Лабораторная работа №3
Транспортная задача

Выполнил: Ермаков Владимир Алексеевич

Группа: ИУ7-82

Вариант: 3

Теоретическая часть

Содержательная постановка транспортной задачи:

Имеется m производителей некоторой однородной продукции; мощность i-го производителя равна $S_i > 0$. Имеется также n потребителей этой продукции; мощность j-го потребителя равна $D_j > 0$. Стоимость перевозки единицы продукции от i-го производителя к j-му потребителю составляет c_{ij} единиц. Необходимо перевезти всю продукцию от производителей к потребителям с учетом ограничений на мощности, при этом общая стоимость перевозок должна быть минимальной.

Математическая постановка задачи:

Матрица стоимостей: $C = (c_{ij})$

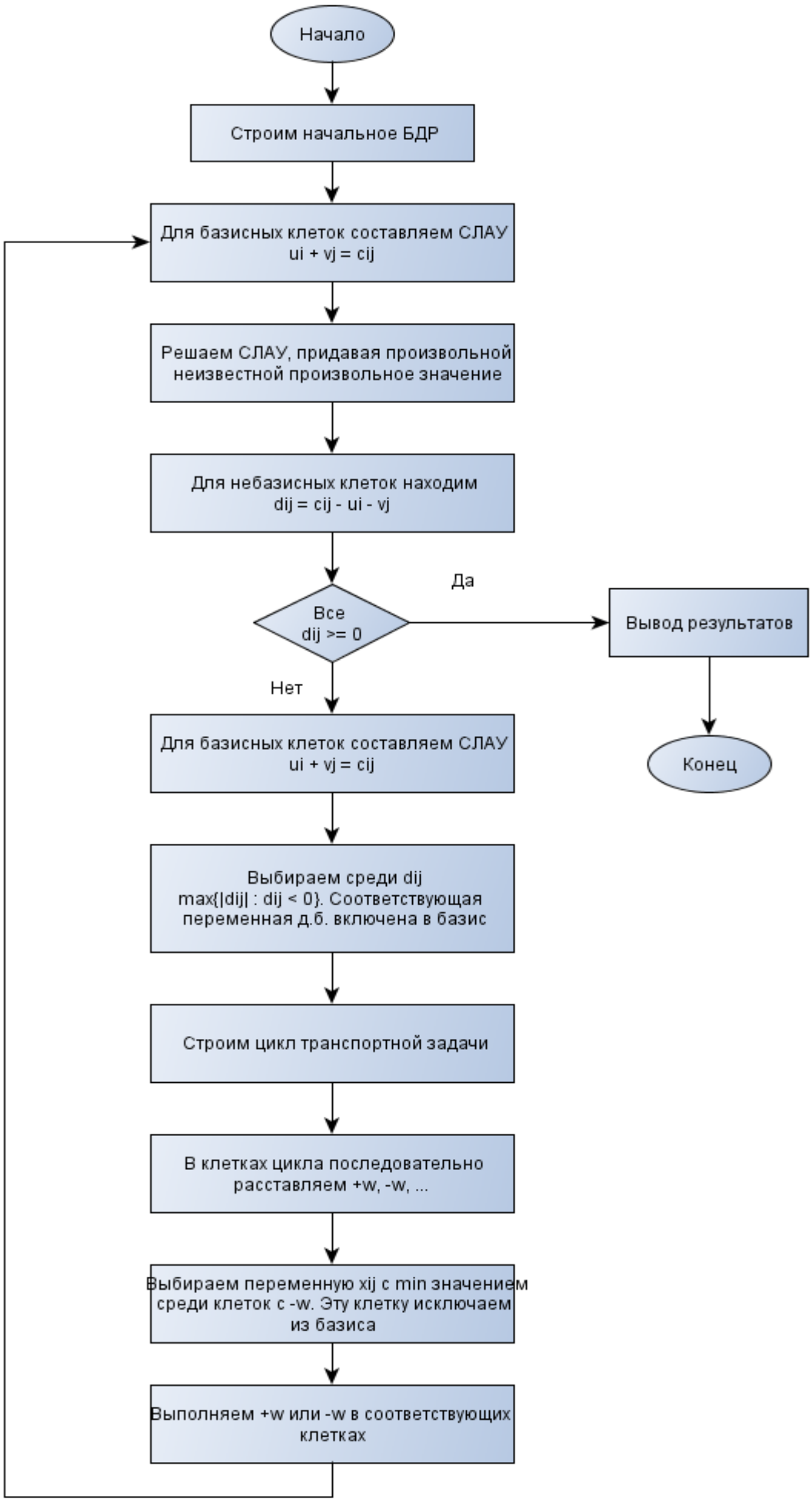
Матрица объемов перевозимой продукции: $X = (x_{ij})$

$$\begin{cases} f = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij} \rightarrow \min \\ \sum_{j=1}^{n} x_{ij} = S_i, \ i = \overline{1..m} \\ \sum_{i=1}^{m} x_{ij} = D_j, \ j = \overline{1..n} \\ x_{ij} \geq 0, \ i = \overline{1..m}, \ j = \overline{1..n} \end{cases}$$

Входные данные алгоритма: Матрица стоимостей C, вектора S и D

Выходные данные алгоритма: Матрица перевозок X

Алгоритм модифицированного симплекс-метода:



Текст программы:

```
// Переменные для работы с алгоритмом
        public int rang;   // размер матрицы для решения задачи о назначениях из Лаб№1
        public int Ssize;  // размер вектора S
        public int Dsize;  // размер вектора D
        public int Fo;     // результат
        public int[] sources;          // вектор S
        public int[] destinations;     // вектор D
        public int[,] prices;          // матрица стоимостей C
        public int[,] counts;          // транспортная таблица
        public List<Coordinate> basis; // список базисных переменных
        public List<Coordinate> path;  // путь +w, -w, ...

        #region  Решение основных алгоритмов
        private void btnSolve_Click(object sender, EventArgs e)
        {
            // решение транспортной задачи методом северо-западного угла и модифицированного симплекс-метода
            if (rbTransp.Checked)
            {
                Ssize = new int();
                Dsize = new int();
                Ssize = (int)numSrang.Value;
                Dsize = (int)numDrang.Value;

                sources = new int[Ssize];
                destinations = new int[Dsize];
                prices = new int[Ssize, Dsize];
                // читаем параметры с форм
                for (int i = 0; i < Ssize; i++)
                {
                    sources[i] = Convert.ToInt32(SView.Rows[i].Cells[0].Value);
                }
                for (int i = 0; i < Dsize; i++)
                {
                    destinations[i] = Convert.ToInt32(DView.Rows[0].Cells[i].Value);
                }
                for (int i = 0; i < Ssize; i++)
                {
                    for (int j = 0; j < Dsize; j++)
                    {
                        prices[i, j] = Convert.ToInt32(DataView.Rows[i].Cells[j].Value);
                    }
                }
            }
            else if (rbNaznachen.Checked)
            {
                rang = new int();
                rang = (int)numRang.Value;

                sources = new int[rang];
                destinations = new int[rang];
                prices = new int[rang, rang];
                // читаем параметры с форм
                for (int i = 0; i < rang; i++)
                {
                    sources[i] = Convert.ToInt32(SView.Rows[i].Cells[0].Value);
                    destinations[i] = Convert.ToInt32(DView.Rows[0].Cells[i].Value);
                }
                for (int i = 0; i < rang; i++)
                {
                    for (int j = 0; j < rang; j++)
                    {
                        prices[i, j] = Convert.ToInt32(DataView.Rows[i].Cells[j].Value);
                    }
                }

                Ssize = new int();
                Ssize = rang;
                Dsize = new int();
                Dsize = rang;
            }

            // Начальные значения установлены - решаем задачу.
            // Строим первоначальное БДР
            // методом северо-западного угла
            if (NordWestAlg(out counts, out basis) == -1)
            {
                return; // если ошибка
            }

            int count = 0;
            do
            {
                if (count == 0) ResultView.Text += "Начальное БДР: \n";
                else ResultView.Text += "Текущая транспортная таблица: \n";
                for (int i = 0; i < Ssize; i++)
                {
                    for (int j = 0; j < Dsize; j++)
                    {
                        ResultView.Text += counts[i, j].ToString() + '\t';
                    }
                    ResultView.Text += "\n";
                }
```

```
            Fo = GetFo();
            ResultView.Text += "Текущее Fo= " + Fo.ToString() + "\n";

            // Для базисных клеток составляем СЛАУ и решаем
            int[] uI, vJ;
            solveSlau(out uI, out vJ);

            // Находим D_ij для всех небазисных клеток
            int[,] dIJ = computeD(uI, vJ, basis);

            // Найдем минимальный элемент, елси он положительный, то работа алгоритма окончена
            Coordinate min = getMinElement(dIJ);

            //Если все D_ij >= 0(или минимальный >= 0), то решение найдено
            if (dIJ[min.row, min.col] >= 0)
            {
                path.Clear();
                basis = basis.Where(p => (counts[p.row, p.col] > 0)).ToList();
                Fo = GetFo();
                // Решение найдено
                ResultView.Text += "\nТекущее БДР оптимально! Результат: ";
                ResultView.Text += "\nИндексы элементов образующие цикл: ";
                foreach (Coordinate pos in basis)
                {
                    ResultView.Text += "[" + (pos.col+1).ToString() + ", " + (pos.row+1).ToString() + "] -"; ;
                }
                ResultView.Text += "\n";
                ResultView.Text += "Fo = " + Fo.ToString() + "\n";
                ResultView.Text += "Число итераций: " + count.ToString() + "\n";
                break;
            }
            else// иначе
            {
                // Выбираем минимальный d_ij и вводим в базим
                basis.Add(min);
                ResultView.Text += "Вводим в базис: [" + (min.col+1).ToString() + ", " + (min.row+1).ToString() +"] \n";

                // Строим цикл транспортной задачи, начиная с клетки,
                // отвечающей вводимой в базис переменной.
                // В клетках цикла расставляем последовательно +w, -w, +w...
                // начиная с клетки, которая отвечает вводимой в базис.
                path = getWPath(min);
                ResultView.Text += "Текущий цикл: ";
                foreach (Coordinate pos in path)
                {
                    ResultView.Text += "[" + (pos.col+1).ToString() + ", " + (pos.row+1).ToString() + "] -"; ;
                }
                ResultView.Text += "\n";


                // Выбираем переменную x_ij, которая имеет наименьшее значение
                // среди переменных, отвечающих клеткам с-w
                // исключаем ее из базиса
                // производим перекачку по циклу
                Coordinate fromBasis = getMinCount(path);
                int w = counts[fromBasis.row, fromBasis.col];
                for (int i = 0; i < path.Count; i++)
                {
                    counts[path[i].row, path[i].col] += (i % 2 == 0 ? 1 : -1) * w;
                }
                removeFromBasis(fromBasis);
                ResultView.Text += "Выводим из базиса: [" + (fromBasis.col + 1).ToString() + ", " + (fromBasis.row + 1).ToString() +
"] \n";
            }
            ResultView.Text += "\n\nНовая итерация.\n";
            count++; // следующий цикл
        }
        while (count < 100); // не более 100 циклов


    } // решение задачи о назначениях на примере Лаб1
    #endregion

    #region  Решение методом северо-западного угла
    // counts - матрица
    // basis - список позиций с базисными переменныи
    public int NordWestAlg(out int[,] counts, out List<Coordinate> basis)
    {
        int nS = Ssize;   // размер S
        int nD = Dsize;   // размер D

        counts = new int[nS, nD]; // матрица размером S*D

        basis = new List<Coordinate>(nS + nD - 1); // число базисных переменных известно

        // проверяем, что сумма по S минус сумма по D = 0
        int sum = 0;
        foreach (int s in sources)
        {
            sum += s;
        }
        foreach (int d in destinations)
        {
            sum -= d;
```

```
    }
    if (sum != 0)
    {
        ResultView.Text += "Ошибка: Разность суммы значений вектора S и суммы значений вектора D не равны 0!";
        return -1;
    }

    // Решаем северо-западным углом
    int i = 0;
    int j = 0;
    // просматриваем матрицу, пытаяюсь распределить по мощностям
    while (destinations[nD - 1] != 0)
    {
        basis.Add(new Coordinate(i, j));
        int send = Math.Min(sources[i], destinations[j]);
        counts[i, j] += send;
        sources[i] -= send;
        destinations[j] -= send;
        if (sources[i] == 0) i++;
        else j++;
    };
    return 0;
}
#endregion

#region Решение СЛАУ
public void solveSlau(out int[] uI, out int[] vJ)
{
    int nS = Ssize;
    int nD = Dsize;
    uI = new int[nS];
    vJ = new int[nD];
    // Принимаем последнее u за 0
    // Формат: v1+...+vn+u1+...+um=cij

    int n = nS + nD - 1;
    int nEq = 0;
    double[,] x = new double[n, n];
    double[] y = new double[n];
    foreach (Coordinate coord in basis)
    {
        // Добавляем уравнение
        x[nEq, coord.col] = 1; // vj
        if (coord.row != nS - 1)
            x[nEq, nD + coord.row] = 1; // ui
        y[nEq] = prices[coord.row, coord.col];
        nEq++;
    }
    alglib.densesolverreport report;
    int info;
    double[] result;
    alglib.rmatrixsolve(x, n, y, out info, out report, out result);
    // Копируем обратно
    for (int i = 0; i < nD; i++)
        vJ[i] = (int)result[i];
    for (int i = 0; i < nS - 1; i++)
        uI[i] = (int)result[nD + i];
    uI[nS - 1] = 0;
}
#endregion

#region Поиск D_ij
int[,] computeD(int[] uI, int[] vJ, List<Coordinate> basis)
{
    int nS = Ssize;
    int nD = Dsize;
    int[,] d = new int[nS, nD];
    for (int i = 0; i < nS; i++)
        for (int j = 0; j < nD; j++)
            if (!inBasis(i, j, basis))
                d[i, j] = prices[i, j] - uI[i] - vJ[j];
    return d;
}

// поиск элемента в базисе
private static bool inBasis(int row, int col, List<Coordinate> basis)
{
    return inList(row, col, basis);
}

// поиск
private static bool inList(int row, int col, List<Coordinate> list)
{
    foreach (Coordinate coord in list)
        if (coord.row == row && coord.col == col)
            return true;
    return false;
}
#endregion

#region Поиск минимального элемента
Coordinate getMinElement(int[,] matrix)
{
    int nS = Ssize;
    int nD = Dsize;
```

```
    Coordinate coord = new Coordinate(-1, -1);
    int min = int.MaxValue;
    for (int i = 0; i < nS; i++)
        for (int j = 0; j < nD; j++)
            if (min > matrix[i, j] && !inBasis(i, j, basis))
            {
                min = matrix[i, j];
                coord.row = i;
                coord.col = j;
            }
    return coord;
}
#endregion

#region Построение цикла транспортной задачи
List<Coordinate> getWPath(Coordinate start)
{
    int nS = Ssize;
    int nD = Dsize;
    List<Coordinate> startPath = new List<Coordinate>();
    startPath.Add(start);
    return pathRec(startPath, true);
    List<Coordinate> path = new List<Coordinate>();
    path.Add(start);
    int row = start.row, col = start.col;
    do
    {
        // По столбцу
        for (int i = 0; i < nS; i++)
        {
            // Базисная необработанная ячейка
            if (inBasis(i, col, basis) && !inList(i, col, path))
            {
                bool isGood = false;
                // В этой строке есть другие элементы
                for (int j = 0; j < nD; j++)
                {
                    if (j != col && inBasis(i, j, basis))
                    {
                        isGood = true;
                        break;
                    }
                }
                if (isGood)
                {
                    path.Add(new Coordinate(i, col));
                    row = i;
                    break;
                }
            }
        }
        // Можем вернуться на старт - закончили цикл
        if (row == start.row)
            break;
        // По (новой) строке
        for (int i = 0; i < nD; i++)
        {
            // Базисная необработанная ячейка
            if (inBasis(row, i, basis) && !inList(row, i, path))
            {
                bool isGood = false;
                // В этой столбце есть другие элементы
                for (int j = 0; j < nS; j++)
                {
                    if (j != row &&
                    (inBasis(j, i, basis) || (j == start.row && i == start.col)))
                    {
                        isGood = true;
                        break;
                    }
                }
                if (isGood)
                {
                    path.Add(new Coordinate(row, i));
                    col = i;
                    break;
                }
            }
        }
    }
    while (true);
    return path;
}

List<Coordinate> pathRec(List<Coordinate> currentPath, bool doRow)
{
    int nS = Ssize;
    int nD = Dsize;
    List<Coordinate> newPath = new List<Coordinate>(currentPath);
    Coordinate start = currentPath[0];
    Coordinate current = currentPath[currentPath.Count - 1];
    Coordinate newPoint = new Coordinate(-1, -1);
    newPath.Add(newPoint);
    if (doRow)
    {
```

```
            // По строке
            for (int i = 0; i < nD; i++)
            {
                // Базисная необработанная ячейка
                if (inBasis(current.row, i, basis) && !inList(current.row, i, newPath))
                {
                    // Добавляем ячейку в путь и проверяем ее рекурсивно
                    newPoint.row = current.row;
                    newPoint.col = i;
                    List<Coordinate> recursivePath = pathRec(newPath, false);
                    if (recursivePath.Count != 0)
                    {
                        return recursivePath;
                    }
                }
            }
            // Ошибочный вариант пути
            return new List<Coordinate>();
        }
        else
        {
            // По столбцу
            for (int i = 0; i < nS; i++)
            {
                // вернулись на старт
                if (start.row == i && start.col == current.col)
                {
                    newPath.RemoveAt(newPath.Count - 1); // Удаляем последнюю точку (-1, -1)
                    return newPath;
                }
                // Базисная необработанная ячейка
                if (inBasis(i, current.col, basis) && !inList(i, current.col, newPath))
                {
                    // Добавляем ячейку в путь и проверяем ее рекурсивно
                    newPoint.row = i;
                    newPoint.col = current.col;
                    List<Coordinate> recursivePath = pathRec(newPath, true);
                    if (recursivePath.Count != 0)
                    {
                        return recursivePath;
                    }
                }
            }
            // Ошибочный вариант пути
            return new List<Coordinate>();
        }
    }
    #endregion

    #region Поиск наименьшой из клеток включающих -w
    Coordinate getMinCount(List<Coordinate> path)
    {
        int min = int.MaxValue;
        Coordinate coord = new Coordinate(-1, -1);
        for (int i = 1; i < path.Count; i += 2)
        {
            int value = counts[path[i].row, path[i].col];
            if (min > value)
            {
                min = value;
                coord.row = path[i].row;
                coord.col = path[i].col;
            }
        }
        return coord;
    }
    #endregion

    #region Исключаем переменную из базиса
    void removeFromBasis(Coordinate el)
    {
        foreach (Coordinate coord in basis)
        {
            if (coord.row == el.row && coord.col == el.col)
            {
                basis.Remove(coord);
                return;
            }
        }
    }
    #endregion

    #region Расчет общей стоимости перевозок
    int GetFo()
    {
        int res = 0;
        foreach (Coordinate coord in basis)
        {
            res += counts[coord.row, coord.col]*prices[coord.row, coord.col];
        }
        return res;
    }
    #endregion
} // конец класса формы
```

Исходные данные для варианта №3:

Для транспортной задачи:

Si	135	45	170

Dj	45	45	100	160

Cij				
	6	7	3	2
	5	1	4	3
	3	2	6	2

Результат :

X				
	0	0	100	35
	0	45	0	0
	45	0	0	125

Fopt = 800

Для метода потенциалов:

1	4	7	9	4
9	3	8	7	4
3	4	6	8	2
8	2	4	6	7
7	6	9	8	5

Результат:

Fopt: 18, совпадает с результатом лабораторной работы №1.