**בעיית המטוס**

נתונה רשת או גרף המייצג לוח משבצת MxN עם M קדקודים על ציר ה-X ו-N קדקודים על ציר ה-Y.כך שבין כל שני קודקודים (כוונה כל צלע) יש משקל/עלות מעבר הצלע.

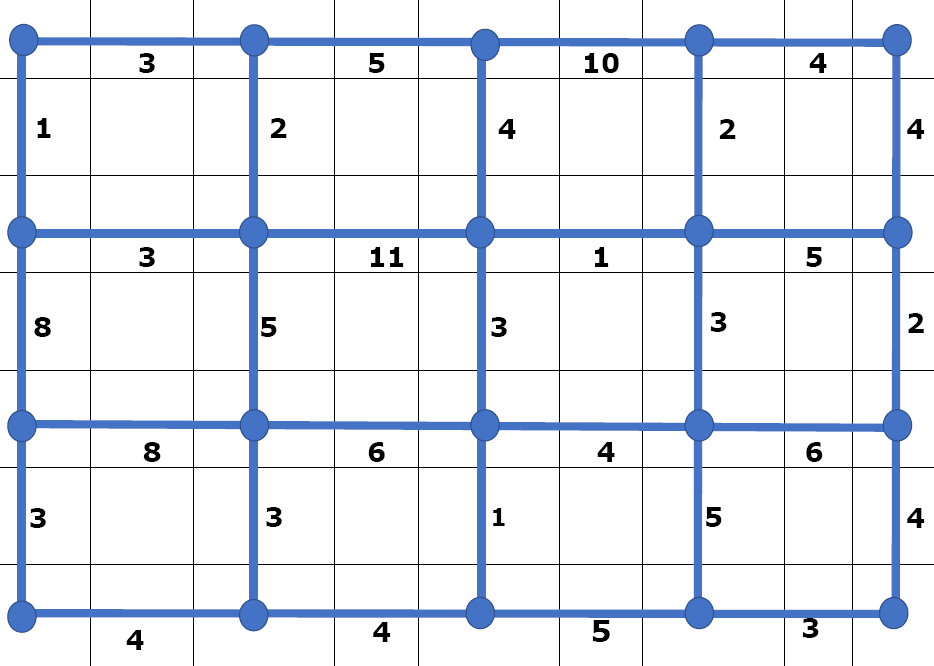
דוגמה:

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),



**מטרה: למצוא מסלול קצר ביותר מקודקוד (0,0) לקודקוד (M,N) כלומר מסלול בעל העלות/המשקל המינימלי כאשר מותר לצעוד למעטה או ימינה בלבד.**

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;

**...**

}

**...**

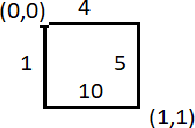
} // end class Node

## גישה 1 אלגוריתם חמדני:

## נבחר בכל שלב את המעבר עם העלות הנמוכה יותר.

**סיבוכיות האלגוריתם**: - מספר המעברים במסלול החל מקודקוד (0,0) לקודקוד (M,N) כאשר מותר לצעוד למעטה או ימינה בלבד.

**נכונות האלגוריתם**: 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.

****להלן הדוגמה:

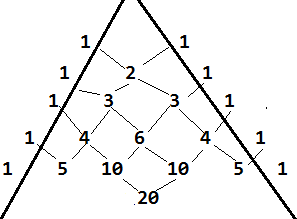
הבחירה הראשונה של החמדני לזוז דרך צלע בעלת עלות 1 ולהמשיך דרך צלע בעלת עלות 10 אבל 1+10 > 4+5 ←האלגוריתם החמדני לא עובד.

## גישה 2 חיפוש שלם:

## נייצר את כל המסלולים מהנקודה (0,0) לקודקוד (M,N), נסכום את כל המשקלים של כל מסלול וניקח את המינימאלי בינהם

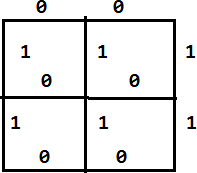
כאן נשאלת השאלה: **מהו מספר המסלולים בין הנקודה (0,0) לקודקוד (M,N)?**

כדי לחשב את מספר המסלולים בין הנקודה (0,0) לקודקוד (M,N) נשתמש בתכנון דינאמי.

מספר מסלולים לנקודה (i,j) שווה לסכום של מספר המסלולים עד הנקודה (i-1,j) פלוס מספר המסלולים עד הנקודה (i,j-1): #(i,j)=#(i-1,j)+#(i,j-1)

נתבונן במטריצה/רשת מצד שמאל, מהם המספרים האלה?

נשרטט את הרשת בצורה אחרת: קבלנו משולש פסקל.

נסמן את הכוון האופקי ב-0 וכוון האנכי ב-1, מקבלים כל מסלול כסדרה של אפסים ואחדות.

**סיבוכיות השיטה**: מספר המסלולים כפול מעבר על כל מסלול וחישוב העלות של המסלול.

אורך של כל מסלול (מעבר על כל מסלול) שווה M+N.

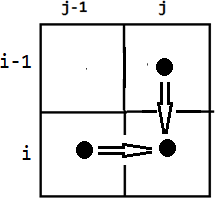
מספר המסלולים בין הנקודה (0,0) לקודקוד (M,N) שווה למספר צירופים עם חזרות של M+N אפסים ואחדות כאשר N הוא מספר אפסים ו-M הוא מספר אחדות:

*מקבלים* מספר גדול מאוד.

***סיבוכיות האלגוריתם****:*

**נכונות השיטה**: בודקים את כל האפשרויות ולכן בהכרח נגיע לתשובה הנכונה.

## גישה 3 תכנות דינאמי:

העלות הטובה ביותר בקדקוד (i,j) שווה לעלות המינימאלי בין העלויות הגעה מנקודה (i-1,j) או מנקודה (i,j-1).

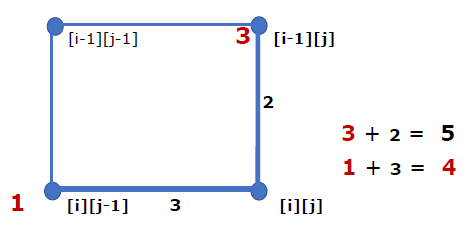
בחיפוש השלם, עשינו בדיקות מיותרות, כי אם 2 מסלולים מגיעים לאותה נקודה ואחד מהם בעלות נמוכה יותר, ברור שניקח אותו ואין טעם להמשיך עם המסלול הארוך יותר.

**נייצר מטריצה** שבה בכל תא [i,j] נשמור את אורך המסלול הקצר ביותר מהנקודה (0,0) לנקודה (i,j) ונמלא את המטריצה באופן הבא:

העלות להגיע לנקודה (i,j) שווה למינימום מבין העלות להגיע ל-(i,j-1) ועלות המעבר לכיוון ימינה, לבין העלות להגיע ל-(i-1,j) ועלות המעבר לכיוון למעטה. כלומר:

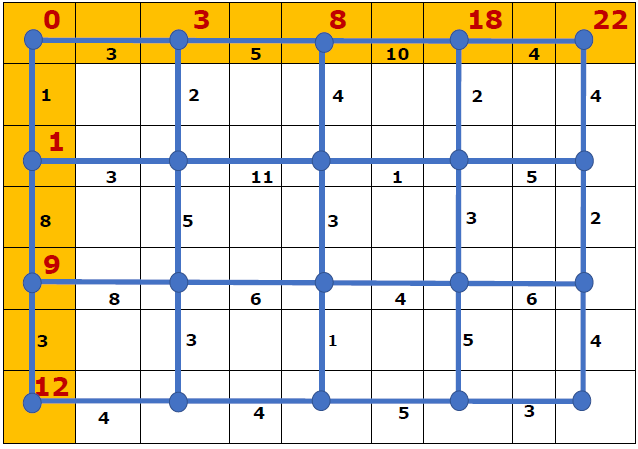
**𝑚𝑎𝑡(𝑖, 𝑗). 𝑝𝑟𝑖𝑐𝑒 = *min* (**

**𝑚𝑎t(𝑖 − 1, 𝑗). 𝑝𝑟𝑖𝑐𝑒 + 𝑚𝑎t(𝑖 − 1, 𝑗). 𝑦,**

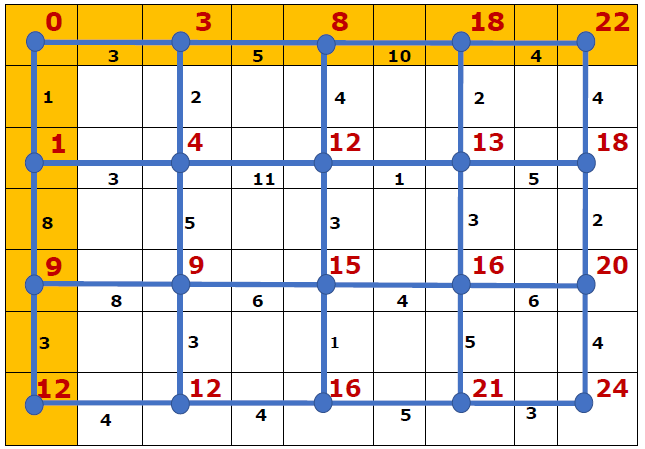
**𝑚𝑎t(𝑖, 𝑗 − 1). 𝑝𝑟𝑖𝑐𝑒 + 𝑚𝑎t(𝑖, 𝑗 − 1). 𝑥)**

1. **מציאת אורך המסלול עם העלות הזולה/הקטנה ביותר**

נמלא תחילה שורה ראשונה ועמודה ראשונה:



בדוגמא שלנו המטריצה תיראה כך:



והתא הימני התחתון (התא האחרון במטריצה) הוא התשובה – **האורך של המסלול עם העלות הזולה/הקטנה ביותר.**

**סיבוכיות האלגוריתם**: - כי ממלאים את המטריצה לפי החוקיות.

***פסאודו קוד:***

int cheapestPrice = 0

**void buildPriceMatrix()**

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

mat[0][0].price = 0

**for** (**int** i=1; i<n; i++) // first column

mat[i][0].price = mat[i-1][0].y+ mat[i-1][0].price

**end-for**

**for** (**int** j=1; j<m; j++) // first row

mat[0][j].price = mat[0][j-1].price + mat[0][j-1].x

**end-for**

**for** (**int** i=1; 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)

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**

cheapestPrice = mat[n-1][m-1].price

**end-buildPriceMatrix**

**int getCheapestPrice()**

return cheapestPrice

**end-getCheapestPrice**

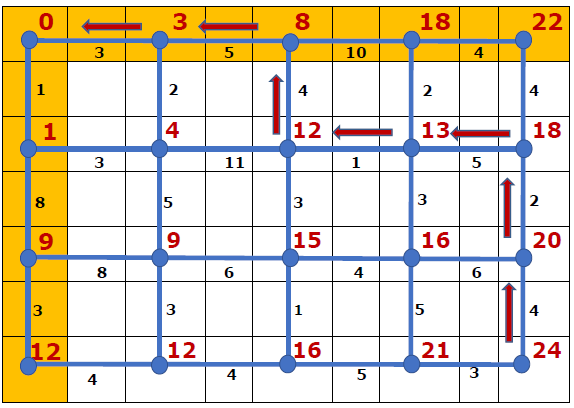
1. **מציאת מסלול אחד הטוב ביותר / מסלול בעל עלות מינימאלית**

**רעיון האלגוריתם:**

מתחילים מנקודה/קודקוד (m,n) ועולים בחזרה אחורה לפי החוקיות בה מילינו את המטריצה כוונה בכל נקודה בודקים מאיזה כיוון הגענו לנקודה זו בעזרת השוואת העלויות:

* כאשר העלויות שונות - הולכים לכיוון העלות הנמוכה ביותר,
* כאשר העלויות שוות - קובעים את הכיוון האחיד למצב כזה לכל הקודוקים. כוונה אם בוחרים בכיוון של y (אנכי) אז תמיד עולים שורה או אם בוחרים בכיוון x(אופקי): תמיד הולכים שמאלה.

**כדי לקבל את המסלול עצמו**) נייצג אותו ע"י '0'-ימינה , ו-'1' למעלה(, נתחיל מהתא האחרון (m,n) ונחזור אחורה לפי החוקיות שמילאנו את המטריצה: אם הגענו מלמטה נוסיף '1' לתשובה (מחזורת המסלול) ונחזור אחורה ל- (i-1,j) . אם הגענו מימין – נוסיף ''0' לתשובה ונחזור אחורה ל- (i,j-1).



**סיבוכיות האלגוריתם**: כדי לקבל את המסלול עצמו נבצע פעולות.

סה"כ: פעולות כי לפני ממלאים את המטריצה לפי החוקיות.

***פסאודו קוד:***

**String getOneCheapestPath()**

String ans = ""

**int** i = mat.length-1, j = mat[0].length-1

**while**(i>0 && j>0){

**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)

ans = "1" + ans

i--

**else** //a>b

ans = "0" + ans

j--

**end-if**

**end-while**

**if** (i==0){

**while** (j>0){

ans = "0" + ans

j--

**end-while**

**else** //j==0

**while** (i>0){

ans = "1" + ans;

i--;

**end-while**

**end-if**

**return** ans

**end-getOneCheapestPath**

1. **מציאת מספר מסלולים בעלי עלות מינימאלית**

**רעיון האלגוריתם:**

כאשר בונים מטריצת העלויות, מחשבים גם לכל כל קודוקד את מספר המסלולים בעלי עלות מינימאלית המאפשרים הגעה לקודקוד זה ((numOfPaths.

נמלא תחילה שורה ראשונה ועמודה ראשונה עם ערך numOfPaths=1

לאחר מכך אחל משורה ועמודה ראשונה, לכל קודקוד משווים את העלויות מלמעלה ומשמאל:

* כאשר העלויות שונות - מעתיקים מספר המסלולים מכיוון של העלות הנמוכה ביותר,
* כאשר העלויות שוות – סוכמים את מספר המסלולים משני הכוונים.

***פסאודו קוד:***

int numOfPaths= 0

**void buildPriceMatrix()**

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

mat[0][0].price = 0

**for** (**int** i=1; i<n; i++) // first column

mat[i][0].price = mat[i-1][0].y+ mat[i-1][0].price

mat[i][0].numOfPaths = 1

**end-for**

**for** (**int** j=1; j<m; j++) // first row

mat[0][j].price = mat[0][j-1].price + mat[0][j-1].x

mat[0][j].numOfPaths = 1

**end-for**

**for** (**int** i=1; 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)

mat[i][j].price = a

mat[i][j].numOfPaths = mat[i-1][j].numOfPaths

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

mat[i][j].price = b

mat[i][j].numOfPaths = mat[i][j-1].numOfPaths

**else** //x==y

mat[i][j].price = a

mat[i][j].numOfPaths =

mat[i][j-1].numOfPaths+mat[i-1][j].numOfPaths

**end-if**

**end-for**

**end-for**

cheapestPrice = mat[n-1][m-1].price

numOfPaths= mat[n-1][m-1].numOfPaths

**end-buildPriceMatrix**

**int getNumOfPaths()**

return numOfPaths

**end-getNumOfPaths**

1. **מציאת כל המסלולים הטובים ביותר (גרסה רקורסיבית)**

היות ומספר מסלולים יכול להיות גדול מדי כאשר N הוא מספר אפסים ו-M הוא מספר אחדות, מגדירים מספר שלם חיובי teta שמהווה גבול קביל למספר המסלולים.

**void** **AllPathsRecurs(int teta)**

**if** (numOfPaths<=teta)

ArrayList<String> paths =

**new** ArrayList<String>(numOfPaths)

***buildPaths***(**new** String(),

mat.length-1,mat[0].length-1, paths)

***print***(paths)

**end-if**

**end-AllPathsRecurs**

**void buildPaths(String path, int i, int j, ArrayList<String> paths)**

**if** (i==0 && j==0)

paths.add(path)

**else** **if** (i==0 && j > 0)

buildPaths("0"+path, i, j-1, paths)

**else** **if** (j==0 && i > 0)

buildPaths("1"+path, i-1, j, paths)

**else**

**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)

buildPaths("1"+path, i-1, j, paths)

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

buildPaths("0"+path, i, j-1, paths)

**else** //a==b

buildPaths("1"+path, i-1, j, paths)

buildPaths("0"+path, i, j-1, paths)

**end-if**

**end-if**

**end-buildPaths**

**הסיבוכיות: O((M+N)\*** **numOfPaths)**

**משימות**:

* לממש את האלגוריתם של תכנות דינאמי עבור בעיית המטוס :לכתוב פונקציה שמקבלת 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.
* לממש את האלגוריתם של תכנות דינאמי להחזרת המסלול עצמו.
* לממש את האלגוריתם לחישוב מספר המסלולים הקצרים ביותר.
* לממש את האלגוריתם להחזרת כל המסלולים הקצרים ביותר.