בס"ד

סיכום – LIS – תת הסדרה העולה הארוכה ביותר

הבעיה: נתון מערך של מספרים. יש למצוא את אורך תת הסדרה העולה הארוכה ביותר מתוך המערך, בנוסף יש למצוא גם את הסדרה עצמה.

דוגמא: עבור המערך:

1	100	101	2	3	4	5	6	7

תוחזר הסדרה:

1	2	3	4	5	6	7

פתרון:

השיטה החמדנית – נקבע שהראשון הוא תחילת הסדרה, נעבור על המערך וניקח בכל שלב את האיבר הבא בגודלו עד שנגיע לסוף המערך.

בדוגמא שלנו, ניקח את 1, לאחר מכן, ניקח את 100 שגדול ממנו ואז את 101, לאחר מכן אין איברים גדולים יותר ולכן יוחזר:

1	100	101

- סיבוכיות השיטה: $O(n)$ - מעבר יחיד על המערך.
- נכונות השיטה: כפי שניתן לראות, השיטה לא החזירה את התשובה הנכונה כי לא תמיד האיבר הראשון הוא חלק מהסדרה. בנוסף, לפעמים כדאי לוותר על מספר איברים כדי לקחת אחרים טובים יותר.
- הקוד:

```
public static Vector<Integer> greedyLIS(int[] arr) {
	Vector<Integer> ans = new Vector<Integer>();
	ans.add(arr[0]);
	int max = arr[0];
	for (int i = 1; i < arr.length; i++) {
		if(arr[i] > max) {
			ans.add(arr[i]);
			max = arr[i];
		}
	}
	return ans;
}
```

השיטה החמדנית המשופרת – במקום לקבוע שהראשון הוא תחילת הסדרה, נקבע בכל פעם איבר אחר (עבור כל איבר במערך) ונעבור על המשך המערך וניקח בכל שלב את האיבר הבא בגודלו עד שנגיע לסוף המערך. (כלומר, נפעיל את השיטה החמדנית הרגילה על המשך המערך)

בדוגמא שלנו, ניקח את 1, לאחר מכן, ניקח את 100 שגדול ממנו ואז את 101, לאחר מכן אין איברים גדולים יותר, נמשיך ונקבע את 100 להיות הראשון ואז נקבל סדרה באורך 2. וכשנגיע לאיבר 2 ונתחיל ממנו נקבל את הסדרה שתוחזר:

2	3	4	5	6	7

- סיבוכיות השיטה: $O(n^2)$ – כי על כל איבר חוזרים ובודקים את המשך המערך החל ממנו.
- נכונות השיטה: כפי שניתן לראות, השיטה עדיין לא החזירה את התשובה הנכונה כי לא פתרנו את הבעיה שלפעמים כדאי לוותר על מספר איברים כדי לקחת אחרים טובים יותר.
- הקוד:

```
public static Vector<Integer> greedyLISImproved(int[] arr) {
	Vector<Integer> ans = new Vector<Integer>();
	int maxLen = 0;
	for (int i = 0; i < arr.length; i++) {
		Vector<Integer> temp = greedyLIS(arr,i);
		if(temp.size() > maxLen) {
			maxLen = temp.size();
			ans = temp;
		}
	}
	return ans;
}
```

<u>שיטת השימוש באלגוריתם דומה קיים</u> – נשתמש באלגוריתם של LCS (מציאת תת המחרוזת המשותפת הארוכה ביותר בין 2 מחרוזות) נשכתב את האלגוריתם שיתאים ל 2 מערכים של מספרים ונפעיל אותו על המערך הנתון ועל המערך הנתון לאחר מיון. כלומר נבצע את הקריאה: $LCS(arr, Sort(arr))$.

בדוגמא שלנו, נפעיל על המערך

1	100	101	2	3	4	5	6	7

עם

1	2	3	4	5	6	7	100	101

ואכן נקבל את התשובה הנכונה.

- סיבוכיות השיטה: $O(n^2)$ – כי LCS היא בסיבוכיות ריבועית (תכנות דינאמי)
- נכונות השיטה: מכיוון שהמטרה היא למצוא תת סדרה עולה ארוכה ביותר אז בהכרח התשובה היא תת סדרה של המערך הממוין כך ששומרים על סדר האיברים במערך ולכן תת הסדרה המשותפת בין 2 המערכים היא בדיוק התשובה.
- הקוד:

```
public static int[] LIS_usingLCS(int[] arr) {
	int[] s_arr = new int[arr.length];
	for (int i = 0; i < s_arr.length; i++) {
		s_arr[i] = arr[i];
	}
	Arrays.sort(s_arr);
	return lcs(arr,s_arr);
}

private static int[] lcs(int[] X,int[] Y){
	int mat[][] = buldMatrix(X,Y);
	int i = mat.length-1;
	int j = mat[0].length-1;
	int end = mat[i][j];
	int[] ans = new int[end];
	int start=0;
	while (start<end) {
		if (X[i-1]==Y[j-1]) {
			ans[end-start-1] = X[i-1];
			i--; j--; start++;
		}
		else if (mat[i-1][j]>=mat[i][j-1]) i--;
		else j--;
	}
	return ans;
}
```

```
private static int[][] buldMatrix(int[] X,int[] Y) {
int row=X.length+1;
int col=Y.length+1;
int mat[][] = new int[row][col];
for (int i=1; i<row; i++) {
      for (int j=1; j<col; j++) {
            if (X[i-1]==Y[j-1]) mat[i][j] = mat[i-1][j-1] + 1;
            else mat[i][j] = Math.max(mat[i-1][j], mat[i][j-1]);
      }
}
return mat;
}
```

<u>חיפוש שלם</u> – נייצר את כל תתי המערכים האפשריים ונבדוק עבור כל תת מערך אם הוא תת סדרה עולה (מתחילתו ועד סופו), אם כן, נבדוק האם הוא הכי ארוך שמצאנו עד עכשיו ונשמור אותו.

בדוגמא שלנו, נייצר את כל תתי המערכים של המערך

1	100	101	2	3	4	5	6	7

ואכן בסופו של דבר נגיע לתשובה הנכונה.

1	2	3	4	5	6	7

- סיבוכיות השיטה: $O(2^n \cdot n)$ – מספר תתי המערכים (תתי הקבוצות) ועל כל תת מערך עוברים ובודקים האם הוא ממוין בסדר עולה.
- נכונות השיטה: בודקים את כל האפשרויות ולכן בהכרח נגיע גם לתשובה הנכונה.
- הקוד:

```
public static int[] wholeSearch(int[] arr) {
            Vector<int[]> allseq = getAllSubSequence(arr);
            int[] ans = null;
            int max = 0;
            for(int[] a : allseq) {
                  boolean increase = true;
                  for (int i = 1; i < a.length; i++) {
                        if(a[i] < a[i-1]) increase = false;
                  }
                  if(increase) {
                        if(a.length > max) {
                              max = a.length;
                              ans = a;
                        }
                  }

            }
            return ans;
      }

private static Vector<int[]> getAllSubSequence(int[] arr) {
Vector<int[]> ans = new Vector<int[]>();
for (int binary = 0; binary < Math.pow(2,arr.length); binary++) {
      Vector<Integer> temp = new Vector<Integer>();
      int b = binary, i = 0;
      while(b != 0) {
            if(b % 2 == 1) temp.add(arr[i]);
                  i++;
                  b /= 2;
            }
```

```
            int[] sequence = new int[temp.size()];
            for (int j = 0; j < sequence.length; j++) {
                sequence[j] = temp.get(j);
            }
            ans.add(sequence);
        }
return ans;
}
```

<u>תכנות דינאמי</u> – בחיפוש השלם, בדקנו תתי מערכים מיותרים כי תת הסדרה העולה הארוכה ביותר החל מאיבר כלשהו היא לפחות אותה תת סדרה אם נחשב החל מתחילת המערך.

נייצר מטריצה שבה כל שורה i תייצג את תת הסדרה העולה הארוכה ביותר באורך $i+1$. נעבור על המערך ועבור כל איבר, נבדוק לאיזו תת סדרה עולה הוא יכול להצטרף, כלומר, נבדוק את מקומו הסידורי על האלכסון, אם האיבר גדול מכל האלכסון, סימן שניתן להוסיף איבר לסדרה העולה הארוכה ביותר , נעתיק את הסדרה הארוכה ביותר לשורה מתחת ונוסיף את האיבר החדש. אם האיבר לא גדול מכולם, הוא יחליף את האיבר הראשון שגדול ממנו, כלומר, נצרף את האיבר החדש לסדרה הארוכה ביותר שבה הוא יכול להחליף את האיבר האחרון לאיבר קטן יותר (איבר קטן יותר = פוטנציאל להמשך הסדרה הוא יותר גדול), נעתיק את הסדרה שאליה האיבר התוסף (שורה מעל) לשורה מתחת.

בדוגמא שלנו

1	100	101	2	3	4	5	6	7

1 ייכנס בשורה ה 0

100 גדול מ 1 ולכן הוא ייכנס בשורה ה 1 – ואז נעתיק את 1 לשורה של 100

101 גדול מכל הקודמים ולכן הוא ייכנס בשורה 2 ונעתיק את האיברים משורה 1 לשורה של 101.

2 אינו גדול מכולם, מקומו בין 1 ל 100 ולכן הוא יחליף את 100 בשורה 1 ונעתיק את כל האיברים בשורה הקודמת לשורה 1. וכך הלאה, בסופו של דבר המטריצה תיראה כך:

1								
1	2							
1	2	3						
1	2	3	4					
1	2	3	4	5				
1	2	3	4	5	6			
1	2	3	4	5	6	7		

והשורה האחרונה שהגענו אליה היא התשובה.

- סיבוכיות השיטה: $O(n^2)$ – עוברים על המערך ועבור כל איבר, מחפשים את מיקומו הסידורי ע"י חיפוש בינארי באלכסון המטריצה ושמים את האיבר במקום המתאים ובנוסף, מעתיקים את השורה שמעל לשורה שבה התוסף האיבר.ולכן סה"כ: $O\left(n \cdot (\log n + n)\right) = O(n^2)$, אם נרצה רק את אורך הסדרה העולה הארוכה ביותר, נוכל לוותר על ההעתקה (כי מספר השורות שהגענו – זה האורך) והסיבוכיות תרד ל $O(n \cdot \log n)$
- נכונות השיטה: הראנו את האלגוריתם.
- הקוד:

```java
public static int[] LISDynamic(int[] arr) {
	int n = arr.length;
	int[][] mat = new int[n][n];
	mat[0][0] = arr[0];
	int len = 1;
	for (int i = 1; i < n; i++) {
		int index = binarySearchBetween(mat,len,arr[i]);
		mat[index][index] = arr[i];
		if(index == len) len++;
		copy(mat,index);
	}
	int[] ans = new int[len];
	for (int i = 0; i < ans.length; i++) {
		ans[i] = mat[len-1][i];
	}
	return ans;
}

private static void copy(int[][] mat, int index) {
	for (int i = 0; i < index; i++) {
		mat[index][i] = mat[index-1][i];
	}
}

private static int binarySearchBetween(int[][] mat, int end, int v) {
	if(v > mat[end-1][end-1]) return end;
	if(v < mat[0][0]) return 0;
	int high = end, low = 0;
	while(low <= high) {
		if(low == high)return low;
		int mid = (low + high)/2;
		if(mat[mid][mid] == v) return mid;
		if(mat[mid][mid] > v) high = mid;
		else low = mid+1;
	}
	return -1;
}
```