

מטלה 1- תכנון דינאמי:

שאלה 1:

א. נוסחה רקורסיבית:

$$\text{MaxSum}(i, j) = \begin{cases} 0 & \text{if } i < 1 \text{ or } i > N \text{ or } j < 1 \text{ or } j > N \\ A[i][j] & \text{if } i = N \\ A[i][j] + \max(\text{MaxSum}(i + 1, j), \text{MaxSum}(i + 1, j + 1)) & \text{else } (i < N) \end{cases}$$

פירוט הפרמטרים בנוסחה:

A - מערך דו ממדי, מערך הערכים

i - מיקום השורה במערך A

j - מיקום העמודה במערך A

N - גודל השורה והעמודה

הסבר הפלט:

הנוסחה MaxSum מייצגת פתרון רקורסיבי לבעיה של מציאת המסלול עם סכום מקסימלי, הפלט הינו הערך של המסלול היקר ביותר.

נכונות הנוסחה:

- אם $i < 1$ or $i > N$ or $j < 1$ or $j > N$ אז ישנה חריגה מהמטריצה ולכן נחזיר 0.
- אם $i = N$: (השורה האחרונה במטריצה), הנוסחה תחזיר את $A[i][j]$. זהו תנאי העצירה של הנוסחה.
- אם לא: מתבצעות 2 קריאות רקורסיביות עבור 2 האופציות שהוגדרו בשאלה כ- "צעד במסלול" ('צעד במסלול ממשבצת $[i, j]$ יכולה להיות ל- $[i+1, j]$ או $[i+1, j+1]$ '), ונבדק המקסימום מבניהם- נערכת השוואה $A[i][j] +$ הנוסחה בודקת את כל האפשרויות עבור הצעד הבא ומחזירה את התשובה הטובה ביותר.

פונקציה: (N הוגדר כ-#define)

```
int MaxSum2(int i, int j, int A[N][N]) {  
    if (i < 0 || i > N-1 || j < 0 || j > N-1)  
        return 0;  
    if (i == N - 1)  
        return A[i][j];  
    return A[i][j] + max(MaxSum2(i + 1, j, A), MaxSum2(i + 1, j + 1, A));  
}
```

ב. סיבוכיות הזמן והמקום של אלגוריתם תכנון דינמי המוצא את המסלול שערכו מקסימלי:

סיבוכיות הזמן - לאלגוריתם מספר לולאות שרצות עד N (סדר גודל זמן ריצה של $O(N)$), אך הלולאה עם זמן הריצה הגבוהה ביותר היא הלולאה המקוננת שרצה בסה"כ בסדר גודל של $O(N^2)$. מכיוון שסיבוכיות זמן נמדדת בהתאם לגודל הקלט, ובמקרה הזה, גודל הקלט הוא N^2 והאלגוריתם עובר גם הוא על N^2 נתונים, ניתן להגיד כי הסיבוכיות הינה ליניארית. נסמן את גודל הקלט m בצורה הבאה - $m = N^2$, נקבל כי סיבוכיות זמן הריצה יוצאת $O(m)$, כלומר ליניארית **ביחס לגודל בקלט**.

סיבוכיות המקום יצרתי מטריצה בשם S בגודל $N \times N$ שתפקידה הוא לאחסן את הערכים בטבלה הדינאמית. כיוון שאת סיבוכיות הזמן ביטאתי בעזרת m , שהוא גודל הקלט ($m = N^2$) כך אעשה גם עם סיבוכיות המקום. לכן, סיבוכיות המקום גם היא $O(m)$, כלומר ליניארית ונמדדת **ביחס לגודל הקלט** שהוא N^2 .

פונקציה: N הוגדר כ- $\#define$ -)

```
void findMaxPath(int A[N][N]) {
    int** S = (int**)malloc(N * sizeof(int*));
    for (int i = 0; i < N; i++)
        S[i] = (int*)malloc(N * sizeof(int));
    for (int j = 0; j < N; j++)
        S[N - 1][j] = A[N - 1][j];
    int currentCol = 0;
    for (int i = N - 2; i >= 0; i--) {
        for (int j = 0; j <= i; j++) {
            if (S[i + 1][j] > S[i + 1][j + 1])
                S[i][j] = A[i][j] + S[i + 1][j];
            else
                S[i][j] = A[i][j] + S[i + 1][j + 1];
        }
        if (S[i][currentCol] < S[i][currentCol + 1])
            currentCol++;
    }
    printf("Max Path: ");
    int i;
    printf("[0, 0] -> ");
    for (i = 1; i < N - 1; i++) {
        printf("[%d, %d] -> ", i, currentCol);
        if (currentCol < i + 1 && S[i + 1][currentCol] < S[i + 1][currentCol + 1]) {
            currentCol++;
        }
    }
    printf("[%d, %d]\n", i, currentCol);
    for (int i = 0; i < N; i++)
        free(S[i]);
    free(S);
}
```

ג. סיבוכיות הזמן והמקום של אלגוריתם תכנון דינמי המוצא את ערך המסלול הגדול ביותר:

סיבוכיות הזמן - מכיוון שסיבוכיות זמן נמדדת בהתאם לגודל הקלט, ובמקרה הזה, גודל הקלט הוא N^2 והאלגוריתם עובר גם הוא על N^2 נתונים (בלולאה מקוננת), ניתן להגיד כי הסיבוכיות הינה ליניארית. נסמן את גודל הקלט m בצורה הבאה - $m = N^2$, נקבל כי סיבוכיות זמן הריצה יוצאת $O(m)$, כלומר ליניארית **ביחס לגודל הקלט**.

סיבוכיות המקום יצרתי מערך בשם `res` בגודל N שתפקידו הוא לאחסן את הערכים המקסימליים במערך הדינמי. כיוון שאת סיבוכיות הזמן ביטאתי בעזרת m , שהוא גודל הקלט ($m = N^2$) כך אעשה גם עם סיבוכיות המקום. כיוון ש $m = N^2$, אז $N = \sqrt{m}$. לכן, סיבוכיות המקום הינה בסדר גודל של $O(\sqrt{m})$, כלומר סיבוכיות של שורש ריבועי ונמדדת **ביחס לגודל הקלט** שהוא N^2 .

פונקציה: (N הוגדר כ-#define)

```
int MaxSumDynamic(int A[N][N])
{
    //int* res= (int*)malloc(N * sizeof(int));
    int res[N];
    res[0] = A[0][0];
    for (int i = 1; i < N; i++) {
        int prev = res[0];
        res[0] = A[i][0] + res[0];
        for (int j = 1; j < i; j++) {
            int current = max(prev, res[j]);
            prev = res[j];
            res[j] = A[i][j] + current;
        }
        res[i] = A[i][i] + prev;
    }
    int maxSum = 0;
    for (int i = 0; i < N; i++) {
        if (res[i] > maxSum)
            maxSum = res[i];
    }
    //free(res);
    return maxSum;
}
```

שאלה 2:

א. מערך A בגודל 5×5 (אינדקס המערך נע בין 1 ל-5) בו קיים מסלול נחש ארוך ביותר המתחיל במיקום $A[2,3]$ ומסתיים במיקום $A[4,4]$.

1	20	30	40	50
10	15	2	100	200
20	20	2	800	700
30	80	3	2	600
40	2	10	80	500

ב. נוסחה רקורסיבית:

$T(i, j) =$

$$\begin{cases}
 0 & \text{if } i < 1 \text{ or } i > N \text{ or } j < 1 \text{ or } j > N \\
 1 & \text{if } ((i=1 \text{ and } j=1) \\
 & \text{or } (i > 1 \text{ and } j > 1 \text{ and } |A[i][j] - A[i-1][j]| > 1 \text{ and } |A[i][j] - A[i][j-1]| > 1) \\
 & \text{or } (i > 1 \text{ and } j=1 \text{ and } |A[i][j] - A[i-1][j]| > 1) \\
 & \text{or } (i=1 \text{ and } j > 1 \text{ and } |A[i][j] - A[i][j-1]| > 1)) \\
 T(i-1, j) + 1 & \text{if } i > 1 \text{ and } j \geq 1 \text{ and } |A[i][j] - A[i-1][j]| \leq 1 \\
 T(i, j-1) + 1 & \text{if } j > 1 \text{ and } i \geq 1 \text{ and } |A[i][j] - A[i][j-1]| \leq 1 \\
 \max(T(i-1, j), T(i, j-1)) + 1 & \text{else - (if } i > 1 \text{ and } j > 1 \text{ and } \\
 & |A[i][j] - A[i-1][j]| \leq 1 \text{ and } |A[i][j] - A[i][j-1]| \leq 1)
 \end{cases}$$

פירוט הפרמטרים בנוסחה:

A - מערך דו ממדי, i - מיקום השורה במערך, j - מיקום העמודה במערך

הסבר הפלט:

הנוסחה T מייצגת פתרון רקורסיבי לבעיה של מציאת אורך מסלול נחש הארוך ביותר המסתיים במיקום $[i, j]$ במערך A ומתחיל איפשהו.

נכונות הנוסחה:

- אם $i < 1 \text{ or } i > N \text{ or } j < 1 \text{ or } j > N$ אז ישנה חריגה מהמטריצה ולכן נחזיר 0.
- אם $i=1$ וגם $j=1$ זה אומר שאנחנו נמצאים בתחילה המטריצה (הכיוונים שהוגדרו בשאלה לא אפשריים במקרה הזה- אי אפשר לעלות למעלה או להתקדם שמאלה), ולכן נחזיר 1.
- אם $i > 1$ וגם $j > 1$ וגם שני הערכים המוחלטים $|A[i][j] - A[i-1][j]|$ ו- $|A[i][j] - A[i][j-1]|$ גדולים מ-1, התנאים שהוגדרו לצעד לא מתקיימים, לכן נחזיר 1.
- אם $i > 1$ וגם $j=1$ וגם הערך המוחלט $|A[i][j] - A[i-1][j]|$ גדול מ-1, התנאי היחיד שהוגדר לצעד ואפשרי פה (כי $j=1$ אז התנאי השני, $|A[i][j] - A[i][j-1]|$, לא יכול להתקיים) לא מתקיים, לכן נחזיר 1.
- אם $i=1$ וגם $j > 1$ וגם הערך המוחלט $|A[i][j] - A[i][j-1]|$ גדול מ-1, התנאי היחיד שהוגדר לצעד ואפשרי פה (כי $i=1$ אז התנאי השני, $|A[i][j] - A[i-1][j]|$, לא יכול להתקיים) לא מתקיים, לכן נחזיר 1.

- אם $i > 1$ ו- $j = 1$, זה אומר שנמצאים כרגע בגבול המערך בציר האופקי ואין לנו מקום להתקדם שמאלה, אז אנו יכולים רק להתקדם למעלה. אם התנאי $|A[i][j] - A[i-1][j]| \leq 1$ (הראשון מבין 2 אופציות לצעד במסלול שהוזכרו בשאלה) מתקיים, המסלול יהיה אורך המסלול בנקודה הקודמת $(i-1, j)$ ועוד אחד עבור השלב הנוכחי.
- אם $j > 1$ ו- $i = 1$, זה אומר שנמצאים כרגע בגבול המערך בציר האנכי ואין לנו מקום להתקדם למעלה, אז אנו יכולים רק להתקדם שמאלה. אם התנאי $|A[i][j] - A[i][j-1]| \leq 1$ (השני מבין 2 אופציות לצעד במסלול שהוזכרו בשאלה) מתקיים, המסלול יהיה אורך המסלול בנקודה הקודמת $(i, j-1)$ ועוד אחד עבור השלב הנוכחי.
- אחרת, אם $j > 1$ ו- $i > 1$, וגם התנאי $|A[i][j] - A[i][j-1]| \leq 1$ וגם התנאי $|A[i][j] - A[i-1][j]| \leq 1$ מתקיימים, ניתן יהיה להתקדם או למטה או ימינה- $(i, j-1)$ ו- $(i-1, j)$, המסלול יהיה הארוך ביותר מבין 2 האופציות, כמו כן, נוסיף אחד עבור השלב הנוכחי.

פונקציה בשפת C: (ROWS, COLS) מוגדרים כ-#define

```
int find_longest_snake_path(int A[ROWS][COLS], int i, int j) {
    if (i == 0 && j == 0)
        return 1;
    int up_length = 0, left_length = 0;
    if (i >= 0 && j >= 0 && abs(A[i][j] - A[i-1][j]) > 1 && abs(A[i][j] - A[i][j-1]) > 1)
        return 1;
    if (i > 0 && abs(A[i][j] - A[i-1][j]) <= 1)
        up_length = find_longest_snake_path(A, i-1, j);
    if (j > 0 && abs(A[i][j] - A[i][j-1]) <= 1)
        left_length = find_longest_snake_path(A, i, j-1);
    return max(up_length, left_length) + 1;
}
```

ג. סיבוכיות הזמן והמקום של אלגוריתמים תכנון דינמי המוצא את אורך מסלול נחש הארוך ביותר במערך A:

סיבוכיות הזמן- מכיוון שסיבוכיות זמן נמדדת בהתאם לגודל הקלט, ובמקרה הזה, גודל הקלט הוא N^2 (נתון- ניתן להניח כי A מערך ריבועי) והאלגוריתם עובר גם הוא על N^2 נתונים (בלולאה מקוננת), ניתן להגיד כי הסיבוכיות הינה ליניארית. נסמן את גודל הקלט m בצורה הבאה- $m = N^2$, נקבל כי סיבוכיות זמן הריצה יוצאת $O(m)$, כלומר ליניארית ביחס לגודל הקלט.

סיבוכיות המקום יצרתי מערך בשם path בגודל N. כיוון שאת סיבוכיות הזמן ביטאתי בעזרת m, שהוא גודל הקלט ($m = N^2$) כך אעשה גם עם סיבוכיות המקום. כיוון ש $m = N^2$, אז $N = \sqrt{m}$. לכן, סיבוכיות המקום הינה בסדר גודל של $O(\sqrt{m})$, כלומר סיבוכיות של שורש ריבועי ונמדדת ביחס לגודל הקלט שהוא N^2 .
פונקציה בשפת C: (N) מוגדר כ-#define משמש עבור עמודה ושורה- כיוון שידוע לנו שעמודה=שורה

```
int find_longest_snake_path_dynamic(int A[N][N]) {
    /*int* path = (int*)malloc(N * sizeof(int));*/
    int path[N], max_length = 0, temp = 0;
```

```

for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        if (i == 0 && j > 0) {
            if (abs(A[i][j] - A[i][j - 1]) <= 1)
                temp = path[j - 1] + 1;
            else
                temp = 1;
        }
        else if (j == 0 && i > 0) {
            if (abs(A[i][j] - A[i - 1][j]) <= 1)
                temp = path[j] + 1;
            else
                temp = 1;
        }
        else if (i > 0 && j > 0 && abs(A[i][j] - A[i - 1][j]) <= 1 && abs(A[i][j] - A[i][j
- 1]) <= 1)
            temp = max(path[j], temp) + 1;
        else if (i > 0 && abs(A[i][j] - A[i - 1][j]) <= 1)
            temp = path[j] + 1;
        else if (j > 0 && abs(A[i][j] - A[i][j - 1]) <= 1)
            temp = temp + 1;
        else
            temp = 1;
        path[j] = temp;
        if (path[j] > max_length)
            max_length = path[j];
    }
}
//free(path);
return max_length;
}

```

שאלה 3:

א. נוסחה רקורסיבית לפתרון הבעיה

$$\text{LCS}(i, j, k) = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \text{ or } k=0 \\ \text{LCS}(i-1, j-1, k-1) + 1 & \text{if } i>0 \text{ and } j>0 \text{ and } k>0 \text{ and } X[i-1] == Y[j-1] == Z[k-1] \\ \max(\text{LCS}(i-1, j, k), \text{LCS}(i, j-1, k), \text{LCS}(i, j, k-1)) & \text{else (if } i>0 \text{ and } j>0 \text{ and } k>0 \text{ and } \\ & (X[i-1] != Y[j-1] \text{ or } Y[j-1] != Z[k-1] \text{ or } X[i-1] != Z[k-1])) \end{cases}$$

פירוט הפרמטרים בנוסחה והסבר הפלט:

הנוסחה LCS מייצגת פתרון רקורסיבי לבעיה של מציאת אורך התת-סדרה המשותפת הארוכה ביותר מבין 3 סדרות.

X, Y, Z – הסדרות שעליהן מופעלת הנוסחה

i – אינדקס נוכחי בסדרה X

j – אינדקס נוכחי בסדרה Y

k – אינדקס נוכחי בסדרה Z

פונקציה בשפת C:

```
int LCS(char* X, char* Y, char* Z, int i, int j, int k) {
    if (i == 0 || j == 0 || k == 0)
        return 0;
    if (X[i-1] == Y[j-1] && Y[j-1] == Z[k-1])
        return LCS(X, Y, Z, i-1, j-1, k-1) + 1;
    int a = LCS(X, Y, Z, i-1, j, k);
    int b = LCS(X, Y, Z, i, j-1, k);
    int c = LCS(X, Y, Z, i, j, k-1);
    return max(a, max(b, c));
}
```

ב. נכונות הנוסחה

הנוסחה מבוססת על העובדה שאם האיבר האחרון בשלושת הסדרות הוא זהה, אז הוא חייב להיות חלק מהתת סדרה המשותפת הארוכה ביותר ולכן נוסף אחד ונשלח לרקורסיה לשאר הבדיקה את הסדרות והקטנת שלושת האינדקסים באחד- לבדיקה של שאר הסדרות. כמו כן, אם אחת מהסדרות נגמרה, יש להחזיר 0 כי לא ייתכן מצב שבו האורך ימשיך לגדול (האורך תלוי בשלושת הסדרות).

אם האיבר האחרון אינו זהה, אז הוא לא יכול להיות חלק מהתת סדרה המשותפת הארוכה ביותר- לכן נחפש את התת סדרה המשותפת הארוכה ביותר, לשם כך ישנן 3 אופציות לבדיקה שהמקסימלית מבניהם תהיה התשובה הנכונה- להוריד את האיבר האחרון מ X, להוריד את האיבר האחרון מ Y ולהוריד את האיבר האחרון מ Z.

ג. סיבוכיות הזמן והמקום של אלגוריתם תכנון דינאמי הפותר את הבעיה:

סיבוכיות הזמן: במקרה זה היא $O(N^3)$ כיוון שהאלגוריתם משתמש בלולאת for מקוננת על המטריצה בגודל $N \times N \times N$ - כל אחת מהלולאות רצה N פעמים, כיוון שיש 3 לולאות, סיבוכיות הזמן הינה $O(N^3)$. כמו כן, בשונה משאר התרגילים שקיבלנו מטריצה, הקלט שלנו הוא שלוש סדרות שהאורך שלהן הוא n, כלומר לינארי, ויש

צורך לעבור על 3 לולאות מקוננות- נוצר מצב שבו הקלט הוא ליניארי אך המעבר הוא N^3 .

סיבוכיות המקום: $O(N^2)$, מכיוון שיצרתי מערך דו ממדי (L) בגודל $N \times N$ שתפקידו לשמור את אורך תת-הסדרה המשותפת ביותר בכל נקודה (i, j, k) של הסדרות X, Y, Z, כך שבכל נקודה (i, j, k) במערך, נמצא האורך של תת הסדרה המשותפת ביותר בסדרות.

פונקציה בשפת C:

```
int lcs3(char* X, char* Y, char* Z) {
    int n = max(strlen(X), max(strlen(Y), strlen(Z)));
    int** L = (int**)malloc((n + 1) * sizeof(int*));
    for (int i = 0; i <= n; ++i)
        L[i] = (int*)malloc((n + 1) * sizeof(int));
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= n; j++) {
            for (int k = 0; k <= n; k++) {
                if (i == 0 || j == 0 || k == 0)
                    L[j][k] = 0;
                else if (X[i - 1] == Y[j - 1] && Y[j - 1] == Z[k - 1])
                    L[j][k] = L[j - 1][k - 1] + 1;
                else
                    L[j][k] = max(L[j - 1][k], max(L[j][k - 1], L[j][k]));
            }
        }
    }
    int num = L[n][n];
    for (int i = 0; i <= n; ++i) {
        free(L[i]);
    }
    free(L);
    return num;
}
```

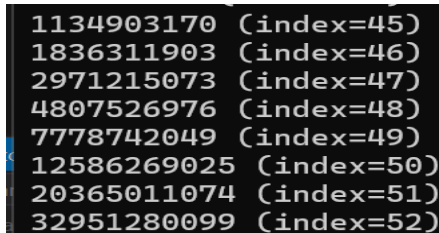

שאלת רשות:

א. התכנית הרקורסיבית:

```
unsigned long long fibonacci_recursive(int n) {
    if (n <= 1)
        return n;
    else
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2);
}

int main() {
    int index;
    printf("Enter the index to print Fibonacci series up to: ");
    scanf_s("%d", &index);
    printf("Fibonacci series up to index %d:\n", index);
    for (int i = 0; i <= index; ++i)
        printf("%lld (index=%d)\n", fibonacci_recursive(i), i);
    return 0;
}
```

לאחר מספר שעות, האינדקס האחרון עד שעצרתי את התכנית היה 52. כמו כן, החל מאינדקס מספר 46 לקח המון זמן על לפלט הבא:



```
1134903170 (index=45)
1836311903 (index=46)
2971215073 (index=47)
4807526976 (index=48)
7778742049 (index=49)
12586269025 (index=50)
20365011074 (index=51)
32951280099 (index=52)
```

ב. תכנית מבוססת תכנון דינאמי:

```
void fibonacci_dynamic(int n) {
    unsigned long long* fib = (unsigned long long*)malloc((n + 1) *
sizeof(unsigned long long));
    if (fib == NULL){
        printf("Memory allocation failed\n");
        return;
    }
    fib[0] = 0, fib[1] = 1;
    printf("0 (index=0)\n1 (index=1)\n ");
    for (int i = 2; i <= n; ++i) {
        fib[i] = fib[i - 1] + fib[i - 2];
        printf("%lld (index=%d)\n", fib[i], i);
    }
    long long result = fib[n];
    free(fib);
}
```