## Отчет о лабораторной работе №3

## «Транспортная задача»

**Студент:** Шушвар Богдан Владимирович

**Группа:** ИУ7-83

**Вариант:** 13

**Цель работы:** изучение метода потенциалов на примере решения транспортной задачи и задачи о назначениях

**Содержательная постановка транспортной задачи:**

Имеется m производителей некоторой однородной продукции; мощность i-го производителя равна Si > 0. Имеется также n потребителей этой продукции; мощность j-го потребителя равна Dj > 0. Стоимость перевозки единицы продукции от i-го производителя к j-му потребителю составляет cij 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:**

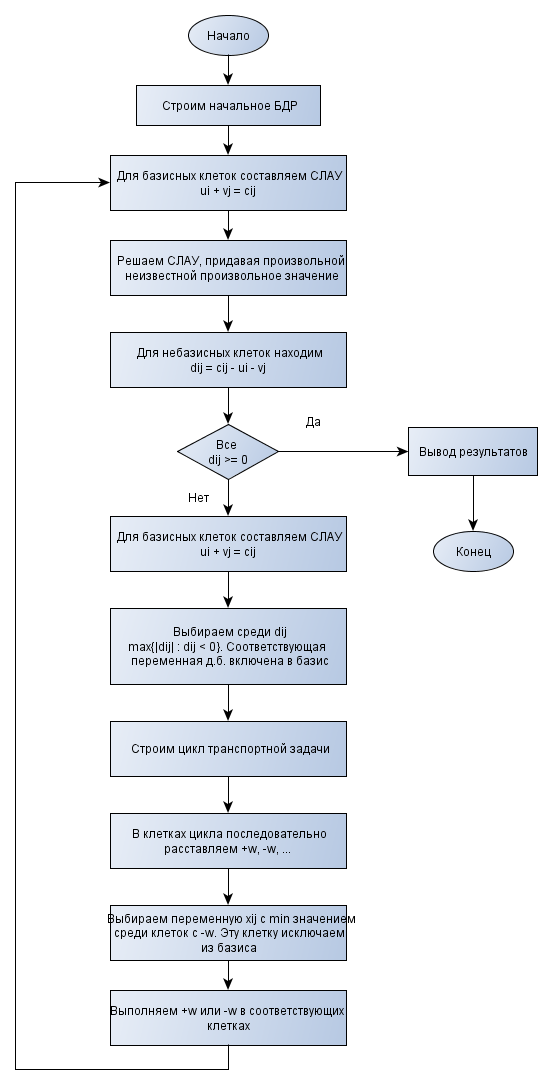
Матрица стоимостей: C = (cij)

Матрица объемов перевозимой продукции: Х = (xij)

**Входные данные:** Матрица стоимостей С, вектора S и D

**Выходные данные:** Матрица перевозок Х

**Краткое описание алгоритма:**



**Результаты расчетов:**

**Транспортная задача:**

**Исходные матрицы:**

**Матрица Х:**

**Задача о назначениях:**

**Исходные матрицы:**

**Матрица Х:**

Код программы:

public class PotentialMethod

{

public TransportTable Table { get; private set; }

private int[,] \_uv;

private int[] \_c;

private int[] \_solved;

private DElem \_toBasis;

private TransportTableElement \_fromBasis;

private List<TransportTableElement> \_cycle = new List<TransportTableElement>();

private class DElem

{

public int I { get; private set; }

public int J { get; private set; }

public int D { get; private set; }

public DElem(int i, int j, int d)

{

I = i;

J = j;

D = d;

}

private DElem()

{

}

};

private List<DElem> \_D = new List<DElem>();

private readonly int \_uvLines;

private readonly int \_uvCols;

public PotentialMethod(TransportTable table)

{

Table = table;

\_uvLines = table.M + table.N - 1;

\_uvCols = table.N + table.M;

\_uv = new int[\_uvLines, \_uvCols];

\_c = new int[\_uvLines];

\_solved = new int[\_uvLines + 1];

}

private PotentialMethod()

{

}

private void FindBasis()

{

NorthWestCornerMethod.Solve(Table);

}

public void Solve()

{

FindBasis();

int **count** = 0;

while (true)

{

Console.ForegroundColor = ConsoleColor.**Red**;

Console.WriteLine("Итерация {0}", **count**++);

Console.ResetColor();

\_cycle.Clear();

\_D.Clear();

\_toBasis = null;

\_fromBasis = null;

\_uv = new int[\_uvLines, \_uvCols];

\_c = new int[\_uvLines];

Console.ForegroundColor = ConsoleColor.**Green**;

Console.WriteLine("Транспортная таблица");

Console.ResetColor();

Console.WriteLine(Table);

MakeUVCMatrix();

SolveUVCMatrix();

FindD();

if (\_D.All(d => d.D >= 0))

{

break;

}

ChooseVarToAdd();

\_cycle.Add(Table[\_toBasis.I, \_toBasis.J]);

\_cycle = BuildCycle\_rec(\_cycle, true);

PrintCycle();

Console.WriteLine();

Console.WriteLine();

int **min** = Int32.**MaxValue**;

foreach (var **c** in \_cycle)

{

if (Table.IsBasis(**c**.I, **c**.J) && \_cycle.IndexOf(**c**) % 2 != 0)

{

if (**c**.X < **min**)

{

\_fromBasis = **c**;

**min** = **c**.X;

}

}

}

Table.Basis.RemoveAll(x => x.I == \_fromBasis.I && x.J == \_fromBasis.J);

Table.Basis.Add(Table[\_toBasis.I, \_toBasis.J]);

for (int **i** = 0; **i** < \_cycle.Count; **i**++)

{

if (**i**%2 == 0)

{

\_cycle[**i**].X += **min**;

}

else

{

\_cycle[**i**].X -= **min**;

}

}

Console.ReadKey();

}

}

public void MakeUVCMatrix()

{

int **count** = 0;

for (int **i** = 0; **i** < Table.M; **i**++)

{

for (int **j** = 0; **j** < Table.N; **j**++)

{

if (Table.IsBasis(**i**, **j**))

{

\_uv[**count**, **i**] = 1;

\_uv[**count**, Table.M + **j**] = 1;

\_c[**count**] = Table[**i**, **j**].C;

**count**++;

}

}

}

}

public void SolveUVCMatrix()

{

double[,] \_uvd = new double[\_uvLines, \_uvCols - 1];

var \_cd = new double[\_uvLines];

for (int **i** = 0; **i** < \_uvLines; **i**++)

{

for (int **j** = 0; **j** < \_uvCols - 1; **j**++)

{

\_uvd[**i**, **j**] = \_uv[**i**, **j**];

}

\_cd[**i**] = \_c[**i**];

}

double[] solved;// = new double[\_uvLines];

int info;

alglib.densesolverreport report;// = new alglib.densesolverreport();

alglib.rmatrixsolve(\_uvd, \_uvLines, \_cd, out info, out report, out solved);

\_solved[\_uvLines] = 0;

for (int **i** = 0; **i** < \_uvLines; **i**++)

{

\_solved[**i**] = (int)Math.Round(solved[**i**]);

}

}

public void FindD()

{

for (int **i** = 0; **i** < Table.M; **i**++)

{

for (int **j** = 0; **j** < Table.N; **j**++)

{

if (!Table.IsBasis(**i**, **j**))

{

\_D.Add(new DElem(**i**, **j**, Table[**i**, **j**].C - \_solved[**i**] - \_solved[Table.M + **j**]));

}

}

}

}

public void ChooseVarToAdd()

{

int **max** = Int32.**MinValue**;

DElem dmax = null;

for (int **i** = 0; **i** < \_D.Count; **i**++)

{

if (Math.Abs(\_D[**i**].D) > **max** && \_D[**i**].D < 0)

{

**max** = Math.Abs(\_D[**i**].D);

dmax = \_D[**i**];

}

}

\_toBasis = dmax;

}

public List<TransportTableElement> BuildCycle\_rec(List<TransportTableElement> currentCycle, bool doRow)

{

var newCycle = new List<TransportTableElement>(currentCycle);

TransportTableElement start = currentCycle[0];

TransportTableElement current = currentCycle[currentCycle.Count - 1];

TransportTableElement **newPoint** = new TransportTableElement(-1, -1, -1);

newCycle.Add(**newPoint**);

if (doRow)

{

for (int **j** = 0; **j** < Table.N; **j**++)

{

if (Table.IsBasis(current.I, **j**) && !newCycle.Any(x => x.I == current.I && x.J == **j**))

{

newCycle[newCycle.IndexOf(**newPoint**)] = Table[current.I, **j**];

**newPoint** = Table[current.I, **j**];

var recursivePath = BuildCycle\_rec(newCycle, false);

if (recursivePath.Count != 0)

{

return recursivePath;

}

}

}

return new List<TransportTableElement>();

}

else

{

for (int **i** = 0; **i** < Table.M; **i**++)

{

if (start.I == **i** && start.J == current.J)

{

newCycle.RemoveAt(newCycle.Count - 1);

return newCycle;

}

if (Table.IsBasis(**i**, current.J) && !newCycle.Any(x => x.I == **i** && x.J == current.J))

{

newCycle[newCycle.IndexOf(**newPoint**)] = Table[**i**, current.J];

**newPoint** = Table[**i**, current.J];

var recursivePath = BuildCycle\_rec(newCycle, true);

if (recursivePath.Count != 0)

{

return recursivePath;

}

}

}

return new List<TransportTableElement>();

}

}

public void BuildCycle()

{

\_cycle.Add(Table[\_toBasis.I, \_toBasis.J]);

int **currCol** = \_toBasis.J, **currRow** = \_toBasis.I;

while (true)

{

for (int **i** = 0; **i** < Table.M; **i**++)

{

if (Table.IsBasis(**i**, **currCol**))

{

var temp = BasisCellInRow(**i**);

if (\_cycle.Contains(Table[**i**, **currCol**]))

{

return;

}

if (temp != null && temp != Table[**i**, **currCol**] || **i** == \_toBasis.I)

{

\_cycle.Add(Table[**i**, **currCol**]);

**currRow** = **i**;

break;

}

}

}

for (int **j** = 0; **j** < Table.N; **j**++)

{

if (Table.IsBasis(**currRow**, **j**))

{

var temp = BasisCellInCol(**j**);

if (\_cycle.Contains(Table[**currRow**, **j**]))

{

return;

}

if (temp != null && temp != Table[**currRow**, **j**] || **j** == \_toBasis.J)

{

\_cycle.Add(Table[**currRow**, **j**]);

**currCol** = **j**;

break;

}

}

}

}

}

private TransportTableElement BasisCellInRow(int i)

{

for (int **j** = 0; **j** < Table.N; **j**++)

{

if (Table.IsBasis(i, **j**))

{

return Table[i, **j**];

}

}

return null;

}

private TransportTableElement BasisCellInCol(int j)

{

for (int **i** = 0; **i** < Table.M; **i**++)

{

if (Table.IsBasis(**i**, j))

{

return Table[**i**, j];

}

}

return null;

}

public void PrintCycle()

{

Console.ForegroundColor = ConsoleColor.**DarkYellow**;

foreach (var **e** in \_cycle)

{

Console.Write("x{0}{1}->", **e**.I+1, **e**.J+1);

}

Console.WriteLine("x{0}{1}", \_cycle[0].I+1, \_cycle[0].J+1);

Console.ResetColor();

}

public void PrintUVC()

{

for (int **i** = 0; **i** < \_uv.GetLength(0); **i**++)

{

for (int **j** = 0; **j** < \_uv.GetLength(1); **j**++)

{

Console.Write(\_uv[**i**, **j**] + " ");

}

Console.WriteLine(\_c[**i**]);

}

}

}

public static class NorthWestCornerMethod

{

public static void Solve(TransportTable table)

{

if (table == null)

{

throw new NullReferenceException();

}

int **i** = 0, **j** = 0;

int **tempS** = table.S[0];

int **tempD** = table.D[0];

int **count** = 0;

while (true)

{

if (**tempS** > **tempD**)

{

table[**i**, **j**].X = **tempD**;

**count**++;

table.AddToBasis(**i**, **j**);

if (**count** == table.N + table.M - 1)

{

break;

}

**tempS** -= **tempD**;

**j**++;

**tempD** = table.D[**j**];

continue;

}

if (**tempS** < **tempD**)

{

table[**i**, **j**].X = **tempS**;

**count**++;

table.AddToBasis(**i**, **j**);

if (**count** == table.N + table.M - 1)

{

break;

}

**tempD** -= **tempS**;

**i**++;

**tempS** = table.S[**i**];

continue;

}

if (**tempS** == **tempD**)

{

table[**i**, **j**].X = **tempS**;

**count**++;

table.AddToBasis(**i**, **j**);

if (**count** == table.N + table.M - 1)

{

break;

}

**i**++;

**tempD** = 0;

**tempS** = table.S[**i**];

continue;

}

}

}

}

public class TransportTableElement

{

public int C { get; set; }

public int X { get; set; }

public int I { get; set; }

public int J { get; set; }

private TransportTableElement()

{

}

public TransportTableElement(int i, int j, int c)

{

I = i;

J = j;

C = c;

}

}

public class TransportTable

{

private readonly TransportTableElement[,] \_table;

public int N { get; private set; }

public int M { get; private set; }

public int[] D { get; set; }

public int[] S { get; set; }

private readonly List<TransportTableElement> \_basis;

public List<TransportTableElement> Basis

{

get { return \_basis; }

}

public TransportTable(int m, int n, int[,] c, int[] s = null, int[] d = null)

{

N = n;

M = m;

\_table = new TransportTableElement[M, N];

for (int **i** = 0; **i** < M; **i**++)

{

for (int **j** = 0; **j** < N; **j**++)

{

\_table[**i**, **j**] = new TransportTableElement(**i** ,**j**, c[**i**,**j**]);

}

}

D = new int[N];

S = new int[M];

\_basis = new List<TransportTableElement>();

}

private TransportTable()

{

}

public TransportTableElement this[int i, int j]

{

get { return \_table[i, j]; }

set { \_table[i, j] = value; }

}

public bool IsBasis(int i, int j)

{

return \_basis.Exists(e => e.I == i && e.J == j);

}

public void AddToBasis(int i, int j)

{

\_basis.Add(\_table[i, j]);

}

public override string ToString()

{

var sb = new StringBuilder();

for (int **i** = 0; **i** < M; **i**++)

{

for (int **j** = 0; **j** < N; **j**++)

{

string b = "";

if (IsBasis(**i**, **j**))

{

b = "\*";

}

sb.Append(\_table[**i**, **j**].X + b + " ");

}

sb.AppendLine();

}

return sb.ToString();

}

}