

Transportation Problem Solver APP



Introduction:

The goal of this application is to solve the transportation problem using the North-West algorithm by using KOTLIN and XML for development

Developed by:

- BENGUELLA MOHAMMED ELAMINE TAHAR
- GHOURBAL AYOUB

North-West algorithm:

```
private fun solveTransportationProblem(supply: IntArray, demand: IntArray): Array<IntArray> {  
    val m = supply.size  
    val n = demand.size  
    val allocations = Array(m) { IntArray(n) }  
  
    var i = 0  
    var j = 0  
  
    while (i < m && j < n) {  
        val quantity = minOf(supply[i], demand[j])  
        allocations[i][j] = quantity  
        supply[i] -= quantity  
        demand[j] -= quantity  
  
        if (supply[i] == 0) {  
            i++  
        }  
  
        if (demand[j] == 0) {  
            j++  
        }  
    }  
  
    return allocations  
}
```

Calculate total cost:

```
private fun calculateTotalCost(allocations: Array<IntArray>, costs: IntArray): Int {  
    var totalCost = 0  
    var k = 0  
  
    for (i in allocations.indices) {  
        for (j in 0 until allocations[i].size) {  
            totalCost += allocations[i][j] * costs[k]  
            k++  
        }  
    }  
  
    return totalCost  
}
```

UI and the functionality of the app:

The first screenshot shows the 'Insert values' screen with two empty input fields: 'N of suppliers >=2' and 'N of consumers >=2'. A blue 'NEXT' button is at the bottom. The second screenshot shows the same screen after user input: 'N of suppliers >=2' contains the value '3' and 'N of consumers >=2' contains the value '4'. The 'NEXT' button remains at the bottom.

Here the user insert the number of suppliers and consumers (in this example it's suppliers=3 and consumers=4), then after clicking NEXT it shows:

The first screenshot shows the 'Insert values' screen with seven input fields: three for 'Supplier 1', 'Supplier 2', and 'Supplier 3'; and four for 'Consumer 1', 'Consumer 2', 'Consumer 3', and 'Consumer 4'. A blue 'NEXT' button is at the bottom. The second screenshot shows the same screen after user input: the three supplier fields contain the values 500, 300, and 200; the four consumer fields contain the values 300, 300, 300, and 100. The 'NEXT' button remains at the bottom.

Here the app displays another input to insert the values of each supplier and consumer (in this example it's 500 300 200 for suppliers and 300 300 300 100 for consumers), then after clicking NEXT it shows:

1:53

Insert transactions

Transaction 14

Transaction 21

Transaction 22

Transaction 23

Transaction 24

Transaction 31

Transaction 32

Transaction 33

Transaction 34

CALCULATE

1:53

Insert transactions

1

10

8

4

2

7

9

11

12

CALCULATE

Here the app displays another input to insert the transactions of each supplier and consumer (in this example it's 6 5 3 1 10 8 4 2 7 9 11 12), then after clicking **CALCULATE** it shows the result:

1:53

Supplier 1	300 units	200 units	0 units	0 units
Supplier 2	0 units	100 units	200 units	0 units
Supplier 3	0 units	0 units	100 units	100 units

Total cost = **6700**

How back-end work (by using KOTLIN):

The application creates dynamically all inputs depending on the number of suppliers and consumers entered firstly.

The application transfers the entered data via SHAREDREFERENCES (named by CASHE), that's why if you click the back button of the phone you can still modify the entered values and get the new result.