<u>שאלה 1 – 40 נק'</u>

א. (15 נק') יש להוכיח באופן פורמלי ומדויק ש-prefix code המיוצג על ידי עץ T הינו אופטימלי אם T בינו עץ בינארי מלא.

ב. (10 נק') נתון קובץ המכיל 8 תווים ששכיחותם כלהלן:

Char	a	b	c	d	e	f	g	h
frequency	1	1	2	3	5	8	13	21

רישמו את קוד הופמן (Huffman code) האופטימלי לכל אחד מהתווים וציירו את העץ המתאים.

הניחו שסדר צמתים בעלי אב משותף הוא כזה שהימני הוא בעל השכיחות הגבוהה יותר. הניחו שסיבית הקוד לצומת הימני היא 1 ולשמאלי 0.

ג. (15 נק') ידוע שהשכיחות הנ"ל הינה סידרת פיבונאצ'י המוגדרת להלן:

$$F(0)=1, \qquad F(1)=1, \qquad F(k)=F(k-1)+F(k-2), \quad k\geq 2$$

בהנחות של סעיף 2 הוכיחו שבמקרה של n תווים ששכיחותם היא n מספרי פיבונצ'י הראשונים, Huffman code האופטימלי עבור התו ה-$k<n$ בסדר השכיחויות היורד הוא $1^{k-1}0$ (שרשור של $k-1$ 1-ים ולבסוף 0), והקוד עבור התו ששכיחותו הקטנה ביותר הוא 1^{n-1} (שרשור של $n-1$ 1-ים).

לצורך ההוכחה העזרו בשוויון $\sum_{i=0}^{k-1}F(i)=F(k+1)-1$ לכל $k\geq 2$ (אין צורך להוכיחו).

<u>פתרון</u>

א. בסעיף זה יש להראות שאם T הוא אופטימלי אז הוא בהכרח עץ בינארי מלא.

יהא T prefix code tree ונניח בשלילה שהוא בינארי לא מלא. לפי ההנחה בהכרח יש קיים צומת $v\in T$ בעל בן בודד $x\in T$. נראה שקיים עץ אחר שבו מספר הסיביות הממוצע לתו קטן יותר. מספר הסיביות הממוצע לתו באלפבית C בעץ T הינו:

$$B(T)=\sum_{c\in C} c.freq\times d_T(c)$$

יהא T' העץ המתקבל ממחיקת v והחלפתו ב-x. יהא μ עלה שהוא צאצא של x (הן ב-T והן ב-T') המתאים לקוד של התו $m\in C$. מספר הסיביות הממוצע ל תו באלפאבית C בעץ T' מקיים:

$$(1)\ B(T')=\sum_{c\in C} c.freq\times d_{T'}(c)=\sum_{c\in C\setminus\{m\}} c.freq\times d_{T'}(c)+m.freq\times d_{T'}(m)$$

כמו כן

$$(2)\ \sum_{c\in C\setminus\{m\}} c.freq\times d_{T'}(c)\leq \sum_{c\in C\setminus\{m\}} c.freq\times d_T(c)$$

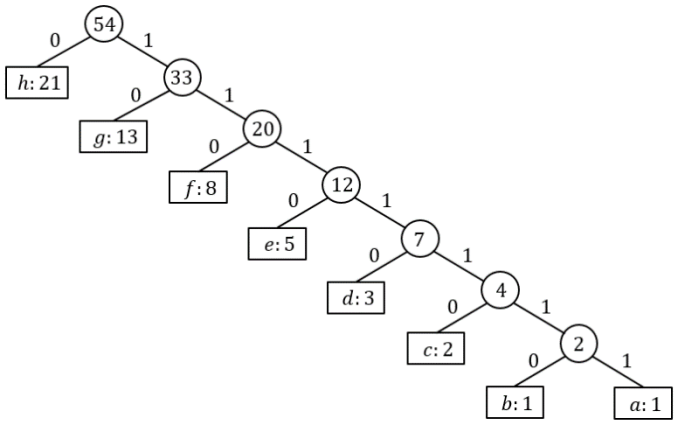
וגם

$$(3)\ \ m.freq\times d_{T'}(m)=m.freq\times (d_T(m)-1)<m.freq\times d_T(m)$$

הצבה של (2) ו (3) ב (1) נותנת $B(T')<B(T)$ כנדרש.

הראינו שקיים עץ T' שבו מספר הסיביות הממוצע לתו קטן יותר מאשר ב-T. מכך נוכל להסיק שההנחה שלנו שהעץ T, שהינו עץ בינארי לא מלא, מהווה ייצוג אופטימלי, בהכרח אינה נכונה. ממילא הוכנו את הדרוש שאם T הוא אופטימלי אז הוא בהכרח עץ בינארי מלא.

ב.



$$1111111 \rightarrow a$$
$$1111110 \rightarrow b$$
$$111110 \rightarrow c$$
$$11110 \rightarrow d$$
$$1110 \rightarrow e$$
$$110 \rightarrow f$$
$$10 \rightarrow g$$
$$0 \rightarrow h$$

ג. הוכחה שהשכיחות בכל צומת פנימי של העץ הנבנה ע"י הקוד לקידוד $k+1$ התוים בעלי השכיחויות הקטנות ביותר הינו סכום השכיחויות $\sum_{i=0}^{k} F(i)$, כלומר $k+1$ האיברים הראשונים בסדרת פיבונצ'י.

אם אכן הדבר כך אזי נראה שתמיד נוסף עלה לעץ שבו מצוי התו בעל השכיחות ע"פ הסדרה, והדבר אומר שהענף השמאלי מהאב אילו מקודד ע"י 0 והענף הימני מהאב לצומת הפנימי של העץ מקודד ע" 1.

ההוכחה באינדוקציה על k.

האלגוריתם ליצירת Huffman code מתחיל בתווים בעלי השכיחות הנמוכה ביותר (בשניהם השכיחות 1) ועל כן שכיחות האב המשותף היא סכומם $F(2) = F(1) + F(0)$, אכן סכום השכיחויות באופן טריוויאלי.

נניח באינדוקציה שבצעד ה-k נוצר צומת פנימי חדש שהשכיחות שלו מקיימת $\sum_{i=0}^{k-1} F(i)$.

מהגדרת סדרת פיבונצ'י ומהנתון $\sum_{i=0}^{k-1} F(i) = F(k+1) - 1$ לכל $k \geq 2$ מתקיים

$$F(k) = F(k-1) + F(k-2) \leq \sum_{i=0}^{k-1} F(i) = F(k+1) - 1 < F(k+1)$$

ולכן בצעד $k+1$ האלגוריתם ימזג את הצומת הפנימי שנוצר בצעד k ששכיחותו $\sum_{i=0}^{k-1} F(i)$ עם העלה ששכיחותו $F(k)$ ושכיחות הצומת המשותף אכן $\sum_{i=0}^{k} F(i)$.

יש לשים לב שאי השוויון ה "סנדוויץ" הנ"ל הכרחי משום שאלמלא הסכום $\sum_{i=0}^{k-1} F(i)$ היה חסום משני צדדיו ע"י $F(k)$ ו $F(k+1)$ היה נוצר מיזוג של שני האחרונים ומבנה ה "שרוך" של העץ לא היה נשמר.

<u>שאלה 2 – 40 נק'</u>

<u>הערה: אין קשר בין הסעיפים.</u>

א. (25 נק') נתון גרף $G(V,E,W)$ באשר W הוא משקל צלעותיו. יהא T MST של G. תהה $e(u,v) \in E$ שמשקלה w אבל $e(u,v) \notin T$. בעת מקטינים את משקלה של e ל-$w' < w$ ומשאירים את משקל שאר הצלעות ללא שינוי. נסמן ב-W' את המשקלות החדשים. הציגו אלגוריתם למציאת MST של $G(V,E,W')$ שזמן הריצה שלו חסום מלמעלה ע"י $O(|V|)$.

על הפתרון לפרט באופן מסודר את כל הצעדים של האלגוריתם ולהראות את נכונותו ואת זמן החישוב הנדרש. ניתן להניח ללא הגבלת הכלליות שגם ב-W וגם ב-W' כל המשקלים שונים.

2. (15 נק') בחברת "העץ הנדיב" נדרש לכתוב אלגוריתם לחישוב MST בגרף G. אלגוריתמאי מדופלם העובד בחברה הציע את אלגוריתם divide and conquer רקורסיבי COOL_MST למציאת MST הפועל באופן הבא:

אלגוריתם $T = \mathbf{COOL_MST(G)}$:

1. מחלקים את הקודקודים לשני חלקים שווים (עד כדי הבדל של קדקד בודד) $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$.
2. מגדירים את תתי הגרפים המתאימים G_1 ו-G_2 הנפרשים ע"י V_1 ו-V_2 בהתאמה ואת החתך $[V_1, V_2]$ המכיל את הקשתות המחברות את שני החלקים.
3. $T_1 = \text{COOL_MST}(G_1)$.
4. $T_2 = \text{COOL_MST}(G_2)$.
5. תהה $e \in [V_1, V_2]$ בעלת משקל מינימלי.
6. $T = T_1 \cup T_2 \cup e$.

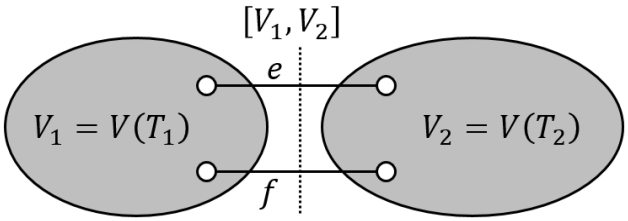
אשרו (ע"י הוכחה) את נכונות האלגוריתם, או הפריכו את נכונותו ע"י דוגמא נגדית.
תשובה סתמית נכון / לא נכון לא תחשב.

<u>פתרון</u>

א.

1. T קשיר ומשתמשים באלגוריתם BFS למציאת המסלול (היחיד) ב T המחבר את הקודקודים u ל-v . זמן הריצה הנדרש הוא $O(|V| + |E|)$, ומאחר וב T מתקיים $|E| = |V| - 1$ אז סה"כ $O(|V|)$
2. מוסיפים את $e(u, v)$ ל T, ומקבלים $T \cup \{e\}$ שכבר איננו עץ אלא גרף בעל מעגל C כך ש $e \in C$. ההוספה לוקחת $O(1)$ או $O(|V|)$ זמן, תלוי במבנה הנתונים שנבחר .
3. מטיילים על הקשתות של C וזורקים את הקשת $f \in C$ בעלת המשקל הגדול ביותר ומקבלים חזרה עץ פורש. זמן בצוע שלב זה חסום ע"י $O(|V|)$.

נטען שהעץ T' המתקבל ב-3 הוא MST ב-$G(V, E, W')$.
<u>אם $e \neq f$:</u>



סילוק e ו-f מותיר שני תת עצים זרים T_1 ו T_2 שכל אחד מהם MST ב V_1 ו V_2 בהתאמה כמתואר בציור, אחרת T איננו MST ב $G(V, E, W)$.
פרט ל-e ו-f אף קשת אחרת בחתך $[V_1, V_2]$ איננה חלק מ-MST של $G(V, E, W')$, אחרת T איננו MST ב-$G(V, E, W)$. ומאחר ומשקל f הוא מקסימלי הרי שאכן $e \in T'$.
<u>אחרת, $e = f$:</u>
כלומר הקשת שהוספנו ל T נזרקה בשלב 3 הרי שממילא חזרנו ל-T.

ב. האלגוריתם $T = \text{COOL_MST}(G)$ שגוי בעליל. נוכיח על ידי דוגמה נגדית:
נתבונן בגרף הבנוי באופן הבא: $G_1(V_1, E_1)$ גרף שלם בעל קשתות שמשקלן 100 ו-$G_2(V_2, E_2)$ גרף שלם בעל קשתות שמשקלן 1. $[V_1, V_2]$ מחבר כל קדקד ב V_1 לכל הקדקדים ב V_2 ולהיפך ע"י קשתות שמשקלן 1.
$T = \text{COOL_MST(G)}$ מכיל תת עץ שפורש את קדקדי G_1 ע"י קשתות מ E_1 ומשקלן 100, בעוד ש-MST תקין איננו מכיל אף אחת מהקשתות של E_1 אלא רק קשתות שמשקלן 1.

<u>שאלה 3 – 40 נק'</u>
נתון קטע הקוד הבא:

```
def f(A, n, target):
```

```
for i in range(n): # loop n times (from 0 to n-1)
    for j in range(i, n, 1): # loop n-i times (from i to n-1)
        if A[i] + A[j] == target:
            return True
return False
```

א. (5 נק') כתבו בקצרה מה הפונקציה `f` עושה ומה סיבוכיות זמן הריצה וסיבוכיות המקום שלה.

בסעיפים הבאים עליכם להציע מימושים חלופיים לפונקציה הנתונה, תחת הדרישות של כל סעיף. אין צורך לרשום קוד מפורט אך יש חובה לפרט את כל השלבים במלל + פסאודו קוד.

הערה לשני הסעיפים: אין הגבלה על שימוש בזיכרון עבור משתנים בעלי סיבוכיות $\mathcal{O}(1)$.

ב. (20 נק') עליכם להציע מימוש **יעיל יותר מבחינת ריצה**. מגבלת זיכרון: לרשותכם מערך עזר נוסף מאותחל באפסים בגודל $\boldsymbol{\mathcal{O}(n)}$. אסור להשתמש בזיכרון נוסף מלבד מערך העזר (ומשתנים בעלי סיבוכיות $\mathcal{O}(1)$).

ג. (15 נק') עליכם להציע מימוש הפועל בזמן ריצה של $\boldsymbol{\mathcal{O}(n)}$. מגבלת זיכרון: לרשותכם מערך עזר נוסף מאותחל באפסים בגודל $\boldsymbol{\mathcal{O}(target)}$. אסור להשתמש בזיכרון נוסף מלבד מערך העזר (ומשתנים בעלי סיבוכיות $\mathcal{O}(1)$).

<u>פתרון</u>

א. פונקציה שבודקת אם יש שני איברים במערך שסכומם שווה ל-target.
(ניתן להוסיף שאם יש איבר שערכו הוא בדיוק חצי מה-target אז גם יוחזר True, כיוון שבכל פעם אנחנו גם בודקים אם $A[i] + A[i] = target$. לא ירדו נקודות למי שלא התייחס לכך).
ב. ניתן למיין את המערך ואז להחזיק מצביע לאיבר הראשון ומצביע לאיבר האחרון, לבדוק את סכומם ולפי היחס שלו ל-target לקדם את המצביעים:

```
def f1(A, n, target):
    A.sort()
    r_idx = 0
    l_idx = n - 1
    while r_idx < l_idx:
        if A[r_idx] + A[l_idx] == target:
            return True
        elif A[r_idx] + A[l_idx] > target:
            l_idx -= 1
        else:
            r_idx += 1
    return False
```

ג. הערה: כוונת השאלה הייתה להתייחס למקרה בו המערך מחזיק מספרים שלמים ואי שליליים בלבד. הנחה זו נשמטה בטעות מטופס הבחינה, אך מי שפתר נכון רק עבור מקרה זה יקבל את מלוא הנקודות.
בונים מערך עזר B בגודל target ומאתחלים אותו באפסים (למעשה מערך עזר זה נתון לשימושינו). כעת עוברים על המערך, אם האיבר גדול או שווה ל-target, ממשיכים הלאה. אחרת, הולכים למיקום ה-$A[i]$ במערך B ושמים שם 1.
בנוסף, הולכים למיקום ה-$(target - A[i])$ – אם יש שם 1 מחזירים True, אחרת ממשיכים הלאה.
עבור המקרה של מספרים אי שליליים, הקוד יראה כך:

```
def f2(A, n, target):

    B = []
    for i in range(target):
        B.append(0)
    for i in range(n):
        if A[i] >= target:
            continue
        B[A[i] - 1] += 1
        if B[target - A[i]] > 0:
            return True
    return False
```

עבור המקרה הכללי הפתרון מעט יותר מורכב, אך עדיין עומד תחת תנאי השאלה. פתרון זה הוא הרחבה פשוטה של המקרה הקודם, ובעיקר יש לשים לב להשמה במערך העזר והתייחסות לאינדקסים השונים. אם אכן יש מספרים שליליים, מערך העזר הוא באורך של $(target + 2 \times abs(\min(A)) = O(target)$. הפתרון המלא הינו:

```python
def f3(A, n, target):
    B = []
    m = min(A)
    if m >= 0:
        return f2(A, n, target)
    for i in range(target + 2 * abs(m)):
        B.append(0)
    for i in range(n):
        # A[i] > target + abs(m)
        if A[i] > target + abs(m):
            continue
        # 0 < A[i] < target
        elif 0 < A[i] < target:
            B[A[i] - 1 + abs(m)] += 1
            if B[target - A[i] + abs(m) - 1] > 0:
                return True
        # A[i] < 0
        elif A[i] < 0:
            B[abs(m) - abs(A[i])] += 1
            if B[target - abs(A[i]) + 2 * abs(m) - 1] > 0:
                return True
        # target < A[i] < target + abs(m)
        else:
            B[A[i] - 1 + abs(m)] += 1
            if B[target - A[i] - abs(m)] > 0:
                return True
    return False
```