```
/**
 * Assignment 1
 * Submitted by:
 * Student 1.Dean Moravia ID# 302491741
 * Student 2.Hana Razilov ID# 31178472
 * Student 3.Aviya Shimon ID# 302813217
 */
```

דחיסת נתונים, עבדות הגשה ראשונה- קוד הופמן

שאלה 1:

דחיסת הקובץ:

מנייה, ובניית עץ הופמן:

בשאלה זאת, החלטנו להשתמש במערך מנייה על מנת לספור את כמות המופעים של כל byte במערך הבתים של הקובץ. להמיר אותו מ-128 to 127 ל0 to 255, וככה לדעת את מספר המופעים של כל אות.

לאחר מכן השתמשנו במחלקה Node המכילה בתוכה פרטים על הבית ומצביעים לNode שמאלי וימני (עץ), וPriority Queue ליצירת ערימה מינימום על פי התדירויות ששמורות בתוך הNode. הערימה מורכבת מבתים להם היה מופע אחד לפחות.

על ידי שליפת שני האיברים המינימליים, חיבורם תחת Node חדש שהתדירות שלו היא סכום תדירויות האלמנטים תחתיו, והוספתו לערימה עד אשר קיים בערימה אלמנט אחד שהוא למעשה ראש העץ. בסופו של דבר, אנו מקבלים ראש עץ הופמן, ומספר האותיות המיוצגות בו.

ייצוג הבתים:

לאחר שיש לנו את עץ הופמן, בנינו מחלקה הנקראת Path המכילה ערך בוליאני המייצג ביט 1/0, ומצביע לאובייקט Path נוסף (רשימה מקושרת). בנוסף בנינו מחלקה PathHead היורשת את Path ומוסיפה גם את ערך הבית המיוצג, ואת אורך ייצוגו.

אנו עושים מערך מנייה של PathHead מ0-255, במטרה שכאשר נעבור על מערך הבתים של הקובץ, נוכל לקודד את הבתים באופן יעיל יותר.

אנו עוברים על עץ הופמן, כאשר המשך לכיוון ימין נמשיך עם אובייקט הPathHead הנוכחי, וכאשר ממשיכים לשמאל מעתיקים את האובייקט PathHead הנוכחי לאובייקט PathHead חדש.

כאשר מגיעים לעלה בעץ, אנו נשים את הPathHead שלנו במערך מנייה על פי הערך של הבית בNode אליה הגענו.

כתיבת הקובץ הדחוס:

לאחר שיש לנו את מערך PathHead שהוא מנייה, אנו יכולים באופן יעיל, להיעזר במערך המנייה של ספירת הבתים בקובץ, על מנת לחשב מראש את גודל הקובץ העתידי, ולהכין מערך הבתים בגודל הדרוש עבור הקובץ העתידי.

איך בנוי הקובץ הדחוס:

10101010	00000101	101	1010101	01011	10001001010
מספר בתים מיוצגים	מספר ביטים עבור אורך ייצוג	שלושה ביטים המייצגים מספר בין 0-7 עבור ביטים מיותרים בסוף הקובץ	ערך הבית	גודל ייצוג המורכב מ5 ביטים כמו בדוגמא הנוכחית	ייצוג ביטים עבור הבית
			החלק הזה נכתב עבור כל בית שיש לייצג		

לאחר שכתבנו את ייצוג הבתים בתחילת הקובץ אנו עוברים על מערך הבתים של קובץ המקור ובעזרת מערך המנייה של הPathHead אנו רושמים את הביטים במערך הבתים של הקובץ הדחוס עבור כל בית.

אופן כתיבת ביטים:

הביטים נכתבים בעזרת מחלקה CompressedFile המכילה בתוכה שלושה אלמנטים. הראשון הוא מערך בתים בוא כל בית מתחיל בערך 128- ושני משתנים האחד הוא הבית בוא אנו נמצאים והשני הוא הביט בתוך הבית בו אנו נמצאים, ונמצאת במעקב אחריהם.

בתוך המחלקה יש את הפונקציה `addToByte` המוסיפה סכום לבית הנוכחי.

`insertBitsIntoCompressedFile`: הפונקציה משתמשת בערך הביט בבית הנוכחי ומוסיפה לו את הסכום, על פי החזקה הרלוונטית של 2, כך שיהיה בבית 0 או 1 בביט הרלוונטי.

לאחר שנכתב הביט האחרון מחזירים את מערך הבתים של CompressedFile וממנו יוצרים את הקובץ המכווץ.

חילוץ הקובץ:

קריאה מהקובץ:

יצרנו מחלקה DecompressedFile המכילה את הקובץ הדחוס, ומעקב אחרי הבית הנוכחי והביט בתוך הבית. למחלקה יש את הפונקציה `getBitAndMove` המחסרת את הערך של הביט הנוכחי בבית, ומחזירה ערך בוליאני המסמל 1 או 0. ככה אפשר לקרוא את הקובץ בביטים.

שיחזור ייצוג הבתים:

שחזרנו את הייצוג על ידי המחלקה DecompressedCodeBook המכילה בתוכה את ערך הבית ואובייקט Path בתוכו שמים את הביטים שמייצגים אותו.

בניית עץ הופמן מהייצוג:

על ידי מעבר על המערך בתוך הDecompressedCodeBook אנו בונים את עץ הופמן על פי הביטים בדרך. כלומר, כל עוד לא הגענו לסוף הדרך נבדוק האם קיים תת עץ בכיוון אליו אנו צריכים ללכת. אם קיים נלך לכיוון, אם לא, ניצור תת עץ חדש ונלך אליו. נעשה זאת עד אשר נגיע לביט האחרון של הייצוג, ונשים בתת עץ הנוכחי את ערך הבית אותו הוא מייצג.

שיחזור הקובץ:

בעזרת רשימה מקושרת של בתים, אנו נשתמש בעץ הופמן המשוחזר ונעבור ביט ביט בקובץ הדחוס. כאשר נגיע לעלה בעץ נשמור את ערך הבית אותו הוא מייצג ברשימה המקושרת, ונתחיל מעבר חדש בעץ עבור הבית הבא. נספור את כמות הבתים ששחזרנו.

בעזרת כמות הבתים המדויקת ששחזרנו נבנה מערך סטטי של בתים, נעביר אליו את הבתים מהרשימה המקושרת.

ממערך הבתים נשחזר את הקובץ.

שאלה 2:

לא הספקנו לסיים את שאלה 2.

מה שרצינו לעשות הוא למצוא שילובים של שני בתים בקובץ שהוספת ייצוגם המשולב לעץ, ייתן קובץ דחוס קטן יותר.

הדרך בה ניסינו לעשות זאת היא שימוש בחישוב גודל קובץ עתידי על מנת למצוא את השילוב הבא הטוב ביותר, וככה להתקרב לגודל קובץ הקטן ביותר שניתן לעשות על ידי שילוב מספר בתים כפולים. כלומר, להתחיל מגודל הקובץ ללא שימוש בשילוב בתים, לאחר מכן לעבור על כל השילובים הקיימים בקובץ, ולחשב איך הכנסתם לעץ תשפיע על הבתים אותם הם מחליפים, למצוא את השילוב הנוכחי הכי יעיל והכנסתו לעץ. לחזור על התהליך עד שלא נמצא שילוב יעיל יותר, או לאחר כמות מסוימת של שילובים.

הקוד עצמו אינו עבר סידור, וניתן לראות את הדרך בה היה אמור לעבוד עבור שם הקובץ הראשון במערך של input_names וניתן למצוא אותו על ידי חיפוש "//question two attempt here"

בבדיקות וניסויים שערכנו היה לדרך פוטנציאל, במיוחד עבור שילובים שאינם אותיות באופן ספציפי. למשל עבור הקובץ OnTheOrigin.txt מצאנו שהבתים בעליה ערך 10 ו13 (newline, carriage return) תמיד הופיעו בסדר זה, ולכן היה האלגוריתם תאורתית מוריד את ייצוגם של 10 ו13 לחלוטין ומכניס במקומו ערך משולב בעל אותה תדירות.