Algorithm Floyd-Warshall

**אלגוריתם פלויד־וורשל** הוא אלגוריתם למציאת כל משקלי המסלולים הקלים/הקצרים ביותר בין כל זוגות הצמתים בגרף. אלגוריתם זה פועל עבור גרף מכוון משוקלל בעל משקלים (אורכי קשתות) חיוביים ושליליים, חסר מעגלים שליליים. האלגוריתם מבוסס על פרדיגמת התכנון הדינמי.

**k - קודקוד מתווך**

**k**

**i j**

**………….**

**Computing the shortest-path weights bottom up**

**for** (**int** k = 0; k < n; k++)

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

**for** (**int** j = 0; j < n; j++)

) **נוסחה (**

:

**נוסחה**

**if (mat[i][k] != *inf* && mat[k][j] != *inf*)**

**mat[i][j] = Math.*min*(mat[i][j], mat[i][k]+mat[k][j])**

**weight**

Build transitive closure of a directed graph

**for** (**int** k = 0; k < n; k++)

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

**for** (**int** j = 0; j < n; j++)

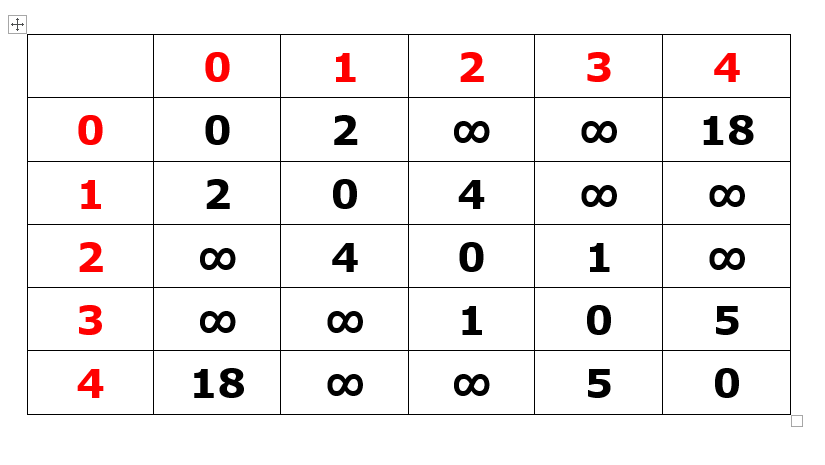
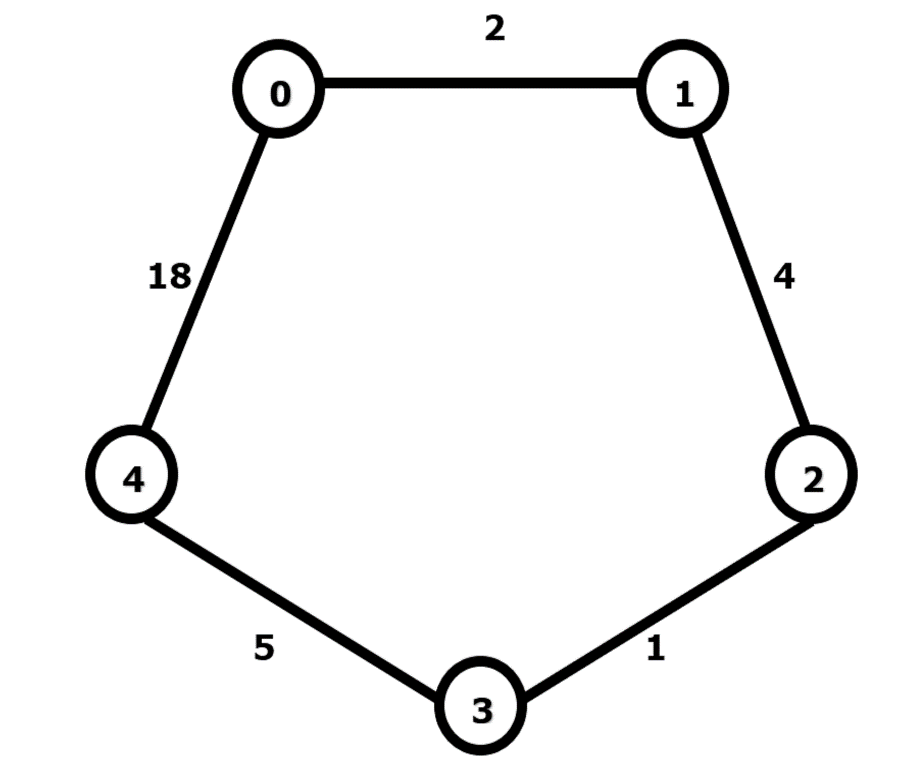
) **נוסחה (**

**נוסחה**

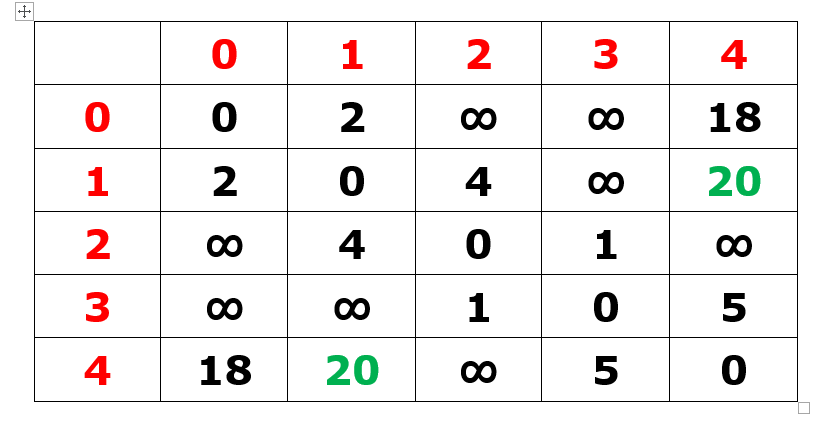
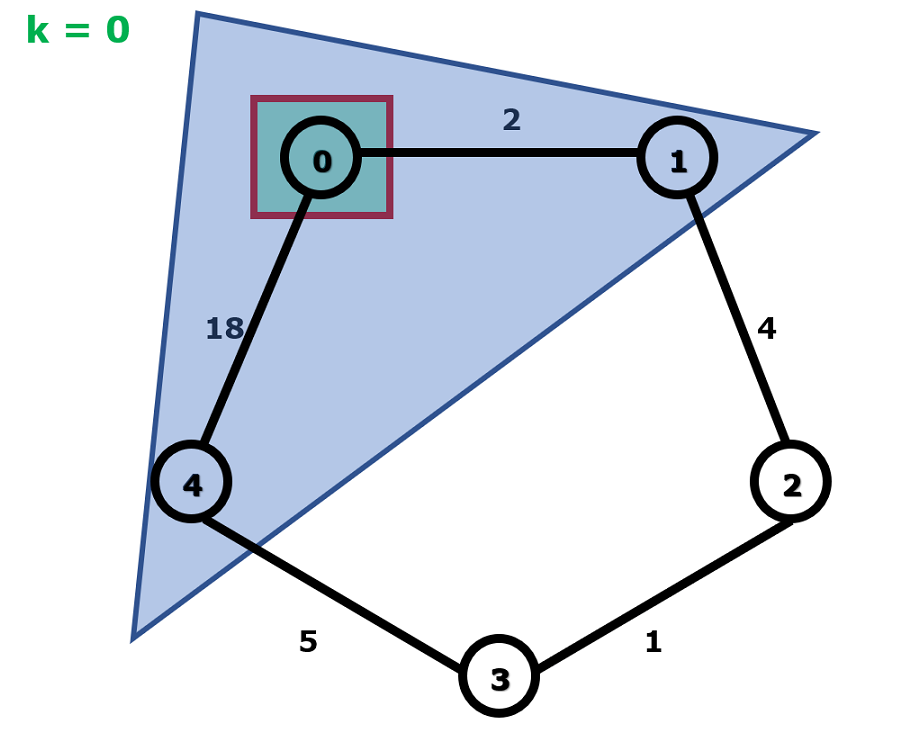
**mat[i,j] 🡨 mat[i,j] or (mat(i,k) and mat(k,j))**

boolean

ממטריצת שכנות ← למטריצת קשירות



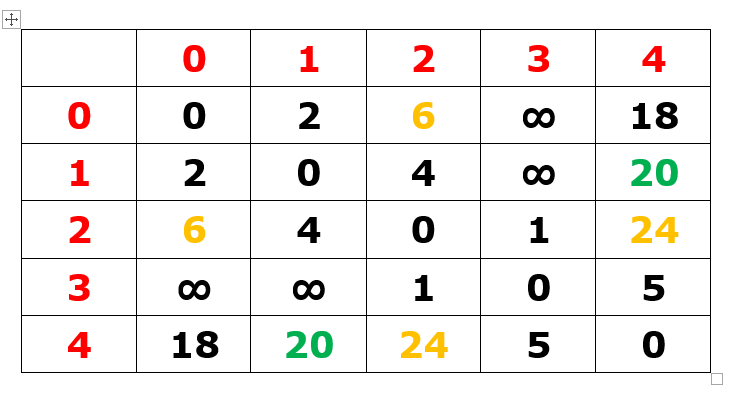
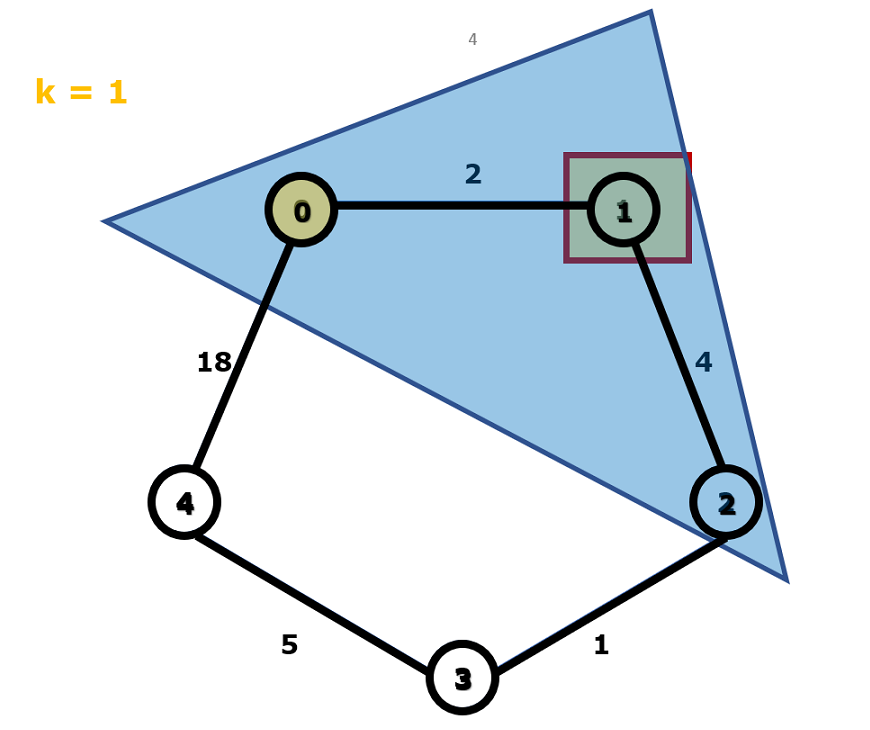
מטריצת שכנות



**K = 0**

**Distance (4🡪0🡪1) = 18 + 2 = 20**

**Distance (1🡪0🡪4) = 18 + 2 = 20**

****

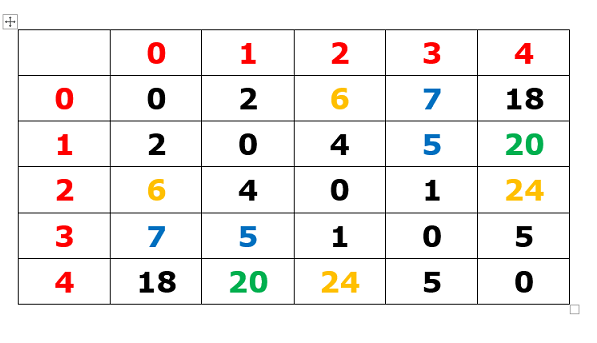
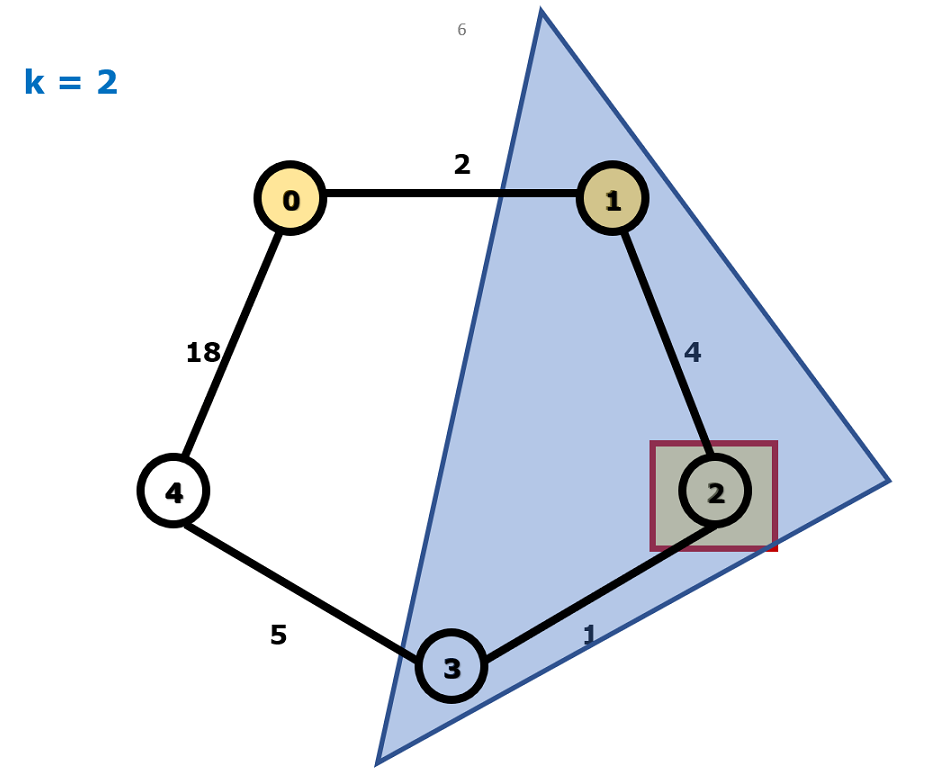
**K = 1**

**Distance (0🡪1🡪2) = 2 + 4 = 6**

**Distance (2🡪1🡪0) = 4 + 2 = 6**

**Distance (4🡪0🡪1🡪2) = 18 + 2 + 4 = 24**

**Distance (2🡪1🡪0🡪4) = 4 + 2 + 18 = 24**



**K = 2**

**Distance (1🡪2🡪3) = 4 + 1 = 5**

**Distance (3🡪2🡪1) = 1 + 4 = 5**

**Distance (0🡪1🡪2🡪3) = 2 + 4 + 1 = 7**

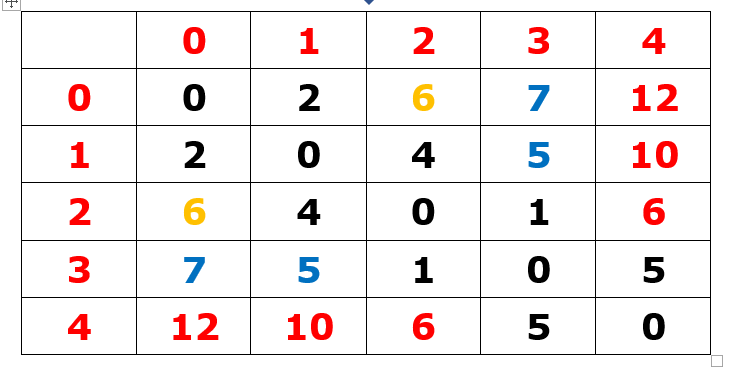
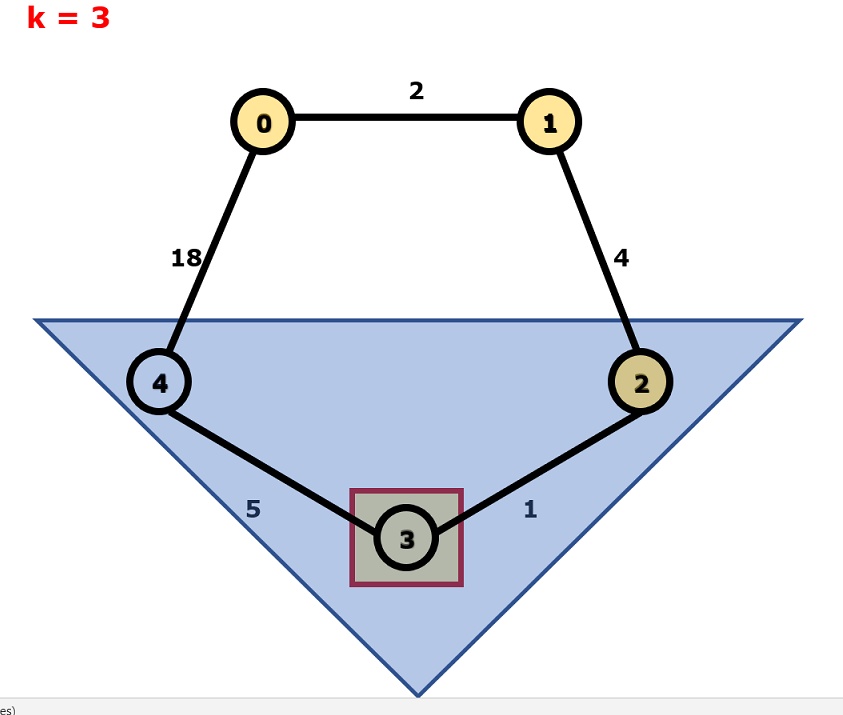
**Distance (3🡪2🡪1🡪0) = 1 + 4 + 2 = 7**

**Distance (4🡪0🡪1🡪2🡪3) = 18 + 2 + 4 + 1 = 25**

**Distance (3🡪2🡪1🡪0🡪4) = 1 + 4 + 2 + 18 = 25**

**Distance (4🡪3) = 5**

**Distance (3🡪4) = 5**



**Distance (2🡪3🡪4) = 1 + 5 = 6**

**Distance (4🡪3🡪2) = 5 + 1 = 6**

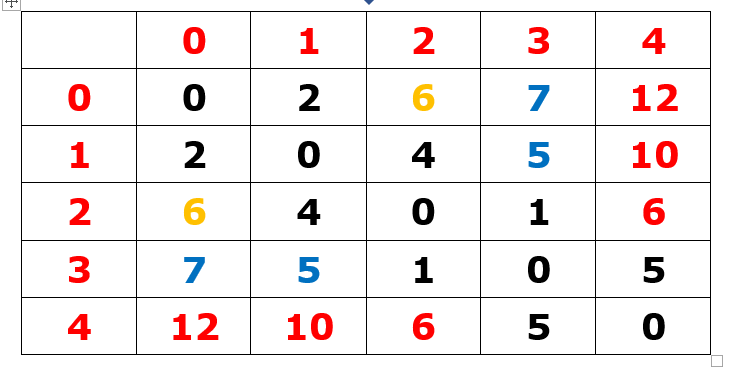
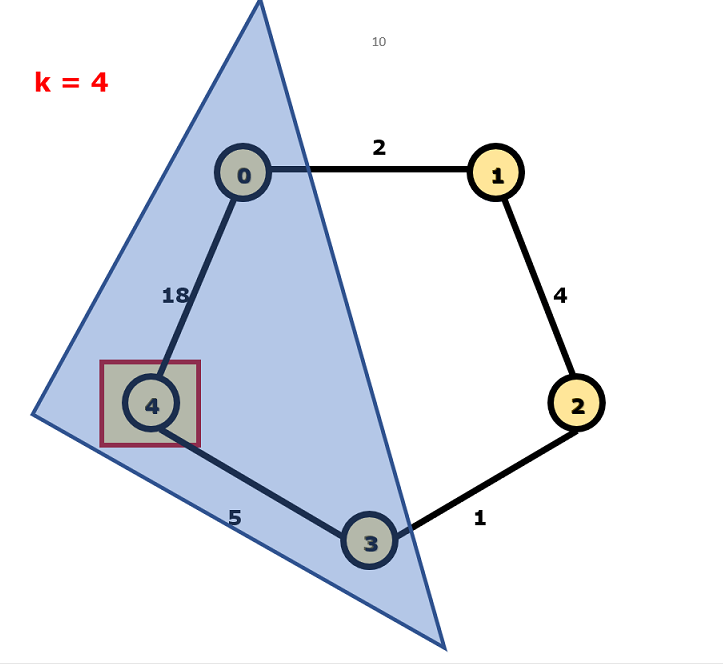
**K = 3**

**Distance (1🡪2🡪3🡪4) = 4 + 1 + 5 = 10**

**Distance (4🡪3🡪2🡪1) = 5 + 1 + 4 = 10**

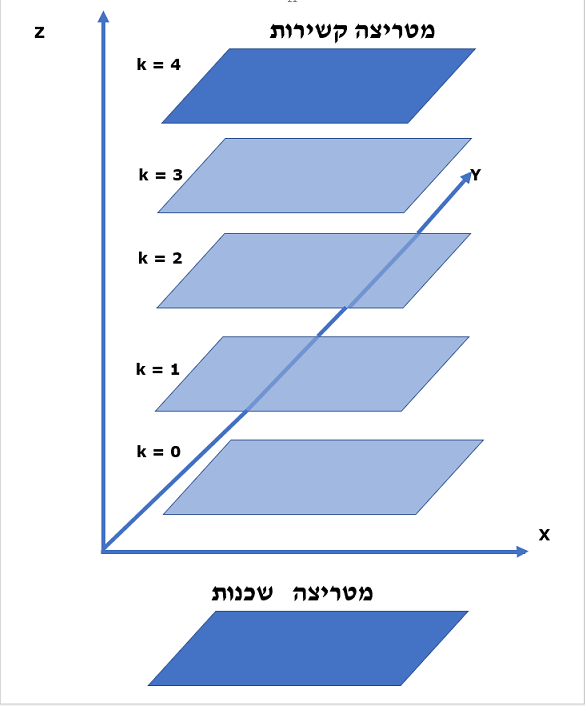
**Distance (0🡪1🡪2🡪3🡪4) = 2 + 4 + 1 + 5 = 12**

**Distance (4🡪3🡪2🡪1🡪0) = 5 + 1 + 4 + 2 = 12**



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** |
| **0** | **0** | **2** | **6** | **7** | **12** |
| **1** | **2** | **0** | **4** | **5** | **10** |
| **2** | **6** | **4** | **0** | **1** | **6** |
| **3** | **7** | **5** | **1** | **0** | **5** |
| **4** | **12** | **10** | **6** | **5** | **0** |

**מטריצת קשירות**

****

**static** **int** *inf* = Integer.***MAX\_VALUE***;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** **boolean**[][] initBool(){

**boolean** [][] mat = {{ **false**, **true**, **false**, **false**, **true**},

{ **true**, **false**, **true**, **false**, **false**},

{ **false**, **true**, **false**, **true**, **false**},

{ **false**, **false**, **true**, **false**, **true**},

{ **true**, **false**, **false**, **true**, **false**}};

**return** mat;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** **int**[][] initInt(){

**int** [][] mat = {{ 0, 2, *inf*, *inf*, 18},

{ 2, 0, 4, *inf*, *inf*},

{ *inf*, 4, 0, 1, *inf*},

{ *inf*, *inf*, 1, 0, 5},

{ 18, *inf*, *inf*, 5, 0}};

**return** mat;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** **void** printBooleanMatrix(**boolean**[][] mat){

**for**(**int** i=0; i<mat.length; i++){

**for**(**int** j=0; j<mat[0].length; j++){

**if** (i == j)

System.***out***.printf("%7s", " -----") ;

**else**

System.***out***.printf("%7s", mat[i][j]) ;

}

System.***out***.println();

}

System.***out***.println("--------------------------------------");

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** **void** printIntMatrix(**int**[][] mat){

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

**for**(**int** i=0; i<mat.length; i++){

**for**(**int** j=0; j<mat[0].length; j++){

**if** (mat[i][j] == *inf*)

System.***out***.format("%6s,","inf");

**else**

System.***out***.format("%6d,",mat[i][j]);

}

System.***out***.println();

}

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

**משימה: לייצר שתי פונקציות:**

1. חישוב משקלי המסלולים הקלים/הקצרים ביותר בין כל זוגות הצמתים בגרף לפי אלגוריתם פלויד־וורשל:

public static int[][] buildWeightMatrix(int [][] mat)

1. יצירת מטריצה בוליאנית של קיום המסלולים הקלים/הקצרים ביותר בין כל זוגות הצמתים בגרף לפי אלגוריתם פלויד־וורשל:

//Build transitive closure of a directed graph

**public** **static** **boolean**[][] buildBooleanMatrix(**boolean** [][] bm)