בעיית השוקולד

נרצה לשבור שורה של קוביות שוקולד באורך n לקוביות נפרדות: [1, ... , n]
כל שבירה עולה (i * (n – i כסף, כאשר i מסמל את מקום השבירה, מה המחיר המינימאלי שנשלם?
בכל דרך שננסה לשבור המחיר הסופי יהיה קבוע : 2 / ((n (n – 1

הוכחה באינדוקציה

עבור השבירה הראשונה במיקום ה-i נשלם: (i * (n – i
על השבירות של החלק השמאלי נשלם: (i * (n – i
על השבירות של החלק הימני נשלם: 2 / ((n – i)(n – i – 1))
לכן: i * (n – i) + i * (n – i) + ((n – i)(n – i – 1)) / 2 = (n (n – 1)) / 2

בעיית הבקבוקים + Floyd Warshall

נקבל שני בקבוקים בגודל [n,m], איך נגיע למצב [a,b] בדרך הכי מהירה? חוקי המשחק:

1. מותר למלא כלי, אך חובה עד הסוף
   - [a,b] -> [n,b]
   - [a,b] -> [a,m]

2. מותר לרוקן כלי, אך חובה עד הסוף.
   - [a,b] -> [0,b]
   - [a,b] -> [a,0]

3. מותר לשפוך מכלי אחד לכלי אחר, אך חובה עד הסוף או עד ככל שניתן.
   - [a,b] -> [min (a + b, n), a + b – min (a + b, n)]
   - [a,b] -> [a + b – min (a + b, m), min (a + b, m)]

4. לא ניתן למלא \ לרוקן שלא עד הסוף או לשפוך שלא ככל שניתן.

מטריצה שכנויות בוליאנית המייצגת את מצבי הבקבוק.
השורות והעמודות מייצגים את הקודקודים.
כל תא מייצג האם ניתן לעבור ממצב אחד לשני באופן ישיר.

Input: הגבהים של הבקבוקים
Output: מטריצה

פונקציית עזר: getIndex

i – גובה המים במצב הנוכחי בבקבוק הראשון.
j – גובה המים במצב הנוכחי בבקבוק השני.

```
getIndex( i , j , n )
        return ( n + 1 )*i + j

initRibs( n, m )
        dim = ( n + 1 )*( m + 1 )
        index1, index2
        For ( i = 0 to n )
                For ( j = 0 to m )

                        index1 = getIndex( i, j, m )
                        mat [ index1 ] [ getIndex( 0, j, m ) ] = true
                        mat [ index1 ] [ getIndex( i, 0, m ) ] = true
                        mat [ index1 ] [ getIndex( n, j, m ) ] = true
                        mat [ index1 ] [ getIndex( i, m, m ) ] = true

                        index2 = getIndex( Min( i + j , n ) , i + j – Min( i + j , n ) , m ) = true
                        mat[ index1 ] [ index2 ] = 1
                        index2 = getIndex( i + j – Min( i + j , m ) , Min( i + j , m )  , m ) = true
                        mat[ index1 ] [ index2 ] = 1

        For ( i = 0 to dim ) mat [ i ] [ i ] = true
        return mat
```

מציאת מטריצה בוליאנית המייצגת האם גם יש קשרים עקיפים בין קודקודים.
Input: מטריצת שכנויות (קשר ישיר)
Output: מטריצה.

```
n = mat.length
For ( k = 0 to n )
        For ( i = 0 to n )
                For ( j = 0 to n )
                        mat[ i ][ j ] = ( mat[ i ][ k ] AND mat[ k ][ j ] ) OR mat[ i ][ j ]
return mat
```

האם יש מסלול כלשהו בין v1 ל-v2?
נקבל true אם כן ו-false אם לא.

```
return mat[v1][v2]
```

מטריצה המציגה בכל תא את המסלולים הישירים \ עקיפים הקצרים ביותר בין הקודקודים.

Input: מטריצת שכנויות בוליאנית.
Output: מטריצת סטרינגים של המסלולים.

```
n = mat.length
path[][] = new String[n][n]
For ( i = 0 to n )
        For ( j = 0 to n )
                a0 = i /  ( n + 1 )
                b0 = i % ( n + 1 )
                a1 = j /  ( n + 1 )
                b1 = j % ( n + 1 )
                IF ( mat[ i ][ j ] = true ) path[ i ][ j ] = "[" + a0 + "," + b0 + "] —> [" + a0 + "," + b0 + "]"
                ELSE path[ i ][ j ] = new String
For ( k = 0 to n )
        For ( i = 0 to n )
                For ( j = 0 to n )
                        IF ( mat[ i ][ j ] == true )  path[ i ][ j ] = path[ i ][ k ] + " >>> " + path[ k ][ j ]
return path
```

האם הגרף קשיר?

Input: מטריצה בוליאנית לאחר FW עם כל הקשרים.
Output: True \ False

```
n = mat.length
For ( i = 0 to n )
        IF ( mat[ 0 ][ i ] == false ) return false
return true
```

בעצם לאחר מציאת המטריצה המייצגת את כל הקשרים, אם כל השורה הראשונה True, זה אומר שהגרף קשיר.
למה? כי אז זה אומר שלקודקוד הראשון יש קשר עם כולם ולכולם יש קשר אליו.

כמה רכיבי קשירות יש בגרף?

בנוסף, במערך helper ניתן לראות איזה קודקודים יחד באותו רכיב קשירות.

```
n = mat.length
counter = 1
helper[n]          // initialize with zero's
For ( i = 0 to n )
        For ( j = 0 to n )
                if ( mat [ i ][ j ] == true && helper[ j ] == 0 )  helper[ i ] == counter
        counter++
return counter
```

סידור מטריצה במקרה שהקודקודים לא לפי הסדר

Floyd Warshall (גרף ממושקל)

טיפול בגרף עם משקלים (מרחקים).
מה המרחק הקצר ביותר בין שני קודקודים?

כאשר המשקלים על הצלעות -
מטריצה המייצגת את המסלול
הקצר ביותר בין כל שני קודקודים

```
V = vertices.length

dist = V*V matrix of minimum distance initialized to ∞
For each vertex v
        dist[ v ][ v ] = 0

For each edges ( u , v )
        dist[ u ][ v ] = weights( u , v )

For ( k = 0 to n )
        For ( i = 0 to n )
                For ( j = 0 to n )
                        dist[ i ][ j ] = Min( dist[ i ][ j ] , dist[ i ][ k ] + dist[ k ][ j ] )
```

בניית מטריצת השכנויות
(אם לא נקבל אותה)

כאשר המשקלים על הקודקודים -
מה המסלול הקצר ביותר בין כל שני קודקודים?

כאן נקבל את מטריצת השכנויות כי לא נוכל לבנות מטריצת שכנויות כשהמשקלים על הקודקודים, מערך של משקלים. בנוסף, נקבל מערך של משקלים.

הרעיון הוא להפוך את הבעיה למשקלים על צלעות ולהשתמש באלגוריתם הקודם. איך? נסכום כל שני קודקודים צמודים. בין הקודקודים שלא מתחברים באופן ישיר נוריד את הקודקודים שנספרו פעמיים.

```
n = mat.length

// אתחול המשקלים במטריצה בין קודקודים שמחוברים באופן ישיר
For ( i = 0 to n )
        For ( j = 0 to n )
                IF ( mat[ i ][ j ] == 1 )
                        mat[ i ][ j ] = weight[ i ] + weight[ j ]

// הרצת פלויד וורשאל למציאת מינימום מרחק
For ( k = 0 to n )
        For ( i = 0 to n )
                For ( j = 0 to n )
                        dist[ i ][ j ] = Min( dist[ i ][ j ] , dist[ i ][ k ] + dist[ k ][ j ] )

// תיקון המשקלים שנספרו פעמיים
For ( i = 0 to n )
        For ( j = 0 to n )
                IF ( mat[ i ][ j ] != ∞ AND i != j)
                        mat[ i ][ j ] += weight[ i ] + weight[ j ]
                        mat[ i ][ j ] /= 2

return mat
```

האם יש מעגל שלילי בגרף?

*** לבדוק אלגו' שנותן דוגמא למעגל שלילי ***

נריץ FW למציאת מטריצת המסלולים הקצרים ביותר

בגרף מכוון – אם יש מספר שלילי באלכסון הראשי – קיים מעגל שלילי.

```
For ( i = 0 to n )
        IF ( mat[ i ][ i ] < 0 ) return true
return false
```

בגרף לא מכוון – מספיק שיש מספר שלילי אחד – קיים מעגל שלילי.

```
For ( i = 0 to n )
        For ( j = 0 to n )
                if ( mat[ i ][ j ] < 0 ) return true
return false
```

מציאת מטריצה המציגה בכל תא את המסלולים עצמם (הישירים \ עקיפים) הקצרים ביותר בין הקודקודים.

Input: מטריצת שכנויות בוליאנית.

Output: מטריצת סטרינגים של המסלולים.

```
shortestPath( mat[ ][ ] )

    n = mat.length
    path[ ][ ] = new String[ n ][ n ]

    FOR i = 0 to n
        FOR j = 0 to n
            IF dist[ i ] ! = ∞ DO path[ i ][ j ] = i “—>” j
            ELSE path[ i ][ j ] = “ ”
    FOR k = 0 to n
        FOR i = 0 to n
            FOR j = 0 to n
                IF ( dist[ i ][ j ] > dist[ i ][ k ] + dist[ k ][ j ] )
                    dist[ i ][ j ] =  dist[ i ][ k ] + dist[ k ][ j ]
                    path[ i ][ j ] = path[ i ][ k ] + path[ k ][ j ]
```

Best

מציאת תת המערך הרציף והמינימאלי, בעל סכום האיברים המקסימאלי ביותר.

```
n = arr.length
temp_index =  sum = 0
s_index =  e_index = -1
maxSum = MIN_VAL
For ( i = 0 to n )
        tempSum += arr[ i ]
        IF (  tempSum > max )
                maxSum =  tempSum
                s_index = temp_index
                e_index = i
        IF (  tempSum < 0 )
                tempSum = 0
                temp_index = i + 1
return { max , s_index , e_index }
```

Best Cycle

```
n = arr.length
arr_neg[n]
sum = 0

For ( i = 0 to n )
        sum += arr[ i ]
        arr_neg[ i ] = – arr[ i ]

best1[ ] = best(arr)
best2[ ] = best(arr_neg)

IF ( best1[ 0 ] < 0   OR   best1[ 0 ]   >=   sum + best2[ 0 ] ) return best1
return { sum+best2[0] , (best2[2] + 1) % n , best2[ 1 ] – 1 }
```

אלגוריתם תחנות הדלתק למציאת האינדקס ההתחלתי

```
n  =a.length
sum = 0
C[]
For ( i to n )
        sum += a[ i ] – b [ i ]
IF ( SUM < 0 )
        return -1          // not enough gas!
ELSE
        For ( i = 0 to n )
                C[ i ] = a [ i ] – b[ i ]
startIndex = bestCycle(C)
return startIndex[ 1 ]     // start point
```

מציאת תת מטריצה מקסימאלית

```
maxSubMatrixSum ( mat[ ][ ] )   // O ( cols^2 * rows )
        maxSum = mat[ 1 ][ 1 ]
        tempArr = new int[ rows ]
        For i = 1 to cols
                For j = 1 to rows
                        tempArr[ i ] += mat[ j ][ i ]
                best = Best( tempArr )
                IF best[ 0 ] > maxSum
                        maxSum = best[ 0 ]
        return maxSum
```

Dijkstra

מציאת כל המרחקים הקצרים ביותר מקודקוד מקור בגרף לשאר הקודקודים.

Input: Graph, s

```
For each vertex v in Graph
        dist[ v ] = ∞
        prev[ v ] = null
        Q.add( v )
dist[ s ] = 0
While Q is not empty
        v = node in Q with smallest dist[ ]
        remove v from Q
        For each neighbor u of v          // where neighbot u has not yet been removed from Q
                IF (  dist[ v ] + length( v , u ) < dist[ u ] )
                        dist[ u ] =  dist[ v ] + length( v , u )
                        prev[ v ] = u
return dist[ ] OR prev [ ]
```

BiDirectional Dijkstra

```
spr ← g(s)
tpr ← g(t)
OpenF ← {s}
OpenB ← {t}
For ( all n ∈ V − {s, t} )
        npr ← ∞
p ← ∞
While ( OpenF 6= ∅ AND OpenB 6= ∅ do )
        prminF = get-min(OpenF )
        prminB = get-min(OpenB)
        IF ( prminF + prminB +  ≥ p )
                return path for p
        IF ( Forward frontier is expanded then )
                n = delete-min(OpenF )
                ClosedF = ClosedF ∪ n
                For ( all succ ∈ nsuccessors )
                        IF ( succ ∈ ClosedF )
                                continue
                        ELSE
                                priority ← pr(succ)
                                IF ( succ ∈ OpenF )
                                        IF ( succpr > priority )
```

```
                    succpr = priority end if
                ELSE
                    succpr = priority
                    OpenF ← OpenF ∪ {succ}
        IF ( succ ∈ OpenB AND gF (succ) + gB(succ) < p )
            p ← gF (succ) + gB(succ)
    else // Expand backward frontier analogously
return failure
```

BFS

שיטה לחשיפת גרף מקודקוד מקור.

משמש למציאת המרחקים הקצרים ביותר של כל קודקוד מקודקוד המקור ואת האבא של כל אחד לצורך מציאת המסלול.

כדי לדעת האם קיים מעגל פשוט בגרף, נבדוק פשוט האם במהלך הריצה הגענו לקודקוד בצבע אפור, כך זה אומר שמישהו כבר "חשף" אותו ולכן יש גישה אליו משני קודקודים, מה שאומר קיים מעגל.
התוספת עבור מעגל פשוט צבוע בכתום ואינו נחוץ עבור האלגוריתם הרגיל.

```
BFS ( G, s)        // O ( V + E )
    For each vertex u in V[G]
        color[ u ] = WHITE
        distances[ u ] = NULL
        parents[ u ] = NULL
    color[ s ] = GRAY
    distances[ s ] = 0
    parents[ s ] = NULL
    Q.enqueue(s)
    While ( Q is not empty )
        u = Q.dequeue
        For each vertex v in adj[ u ]
            IF ( color v == WHITE )
                distances[ v ] = distances[ u ] + 1
                parents[ v ] = u
                Q.enqueue( v )
            ELSE
                IF ( color[ u ] == GRAY )
                    return TRUE

        color[ u ] = BLACK
    return FALSE
    return [ distances, parents, color ]
```

איך אפשר לדעת שהגרף קשיר?
נעבור על מערך הצבעים, אם כולם מסומנים בשחור, סימן שהגרף קשיר (האלגוריתם הצליח להגיע מקודקוד המקור לכל שאר הקודקודים).

איך אפשר לשחזר את העץ?
נעבור על מערך האבות, כל תא מייצג קודקוד, הערך באותו התא הוא קודקוד האב.

איך אפשר לדעת כמה רכיבי קשירות יש בגרף?
בהרצה הראשונה של האלגוריתם, הקאונטר שלנו יהיה 1.
לאחר כל הרצה של האלגוריתם נבדוק אם יש קודקודים לבנים במערך הצבעים.
אם כן – נריץ את האלגוריתם על הקודקוד הלבן ונספור 1+ על רכיבי הקשירות (נחזור כך עד שאין קודקודים לבנים).

איך אפשר לדעת מיהם הקודקודים בכל רכיבי הקשירות?
ניצור מערך בגודל הקודקודים – תא באינדקס i מסמל את קודקוד מספר i והערך בפנים יסמן לאיזה רכיב קשירות הוא משתייך.
בכל פעם שנחשוף קודקוד, נשמור עבורו את מספר האיטרציה של הרצת האלגוריתם כסימון למספר רכיב הקשירות שבו הקודקוד נמצא.

מה המרחק הקצר ביותר בין שני קודקודים?

נקבל כקלט את הגרף, קודקוד מקור וקודקוד יעד. האלגוריתם הפשוט הוא להריץ BFS ולהחזיר את הערך שבמערך המרחקים באינדקס של קודקוד היעד.

מה המסלול ביניהם?
נחזור אחורה במערך האבות.

הגדרה: קוטר בגרף הוא המרחק המקסימאלי מבין כל המרחקים הקצרים ביותר בין כל שני קודקודים בגרף.

ניתן למצוא קוטר ע"י BFS

O(V + E)

```
Select s in V
call BFS(G, s)
u  = find the vertex with max value in ‘d’ array
call BFS(G,u)
return max value in ‘distances’ array
```

דוגמא למסלול בגודל הקוטר: בקריאה השניה של האלגוריתם, נחזור במערך האבות אחורה מהתא (הקודקוד) עם הערך המקסימאלי.

DFS

```
time = 1

Foreach ( v in G)
	if Color[v] = white		// if not visited
		v.startTime = time
		time++
		DFS-visit(v)

DFS-visit(v)
	Foreach ( u in neighbors of v )
		If ( colot[u] = white )
			u.startTime = time
			time++
			DFS-visit(u)
	v.endTime = time
	time++
	color[v] = black
```

שריפת עלים (למציאת קוטר, רדיוס ומרכזי העץ)

הגדרה: עץ הואגרף קשיר עם n-1 צלעות וללא מעגלים

האלגוריתם לשריפת עלים

בכל איטרציה נוריד את כל העלים שיש לעץ
באותו הרגע עד שנישאר עם קודקוד אחד \ שניים.
לאחר השריפה:

1. כמות המרכזים?
כמות הקודקודים שנשארו.
2. קוטר זוגי או אי זוגי?
לפי כמות הקודקודים שנשארו:
אם 1 –> זוגי
אם 2 –> אי זוגי
3. רדיוס?
אם הקוטר זוגי -> כמות השריפות
אם הקוטר אי זוגי -> כמות השריפות + 1
4. קוטר?
אם הקוטר זוגי -> פעמיים הרדיוס
אם הקוטר אי זוגי -> פעמיים הרדיוס - 1

```
fire( vector[] tree )
	radius = 0
	diameter = 0
	numOfCenters = 0
	vertex = 0
	leaf = 0

	leaves = new vector( n )
	defrees = new int[ n ]
```

עצים איזומורפיים

עצים בעלי שורש

1. איך נבדוק אם שני עצים איזומורפיים?
ע"י מחרוזת של '0' ו-'1' המתארת את העץ.

עצים ללא שורש

בניית עץ מרשימת דרגות

משפטים:
א. בכל עץ ---> $(|E| = |V| -1)$
ב. לפי למת לחיצות הידיים ---> סכום הדרגות = $2(|V|-1)$
ג. בכל עץ יש לפחות שני עלים.

BuildTreeFromDegreesArray(deg[]) input: Array of degrees – deg[] Output: the tree of parent array
N = deg.size tree[N]

האלגוריתם:
הרעיון - ננסה לחבר (במידת האפשר) עלה לקודקוד שהדרגה שלו גדולה מ-1 (כדי שישארו עלים).

```
sum = 0
for i = 1 to N do
        sum += deg[i]
if( (sum/2) + 1 != N )
        print(Not a tree degrees array)
        return
deg[] = sort(deg)
j = first index that deg[j] > 1
for i  = 1 to N – 2 do
        tree [i] = j
        deg[j]--
        if (deg[j] = 1)
                j++
tree[N-1] = N
return tree
```

דוגמא:

v1	v2	v3	v4	v5	v6	v7	v8
1	1	1	1	2	2	3	3

v1	v2	v3	v4	v5	v6	v7	v8
0	1	1	1	1	2	3	3

v1	v2	v3	v4	v5	v6	v7	v8
0	0	1	1	1	1	3	3

v1	v2	v3	v4	v5	v6	v7	v8
0	0	0	1	1	1	2	3

v1	v2	v3	v4	v5	v6	v7	v8
0	0	0	0	1	1	1	3

v1	v2	v3	v4	v5	v6	v7	v8
0	0	0	0	0	1	1	2

v1	v2	v3	v4	v5	v6	v7	v8
0	0	0	0	0	0	1	1

v1	v2	v3	v4	v5	v6	v7	v8
0	0	0	0	0	0	1	1

אוילר

יהי גרף G(V,E) גרף לא מכוון.
מסלול Px,y נקרא מסלול אוילר ב-G, אם הוא עובר בכל הצלעות של G כך שכל צלע מופיעה בו פעם אחת בלבד ו- x != y.

מעגל אוילר

יהי גרף G(V,E) גרף לא מכוון.
מעגל אוילר ב-G הוא מסלול אוילר סגור, כלומר מסלול העובר בכל צלעות הגרף פעם אחת בלבד והקודקוד ההתחלתי הוא גם קודקוד הסיום.

גרף אוילריאני

גרף המכיל מעגל אוילר.

משפטי זיהוי אוילר בגרפים

1. יש בגרף G מעגל אוילר (גרף אוילריאני) אמ"מ G קשיר וכל דרגות הגרף זוגיות.
2. יש בגרף G מסלול אוילר אמ"מ G קשיר ובדיוק 2 קודקודים בעלי דרגות אי זוגיות.

```
fineEulerCycle (deg[])
input: Eulerian graph G
Output: The euler cycle
-----------------------------------
Stack S          // our check path
Stack C          // out cycle path
S.push(v0)
while S is not empty DO
        u = S.top
```

```
        if (deg(u) = 0)    // no neighbords
                S.pop
                C.push(u)
        else
                v = some v in u neighbords
                S.push(u)
return C
```

קידוד הופמן

1. אם לא ממוין – הקוד הרגיל O(nlogn)
2. אם ממוין – שתי מחסניות O(n)

```
huffman (C)       // O(nlogn)
input: nodes array of chars and frequency
Output: root of the tree
---
n = |C|
Q <- C               // insert C into Q
for i = 1 to n-1
        z = allocate_Node
        x = z.left = extractMin(Q)
        y = z.right = extractMin(Q)
        z.freq = x.freq + y.freq
        insert(Q,Z)
return extractMin(Q)
```

עץ פורש מינימאלי בגרף עם משקלים

1. Kruskal – נבחר בכל פעם את הצלע המינימאלית, כל עוד היא לא מייצרת מעגל, עד שנגיע ל-n-1 צלעות.
2. Kruskal הפוך – נמחק בכל פעם את הצלע המקסימאלית, כל עוד הגרף נשאר קשיר, עד שנגיע ל-n-1 צלעות.

Kruskal Algorithm

```
// YouTube Algorithm

A          // vector of Edges
foreach v in V
           make-disjoint-set(v)
Sort(E)  // min to max
for each v1 and v2 in E
           if Find(v1) != Find (v2)
                      A = Union(A, (v1,v2)}
                      Union(v1, v2)
return A
```

בעיית המטוס עבור 3X3

1. נגדיר מטריצה 4X4.
2. נאתחל את כל העלויות \ משקלים של המטריצה.

(0,0) X = 4 Y = 3 PRICE = 0 nPATH = 1	(0,1) X = 3 Y = 9 PRICE = 4 nPATH = 1	(0,2) X = 3 Y = 5 PRICE = 7 nPATH = 1	(0,3) X = null Y = 2 PRICE = 10 nPATH = 1
(1,0) X = 1 Y = 7 PRICE = 3 nPATH = 1	(1,1) X = 5 Y = 8 PRICE = 13 nPATH = 1	(1,2) X = 7 Y = 3 PRICE = 10 nPATH = 1	(2,3) X = null Y = 5 PRICE = 12 nPATH = 1
(2,0) X = 2 Y = 3 PRICE = 10 nPATH = 1	(2,1) X = 3 Y = 4 PRICE = 12 nPATH = 1	(2,2) X = 8 Y = 6 PRICE = 13 nPATH = 1	(2,3) X = null Y = 4 PRICE = 17 nPATH = 1
(3,0) X = 8 Y = null PRICE = 13 nPATH = 1	(3,1) X = 5 Y = null PRICE = 16 nPATH = 1	(3,2) X = 8 Y = null PRICE = 19 nPATH = 1	(3,3) X = null Y = null PRICE = 21 nPATH = 1

אתחול הערכים, מציאת העלות המינימאלית ומס' המסלולים המינימאליים הקיימים.

סיבוכיות - O(M*N)

1. מילוי "מחיר ההגעה" של השורה הראשונה + נאתחל את מס' המסלולים להיות 1.
2. מילוי "מחיר ההגעה" של העמודה הראשונה + נאתחל את מס' המסלולים להיות 1.
3. נמשיך למלא את "מחיר ההגעה" + מס' המסלולים שורה שורה (נתחיל מ-(1,1)).
4. המטריצה במיקום (4,4) מכילה את "מחיר ההגעה" ומס' המסלולים המינימאלי.

```
Node bestPath(Node mat[][])
    Int m =     mat.length();
    Int n =     mat[0].length();

    for( j = 1 to n )
        mat[0][j].price   =   mat[0][j-1].price + mat[0][j-1].x;
        mat[0][j].nPath   =   1;
    for( i = 1 to m )
        mat[i][0].price   =   mat[i-1][0].price + mat[i-1][0].y;
        mat[i][0].nPath   =   1;

    for( i = 1 to m )
        for( j = 1 to n )
            a   =   mat[i-1][j].price + mat[i-1][j].y;
            b   =   mat[i][j-1].price + mat[i][j-1].x;
            if( a < b )
                mat[i][j].price  = a;
                mat[i][j].nPath  = mat[i-1][j].nPath;
            else if( a > b )
                mat[i][j].price  = b;
                mat[i][j].nPath  = mat[i][j-1].nPath;
```

```
        else
                mat[i][j].price   = a;
                mat[i][j].nPath = mat[i-1][j].nPath + mat[i][j-1].nPath;
return mat[m][n];
```

<u>מציאת מסלול יחיד בעל עלות מינימאלית.</u>

O(M+N)סיבוכיות -

1. נגדיר String שיתאר את המסלול (down \ right)
2. מתחילים מהסוף להתחלה i = m-1, j = n-1.
3. נבדוק מאיפה הגענו בדרך "זולה" יותר (מלמעלה או משמאל) כל עוד I && j > 0
4. נוסיף ל-String את הדרך
   a. אם מלמעלה יותר זול – נוסיף "down".
   b. אם משמאל יותר זול – נוסיף "right"

דוגמא ל-String סופי : "down right right down right down"

5. במקרה שבו i > 0 נוסיף "down" ל-String.
6. במקרה שבו j > 0 נוסיף "right" ל-String.
7. אורך המסלול יהיה m+n.

```
String onePath(Node mat[][])
        Int m           =       mat.length();
        Int n           =       mat[0].length();
        Int I           =       m-1;
        Int j           =       n-1;
        String ans      =       "";

        while( I > 0 && j > 0 )
                a       =       mat[i-1][j].price + mat[i-1][j].y;
                b       =       mat[i][j-1].price + mat[i][j-1].x;
                if ( a < b )
                        ans     =       "down" + ans;
                        i--;
                if ( a > b )
                        ans     =       "right" + ans;
                        j--;
```

```
while( i > 0 )
        ans     =       "down";
        i--;
while( j > 0 )
        ans     =       "right";
        j--;

return ans;
```

<u>מציאת כל המסלולים הטובים ביותר.</u>

O(M+N)*nPathסיבוכיות –

1. נשתמש בפונקציה רקורסיבית.
2. נגדיר מס' שלם teta שיהווה גבול למס' המסלולים, מכיוון שהוא עלול להיות גדול מידי.
3. נשמור את המסלולים ב-ArrayList<String>

```
Public void allPathsRecurs(teta)
        If( nPaths <= teta )
                ArrayList<String> paths    =       new ArrayList<String<(nPath);
```