Aeroplane algorithm

Node class

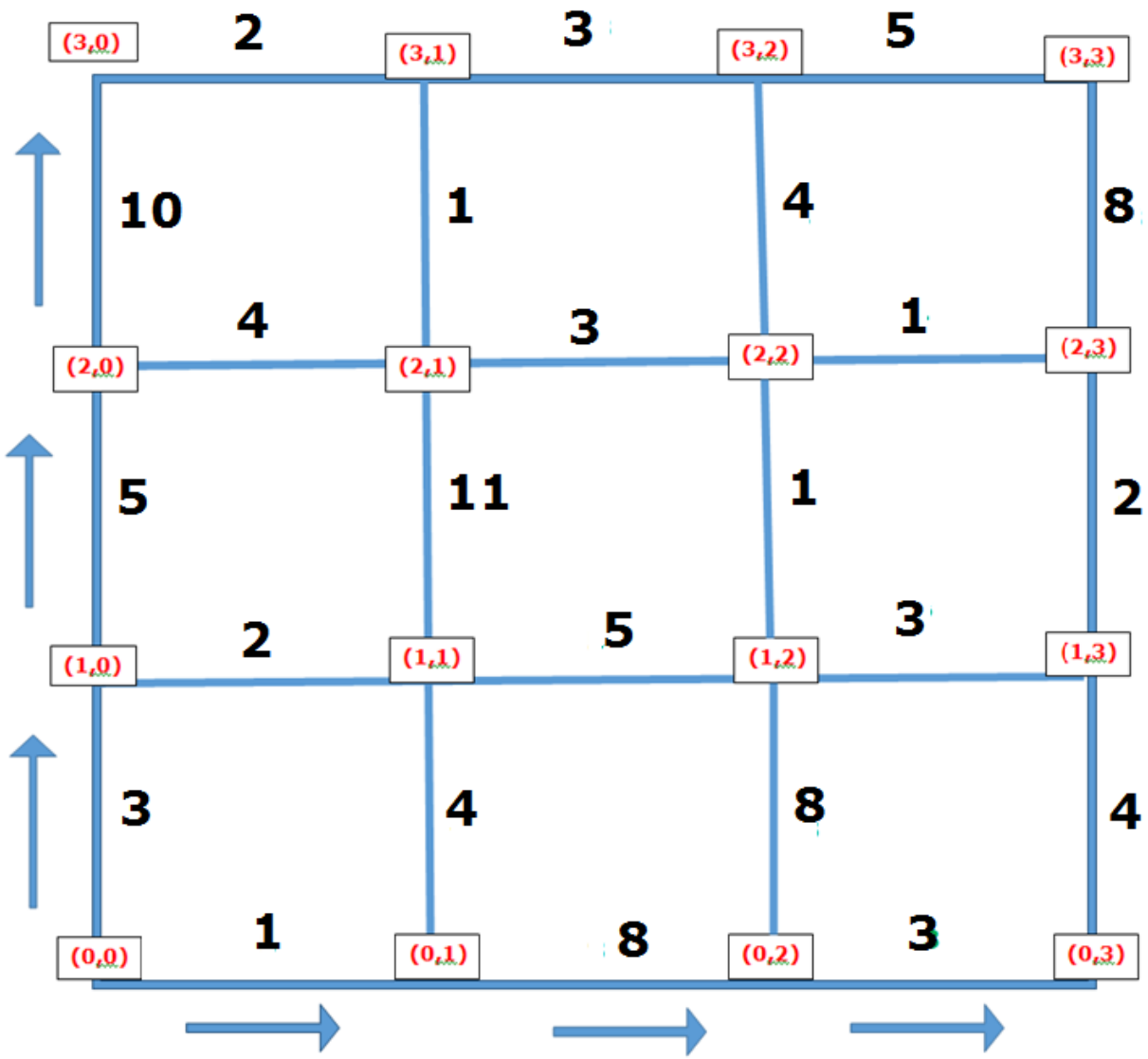
```
public class Node {
    int     x, y;
    int     price;
    int     numOfPaths;

    public Node(int x, int y){
        this.x = x;
        this.y = y;
        this.price = 0;
        this.numOfPaths = 0;
    }

    public String toString(){
        return   "x="+x+", y="+y+", price="+price+
                 ", np="+numOfPaths+"; ";
    }
}
```

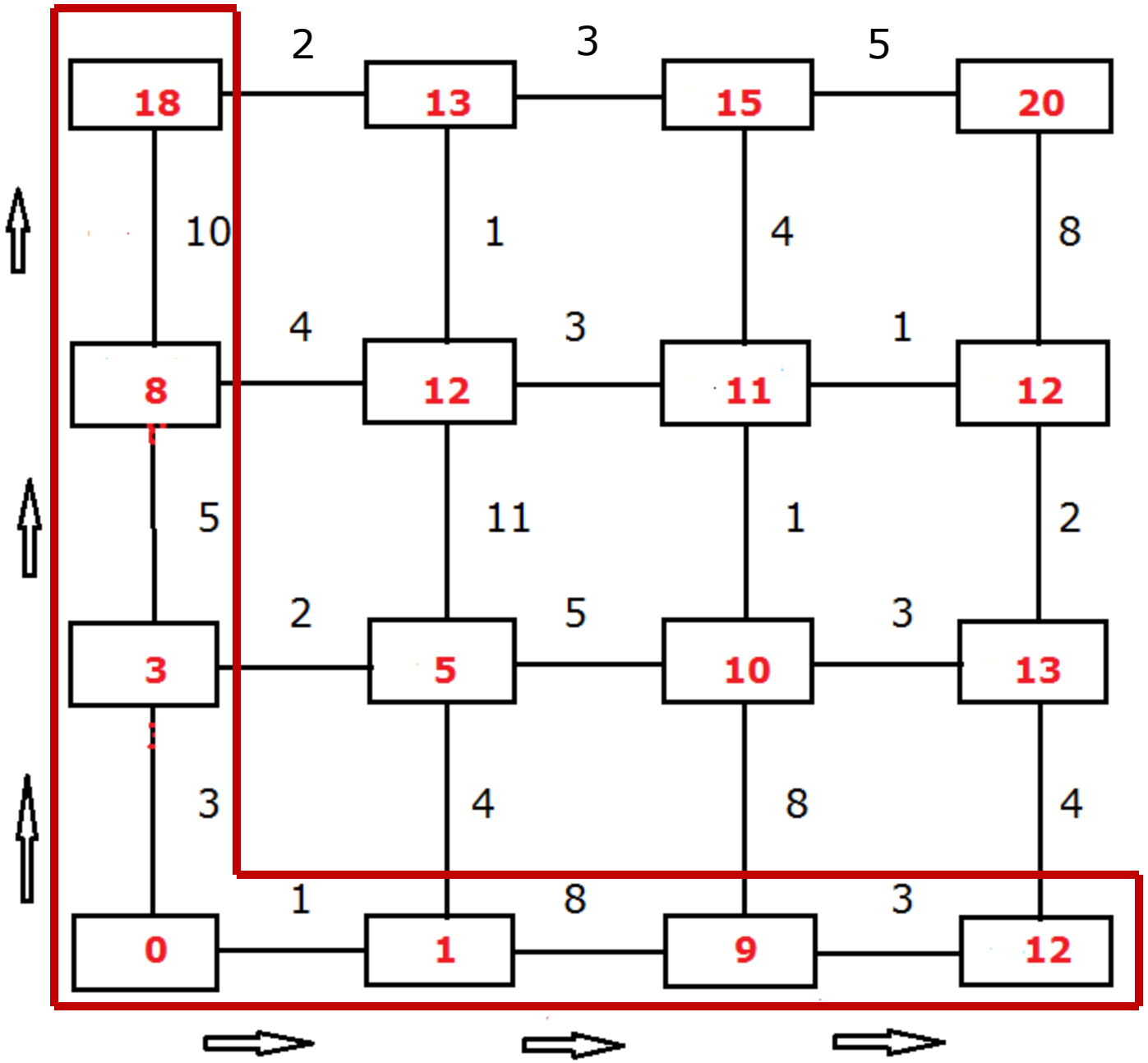
InitMatrixOfPrices class

```
public class InitMatrixOfPrices {
    public static Node[][] initMatOfNodes(){ // n = 4
        int n=4;
        Node mat[][] = new Node[n][n];
        // the 1-st row
        mat[0][0] = new Node(1,3);
        mat[0][1] = new Node(8,4);
        mat[0][2] = new Node(3,8);
        mat[0][3] = new Node(0,4);
        // the 2-nd row
        mat[1][0] = new Node(2,5);
        mat[1][1] = new Node(5,11);
        mat[1][2] = new Node(3,1);
        mat[1][3] = new Node(0,2);
        // the 3-d row
        mat[2][0] = new Node(4,10);
        mat[2][1] = new Node(3,1);
        mat[2][2] = new Node(1,4);
        mat[2][3] = new Node(0,8);
        // the 4-th row
        mat[3][0] = new Node(2,0);
        mat[3][1] = new Node(3,0);
        mat[3][2] = new Node(5,0);
        mat[3][3] = new Node(0,0);
        return mat;
    }
}
```




```
mat[0][0] = new Node(1,3);
mat[0][1] = new Node(8,4);
mat[0][2] = new Node(3,8);
mat[0][3] = new Node(0,4);
```

```
x[0,0] = 1    y[0][0] = 3
x[0,1] = 8    y[0][1] = 4
x[0,2] = 3    y[0][2] = 8
x[0,3] = 0    y[0][3] = 4
```


2
3
5
18
13
15
20
10
1
4
8
4
3
1
8
12
11
12
5
11
1
2
2
5
3
3
5
10
13
3
4
8
4
1
8
3
0
1
9
12

Step 2

Step 1

Price : Step 1

row 0

price (0,0) = 0

price (0,1) = price (0,0) + x(0,0) = 0 + 1 = 1

price (0,2) = price (0,1) + x(0,1) = 1 + 8 = 9

price (0,3) = price (0,2) + x(0,2) = 9 + 3 = 12

Formula : price (0, j) = price (0, j-1) + x(0, j-1)

column 0

price (0,0) = 0

price (1,0) = price (0,0) + y(0,0) = 0 + 3 = 4

price (2,0) = price (1,0) + y(1,0) = 3 + 5 = 8

price (3,0) = price (2,0) + y(2,0) = 8 + 10 = 18

Formula : price (i, 0) = price (i-1, 0) + y(i-1, 0)

Price : Step 2

row 1

price (1,1) = **min** (price (1,0) + x(1,0), price (0,1) + y(0,1))

price (1,0) + x(1,0) = 3 + 2 = **5**

price (0,1) + y(0,1) = 1 + 4 = **5**

---

price (1,2) = **min** (price (1,1) + x(1,1), price (0,2) + y(0,2))

price (1,1) + x(1,1) = 5 + 5 = **10**

price (0,2) + y(0,2) = 9 + 8 = 17

---

price (1,3) = **min** (price (1,2) + x(1,2), price (0,3) + y(0,3))

price (1,2) + x(1,2) = 10 + 3 = **13**

price (0,3) + y(0,3) = 12 + 4 = 16

==========

row 2

price (2,1) = **min** (price (2,0) + x(2,0), price (1,1) + y(1,1))

price (2,0) + y(2,0) = 8 + 4 = **12**

price (1,1) + x(1,1) = 5 + 11 = 16

---

price (2,2) = **min** (price (2,1) + x(2,1), price (1,2) + y(1,2))

price (2,1) + x(2,1) = 12 + 3 = 15

price (1,2) + y(1,2) = 10 + 1 = **11**

---

price (2,3) = **min** (price (2,2) + x(2,2), price (1,3) + y(1,3))

price (2,2) + x(2,2) = 11 + 1 = **12**

price (1,3) + y(1,3) = 13 + 2 = 15

==========

row 3

price (3,1) = **min** (price (3,0) + x(3,0), price (2,1) + y(2,1))

price (3,0) + x(3,0) = 18 + 2 = 20

price (2,1) + y(2,1) = 12 + 1 = **13**

---

price (3,2) = **min** (price (3,1) + x(3,1), price (2,2) + y(2,2))

price (3,1) + x(3,1) = 13 + 3 = 16

price (2,2) + y(2,2) = 11 + 4 = **15**

---

price (3,3) = **min** (price (3,2) + x(3,2), price (2,3) + y(2,3))

price (3,2) + x(3,2) = 15 + 5 = **20**

price (2,3) + y(2,3) = 12 + 8 = **20**

==========

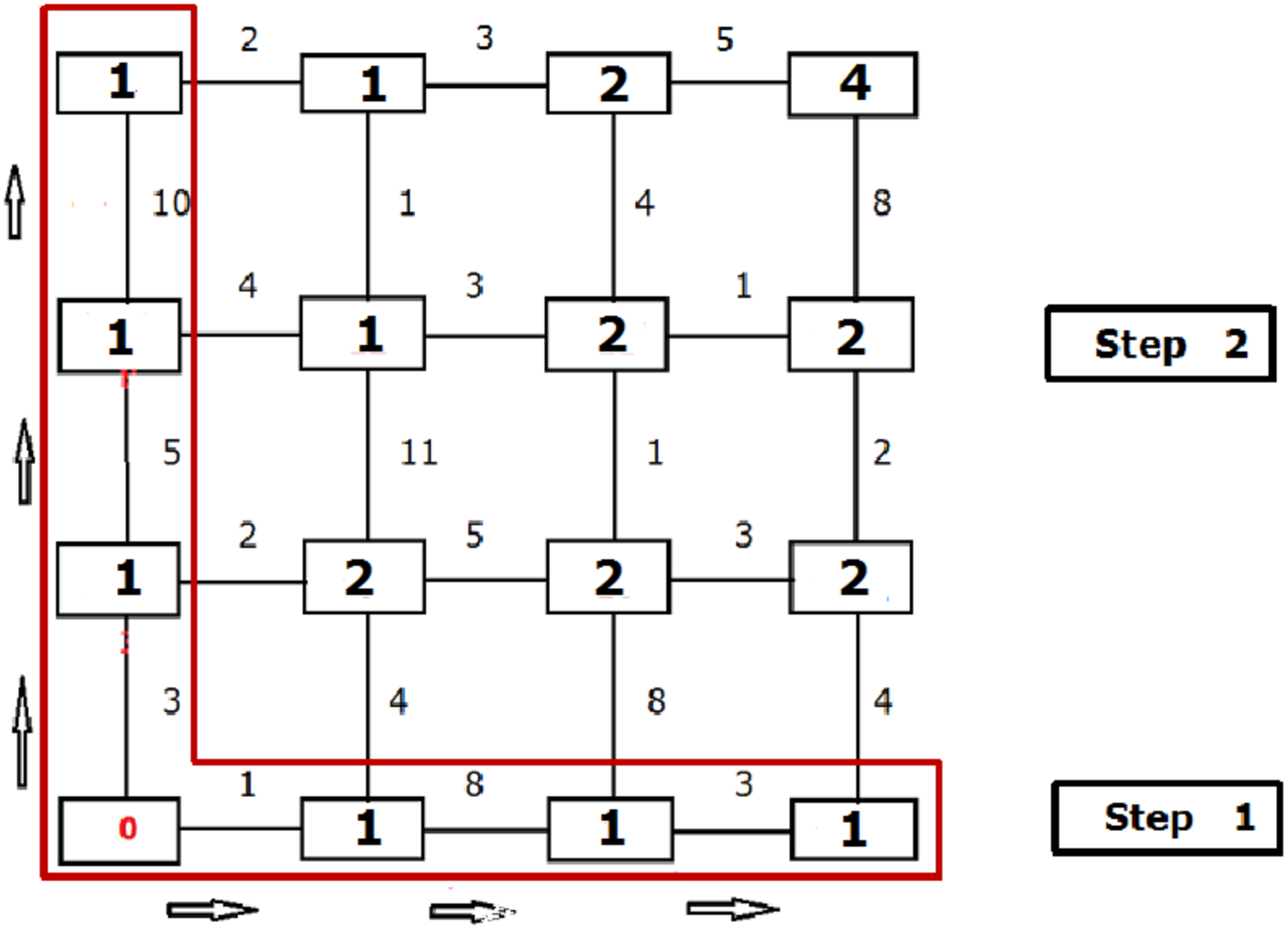
Formula :

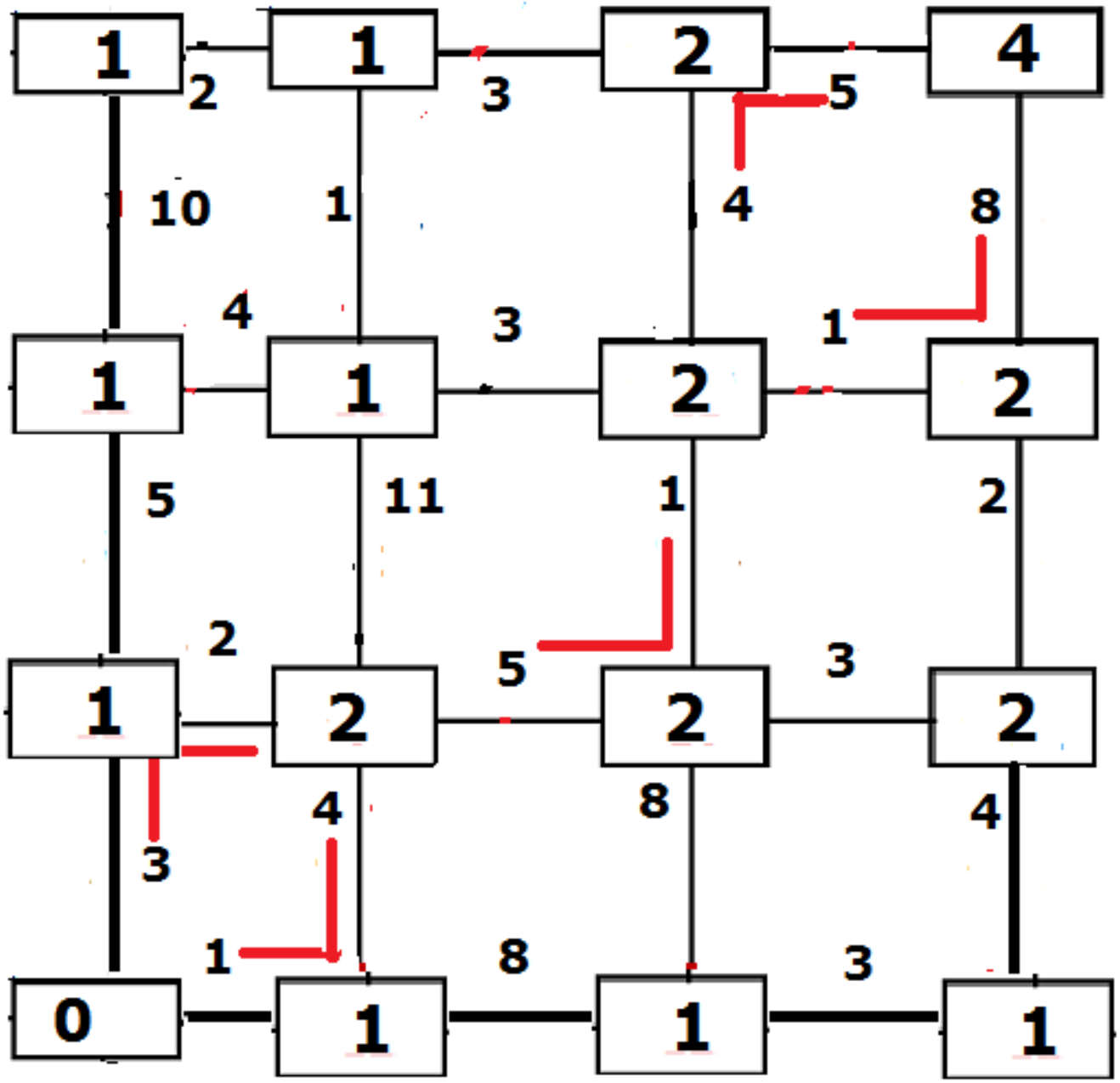
price (i,j) = min (price (i,j-1) + x(I,j-1), price (i-1,j) + y(i-1,j))

Result mat[][].price

18	13	15	20
8	12	11	12
3	5	10	13
0	1	9	12

Matrix number of paths




1
1
2
4
2
3
5
10
1
4
8
4
3
1
1
1
2
2
5
11
1
2
2
5
3
1
2
2
2
3
4
8
4
1
8
3
0
1
1
1

```
1. public void getBestPrice()      // n rows, m columns
2. public void getBestPath()       // n rows, m columns
3. public String getOneCheapestPath()
4. public  void AllPathsRecurs()
5. public void buildPaths(String path, int i, int j,
                                   ArrayList<String> paths)
```

public void getBestPrice() **Pseudocode**

```
N – number of rows
M – number of columns
mat[0][0].price = 0

loop i from 1 to N-1 (including) step 1
    mat[i][0] = mat[i-1][0].y + mat[i-1][0].price

loop j from 1 to M-1 (including) step 1
    mat[0][j] = mat[0][j-1].y + mat[0][j-1].price

loop i from 1 to N-1 (including) step 1
    loop j from 1 to M-1 (including) step 1
        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
        else if (a > b)         mat[i][j] ].price = b
        else                        mat[i][j] ].price = a //x==y
cheapestPrice = mat[n-1][m-1].price
```

public void getBestPath() **Pseudocode**

```
N – number of rows
M – number of columns
mat[0][0].price = 0

loop i from 1 to N-1 (including) step 1
    mat[i][0] = mat[i-1][0].y + mat[i-1][0].price
    mat[i][0].numOfPaths = 1

loop j from 1 to M-1 (including) step 1
    mat[0][j] = mat[0][j-1].y + mat[0][j-1].price
    mat[0][j].numOfPaths = 1


loop i from 1 to N-1 (including) step 1
  loop j from 1 to M-1 (including) step 1
    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] = mat[i-1][j].numOfPaths
```

```
        else if (a > b)
                mat[i][j] = mat[i][j-1].numOfPaths
        else
                mat[i][j] = mat[i][j-1].numOfPaths +
                            mat[i-1][j].numOfPaths

numOfPaths = mat[n-1][m-1].numOfPaths
```

public String getOneCheapestPath()

Pseudocode

```
i – number of rows - 1
j – number of columns - 1
String ans = ""

loop (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 = "1" + ans
                i = i – 1
        else            //a>b
                ans = "0" + ans
                j = j - 1
if (i == 0)
    loop (j > 0)
        ans = "0" + ans
        j = j - 1
else
    loop (i > 0)
```

```
        ans = "1" + ans
        i = i – 1

return ans
```

public void AllPathsRecurs() **Pseudocode**

```
ArrayList<String> paths =
                    new  ArrayList<String>(numOfPaths)
buildPaths(new String(), mat.length-1,
                              mat[0].length-1, paths)
System.out.println(paths)
```

```
public void buildPaths(String path, int i, int j,
                       ArrayList<String> paths)
    if (i > 0 && j > 0)
        a = mat[i-1][j]. numOfPaths + mat[i-1][j]. y
        b = mat[i][j-1]. numOfPaths + mat[i][j-1].x

        if (a < b)
            buildPaths("1"+path, i-1, j, paths)
        else if (a>b)
            buildPaths("0"+path, i, j-1, paths)
        else  //a==b
            buildPaths("1"+path, i-1, j, paths)
            buildPaths("0" + new String(path), i, j-1, paths)
    else if (i == 0 && j == 0)
        paths.add(path)
    else if (i==0)
        String t = new String()
        for(int k=0; k<j; k++) t = t +  "0"
        paths.add(t + path)
```

```
else if (j==0)
    String t = new String()
    for(int k=0; k<i; k++) t = t +  "1"
    paths.add(t + path)
```