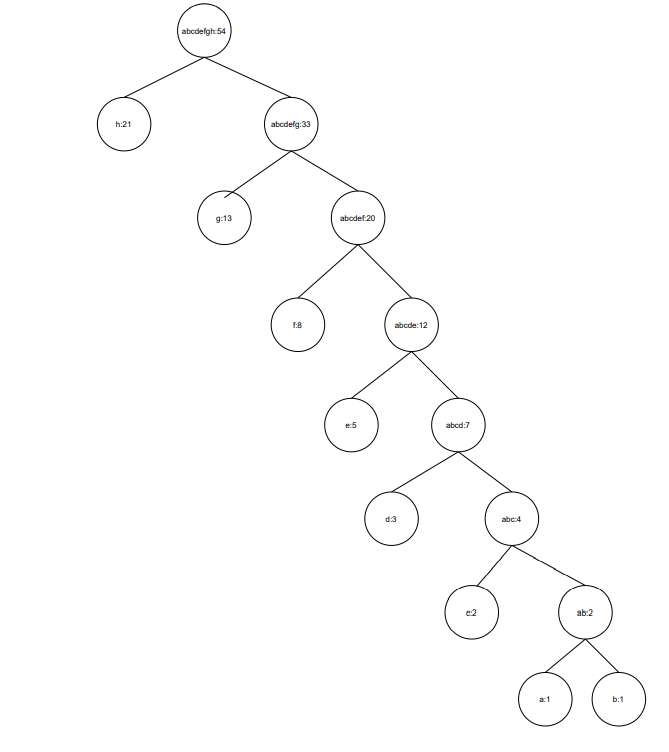
תרגיל בית מס 6- גלי גלר 207704842

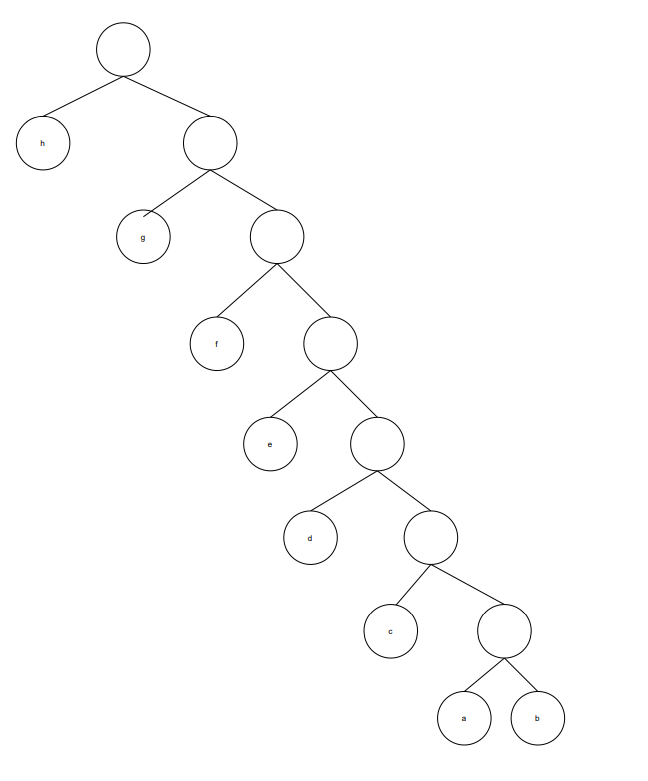
שאלה 2

סעיף א

נרצה למצוא את קוד האפמן עבור הקורפוס: d = {a:1, b:1, c:2, d:3, e:5, f:8, g:13, h:21}

נבנה את עץ האפמן עבור הקורפוס. נסדר את איברי הקורפוס בתור לפי המשקל שלהם, בכל פעם ניצור צומת חדש עם המשקל של חיבור המשקלים של שני הצמתים המינימליים. בשלב האחרון נקבל את העץ:

וממנו נבנה את העץ שממנו נבנה את הקידוד הסופי: כל צעד בין צומת לבן השמאלי שלו יקבל את הערך 0 וכל צעד בין עץ לבן הימני שלו יקבל 1, הקידוד יהיה המסלול בין השורש לעלה של האות



1

1

1

1

1

1

1

0

0

0

0

0

0

0

קיבלנו את הקוד:

a = 1111110

b = 1111111

c = 111110

d= 11110

e = 1110

f = 110

g = 10

h = 0

סעיף ב

נשים לב שלפני שנגיע לאות שמשקלה 3 העץ שניצור תמיד יהיה שווה לצומת שמשקלה מינימלי, ולאחר מכן העץ תמיד ייכנס לתור הקדימויות במקום השני, לכן לפי אלגוריתם בניית עץ האפמן נקבל תמיד עץ מוטה שצורתו היא צורת העץ שקיבלנו בסעיף הקודם ועומקו הוא n-1, כלומר אם נסדר את האותיות לפי משקלן במספרי פיבונאצ'י קידוד האות ה- k כאשר n > k > 1 (כלומר האינדקס רץ מ0 עד n-1) יהיה (n-1)-k פעמים הספרה 1 ואז 0, עבור האותהראשונה בקידוד שלנו יהיו n-2 פעמים 1 ואז 0 ועבור האות השנייה יהיו n-1 פעמים 1

סעיף ג

תשובה: ההפרש בין ל הוא 0.  
נשים לב כי נתון n = 256

"נבנה" את עץ האפמן ונראה שהוא עץ מלא, כלומר כל העלים שלו באותה רמה ולכן כל האותיות מיוצגות ע"י אותו מספר ביטים.

נתון:   
נשים לב כי לכן אם נכתוב את התור של תדירויות התווים כאשר כל תו מיוצג ע"י המשקל שלו בקורפוס (ניתן לעשות זאת כי המשקלים שונים, קיים ביניהם יחס של גדול ממש) נקבל בשלב הראשון:

ובשלב השני נאחד את ו לצומת אחת ונקבל מהנתון כי :

בגלל שמספר האותיות זוגי ובין כל המשקלים יש יחס של גדול ממש נקבל לבסוף שהתור יראה כך:

נגדיר איטרציה כביצוע האלגוריתם עד ש מופיע בסכום שנמצא בראש התור, ונקבל שבכל איטרציה כזו גודל התור יקטן פי 2, ומכך ש256 הוא חזקה של 2 נקבל עץ מלא- כלומר כל העלים יהיו באותה הרמה, ולכן מספר הביטים שיידרשו כדי לייצג כל אות הוא זהה.

שאלה 3

סעיף ב

ננתח את סיבוכיות המקום של triple\_dict במקרה הגרוע ביותר, זהו המקרה בו אין חפיפות, כלומר נשמור במילון N-2 שלשות.

נניח שהקידוד של כל אות בא"ב הוא c קבוע כלשהו, אז כל מפתח במילון יהיה באורך 3c ביטים (לפי תגובת מתרגל בפורום הפיאצה ניתן להניח כי שמירת 3 אותיות היא O(1)). כמו כן, במקרה זה יישמר המספר המקסימלי האפשרי של המפתחות, כלומר נשמור N-2 מפתחות. כמו כן, עבור כל מפתח נשמור רשימה באורך 1 עם האינדקס שבו השלשה מתחילה. האינדקס הוא לכל היותר N, והמספר הבינארי שמייצג אותו יהיה O(logN) ספרות, לכן קיבלנו שסיבוכיות הזיכרון היא **NlogN**

סעיף ג

ראשית, נשים לב שכל הפעולות שמחוץ ללולאת ה for הן O(1): לפני הלולאה מבוצעות הפעולות הבאות: השמת ערך במשתנים, בדיקת type של מחרוזת, השוואת משתנים מסוג int, סלייס בגודל 3 של רשימה (שלוקח O(1) פעולות כי גודל הסלייס הוא 3) וחיפוש של ערך במילון שראינו בהרצאה שהוא O(1) בממוצע.

כעת נרצה לחשב את סיבוכיות זמן הריצה של הלולאות. לולאת הfor היא O(1) בגלל שנתון כי מספר המופעים של כל שלושת תווים חסום ע"י c קבוע כלשהו, כלומר אורך הרשימה match\_indexes הוא לכל היותר c ולכן הלולאה תרוץ מספר קבוע של פעמים. כמו כן הפעולות שנעשות בתוך הלולאה ומחוץ ללולאת הwhile הן O(1)

כתוצאה מכך הסיבוכיות תלויה בלולאת הwhile. לולאת הwhile תרוץ לכל היותר max\_length פעמים, ובתוך הלולאה מתבצעות פעולות שהן O(1), לכן סיבוכיות זמן הריצה הממוצעת של הפונקציה כולה תהיה O(max\_length)

שאלה 4

סעיף א

מרחק Hamming של הקוד מוגדר להיות מרחק הHammig המינימלי בין שתי מילים חוקיות בקוד. נשים לב כי המילים 1111110001 ו 1111111000 הן מילים חוקיות בשיטת Berger, וכן מרחק האמינג הוא סכום הביטים השונים בין שתי המילים, לכן מרחק האמינג יהיה **2**.

סעיף ב

1. ראינו בהרצאה שאם d הוא מרחק האמינג המינימלי של הקוד ניתן לגלות לכל היותר d-1 שגיאות. במקרה שלנו d=2, לכן נוכל לגלות לכל היותר שגיאה אחת לפי נוסחא זו.

נשים לב שאם יש שגיאה אחת אז בהכרח מספר האפסים בהודעה יהיה שונה ממספר האפסים שאמור להיות לפי שלושת הביטים שאנו משתמשים בהם לביקורת, לכן תמיד נוכל לגלות שגיאה אחת.

1. ראינו בהרצאה כי נוכל לתקן עד = = 0 שגיאות, כלומר לא נוכל לתקן שגיאות.

סעיף ג

במצב שבו השגיאה האפשרית היחידה היא 0 שהופך ל1 נוכל לגלות כל כמות של שגיאות: בדאטא עצמו נוכל רק להקטין את מספר האפסים באמצעות הפיכתם ל1, בעוד שנוכל רק להגדיל את מספר הביקורת, שמייצג את כמות האפסים בדאטא, לכן תמיד נוכל למצוא שגיאה.

שאלה 5

סעיף א

ניתן ליצור גנרטור שמייצר את איחוד הקבוצות. רעיון המימוש:

נתונים לנו שני גנרטורים: g1, g2, כל אחד מהם מייצג קבוצה של מספרים טבעיים ובכל גנרטור אין חזרות.

בכל קריאה של next עם גנרטור האיחוד אם g1,g2 שניהם לא מוצו עבור קריאה שמספרה אי זוגי נרצה להחזיר איבר מg1 ועבור קריאה זוגית נרצה להחזיר איבר מ g2, אם אחד הגנרטורים מוצה נחזיר רק איברים מהגנרטור שעוד לא מוצה.

כאשר נבחר גנרטור נרצה להחזיר ערך מתוכו, אך נרצה לוודא שהערך לא הוחזר כבר ע"י הגנרטור השני. נשתמש כאן בזיכרון העזר האינסופי: נקרא לnext ונשמור את הערך שלו במשתנה, נבדוק האם הערך נמצא במשתנה העזר- כלומר הוא כבר הוחזר ויש למצוא ערך חדש. נחזור על פעולה זו עד שנמצא ערך בגנרטור שלא הוחזר כבר, נוסיף אותו לזכרון העזר ונחזיר אותו בעזרת הפקודה yield

סעיף ב

לא ניתן ליצור גנרטור שמייצר חיתוך של שתי קבוצות. נניח בשלילה שכן- הדרך לעשות זאת היא כנראה לעשות next על שני הגנרטורים, לשמור את התוצאות בזכרון האינסופי ואם עשינו next שהביא ערך שכבר קיים- להחזיר אותו, אך נסתכל כעת על שתי הקבוצות הזרות הבאות: קבוצת כל המספרים הטבעיים הזוגיים וקבוצת כל המספרים הטבעיים האי זוגיים, החיתוך הוא ריק והקבוצות אינסופיות, לכן הגנרטור לא ימצא אף איבר שנמצא בחיתוך בזמן סופי. כמו כן, בגלל שהקבוצות שמייצרים הגנרטורים הנתונים יכולות להיות אינסופיות לא ניתן לקבוע בוודאות ששתי הקבוצות זרות, לכן לא נוכל לממש גנרטור כזה.

סעיף ג

ניתן ליצור גנרטור שמייצר את כל המספרים הראשוניים הגדולים מ100.

ראשית, נאתחל משתנה בתוך הגנרטור ל101. בכל קריאה לnext נחפש את ה"מועמד" הבא להיות ראשוני: מספר אי זוגי שספרת האחדות שלו אינה 5, כלומר מספר שספרת האחדות שלו היא 1,3 או 7. נבצע עבורו את בדיקת הראשוניות היעילה יחסית שראינו בהרצאה, באמצעות העדים, ואם הוא ראשוני נחזיר אותו באמצעות yield, אחרת נחפש את ה"מועמד" הבא. בהנחה שיש אינסוף מספרים ראשוניים נמצא עוד מספר ראשוני בזמן סופי.

שאלה 7

סעיף ג: סעיף ב: סעיף א:

