**בעיית מטוס - שאלות יצירתיות**

נתון גרף המייצג לוח משבצת n X m, כך שבין כל שני קודקודים יש משקל. יש להחזיר את המסלול מקודקוד (0,0) לקודקוד (n,m) בעל העלות/המשקל המינימלי.

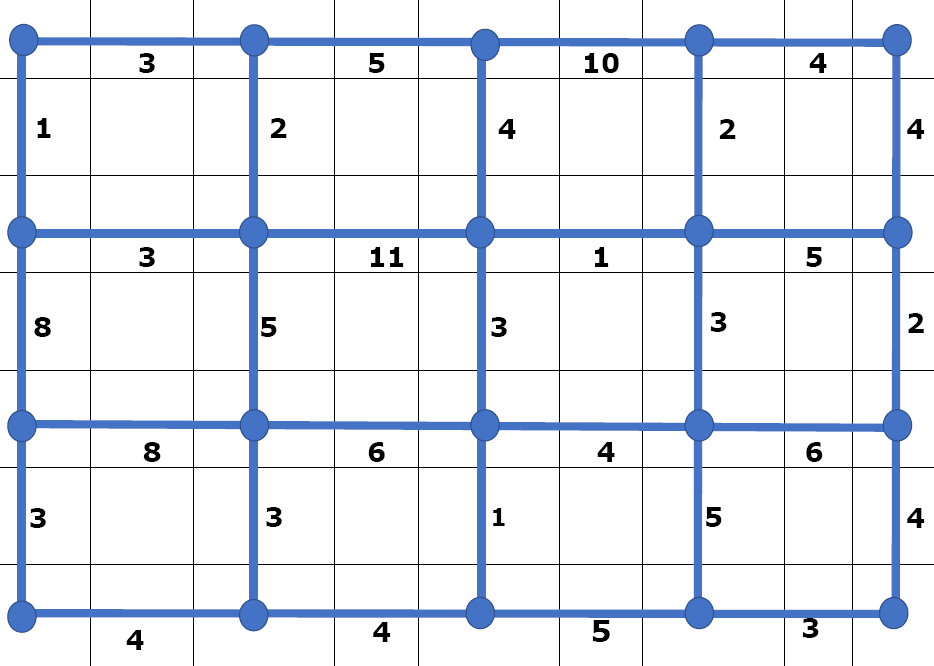
דוגמה:

להלן מטריצה של קודקודים, כל קודקוד Node מכיל את העלויות של שתי צלעות היוצאות ממנו לכיוונים ימינה (ציר X) ומטה (ציר Y). מספר שורות n=4, מספר עמודות m=5

לקודקוד (0,0) ערך שלו (3,1),

לקודקוד (3,4) ערך שלו (0,0),

לקודקוד (1,3) ערך שלו (5,3),



המטריצה מיוצגת ע"י מערך דו מימדי של קדקודים – אובייקט המחלקה Nodeעם המאפיינים הבאים:

x - עלות של הצלע האופקי,

y - עלות של הצלע האנכי,

price -העלות המינימאלית של הגעה לקדקוד,

numOfPaths - מספר המסלולים בעלי עלות מינימאלית המאפשרים הגעה לקודקוד

**class** Node{

**int** x, y, price, numOfPaths;

**public** Node(**int** x, **int** y){

**this**.x = x;

**this**.y = y;

**this**.price = 0;

**this**.numOfPaths = 0;

}

**...**

} // end class Node

**שאלות יצירתיות:**

1. **האלגוריתם להחזרת כל המסלולים הקצרים ביותר באינדוקציה.**

**רעיון האלגוריתם:**

להחזיר מסלולים תוך כדי בניית מטריצת העליות.

כאשר בונים מטריצת העלויות, יוצרים גם לכל כל קודקוד את רשימת המסלולים בעלי עלות מינימאלית המאפשרים הגעה לקודקוד זה ((ArrayList<String> allPaths.

לכל קודקוד משווים את העלויות מלמעלה ומשמאל:

* כאשר העלויות שונות - מעתיקים רשימת המסלולים מכיוון של העלות הנמוכה ביותר ולכל מסלול (מחרוזת) מוסיפים את התו המתאים) אם הגעה מימינה נוסיף '0' ואם הגעה מלמעלה נוסיף '1').
* כאשר העלויות שוות - מעתיקים רשימת המסלולים משני הכוונים ולכל מסלול (מחרוזת) מוסיפים את התו המתאים) אם הגעה מימינה נוסיף '0' ואם הגעה מלמעלה נוסיף '1') .

**class** Node{

**int** x, y, price, numOfPaths;

ArrayList<String> allPaths;

**public** Node(**int** x, **int** y){

**this**.x = x;

**this**.y = y;

**this**.price = 0;

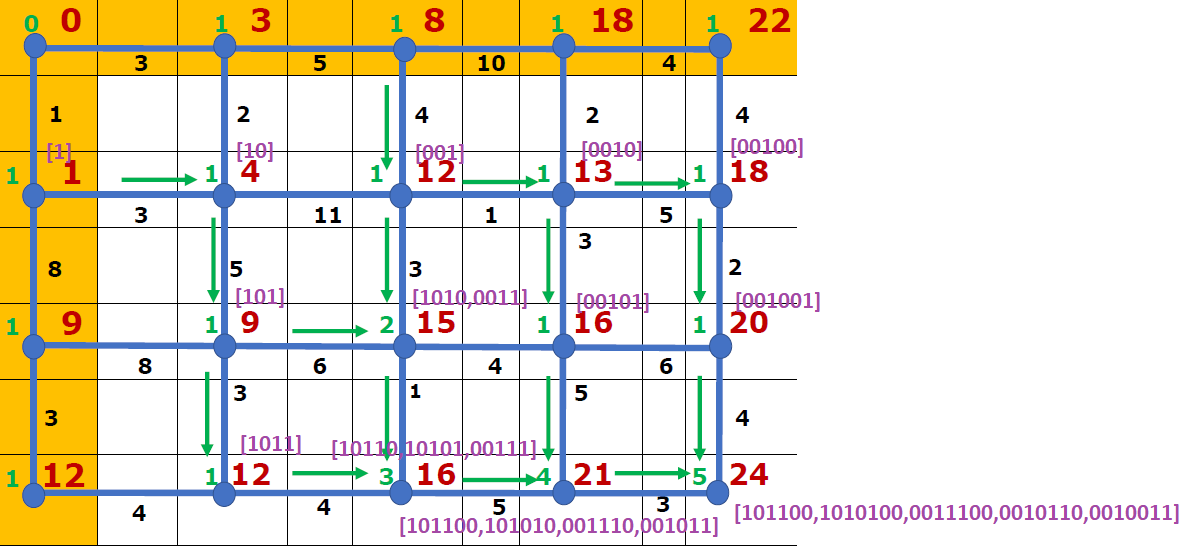
**this**.numOfPaths = 0;

**this**.allPaths = new ArrayList<String>();

}

**...**

} // end class Node

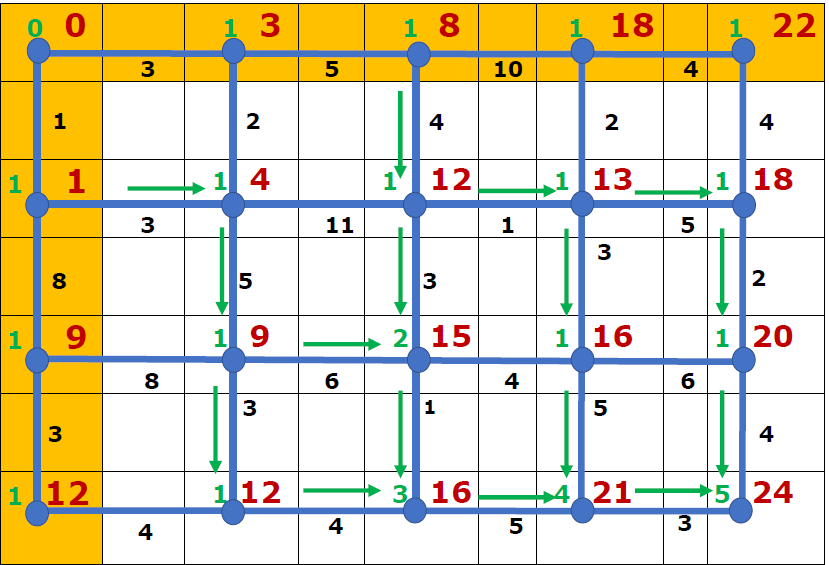


מה גודל ה-list במקרה גרוע ביותר?

כאשר k – מספר המסלולים בעלי עלות מינימאלית המאפשרים הגעה לקודקוד, אורך של כל מסלול (מעבר על כל מסלול) שווה ל- M+N, כמות הקודקודים שווה ל M\*N,

אז גודל ה-list הוא בסדר גודל

היות ומספר מסלולים יכול להיות גדול מדי כאשר N הוא מספר אפסים ו-M הוא מספר אחדות, לכן קודם ניצור מטריצת העליות עם חישוב כמות המסלולים לכל קודקוד ולאחר מכך נגדיר מספר שלם חיובי teta שמהווה גבול קביל לכמות המסלולים שמחזירים.



**void** **AllPathsInductive(int teta)**

**if** (numOfPaths<=teta)

**int** n = mat.length, m = mat[0].length

**for** (**int** i=0; i<n; i++){

**for** (**int** j=1; j<m; j++){

**int** a = mat[i-1][j].price+mat[i-1][j].y

**int** b = mat[i][j-1].price+mat[i][j-1].x

**if** (a<b)

**for**(**String** s : mat[i-1][j].allPaths)

mat[i][j].allPaths.add(s + "1")

**else** **if** (a>b)

**for**(**String** s : mat[i][j-1].allPaths)

mat[i][j].allPaths.add(s + "0")

**else** //a==b

**for**(**String** s : mat[i-1][j].allPaths)

mat[i][j].allPaths.add(s + "1")

**for**(**String** s : mat[i][j-1].allPaths)

mat[i][j].allPaths.add(s + "0")

**end-if**

**end-for**

**end-for**

**end-if**

**end-AllPathsInductive**

1. **אלגוריתם להחזרת המסלול האופטימאלי – מסלול קצר ביותר עם מינימום פניות (שינוי כיוון).**

**רעיון האלגוריתם:**

לאחר קבלת רשימת המסלולים בעלי עלות מינימאלית, מחשבים לכל מסלול את מספר הפניות (turning). מחזירים מסלול בעל מספר הפניות המינימלי.

**String** **optimalPath()**

ArrayList<String>paths=*AllPathsInductive)teta)*

String ans=””

int min = Integer.MAX\_VALUE;

for(String path : paths)

int turning = 0;

for (int i = 1; i < path.length(); i++)

if(path.charAt(i) != path.charAt(i-1))

turning++

end-if

if(turning < min)

ans = path

min = turning

end-if

end-for

end-for

return ans

**end-optimalPath**

1. **אלגוריתם שמחזיר האם קודקוד נמצא על אחד המסלולים הקצרים ביותר.**

**רעיון האלגוריתם:**

נשווה האם אורך המסלול הקצר ביותר מקודקוד (0,0) לקודקוד (M,N) שווה לסכום אורך המסלול הקצר ביותר מקודקוד (0,0) לקודקוד הנתון ואורך המסלול הקצר ביותר מקודקוד הנתון לקודקוד (M,N)

1. למצוא אורך המסלול הקצר ביותר מקודקוד (0,0) לקודקוד (M,N)
2. למצוא אורך המסלול הקצר ביותר מקודקוד (0,0) לקודקוד הנתון – כבר חושב תוך ביצוע שלב 1.
3. למצוא אורך המסלול הקצר ביותר מקודקוד הנתון לקודקוד (M,N)
4. להשוות האם אורך (1) שווה לסכום אורך (2) ואורך (3)

**boolean** **isOnBestPath**(**int** i,**int** j)

**return** getCheapestPrice()==

getBestPrice(0, 0, i, j) +

getBestPrice(i, j, mat.length-1, mat[0].length-1)

**end-isOnBestPath**

**int** **getBestPrice**(**int** p1, **int** q1, **int** p2, **int** q2){

// Assuming p2>=p1 and q2>=q1

mat[p1][q1].price = 0

**for** (**int** i=p1+1; i<=p2; i++){ // first column

mat[i][q1].price =

mat[i-1][q1].y + mat[i-1][q1].price

**end-for**

**for** (**int** j=q1+1; j<=q2; j++){// first row

mat[p1][j].price =

mat[p1][j-1].price + mat[p1][j-1].x

**end-for**

**for** (**int** i=p1+1; i<=p2; i++)

**for** (**int** j=q1+1; j<=q2; j++)

**int** a = mat[i-1][j].price+mat[i-1][j].y

**int** b = mat[i][j-1].price+mat[i][j-1].x

**if** (a<b){

mat[i][j].price = a

**else** **if** (a>b)

mat[i][j].price = b

**else**{//x==y

mat[i][j].price = a

**end-if**

**end-for**

**end-for**

**return** mat[p2][q2].price

**end-getBestPrice**

1. **אלגוריתם שמחזיר את כל הקודקודים שנמצאים על לפחות אחד מהמסלולים הזולים.**

**רעיון האלגוריתם:**

1. נוסיף ל-Node תכונה נוספת price2 (priceFromTheEnd) -העלות המינימאלית של הגעה לקודקוד מקודקוד (M,N).
2. ניצור מטריצת העליות מקודקוד (M,N) לקודקוד (0,0).
3. לכל קודקוד נשווה האם אורך המסלול הקצר ביותר מקודקוד (0,0) לקודקוד (M,N) שווה לסכום אורך המסלול הקצר ביותר מקודקוד (0,0) לקודקוד ואורך המסלול הקצר ביותר מקודקוד לקודקוד (M,N)

פונקציות עזר:

**boolean** **isOnBestPath(int i,int j)**

*buildMatrixFromTheEnd*()

**return** (mat[i][j].price + mat[i][j].priceFromTheEnd) == cheapestPrice;

**end-buildMatrixFromTheEnd**

**void** **buildMatrixFromTheEnd**()

**int** n = mat.length-1, m = mat[0].length-1

**for** (**int** i = n-1; i >= 0; i--){

mat[i][m].priceFromTheEnd = mat[i][m].y

+ mat[i+1][m].priceFromTheEnd;

**end-for**

**for** (**int** j = m-1; j >= 0; j--){

mat[n][j].priceFromTheEnd = mat[n][j+1].priceFromTheEnd + mat[n][j].x;

**end-for**

**for** (**int** i = n-1; i >= 0; i--){

**for** (**int** j = m-1; j >= 0; j--){

**int** y = mat[i+1][j].priceFromTheEnd + mat[i][j].y;

**int** x = mat[i][j+1].priceFromTheEnd + mat[i][j].x;

mat[i][j].priceFromTheEnd = x <= y ? x : y;

**end-for**

**end-for**

**end-buildMatrixFromTheEnd**

**משימות**:

1. ממש את האלגוריתם של תכנות דינאמי עבור בעיית המטוס – הפונקציה תקבל מטריצה של Nodes כאשר בכל Node יש ערך x - עלות הצלע אופקי וערךy - עלות הצלע האנכי. הפונקציה תחזיר את אורך המסלול עם העלות הנמוכה ביותר.
2. ממש את האלגוריתם של תכנות דינאמי להחזרת המסלול עצמו.
3. ממש את האלגוריתם לחישוב מספר המסלולים הקצרים ביותר.
4. ממש את האלגוריתם להחזרת כל המסלולים הקצרים ביותר – גרסה רקורסיבית.

שאלות יצירתיות:

1. ממש את האלגוריתם להחזרת כל המסלולים הקצרים ביותר– גרסה רקורסיבית. במידה ןכמות המסלולים גדולה מ-1000, להחזיר רק 1000 מסלולים).
2. ממש פונקציה המחזירה את המסלול האופטימאלי – מסלול קצר ביותר עם מספר פניות מינימאלי.
3. ממש פונקציה המקבלת נקודה ומחזירה האם הנקודה נמצאת על אחד המסלולים הקצרים
4. ביותר .
5. ממש פונקציה שמחזירה את כל הנקודות נמצאות לפחות על אחד המסלולים הקצרים
6. ביותר .