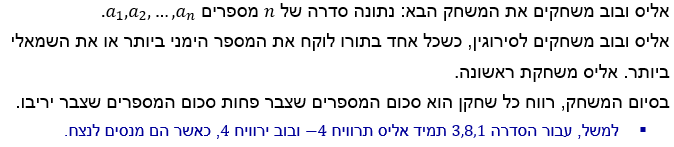
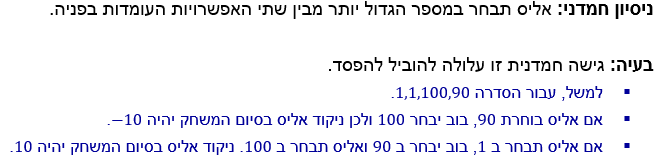
**משחק מספרים - איסטרטגיה אופטימאלית**



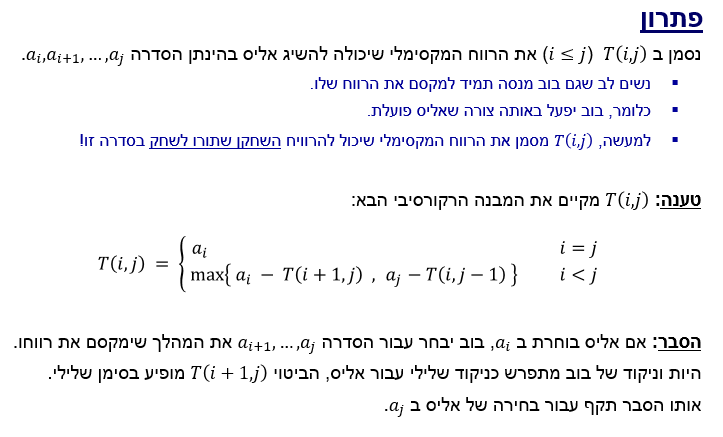


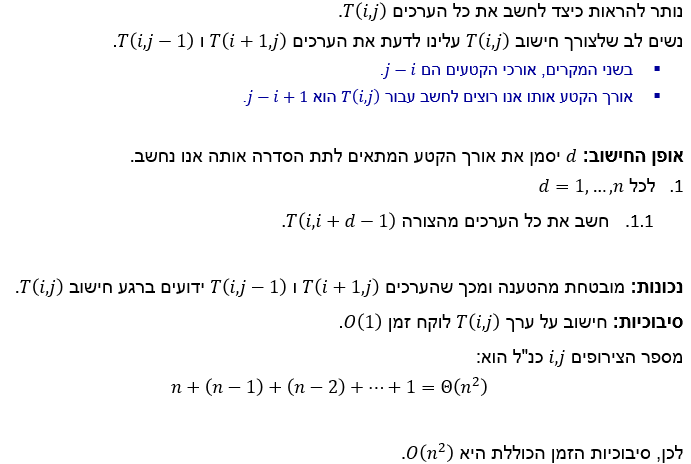


**אסטרטגיה של חיפוש שלם** מביאה לניצחון עם רווח מקסימלי אך דורשת הקמת עץ בינארי שלם, שגובו הוא n דהיינו מספר צמתים בעץ הוא . למשל, בסדרה של 10 מספרים מספר אפשרויות בוא .

איך אפשר לשפר את הביצוע? קודם כל נשים לב כי בחיפוש שלם יש הרבה חזרות. נשתמש בתכנון דינאמי ונתבונן רק בצמתים שונים. במקום עץ בינארי נשתמש במערך דו ממדי )מטריצה( המטרה שלנו היא צומת (1,n).

**פתרון אופטימלי:**

****

****

נבנה מערך דו-ממדי (מטריצה). מטרה שלנו למצוא רווח בקטע (1,n):



כמו בחיפוש שלם ממלאים את המטריצה מלמטה כלפי מעלה.

אתחול המטריצה: איברי הסדרה שמים באלכסון:

**דוגמה**:

ניקח סדרת המספרים הבאה: 1, 3, 6, 1, 3, 6

**בניית מטריצת הרווחים:**

T(𝑖,𝑗)=*max* (𝑎𝑖 − T(𝑖+1,𝑗), 𝑎𝑗 − T(𝑖,𝑗−1)) ,𝑖=𝑛−2,…0, 𝑗=𝑖+1,…,𝑛−1

פסדו-קוד לבניית המטריצה:

**buildMatrix(game[])**

n = game.length

mat[n,n]

for i=0 to n-1

mat[i,i] = game[i]

end-for

for i=n-2 to 0

for j=i+1 to n-1

mat[i,j]=*max*(game[i]-mat[i+1,j],game[j]-mat[i,j-1])

end-for

end-for

return mat

**end-buildMatrix**

נמלא את המטריצה:

𝑖=4,𝑗=5, T(4,5) = max(𝑎4−T(5,5), 𝑎5−T(4,4)) = max(3−6,6−3)=3

𝑖=3,𝑗=4, T(3,4)=max(𝑎3−T(4,4), 𝑎4−T(3,3)) = max(1−3,3−1)=2

𝑖=3,𝑗=5, T(3,5)=max(𝑎3−T(4,5),𝑎5−T(3,4))= max(1−3,6−2)=4

וכן הלאה.



הרווח הגדול ביותר של שחקן הראשון A: T(0,5)=6

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 𝑎5 | 𝑎4 | 𝑎3 | 𝑎2 | 𝑎1 | 𝑎0 |
| 6 | 3 | 1 | 6 | 3 | 1 |

**בניית האסטרטגיה של השחקנים:**

1. הרווח המקסימאלי של A מתקבל במצב (0,5) :

T(0,5) = max(𝑎0−T(1,5), 𝑎5−T(0,4)) = max(1−1,6−0)=6

המקסימום מתקבל במצב 𝑎5 כלומר A צריך לקחת את 𝒂𝟓=𝟔 הסכום שלו sumA=6

ל B-נשאר לבחור בין 𝑎0 לבין :𝑎4

T(0,4)=max(𝑎0−T(1,4),𝑎4−T(0,3))= max(1−5,3−3)=0

המקסימום מתקבל במצב 𝑎4 לכן B אמור לקחת 𝒂𝟒=𝟑 הסכום שלו sumB = 3

1. ל- A יש לבחור בין 𝑎0 לבין 𝑎3

T(0,3)=max(𝑎0−T(1,3),𝑎3−T(0,2))= max(1+2,6−4)=3

המקסימום מתקבל במצב 𝑎0 כלומר A צריך לקחת את 𝒂𝟎=𝟏 הסכום שלו sumA = 6+1=7

ל B-נשאר לבחור בין 𝑎1 לבין 𝑎3

T(1,3)=max(𝑎1−T(1,4),𝑎3−T(1,2))= max(3−5,6−3)=3

המקסימום מתקבל במצב 𝑎3 לכן B אמור לקחת 𝒂𝟑=𝟏 הסכום שלו sumB=3+1 = 4

1. ל- A יש לבחור בין 𝑎1 לבין 𝑎2

T(1,2)=max(𝑎1−T(2,2),𝑎2−T(1,1))= max(1−6,6−3)=3

המקסימום מתקבל במצב 𝑎2 כלומר A צריך לקחת את 𝒂𝟐=𝟔 הסכום שלו sumA = 7+6=13

ל B-נשאר 𝒂𝟏=𝟑 הסכום שלו sumB=4+3 = 7

**הרווח של** A **הוא 13-7=6**

**סיכום המשחק:**

|  |  |  |
| --- | --- | --- |
| STEP | B | A |
| 1 | 𝑎4=3 | 𝑎5=6 |
| 2 | 𝑎3=1 | 𝑎0=1 |
| 3 | 𝑎1=3 | 𝑎2=6 |
| SUM | 7 | 13 |

**פסדו-קוד של משחק:**

**gameStrategy(game[])**

i=0, n = game.length, j = n - 1

first=0, second=0, firstSum = 0, secondSum = 0

mat[][] = *buildMatrix*(game)

**for** k = 0 to n/2

// the first takes a number

**if** (game[i]-mat[i+1][j] > game[j] - mat[i][j-1])

firstSum = firstSum + game[i]

first = i++

**else**

firstSum = firstSum + game[j]

first = j--

**end-if**

// the second takes a number

**if** (i != j) //j>0 && i<n-1

**if** (game[i]- mat[i+1][j] > game[j] - mat[i][j-1])

secondSum = secondSum + game[i]

second = i++

**else**

secondSum = secondSum + game[j]

second = j--

**end-if**

**else** //j=0 or i=n-1, the second takes the last element

second = j

secondSum = secondSum + game[j]

**end-if**

*print*("first: game["+first+"] = "+game[first]+

", second: game["+second+"] = "+game[second])

**end-for**

**print**("firstSum = "+firstSum+", secondSum = "+secondSum+ ", diff = "+(firstSum-secondSum))

**end-gameStrategy**