

בעיית השוקולד

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

הוכחה באינדוקציה

עבור השבירה הראשונה במיקום i -י נשלם: $i * (n - i)$
 על השבירות של החלק השמאלי נשלם: $i * (n - i)$
 על השבירות של החלק הימני נשלם: $((n - i)(n - i - 1)) / 2$
 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

- $[a, b] \rightarrow [n, b]$
- $[a, b] \rightarrow [a, m]$

2. מותר לרוקן כלי, אך חובה עד הסוף.

- $[a, b] \rightarrow [0, b]$
- $[a, b] \rightarrow [a, 0]$

3. מותר לשפוך מכלי אחד לכלי אחר, אך חובה עד הסוף או עד ככל שניתן.

- $[a, b] \rightarrow [\min(a + b, n), a + b - \min(a + b, n)]$
- $[a, b] \rightarrow [a + b - \min(a + b, m), \min(a + b, m)]$

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

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

מטריצה שכנויות בוליאניות המייצגת את מצבי הבקבוק.

השורות והעמודות מייצגים את הקודקודים.

כל תא מייצג האם ניתן לעבור ממצב אחד לשני באופן ישיר.

```
initRibs(n, m)
```

```
  dim = (n + 1) * (m + 1)
```

```
  index1, index2
```

```
  For (i = 0 to n)
```

```
    For (j = 0 to m)
```

Input: הגבהים של הבקבוקים

Output: מטריצה

פונקציית עזר: getIndex

i – גובה המים במצב הנוכחי בבקבוק הראשון.

j – גובה המים במצב הנוכחי בבקבוק השני.

```
  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
```

<pre> For (i = 0 to dim) mat [i] [i] = true return mat </pre>	
<pre> 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 </pre>	<p>מציאת מטריצה בוליאנית המייצגת האם גם יש קשרים עקיפים בין קודקודים.</p> <p>Input: מטריצת שכנויות (קשר ישיר)</p> <p>Output: מטריצה.</p>
<pre> return mat[v1][v2] </pre>	<p>האם יש מסלול כלשהו בין v1 ל-v2?</p> <p>נקבל true אם כן ו-false אם לא.</p>
<pre> 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 </pre>	<p>מטריצה המציגה בכל תא את המסלולים הישירים \ עקיפים הקצרים ביותר בין הקודקודים.</p> <p>Input: מטריצת שכנויות בוליאנית.</p> <p>Output: מטריצת סטרינגים של המסלולים.</p>
<pre> n = mat.length For (i = 0 to n) IF (mat[0][i] == false) return false return true </pre>	<p>האם הגרף קשיר?</p> <p>Input: מטריצה בוליאנית לאחר FW עם כל הקשרים.</p> <p>Output: True \ False</p> <p>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.</p>
<pre> 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 </pre>	<p>כמה רכיבי קשירות יש בגרף?</p> <p>בנוסף, במערך helper ניתן לראות איזה קודקודים יחד באותו רכיב קשירות.</p>
	<p>סידור מטריצה במקרה שהקודקודים לא לפי הסדר</p>

Floyd Warshall (גרף ממושקל)

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

$V = \text{vertices.length}$

$\text{dist} = V \times V$ matrix of minimum distance initialized to ∞

For each vertex v

$\text{dist}[v][v] = 0$

For each edges (u, v)

$\text{dist}[u][v] = \text{weights}(u, v)$

For $(k = 0 \text{ to } n)$

For $(i = 0 \text{ to } n)$

For $(j = 0 \text{ to } n)$

$\text{dist}[i][j] = \text{Min}(\text{dist}[i][j], \text{dist}[i][k] + \text{dist}[k][j])$

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

בניית מטריצת השכנויות
(אם לא נקבל אותה)

$n = \text{mat.length}$

// אתחול המשקלים במטריצה בין קודקודים שמחוברים באופן ישיר

For $(i = 0 \text{ to } n)$

For $(j = 0 \text{ to } n)$

IF $(\text{mat}[i][j] == 1)$

$\text{mat}[i][j] = \text{weight}[i] + \text{weight}[j]$

// הרצת פלואיד וורשאל למציאת מינימום מרחק

For $(k = 0 \text{ to } n)$

For $(i = 0 \text{ to } n)$

For $(j = 0 \text{ to } n)$

$\text{dist}[i][j] = \text{Min}(\text{dist}[i][j], \text{dist}[i][k] + \text{dist}[k][j])$

// תיקון המשקלים שנספרו פעמיים

For $(i = 0 \text{ to } n)$

For $(j = 0 \text{ to } n)$

IF $(\text{mat}[i][j] != \infty \text{ AND } i != j)$

$\text{mat}[i][j] += \text{weight}[i] + \text{weight}[j]$

$\text{mat}[i][j] /= 2$

return mat

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.

*** לבדוק אלגור' שנותן דוגמא למעגל שלילי ***

האם יש מעגל שלילי בגרף?

נריץ FW למציאת מטריצת המסלולים הקצרים ביותר

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

For $(i = 0 \text{ to } n)$

IF $(\text{mat}[i][i] < 0)$ return true

return false

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

For $(i = 0 \text{ to } n)$

For $(j = 0 \text{ to } n)$

if $(\text{mat}[i][j] < 0)$ return true

return false

shortestPath(mat[][])

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

n = mat.length

path[][] = new String[n][n]

Input: מטריצת שכנויות בוליאנית.

Output: מטריצת סטרינגים של המסלולים.

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 neighbor 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 ≠ ∅ AND OpenB ≠ ∅ 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 )
```

<pre> 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 </pre>	
<h2 style="text-align: center;">BFS</h2>	
<pre> 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] </pre>	<p>שיטה לחשיפת גרף מקודקוד מקור.</p> <p>משמש למציאת המרחקים הקצרים ביותר של כל קודקוד מקודקוד המקור ואת האבא של כל אחד לצורך מציאת המסלול.</p> <p>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.</p>
<p>איך אפשר לדעת שהגרף קשיר? נעבור על מערך הצבעים, אם כולם מסומנים בשחור, סימן שהגרף קשיר (האלגוריתם הצליח להגיע מקודקוד המקור לכל שאר הקודקודים).</p>	
<p>איך אפשר לשחזר את העץ? נעבור על מערך האבות, כל תא מייצג קודקוד, הערך באותו באותו התא הוא קודקוד האב.</p>	
<p>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).</p>	
<p>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.</p>	
<p>מה המרחק הקצר ביותר בין שני קודקודים?</p>	

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.	
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	הגדרה: קוטר בגרף הוא המרחק המקסימאלי מבין כל המרחקים הקצרים ביותר בין כל שני קודקודים בגרף. ניתן למצוא קוטר ע"י BFS
דוגמא למסלול בגודל הקוטר: בקריאה השניה של האלגוריתם, נחזור במערך האבות אחורה מהתא (הקודקוד) עם הערך המקסימאלי.	
DFS	
<pre>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 (color[u] = white) u.startTime = time time++ DFS-visit(u) v.endTime = time time++ color[v] = black</pre>	

שריפת עלים (למציאת קוטר, רדיוס ומרכזי העץ)

fire(vector[] tree)

radius = 0

diameter = 0

numOfCenters = 0

vertex = 0

leaf = 0

leaves = new vector(n)

defrees = new int[n]

הגדרה: עץ הוא גרף 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 $1 \leq$ זוגי
אם $2 \leq$ אי זוגי
3. רדיוס?
אם הקוטר זוגי - \leq כמות השריפות
אם הקוטר אי זוגי - \leq כמות השריפות + 1
קוטר?
4. אם הקוטר זוגי - \leq פעמיים הרדיוס
אם הקוטר אי זוגי - \leq פעמיים הרדיוס - 1

[illegible]

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:

- בכל עץ $(|E| = |V| - 1)$
- לפי למת לחיצות הידיים $<---$ סכום הדרגות $= 2(|V| - 1)$
- בכל עץ יש לפחות שני עלים.

האלגוריתם:

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

דוגמא:

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

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

אווילר

יהי גרף $G(V,E)$ גרף לא מכוון.

מסלול $P_{x,y}$ נקרא מסלול אוילר ב- G , אם הוא עובר בכל הצלעות של G כך שכל צלע מופיעה בו פעם אחת בלבד ו- $x \neq y$.

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.

1. יש בגרף G מעגל אוילר (גרף אוילריאני) אם"מ G קשיר וכל דרגות הגרף זוגיות.
2. יש בגרף G מסלול אוילר אם"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 neighbors
        S.pop
        C.push(u)
    else
        v = some v in u neighbors
        S.push(v)
return C

```

קידוד הופמן

1. אם לא ממוין – הקוד הרגיל $O(n \log n)$
2. אם ממוין – שתי מחסניות $O(n)$

```

huffman (C)    // O(n log n)
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)

```

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
<pre>// 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</pre>

בעיית המטוס עבור 3x3

1. נגדיר מטריצה 4x4.
2. נאתחל את כל העלויות \ משקלים של המטריצה.

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

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];

```

מציאת מסלול יחיד בעל עלות מינימאלית.

$O(M+N)$ 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 $j > 0 \ \&\& \ i > 0$
4. נוסיף ל-String את הדרך
 - a. אם מלמעלה יותר זול – נוסיף "down".
 - b. אם משמאל יותר זול – נוסיף "right".
- String דוגמא ל-"down right right down right down" סופי :
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;

```

מציאת כל המסלולים הטובים ביותר.

– $O((M+N)*nPath)$ סיבוכיות

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);

```