**נושאים לתרגול – 2**

1. מציאת מטריצת מסלולים Floyd - Warshall
2. האם יש מסלול בין קודקוד לבין קודקוד
3. מציאת מסלולים עצמם,
4. כמה רכיבי קשירות יש בגרף
5. מהם הקודקודים ברכיבי הקשירות
6. האם גרף קשיר
7. בעיית הבקבוקים – האם קיים מסלול בין מצב כלשהו (i,j) של תכולה בשני בקבוקים למצב כלשהו k,l)).
8. אם כן, תנו דוגמא.
9. בעיית השוקולד

שיעור שעבר: מטריצה שכנות – ערך true שיש צלע בין קודוקדים שמאפיינים מצב תכולה של הבקבוקים

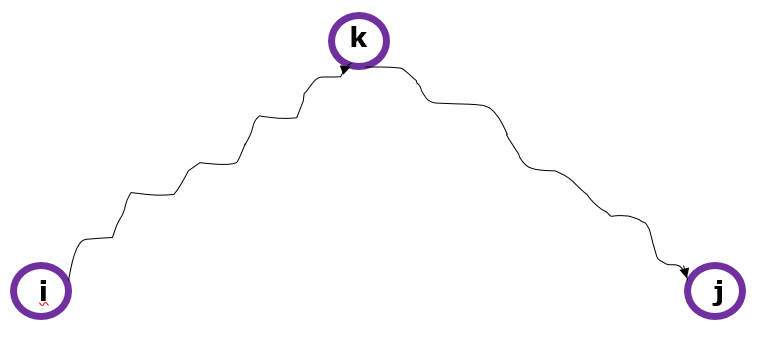
האם קיים מסלול בין המצבים? אלגוריתם של פלויד-וורשל עונה לשאלה זאת ומוצא מסלול קצר/זול ביותר

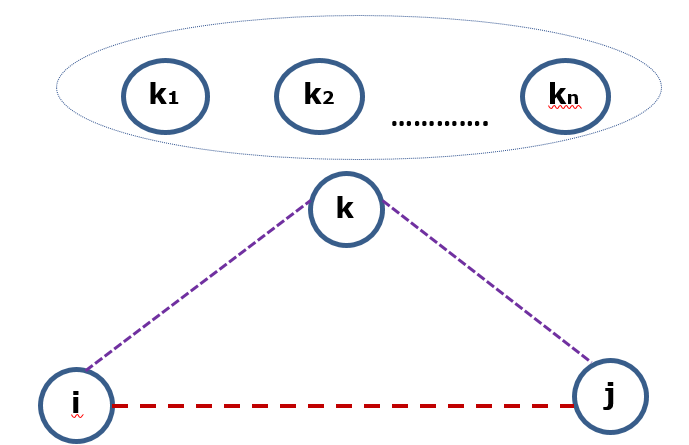
Algorithm Floyd-Warshall

**אלגוריתם פלויד־וורשל** הוא אלגוריתם למציאת כל 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**.

**רעיון האלגוריתם:**

**k - קודקוד מתווך**

****

****

Build transitive closure of a graph

**מציאת כל המסלולים בין כל זוגות הצמתים בגרף**

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

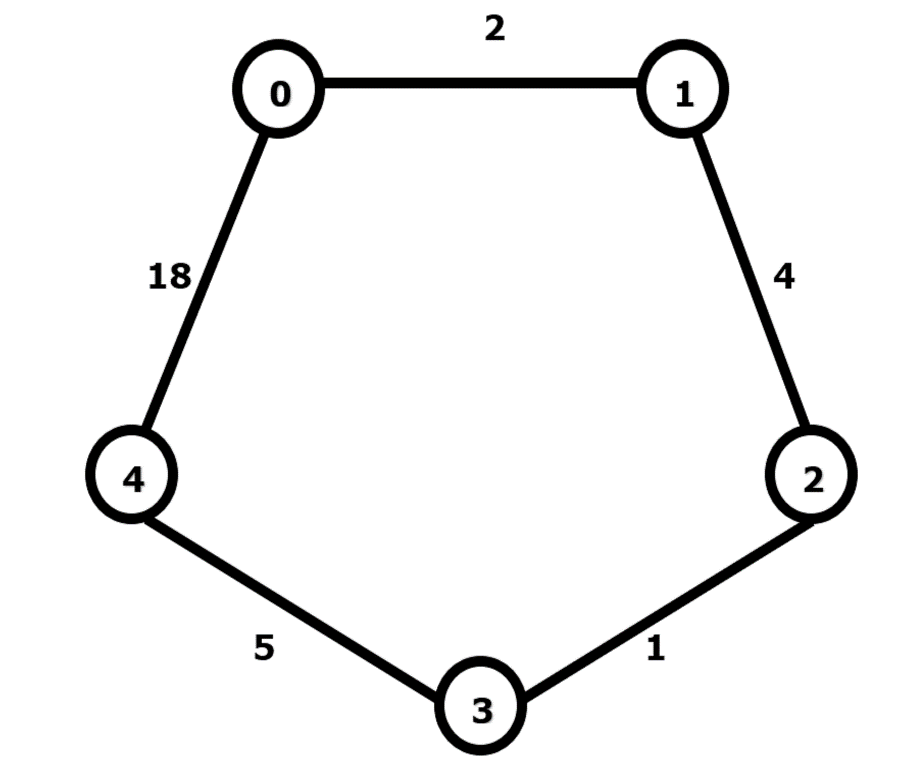
**for** (**int** i = 0; i < n; i++)

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

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

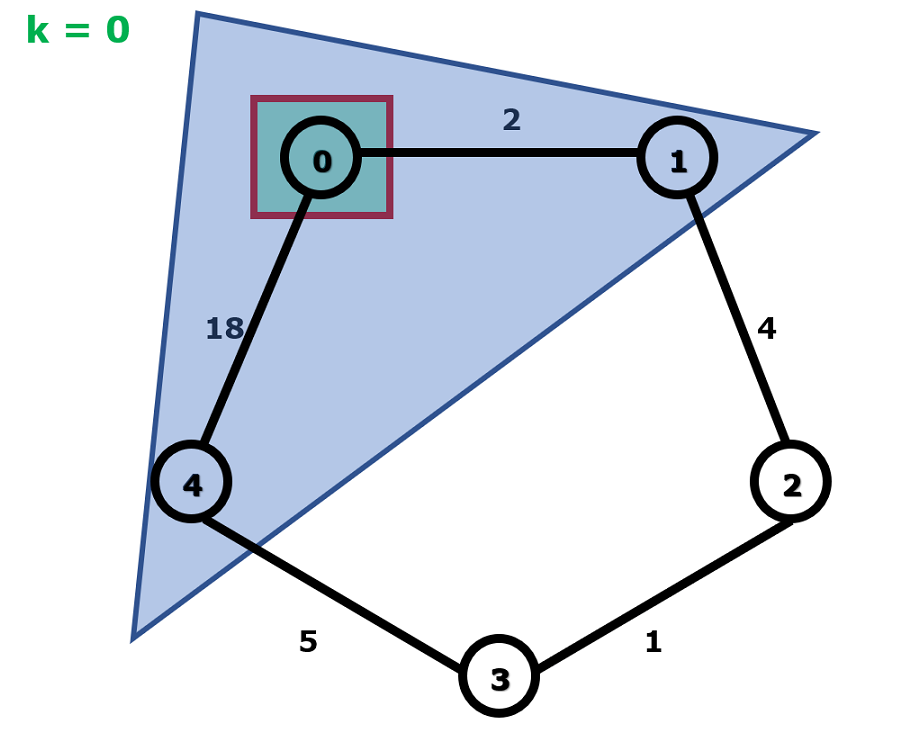
Boolean Matrix

ממטריצת שכנות ← למטריצת קשירות



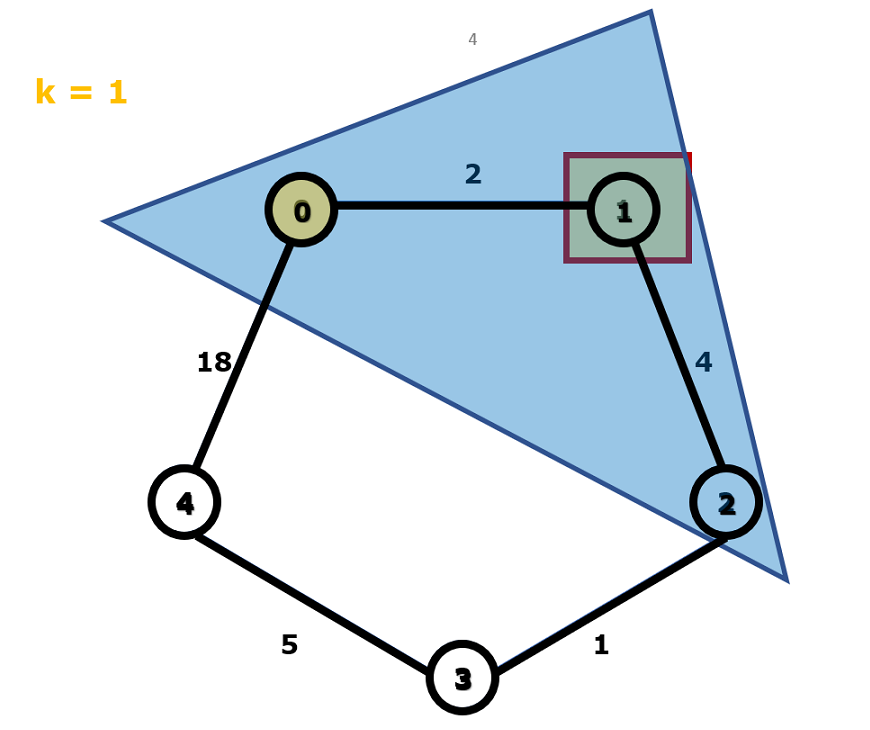
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | false | false | true |
| 1 | true | true | true | false | false |
| 2 | false | true | true | true | false |
| 3 | false | false | true | true | true |
| 4 | true | false | false | true | true |

מטריצת שכנות



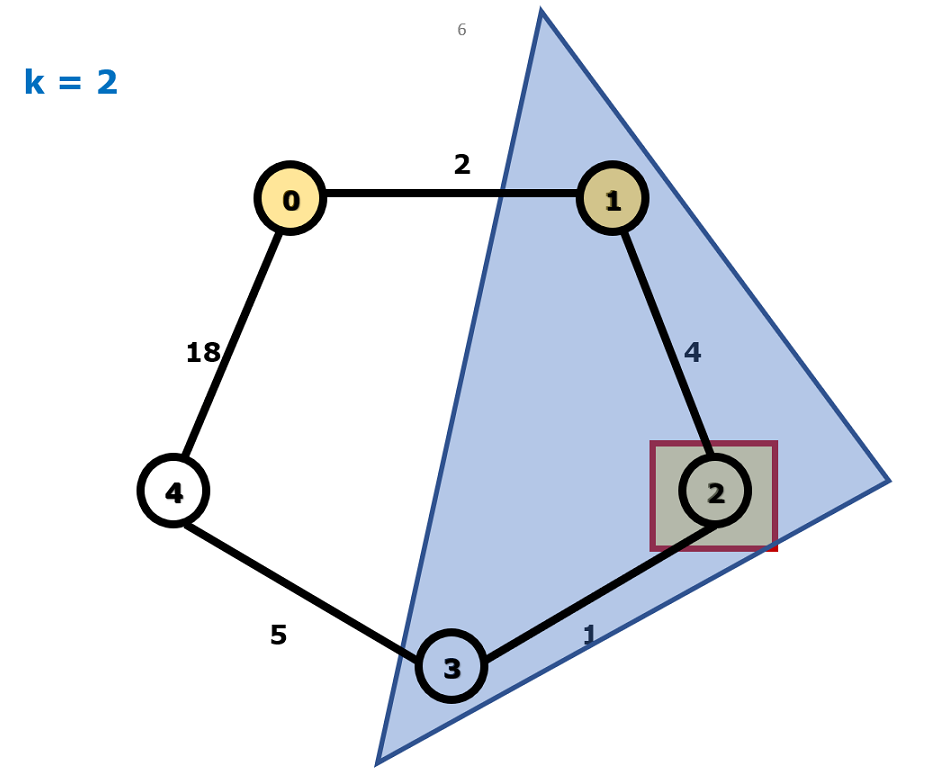
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | false | false | true |
| 1 | true | true | true | false | true |
| 2 | false | true | true | true | false |
| 3 | false | false | true | true | true |
| 4 | true | true | false | true | true |

**K = 0**

****

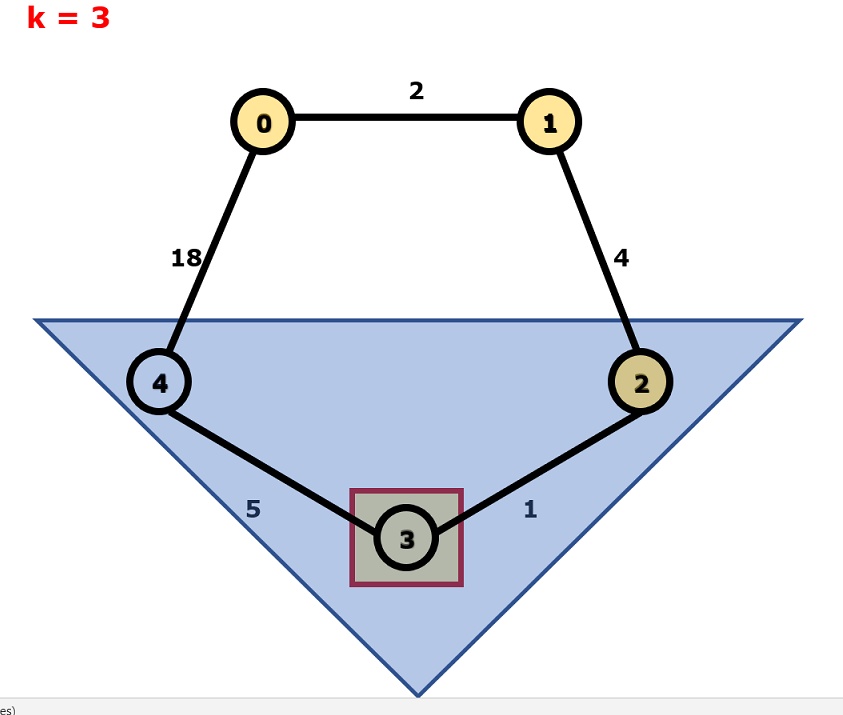
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | true | false | true |
| 1 | true | true | true | false | true |
| 2 | true | true | true | true | true |
| 3 | false | false | true | true | true |
| 4 | true | true | true | true | true |

**K = 1**



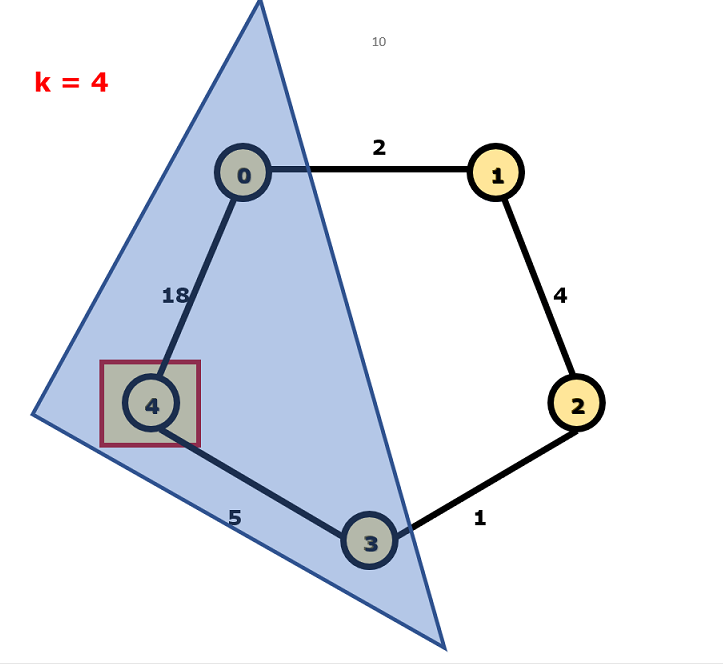
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | true | true | true |
| 1 | true | true | true | true | true |
| 2 | true | true | true | true | true |
| 3 | true | true | true | true | true |
| 4 | true | true | true | true | true |

**K = 2**



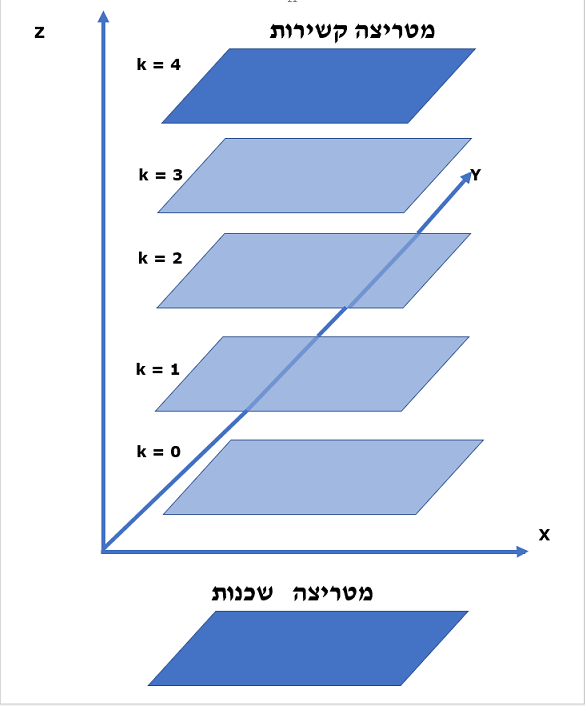
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | true | true | true |
| 1 | true | true | true | true | true |
| 2 | true | true | true | true | true |
| 3 | true | true | true | true | true |
| 4 | true | true | true | true | true |

**K = 3**



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | true | true | true |
| 1 | true | true | true | true | true |
| 2 | true | true | true | true | true |
| 3 | true | true | true | true | true |
| 4 | true | true | true | true | true |

**מטריצת קשירות**

****

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

**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** **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("--------------------------------------");

}

**משימה:**

1. ליצור פונקציה שבונה מטריצה בוליאנית של קיום המסלולים בין כל זוגות הצמתים בגרף לפי אלגוריתם פלויד־וורשל:

//Build transitive closure of a graph

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

**Constructing paths**

**מציאת מסלולים עצמם בין כל זוגות הצמתים בגרף**

**int** n = bm.length;

// path matrix initialization

String [][] pathMat = **new** String[n][n];

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

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

**if** (bm[i][j])

pathMat[i][j] = " "+i+"->"+j+" ";

**else**

pathMat[i][j] = "";

}

}

// path matrix building

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

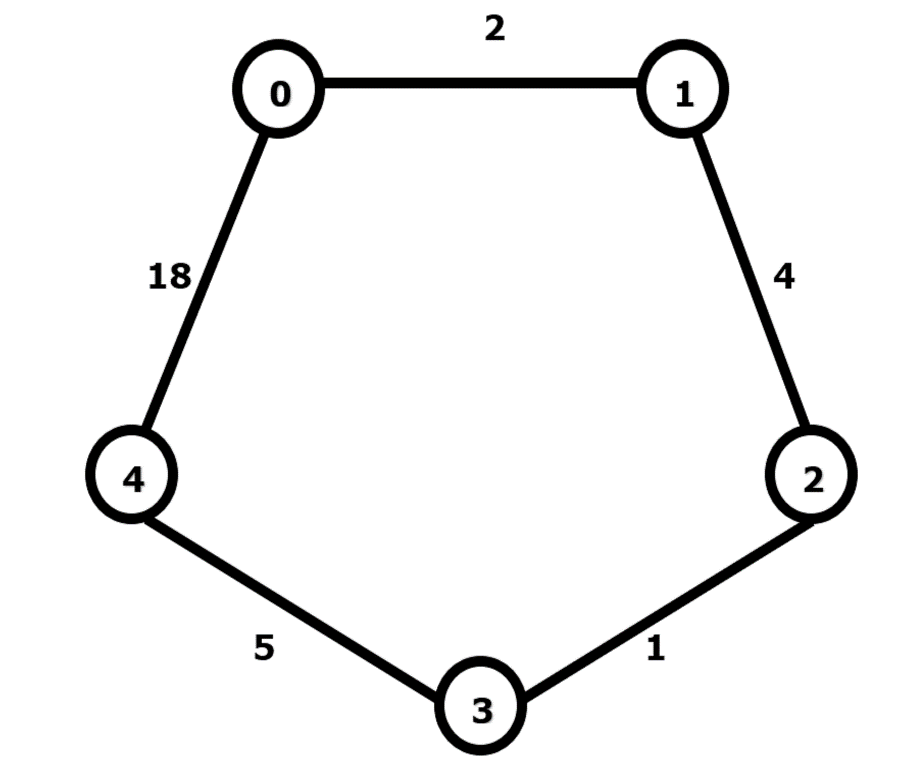
**for** (**int** i = 0; i < n; i++)

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

**if (bm [i][j]==false and (bm [i][k]==true and bm[k][j]==true)) pathMat[i][j] = pathMat[i][k]+pathMat[k][j]** // **bm [i,j] 🡨 bm [i,j] or (bm (i,k) and bm (k,j))**

String Matrix

ממטריצת שכנות ← למטריצת מסלולים

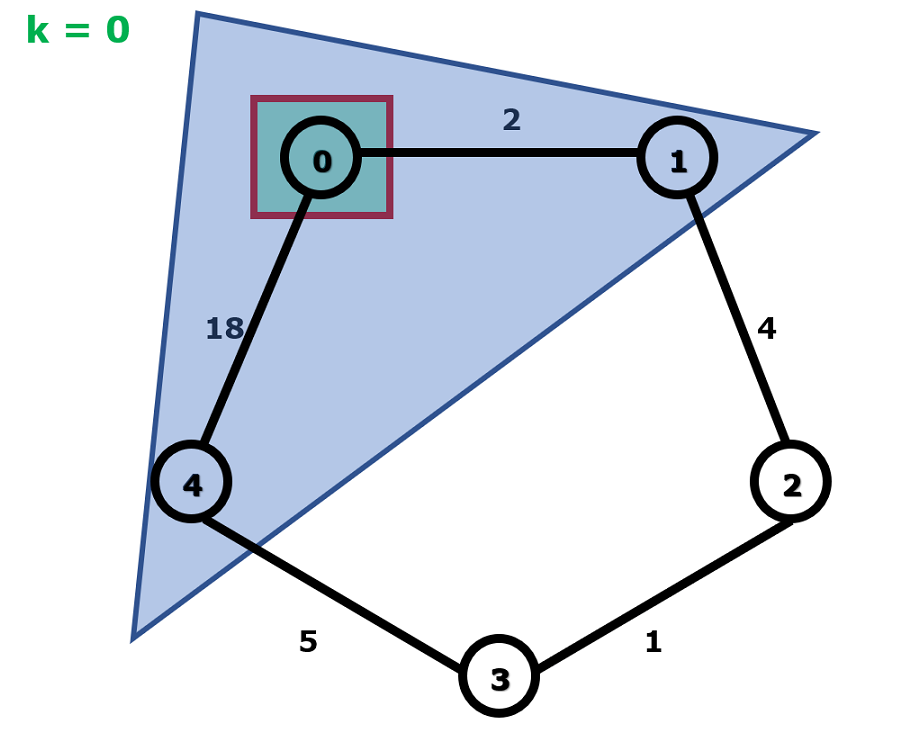


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | false | false | true |
| 1 | true | true | true | false | false |
| 2 | false | true | true | true | false |
| 3 | false | false | true | true | true |
| 4 | true | false | false | true | true |

← מטריצת שכנות

מטריצת מסלולים

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** |
| **0** | **0🡪0** | **0🡪1** |  |  | **0🡪4** |
| **1** | **1🡪0** | **1🡪1** | **1🡪2** |  |  |
| **2** |  | **2🡪1** | **2🡪2** | **2🡪3** |  |
| **3** |  |  | **3🡪2** | **3🡪3** | **3🡪4** |
| **4** | **4🡪0** |  |  | **4🡪3** | **4🡪4** |

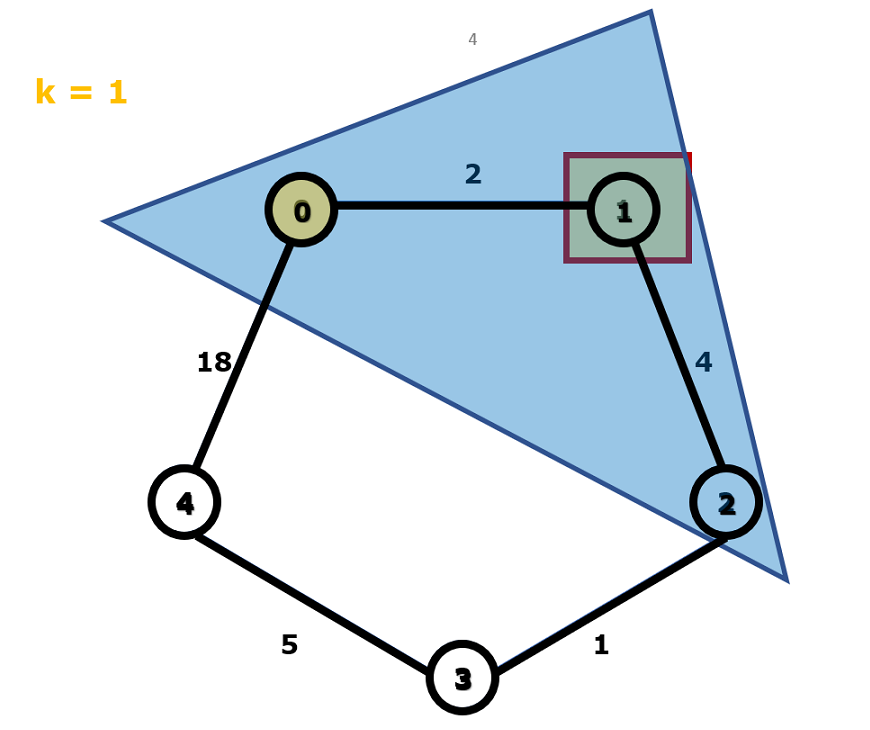


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | false | false | true |
| 1 | true | true | true | false | true |
| 2 | false | true | true | true | false |
| 3 | false | false | true | true | true |
| 4 | true | true | false | true | true |

**K = 0**

מטריצת מסלולים

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** |
| **0** | **0🡪0** | **0🡪1** |  |  | **0🡪4** |
| **1** | **1🡪0** | **1🡪1** | **1🡪2** |  | **1🡪0🡪4** |
| **2** |  | **2🡪1** | **2🡪2** | **2🡪3** |  |
| **3** |  |  | **3🡪2** | **3🡪3** | **3🡪4** |
| **4** | **4🡪0** | **4🡪0🡪1** |  | **4🡪3** | **4🡪4** |

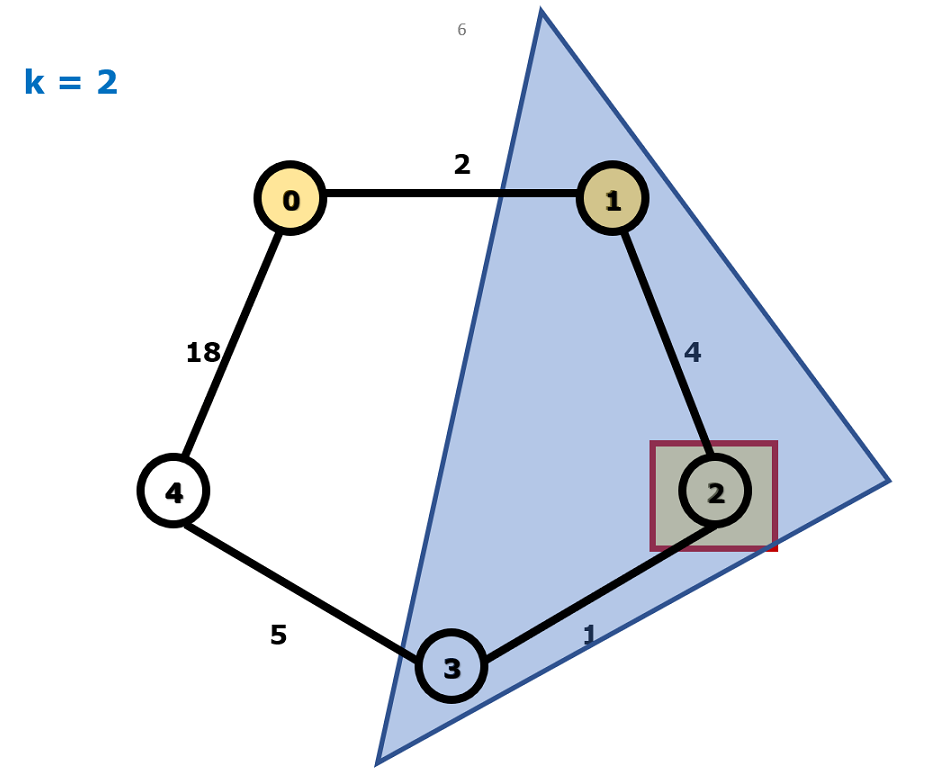
****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | true | false | true |
| 1 | true | true | true | false | true |
| 2 | true | true | true | true | true |
| 3 | false | false | true | true | true |
| 4 | true | true | true | true | true |

**K = 1**

מטריצת מסלולים

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** |
| **0** | **0🡪0** | **0🡪1** | **0🡪1🡪2** |  | **0🡪4** |
| **1** | **1🡪0** | **1🡪1** | **1🡪2** |  | **1🡪0🡪4** |
| **2** | **2🡪1🡪0** | **2🡪1** | **2🡪2** | **2🡪3** | **2🡪1🡪0🡪4** |
| **3** |  |  | **3🡪2** | **3🡪3** | **3🡪4** |
| **4** | **4🡪0** | **4🡪0🡪1** | **4🡪0🡪1🡪2** | **4🡪3** | **4🡪4** |

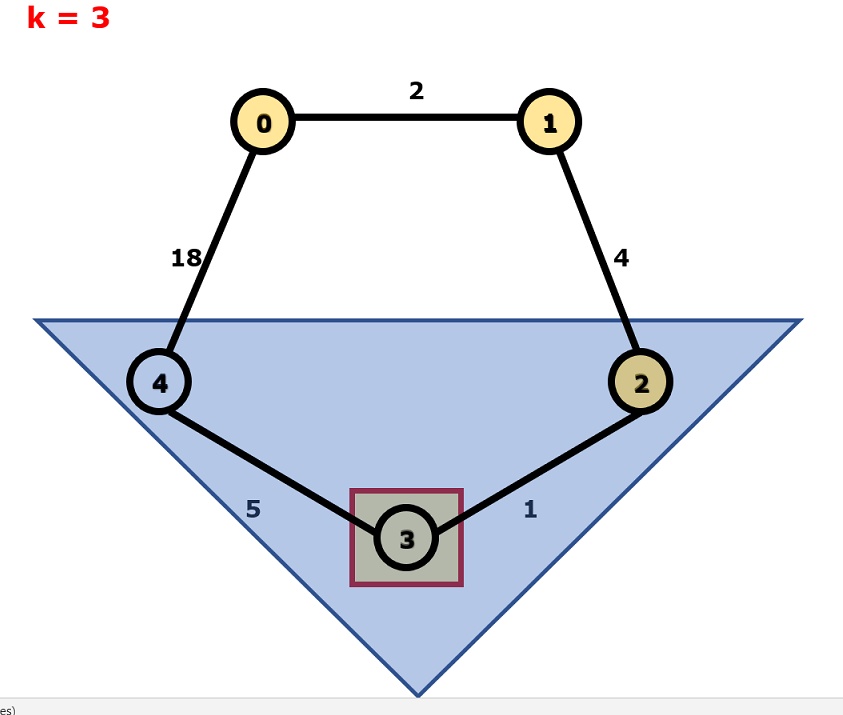


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | true | true | true |
| 1 | true | true | true | true | true |
| 2 | true | true | true | true | true |
| 3 | true | true | true | true | true |
| 4 | true | true | true | true | true |

**K = 2**

מטריצת מסלולים

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** |
| **0** | **0🡪0** | **0🡪1** | **0🡪1🡪2** | **0🡪1🡪2🡪3** | **0🡪4** |
| **1** | **1🡪0** | **1🡪1** | **1🡪2** | **1🡪2🡪3** | **1🡪0🡪4** |
| **2** | **2🡪1🡪0** | **2🡪1** | **2🡪2** | **2🡪3** | **2🡪1🡪0🡪4** |
| **3** | **3🡪2🡪1🡪0** | **3🡪2🡪1** | **3🡪2** | **3🡪3** | **3🡪4** |
| **4** | **4🡪0** | **4🡪0🡪1** | **4🡪0🡪1🡪2** | **4🡪3** | **4🡪4** |

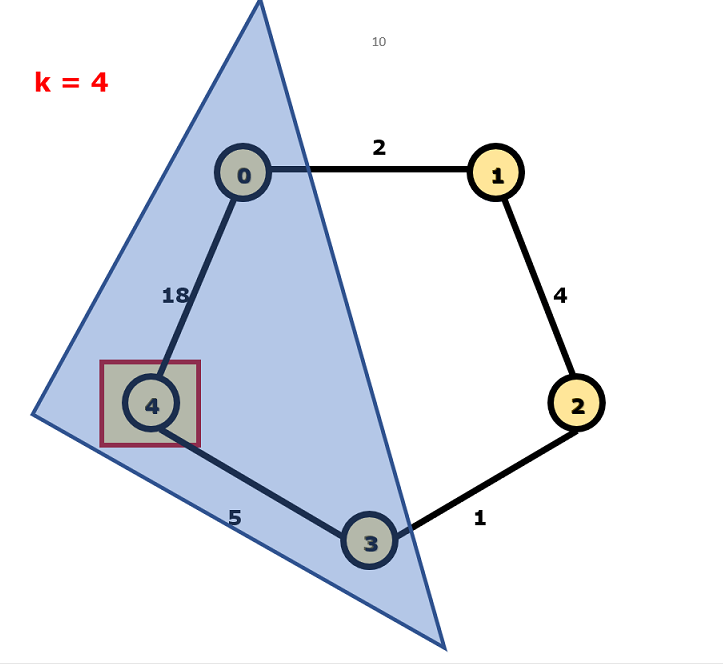


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | true | true | true |
| 1 | true | true | true | true | true |
| 2 | true | true | true | true | true |
| 3 | true | true | true | true | true |
| 4 | true | true | true | true | true |

**K = 3**

מטריצת מסלולים

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** |
| **0** | **0🡪0** | **0🡪1** | **0🡪1🡪2** | **0🡪1🡪2🡪3** | **0🡪1🡪2🡪3🡪4** |
| **1** | **1🡪0** | **1🡪1** | **1🡪2** | **1🡪2🡪3** | **1🡪2🡪3🡪4** |
| **2** | **2🡪1🡪0** | **2🡪1** | **2🡪2** | **2🡪3** | **2🡪3🡪4** |
| **3** | **3🡪2🡪1🡪0** | **3🡪2🡪1** | **3🡪2** | **3🡪3** | **3🡪4** |
| **4** | **4🡪3🡪2🡪1🡪0** | **4🡪3🡪2🡪1** | **4🡪3🡪2** | **4🡪3** | **4🡪4** |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | true | true | true | true | true |
| 1 | true | true | true | true | true |
| 2 | true | true | true | true | true |
| 3 | true | true | true | true | true |
| 4 | true | true | true | true | true |

**מטריצת קשירות**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** |
| **0** | **0🡪0** | **0🡪1** | **0🡪1🡪2** | **0🡪1🡪2🡪3** | **0🡪1🡪2🡪3🡪4** |
| **1** | **1🡪0** | **1🡪1** | **1🡪2** | **1🡪2🡪3** | **1🡪2🡪3🡪4** |
| **2** | **2🡪1🡪0** | **2🡪1** | **2🡪2** | **2🡪3** | **2🡪3🡪4** |
| **3** | **3🡪2🡪1🡪0** | **3🡪2🡪1** | **3🡪2** | **3🡪3** | **3🡪4** |
| **4** | **4🡪3🡪2🡪1🡪0** | **4🡪3🡪2🡪1** | **4🡪3🡪2** | **4🡪3** | **4🡪4** |

**מטריצת מסלולים**

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

**public** **static** **void** printPath(String [][]pathMat, **int**[][] mat){

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

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

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

**if** (mat[i][j] == *inf*) System.***out***.print("\*, ");

**else** System.***out***.printf("%25s",pathMat[i][j]+"; ");

}

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

}

}

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

**public** **static** String [][] buildPathMatrix(**boolean** [][] bm){

**int** n = mat.length;

// path matrix initialization

String [][] pathMat = **new** String[n][n];

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

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

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

pathMat[i][j] = " "+**i**+"**->**"+**j**+" ";

**else**

pathMat[i][j] = "";

}

}

// path matrix building

........

........

........

{

**משימה:**

1. לייצר מסלולים בין כל זוגות הצמתים בגרף לפי אלגוריתם פלויד־וורשל – להשלים את הפונקציה:

**public** **static** String [][] buildPathMatrix(**boolean** [][] bm)

**מספר רכיבי קשירות וקודקודי רכיבי קשירות**

נא לראות קובץ בשם **Problem 3\_Number of connected components.docx**

**משימה:**

1. ליצור פונקציה שמחשבת ומדפיסה אתמספר רכיבי הקשירות וגם מדפיסה את כל הקודקודים של כל אחד מרכיב הקשירות:

//Problem 3: Number connected component + List connected component

**public** **static** **void** connectComponentsOfGraphBoolean(**boolean** [][] mat)

**האם גרף קשיר**

נא לראות קובץ בשם **Problem 4\_Check the connectivity of the graph.docx**

**משימה:**

1. לבדוק האם גרף קשיר (בעל רכיב הקשירות אחד) –

* ליצור פונקציה שבודקת האם גרף קשיר בסיבוכיות

**public** **static** **boolean** isConnected(**boolean** mat[][])//O(n^2)

* ולאחר מכך לשפר את היעילות ל- - ליצור פונקציה נוספת שבודקת האם גרף קשיר בסיבוכיות

**public** **static** **boolean** isConnectedComplexN(**boolean** mat[][])//O(n)

**בעיית הבקבוקים**

**משימה:**

1. השלמת בעיית הבקבוקים – מציאת מסלולים
   * אתחול מטריצה: לעדכן את האלכסון ב-false
   * יצירת מטריצת מסלולים לפי אלגוריתם פלויד-וורשל
   * האם קיים מסלול ממצב אחד למצב אחר. אם כן, מהו המסלול

**בעיית השוקולד**

נא לראות קובץ בשם **Chocolate bar divide.docx**

**משימה:**

1. ליישם את הבעיה