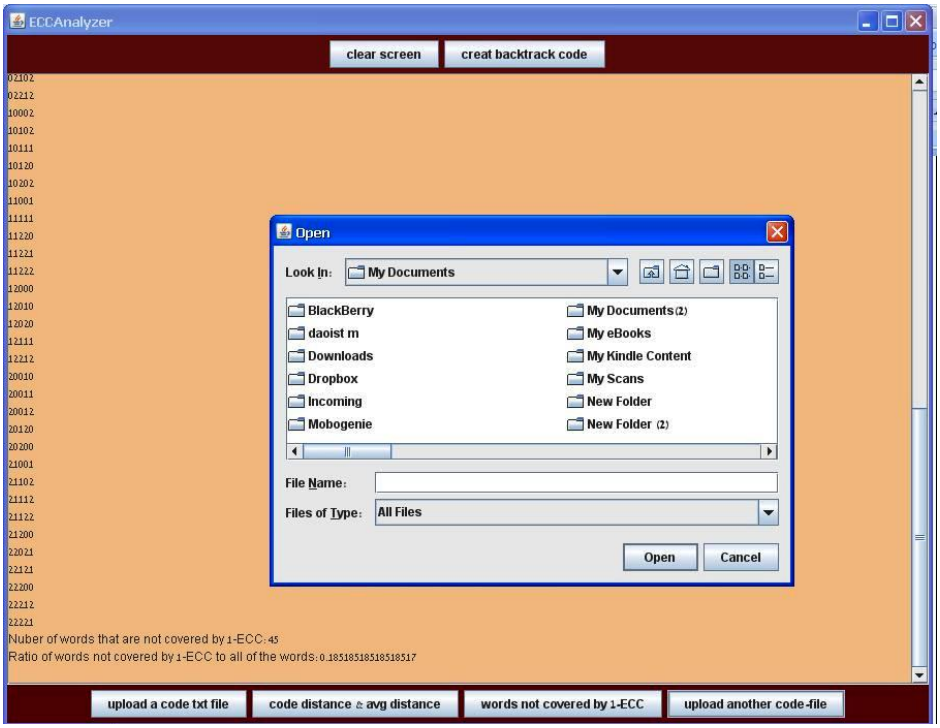
תמונה 5: חישוב מרחק ומרחק ממוצע

ה. חישוב מילים שלא מכוסות ע"י 1-ECC (כפתור שני מימין למטה):



תמונה 6: חישוב מילים שלא מכוסות ע"י 1-ECC

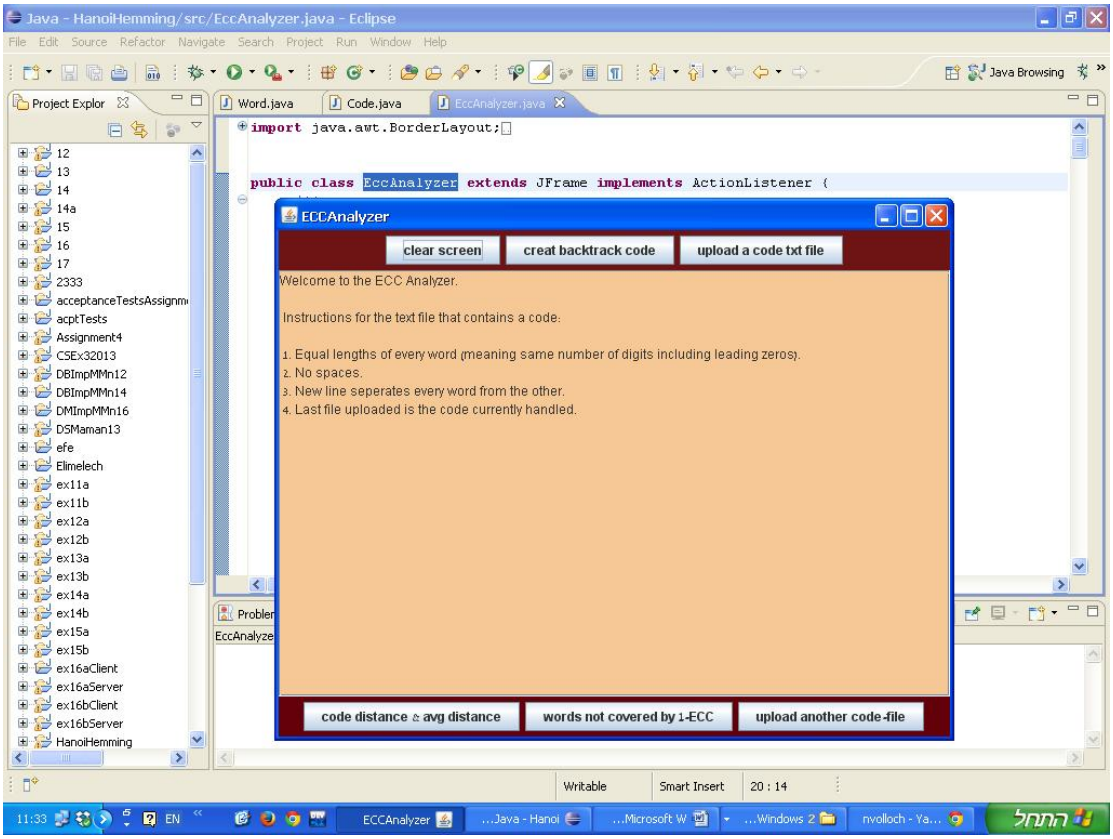
ו. העלאת קובץ טקסט נוסף (כפתור ימני למטה):



תמונה 7: העלאת קובץ טקסט נוסף

מבנה המערכת

המערכת בנויה משלושה קבצים שהינם Word.java (אובייקט, מייצג מילה כללית), Code.java (אובייקט, מייצג קוד לתיקון שגיאות) וEccAnalyzer.java שהינה מחלקת הרצה המכילה גם את ה-GUI של המערכת ועובדת עם שני האובייקטים שצוינו לעיל דרך פונקציות שונות, שתפורטנה בהמשך מסמך זה, ומתרגמת את הקוד התיאורטי לממשק ויזואלי נוח למשתמש, כפי שניתן לראות בתמונה למטה.



תמונה 8: הפעלת המערכת דרך התכנית בסביבת עבודה Eclipse

קובץ `Word.java` (קבצי המערכת)

משתני מחלקה (משתנים גלובליים)

`int alphabetSize` – מייצג את גודל האלפבית.

`int length` – אורך המחרוזת (מילה).

`int[] wordArr` – מערך של מספרים שלמים המייצג את המילה עצמה.

פונקציות המחלקה

`public Word(int[] newWordArr,int newAlphabetSize)` – בנאי רגיל המקבל מערך של מספרים **שלמים** וגודל אלפבית.

`public Word()` - בנאי ברירת מחדל.

`public int getAlphabetSize()` - `getter` לגודל האלפבית.

`public int getLength()`-`getter` לאורך המילה.

`public int[] getWordArr()` - `getter` למערך ייצוג המילה.

`public int distance(Word other)` - פונקציית חישוב מרחק המינג בין שתי מילים.

`public Word decimalToAlphabetBase(int decimalInt, int alphabetSize, int strLength )` – פונקציית המרה של מילה בבסיס דצימלי לבסיס האלפבית המבוקש.

`public int wordCompare (Word other)` - פונקציית השוואה לקסיקוגרפית בין שתי מילים (מחזירה את תוצאת ההשוואה בין המילה למילה שמתקבלת כפרמטר- קטנה מ-, גדולה מ- או שווה).

`public int wordCompareEndToStart (Word other)` - פונקציית השוואה לקסיקוגרפית בין שתי מילים – כמו הקודמת, רק מימין לשמאל (מחזירה את תוצאת ההשוואה בין המילה למילה שמתקבלת כפרמטר- קטנה מ-, גדולה מ- או שווה).

`public int[] offset(Word other)` - פונקציה הבודקת את ההפרש בין המילה הנוכחית לזו שמתקבלת כפרמטר (מחסרת ביניהן) – לטובת בדיקת שקילות הקוד.

`public String toString()` - פונקציית המחזירה ייצוג של המילה בצורת מחרוזת.

קובץ Code.java (קבצי המערכת)

משתני מחלקה (משתנים גלובליים)

Vector<Word> words – מייצג (על ידי וקטור של מילים) את אוסף מילות הקוד.

int alphabetSize – מייצג את גודל האלפבית.

int length – אורך המחרוזת (מילה).

פונקציות המחלקה

public Code(Vector<Word> newWords) – בנאי רגיל המקבל וקטור של מילים.

public Code() - בנאי ברירת מחדל.

public Vector<Word> getWords() - getter לוקטור מילות הקוד.

public int distance() - פונקציית חישוב מרחק הקוד, משתמשת בפונקציה באותו שם של Word.

public double averageDistance()- פונקציית חישוב מרחק הקוד **הממוצע**, משתמשת בפונקציה distance של Word.

public boolean isLinear()– פונקציה הבודקת את ליניאריות הקוד.

public Word bitWiseWordsSum(Word a, Word b) - פונקצית עזר לבדיקת הליניאריות – מחשבת את הסכום (bitwise) של שתי מילים שמתקבלות.

public Vector<Word> allPossibleWords() - פונקציה המחזירה את כל המילים האפשריות באלפבית מסוים, בגודל מחרוזת מסוים. משתמשים בפונקציה זו לטובת יצירת הקוד.

public Code codeSort() - פונקציה הממיינת את הקוד לקסיקוגרפית.

public Code codeSortEndToStart()- פונקציה הממיינת את הקוד לקסיקוגרפית, רק שהמיון מתבצע בסדר ההפוך- כלומר הביט הימני ביותר הוא הMSB.

public String isEqualTo(Code other)- פונקציה הבודקת האם הקוד הנוכחי זהה לקוד המתקבל כפרמטר, ומחזירה מחרוזת עם התוצאה. אם הקוד אינו זהה-מציגה במחרוזת גם אילו מילים לא שוות.

קובץ ECCAnalyzer.java (קבצי המערכת)

משתני מחלקה (משתנים גלובליים)

public JFrame window – חלון המערכת (GUI).

public JPanel panel - פאנל הממשק עליו נמצאים חלק מהרכיבים (כפתורים, תיבת טקסט וכולי).

public JPanel techPanel - פאנל הממשק עליו נמצאים חלק מהרכיבים (כפתורים, תיבת טקסט וכולי).

public JButton bUpload - כפתור העלאת קובץ.

public JButton bClear - כפתור ניקוי חלון הטקסט.

public JButton bDistance - כפתור הפעלת חישוב מרחק הקוד.

public JButton bNotCovered - כפתור הפעלת חישוב המילים הלא מכוסות ע"י ECC-1.

public JButton bUploadSec - כפתור העלאת קובץ שני.

public JButton bCreatBTCode - כפתור יצירת קוד באלגוריתם backtrack.

public JTextArea text – איזור הטקסט בו מופיעים הקודים והמידע עליהם.

public static int *alphabetSize* - גודל האלפבית בקוד בו אנו עובדים.

public static int *StrLength* - אורך המחרוזת בקוד בו אנו עובדים.

public static int *decStrLength* - אורך המחרוזת בקוד בו אנו עובדים כאשר היא בבסיס דצימלי.

public static Code *code* - אובייקט הקוד עליו אנו עובדים.

public static Code *decode* - אובייקט הקוד עליו אנו עובדים, כאשר המילים בבסיס דצימלי.

public static Code *secCode* - אובייקט הקוד השני עליו אנו עובדים.

public static Code *secDecCode* - אובייקט הקוד השני עליו אנו עובדים, כאשר המילים בבסיס דצימלי.

public static boolean *isFirstFileUploaded* - דגל הבודק האם הועלה כבר קובץ קוד אחד או לא.

public static boolean *convDone* - דגל הבודק האם הסתיימה המרה מדצימלי לבסיס האלפבית הנתון.

public static int *resConv* - תוצאת הקלט מהמשתמש לגבי האם יש צורך לבצע המרה לקוד מבסיס דצימלי או לא.

public static int *resSort* - תוצאת הקלט מהמשתמש לגבי האם יש צורך לבצע מיון של הקוד או לא.

public static int *resSortEndToStart* - תוצאת הקלט מהמשתמש לגבי האם יש צורך לבצע מיון לקסיקוגרפי הפוך (ביט ימני הוא MSB) של הקוד או לא.

public static String *fileStr* – מחרוזת המייצגת קוד, אחרי שהוא עובר המרה מקובץ טקסט למחרוזת.

פונקציות המחלקה

public EccAnalyzer ()– בנאי המאתחל את המערכת מבחינת GUI על רכיביה השונים ומסדרם.

public static void main(String[] args)- פונקציית ההרצה הראשית.

public static String readFromFile(File f) - פונקציה המקבלת אובייקט קובץ עם הקוד והופכת אותו למחרוזת.

public static void writeCodeStringToFile(File f, String codeStr)- פונקציה המקבלת מחרוזת ואובייקט קובץ וכותבת קוד שנמצא בצורת מחרוזת לקובץ טקסט.

public static Code StringToCode(String codeStr, int strLength, int alphabetSize) - פונקציה ההופכת מחרוזת המייצגת קוד לאובייקט מסוג קוד.

public static String CodeToString(Code strCode) - פונקציה ההופכת אובייקט מסוג קוד למחרוזת המייצגת קוד.

public static int decWordToInt(Word decWord) – פונקציה המקבלת אובייקט מילה דצימלי והופכת אותו לint.

public static Code decimalCodeToAlphabetSizeCode(Code decimalCode, int alphabetSize) - פונקציה המחזירה קוד בבסיס האלפבית הנתון, כאשר הפרמטר שמתקבל הינו קוד המיוצג דצימליות.

public static String areEquivalent(Code thisCode, Code other) - פונקציה הבודקת עבור שני קודים האם הם שקולים (לא מכסה את כל המקרים, אך כן את המקרים בהם ישנה הזחה ליניארית בין מילות הקוד המתאימות).

public static Code createCodeBT() - פונקציה היוצרת קוד חדש באמצעות שימוש בפונקצית backtrack .

public static Code convertWordArrCode (Vector<int[]> newVec)- פונקצית עזר הממירה וקטור של מערכי מספרים שלמים לאובייקט המייצג קוד.

public static Vector<int[]> creatCodeOfWordArr(int[][] aWords, Vector<int[]> cWords, int minHammingDist)- פונקציה היוצרת קוד כווקטור של מערכים לפי האלפבית המוזן, אורך מחרוזת המוזנת ומרחק המינג שהתקבל כפרמטר, משתמשת בallWords ומהווה פונקצית עזר ל**createCodeBT** .

public static Vector<int[]> backTrack(int[][] aWords, Vector<int[]> cWords, int minHammingDist, int index, int cBound)- פונקצית עזר המממשת את אלגוריתם הbacktrack עבור יצירת הקוד.

public static int distanceCount(int[] arr1, int[] arr2)- פונקצית עזר המחשבת מרחק בין שתי מילים כמערכי מספרים שלמים.

public static void deepCopyWordArray(int[] arr1,int [] arr2)- פונקצית עזר המבצעת העתקה עמוקה בין שתי מילים בייצוג של מערכי שלמים.

public void actionPerformed(ActionEvent e)- פונקצית הGUI אשר מכילה את תסריטי הפעולה השונים בלחיצות הכפתור השונות- מפעילה את הפונקציות הנדרשות בהתאם לאיזה כפתור נלחץ.

תודות

ברצוני להודות ראשית לשני המנחים שלי, ד״ר אלעזר בירנבוים וד״ר אמיר ספיר, שבסבלנות ודייקנות רבה עזרו לי בכל תהליך זה של הפרויקט, הסמינר, המאמר והתזה, לפרופ׳ שלומי דולב מאוניברסיטת בן גוריון בנגב על הרעיון המקורי למחקר, לפרופ׳ דניאל ברנד על דיונים ועזרה רבה במחקר, ולמשפחתי הנפלאה שתמכה בי לאורך כל הדרך בסבלנות רבה.

רשימת מקורות

[1] Biggs, N. (1973), Perfect Codes in Graphs, J. Comb. Theo. B, vol. 15, p. 289—296.
[2] Brouwer, A. Heikki E. Ha'ma'la'inen O., Patric R. J. O' sterga°rd, and Sloane N. J. A., (1998) "Bounds on Mixed Binary/Ternary Codes", IEEE Trans. on Inf. Theo. 44(1).
[3] Cull, P. (1997) "Perfect codes on graphs", Proc. 1997 IEEE Inter. Symp. on Inf. Theo.
[4] Cull, P. and Nelson, I. (1999), Perfect codes, NP-Completeness, and Tower of Hanoi graphs, Bull. Inst. Comb. Appl. 26, p. 13—38.
[5] Forney, G.D., Jr. (2001) " Codes on graphs: normal realizations", IEEE Trans. on Inf. Theo., vol.47, p.520-548.
[6] Klavzar, S. and Milutinović, U. and Petr, C., (2002) 1-Perfect codes in Sierpinski graphs, Bull. Austral. Math. Soc. 66, p. 369—384
[7] Kratochvil, J. (1986), Perfect Codes over Graphs, J. Comb. Theo. B, vol. 40, p. 224—228.
[8] Voloch N., Birnbaum E., Sapir A., (2014) "Generating Error-Correcting Codes based on Tower of Hanoi Configuration Graphs", IEEE 28-th Convention of Electrical and Electronics Eng., Israel.
[9] Wiberg, N., Loeliger, H.-A. and Kotter, R. (1995), "Codes and iterative decoding on general graphs". Eur. Trans. Tel., 6: 513 – 525.

נספח א' – האלגוריתמים למציאת קודים

א. מציאת קודים בסגנון backtracking:

1. קבל את הגדלים של אורך המחרוזת, גודל האלפבית ומרחק ההמינג הרצוי מן המשתמש.
2. אתחל אוסף של כל המילים האפשריות עבור גדלים אלו.
3. אתחל אוסף (דינמי- וקטור) של מילות הקוד.
4. הוסף (באופן טריוויאלי) את המילה 0^n באשר n הוא אורך המחרוזת לאוסף מילות הקוד.
5. המשך לחפש איטרטיבית מילים באוסף כל המילים ועבור כל אחת בדוק האם היא במרחק המינג הרצוי מכל מילות הקוד הנוכחיות באוסף מילות הקוד- הוסף אותה לאוסף.
6. בסיום המעבר על כל המילים שמור את האוסף וחזור לשלב 4 עם המילה הבאה (כלומר בשלב ראשון $1^{n-1}0$ וכן הלאה) למציאת קוד נוסף.
7. בחר מבין כל האוספים את האוסף המקסימלי בגודלו.
8. Backtrack- עבור קוד שנמצא – צור עותק ובו הורד את המילה האחרונה בו (טריוויאלית).
9. חזור למילה האחרונה הנוכחית בקוד (שהייתה לפני האחרונה לפני ההורדה).
10. המשך לבצע חיפוש (חזור לשלב 5).
11. באין מציאת קוד גדול יותר – חזור לשלב 8.

ב. מציאת קודים עם היוריסטיקה מסוימת (לפי הגרפים שצוינו- התאמה לפי סיומות של מחרוזות):

1. קבל קוד שנעשה לפי אלגוריתם א' או בדרך אחרת.
2. בצע מיון לקסיקוגרפי הפוך של מילות הקוד (כלומר הmsb הינו הביט הימני ביותר, השני מימין הוא השני בחשיבותו וכן הלאה).
3. בצע חלוקה של מילות הקוד לקבוצות- לפי סיומת הביט האחרון.
4. בצע תת חלוקה בקבוצות שנוצרו לפי הביט הלפני אחרון וכן הלאה עד לקבלת חלוקה שווה.
5. עבור כל תת קבוצה – שייך לתת-גרף של הגרף הכולל את כל המילים האפשריות- באשר מילות הקוד מהוות קבוצה שלטת על הקודקודים בגרף.

נספח ב' – כיוונים עתידיים

מגרעה של התכנית היא שהאלגוריתם למציאת קודים רץ בזמן ריצה אקספוננציאלי, זאת כיוון שבפעולתו הוא לוקח מילה פוטנציאלית (החל בוקטור ה-0) וממנה מתחיל ליצור קוד באופן איטרטיבי בתחילה, ואחרי מציאת קוד כזה, אם אינו עומד בחסם מס' המילים המבוקש (שמוזן ע"י המשתמש), ממשיך למילה הבאה ($0^{n-1}1$) ומבצע את אותה פעולה בדיוק, כאשר נקודת המוצא היא מילה זו ולא וקטור ה-0 כמו באיטרציה הקודמת. פעולה זו מביאה לתוצאות טובות יותר מבחינת חסמי גודל הקוד, אך יקרה מבחינת סיבוכיות. שיפור אפשרי שניתן לתת לתכנית הוא מציאת אלגוריתם יעיל יותר, אולי עם איזשהו תחכום או בפשרה על מציאת קודים גדולים רק בהסתברות כלשהי.