**משקלים מוגדרים על קדקודי הגרף**

***fi –*** משקל של קדקוד ***i***

משימה: למצוא מסלול הזול ביותר מכל קודקוד לכל קודקוד אחר

f2 + f3

f3 + f4

f1 + f2

f4 + f5

הפתרון מבוסס על האלגוריתם של משקלים על הצלעות:

נמיר משקלי הקודקודים למשקלים על הצלעות ע"י פעולת החיבור של משקלי הקודקודים. כלומר, משקל על הצלע בין שני קודקודים יהיה שווה לסכום משקלים של שני קודקודים. לדוגמא, משקל על צלע בין קודוקים f1, f2 שווה ל- f1 + f2.

אך לפי שיטת ההמרה הזאת, חישוב עלות המסלול/המרחק בין קודקוד אחד לקודקוד אחר לא יתן תוצאה נכונה:

**e(i,j)** עלות המעבר בין קדקוד **i** לקדקוד **j** לפי צלעות (**edge**),   
 **v(i,j)** עלות המעבר בין קדקוד **i** לקדקוד **j** לפי הקדקודים (**vertex**).

1. **v(i,j)**=f**i**+ f**i+1** +. . .+ f**j-1** + f**j**
2. **e(i,j)**=f**i**+2(f**i+1** +. . .+f**j-1**) + f**j**

על מנת להשוות את הביטויים לעיל נעשה מספר הפעולות המתמטיות הבאות:

נכפיל את הביטוי (1) ב-2:

**2v(i,j)**=2f**i**+ 2f**i+1** +. . .+ 2f**j-1** + 2f**j**

-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**2v(i,j)**=2f**i**+ 2(f**i+1** +. . .+ f**j-1**)+ 2f**j**

-

**e(i,j)**=f**i**+2(f**i+1** +. . .+f**j-1**) + f**j**

=================================================================================================

**2v(i,j) - e(i,j)**= **fi + fj**

**e(i,j)**= **2v(i,j) - fi - fj**

**v(i,j) = (e(i,j) + fi + fj)/2**

שיטה חלופית של פעולות מתמטיות:

בביטוי (2) נכפיל את האיבר הראשון ואת האיבר האחרון ב-2 ולאחר מכך נחלק את הביטוי ב-2:

**v(i,j)**=(2f**i**+2(f**i+1** +. . .+f**j-1**)+2f**j**)/2 **=(e(i,j)+fi+fj)/2**

**תיאור האלגוריתם למציאת את כל המסלולים הקצרים ביותר בין קדקודי הגרף כאשר המשקלים מוגדרים על קדקודי הגרף:**

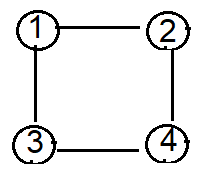
**קלט**: מערך של משקלים המוגדרים על קדקודי הגרף ומטריצה בוליאנית המגדירה את צלעות הגרף (מטריצת השכנויות).

**פלט**: מטריצה המייצגת את המרחקים הקצרים ביותר בין קדקודי הגרף.

**שלבי האלגוריתם:**

1. בונים מטריצה המייצגת את המשקלים על צלעות הגרף (ביצוע המרה של מטריצת השכניות למטריצה של משקלים על הצלעות)
2. מפעילים על המטריצה שקיבלנו בשלב 1 את האלגוריתם של פלויד-וורשל, מקבלים מטריצת העלויות הקטנות ביותר, כאשר המשקלים מוגדרים על צלעות הגרף.
3. ממירים במטריצה את העלויות המוגדות על הצלעות לעלויות המוגדרות על הקדקודים לפי הנוסחה לעיל – תיקון עלויות המסלולים.

**דוגמה:**

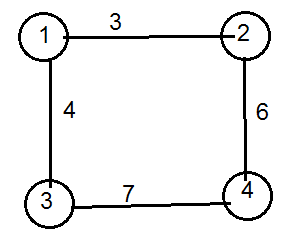
נתון גרף עם משקלים על הקדקודים:

***f1***=1, ***f2=2, f3=3, f4=4***

ונתונה מטריצה בוליאנית המייצגת את צלעות בגרף – מטריצת השכנויות:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***f4*** | ***f3*** | ***f2*** | ***f1*** |  |
| f | t | t | t | ***f1*** |
| t | f | t | t | ***f2*** |
| t | t | f | t | ***f3*** |
| t | t | t | f | ***f4*** |

1. נעבור למשקלי הצלעות:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***f4*** | ***f3*** | ***f2*** | ***f1*** |  |
| ∞ | 4 | 3 | 0 | ***f1*** |
| 6 | ∞ | 0 | 3 | ***f2*** |
| 7 | 0 | ∞ | 4 | ***f3*** |
| 0 | 7 | 6 | ∞ | ***f4*** |

weights of edges

0, 3, 4, \*,

3, 0, \*, 6,

4, \*, 0, 7,

\*, 6, 7, 0,

1. נפעיל אלגוריתם של Floyd-Warshall על מטריצת משקלי הצלעות:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***f4*** | ***f3*** | ***f2*** | ***f1*** |  |
| 9 | 4 | 3 | 0 | ***f1*** |
| 6 | 7 | 0 | 3 | ***f2*** |
| 7 | 0 | 7 | 4 | ***f3*** |
| 0 | 7 | 6 | 9 | ***f4*** |

matrix of prices

0, 3, 4, 9,

3, 0, 7, 6,

4, 7, 0, 7,

9, 6, 7, 0,

1. נחזור לעלויות לפי הקדקודים:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***f4*** | ***f3*** | ***f2*** | ***f1*** |  |
| 7 | 4 | 3 | 0 | ***f1*** |
| 6 | 6 | 0 | 3 | ***f2*** |
| 7 | 0 | 6 | 4 | ***f3*** |
| 0 | 7 | 6 | 7 | ***f4*** |

matrix of vertices weights

0, 3, 4, 7,

3, 0, 6, 6,

4, 6, 0, 7,

7, 6, 7, 0,

**משימה:**

ליישם את האלגוריתם – להשלים פונקציות להלן:

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

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

**public** **static** **int**[][] *vertexWeightArray2edgeWeightMatrix* (**int**[] vertexWeight, **boolean** mb[][]){

// building matrix of edge weights

**int** n = mb.length;

**int** matWEdges[][] = **new** **int**[mb.length][mb.length];

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

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

**if** (i != j)

matWEdges[i][j] = *infinity*;

**if** (mb[i][j] && i != j)

matWEdges[i][j] = vertexWeight[i] + vertexWeight[j];

}

}

**return** matWEdges;

}

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

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

**int** n = mat.length;

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

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

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

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

**if** (mat[i][j] > mat[i][k]+mat[k][j]){

mat[i][j] = mat[i][k]+mat[k][j];

}

}

}

}

}

}

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

**public** **static** **int**[][] *edgeWeightMatrix2VertexWeightMatix* (**int**[] vertexWeight, **int** matWEdges[][]){

**int** n = matWEdges.length;

**int**[][] matWVert = **new** **int**[n][n];

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

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

**.....**

**.....**

}

}

**return** matWVert;

}

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

**public** **static** **int**[][] verToEdgeWeights(**int**[] vertexWeight, **int** matWVert[][]){

**int** n = matWVert.length;

**int** [][] matWEdges = **new** **int**[n][n];

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

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

**.....**

**.....**

}

}

**return** matWEdges;

}

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

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

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

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

**if** (i==j) System.***out***.print("0, ");

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

**else** System.***out***.print(mat[i][j]+", ");

}

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

}

}

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

**public** **static** **boolean**[][] initB1(){

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

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

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

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

**return** mat;

}

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

**public** **static** **boolean**[][] initB2(){

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

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

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

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

**return** mat;

}

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

**public** **static** **void** main(String[] args) {

**boolean** mb[][] = *initB1*();

**int** [] weVert = {1,2,3,4};

//\*\*\*\*\* Step 1 \*\*\*\*\*

**int**[][] matWEdges = *vertexWeightArray2edgeWeightMatrix* (weVert, mb);

System.***out***.println("\nweights of edges");

*printMatrix*(matWEdges);

//\*\*\*\*\* Step 2 \*\*\*\*\*

*FWAlgorithm*(matWEdges);

System.***out***.println("matrix of prices");

*printMatrix*(matWEdges);

//\*\*\*\*\* Step 3 \*\*\*\*\*

System.***out***.println("matrix of vertices weights");

**int**[][] matWeightVert = *edgeWeightMatrix2VertexWeightMatix (weVert, matWEdges);*

*printMatrix*(matWeightVert);

//\*\*\*\*\* Step 4 \*\*\*\*\*

System.***out***.println("matrix of edges weights");

**int**[][] matWeightEdge = *vertexWeightMatrix2edgeWeightMatrix* (weVert, matWeightVert);

*printMatrix*(matWeightEdge);

}

}

**Output**

weights of edges

0, 3, 4, \*,

3, 0, \*, 6,

4, \*, 0, 7,

\*, 6, 7, 0,

matrix of prices

0, 3, 4, 9,

3, 0, 7, 6,

4, 7, 0, 7,

9, 6, 7, 0,

matrix of vertices weights

0, 3, 4, 7,

3, 0, 6, 6,

4, 6, 0, 7,

7, 6, 7, 0,

matrix of edges weights

0, 3, 4, 9,

3, 0, 7, 6,

4, 7, 0, 7,

9, 6, 7, 0,