מטריצת אחדות

תיאור הבעיה

בהינתן מערך דו-מימדי **A** (מטריצה) המורכב מאפסים ואחדות בלבד, יש למצוא את תת-המטריצה **הריבועית** הגדולה ביותר המורכבת מאחדות בלבד.

בדוגמה הבאה, נסתכל על המטריצה A, מטריצת האחדות הגדולה ביותר היא בגודל 3.

	0	1	2	3	4
0	1	1	0	0	0
1	1	1	1	1	1
2	0	1	1	1	0
3	1	1	1	1	1
4	0	1	0	0	0

- אפשרות א':
  - נעזר ברעיון לפתרון הדינאמי עבור בעיית רצף האחדות הגדול ביותר במערך.

תזכורת

	0	1	2	3	4	5	6	7	8	9
arr	1	1	0	1	0	0	**1**	**1**	**1**	**1**

	0	1	2	3	4	5	6	7	8	9
count	1	2	0	1	0	0	**1**	**2**	**3**	**4**

⇐ נבנה 3 מטריצות עזר:

1. מטריצה **B** המייצגת את רצף האחדות לפי שורות.
2. מטריצה **C** המייצגת את רצף האחדות לפי עמודות.
3. מטריצה **D** המייצגת את גודל ריבוע האחדות בכל תא.

מטריצה D מייצגת את גודל ריבוע האחדות בכל תא. לכן נצטרך לבדוק מה קורה בתא זה במטריצות האחרות שבנינו כדי לדעת האם אנחנו ממשיכים ריבוע קיים או לא.

יש בידינו 4 מטריצות.

C עמודות

	0	1	2	3	4
0	1	1	0	0	0
1	2	2	1	1	1
2	0	3	2	2	0
3	1	4	3	3	1
4	0	5	0	0	0

A מקורית

	0	1	2	3	4
0	1	1	0	0	0
1	1	1	1	1	1
2	0	1	1	1	0
3	1	1	1	1	1
4	0	1	0	0	0

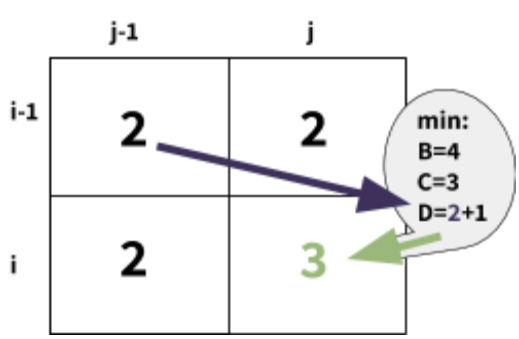
B שורות

	0	1	2	3	4
0	1	2	0	0	0
1	1	2	3	4	5
2	0	1	2	3	0
3	1	2	3	4	5
4	0	1	0	0	0

איך נמלא את מטריצת הפתרון D?
עבור תא ספציפי: אם במטריצה המקורית יש בו 0 נרשום 0 אך אם יש בו 1 נצטרך לבדוק האם הוא משלים לנו ריבוע אחדות מלמעלה, משמאל ומהאלכסון.
נעתיק את השורה הראשונה של A ל-D וגם את העמודה הראשונה של A ל-D ונתחיל לחשב:

$$D[i][j] = min(B[i][j]\ ,\ C[i][j],\ D[i-1][j-1]\ +1)$$

במטריצה D נבצע לדוגמה:



מימוש שיטה א':

```
public static int solution(int[][] A) {
   printMatrix(A);
   int[][] D = generateD(A);
   printMatrix(D);
   return getBiggest(D);
}


public static void printMatrix(int[][] mat) {

   for(int i = 0; i < mat.length; i++) {
       for(int j = 0; j < mat[0].length; j++) {
           System.out.print(mat[i][j] + " ");
       }
       System.out.println();
   }
   System.out.println();
}

//Rows
public static int[][] generateB(int[][] A) {
   int[][] B = new int[A.length][A[0].length];

   for(int i = 0 ; i < A.length; i++) {
       B[i][0] = A[i][0];
   }
   for(int i=0; i < A.length; i++) {
       for(int j=1; j < A[0].length; j++) {
           if(A[i][j] == 1) {
               B[i][j] = B[i][j-1] + 1;
           }
       }
   }

   return B;
}

//Columns
public static int[][] generateC(int[][] A) {
   int[][] C = new int[A.length][A[0].length];

   for(int i = 0 ; i < A[0].length; i++) {
       C[0][i] = A[0][i];
   }
   for(int i=1; i < A.length; i++) {
       for(int j=0; j < A[0].length; j++) {
```

```
            if(A[i][j] == 1) {
                C[i][j] = C[i-1][j] + 1;
            }
        }
    }
    return C;
}

// solution matrix
public static int[][] generateD(int[][] A) {
    int[][] B = generateB(A); // rows
    int[][] C = generateC(A); // columns
    int[][] D = new int[A.length][A[0].length];

    for(int i = 0 ; i < A.length; i++) {
        D[i][0] = A[i][0];
    }
    for(int i = 0 ; i < A[0].length; i++) {
        D[0][i] = A[0][i];
    }
    for(int i=1; i < A.length; i++) {
        for(int j=1; j < A[0].length; j++) {
            if(A[i][j] == 1) {
                D[i][j] = min(B[i][j],C[i][j],D[i-1][j-1] +1) ;
            }
        }
    }
    return D;
}

public static int min(int b, int c, int d) {
    int min = b;
    if(min > c) min = c;
    if(min > d) min = d;
    return min;
}

public static int getBiggest(int[][] D) {
    int max = 0;

    for(int i = 0 ; i < D.length; i++) {
        for(int j = 0 ; j < D[0].length; j++) {
            if(D[i][j] > max)
                max = D[i][j];
        }
    }
```

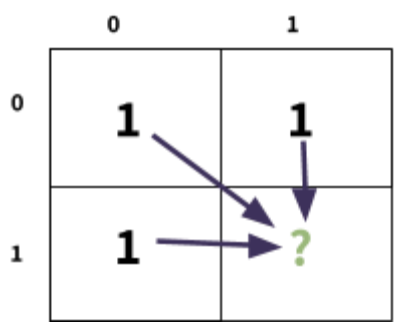
```
    return max;
}

public static void main(String[] args) {

    int[][] A = {{1,1,0,0,0},
                 {1,1,1,1,1},
                 {0,1,1,1,0},
                 {1,1,1,1,1},
                 {0,1,0,0,0}};
    System.out.println("the biggest square is: " +solution(A)); // prints 3
}
```

- **סיבוכיות זמן ריצה:** בניית 3 מטריצות בגודל nxn , $O(3 \cdot n^2)$ וגם החזרת הערך הגדול ביותר $O(n^2)$ - סה"כ $O(4 \cdot n^2)$.
- **נכונות הפתרון:** מבטיח בהכרח פתרון, אבל נאלץ להשתמש ב-3 מטריצות, האם אפשר לייעל את זה? -

❖ אפשרות ב':
- נרצה לקבל מיד את מטריצת התשובות ללא שימוש במטריצות עזר.
- המטריצות היחידות שבידינו הם מטריצה A ומטריצת הפתרון D.

איך נעשה זאת? -

1. נעתיק את השורות העמודות הראשונות של A ל-D.
2. נתבונן בציור, אפשר לראות כי עבור **'?'** נציב במטריצה D את הערך 2, לכן אין לנו באמת צורך יותר במטריצות העזר שהוצגו באפשרות הראשונה.



3. אם במטריצה A יופיע בתא את 0 אז נציב 0 גם אצל D.
4. אם במטריצה A יופיע בתא את 1 אז נפעל לפי הצעד הבא:

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

נשים לב כי הפעם חיברנו 1+ עבור התא.

מימוש אפשרות ב'

```
public static int solution(int[][] A) {
   int max = 0;
   int[][] D = new int[A.length][A[0].length];

   for(int i = 0 ; i < A.length; i++) {
       D[i][0] = A[i][0];
       D[0][i] = A[0][i];
   }

   for(int i = 1 ; i < A.length; i++) {
       for(int j = 1 ; j < A[0].length; j++) {
           if(A[i][j] == 1) {
               D[i][j] = min(D[i-1][j], D[i][j-1], D[i-1][j-1]) + 1;
               if(D[i][j] > max)
                   max = D[i][j];
           }
       }
   }

   return max;
}

public static int min(int a, int b, int c) {
   int min = a;
   if(min > b) min = b;
   if(min > c) min = c;
   return min;
}
public static void main(String[] args) {

   int[][] A = {{1,1,0,0,0},
           {1,1,1,1,1},
           {0,1,1,1,0},
           {1,1,1,1,1},
           {0,1,0,0,0}};
   System.out.println("the biggest square is: " +solution(A));
}
```

- **סיבוכיות זמן ריצה:** בניית מטריצה D ובמקביל קבלת max ולכן $O(n^2)$.
- **נכונות הפתרון:** מבטיח בהכרח פתרון עבור כל מטריצה (nxn).