מרתון אלגוריתמים

שאלה 1

כתבו אלגוריתם המקבל מטריצה של אפסים ואחדות ומוצא את הסימן "+" של אחדות הגדול ביותר.
כאשר מספר האחדות בתוך ה "+" זהה לכל הכיוונים (סימטרי).
לדוגמא: עבור המטריצה:

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

יוחזר 17.

פתרון:

נשתמש בתכנות דינאמי:

פונקציית מטרה: $u(i,j), d(i,j), l(i,j), r(i,j)$ להיות רצף האחדות הגדול ביותר מתא (i,j) לכיוון המתאים.

$$ans(i,j) = min(u(i,j), d(i,j), l(i,j), r(i,j))$$

התשובה תהיה: $max_{i,j}(ans(i,j) - 1) * 4 + 1$.

בהינתן שאכן $u(i,j), d(i,j), l(i,j), r(i,j)$ נכונות. בתא i,j יהיה פלוס בגודל לכל היותר k אחדות לכל כיוון אם ורק אם כל אחד $u(i,j), d(i,j), l(i,j), r(i,j)$ הוא לפחות k ואחד מהם הוא בדיוק k כי אחרת, אם כולם גדולים מ k אז נוכל ליצור + גדול יותר ואם קיים מישהו קטן מ k אז לא נוכל ליצור פלוס בגודל k כי בכיוון הנ"ל לא יהיו מספיק אחדות. ולכן תחזור התשובה הנכונה שהיא כמות האחדות לכל צד (לא כולל האמצע ולכן מחסרים 1) כפול 4 (כיוונים) + 1 (האמצע).

כל כיוון מתמלא נכון, נוכיח זאת באינדוקציה:

בסיס: עבור למעלה = אכן ממלאים את השורה הראשונה של המטריצה המקורית כי אם יש 0 אז גודל רצף האחדות הוא 0 ואם יש 1 אז מכיוון שזו השורה העליונה ביותר אז רצף האחדות עד מיקום זה מלמעלה הוא 1 (באותו אופן וסימטרי עבור שאר הפונקציות)

צעד, נניח שהחישוב עד מיקום (i,j) נכון ונוכיח עבור מיקום (i+1,j) לכל j.

אם במיקום (i+1,j) יש 0 אז הרצף נקטע ולכן $u(i+1,j) = 0$ ואכן זה מחושב לפי האלגוריתם.

אם יש שם 1 אז ניתן להאריך את הרצף למטה בעוד 1 ולכן לפי הנחת האינדוקציה, רצף האחדות הוא $u(i,j) + 1$ ואכן זה מחושב לפי האלגוריתם. מש"ל.

חישוב פונקציות העזר:

$$u(0,j) = mat(0,j)$$

$u(i,j) = 0\ if\ mat(i,j) = 0$ וגם $u(i,j) = u(i-1,j) + 1,\ if\ mat(i,j) = 1$

$$d(n-1,j) = mat(n-1,j)$$

$d(i,j) = 0\ if\ mat(i,j) = 0$ וגם $d(i,j) = d(i+1,j) + 1,\ if\ mat(i,j) = 1$

$$l(i,0) = mat(i,0)$$

$l(i,j) = 0\ if\ mat(i,j) = 0$ וגם $l(i,j) = l(i,j-1) + 1,\ if\ mat(i,j) = 1$

$$r(i,m-1) = mat(i,m-1)$$

$r(i,j) = 0\ if\ mat(i,j) = 0$ וגם $r(i,j) = r(i,j+1) + 1,\ if\ mat(i,j) = 1$

סיבוכיות: אתחול 4 מטריצות $O(nm)$ וחיפוש המקסימום: $O(nm)$.
סה"כ: $O(nm)$.

קוד:

```
public static int largestPlus(int[][] mat) {
		int n = mat.length;
		int m = mat[0].length;
		int[][] u = new int[n][m];
		int[][] d = new int[n][m];
		int[][] l = new int[n][m];
		int[][] r = new int[n][m];
		for (int i = 0; i < m; i++) {u[0][i] = mat[0][i];}
		for (int i = 0; i < m; i++) {d[n-1][i] = mat[n-1][i];}
		for (int i = 0; i < n; i++) {l[i][0] = mat[i][0];}
		for (int i = 0; i < n; i++) {r[i][m-1] = mat[i][m-1];}
		int max = 0;
		for (int i = 0; i < n; i++) {
			for (int j = 0; j < m; j++) {
				if(mat[i][j] != 0) {
					if(i != 0) {u[i][j] = u[i-1][j] + 1;}
					if(j != 0) {l[i][j] = l[i][j-1] + 1;}
				}
			}
		}
		for (int i = n-1; i >= 0; i--) {
			for (int j = m-1; j >= 0; j--) {
				if(mat[i][j] != 0) {
					if(i != n-1) {d[i][j] = d[i+1][j] + 1;}
					if(j != m-1) {r[i][j] = r[i][j+1] + 1;}
				}
			}
		}
		for (int i = 0; i < n; i++) {
			for (int j = 0; j < m; j++) {
			if(Math.min(Math.min(u[i][j], d[i][j]),Math.min(l[i][j], r[i][j])) > max) {
			max = Math.min(Math.min(u[i][j], d[i][j]),Math.min(l[i][j], r[i][j]));
				}
			}
		}
		return (max-1)*4 + 1;
	}
```

שאלה 2

ברשותך 25 סוסים. מהו המספר המינימלי של מרוצים שעליך לערוך כדי לאתר את 3 הסוסים המהירים ביותר?

חשוב לציין כי:

1. בכל מרוץ לוקחים חלק לא יותר מ-5 סוסים.
2. מהירות הסוס לא תלויה במרוץ בו הוא משתתף.
3. אין שני סוסים שמהירויותיהם זהות.
4. אין לך שעון למדידת זמן.

אלגוריתם, הוכחות, סיבוכיות ודוגמה.

פתרון:
כל אחד מהסוסים יכול להיות המהיר ביותר ולכן כל סוס חייב להשתתף בלפחות תחרות אחד. מכאן חייבים להתקיים לפחות 5 מירוצים.
כל מירוץ קובע סדרה של 5 סוסים ממויינים לפי המהירות:
$a_1 > a_2 > a_3 > a_4 > a_5$.
$a_6 > a_7 > a_8 > a_9 > a_{10}$.
$a_{11} > a_{12} > a_{13} > a_{14} > a_{15}$.
$a_{16} > a_{17} > a_{18} > a_{19} > a_{20}$.
$a_{21} > a_{22} > a_{23} > a_{24} > a_{25}$.
כעת חייבים מירוץ נוסף כדי לגלות מי המהיר ביותר. יש 5 פוטנציאליים.
לאחר המירוץ ה 6 נקבל סדרה ממויינת של הראשונים: $a_1 > a_6 > a_{11} > a_{16} > a_{21}$.
מכאן: a_1 הוא המהיר ביותר.
לכן: המועמדים למקום 2+3 הם: $a_2, a_3, a_6, a_7, a_{11}$
ולכן מירוץ אחד נוסף בין המועמדים הללו יגלה מיהם 3 הראשונים.
סה"כ: 7 מירוצים.

שאלה 3

א) נתונה סדרת מספרים חיוביים: $a_1, a_2, \ldots, a_i, \ldots, a_n$.
במשחק זה משתתפים שני שחקנים.
המשחק מתנהל לפי תורות: כל שחקן בתורו לוקח מספר
אחד מהקצה הימני או השמאלי של הסדרה.
המטרה הכללית של כל שחקן: להגיע להפרש הגדול ביותר בין סכום המספרים
שיצבור עד סוף המשחק (עד שייגמרו כל המספרים) לבין הסכום אותו יצבור
השחקן השני.

דוגמה של משחק:
הסדרה: **2, 8 ,7, 10, 4**.
שחקן **1**: **2,7,4**, נקודות שנצברו – **13**, ההפרש בין הסכומים = **-5**.
שחקן **2**: **8,10**, נקודות שנצברו – **18**, ההפרש בין הסכומים = **5**.

יש למצוא את האסטרטגיה האופטימאלית למשחק זה וליישם אותה.

ב) נתונה סדרת מספרים חיוביים הנמצאים **על המעגל**: $a_1, a_2, \ldots, a_i, \ldots, a_n$.
במשחק משתתפים **2** שחקנים. בתחילת המשחק השחקן הראשון לוקח מספר
כלשהו (נגיד a_i) מתוך המעגל. אחרי הפעולה הזאת נשארת סדרה רגילה הבאה:
$a_{i+1}, \ldots, a_n, a_1, \ldots, a_{i-1}$ ומנקודה זו מתנהל המשחק לפי המתואר בסעיף א'.

פתרון:
א. נשתמש בתכנות דינאמי: $f(i,j)$ = ההפרש הכי טוב בין השחקן הראשון (זה שמתחיל) לשני כאשר תת המערך מתחיל במיקום i ומסתיים במיקום j. כאשר: $i \le j$.
אתחול: $f(i,i) = a[i]$.
הפונקציה: $f(i,j) = max(a[i] - f(i+1,j),\ a[j] - f(i,j-1))$.
דוגמא:

3	6	1	4

מכאן:

3	3	-2	6
	6	5	3
		1	3
			4

סיבוכיות: $O(n^2)$ - מילוי המטריצה.

ב. פונקציית המטרה: $ans = max_i(a[i] - f(i+1, i-1))$
קוד:

```
public static int numberGame(int[] arr) {
		int n = arr.length;
		int[][] m = new int[n][n];
		for (int i = 0; i < m.length; i++) {
			m[i][i] = arr[i];
		}
		for (int i = n-1; i >= 0; i--) {
			for (int j = i+1; j < n; j++) {
				m[i][j] = Math.max(arr[i] - m[i+1][j], arr[j] - m[i][j-1]);
			}
		}
		for (int i = 0; i < m.length; i++) {
			System.out.println(Arrays.toString(m[i]));
		}
		return m[0][n-1];
	}

	public static int numberGameCycle(int[] arr) { // O(n^3)
		int n = arr.length;
		int max = Integer.MIN_VALUE;
		for (int i = 0; i < n; i++) {
			int a = arr[i];
			int[] b = new int[n-1];
			int k = (i+1) % n;
			for (int j = 0; j < n-1; j++) {
				b[j] = arr[k];
				k = (k+1) % n;
			}
			int f = numberGame(b);
			if(a-f > max) max = a-f;
		}
		return max;
	}
```

שאלה 4
ממשו אלגוריתם אופטימאלי עבור משחק המספרים עם 2 מערכים כאשר בכל שלב מותר לקחת איבר מהצד השמאלי או הימני של המערך הראשון או השני.

פתרון:
נגדיר את הפונקציה: $f(a,b,x,y)$ הרווח (ההפרש) של השחקן הראשון כאשר המערך הראשון הוא ממקום a עד b והמערך השני הוא ממקום x עד y.

$$f(a,b,x,y) = max(A[a] - f(a+1,b,x,y),\ A[b] - f(a,b-1,x,y),\ B[x] - f(a,b,x+1,y),\ B[y] - f(a,b,x,y-1))$$

אם $a = b$ אז: $f(a,b,x,y) = max(A[a] - g(x,y), B[x] - f(a,b,x+1,y), B[y] - f(a,b,x,y-1))$

אם $x = y$ אז: $f(a,b,x,y) = max(A[a] - f(a+1,b,x,y), A[b] - f(a,b-1,x,y), B[x] - g(a,b))$

אם $x = y, a = b$ אז: $f(a,b,x,y) = max(A[a] - B[x], B[x] - A[a])$.

סיבוכיות:בניית המערך: $O(n^2m^2)$ כאשר n,m הם גדלי המערכים A,B.

שאלה 5

ממשו את בעיית המטוס כאשר נתון "שטח מת" = שטח שלא ניתן לעבור בו המוגדר ע"י 2 נקודות: $p_1=(x_1,y_1),\ p_2=(x_2,y_2)$ (2 קצוות נגדיות של מלבן כאשר p_1 היא הנקודה השמאלית התחתונה (בבעיית המטוס), p_2 היא הנקודה הימנית העליונה).

פתרון:

עבור כל מעבר היוצא מנקודה מ"השטח המת" להפוך אותו למחיר אינסוף. ואז להריץ את האלגוריתם הרגיל.

סיבוכיות: $O(nm)$ עבור שינוי הקלט + $O(nm)$ עבור האלגוריתם הרגיל.

קוד:

```
class DoubleNode {
	double price,x,y;

	public DoubleNode(double x, double y) {
		this.x = x;
		this.y = y;
	}
}
public class Q5 {

	public static int minPriceWuthDeadArea(Node[][] mat, Point p1, Point p2) {
		DoubleNode[][] mat2 = createNewMatrix(mat, p1, p2);
		return minPrice(mat2);
	}

	private static DoubleNode[][] createNewMatrix(Node[][] mat, Point p1, Point p2) {
		int n = mat.length;
		int m = mat[0].length;
		DoubleNode[][] newMat = new DoubleNode[n][m];
		for (int i = 0; i < n; i++) {
			for (int j = 0; j < m; j++) {
				if(i >= p1.y && i <= p2.y && j >=p1.x && j <= p2.x) {
					newMat[i][j] = new
DoubleNode(Double.POSITIVE_INFINITY, Double.POSITIVE_INFINITY);
				}
				else {
					newMat[i][j] = new DoubleNode(mat[i][j].x, mat[i][j].y);
				}
			}
		}
		return newMat;
	}

	public static int minPrice(DoubleNode[][] mat) {
		int n = mat.length, m = mat[0].length;
		mat[0][0].price = 0;
		for (int i = 1; i < n; i++) { mat[i][0].price = mat[i-1][0].price + mat[i-1][0].y;}
		for (int i = 1; i < m; i++) { mat[0][i].price = mat[0][i-1].price + mat[0][i-1].x;}
```

```
			for (int i = 1; i < n; i++) {
				for (int j = 1; j < m; j++) {
					mat[i][j].price = Math.min(mat[i-1][j].price + mat[i-1][j].y,
mat[i][j-1].price + mat[i][j-1].x);
				}
			}
			if(mat[n-1][m-1].price == Double.POSITIVE_INFINITY) return -1;
			return (int) mat[n-1][m-1].price;
		}
}
```

שאלה 6

כתבו אלגוריתם המקבל מערך ומחזיר את אורך תת הסדרה הארוכה ביותר כך שכל איבר הוא ריבוע של האיבר שלפניו או שורש של האיבר שלפניו.

לדוגמא: $2,\ 7,\ 1,\ 49,\ 3,\ 4,\ 1,\ 9,\ 2,\ 30,\ 81$

התשובה תהיה: $3, 9, 81$.

פתרון:

נשתמש בתכנות דינאמי: נגדיר: $f(i)$ = אורך הסדרה החוקית הארוכה ביותר כך שהאיבר ה i הוא האחרון בה. מכאן: $f(i) = max_{j<i}(f(j) + 1\ if\ *,\ 1\ else)$ כאשר $*$ אומר ש $a[j]^2 = a[i]$ או $a[i]^2 = a[j]$.

לדוגמא:

1	1	1	2	1	2	2	2	3	1	3

סיבוכיות: $O(n^2)$ כי על כל איבר עוברים על כל אלה שלפניו ולוקחים את המקסימום.

איך משחזרים את הסדרה?

חוזרים מהמקסימום ובודקים איזה איבר נתן אותו ע"י מעבר על כל הקודמים.