ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ПМиК

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«Нахождение начального опорного плана транспортной задачи»

по дисциплине «Алгоритмы и вычислительные методы оптимизации»

Выполнил: студент гр. ИП-814
Краснов И.В.
Проверил: ассистент кафедры ПМиК
Новожилов Д. И.

Новосибирск 2021

Содержание

Задание

Написать программу, находящую начальный опорный план транспортной задачи одним из указанных методов (номер метода находится как n mod 3, где n – номер бригады).

0. Метод северо-западного угла.

1. Метод минимальной стоимости.

2. Метод Фогеля

Программа должна работать как с открытой, так и закрытой моделью транспортной задачи. Предусмотреть программное нахождение вырожденного плана. Вывести распределение перевозок и затраты. Для тестирования использовать несколько заданий из практических занятий.

Текст программы

```
import math


class Trans:

    cost = 0
    weight = 0

    def __init__(self):
        self.cost = 0
        self.weight = math.inf

    def __str__(self):
        if (self.weight != -1) and (self.weight != math.inf):
            return '{}:{}'.format(self.weight, self.cost)
        elif self.weight == -1:
            return '-:{}'.format(self.cost)
        else:
            return '{}'.format(self.cost)


def PrintMatrix(matrix, stock, need, N, M):

    for i in range(N):
        for j in range(M):
            print("|%5s" % matrix[i][j], end='')
            if j == M - 1:
                print("|%5s" % stock[i], end='')

        print()
        if i == N - 1:
```

```
        for j in range(M):
            print("|%5s" % need[j], end='')
        print()


def StartPlanMin(matrix, stock, need, N, M):

    print("\nStart making plan")
    sumNeed = 0
    sumStock = 0
    count = 1
    zeros = 0
    while (True):

        sumNeed = 0
        sumStock = 0
        for i in range(N):
            sumStock += stock[i]
        for j in range(M):
            sumNeed += need[i]

        if sumStock == 0 and sumNeed == 0:
            sum = 0
            for i in range(N):
                for j in range(M):
                    if matrix[i][j].weight > 0:
                        sum = sum + matrix[i][j].cost * matrix[i][j].weight

            print("\n", "Sum = ", sum)
            break

        else:
```

```
print("\nStep ", count)


min = math.inf
min_i = -1
min_j = -1

for i in range(N):
    for j in range(M):
        if matrix[i][j].cost < min and matrix[i][j].weight == math.inf and matrix[i][j].cost != 0:
            min = matrix[i][j].cost
            min_i = i
            min_j = j

if min == math.inf:
    for i in range(N):
        for j in range(M):
            if matrix[i][j].cost < min and matrix[i][j].weight == math.inf:
                min = matrix[i][j].cost
                min_i = i
                min_j = j

    if min == math.inf:
        print("Start to panic")
        break

if stock[min_i] > need[min_j]:
    matrix[min_i][min_j].weight = need[min_j]
    stock[min_i] -= need[min_j]
    need[min_j] = 0

    for i in range(N):
        if matrix[i][min_j].weight == math.inf:
```

```
            matrix[i][min_j].weight = -1

    elif stock[min_i] < need[min_j]:
        matrix[min_i][min_j].weight = stock[min_i]
        need[min_j] -= stock[min_i]
        stock[min_i] = 0

        for i in range(M):
            if matrix[min_i][i].weight == math.inf:
                matrix[min_i][i].weight = -1

    elif stock[min_i] == need[min_j]:
        matrix[min_i][min_j].weight = stock[min_i]
        stock[min_i] = 0
        need[min_j] = 0

        kof = True



        if count != int(M + N - 1 - zeros):
            for i in range(min_j, N):
                if matrix[i][min_j].weight == -1:
                    matrix[i][min_j].weight = 0
                    kof = False
                    zeros += 1
                    break

        if kof and count != int(M + N - 1 - zeros):
            for i in range(min_i, M):
                if matrix[min_i][i].weight != -1 and i != min_j:
                    matrix[min_i][i].weight = 0
                    kof = False
```

```
                zeros += 1
                break

        for i in range(N):
            if matrix[i][min_j].weight == math.inf:
                matrix[i][min_j].weight = -1

        for i in range(M):
            if matrix[min_i][i].weight == math.inf:
                matrix[min_i][i].weight = -1


        PrintMatrix(matrix, stock, need, N, M)
        count += 1


if __name__ == '__main__':

    matrix = []
    stock = []
    need = []
    sumStock = 0
    sumNeed = 0

    print("Input number of providers: ")
    N = int(input())
    print("Input number of consumers: ")
    M = int(input())

    for i in range(N):
        matrix.append([])
        for j in range(M):
```

```
        matrix[i].append(Trans())

for i in range(N):
    stock.append(int(0))

for i in range(M):
    need.append(int(0))

print("Input your matrix")

for i in range(N):
    for j in range(M):
        matrix[i][j].cost = int(input())

print("Input stocks: ")
for i in range(N):
    print("\t", i + 1, " provider: ")
    stock[i] = int(input())

print("Input needs: ")
for i in range(M):
    print("\t", i + 1, " consumer: ")
    need[i] = int(input())

PrintMatrix(matrix, stock, need, N, M)

for i in range(N):
    sumStock += stock[i]

for i in range(M):
    sumNeed += need[i]
```

```
if sumStock == sumNeed:
    print("System is closed")
else:
    print("System is open")
    if sumStock > sumNeed:
        for i in range(N):
            matrix[i].append(Trans())
        M += 1
        need.append(sumStock - sumNeed)
        sumNeed = sumStock

    elif sumNeed > sumStock:
        matrix.append([])
        for i in range(M):
            matrix[N].append(Trans())
        N += 1
        stock.append(sumNeed - sumStock)
        sumStock = sumNeed

    PrintMatrix(matrix, stock, need, N, M)

StartPlanMin(matrix, stock, need, N, M)
```

Результат работы программы

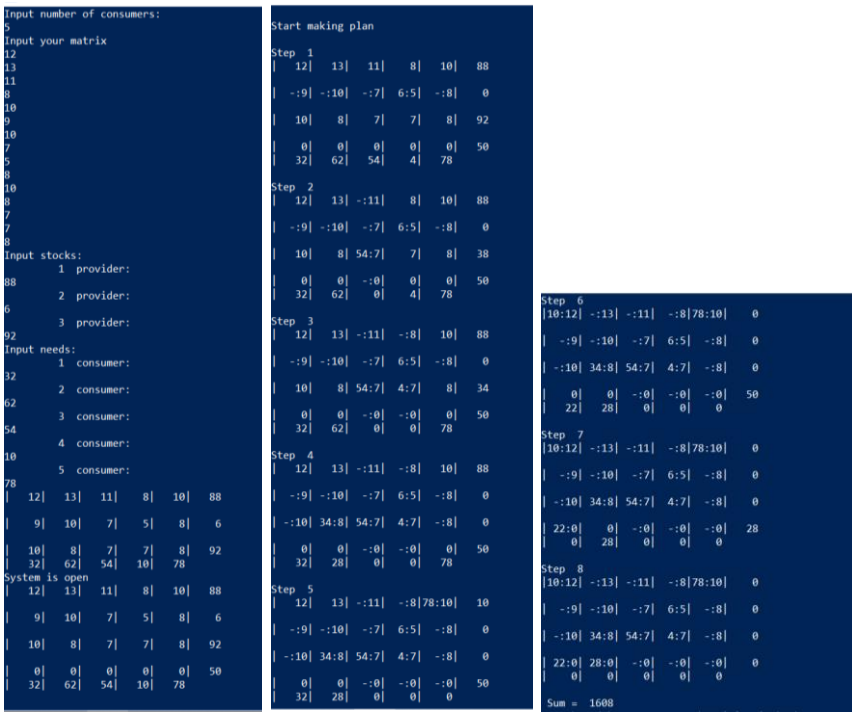

Рис. 1,2,3 Система из практического занятия

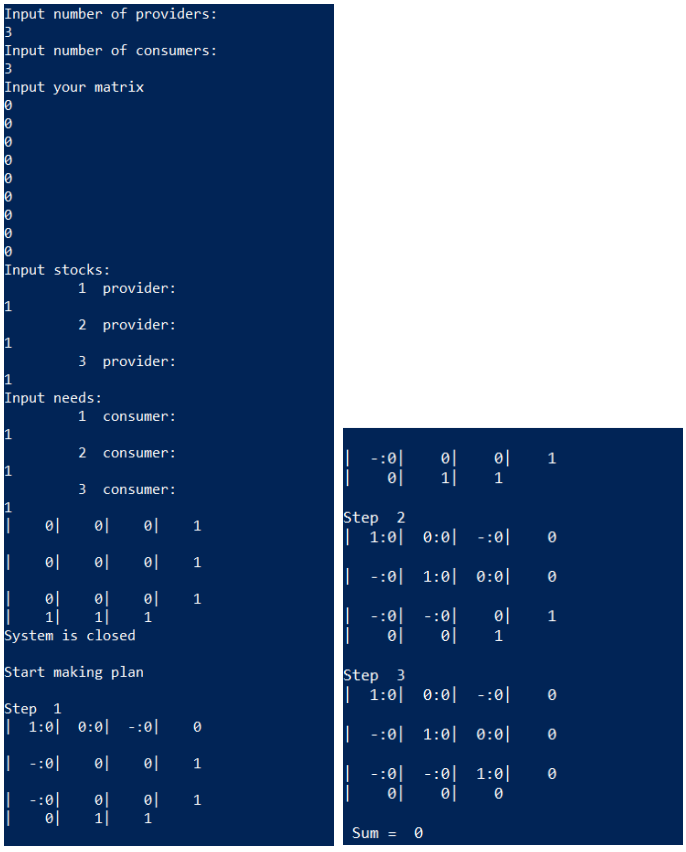

Рис. 4 Система, демонстрирующая добавление нулей