**מבוא מורחב למעדי המחשב - תרגיל בית 6**

1. ג. המקרה הגרוע ביותר מבחינת סיבוכיות הוא כשאין חזרה של תת תמונה, מפני שאז התכנית צריכה לרוץ על כל המטריצה.

ד. הפונקציה עלולה להחזיר True כאשר אין חזרה של תת תמונה, כיוון שהיא מתבססת רק על סכומי הפיסקלים. כדי להתגבר על כך, אפשר להתחשב גם **במיקומי** הפיקסלים בתת התמונה באמצעות פונקציית fingerprint שבדומה ל-karp-rabin במחרוזות (כפי שהודגם בכיתה), תחבר כל ערך של פיקסל כפול 2^16 בחזקת המיקום שלו (מהסוף) ולסכומם תעשה מודולו מספר ראשוני גדול כלשהו. נקבל fingerprint ייחודי יותר, וכמעט ולא יהיו התאמות שגויות.

2. א. נוכיח באינדוקציה:

נראה נכונות עבור n=4: => – נכון.

נניח נכונות עבור ונוכיח עבור , כלומר נניח ונוכיח **.**

ידוע כי בסדרת פיבונאצ'י, אי השוויון הבא מתקיים כיוון שהיא סדרה עולה: .

כעת, מחיבור משוואה זו עם משוואת ההנחה נקבל: =>

*וקיבלנו בעזרת האינדוקציה שאי השוויון מתקיים לכל .*

*ב. בגלל המשפט שהוכחנו בסעיף א', שאומר שלכל איבר (מ-n=4), סכום האיברים שלפניו, גדול ממנו וקטן מהאיבר הבא אחריו, נוצר מצב שביצירת עץ הופמן, בכל שלב נוצרת "קומה" נוספת עם עלה אחד, כיוון ששני האיברים שסכומם הוא הקטן ביותר הם תמיד האיבר הנוכחי בסדרה וסכום האיברים שלפניו (בשלושת האיברים הראשונים, השכיחויות הן 1,1 ו-2 והעץ נבנה בצורה זהה). לפיכך, נקבל שאורך הקידוד הקצר ביותר הוא 1 ושאורך הקידוד הארוך ביותר הוא* n-1.

*ג. נבחן את העץ המתקבל: נחבר את שני האיברים הקטנים ביותר, , כעת זהו האיבר הגדול ביותר, כיוון ש- ומכאן- כי . כעת שני האיברים הקטנים ביותר הם נחבר אותם והם יהפכו לאיבר הגדול ביותר שכן סכומם גדול מסכומם של שני המספרים שלפניהם. כך נמשיך על כל זוג איברים עוקבים, ונקבל 64 זוגות עוקבים. שלב זה יהיה זהה לשלב ההתחלתי אך עם 64 איברים:*

*(אי השוויון: נשמר כיוון ש- ו- קרובים יותר משני האיברים הקיצוניים).*

*כך אפשר להמשיך עד סוף העץ, ומה שיתקבל בסוף זה עץ בינארי שלם, שבו כל העלים באותו העומק, לכן מספר הביטים הדרושים לקידוד כל תו זהה וההפרש בין ו- הוא 0.*

*3. א. ייתכן מצב כזה. למשל: 'lxtyqwabcdefghijkljklxtyqw'*

*עבור l=3 נקבל:*

*['l', 'x', 't', 'y', 'q', 'w', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', [3, 3], [20, 5]]*

*עבור l=4 נקבל:*

*['l', 'x', 't', 'y', 'q', 'w', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'j', 'k', [20, 6]]*

*ניתן לראות שעבור l=3 נקבל תוצאה זהה מבחינת "עלות", חוץ מכך שבמקום שני התווים j ו-k ישנה מחרוזת שמייצגת אינדקס ואורך מילה. כפי שנאמר בכיתה, אורכה בכתיב הבינארי הוא 18 ביטים לעומת 16 ביטים במקרה של שני תווים ולכן "עלותה" גבוהה יותר. לכן קיבלנו דחיסה יעילה יותר עבור l=4.*

*ב. i. Huffman יהיה יותר יעיל כאן, מפני שהאות a תופיע בערך במחצית מהטקסט ולכן קידודה יהיה באורך ביט אחד. עבור שאר האותיות, קידודן יהיה באורך 6 ביטים בערך, מפני שייווצר (פרט ל-a) עץ שמתקרב להיות עץ שלם, כיוון ששכיחויותיהן זהות. עבור LZ, כל אות תיוצג על ידי 8 ביטים אם אין חזרה, וכיוון שהטקסט הוא אקראי ולא מעשה ידי אדם, החזרות בו יהיו מעטות, ונקבל דחיסה פחות טובה. דוגמאות הרצה:*

>>> s=genString(100000)

<<<len ( str\_to\_bin (s ))

700000

>>> huff\_tree = build\_huffman\_tree ( char\_count (s ))

>>> huff\_code = generate\_code ( huff\_tree )

>>> C= compress (s , huff\_code )

>>> len(C)

335280

>>> inter = lz77\_compress2 ( s)

>>> binform=inter\_to\_bin(inter)

>>> len(binform)

516884

>>> 335280/700000

**0.47897142857142855**

>>> 516884/700000

**0.7384057142857143**

>>> s=genString(100000)

<<<len ( str\_to\_bin (s ))

700000

>>> huff\_tree = build\_huffman\_tree ( char\_count (s ))

>>> huff\_code = generate\_code ( huff\_tree )

>>> C= compress (s , huff\_code )

>>> len(C)

334513

>>> inter = lz77\_compress2 ( s)

>>> binform=inter\_to\_bin(inter)

>>> len(binform)

534547

>>> 334513/700000

**0.4778757142857143**

>>> 534547/700000

**0.7636385714285714**

1. במקרה זה, המחרוזת שתתקבל תכיל ברובה המכריע את התו ‘a’, וב-LZ יתקבלו הרבה מאוד חזרות שלו והרבה חזרות "מלאות" באורך 31. אורך כל חזרה כזו בביטים הינו 18, מול 32 במקרה של Huffman. לכן התשובה תשתנה ו-LZ יהיה יעיל יותר.

ג. ייצוג הביניים יראה כך:

**['0', '1', [2, n-2]]**

יחס הדחיסה יהיה O(log(n)/n)