

## אלגוריתמים – פתרון תרגיל 5

### 1. תאור האלגוריתם:

נגדיר מדי  $G = g_1, g_2, \dots, g_n$  אשר יסמל את הרווח ליחידת משקל. נחשב ערכי  $G$  עבור כל אחד מהאובייקטים על ידי חלוקה של הרווח הנתון במשקל של האובייקט. כלומר  $g_i = p_i/w_i$ . יהיה מדי  $G$  עבור האובייקט  $o_i$ .

לאחר חישוב המדי לכל אחד מהאובייקטים, נמיין את האובייקטים בסדר יורד ע"פ המדי שחישבנו. עתה נפעיל אלגוריתם "חמדן": נשמור את מגבלת המשקל הכולל במשתנה  $T$ . כל עוד  $T > 0$  (ועדיין קיימים אובייקטים שלא נבחרו), נבחר את האובייקט עם מדי  $g_i$  הגדול ביותר. אם  $W_i < T$  נבחר את אובייקט  $o_i$  ונעדכן את  $T$  להיות  $T - W_i$ . אחרת, נבחר את החלק היחסי  $c$  של  $o_i$  כך ש- $a \cdot w_i = T$ .

בסיום הלולאה, נחזיר את רשימת האובייקטים שבחרנו.

### הוכחת נכונות:

מגבלת המשקל – הלולאה הבוחרת את האובייקטים רצה כל עוד לא נוצל המשקל המקסימלי. עבור הבחירה האחרונה, דואגת הלולאה לקחת את החלק היחסי של האובייקט האחרון כך שמגבלת המשקל לא תופר. לכן, האלגוריתם שומר על מגבלת המשקל המבוקשת.

רווח מקסימלי – נניח בשלילה שהאלגוריתם לא מחזיר את הרווח המקסימלי. כלומר, קיים אובייקט  $o_k$  כך שאם נבחר אותו במקום אחד האובייקטים האחרים שבחרנו  $o_l$  (באופן יחסי במשקלים), הרווח מהאובייקטים שנבחרו יגדל. כלומר  $\frac{p_k}{w_k} > \frac{p_l}{w_l}$ . אבל הלולאה רצה על האובייקטים ע"פ מדי  $G$  בסדר יורד ולכן אם היה קיים  $o_k$  כזה הוא היה נבחר לפני שהגענו אל  $o_l$  וזאת בסתירה להנחה שלא בחרנו את  $o_k$ .

### ניתוח סבוכיות:

א. חישוב מדי  $G$  עבור כל אחד מהאובייקטים –  $O(n)$

ב. מיון האובייקטים ע"פ מדי  $G$  –  $O(n \log n)$

ג. לולאת בחירת האובייקטים –  $O(n)$

סה"כ סבוכיות זמן הריצה –  $O(n \log n)$

■ מ.ש.ל

### 2.

א. הראה שהמימוש המצורף לפונקציה Fun מבוצעת בסבוכיות זמן ריצה  $O(2^n)$ .

### הוכחה:

נוכיח באינדוקציה.

שלב א: שלב הבדיקה

עבור  $n=0$ , הפונקציה Fun מבצעת בדיקת IF יחידה ומחזיר 0. סה"כ סבוכיות זמן ריצה במקרה זה  $O(1) = O(2^0)$ .

שלב ב: שלב המעבר

נניח שהטענה נכונה עבור  $n$  ונוכיח עבור  $n+1$ :  
 $n+1 \neq 0$  ולכן מבוצעת הלולאה  $n+1$  פעמים ועבור כל  $0 \leq i < n+1$  מחשבת את Fun(i).  
לכן מתקיים כי  $T(n+1) = T(n) + T(n-1) + \dots + T(0)$ . בצורה דומה, עבור  $n$  מתקיים כי  $T(n) = T(n-1) + \dots + T(0)$ . נציב ונקבל  $T(n+1) = T(n) + T(n) = 2T(n)$ . ע"פ הנחת האינדוקציה  $T(n) = O(2^n)$  ומכאן כי  $T(n+1) = O(2^{n+1}) = O(2 \cdot 2^n)$ .

■ מ.ש.ל

ב. ממש אלגוריתם תכנות דינאמי המשתמש במערך בגודל  $n$  ומחשב את הפונקציה בסבכויות זמן ריצה  $O(n^2)$ .

### אלגוריתם:

נגדיר מערך  $A$  באורך  $n$ . במערך זה, נשמור בכל תא  $i$  את הערך של  $Fun(i)$  באופן הבא:

i. נאתחל את כל התאים במערך להיות 0 (בעיקרון, ניתן לוותר על שלב זה):

```
for (int i=0; i<n; i++)  
{  
    A[i]=0;  
}
```

ii. נחשב את ערך הפונקציה:

```
sum = 0;  
  
for (int i=0; i<n; i++)  
{  
    A[i] = i * sum + i;  
    sum = sum + a[i];  
}  
  
if (n==0)  
{  
    return 0;  
}  
else  
{  
    return n * sum + n;  
}
```

### הסבר החישוב:

פונקציה  $Fun(n)$  המקורית, מבצעת סיכום של כל ערכי  $Fun(i)$  עבור כל  $0 \leq i < n$  ואת סכום זה מכפילה ב- $n$  ומוסיפה  $n$ . כיוון ש- $Fun$  המקורית כתובה בצורה רקורסיבית, היא מבצעת חישובים רבים של אותם ערכים (לדוגמה את  $F(0)$  היא מחשבת  $n$  פעמים,  $F(1)$  מחושב  $n-1$  וכ'').

לאחר השינוי, הפונקציה החדשה מחשבת את  $Fun$  מ-0 עד  $n$  ובכל שלב שומרת את התוצאה במערך. כך,  $Fun(i)$  יחושב פעם אחת בלבד עבור כל  $0 \leq i < n$ . בכל איטרציה של הלולאה, הפונקציה החדשה משתמשת בסכום שנצבר עד כה כדי לחשב את  $Fun(i)$ . בסוף הלולאה,  $sum$  מכיל את  $Fun(n-1)$ . הפונקציה מכפילה ערך זה ב- $n$  ומוסיפה לו  $n$  וזהו למעשה ערך  $Fun(n)$ .

### ניתוח סבוכיות:

i. אתחול המערך (ניתן לוותר על שלב זה) –  $O(n)$

ii. חישוב הפונקציה מבוצע ע"י לולאה הרצה בדיוק  $n$  איטרציות ולכן –  $O(n)$

סה"כ סבוכיות זמן הריצה של האלגוריתם –  $O(n)$

■ מ.ש.ל

3. הצג אלגוריתם מבוסס תכנות דינאמי המחשב את הערך של הפונקציה הבאה

$$F(n) = \sum_{i=1}^{n-1} F(i)F(i-1) \text{ for } n > 1 \text{ with } f(0) = f(1) = 2$$

לכל ערך חיובי של  $n$  וחשב  $T(n)$ .

### אלגוריתם:

נגדיר מערך  $A$  באורך  $n$ . במערך זה, נשמור בכל תא  $i$  את הערך של  $F(i)$  באופן הבא:

א. נאתחל את כל התאים במערך להיות 0 (בעיקרון, ניתן לוותר על שלב זה):

```
for (int i=0; i<n; i++)  
{  
    A[i]=0;  
}
```

ב. נחשב את ערך הפונקציה:

```
for (int i=0; i<n; i++)  
{  
    if (i==0 || i==1)  
    {  
        A[i]=2;  
    }  
    else  
    {  
        A[i] = A[i-1]*A[i-2] + A[i-1];  
    }  
}
```

```
if (i==0 || i==1)  
{  
    return 2;  
}  
else if (i==2)  
{  
    return A[n-1]*A[n-2];  
}  
else  
{  
    return A[n-1]*A[n-2] + A[n-1];  
}
```

### הסבר החישוב:

פונקציה  $F(n)$  המקורית, מבצעת סיכום של מכפלת כל הזוגות העוקבים  $F(i)F(i-1)$  עבור כל  $0 < i < n$ . כיוון ש- $F$  המקורית כתובה בצורה רקורסיבית, היא מבצעת חישובים רבים של אותם ערכים.

לאחר השינוי, הפונקציה שכתבנו, מחשבת את  $F$  מ-0 עד  $n$  ובכל שלב שומרת את התוצאה במערך. כך,  $F(i)$  יחושב פעם אחת בלבד עבור כל  $0 \leq i < n$ . בכל איטרציה של הלולאה, הפונקציה החדשה משתמשת בערכים שחושבו עד כה ונשמרו במערך לחשב את  $F(i)$ . כך, אין צורך לחשב שוב ושוב את אותם ערכים וכל ערך מחושב פעם אחת בהסתמך על הערכים שחושבו קודם לכן ונשמרו במערך.

### ניתוח סבוכיות:

א. אתחול המערך (ניתן לוותר על שלב זה) –  $O(n)$

ב. חישוב הפונקציה מבוצע ע"י לולאה הרצה  $n$  איטרציות בדיוק ולכן –  $O(n)$

סה"כ סבוכיות זמן הריצה של האלגוריתם –  $O(n)$

■ מ.ש.ל