מרתון אלגוריתמים

החומר הנלמד:

- בעיית החניה
- min/max
- משחק המספרים
- LCS
- LIS
- בעיית המטוס
- בעיית המזכירה/הקומפיילר
- בעיית האסירים
- פיצה/סופגניות
- חציון
- מטריצת אחדות
- כדורי זכוכית
- תלת קרב
- אליס ובוב

חזרה על הבעיות הקשות

1. אלגוריתם LCS - מציאת תת מחרוזת משותפת ארוכה ביותר

נתונות 2 מחרוזות $S_1 = "abcde",\ S_2 = "eafddbac"$. יש למצוא את תת המחרוזת המשותפת (שנמצאת בתוך 2 המחרוזות) הארוכה ביותר. $LCS(S_1, S_2) = "abc"$.

תכנות דינאמי:

הגדרת תת בעיה = פונקציית המטרה ואז שמירת כל הערכים של הפונקציה בטבלה (מטריצה).

תת הבעיה: $f(i,j)$ = אורך תת המחרוזת משותפת ארוכה ביותר של S_1 עד מיקום i ושל S_2 עד מיקום j.

טווח הערכים: $0 \le i \le n\ ,\ 0 \le j \le m$.

איפה תמצא התשובה לבעיה הגדולה? $f(n,m)$.

הגדרת הפונקציה: $f(0,j) = f(i,0) = 0$.

$f(i,j) = 1 + f(i-1,j-1)\ if\ S_1[i] = S_2[j]$

אם התווים זהים אז מצאנו התאמה אחת פוטנציאלית והיא לא יכולה להרוס התאמות אחרות כי אלו התווים האחרונים ב 2 המחרוזות. ולכן: ערך ההתאמה הוא 1 + ההתאמה הכי טובה בין מה שנשאר (ללא התו האחרון בראשונה וללא התו האחרון בשנייה)

$f(i,j) = max(f(i,j-1), f(i-1,j)),\ if\ S_1[i] \ne S_2[j]$

אם התווים לא זהים אז ביניהם אין התאמה אבל ייתכן שיש התאצה בין האחרון של הראשונה עם השאר של השנייה (ללא האחרון שלה) או שיש התאמה טובה יותר בין האחרון של השנייה עם כל הראשונה ללא האחרון שלה.

החזרת המחרוזת עצמה: מתחילה מהתא האחרון של המטריצה וחוזרים אחורה לפי הפונקציה =

אם התווים היו זהים אז משרשרים את התו לתשובה וחזרים באלכסון.

אם התווים שונים אז בוחרים את המקסימום מבין תא אחד מעל או תא אחד משמאל.

סיבוכיות: $O(nm)$ עבור בניית המטריצה. (עוברים פעם אחת על כל תא וממלאים אותו ב $O(1)$.

שחזור המחרוזת: $O(n+m)$ כי מתקדמים על מסלול אחורה במטריצה.

קוד:

```
public static String lcs(String s1, String s2) {
        int n = s1.length();
        int m = s2.length();
        int[][] f = new int[n+1][m+1];
        for (int i = 0; i < n+1; i++) {f[i][0] = 0;}
        for (int j = 0; j < m+1; j++) {f[0][j] = 0;}
        for (int i = 1; i < n+1; i++) {
                for (int j = 1; j < m+1; j++) {
                        if(s1.charAt(i-1) == s2.charAt(j-1)) f[i][j] = 1 + f[i-1][j-1];
                        else f[i][j] = Math.max(f[i][j-1], f[i-1][j]);
                }
        }
        // build ans
        int i = n, j = m;
        String ans = "";
        while(f[i][j] != 0) {
                if(s1.charAt(i-1) == s2.charAt(j-1)) {
                        ans = s1.charAt(i-1) + ans;
                        i--; j--;
                }
                else {
                        if(f[i][j-1] > f[i-1][j]) j--;
                        else i--;
                }
        }
        return ans;
    }
```

2. אלגוריתם LIS - מציאת תת סדרה עולה ארוכה ביותר.

נתון מערך A, וצריך למצוא את תת הסדרה (לא בהכרח ברצף) העולה האורכה ביותר ב A.

דוגמא: $A = [5, 2, 9, 7, 8, 5, 3, 2, 1, 10, 6]$ אז: $LIS(A) = [5, 7, 8, 10]$.

תכנות דינאמי:

תת הבעיה:

נגדיר שאיבר אחרון בסדרה עולה יהיה יותר פוטנציאלי מאיבר אחרון אחר בסדרה עולה באותו אורך אם הוא קטן יותר (ואז מאפשר יותר אופציות להוסיף אחריו לסדרה העולה).

הפונקציה: $f(i)$ = תת הסדרה העולה באורך $i+1$ כאשר האיברים בה הם הפוטנציאלים ביותר.

אתחול: $f(0) = A[0]$.

כל איבר נוסף שנפגוש במערך, נחפש היכן כדאי להכניס אותו.

לכן האלגוריתם יעבור על כל איבר, יחפש בחיפוש בינארי (כי המערך תשובה כבר ממויין) את המיקום המתאים לאיבר ויכניס אותו שם במקום איבר קודם שהיה באותו מיקום. אם האיבר החדש גדול מכולם אז הוא מאריך את אורך תת הסדרה העולה.

התשובה תהיה $f(k)$ עבור ה k הכי גדול שהגענו אליו.

סיבוכיות: $O(nlog_2n)$ אם רוצים רק את אורך תת הסדרה.

$O(n^2)$ אם רוצים את הסדרה עצמה.

קוד:

```
public static int lisSize(int[] arr) { // only size - O(nlog(n))
		int n = arr.length;
		int[] lis = new int[n];
		lis[0] = arr[0];
		int k = 1;
		for (int i = 1; i < n; i++) {
			int index = Arrays.binarySearch(lis, 0, k, arr[i]);
			if(index < 0) index = -index - 1; // fix java's results
			if(index == k) k++;
			lis[index] = arr[i];
		}
		System.out.println(Arrays.toString(lis));
		return k;
	}

	public static int[] lis(int[] arr) { // O(n^2)
		int n = arr.length;
		int[] lis = new int[n];
		int[][] mat = new int[n][n];
		lis[0] = mat[0][0] = arr[0];
		int k = 1;
		for (int i = 1; i < n; i++) {
			int index = Arrays.binarySearch(lis, 0, k, arr[i]);
			if(index < 0) index = -index - 1; // fix java's results
			if(index == k) k++;
			lis[index] = mat[index][index] = arr[i];
			for (int j = 0; j < index; j++) {
				mat[index][j] = mat[index-1][j];
			}
		}
		return Arrays.copyOf(mat[k-1], k);
	}
```

מבחן 2019 סמסטר קיץ מועד א

בעיה מס' 1

25 = 10 + 15

10

15

א) בהינתן **שתי** מחרוזות יש לפתח אלגוריתם שמוצא את התת-מחרוזת המשותפת הארוכה ביותר (אחת לפחות).

ב) בהינתן **שלוש** מחרוזות יש לפתח אלגוריתם שמוצא את התת-מחרוזת המשותפת הארוכה ביותר (אחת לפחות).

אלגוריתם, הוכחות, סיבוכיות, ודוגמה.

פתרון:

א. LCS כמו שעשינו.

ב. $s_1 = "zwzabcde",\ s_2 = "cbdxyze",\ s_3 = "cdexyzwz"$.

$LCS(s_1, s_2) = lcs = "bde",\ LCS(lcs, s_3) = "de"$

$LCS(s_1, s_3) = lcs = "zwz",\ LCS(lcs, s_2) = "z"$

$LCS(s_2, s_3) = lcs = "xyz",\ LCS(lcs, s_1) = "z"$

נכליל את הרעיון של 2 מחרוזות בתכנות דינאמי ל 3.

הפונקציה:

$f(0,j,k) = f(i,0,k) = f(i,j,0) = 0$.

$f(i,j,k) = 1 + f(i-1, j-1, k-1)\ ,\ if\ s_1[i] = s_2[j] = s_3[k]$

$f(i,j,k) = max(f(i-1,j,k), f(i,j-1,k), f(i,j,k-1))$

חזרה אחורה על אותו רעיון.

מבחן 2020 סמסטר א מועד א

בעיה מס' 1

25 = 10 + 15

10

א) נתונות **שתי** מחרוזות. יש לפתח אלגוריתם שמחזיר את האורך של התת-מחרוזת המשותפת הארוכה ביותר ומוצא את התת-מחרוזת המשותפת בעלת אורך זה (אחת לפחות).

ב) נתונות **שתי** מחרוזות X ו-Y. יש לפתח אלגוריתם שמוצא את אורכה של המחרוזת **הקצרה** ביותר, כך שמחרוזות X ו- Y הן תתי-מחרוזות שלה.

דוגמה:

קלט: X=abcbdab Y=bdcaba

פלט: 9 (המחרוזת הקצרה ביותר היא abdcabdab).

15

אלגוריתם, הוכחות, סיבוכיות, ודוגמה.

פתרון:

א. LCS רגיל כמו שעשינו.

ב. $|SCS(X,Y)| = |X| + |Y| - |LCS(X,Y)|$

בדוגמא: $LCS(X,Y) = "bcab"$

$ans = "abdcbdaba"$

3. בעית המטוס: נתונה מטריצה $n \times m$ כאשר לכל תא יש מחיר מעבר לתא שמימינו x ולתא שמעליו y.
המטרה: למצוא את המסלול בעל המחיר הכולל הנמוך ביותר מתא $(0,0)$ לתא (n,m).
מותר לזוז רק למעלה או ימינה.

תכנות דינאמי:

הפונקציה: $f(i,j)$ = המחיר המינימאלי של המסלול מ $(0,0)$ ל (i,j).

התשובה תהיה ב: $f(n,m)$.

(פה מטריצה מצויירת קצת הפוך = מתחילים מתא שמאלי תחתון ומסיימים בתא ימני עליון)

הגדרת הפונקציה:

$f(0,0) = 0$.

$f(0,i) = f(0,i-1) + M[0][i-1].x$

$$f(i,0) = f(i-1,0) + M[i-1][0].y$$
$$f(i,j) = min(f(i-1,j) + M[i-1][j].y,\ f(i,j-1) + M[i][j-1].x)$$
המעבר $M[i][j].x$ הוא מנקודה (i,j) ימינה לנקודה $(i,j+1)$
המעבר $M[i][j].y$ הוא מנקודה (i,j) למעלה לנקודה $(i+1,j)$.

סיבוכיות: $O(nm)$ עבור בניית המטריצה, $O(n+m)$ עבור החזרת המסלול.

שאלה ממבחן 2020 סמסטר א מועד א

בעיה מס' 2:

25 = 10 + 15

נתון "דף חשבוני" (עם משקלים מונחים על הצלעות) בגודל M על N, ושני קדקודים עם קואורדינאטות (p_1,q_1) ו-(p_2,q_2).

10

א) יש לפתח אלגוריתם המחזיר את עלות המסלול **הזול** ביותר בין הקודקודים $(0,0)$ ו- (N,M).

ב) יש לפתח אלגוריתם שמחזיר:

15

"**כן**", אם הקודקודים (p_1,q_1) ו-(p_2,q_2) שייכים למסלול קצר ביותר (אחד לפחות) בין הקודקודים $(0,0)$ ו- (N,M).

"**לא**", אם הקודקודים (p_1,q_1) ו-(p_2,q_2) **לא** שייכים לאף מסלול קצר ביותר בין הקודקודים $(0,0)$ ו- (N,M).

אלגוריתם, הוכחות, סיבוכיות, ודוגמה.

פתרון:
א. בעיית המטוס הרגילה כמו שעשינו.
ב. 2 הנקודות נמצאות על מסלול קצר ביותר בין $(0,0)$ ל (n,m) אם ורק אם:
אורך המסלול הקצר בין $(0,0)$ ל (n,m) = אורך המסלול הקצר בין $(0,0)$ ל (p_1,q_1) + אורך המסלול הקצר בין (p_1,q_1) ל (p_2,q_2) + אורך המסלול הקצר בין (p_2,q_2) ל (n,m).

בעיות:
1. לא ידוע האם ניתן להגיע מהנקודה הראשונה לשנייה או מהשנייה לראשונה או שלא ניתן להגיע מאף אחת מהן לאף אחת מהן.

הפתרון:
תחילה נשווה בין (p_1,q_1) ל (p_2,q_2)
אם $p_1 \le p_2$ וגם $q_1 \le q_2$ אז הנקודה הראשונה במסלול היא (p_1,q_1)
אם $p_2 \le p_1$ וגם $q_2 \le q_1$ אז הנקודה הראשונה במסלול היא (p_2,q_2)
בכל מקרה אחר - התשובה היא שהן לא על מסלול קצר ביותר כי הן לא יכולות להיות על מסלול אחד.

כדי לחשב את מסלולי הביניים, נשנה את התכנות הדינאמי לעבוד מכל קודקוד לכל קודקוד ולא רק מ $(0,0)$ ל (n,m).