 МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

**Отчет по лабораторной работе №3**

**по учебной дисциплине «Математическое программирование»**

**по теме:**

**«Решение транспортной задачи»**

Выполнил студент группы М3О-310Б-20

Кильмишкин Н. В.

Принял:

Григоревский Н. В.

Оглавление

[Задание 3](#_Toc120205665)

[Подпрограмма minElemMethod 3](#_Toc120205666)

[Подпрограмма westNorthCornerMethod 4](#_Toc120205667)

[Код основной программы 5](#_Toc120205668)

[Решение задачи 9](#_Toc120205669)

[Метод северо-западного угла 9](#_Toc120205670)

[Метод минимального элемента 11](#_Toc120205671)

[Вывод 11](#_Toc120205672)

## Задание

Решить транспортную задачу используя два метода поиска начального приближения: метод северо-западного угла и метод минимального элемента.

Исходная задача:

Матрица тарифов имеет следующий вид:

Потребности :

Запасы:

## Подпрограмма minElemMethod

|  |  |  |
| --- | --- | --- |
| **Тип переменной** | **Название** | **Описание** |
| int | consumer\_ct | Число потребителей (число столбцов) |
| int | provider\_ct | Число поставщиков (число строк) |
| int\*\*\* | cost\_mat | Указатель на матрицу стоимости доставки |
| int\*\*\* | weight\_mat | Указатель на матрицу количества доставки |
| int\*\* | consumer\_vec | Указатель на вектор потребностей потребителей |
| int\*\* | provider\_vec | Указатель на вектор возможностей поставщика |

**Код:**

void minElemMethod(

int consumer\_ct, //число потребителей (число столбцов)

int provider\_ct, //число поставщиков (число строк)

int\*\*\* cost\_mat, //матрица стоимости доставки

int\*\*\* weight\_mat, //матрица количества доставки

int\*\* consumer\_vec, //вектор потребностей потребителей

int\*\* provider\_vec //вектор возможностей поставщикоу

)

{

//мы должны найти вес для каждой ячейки, значит число итераций равно произведению числа столбцов на число строк

for (int ct = 0; ct < provider\_ct \* consumer\_ct; ct++)

{

//координаты мин. элемента

int min\_cost\_i = -1;

int min\_cost\_j = -1;

int consumer\_sum = 0; //сумма в столбце

int provider\_sum = 0; //сумма в строке

//выбираем для поиска минимума первый неопределенный элемент как опорный

for (int i = 0; i < provider\_ct; i++)

{

for (int j = 0; j < consumer\_ct; j++)

if ((\*weight\_mat)[i][j] == -1)

{

min\_cost\_i = i;

min\_cost\_j = j;

break;

}//..if

}//..for i

//поиск минимальной стоимости

for (int i = 0; i < provider\_ct; i++)

{

for (int j = 0; j < consumer\_ct; j++)

{

//элемент должен быть меньше текуцей стоимости и быть неопределенным

if ((\*cost\_mat)[i][j] < (\*cost\_mat)[min\_cost\_i][min\_cost\_j] && (\*weight\_mat)[i][j] == -1)

{

min\_cost\_i = i;

min\_cost\_j = j;

}//..if

}//..for j

}//..for i

//ищем сумму в столбце

for (int i = 0; i < provider\_ct; i++)

{

if ((\*weight\_mat)[i][min\_cost\_j] == -1) provider\_sum += 0;

else provider\_sum += (\*weight\_mat)[i][min\_cost\_j];

}//..for i

//ищем сумму в строке

for (int j = 0; j < consumer\_ct; j++)

{

if ((\*weight\_mat)[min\_cost\_i][j] == -1) consumer\_sum += 0;

else consumer\_sum += (\*weight\_mat)[min\_cost\_i][j];

}//..for i

(\*weight\_mat)[min\_cost\_i][min\_cost\_j] = min((\*provider\_vec)[min\_cost\_i] - consumer\_sum, (\*consumer\_vec)[min\_cost\_j] - provider\_sum);

}//..for ct

}//..minElemMethod()

## Подпрограмма westNorthCornerMethod

|  |  |  |
| --- | --- | --- |
| **Тип переменной** | **Название** | **Описание** |
| int | consumer\_ct | Число потребителей (число столбцов) |
| int | provider\_ct | Число поставщиков (число строк) |
| int\*\*\* | cost\_mat | Указатель на матрицу стоимости доставки |
| int\*\*\* | weight\_mat | Указатель на матрицу количества доставки |
| int\*\* | consumer\_vec | Указатель на вектор потребностей потребителей |
| int\*\* | provider\_vec | Указатель на вектор возможностей поставщику |

**Код:**

void westNorthCornerMethod(

int consumer\_ct, //число потребителей (число столбцов)

int provider\_ct, //число поставщиков (число строк)

int\*\*\* cost\_mat, //матрица стоимости доставки

int\*\*\* weight\_mat, //матрица количества доставки

int\*\* consumer\_vec, //вектор потребностей потребителей

int\*\* provider\_vec //вектор возможностей поставщикоу

)

{

//заполнение методом северо-западного угла

for (int i = 0; i < min(consumer\_ct, provider\_ct); i++)

{

int consumer\_sum = 0; //сумма в столбце

int provider\_sum = 0; //сумма в строке

//сумма элементов в строке и столбце, до текущего элемента на главной диагонали матрицы

for (int j = 0; j < i; j++)

{

consumer\_sum += (\*weight\_mat)[j][i];

provider\_sum += (\*weight\_mat)[i][j];

}//..for j

//значение текущего элемента на главной диагонале будет равно минимуму от двух чисел : отсавшегося места у потребителя и имеющегося товара у постовщика

(\*weight\_mat)[i][i] = min((\*consumer\_vec)[i] - consumer\_sum, (\*provider\_vec)[i] - provider\_sum);

//если места у потребителя меньше, чем у поставщика товаров

if ((\*weight\_mat)[i][i] == (\*consumer\_vec)[i] - consumer\_sum)

{

//так как места больше нет, то потребитель не сможет больше заказать у других поставщиков

//значит потребление у него в остальных строках равно нулю

for (int j = i + 1; j < provider\_ct; j++)

(\*weight\_mat)[j][i] = 0;

//остаток товара у поставщика

int remainder = (\*provider\_vec)[i] - provider\_sum - (\*weight\_mat)[i][i];

//заполняем остатки заказов у поставщика

for (int j = i + 1; j < consumer\_ct; j++)

{

//выбираем минимум между остатком заказов и местом у потербителя

(\*weight\_mat)[i][j] = min(remainder, (\*consumer\_vec)[j]);

remainder -= (\*weight\_mat)[i][j];

}//for j

}

else

{

//так как у поставщика больше нет товара, то потребители больше не смогут заказать товар

//значит число заказов у всех потребителей равно нулю

for (int j = i + 1; j < consumer\_ct; j++)

(\*weight\_mat)[i][j] = 0;

//остаток места у потребителя

int remainder = (\*consumer\_vec)[i] - consumer\_sum - (\*weight\_mat)[i][i];

//заполняем остатки места у потребителя

for (int j = i + 1; j < consumer\_ct; j++)

{

//выбираем минимум между остатком места и остатком товара у поставщика

(\*weight\_mat)[j][i] = min(remainder, (\*provider\_vec)[j]);

remainder -= (\*weight\_mat)[j][i];

}//for j

}//..if..else

}//..for i

}//..westNorthCornerMethod()

## Код основной программы

int main()

{

system("color F0");

int consumer\_ct = 3; //число потребителей (число столбцов)

int provider\_ct = 4; //число поставщиков (число строк)

int\*\* cost\_mat = new int\* [provider\_ct]; //матрица стоимости доставки

int\*\* weight\_mat = new int\* [provider\_ct]; //матрица количества доставки

int\*\* potential\_mat = new int\* [provider\_ct]; //матрица потенциалов

bool\*\* base\_mat = new bool\* [provider\_ct]; //вспомогательная матрица для базиса

int\* consumer\_vec = new int[consumer\_ct]; //вектор потребностей потребителей

int\* provider\_vec = new int[provider\_ct]; //вектор возможностей поставщикоу

int\* consumer\_potencial\_vec = new int[consumer\_ct]; //вектор потенциалов потребителя

int\* provider\_potencial\_vec = new int[provider\_ct]; //вектор возможностей поставщика

//координаты ячейки с мин. потенциалом

int min\_i = 0;

int min\_j = 0;

//значение функции

int total\_cost = 0;

//выделение памяти под матрицы таблицы

for (int i = 0; i < provider\_ct; i++)

{

cost\_mat[i] = new int[consumer\_ct];

weight\_mat[i] = new int[consumer\_ct];

potential\_mat[i] = new int[consumer\_ct];

base\_mat[i] = new bool[consumer\_ct];

}//..for i

//открытие файла с данными

ifstream fin("cost\_source.txt");

//заполнение матриц-таблиц

for (int i = 0; i < provider\_ct; i++)

{

for (int j = 0; j < consumer\_ct; j++)

{

int file\_read;

fin >> file\_read;

cost\_mat[i][j] = file\_read;

weight\_mat[i][j] = -1;

potential\_mat[i][j] = 0;

}//..for j

}//..for i

//заполнение векторов с потенциалами и ограничениями возможностей

for (int i = 0; i < consumer\_ct; i++)

{

int file\_read;

fin >> file\_read;

consumer\_vec[i] = file\_read;

consumer\_potencial\_vec[i] = 999;

}//..for i

for (int i = 0; i < provider\_ct; i++)

{

int file\_read;

fin >> file\_read;

provider\_vec[i] = file\_read;

provider\_potencial\_vec[i] = 999;

}//..for i

//закрытие файла с данными

fin.close();

minElemMethod(consumer\_ct, provider\_ct, &cost\_mat, &weight\_mat, &consumer\_vec, &provider\_vec);

//счетчик итераций

int iters = 1;

//минимизация функции

while (true)

{

//поиск базисов

for (int i = 0; i < provider\_ct; i++)

{

for (int j = 0; j < consumer\_ct; j++)

{

if (weight\_mat[i][j] != 0)

base\_mat[i][j] = true;

else

base\_mat[i][j] = false;

}//..for j

}//..for i

//заполнение потенциалов

for (int i = 0; i < provider\_ct; i++)

{

//в предыдущей итерации м должны были узнать потенциал строки,

//но так как i = 0 это 1-ая итерация, то потенциал будет равен 0

if (i == 0) provider\_potencial\_vec[0] = 0;

//ищем в строке базовывй элемент, так как потенциал строки известен,

//то можем при помощи него найти потенциал столбца

for (int j = 0; j < consumer\_ct; j++)

{

if (base\_mat[i][j] == true)

{

consumer\_potencial\_vec[j] = cost\_mat[i][j] - provider\_potencial\_vec[i];

//теперь, зная потенциал столбца, можно найти потенциал следующей строки,

//для следующей итерации

for (int k = i + 1; k < provider\_ct; k++)

{

if (base\_mat[k][j] == true)

{

provider\_potencial\_vec[k] = cost\_mat[k][j] - consumer\_potencial\_vec[j];

}//..if

}//..for k

}//..if

}//..for j

}//..for i

//расчет потенциалов для каждой ячейки

for (int i = 0; i < provider\_ct; i++)

{

for (int j = 0; j < consumer\_ct; j++)

potential\_mat[i][j] = cost\_mat[i][j] - provider\_potencial\_vec[i] - consumer\_potencial\_vec[j];

}//..for i

//значение функции

total\_cost = 0;

//считаем значение функции

for (int i = 0; i < provider\_ct; i++)

{

for (int j = 0; j < consumer\_ct; j++)

total\_cost += weight\_mat[i][j] \* cost\_mat[i][j];

}//..for i

//вывод таблицы

cout << "ITERATION #" << iters << endl;

printTab(consumer\_ct, provider\_ct, &cost\_mat, &weight\_mat, &potential\_mat, &consumer\_vec, &provider\_vec, &consumer\_potencial\_vec, &provider\_potencial\_vec);

cout << "\nF = " << total\_cost << endl;

//координаты ячейки с мин. потенциалом

min\_i = 0;

min\_j = 0;

//поиск ячейки с наименьшим потенциалом

for (int i = 0; i < provider\_ct; i++)

{

for (int j = 0; j < consumer\_ct; j++)

{

if (potential\_mat[min\_i][min\_j] > potential\_mat[i][j])

{

min\_i = i;

min\_j = j;

}//..if

}//..for j

}//..for i

//условие выходы из цикла

if (potential\_mat[min\_i][min\_j] >= 0) break;

//поиск контура

//флаг проверки на возможность замкунть прямоуголный контур

bool have\_rec\_contur = false;

//p для пересчета таблицы

int p = 0;

//#1 попробовать замкнуть прямоугольник

for (int i = 0; i < provider\_ct; i++)

{

if (i == min\_i) continue;

for (int j = 0; j < consumer\_ct; j++)

{

if (j == min\_j) continue;

//контур сторится по элементам базиса

if (

base\_mat[i][min\_j] == true &&

base\_mat[min\_i][j] == true &&

base\_mat[i][j] == true

)

{

p = min(weight\_mat[i][min\_j], weight\_mat[min\_i][j]);

weight\_mat[i][min\_j] -= p;

weight\_mat[min\_i][j] -= p;

weight\_mat[i][j] += p;

weight\_mat[min\_i][min\_j] += p;

have\_rec\_contur = true;

break;

}//..if

}//..for j

}//..for i

//#2 если не удалось замкнуть прямоугольник, то ищем контур n-угольник

if (!have\_rec\_contur)

{

int m = min\_i;

int n = min\_j;

int cur\_i = min\_i;

int cur\_j = min\_j;

while(true)

{

//ищем базисный элемент в текущем столбце

m = 0;

while (base\_mat[m][n] != true || m == cur\_i) m++;

cur\_i = m;

//выбираем первый элемент в контуре, либо тот, что меньше текущего

if (weight\_mat[m][n] < p || n == min\_j) p = weight\_mat[m][n];

//если мы находимся в той же строке, с которой начинали, то контур создан

if (m == min\_i) break;

//ищем базисный элемент в текущей строке

n = 0;

while (base\_mat[m][n] != true || n == cur\_j) n++;

cur\_j = n;

}//..while

//пересчет весов по контуру

m = min\_i;

n = min\_j;

cur\_i = min\_i;

cur\_j = min\_j;

weight\_mat[m][n] += p;

while (true)

{

//ищем базисный элемент в текущем столбце

m = 0;

while (base\_mat[m][n] != true || m == cur\_i) m++;

cur\_i = m;

weight\_mat[m][n] -= p;

//если мы находимся в той же строке, с которой начинали, то контур создан

if (m == min\_i) break;

//ищем базисный элемент в текущей строке

n = 0;

while (base\_mat[m][n] != true || n == cur\_j) n++;

cur\_j = n;

weight\_mat[m][n] += p;

}//..while

}//..if

cout << "P = " << p << endl << endl;

iters++; //увеличение счетчика итераций

}//..while

//освобождение памяти от массивов

for (int i = 0; i < provider\_ct; i++)

{

delete[] cost\_mat[i];

delete[] weight\_mat[i];

delete[] potential\_mat[i];

delete[] base\_mat[i];

}//..for i

delete[] cost\_mat;

delete[] weight\_mat;

delete[] potential\_mat;

delete[] base\_mat;

delete[] consumer\_vec;

delete[] provider\_vec;

delete[] consumer\_potencial\_vec;

delete[] provider\_potencial\_vec;

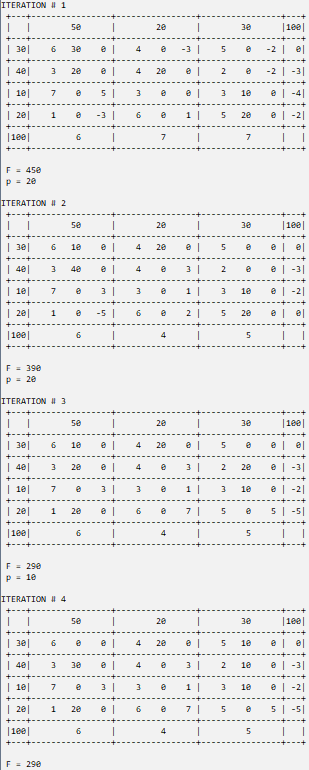
return 0;

}//..main()

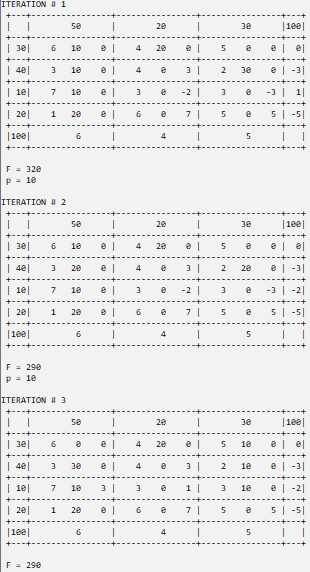
## Решение задачи

### 

### Метод северо-западного угла

Поиск минимума завершился за 4 итерации

### Метод минимального элемента



Поиск минимума завершился за 3 итерации

## Вывод

При начальном приближении, найденном методом минимального элемента, функция изначально находится ближе к решению, чем при поиске начального приближения методом северо-западного угла. Это значит, что при использовании метода минимального элемента, поиск решения с большей вероятностью займет меньшее число итерации, чем если использовать метод северо-западного угла.