בעיית Longest Common Subsequence - LCS:

בעיית תת מחרוזת משותפת ארוכה ביותר

יהיו X ו- Y שתי מחרוזות מעל א"ב Σ מאורך m ו- n בהתאמה.

מהו אורך תת-המחרוזת המשותפת הארוכה ביותר של X ו- Y?

מהי תת-המחרוזת המשותפת הארוכה ביותר של X ו- Y?

1. **אלגוריתם חיפוש חמדני**

תיאור האלגוריתם: נעבור על המחרוזת X , נתחיל לחפש את המופע של האות הראשונה במחרוזת Y. אם מצאנו, נמשיך לאות הבאה ב-X ונחפש אותה החל מאותו מקום בו עצרנו ב-Y, עד שנגיע לסוף של Y.

דוגמה 1:

X = abcbdab,
Y = bdcaba
Gready(X, Y) = aba
Gready(Y, X) = bdab
Max(Gready(X, Y), Gready(Y, X)) = { aba, bdab} = bdab

דוגמה 2:

X = abca,
Y = adcbc
Gready(X, Y) = abc
Gready(Y, X) = ac
Max(Gready(X, Y), Gready(Y, X)) = { abc, ac} = abc

דוגמה 3:

X = acccccccb,
Y = bcccccccca
Gready(X, Y) = a
Gready(Y, X) = b
Max(Gready(X, Y), Gready(Y, X)) = { a, b} = a or b

למרות ו-LCS(X, Y) = ccccccc

נכונות האלגוריתם: אלגוריתם חמדני לא נותן פתרון אופטימאלי. השיטה לא מחזירה את התשובה הנכונה תמיד כי לא תמיד האיבר הראשון שנמצא הוא חלק מהסדרה .בנוסף ,לפעמים כדאי לוותר על מספר איברים כדי לקחת אחרים טובים יותר.

סיבוכיות של אלגוריתם חמדני היא O(m*n)

```
String greedy(String X, String Y)
        ans = "", start = 0, index = 0, i = 0
        m = X.length(), n = Y.length()
        while (i<m && index<n)
                index = Y.indexOf(X.charAt(i), start)
                if (index != -1)
```

```
                ans = ans + X.charAt(i)
                start = index + 1
            end-if
            i++
        end-while
        return ans
end-greedy
```

2. אלגוריתם חיפוש חמדני משופר

רעיון: לייצר מערך מונים לתווים. הדבר ייעל את זמן הריצה.

תיאור האלגוריתם: לחפש במחרוזת אחת (יותר ארוכה בין שתיהן) רק את האותיות שקיימות במחרוזת השנייה (יותר קצרה במספר התווים בין שתיהן).

0<=LCS(X, Y)<=min(|X|,|Y|)

בוחרים בסדרה הקצרה משתי הסדרות הנתונות נניח בסדרה X.

יוצרים מערך עזר של מספר המופעים כל אות ב-X.

עוברים על כל אות ב-Y, אם אות ב- Y מופיעה ב-X, מקטינים ב-1 את מספר הופעות האות במערך עזר. מעדכנים מיקום בשתי המחרוזות.

דוגמה:

X = accccccb,
Y = bcccccca

X	a	b	c
help	1	1	7

אות ראשונה ב-Y: b. אות קיימת ב-X במיקום האחרון. מעדכנים מערך help: help[b]=0. למרות ועדיין לא עברנו על כל תווים ב-Y, ב-X כבר סיימנו.

נכונות האלגוריתם: אלגוריתם חמדני משופר עדיין לא נותן פתרון אופטימאלי.

סיבוכיות של אלגוריתם חמדני משופר היא O(m+n) + O(min(m,n)).

```
String greedyWithHelp(String X, String Y)
        help[26], m = X.length(), n = Y.length()
        for i=0; to m //O(min(m,n))
            place = X.charAt(i)-'a'
            help[place]++
        end-for
        ans = "", start = 0, index = 0, i = 0
        while (i<n && index<m) //O(m+n)
            place = Y.charAt(i) - 'a'
            if (help[place] > 0)
                index = X.indexOf(Y.charAt(i), start)
                if (index != -1)
                    ans = ans + Y.charAt(i)
                    start = index+1
                    help[place]--
                else help[place] = 0
                end-if
            i++
```

```
        end-while
        return ans
end-greedyWithHelp
```

3. <u>אלגוריתם חיפוש שלם (brute-force search or exhaustive search)</u>

רעיון: יצרת כל תתי המחרוזות האפשריים בעזרת מספרים בינאריים.

תיאור האלגוריתם: לקחת כל תתי-מחוזות של X וכל תתי-מחוזות של Y ולחפש מחרוזת משותפת ארוכה ביותר.

מספר תתי-מחרוזות של מחרוזת בגודל n כולל מחרוזת ריקה הוא 2^n.

אלגוריתם לבניית כל תתי-מחרוזות של מחרוזת נתונה מבוסס על מתג חשמלי: אם אות קיימת ערכה 1, אחרת, 0. נשמור כל המחרוזות בייצוג בינארי מ-0 עד 1...111 במערך עזר.

דוגמה:

X = abc, $2^3 = 8$

תת-מחרוזת	a	b	c
∅	0	0	0
c	0	0	1
b	0	1	0
bc	0	1	1
a	1	0	0
ac	1	0	1
ab	1	1	0
abc	1	1	1

נכונות האלגוריתם: בודקים את כל האפשרויות ולכן בהכרח נגיע גם לתשובה הנכונה.

פסדו-קוד של **בניית המערך שמכיל מספר בינארי סידורי מ 0- עד 1...1111**:

```
plus1(arr[])
        i=arr.length-1
        while( i>=0 && arr[i]==1)
                arr[i--] = 0
        end-while
        if (i>=0) arr[i] = 1
end-plus1
```

פסדו-קוד של **בניית כל הצירופים של n תווים**:

```
String[] allCombinations(String X) //O(2^n)
        n = X.length(), count = 2^n - 1
        String list[count]
        int bin[n]
        for i=0 to count-1
                plus1(bin)
                String t = ""
                for j=0; to n-1
                        if (bin[j] == 1) t = t + X.charAt(j)
                end-for
```

```
        list[i] = t
    end-for
    return list
end-allCombinations
```

פסדו-קוד של חיפוש שלם:

```
String exhaustiveSearch (String X, String Y)
    m = X.length(), n = Y.length()
    String ans = "", maxLen = 0, sShort = X, sLong = Y
    if (m > n)
        sShort = Y
        sLong = X
    end-if
    String tshort[] = allCombinations(sShort)
    String tlong[] = allCombinations(sLong)
    for i=0 to tshort.length-1 //O(2^m)
        len = tshort[i].length()
        for j = 0 to tlong.length-1 //O(2^n)
            if (tshort[i].equals(tlong[j]) && len > maxLen)//O(m)
                maxLen = len
                ans = tshort[i]
            end-if
        end-for
    end-for
    return ans
end-exhaustiveSearch
```

סיבוכיות האלגוריתם: $\mathbf{O(2^{m+n} * \min(m, n))}$

4. <u>תכנון דינאמי</u>

רעיון: בחיפוש השלם בדקנו תתי מחרוזות מיותרות כי תת המחרוזת המשותפת הארוכה ביותר החל מאיבר כלשהו היא לפחות אותה תת מחרוזת אם נחשב החל מתחילת המחרוזות.

תת-מחרוזת $Z = z_1 z_2,...z_k$ של מחרוזת נתונה $X = x_1 x_2 ... x_m$ מעל א"ב Σ, היא מחרוזת כך שקיימת סדרה עולה ממש $i_1, i_2,..., i_k$ של אינדקסים מ-X כך שלכל $j = 1,2,..., k$ מתקיים כי $x_{i_j} = z_j$.

Z היא **תת-מחרוזת משותפת** (common subsequence) של X ו- Y אם היא תת-מחרוזת של X וגם תת-מחרוזת של Y.

<u>סימון:</u> עבור מחרוזת נתונה X מאורך m, נסמן ב- X_i, $0 \le i \le m$, את **הרישא (prefix)** הבאה של X: $X_i = x_1 x_2 ... x_i$ (עבור $i = 0$ נקבל רישא ריקה).

<u>טענה:</u>

תהא $Z = z_1 z_2,...z_k$ תת-המחרוזת המשותפת הארוכה ביותר של X ו- Y הנתונים. אזי:

1. אם $x_m = y_n$, אזי $z_k = x_m = y_n$ ו- Z_{k-1} היא תת-מחרוזת משותפת ארוכה ביותר של X_{m-1} ו- Y_{n-1}.
2. אם $x_m \neq y_n$:
   א. אם $z_k \neq x_m$ אזי Z היא תת-מחרוזת משותפת ארוכה ביותר של X_{m-1} ו- Y.
   ב. אם $z_k \neq y_n$ אזי Z היא תת-מחרוזת משותפת ארוכה ביותר של X ו- Y_{n-1}.

<u>הוכחה:</u> נוכיח כל מקרה בנפרד.

מקרה 1: נניח בשלילה כי $z_k \neq x_m$. אזי ניתן יהיה לשרשר לסוף Z את האיבר x_m (כי $x_m = y_n$), ובכך נקבל מחרוזת משותפת של X ו- Y אשר יותר ארוכה מ- Z, בסתירה לכך ש- Z ארוכה ביותר. ברור כי $Z_{k-1} = z_1 z_2 \ldots z_{k-1}$ היא מחרוזת משותפת של X_{m-1} ו- Y_{n-1}, כל שנותר להראות הוא שהיא גם ארוכה ביותר. נניח בשלילה שלא, אזי קיימת מחרוזת משותפת ארוכה ביותר של X_{m-1} ו- Y_{n-1} שנסמנה W, אשר אורכה גדול ממש מ- $k-1$. על-ידי שרשור האיבר x_m (כי $x_m = y_n$) ל- W, נקבל מחרוזת משותפת ל- X ו- Y באורך גדול ממש מ- k – סתירה.

מקרה 2.א. (מקרה 2.ב דומה - הוכחה סימטרית):

Z תת-מחרוזת משותפת של X_{m-1} ו- Y, כי $z_k \neq x_m$. נותר להראות כי Z תת-מחרוזת ארוכה ביותר. נניח בשלילה כי קיימת תת-מחרוזת משותפת של X_{m-1} ו- Y, W, אשר אורכה גדול ממש מ- k. אזי W גם מחרוזת משותפת של X ו- Y (ארוכה יותר מ- Z) – סתירה. (מ.ש.ל.)

הטענה שהוכחנו קודם מראה שתת-מחרוזת משותפת ארוכה ביותר של שתי מחרוזות מכילה בתוכה תת-מחרוזת משותפת ארוכה של הרישות של שתי המחרוזות.

נסמן ב- $c(i, j)$ את **אורך המחרוזת המשותפת הארוכה ביותר** של X_i ו- Y_j, $0 \leq i \leq m$ ו- $0 \leq j \leq n$. אם $i = 0$ או $j = 0$, אזי ברור כי $c(i, j) = 0$. לפי עובדה זו והטענה שהוכחנו קודם, ניתן לקבל את נוסחת הנסיגה הבאה:

$$c(i, j) = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ c(i-1, j-1) + 1 & i, j > 0, x_i = y_j \\ \max\{c(i, j-1), c(i-1, j)\} & i, j > 0, x_i \neq y_j \end{cases}$$

אם נחשב פונקציה זו באופן רקורסיבי, סיבוכיות הזמן תהיה אקספוננציאלית, זאת משום שיהיו ערכים של הפונקציה אשר יחושבו מספר פעמים.

ניתן לחשב את הפונקציה c ביעילות בעזרת מטריצה $(m+1)\times(n+1)$ (מספור האינדקסים מתחיל מאפס), כאשר הערך שמעניין אותנו הוא $c(m,n)$. ניתן למלא את ערכי מטריצה זו שורה אחר שורה, ובאופן זה נקבל אלגוריתם יעיל לפתרון הבעיה כי יש $O(m*n)$ תאים במטריצה כאשר זמן החישוב של כל תא הוא $O(1)$.

LCS-Matrix(X, Y) //O(m*n)

(1) m ← X.length
(2) n ← Y.length
(3) c[m+1][n+1]
(4) for $i = 0$ to m do: $c(i,0) \leftarrow 0$
(5) for $j = 0$ to n do: $c(0,j) \leftarrow 0$
(6) for $i = 1$ to m do:
(7)   for $j = 1$ to n do:
(8)   if $x_i = y_j$ then $c(i,j) \leftarrow c(i-1,j-1)+1$ // if (X.charAt(i-1) == Y.charAt(j-1))
(9)   else $c(i,j) \leftarrow max\{c(i-1,j), c(i,j-1)\}$
(10)   return c

end-LCS-Matrix

בתא $c(m,n)$ - משבצת בפינה הימנית התחתונה של הטבלה - מאוחסן אורכה של תת-המחרוזת המשותפת הארוכה ביותר:

LCS- Length (X, Y)

(1) m ← X.length
(2) n ← Y.length
(3) c[][] = *LCS-Matrix(X, Y)*
(4) return c[m][n]

end-LCS- Length

נעיר כי **ניתן בזמן של** $O(m+n)$ **למצוא את תת-המחרוזת המשותפת הארוכה ביותר של** X **ו-** Y **בעזרת המטריצה** c**.**

<u>**הסבר:**</u> מתחילים מהתא $c(m,n)$. בשלב שבו נמצאים בתא $c(i,j)$ עוברים לתא ה-$c(i-1,j-1)$ במידה ו-$c(i,j)=c(i-1,j-1)+1$. במידה ומתבצע מעבר כזה, מתקיים $x_i = y_j$. אם אין אפשרות כזו, אז עוברים לתא ה - $c(i,j-1)$ אם $c(i,j)=c(i,j-1)$. במידה וגם תנאי זה לא מתקיים, עוברים לתא $c(i-1,j)$ (ומתקיים $c(i,j)=c(i-1,j)$ מהגדרת האלגוריתם). עוצרים כאשר מגיעים לתא $c(0,j)$ או $c(i,0)$ עבור i,j כלשהם.

```
maxSequence(X, Y)  //O(m*n) + O(min(m,n))
        mat[][] = LCS-Matrix(X, Y)
        row = mat.length, col = mat[0].length
        seqLength = mat[row-1][col-1],  result = ""
        i=row-1, j=col-1, count=seqLength-1
        while (count>=0)
                if (X.charAt(i-1) == Y.charAt(j-1))
                        result = X.charAt(i-1) + result
                        i--, j--, count--
                else if (mat[i][j] == mat[i][j-1])
                        j--
                else
                        i--
                end-if
        end-while
        return result
end-maxSequence
```

<u>דוגמה:</u>

X=abcbdab, Y=bdcaba

	X	a	b	c	b	d	a	b
Y	0	0	0	0	0	0	0	0
b	0	0	1	1	1	1	1	1
d	0	0	1	1	1	2	2	2
c	0	0	1	2	2	2	2	2
a	0	1	1	2	2	2	3	3
b	0	1	2	2	3	3	3	4
a	0	1	2	2	3	3	4	4

אורך תת-המחרוזת המשותפת הארוכה ביותר מאוחסן במשבצת בפינה הימנית התחתונה של הטבלה ושווה ל- 4.

כדי לבנות תת-המחרוזת המשותפת הארוכה ביותר נתחיל מהתא בפינה הימנית התחתונה ונחזור אחורה (כלפי למטה) לפי החוקיות בה מילאנו את המטריצה: אם התווים זהים – נוסיף את התו לתשובה ונחזור אחורה ל- $c(i-1, j-1)$, אם התווים שונים – נחזור אחורה לתא הגדול מבין $c(i, j-1)$ ו- $c(i-1, j)$ ללא הוספת תו לתשובה.

דוגמא לתת-המחרוזת המשותפת הארוכה ביותר היא
LCS(X,Y) = bdab (מסלול)

דוגמא נוספת לתת-מחרוזת המשותפת הארוכה ביותר היא
LCS(X,Y) = bcba (מסלול)