אלגוריתם הכיווץ של
Huffman

גיל כהן
11.2000

מוקדש לדוני

 webmaster@flashoo.co.il

הקדמה

כיווץ תמיד נתפס אצלי כפעולה קוסמית חסרת הסבר, שהרי איך ייתכן שמידע יכול להידחס? עד כמה הוא יכול להידחס? האם ישנו אלגוריתם דחיסה אופטימלי לכל הפורמטים או שלכל פורמט אלגוריתם כיווץ אופטימלי משלו? על השאלות הללו אין אני מתכוון לענות במסמך זה, המסמך הזה בסה"כ הולך להדריך אותך, הקורא, שלב אחר שלב באלגוריתם כיווץ אחד, אלגוריתם הכיווץ של *Huffman*, שהרי איך אפשר לענות על שאלות הנוגעות לכיווץ ללא הכרת אלגוריתם כיווץ אחד לפחות?

אלגוריתם הכיווץ של *Huffman* מנצל את העובדה שברוב הקבצים ישנם תווים המופיעים בתדירות גבוהה יותר מתווים אחרים, טענה זו נכונה במיוחד לגבי קבצי גרפיקה, אנימציה וטקסט. האלגוריתם מחולק לשלושה שלבים:

1. בניית טבלת התדירויות – *Frequency Table*.
2. בניית "עץ האופמן" - *Huffman Tree*.
3. קידוד הקובץ בעזרת עץ האופמן.

טבלת התדירויות - *Frequency Table*

בניית טבלת התדירויות היא השלב הראשון באלגוריתם. הרעיון הוא לסרוק את הקובץ תו אחר תו וליצור טבלה של תו מול מספר הפעמים בו הוא מופיע בקובץ (תדירותו), הטבלה בסופו של תהליך, צריכה להבנות כך שהתו שתדירותו בקובץ הכי גבוהה יהיה האלמנט העליון בטבלה והתו שתדירותו בקובץ הכי נמוכה יהיה האלמנט התחתון.

חשוב להוסיף: אם ישנם מספר תווים בעלי אותה תדירות, הסדר שלהם בטבלה יקבע לפי הסדר שלהם בקובץ, במילים אחרות אם האות א' הופיעה לפני האות ב' ושניהן מופיעות באותה תדירות (לדוגמא: "אבטיח") א' תהיה מעל ב' בטבלה.

מניסיון אישי אני יודע שדוגמא היא ההסבר הברור ביותר אז הבה ננסה לבנות את טבלת התדירויות של המחרוזות: *"this is a test"*, "chiwawa" ו "mishmesh".

אחרי שנסרוק את המחרוזת "this is a test", נראה שהתווים t, s ותו הרווח מופיעים כל אחד מהם שלוש פעמים במחרוזת, התו l מופיע פעמיים ושאר התווים מופיעים כל אחד פעם אחת.

mishmesh

תו	תדירות
m	2
s	2
h	1
i	1
e	1

this is a test

תו	תדירות
t	3
s	3
<space>	3
i	2
h	1
a	1
e	1

chiwawa

תו	תדירות
w	2
a	2
c	1
h	1
i	1

נשים לב, שוב, הטבלאות בנויות מתדירות מקסימלית למינימלית, כאשר לתווים בעלי אותה תדירות סדר הופעה זהה לסדר הופעתם בקובץ.

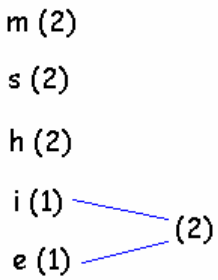
עץ האופמן - Huffman Tree

השלב השני באלגוריתם - עץ האופמן. בשלב זה נשתמש בטבלת התדירויות בכדי ליצור את מה שנקרא "עץ האופמן"

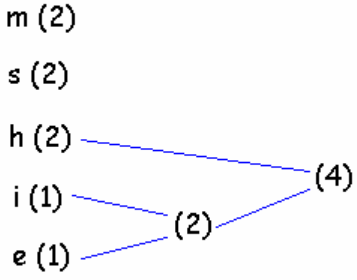
הרעיון הוא לבנות את העץ הפוך, מהעלים לשורש, כאשר העלים הם בעצם האלמנטים המרכיבים את טבלת התדירויות, כל צומת בעץ מכילה ערך מספרי שסיבת קיומו תוסבר מיד. העלים, בנוסף לערך המספרי מכילים גם את התו אותו הם מייצגים, במקרה של העלים הערך המספרי מייצג את תדירות התו בקובץ.

איך נבנה את העץ ולמה משמש הערך המספרי? המטרה היא לחפש ממטה לכיוון מעלה בעץ עבור כל צומת ללא הורה זוג צמתים שסכום ערכיהם המספריים הוא מינימלי, כשזה נמצא אנו יוצרים להם הורה אשר ערכו המספרי הוא סכום ערך המספרים של ילדיו, אצרף דוגמא מאוד מורחבת, נשתמש בטבלת התדירויות של המחרוזת "*mishmesh*":

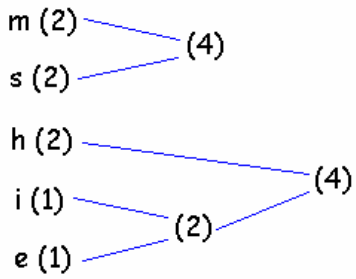
בשלב הראשון אנו מוציאים שהסכום המינימלי נוצר מהערך המספרי של העלים בעל התו I ו E, מכיוון שכך אנו יוצרים לעלים אלו אב שערכו המספרי הוא 2, סכום הערכים המספריים של ילדיו (1+1). עכשיו, כשנחפש את הסכום המינימלי הבא, לא נתייחס לתווים I ו E מכיוון שיש להם אב שהוא אגב כן ייכלל בחיפוש.



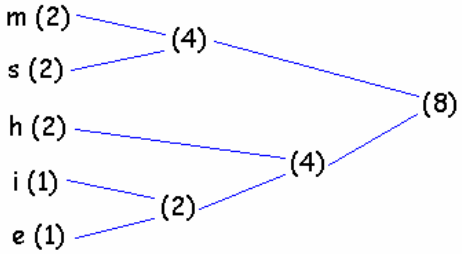
בשלב הבא אנו רואים שהאב הקודם וH יוצרים סכום מינימלי 4 (2+2), אם אתה תוהה למה לא חיברנו את H ו S או כל זוג צמתים אחרים בעלי סכום של 4, זה פשוט מאוד בגלל שאנו באלגוריתם זה סורקים את העץ מלמטה למעלה ומכיוון שכך לצמתים הנמצאים בתחתית העץ יש עדיפות על תווים הנמצאים בראש העץ.



השלב הבא די פשוט, חיבור די ברור של M וS , יש לשים לב שלאב של E וI אין התייחסות בחיפוש זה מכיוון שיש לו אב, במילים אחרות התווים היחידים שאנו סורקים בסריקה הבאה הם M, S והאב של H.



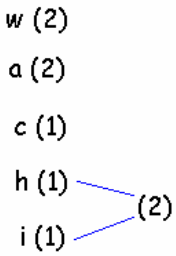
השלב האחרון הוא חיבור שני הצמתים הנותרים....



כפי שאתה רואה, אנו יצרנו את העץ מהעלים לשורש, כאשר לכל צומת חוץ מהשורש יש אב אחד בלבד, ולכל צומת חוץ מהעלים ישנם שני בנים.

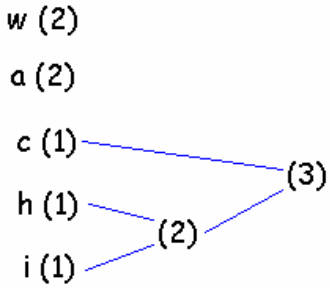
אני די בטוח שהבנת את הפואנטה, אבל דוגמא נוספת לא תהרוג אותך. עכשיו נשתמש בטבלת התדירויות של "*chiwawa*" ונבנה את עץ האופמן בהתאם:

בתחילה אנו מחברים את H ו I שהרי סכומם מינימלי.

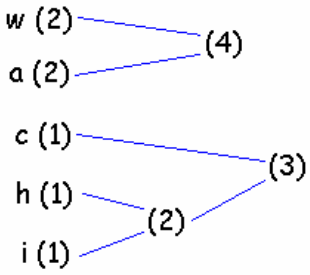


לאחר מכן נחבר את C והאב של H וI. אם אתה שואל את עצמך מדוע לא חיברנו את A וC או אפילו את W וC התשובה לכך פשוטה והוסברה קודם לכן, ישנה עדיפות לצמתים תחתונים מאשר לצמתים עליונים.

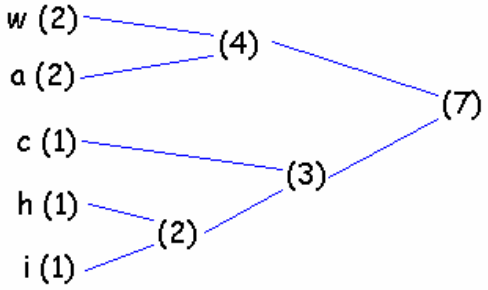
  webmaster@flashoo.co.il



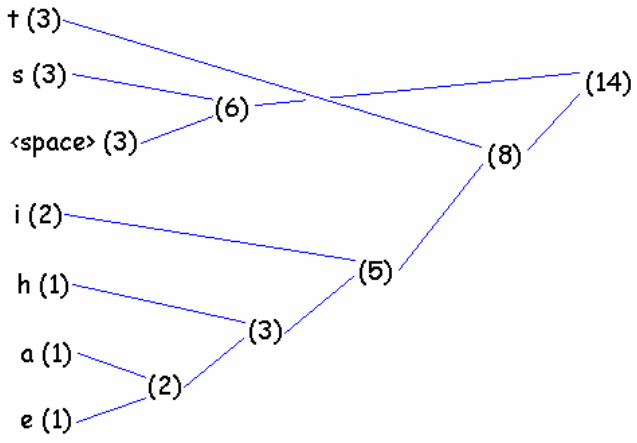
השלב הבא יהיה כמובן לחבר את W וAi שהרי סכומם הוא המינימלי.



והשלב הסופי יהיה לחבר את שני הצמתים הנותרים. ובכך ליצור את השורש...

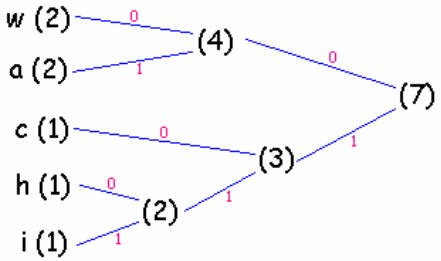


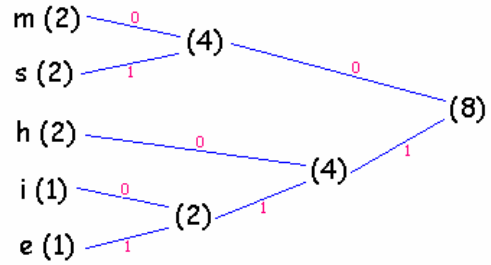
רק בכדי להיות בטוח לגמרי שהבנת את השלב השני (והכי קשה) באלגוריתם בניתי את התמונה הסופית של עץ האופמן של המחרוזת "this is a test":

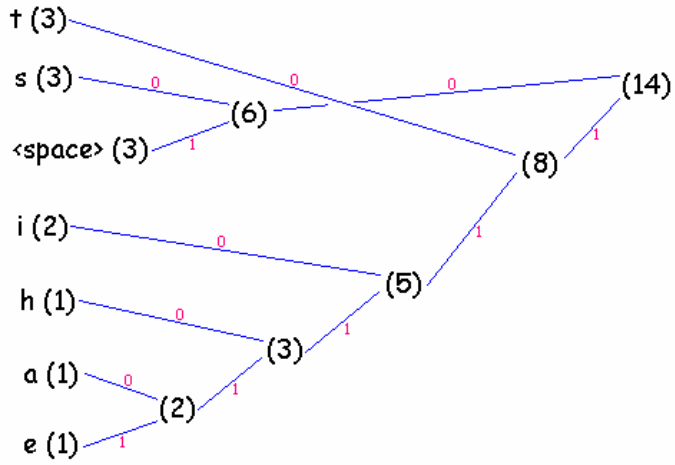


קידוד הקובץ בעזרת עץ האופמן

אם נשים לב נראה ונבין שלכל תו יש נתיב מיוחד מהשורש עד אליו, בדוגמא האחרונה הנתיב מהשורש לתו *l* לדוגמא, הוא שמאל-שמאל-ימין, ושל התו *A*: שמאל-שמאל-שמאל-ימין.
אם נחליף את המושג שמאל בספרה הבינארית '1' ואת המושג ימין בספרה הבינארית '0' נקבל בעצם צירוף בינארי המייצג את התו, לדוגמא הצירוף הבינארי המייצג את התו *A* בדוגמא האחרונה הוא: 11110, והצירוף הבינארי של התו *S* הוא 00.
חשוב לציין שהייצוג של ימין כ '1' בינארי ושמאל כ '0' בינארי הוא שרירותי לחלוטין, במילים אחרות אין כל סיבה לא לקבוע את ייצוג הימין כ '0' בינארי וייצוג השמאל כ '1' בינארי. הדרישה היא כמובן להיות עקבי עם הבחירה. הנה כל שלושת העצים שבנינו בתוספת של אפסים ואחדות לוגים:







הבה נבצע פעולה זאת על המחרוזת *"chiwawa"*:

w = 00
a = 01
c = 10
h = 110
i = 111

c	h	i	w	a	w	a
10	110	111	00	01	00	01

נזכור שכל תו בקוד *ASCII* מיוצג ע"י 8 בתים ומכן שהמחרוזת "*chiwawa*" שאורכה 7 תווים התכווצה לה לשני תווים.

הבה נקודד גם את המחרוזת "*mishmesh*", במחרוזת זו, מספר הבתים לאחר כיווץ הוא 18, זאת אומרת 2 בייטים ו2 בתים, מכיוון שרשימה לקובץ נעשית בבייטים אנו צריכים להשלים את שני הבתים האחרונים לבייט, זאת אומרת להוסיף להם עוד שישה בתים חסרי משמעות, בתים אלו נקראים בתים משלימים.

לסיום חלק זה, נקודד את המחרוזת האחרונה "*this is a test*":

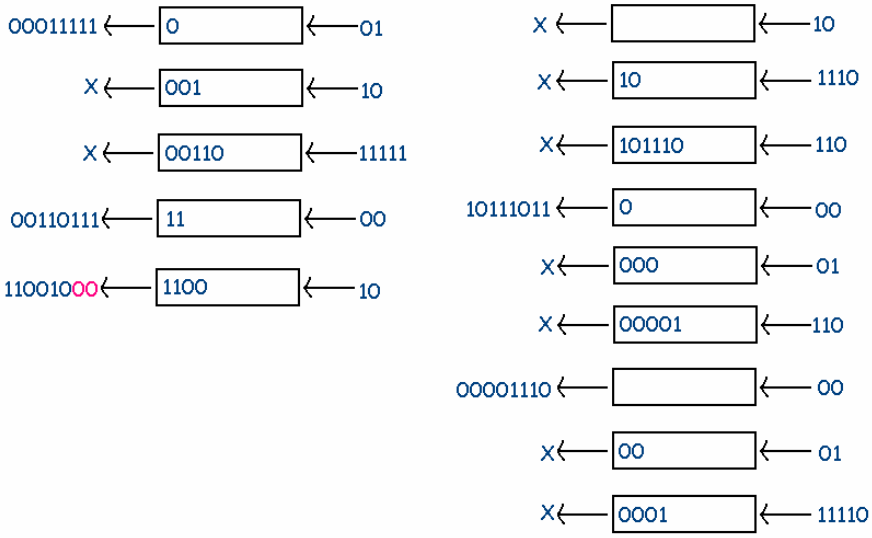
t = 10
s = 00
<space> = 01
i = 110
h = 1110
a = 11110
e = 11111

t	h	i	s	-	i	s	-	a	-	t	e	s	t
10	1110	110	00	01	110	00	01	11110	01	10	11111	00	10

פורמט הקובץ המכווץ

בתיאוריה בכדי לקודד את הקובץ אנו צריכים לסרוק את הקובץ תו אחר תו ולכל תו מקובץ המקור לכתוב אל קובץ המטרה את הצירוף הבינארי המאפיין אותו.

במציאות זה קצת יותר מורכב, מכיוון שהעבודה עם קבצים מתבצעת עם בייטים ולא עם בתים אין אנו יכולים לכתוב פחות משמונה בתים (בייט) בכל פעם, מכיוון שכך אנו צריכים איזושהי מחסנית שבכל קריאה של תו מקובץ המקור נוסיף אליה את הצירוף הבינארי המתאים, כל פעם שהמחסנית עוברת או מגיעה לגודל של שמונה בתים היא רושמת לקובץ המטרה את שמונה הבתים הוותיקים, שאר הבתים יתקדמו במקום השמונה הוותיקים.. כך חוזר חלילה..
במקרה שאנו מסיימים לקרוא את קובץ המקור ונשאר לנו בכל זאת כמה בתים במחסנית אנו נוציא את הבתים הללו כפלט עם בתים משלימים,
הנה דוגמא וויזואלית לתהליך, בהתאם למחרוזת "*this is a test*"



נשים לב לאפסים הורודים בסוף התהליך, אפסים אלו מציינים בתים משלימים לבייט.

בנוסף לקידוד הקובץ יש להוסיף לקובץ המכווץ את טבלת התדירויות ואת מספר הבתים המשלימים לשם מתן אפשרות פתיחת הקובץ מאוחר יותר.

אני ממליץ לרשום בתחילת הקובץ המכווץ איזושהי חתימה כך שתוכנת הפתיחה תדע בוודאות שזהו באמת קובץ שכווץ בתוכנה שלך, כמו כן מומלץ לרשום את הגרסה של תוכנת הכיווץ שבנית, שהרי אם בעתיד תשפר את האלגוריתם או את פורמט הקובץ, תוכל לפתוח קבצים שכווצו באלגוריתם או בפורמט הישן.

המלצה נוספת היא לרשום את גודל קובץ המקור, בקובץ המטרה משתי סיבות: אינדיקציה לתוכנת הפתיחה שאכן הקובץ נפתח לגודלו המקורי, חישוב אחוז הכיווץ, שהרי אם תרצה לבנות תוכנה הנותנת מידע על הקובץ המכווץ זהו ללא ספק אלמנט חשוב ומעניין.

הבה לכן נסכם בצורה וויזואלית את פורמט הקובץ שלנו:

חתימה	גירסה	גודל הקובץ המקורי	מספר המשלימים	טבלת התדירויות	קידוד הקובץ

webmaster@flashoo.co.il

כמובן שאין זהו הפורמט היחיד או הטוב ביותר, אתה מוזמן להוסיף פרמטרים רבים כגון: תאריך הכיווץ, סיסמא, וכל פרמטר אחר העולה על דעתך, רק חשוב לזכור לכתוב את טבלת התדירויות, מספר המשלימים ומן הסתם.. קידוד הקובץ.

הבנת הכיווץ

הבנת הכיווץ היא לא דבר שנרכש ברגע שלומדים איך לכווץ, יש הבדל עצום בין יישומו של רעיון לבין הבנתו, סעיף זה בא לעזור לך להבין למה הדברים פועלים כמו שהם פועלים באלגוריתם, במילים אחרות, באיזו נקודה בדיוק מתרחש הקסם.

ובכן, בתחילת המדריך ציינתי שהאלגוריתם של האופמן משתמש בעובדה שישנם תווים המופיעים בתדירות גבוהה יותר מתווים אחרים בקובץ, ובכן די קל לראות כיצד האלגוריתם מבצע זאת, אם נסתכל לדוגמא על הקידוד של המחרוזת "*this is a test*"

t = 10

s = 00

<space> = 01

i = 110

h = 1110

a = 11110

e = 11111

t	h	i	s	-	i	s	-	a	-	t	e	s	t
10	1110	110	00	01	110	00	01	11110	01	10	11111	00	10

נראה שהצירוף הבינארי של אותיות בעלי תדירות גבוהה יותר הוא קצר יותר, לדוגמא ל *T* יש צירוף בינארי באורך 2 בתים לעומת *E* שלה צירוף בינארי באורך 5 בתים, מכיוון שכך, תווים המופיעים יותר פעמים ייתפסו בכל מופע פחות בתים.

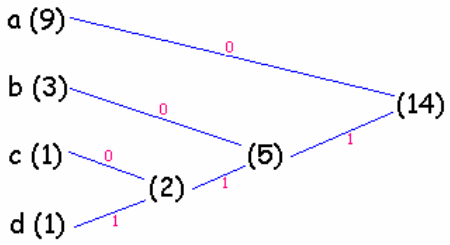
עדיין נשאר להסביר מדוע לתווים בעלי תדירות גבוהה יותר יש צירוף בינארי קצר יותר, ובכן.. מצב זה נוצר בתהליך יצירת עץ האופמן. מכיוון שאנו מחברים את הסכום המינימלי בכל פעם בתהליך בניית העף, העלים בעלי הערך המספרי הגבוהה, מתחברים רק בהמשך התהליך ולא בתחילתו ומכאן שהקישור שלהם אל השורש קצר יותר ומכיוון שכך אורך הצירוף הבינארי המייצג אותם, קצר יותר.

הבה ניקח דוגמא מאוד מוגזמת לתיאור העניין, אנו נבצע את תהליך הכיווץ כולו לגבי המחרוזת: "*aacaabaabbdaaa*"

תחילה נבנה את טבלת התדירויות:

תו	תדירות
a	9
b	3
c	1
d	1

לאחר מכן נבנה את עץ האופמן:



ולבסוף נקודד את המחרוזת:

a = 0
b = 10
c = 110
d = 111

a a c a a b a a b b d a a a
0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 1 1 1 0 0 0

נשים לב ש *A* מיוצגת על ידי צירוף בינארי באורך של בית אחד לעומת *D* המאופיינת על ידי 3 בתים, ההסבר הוא בבניית עץ האופמן, מכיוון ש *A* מתחבר בסופו של תהליך עם הסב של *C* ו *D* אז הוא קרוב יותר אל השורש ולכן חיבורו קצר יותר, לעומת תווים הרחוקים מהשורש כמו *C* ו *D*.

אם נסתכל מבחינה מעמיקה יותר על העניין ונצא מנקודת הנחה שישנם באמת רק 256 תווי ASCII נראה שהצירוף הבינארי הארוך ביותר משורש העץ לעלה עלול להגיע עד ל255 בתים, במילים אחרות במקום כיווץ נמצא את עצמנו מגדילים את גודל הקובץ, כעיקרון מצב זה יכול לקרות, אבל אסור לנו לשכוח שככל שהעלה רחוק מהשורש תדירותו בקובץ נמוכה יותר ועל כן צירופו הבינארי ייכתב פחות פעמים בעוד תווים בעלי תדירות גבוהה מיוצגים ע"י צירופים קצרים יותר ומופיעים יותר פעמים, במילים אחרות, אם באמת מצב כזה יקרה, רוב הסיכויים הם שהצירוף הבינארי הארוך של העלה המרוחק יתקזז עם הצירופים הבינאריים הקצרים של העלים הקרובים יותר לשורש.

הפסקה האחרונה מבהירה לנו שככל שישנם פחות סוגים שונים של תווים בקובץ המקור הכיווץ יהיה יעיל יותר, מבט מעמיק יותר ינחה אותנו לעבר העובדה שבמצב בו ישנם פחות או שמונה תווים שונים בקובץ, במילים אחרות אורך טבלת התדירויות קטנה או שווה לשמונה, אין סיכוי שהקובץ לא יתכווץ, שהרי אפילו הצירוף הבינארי הארוך ביותר האפשרי יגיע רק לשמונה בייתים בינאריים, שזהו בעצם אורך כל תו ASCII.

פתיחת הכיווץ

פתיחת הכיווץ מתבצעת בהתאם לפורמט הכיווץ שלנו, אנו נניח שבקובץ המכווץ יש את טבלת התדירויות את מספר הבתים המשלימים ואת הקובץ המקודד.

בתיאוריה בכדי לפתוח את הקובץ אנו צריכים לבצע את הפעולות הבאות: נבנה בעזרת טבלת התדירויות את עץ האופמן, ונעמוד על שורש העץ. לאחר מכן נקרא בית מהקובץ, אם הבית הנקרא הוא '1' נקפוץ לנו לכיוון הבן הימני אחרת ('0') נקפוץ לביקור אצל הבן השמאלי, אם הגענו לעלה (צומת ללא ילדים) נכתוב את התו שהעלה מייצג ונחזור לשורש העץ... כך חוזר חלילה.

כמובן שבמציאות התהליך קצת מורכב יותר שהרי אין אפשרות לקרוא בית מקובץ, מכיוון שכך יש להשתמש בטריק הדומה לטריק המחסנית שתיארתי בהרחבה בסעיף "פורמט הקובץ המכווץ".

בנוסף, בל לנו לשכוח את השימוש בבתים המשלימים. בבואנו לכתוב את התו האחרון, אנו צריכים לבדוק כמה מהבתים שבתו זה הם חסרי משמעות שנועדו בסה"כ להשלים את התו, לאחר מכן נפתח רק את שאר הבתים. לדוגמא אם מספר הבתים המשלימים הוא 3, אנו נתייחס רק לחמשת הבתים הראשונים בתו האחרון.

סיכום

אני מקווה שלא השארתי לך, הקורא, שאלות ללא תשובות בכל הנוגע לאלגוריתם הכיווץ של האופמן. אני ממליץ בחום לבנות תוכנת כיווץ נחמדה המתבססת על אלגוריתם זה,

אם אתה מתעניין באיכות האלגוריתם מבחינת זמן כיווץ ואחוז כיווץ, אל דאגה, אתה יכול לסמוך על האלגוריתם. אני אישית בניתי תוכנת כיווץ לאלגוריתם בשפת *C++*, ואני גאה לציין שללא אופטימיזציה, התוכנה כיווצה מגה-בייט בפחות מחמש שניות, כמעט לחצי מגודלו, בעוד תוכנת הכיווץ *PKUnzip* כיווצה את אותו קובץ בקצת פחות משתי שניות, לשישים וחמש אחוז מגודלו.

יחסית לתוכנה פרטית מול מסחרית אני חייב לציין שעלה חיוך לא קטן על פניי.