בס"ד

סיכום – מטריצת אחדות

הבעיה: נתונה מטריצה $n \times m$ המכילה אפסים ואחדות. יש למצוא את ריבוע האחדות הגדול ביותר במטריצה.

לדוגמא: עבור המטריצה:

1	0	0	1	1	1
0	0	1	1	1	0
0	1	1	1	0	0
0	1	1	1	0	1
1	1	1	1	0	0

ריבוע האחדות הגדול ביותר הוא באורך 3.

פתרון:

חיפוש שלם – נייצר את כל תתי הריבועים ונבדוק עבור כל ריבוע אם כולו מלא אחדות, אם כן, נבדוק האם גודלו הכי גדול שמצאנו ואם כן נשמור את התשובה.

- סיבוכיות השיטה: $O((n \cdot m)^2 \cdot \min(n,m))$ – מספר תתי הריבועים בגודל 1 הוא $n \cdot m$, בגודל 2 הוא: $(n-1)(m-1)$ וכן הלאה כך שמספר כל תתי הריבועים הוא: $n \cdot m + (n-1)(m-1) + \cdots + (n - \min(n,m))(m - \min(n,m))$, כפול $n \cdot m$ כדי לעבור על כל תת ריבוע.
- נכונות השיטה: בודקים את כל האפשרויות ולכן בהכרח נגיע גם לתשובה הנכונה.
- הקוד:

```
public static int getBiggestSubMatrix(int[][] mat) {
	Vector<int[][]> allMatrix = getAllSubMatrix(mat);
	int maxSize = 0;
	for (int[][] m : allMatrix) {
		boolean isOnes = true;
		for (int i = 0; i < m.length; i++) {
			for (int j = 0; j < m[0].length; j++) {
				if(m[i][j] == 0) isOnes = false;
			}
		}
		if(isOnes) {
			if(m.length > maxSize) maxSize = m.length;
		}
	}
	return maxSize;
}

private static Vector<int[][]> getAllSubMatrix(int[][] mat) {
	Vector<int[][]> ans = new Vector<int[][]>();
	for (int i = 1; i < Math.min(mat.length, mat[0].length)+1; i++)
{
		for (int j = 0; j < mat.length-i+1; j++) {
			for (int k = 0; k < mat[0].length-i+1; k++) {
				int[][] temp = new int[i][i];
				for (int i1 = j; i1 < j+i; i1++) {
					for (int j1 = k; j1 < k+i; j1++) {
						temp[i1-j][j1-k] = mat[i1][j1];
					}
```

```
                        }
                        ans.add(temp);
                    }
                }
            }
    return ans;
}
```

<u>תכנות דינאמי</u> – נייצר מטריצה שבה בכל תא i,j יישמר אורך תת הריבוע שמסתיים ב (i,j) כך שאורך תת הריבוע הגדול ביותר יישמר באחד מהתאים במטריצה – זו תהיה התשובה.

נמלא את המטריצה באופן הבא:

אם במטריצה המקורית היה 0 במקום (i,j) אז לא נעשה דבר כי אין ריבוע.

אם היה 1 – אז ניקח את המינימום מבין 3 השכנים ונוסיף 1, כלומר:

$$mat[i][j] = \min(mat[i][j-1], mat[i-1][j], mat[i-1][j-1]) + 1$$

הסיבה היא – כי גודל תת הריבוע יגדל רק אם 3 השכנים שלו היו 1 כדי שנקבל ריבוע מלא ואז התא החדש יתוסף לריבוע שבו משתתפים כל 3 השכנים שלו ולכן ניקח את הריבוע עם הגודל המינימאלי.

נמלא תחילה עמודה ראשונה ושורה ראשונה כי מילוי המטריצה תלוי בשכנים הקודמים.

בדוגמא שלנו המטריצה תיראה כך:

1	0	0	1	2	1
0	0	1	2	1	0
0	1	2	3	0	0
0	1	2	2	0	1
1	1	1	1	0	0

והתשובה היא אכן 3.

כדי לקבל את הריבוע עצמו– נשמור תוך כדי המילוי את הגודל הגדול ביותר ואת מיקום התא שבו נמצא המקסימום וזה הקודקוד הימני העליון של הריבוע.

- סיבוכיות השיטה: $O(n \cdot m)$ – ממלאים את המטריצה לפי החוקיות.
- נכונות השיטה: הראנו את האלגוריתם.
- הקוד:

```
public static int getBiggestSubMatrix(int[][] mat) {
    int n = mat.length;
    int m = mat[0].length;
    int[][] help = new int[n][m];
    int max = 0,imax = 0,jmax = 0;
    for (int i = 0; i < n; i++) {
        help[i][0] = mat[i][0];
    }
    for (int i = 0; i < m; i++) {
        help[0][i] = mat[0][i];
    }
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < m; j++) {
            if(mat[i][j] == 1) {
```

```
                        help[i][j] = min(help[i][j-1],help[i-
1][j],help[i-1][j-1]) + 1;
                        if(help[i][j] > max) {
                              max = help[i][j];
                              jmax = j-max+1;
                              imax = i-max+1;
                        }
                  }
            }
      }
      if(max != 0)System.out.println("Max square length is - " + max
+ ", start at: (" + imax + "," + jmax +")");
      return max;
}

private static int min(int i, int j, int k) {
      if(i <= j && i <= k) return i;
      if(j <= i && j <= k) return j;
      if(k <= i && k <= j) return k;
      else return -1;
}
```