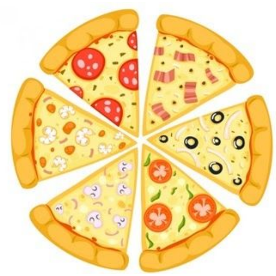
דור עזריה

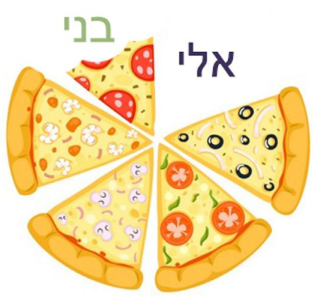
בעיית הפיצה

1) נתבונן באיור הבא:
אלי ובני הזמינו פיצה. אלי מחלק את הפיצה ל6 חלקים שווים. אלי אוכל 2 משולשים בזמן שבני אוכל משולש 1.

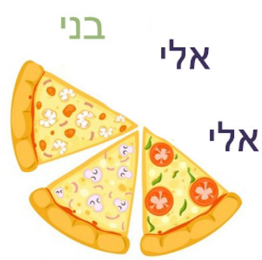


המטרה: שאלי יאכל כמה שיותר משולשים.

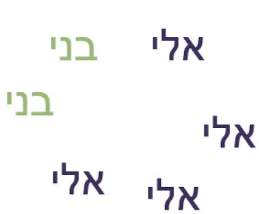
2) כל אחד לוקח משולש, אלי כבר סיים את המשולש שלו ובני רק בחצי של המשולש הראשון.



3) אלי לוקח משולש נוסף בזמן שבני עדיין עם הראשון, ושניהם מסיימים כל אחד משולש ביחד.



4) כעת קל לראות למבט הלאה (וממבט ראשון) כי אלי כמובן יאכל פי 2 משולשים ממה שבני אכל.



לכן עבור מקרה כללי:

תיאור הבעיה

- מהירות האכילה של אלי גדולה **פי-X** ממהירות האכילה של בני.
- ניתן לחלק את הפיצה ל-N משולשים **שווים**.
- במהלך הארוחה כל אחד לוקח משולש נוסף לאחר שסיים את הקודם.
- אסור ששניהם יגיעו אל המשולש האחרון בו זמנית!.
- יש למצוא את החלוקה האופטימלית כך שאלי יאכל כמה שיותר משולשי פיצה.

נשים לב שיש מקרים בהם אלי לא ינצח כמו למשל אם הוא יחלק את הפיצה ל- ½ .

וגם אם נחלק למשל את הפיצה ל-¼ , נגיע למצב שרבים על המשולש האחרון וזה אסור לפי תיאור הבעיה.
אם נמשיך לחלק את הפיצה לכל מיני חלקים אנחנו נגלה כי מסתתרת פה חוקיות מסוימת.
בעצם, ביצענו כאן מעין מחקר קטן שבעזרתו גילינו מה החלוקה הטובה ביותר עבור אלי.

בהסתמך על הדוגמאות שראינו, אז אם מספר המשולשים הוא $N = 3$ ואלי אוכל פי $X = 2$,
קל וברור שמספר המשולשים N צריך להיות גדול או שווה ל-X+1 .

נוכיח כי $N \geq X + 1$.

הוכחה

- ננית שהחלוקה האופטימלית היא N=X (כלומר, כמות משולשי הפיצה שווה למהירות האכילה של אלי).
  במקרה זה, אלי יאכל $\frac{N-1}{N}$ משולשים, ובני יאכל $\frac{1}{N}$ משולשים, ואז הם לא יסיימו בו זמנית.
  (למשל X=2 אז הפיצה מחולקת ל-½ , אלי אכל את המשולש שלו בזמן שבני אכל רק חצי מהמשולש שלו).

- אם נחלק ל - $N = X + 1$ חלקים, אז אלי יאכל $\frac{N}{N+1}$ ובני יאכל $\frac{1}{N+1}$ ואז הם סיימו בו זמנית.

צריך להוכיח ש: $\frac{N-1}{N} < \frac{N}{N+1}$,

$$\frac{N-1}{N} < \frac{N}{N+1} \quad \backslash \cdot N \cdot (N+1)$$
$$(N-1)\cdot(N+1) < N^2$$
$$N^2 - 1 < N^2$$

אי השיוויון מתקיים ולכן עדיף לחלק ל- X+1 חלקים.

∎

אך אנחנו רוצים יותר מזה, ונרצה להימנע ממקרה שבו שניהם מגיעים למשולש האחרון בו זמנית.
כאשר X+1 הוא מספר המשולשים שהם מסיימים לאכול בו זמנית, ו-P היא הכפולה (כמות הסיבובים),
אז מספר החלקים N צריך להיות $N \neq (X+1)\cdot P + 1$ כדי להימנע מהמצב שבו שניהם מגיעים למשולש האחרון בו זמנית. (הם אוכלים יחד $(X+1)\cdot P$ ואז ישאר המשולש האחרון שזה ה- (+1) שנשאר הם ילחמו על המשולש הזה וזה אסור לפי תיאור הבעיה).

הוכחה

נניח כי $N = (X+1)\cdot P + r$ כאשר r הוא השארית ו- $2 \le r \le X$.
במקרה זה, אלי יאכל $\frac{X\cdot P + r - 1}{(X+1)\cdot P + r}$ מהפיצה ובני יאכל $\frac{P+1}{(X+1)\cdot P + r}$ מהפיצה. (אלי אוכל את השארית r כי הוא יגיע אליה לפני שבני יספיק לסיים).

נוכיח שאלי צריך לחלק את הפיצה ל-X+1 משולשים, כלומר צ"ל כי - $\frac{X}{X+1} < \frac{X\cdot P + r - 1}{(X+1)\cdot P + r}$.

$$\frac{X\cdot P + r - 1}{(X+1)\cdot P + r} < \frac{X}{X+1} \quad \backslash \cdot [\,(X+1)\cdot P + r\,]\cdot[X+1]$$
$$(X\cdot P + r - 1)\cdot(X+1) < X\cdot((X+1)\cdot P + r)$$
$$X^2\cdot P + XP + Xr + r - X - 1 < X^2\cdot P + XP + Xr$$
$$r < X + 1$$

השארית קטנה מהחלוקה ולכן מתקיים והם לא יגיעו בו זמנית למשולש האחרון.
הפונקציה היא $f(X) = X + 1$ וזוהי החלוקה האופטימלית.

∎

מימוש הבעיה, מחזיר true אם הפרמטרים מאפשרים מצב אופטימלי בו אלי יאכל יותר משולשים מבני ולא יקרה מצב שנשאר משולש אחד ואחרון לשניהם.

```java
public static boolean pizza(double X , int N) {
   int F = (int)X + 1 ;
   int P =  N / (F + 1);
   int r = N % (F+1);

   if(2 <= r && r <= (int)X) {
       double t = (X*P + r - 1) / ((X+1)*P + r);
       if(t < X/(X+1) ) {
           return true;
       }
   }
   return false;
}

public static void main(String[] args) {
   System.out.println(pizza(2,6)); // true
   System.out.println(pizza(2,4)); // false
}
```

בעיית המזכירה

תיאור הבעיה

- משרד מסוים נותן שירות ל-n לקוחות, מטרת מזכירת המשרד היא להקטין ככל האפשר את זמן הממוצע שהלקוחות נמצאים במשרד.
- הזמן שהלקוח נמצא במשרד מורכב מזמן ההמתנה שלו עד תורו יחד עם זמן הטיפול שלו.
- ספויילר: המצב הטוב ביותר הוא כאשר זמני הטיפול של הלקוחות נמצאים בסדר עולה.

ניתוח

נסמן i כלקוח , לכן $1 \leq i \leq n$ כך ש- ti - הוא זמן הטיפול של אחד מ-n הלקוחות.
Ti - הזמן שהלקוח נמצא במשרד. (זמן ההמתנה + זמן הטיפול).
לכן,

$$T_1 = t_1$$

(כי הלקוח הראשון לא צריך להמתין בתור, אז נציין רק את זמן הטיפול שלו).

$$T_2 = t_1 + t_2$$

(כי עבור הלקוח השני, זמן ההמתנה הוא הזמן של הטיפול ללקוח הראשון + זמן הטיפול שלו)

$$....$$

$$T_n = t_1 + t_2 + ... + t_n$$

על המזכירה למצוא הממוצע של זמן ההמתנה המינימלי ביותר: $Average = \frac{T_1 + T_2 + ...+ T_n}{n}$.
מספיק לחשב $min(T_1 + T_2 + ... + T_n)$, כי ה-n במכנה הוא גם ככה מספר קבוע.

דוגמה

עבור: $t_1 = 10 , t_2 = 1, t_3 = 8$.

תור הלקוחות	זמן המתנה ממוצע
(t_1, t_2, t_3)	$(10) + (1 + 10) + (1 + 10 + 8) = 40$
(t_1, t_3, t_2)	$(10) + (10 + 8) + (10 + 8 + 1) = 47$
(t_2, t_1, t_3)	$(1) + (1 + 10) + (1 + 10 + 8) = 31$
(t_2, t_3, t_1)	$(1) + (1 + 8) + (1 + 8 + 10) = 29$
(t_3, t_1, t_2)	$(8) + (8 + 10) + (8 + 10 + 1) = 45$
(t_3, t_2, t_1)	$(8) + (8 + 1) + (8 + 1 + 10) = 37$

פתרון

- עבור חיפוש שלם צריך לעבור על כל האפשרויות לכן $n! > 2^n$ וזה ממש לא יעיל.
- נשים לב כי בדוגמה שהצגנו וסימנו בכחול, התשובה הטובה ביותר מתקבלת כאשר מערך של זמני הטיפול ממוין מקטן לגדול.

זמן ההמתנה הכולל הוא:

$$sum = T_1 + ... + T_i + T_{i+1} + ... + T_n = t_1 + ... + (t_1 + ... + t_i) + (t_1 + ... + t_i + t_{i+1}) + ... \\ + (t_1 + ... + t_i + t_{i+1} + ... + t_n)$$

נוכיח את הטענה בדרך השלילה:

נניח שזמן ההמתנה ti (איבר כלשהו במערך לא ממויין) גדול יותר מזמן ההמתנה של הבא בתור, $t_i > t_{i+1}$,

אם נחליף ביניהם $t_{i+1} \leftrightarrow t_i$ נקבל :

$$sum' = T_1 + ... + T_i' + T_{i+1} + ... + T_n = t_1 + ... + (t_1 + ... + t_{i+1}) + (t_1 + ... + t_{i+1} + t_i) + ... \\ + (t_1 + ... + t_i + t_{i+1} + ... + t_n)$$

ונקבל:

$$sum - sum' = t_i - t_{i+1} > 0$$

ומכאן נקבל שהזמן הממוצע קטן יותר כאשר האיברים (זמני הטיפול) הסמוכים נמצאים בסדר עולה.
המסקנה היא שעל המזכירה למיין את מערך זמני הטיפול מהקצר לארוך והתאמה לקבוע את התור.

מימוש:

```java
public static double getAverageTime(int[] times) {
   Arrays.sort(times);
   double avg = 0;

   for(int i = 0 ; i < times.length ; i++){
       double temp_avg = 0;

       for(int j = 0 ; j <= i ; j++) {
           temp_avg += times[j];
       }
       avg += temp_avg;
   }
   return avg/times.length;
}

public static void main(String[] args) {
   System.out.println(getAverageTime(new int[]{10,1,8}));
}
```

בעיית החציון

תיאור הבעיה

- בהינתן מערך לא ממוין של מספרים אקראיים, יש למצוא את איבר שהוא גדול מהחציון של המערך.
- כלומר, מספר שחצי מאיברי המערך גדולים ממנו וחצי מאיברי המערך קטנים ממנו.

הגדרה

חציון (**median**) הוא מספר (לא בהכרח איבר המערך) שמחצית מאיברי המערך גדולים ממנו ומחצית מאיברי המערך קטנים ממנו.
כאשר מספר איברי המערך אי-זוגי – החציון הוא איבר שנמצא באמצע המערך (**ממוין** !)
כאשר מספר איברי המערך זוגי – החציון שווה לממוצע של שני איברים הנמצאים באמצע המערך (**ממוין** !).

$$\mathrm{Med}(X) = \begin{cases} \frac{X[\frac{n}{2}] + X[\frac{n+1}{2}]}{2} & \text{if n is even} \\ X[\frac{n+1}{2}] & \text{if n is odd} \end{cases}$$

דוגמה

עבור אורך מערך אי-זוגי: median=8   arr[]={1,3,6,**8**,12,23,77},
עבור אורך מערך זוגי: median=(6+12)/2=9   arr[]={1,3,**6**,**12**,23,77},

פתרון הבעיה

- אפשרות א' - פתרון נאיבי
  - נמיין את המערך בסיבוכיות $O(n \cdot (log\ n))$.
  - נשלוף את האיבר האחרון במערך במיקום $arr[length - 1]$ כי הוא בהכרח גדול מהחציון של כל איברי המערך - לקיחת האיבר - $O(1)$.

```java
public static int naive(int[] arr) {
   Arrays.sort(arr); // O(n*log n)
   return arr[arr.length-1]; // O(1)
}
```

סיבוכיות זמן הריצה: $O(n \cdot (log\ n)\ + 1)$.

❖ אפשרות ב' - אופטימלי

⇚ אם ניקח את האיבר הראשון במערך, ההסתברות שהוא גדול מהחציון הוא 50%.

⇚ אם ניקח את שני האיברים הראשונים במערך :

(0 = מחצית שמאלית, 1= מחצית ימנית)	$a[1]$	$a[0]$
שניהם נמצאים במחצית השמאלית של המערך	0	0
$a[0]$ בשמאלית, $a[1]$ בימנית ולכן $a[0] < a[1]$	1	0
$a[0]$ בימנית , $a[1]$ בשמאלית ולכן $a[0] > a[1]$	0	1
שניהם נמצאים במחצית הימנית של המערך	1	1

לכן ההסתברות שאחד האיברים $a[0]$ או $a[1]$ יהיו במחצית הימנית (כלומר גדולים מהחציון) היא $\frac{3}{4}$ (75%).
לכן נבחר את המקסימום כדי להבטיח זאת $max(\ a[0],\ a[1]\ )$.

⇚ אם ניקח את שלושת האיברים הראשונים של המערך, ההסתברות שהמקסימום מבניהם יהיה גדול מהחציון היא הסתברות $\frac{7}{8}$.

⇚ אם ניקח את 64 האיברים הראשונים של המערך, ההסתברות שהגדול מביניהם נמצא במחצית הימנית של המערך היא (מספר כל תתי הקבוצות של קבוצה בת 64 איברים כלומר 2^{64}).
כלומר $1 = 1 - \frac{1}{64} = \frac{63}{64}$, לכן מספיק לחשב מקסימום של 64 איברים ראשונים של המערך ואנו נקבל איבר שגדול מהחציון. הסיבוכיות היא $O(1)$ מכיוון שאין חשיבות למספר 64, אלא המספר צריך להיות גדול מספיק על מנת שישאף ל-0.

```java
public static int optimal(int[] arr) {
   int max = arr[0];

   for(int i = 1; i < arr.length-1 && i < 64 -1; i+=2) {

       if(arr[i] > arr[i+1] && max < arr[i]) {
           max = arr[i];
       }
       else if(max < arr[i+1]) {
           max = arr[i+1];
       }
   }

   return max;
}
```