בס"ד

סיכום – בעיית המטוס

הבעיה: נתונה מטריצה המייצגת לוח משבצות $n \times m$ כך שעבור כל מעבר בין 2 קודקודים סמוכים יש משקל. דוגמא למטריצה 4×4 (כל קודקוד הוא תא במטריצה ולא משבצת ומתחילים מפינה שמאלית תחתונה):

7 1	8 5	18 3 1
10 9	4 3	7 6 8
4 5 3	6 1 3	3 5 9 1

יש למצוא את המסלול עם העלות הנמוכה ביותר החל מקודקוד 0,0 עד קודקוד n,m כאשר מותר לצעוד למעלה או ימינה בלבד.

המטריצה מיוצגת ע"י מערך דו מימדי של Nodes כאשר בכל Node יש x,y.

פתרון:

השיטה החמדנית – נבחר בכל שלב את המעבר עם העלות הנמוכה יותר.

- סיבוכיות השיטה: $O(n+m)$ – מספר המעברים במסלול החל מקודקוד 0,0 עד קודקוד n,m כאשר מותר לצעוד למעלה או ימינה בלבד.
- נכונות השיטה: השיטה לא מחזירה את התשובה הנכונה כי ייתכן ואחרי מעבר זול יגיע מעבר יקר ולהיפך ולכן לפעמים יהיה כדאי לוותר על מעבר זול כדי להרוויח בהמשך.

חיפוש שלם – נייצר את כל המסלולים מהנקודה 0,0 ל n,m ונסכום את כל המשקלים של כל מסלול,

בדוגמא שלנו, מסלול ראשון: $(0,0) \to (0,1) \to (0,2) \to (0,3) \to (1,3) \to (2,3) \to (3,3)$, נסכום את המעברים: $5+9+1+7+8+18=48$. וכך נמשיך עבור כל מסלול וניקח את המינימאלי.

- סיבוכיות השיטה: $O(\binom{n+m}{n} \cdot (n+m))$ – מספר המסלולים כפול מעבר על כל מסלול וחישוב העלות של המסלול.
- נכונות השיטה: בודקים את כל האפשרויות ולכן בהכרח נגיע גם לתשובה הנכונה.
- הקוד:

```
public static int wholeSearch(Node[][] mat) {
      Vector<String> allPaths = getAllPermotstions(mat);
      int min = Integer.MAX_VALUE;
      for(String path : allPaths) {
            int sum = 0,i=0,j=0;
            for (int k = 0; k < path.length(); k++) {
                  if(path.charAt(k) == '0') {
                        sum += mat[i][j].x;
                        j++;
                  }
                  else {
                        sum += mat[i][j].y;
                        i++;
                  }
```

```
            }
            if(sum < min) min =sum;
        }
        return min;
}

public static Vector<String> getAllPermotstions(Node[][] mat) {
        Vector<String> ans = new Vector<String>();
        getAllPaths(ans,mat,mat.length,mat[0].length,"");
        return ans;
}

private static void getAllPaths(Vector<String> ans, Node[][] mat,int
i, int j, String temp) {
        if(i == 0 && j == 0) {
            ans.add(temp);
            return;
        }
        else if(i == 0) {
            getAllPaths(ans,mat,0,j-1,temp + "0");
        }
        else if(j == 0) {
            getAllPaths(ans,mat,i-1,0,temp + "1");
        }
        else {
            getAllPaths(ans,mat,i,j-1,temp + "0");
            getAllPaths(ans,mat,i-1,j,temp + "1");
        }
}
```

<u>תכנות דינאמי</u> – בחיפוש השלם, עשינו בדיקות מיותרות, כי אם 2 מסלולים מגיעים לאותה נקודה ואחד מהם בעלות נמוכה יותר, ברור שניקח אותו ואין טעם להמשיך עם המסלול הארוך יותר.

נייצר מטריצה שבה בכל תא i,j נשמור את אורך המסלול הקצר ביותר מהנקודה $(0,0)$ עד הנקודה (i,j) ונמלא את המטריצה באופן הבא:

העלות להגיע לנקודה (i,j) שווה למינימום מבין העלות להגיע ל $(i,j-1)$ ואז ללכת ימינה לבין העלות להגיע ל $(i-1,j)$ ואז ללכת למעלה.

כלומר:

$$mat[i][j].entry = \min\,(mat[i][j-1].entry + mat[i][j-1].x,$$

נמלא תחילה עמודה ראשונה ושורה ראשונה.

בדוגמא שלנו המטריצה תיראה כך:

15	12	14	19
14	7	11	18
5	4	10	13
0	3	6	7

והתא הימני העליון (התא האחרון במטריצה) הוא התשובה – האורך של המסלול עם העלות הקטנה ביותר.

כדי לקבל את המסלול עצמו (נייצג אותו ע"י '0' – ימינה , '1' – למעלה) – נתחיל מהתא האחרון ונחזור אחורה לפי החוקיות בה מילינו את המטריצה – אם הגענו מלמטה נוסיף '1' לתשובה ונחזור אחורה ל $(i-1,j)$. אם הגענו מימין – נוסיף '0' לתשובה ונחזור אחורה ל $(i,j-1)$.

- סיבוכיות השיטה: $O(n \cdot m)$ – ממלאים את המטריצה לפי החוקיות. כדי לקבל את המסלול נבצע $O(n+m)$ פעולות. סה"כ: $O(n \cdot m)$
- נכונות השיטה: הראנו את האלגוריתם.
- הקוד:

```
class Node {
      int x,y,entry;

      public Node(int x, int y) {
            this.x = x;
            this.y = y;
            entry = 0;
      }
}

private void buildMatrix(){
      int n = mat.length, m = mat[0].length;
      for (int i = 1; i < n; i++){
            mat[i][0].entry = mat[i-1][0].y +  mat[i-1][0].entry;
            mat[i][0].numOfPaths = 1;
      }
      for (int j = 1; j < m; j++){
            mat[0][j].entry = mat[0][j-1].entry +  mat[0][j-1].x;
            mat[0][j].numOfPaths = 1;
      }
      for (int i = 1; i < n; i++){
            for (int j = 1; j < m; j++){
                  int y = mat[i-1][j].entry + mat[i-1][j].y;
                  int x = mat[i][j-1].entry + mat[i][j-1].x;
                  mat[i][j].entry = x <= y ? x : y;
            }
      }
      cheapestPrice = mat[n-1][m-1].entry;
      }

      public String getOnePath(){
            String ans = "";
            int i = mat.length-1, j = mat[0].length-1;
            while(i>0 && j>0){
                  int a = mat[i-1][j].entry+mat[i-1][j].y;
                  int b = mat[i][j-1].entry+mat[i][j-1].x;
                  if (a < b){
                        ans = "1" + ans;
                        i--;
                  }
                  else{
                        ans = "0" + ans;
                        j--;
                  }
            }
            while (j > 0){
                  ans = "0" + ans;
                  j--;
            }
            while (i > 0){
```

```
            ans = "1" + ans;
            i--;
        }
        return ans;
    }
```