

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	3
Текст программы	4
Результат работы программы.....	11

Задание

Написать программу, находящую начальный опорный план транспортной задачи одним из указанных методов (номер метода находится как $n \bmod 3$, 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

```

Input number of consumers:
5
Input your matrix
12
13
11
9
10
9
10
7
5
8
10
8
7
7
0
Input stocks:
88 1 provider:
6 2 provider:
92 3 provider:
Input needs:
32 1 consumer:
62 2 consumer:
54 3 consumer:
10 4 consumer:
78 5 consumer:
12| 13| 11| 8| 10| 88
| 9| 10| 7| 5| 8| 6
| 10| 8| 7| 7| 8| 92
System is open
12| 13| 11| 8| 10| 88
| 9| 10| 7| 5| 8| 6
| 10| 8| 7| 7| 8| 92
| 0| 0| 0| 0| 0| 50
| 32| 62| 54| 10| 78| 0

Start making plan
Step 1
| 12| 13| 11| 8| 10| 88
| -19| -10| -7| 6:5| -18| 0
| 10| 8| 7| 7| 8| 92
| 0| 0| 0| 0| 0| 50
| 32| 62| 54| 4| 78| 0
Step 2
| 12| 13| -11| 8| 10| 88
| -19| -10| -7| 6:5| -18| 0
| 10| 8| 54:7| 7| 8| 38
| 0| 0| -0| 0| 0| 50
| 32| 62| 0| 4| 78| 0
Step 3
| 12| 13| -11| -0| 10| 88
| -19| -10| -7| 6:5| -18| 0
| 10| 8| 54:7| 4:7| 8| 34
| 0| 0| -0| -0| 0| 50
| 32| 62| 0| 0| 78| 0
Step 4
| 12| 13| -11| -0| 10| 88
| -19| -10| -7| 6:5| -18| 0
| -10| 34:8| 54:7| 4:7| -18| 0
| 0| 0| -0| -0| 0| 50
| 32| 28| 0| 0| 78| 0
Step 5
| 12| 13| -11| -0| 78:10| 10
| -19| -10| -7| 6:5| -18| 0
| -10| 34:8| 54:7| 4:7| -18| 0
| 0| 0| -0| -0| -0| 50
| 32| 28| 0| 0| 0| 0
Step 6
| 10:12| -13| -11| -0| 78:10| 0
| -19| -10| -7| 6:5| -18| 0
| -10| 34:8| 54:7| 4:7| -18| 0
| 0| 0| -0| -0| -0| 50
| 22| 28| 0| 0| 0| 0
Step 7
| 10:12| -13| -11| -0| 78:10| 0
| -19| -10| -7| 6:5| -18| 0
| -10| 34:8| 54:7| 4:7| -18| 0
| 22:0| 0| -0| -0| -0| 28
| 0| 28| 0| 0| 0| 0
Step 8
| 10:12| -13| -11| -0| 78:10| 0
| -19| -10| -7| 6:5| -18| 0
| -10| 34:8| 54:7| 4:7| -18| 0
| 22:0| 28:0| -0| -0| -0| 0
| 0| 0| 0| 0| 0| 0
Sum = 1608

```

Рис. 1,2,3 Система из практического занятия

```

Input number of providers:
3
Input number of consumers:
3
Input your matrix
0
0
0
0
0
0
0
0
0
0
Input stocks:
1 1 provider:
1 2 provider:
1 3 provider:
Input needs:
1 1 consumer:
1 2 consumer:
1 3 consumer:
| 0| 0| 0| 1
| 0| 0| 0| 1
| 0| 0| 0| 1
| 1| 1| 1| 1
System is closed
Start making plan
Step 1
| 1:0| 0:0| -0| 0
| -0| 0| 0| 1
| -0| 0| 0| 1
| 0| 1| 1| 1
Step 2
| 1:0| 0:0| -0| 0
| -0| 1:0| 0:0| 0
| -0| -0| 0| 1
| 0| 0| 1| 1
Step 3
| 1:0| 0:0| -0| 0
| -0| 1:0| 0:0| 0
| -0| -0| 1:0| 0
| 0| 0| 0| 0
Sum = 0

```

Рис. 4 Система, демонстрирующая добавление нулей