נושאים בביואינפורמטיקה

עדן זיו 208932079, עמית שרגא 208993493

חלק ב'

<u>תיאור האלגוריתם:</u>

<u>שלב א'-</u> הקלט: נריץ את האלגוריתם מסעיף א' עם q=1 על מנת לקבל את כל התוצאות בהם הcog מופיע ונשתמש בתוצאות הממוינות לחלק זה. נשים לב שהתוצאות הן במבנה נתונים של מילון כך שה-key הוא אורך מילה וה-value הוא מערך ממוין של מילים בסדר יורד על פי שכיחויות כאשר כל מילה מופיעה לפחות פעם אחת.

<u>שלב ב'-</u> חיפוש קבוצות המילים שבהן יש לכל היותר k הכנסות:
נשים לב שבחלק א' מצאנו מילים באורכים שונים שבהן הcog מופיע. נוכל לצמצם את החיפוש ולחפש מילים רק בתוצאות של חלק א'. נשלח לפונקציה *k_instances* את הערך k, את הקבוצות הממוינות שמצאנו בחלק א' ו- *counter* ריק. עבור כל מילה (W) שהתקבלה בחלק א', נחפש את המופעים שלה רק בקבוצות המילים באורך $|W| + i$ כך ש - $1 \leq i \leq k$ (קבוצות המילים נמצאו ומוינו בחלק א').

<u>שלב ג' –</u> חיפוש המילה W בתוך קבוצת מילים מסוימת: מ-*k_instances* נקרא לפונקציה *is_find_word_k* ונשלח אליה את המילה W ואת קבוצת המילים בגודל הרלוונטי, נעבור על כל המילים בקבוצה ועבור כל מילה נחפש האם W היא *substring* שלה. בגלל שהקבוצות מכילות רק מילים באורך יחיד וספציפי (שתואם את הדרישות), כשנמצא מילה המכילה בתוכה את W נוכל לספור אותה כהופעה של W. נעדכן את הcounter עבור המילה. נשים לב שההכנסות יכולות להתבצע רק באמצע רצף המילה ולכן נבצע תחילה בדיקה של השוואה עבור המיקום הראשון והאחרון בין W לבין המילה מהקבוצה שעוברים עליה.

<u>שלב ד'-</u> מיון התוצאות + החזרת הפלט: נאחד את הcounter מסעיף א' (0 הכנסות) עם הcounter מסעיף ב' (1 עד k הכנסות) בפונקציה update_counter2. נשלח הcounter לפונקציה שתעדכן בה רק את המילים בהם ההופעות הן מעל q . בסוף נבצע מיון לcounter המעודכן באופן דומה למיון שבוצע בחלק א'- קבוצות על פי אורך מילה עולה, כך שכל קבוצה מסודרת בסדר יורד לפי כמות הופעות.

<u>ניתוח זמן ריצה ומקום בזיכרון של האלגוריתם:</u>

- $|text| = m \cdot n$, כך שm מייצג את מספר השורות בטקסט ו-n מייצג את אורך השורה.
- def k_instances(sorted_groups, output, k, counter2):
    1. for word in output:
    2. len_word = len(word)
    3. sum_k = 1
    4. check_key = len_word + sum_k
    5. while sum_k <= k and check_key in sorted_groups.keys():
        a. group = sorted_groups.get(check_key)
        b. find_word = is_find_word_k(word, group)
        c. counter2[word] = counter2[word] + find_word
        d. sum_k += 1
        e. check_key += 1

ניתוח זמן ריצה: מעבר על כל המילים מחלק א' כאשר עבור כל מילה נרצה לבדוק במילון sorted_groups את קבוצת המילים באורך גדול ב-1 ממנה עד שנגיע במקסימום לקבוצת המילים באורך המילה+K. סה"כ זמן ריצה: כ $\underbrace{O(m \cdot n)}_{|output|} \cdot (\underbrace{k}_{\substack{loop\ line\ 5- \\ count}} \cdot \underbrace{O(m \cdot n)}_{\text{is\_find\_word\_k()}}) = O(k \cdot m^2 \cdot n^2)$

סיבוכיות מקום: בפונקציה השתמשנו בפרמטרים שנשלחו מפונקציית ה *Main* ללא הקצאת מקום נוספת.

- def is_find_word_k(word, group):
    1. total_sum = 0
    2. for arr in group:
    3. if word[0] == arr[0] and word[len(word) - 1] == arr[len(arr) - 1]:
        a. j = 1
        b. for i in range(1, len(arr)):
        c. if word[j] == arr[i]:
            i. j += 1
        d. if j == len(word): # so we arrive to the last word in the arr, and we check before it. so we done
            i. total_sum += 1
            ii. break
    4. return total_sum

ניתוח זמן ריצה: מעבר על כל איבר בקבוצה group (מילים באורך ספציפי קבוע) כאשר על כל איבר נעבור במקסימום פעם אחת ונבצע השוואה למילה word (זמן קבוע). לכן מקסימום זמן הריצה הוא כגודל ה data - כמות המילים שיכולות להיות בקבוצה - כלומר $O(m \cdot n)$

סיבוכיות מקום: בפונקציה השתמשנו בפרמטרים שנשלחו מפונקציית ה *Main* ללא הקצאת מקום נוספת.

- def update_counter2(counter1, counter2):
    1. for word in counter2:
        a. counter2[word] += counter1[word]

ניתוח זמן ריצה: מעבר על counter1 כלומר סה"כ $O(m \cdot n)$

סיבוכיות מקום: בפונקציה השתמשנו בפרמטרים שנשלחו מפונקציית ה *Main* ללא הקצאת מקום נוספת.

- def bigger_than_q2(q, counter):
    1. new_counter = Counter*()*
    2. for word in counter*:*
    3. count = counter[word]
    4. if counter[word] >= q:
    5. new_counter[word] = count
    6. return new_counter

ניתוח זמן ריצה: מעבר על counter2 כלומר סה"כ $O(m \cdot n)$

סיבוכיות מקום: בפונקציה הגדרנו את *counter* שהחזרנו בסוף לפונקציית ה-*Main*. סה"כ: $O(m \cdot n)$

- Main

```
...
if k == 0:    # solution from part A
new_counter = bigger_than_q2(q, counter1)
output = []
sorted_groups_k = sort_output(new_counter, output)
print(sorted_groups)

else:

counter2 = Counter()
k_instances(sorted_groups, output, k, counter2)
update_counter2(counter1, counter2)
new_counter = bigger_than_q2(q, counter2)
output = []
sorted_groups_k = sort_output(new_counter, output)
print(sorted_groups_k)
```

<u>ניתוח זמן ריצה:</u> ננתח את זמן הריצה של פונקציית main בהתבסס על זמן הריצה מהאלגוריתם של חלק א'. כל הפעולות הן פעולות בזמן ריצה של $O(1)$ (שליחה לפונקציה, הדפסה, שינוי ערך של מערך). את זמן הריצה של כל פונקציה ניתחנו למעלה. בחלק זה ביצענו שינוי קטן נוסף כך שהפרמטר l נשלח על ידי המשתמש ולכן הוא גם חלק מהזמן ריצה בחלק א'. <u>זמן הריצה הכולל של חלק זה הוא</u>:

$$\underbrace{O(l\cdot m\cdot n^2)}_{part\ A:\ find\_cog\ +\ bigger\_than\_q} + \underbrace{O(k\cdot m^2\cdot n^2)}_{k\_instances} + \underbrace{O(m\cdot n)}_{update\_counter2} + \underbrace{O(m\cdot n\cdot log(m\cdot n))}_{sort\_output}$$
$$= O(mn\cdot(l\cdot n + k\cdot m\cdot n)) = O(|text|\cdot(l\cdot|line| + k\cdot|text|))$$

<u>סיבוכיות מקום:</u> בחלק זה הוספנו מקום בזיכרון עבור המשתנה counter2. משתנה זה, בדומה לcounter1 תופס בזיכרון מקום של $O(m\cdot n)$.

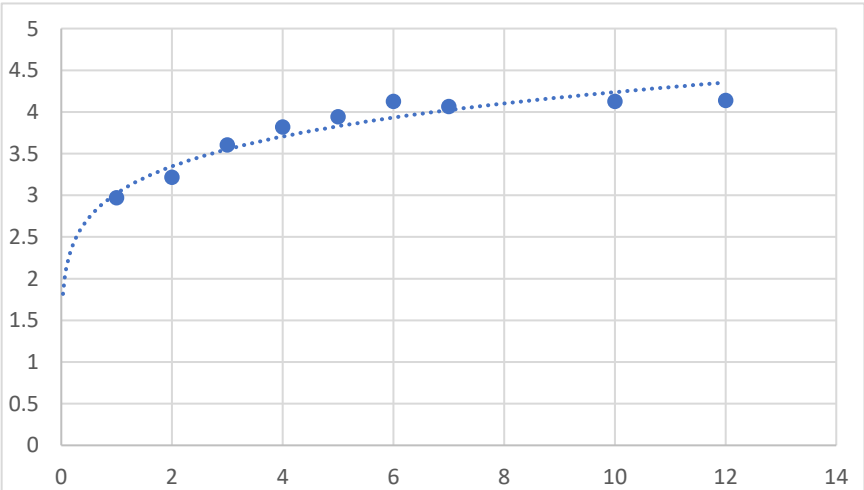
לסיכום, הרחבת הבעיה כך שניתן להכניס עד k אותיות לתבנית של מילה גרמה לזמן ריצה ארוך יותר. זמן הריצה של חלק א' היה $O(m\cdot n^2) = O(|text|\cdot|line|)$. בעוד שזמן הריצה של חלק ב' הוא $O(|text|\cdot(l\cdot|line| + k\cdot|text|))$. כלומר, בחלק א' זמן הריצה היה תלוי באופן ליניארי בגודל הטקסט וכעת תלוי בריבוע גודל הטקסט.

לגבי סיבוכיות מקום – בחלק א' השתמשנו ב$O(2\cdot|text|) = O(|text|)$ מקום בזיכרון. בחלק ב' הוספנו counter 2 ב -$O(|text|)$ מקום, לכן סה"כ השתמשנו ב- $O(4\cdot|text|) = O(|text|)$ מקום. לכן, סיבוכיות המקום לא השתנתה.

<u>תלות של K על זמן הריצה</u>:

הרצנו את התוכנית עם l=10,q=20 ושינינו רק את k.

K	time (part2)
0	
1	2.9710006
2	3.2164298
3	3.605381
4	3.8198955
5	3.941349
6	4.1275692
7	4.0660156
10	4.1275219
12	4.13768181



לפי הגרף ניתן לראות שככל שK גדל - זמן הריצה עולה. עבור ערכי K גדולים צריך לבדוק יותר קבוצות אורכים של מילים, לבצע יותר השוואות ולכן זמן הריצה במגמת עלייה.

עם זאת, השינוי בזמן הריצה לא משמעותי וככל שערכי k גדלים השיפוע של הגרף נעשה פחות תלול. ככל ש-k גדול יותר, מאפשרים יותר הכנסות, וצריך לחפש מילים באורך $k + |W|$. גודלן של קבוצות אורכים גדולים יהיו קטנות משמעותית מקבוצות אורכים קצרים (לדוגמא, בקבוצת המילים שאורכן 8 יהיו פחות מילים מאשר בקבוצת המילים שאורכן 2). לכן, ככל שנעלה את K נצטרך לחפש ביותר קבוצות אורכים אך במקביל כל קבוצה תכיל פחות מילים, ועל כן ניתן לראות מיתון בשיפוע הגרף. נצפה שנגיע לנקודה בה הגרף יתיישר כמעט (מגמה שניתן לזהות בגרף). נובע מכך שאין עוד קבוצות לחפש בהן את המילים שלנו וכל חיפוש של K מסוים מוכל בK לפניו. לכן הגרף לא ירד אלא יתייצב.

<u>השערת תפקיד הcog שבחרנו: cog0586</u>

הרצף (0285, 0777, 0586) הוא בין הרצפים השכיחים ביותר שמצאנו. הרצף המדויק (K=0) מופיע ב-104 אורגניזמים שונים. ככל שנעלה את ערכי K נקבל כי הרצף (0285, 0777, 0586) שכיח אף יותר ונמצא במספר גדול יותר של אורגניזמים.

על מנת לפענח את תפקיד cog0586 נתבונן בcog השכנים:

<u>Cog0285 –</u> משמש כסובסטרט שמעביר קבוצות כימיות בין האנזימים שיכולים להקשר אליו. בנוסף, מעורב בתהליך המטבוליזם. מקודד בקבוצת הגנים FPGS. גנים אלו מקודדים ליצירת אנזים שנקרא folypolyglutamate synthetase . תפקידו העיקרי הוא שמירה על ריכוז folypolyglutamate תקין ויציב בציטופלזמה ובמיטוכונדריה.
<u>Cog0777 –</u> דרוש לשינוע חומצות שומן וליצירת ליפיד מורכב שיאוחסן באופן זמני. מעורב בתהליך המטבוליזם. הינו חלק מקבוצת גנים המקודדים לאנזים שנקרא Acetyl-CoA carboxylase (ACC) . תפקידו העיקרי הוא לווסת את המטבוליזם של חומצות השומן.

שני הcog'ים בעלי מאפייני תפקיד יחסית דומים – העברת מולקולות ותפקוד כחלק מוויסות ריכוז חומרים שונים בתא. השערתנו היא ש- cog0586 אחראי על וויסות חומר נוסף הקשור לתהליך המטבוליזם או שקשור ביצירת אחד האנזימים שהזכרנו מעלה.

נשער שהכנסות נוספות העלו את שכיחות הרצף כחלק מתהליך אבולוציוני. מצד אחד, יכול להיות שבחלוף השנים קרו מוטציות לאורגניזם אב (עם הרצף המקורי) שהביאו ליצירת אורגניזמים שונים עם רצפים מעט שונים (אבולוציה מתבדרת). מצד שני, יכול להיות שבאורגניזמים שונים ובלתי תלויים נוצרו עם הזמן מנגנונים דומים (כי כנראה הם יעילים). אך בגלל שנוצרו באורגניזמים שונים ללא תלות אז הם לא זהים לחלוטין, ולכן כאשר נאפשר הכנסות נמצא שיותר אורגניזמים מכילים את הרצף המקורי (אבולוציה מתכנסת).