Ministerul Educației, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

RAPORT

Lucrarea de laborator Nr.4
la disciplina Modele si metode de calcul (2)

TEMA: Rezolvarea problemelor de programare liniară prin metoda Simplex

A efectuat: st.gr.TI-194
Duca Dumitru
A verificat: asis. univ.
Buldumac Oleg

Chisinau 2020

Tema: Rezolvarea problemelor de programare liniară prin metoda Simplex.

Scopul lucrarii:

- De gasit o problema din lumea reala care are cel putin 4 variabile si de rezolvat prin metoda simplex.
- De rezolvat problema analitic, pe foaie si la nivel de cod.
- De efectuat un raport unnde va trebui sa fie problema si modelul matematic (functia obiectiva si constrangile (sisteme de inecuatii / ecuatii)).

Formularea problemei:
Problema din lab.3 cu adăugarea a 2 variabile supimentare:

Un local ofera clientilor sai 4 tipuri de bussines-lunchuri: de tip T_1, T_2, T_3 si T_4. Profitul obtinut in urma vanzarii unui lunch de tip T_1 este de 40 de lei. Profitul obtinut in urma vanzarii unui lunch de tip T_2 este de 45 de lei, pentru un lunch de tip T_3 aceasta valoare atinge 75 de lei, iar pentru un lunch de tip T_4 – 30 de lei. Costul de productie a unui lunch de tip T_1 este de 60 de lei. Costul de productie a uni lunch de tip T_2 este de 50 de lei. Pentru un lunch de tip T_3 aceasta valoare atinge 100 de lei, iar pentru un lunch T_4 – 42 de lei. Pentru a produce lunchuri de tip T_1 si T_2 s-a preconizat o valoare maxima de 870 lei / zi. Pentru a produce lunchuri de tip T_2 si T_3 – 750 de lei / zi. Pentru a produce lunchuri de tip T_3 si T_4 – 752 de lei, iar pentru a produce doar lunchuri de tip T_4 – 350 de lei. Cate lunchuri de fiecare fel trebuie sa se produca pe zi, pentru ca profitul sa fie maxim?

Rezolvare:

Fie: x1=nr de lunchuri de tip T_1 produse într-o zi
x2=nr de lunchuri de tip T_2 produse într-o zi
x3=nr de lunchuri de tip T_3 produse într-o zi
x4=nr de lunchuri de tip T_4 produse într-o zi

Restricțiile:
1)x1,x2,x3,x4 trebuie să fie nenegative
2)Prețul de producție a unui lunch de tip T_1 este 60 lei, pentru un lunch de tip tip T_2 este 50 lei.Pentru producerea lor s-au preconizat 870 lei.
3) Prețul de producție a unui lunch de tip T_2 este 50 lei, pentru un lunch de tip tip T_3 este 100 lei.Pentru producerea lor s-au preconizat 750 lei.
4)Prețul de producție a unui lunch de tip T_3 este 100 lei, pentru un lunch de tip tip T_4 este 42 lei.Pentru producerea lor s-au preconizat 752 lei.
5) Prețul de producție a unui lunch de tip T_4 este 42 lei.Pentru producere a lunchrui de tip 4 s-au preconizat 350 lei.

Funcția obiectivă: F(x1,x2,x3,x4)=p
p=40x1+45x2+75x3+30x4- care trebuie de maximizat

Restricții: $\begin{cases} x1 \geq 0, x2 \geq 0, x3 \geq 0, x4 \geq 0 \\ 60x1 + 50x2 \leq 870 \\ 50x2 + 100x3 \leq 750 \\ 100x3 + 42x4 \leq 752 \\ 42x4 \leq 350 \\ p - 40x1 - 45x2 - 75x3 - 30x4 = 0 \end{cases}$

Adăugarea variabilelor auxiliare și transformarea inecuațiilor în ecuații: $\begin{cases} 60x1 + 50x2 + r = 870 \\ 50x2 + 100x3 + s = 750 \\ 100x3 + 42x4 + t = 752 \\ 42x4 + u = 350 \end{cases}$

Tabelul nr.1:

Variabilele	X1	X2	X3	X4	r	s	t	u	Valoarea	Operații asupra rândurilor	Q=valoarea/coloana piv.
r	60	50	0	0	1	0	0	0	870	R1	870/0=INF
s	0	50	100	0	0	1	0	0	750	R2	750/100=7.5
t	0	0	100	42	0	0	1	0	752	R3	752/100=7.52
u	0	0	0	42	0	0	0	1	350	R4	350/0=INF
p	-40	-45	**-75**	-30	0	0	0	0	0	R5	

-75 –coloana cu cel mai mic element negativ; 100-PIVOT

Tabelul nr.2:

Variabilele	X1	X2	X3	X4	r	s	t	u	Valoarea	Operații asupra rândurilor	Q=valoarea/ coloana piv.
R	60	50	0	0	1	0	0	0	870	R6=R1	870/60=1.45
X3	0	0.5	1	0	0	0.01	0	0	7.5	R7=R2/100	7.5/0=INF
t	0	-50	0	42	0	-1	1	0	2	R8=R3-100R7	2/0=INF
u	0	0	0	42	0	0	0	1	350	R9=R4	350/0=INF
p	**-40**	-7.5	0	-30	0	0.75	0	0	562.5	R10=R5+75*R7	

-40 –coloana cu cel mai mic element negativ ; 60-PIVOT

Tabelul nr.3:

Variabilele	X1	X2	X3	X4	r	s	t	u	Valoarea	Operații asupra rândurilor	Q=valoarea/coloana piv.
X1	1	50	0	0	0.016	0	0	0	14.5	R11=R6/60	14.5/0=INF
X3	0	0.5	1	0	0	0.01	0	0	7.5	R12=R7	7.5/0=INF
t	0	-50	0	42	0	-1	1	0	2	R13=R8	2/42=0.047
u	0	0	0	42	0	0	0	1	350	R14=R9	350/42=8.333
p	0	25.833	0	**-30**	0.6	0.75	0	0	1142.5	R15=R10+40R11	

-30 –coloana cu cel mai mic element negativ ; 42-PIVOT

Tabelul nr.4:

Variabile	X1	X2	X3	X4	r	s	t	u	Valoarea	Operații asupra rândurilor	Q=valoarea/coloana piv.
X1	1	0.83	0	0	0.016	0	0	0	14.5	R16=R11	14.5/0.83=17.46
X3	0	0.5	1	0	0	0.01	0	0	7.5	R17=R12	7.5/0.5=15
X4	0	-1.19	0	1	0	-0.023	0.023	0	0.04	R18=R13/42	0.04/-1.19=0.03
u	0	50	0	0	0	0	0	1	348.32	R19=R14-42*R18	348.32/50=6.96
p	0	**-9.96**	0	0	0.64	0.06	0.69	0	1143.7	R20=R15+30*R18	

-9.96–coloana cu cel mai mic element negativ ; 50-PIVOT

Tabelul nr.5:

Variabile	X1	X2	X3	X4	r	s	t	u	Valoarea	Operații asupra rândurilor	Q
X1	1	0	0	0	0.016	-0.016	0.016	-0.016	8.718	R21=R16-0.83*R24	-
X3	0	0	1	0	0	0	0.01	-0.01	4.017	R22=R17-0.5*R24	-
X4	0	0	0	1	0	0.0008	-0.0008	0.023	8.322	R23=R18+1.19*R24	-
X2	0	1	0	0	0	0.02	-0.02	0.02	6.966	R24=R19/50	-
p	0	0	0	0	0.064	0.259	0.49	0.199	1213.13	R25=R20+9.967*R24	-

Am obținut următoarele rezultate:
x1=8.718
x2=6.966
x3=4.017
x4=8.322
p=1213.13

Deci, funcția atinge valoarea maximală de 1213,13 în punctul M(8.718,6.966,4.017,8.322).Pentru ca să fie profitul maximal, este nevoie de produs zilnic 9 lunchuri de tip T_1, 7 lunchuri de tip T_2, 4 lunchuri de tip T_3 și 8 lunchuri de tip T_4.

.

Implementarea metodei Simplex la nivel de cod:

1. Listing-ul programului/Github source:

```
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;

class Simplex {

private:
    int rows, cols;
    //stores coefficients of all the variables
    std::vector <std::vector<float> > A;
    //stores constants of constraints
    std::vector<float> B;
    //stores the coefficients of the objective function
    std::vector<float> C;

    float maximum;

    bool isUnbounded;

public:
    Simplex(std::vector <std::vector<float> > matrix, std::vector<float> b, std::vector<float> c)
{
        maximum = 0;
        isUnbounded = false;
        rows = matrix.size();
        cols = matrix[0].size();
        A.resize(rows, vector<float>(cols, 0));
        B.resize(b.size());
        C.resize(c.size());



        for (int i = 0; i < rows; i++) {             //pass A[][] values to the metrix
            for (int j = 0; j < cols; j++) {
                A[i][j] = matrix[i][j];

            }
        }



        for (int i = 0; i < c.size(); i++) {      //pass c[] values to the B vector
            C[i] = c[i];
        }
        for (int i = 0; i < b.size(); i++) {      //pass b[] values to the B vector
            B[i] = b[i];
        }
```

```
    }

    bool simplexAlgorithmCalculataion() {
        //check whether the table is optimal,if optimal no need to process further
        if (checkOptimality() == true) {
            return true;
        }

        //find the column which has the pivot.The least coefficient of the objective function(C
array).
        int pivotColumn = findPivotColumn();


        if (isUnbounded == true) {
            cout << "Error unbounded" << endl;
            return true;
        }

        //find the row with the pivot value.The least value item's row in the B array
        int pivotRow = findPivotRow(pivotColumn);

        //form the next table according to the pivot value
        doPivotting(pivotRow, pivotColumn);

        return false;
    }

    bool checkOptimality() {
        //if the table has further negative constraints,then it is not optimal
        bool isOptimal = false;
        int positveValueCount = 0;

        //check if the coefficients of the objective function are negative
        for (int i = 0; i < C.size(); i++) {
            float value = C[i];
            if (value >= 0) {
                positveValueCount++;
            }
        }
        //if all the constraints are positive now,the table is optimal
        if (positveValueCount == C.size()) {
            isOptimal = true;
            print();
        }
        return isOptimal;
    }

    void doPivotting(int pivotRow, int pivotColumn) {

        float pivetValue = A[pivotRow][pivotColumn];//gets the pivot value

        float* pivotRowVals = new float[cols];//the column with the pivot

        float* pivotColVals = new float[rows];//the row with the pivot

        float* rowNew = new float[cols];//the row after processing the pivot value

        maximum = maximum - (C[pivotColumn] * (B[pivotRow] / pivetValue));  //set the maximum
step by step
         //get the row that has the pivot value
        for (int i = 0; i < cols; i++) {
            pivotRowVals[i] = A[pivotRow][i];
```

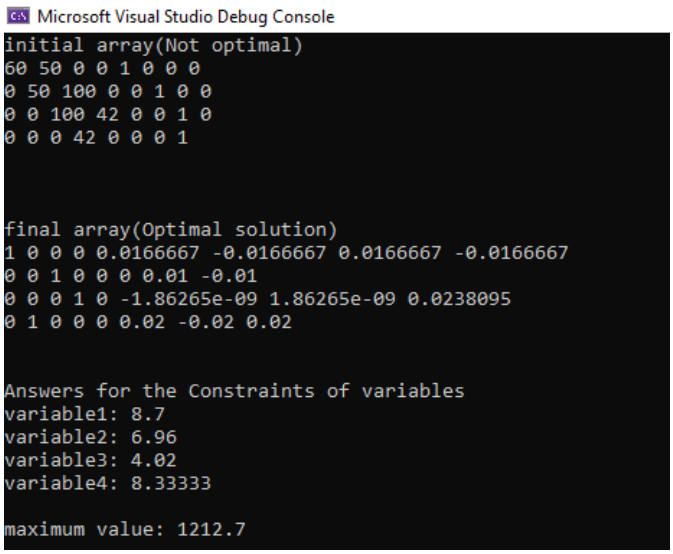
```
        }
        //get the column that has the pivot value
        for (int j = 0; j < rows; j++) {
            pivotColVals[j] = A[j][pivotColumn];
        }

        //set the row values that has the pivot value divided by the pivot value and put into new
row
        for (int k = 0; k < cols; k++) {
            rowNew[k] = pivotRowVals[k] / pivetValue;
        }

        B[pivotRow] = B[pivotRow] / pivetValue;


        //process the other coefficients in the A array by subtracting
        for (int m = 0; m < rows; m++) {
            //ignore the pivot row as we already calculated that
            if (m != pivotRow) {
                for (int p = 0; p < cols; p++) {
                    float multiplyValue = pivotColVals[m];
                    A[m][p] = A[m][p] - (multiplyValue * rowNew[p]);
                    //C[p] = C[p] - (multiplyValue*C[pivotRow]);
                    //B[i] = B[i] - (multiplyValue*B[pivotRow]);
                }

            }
        }

        //process the values of the B array
        for (int i = 0; i < B.size(); i++) {
            if (i != pivotRow) {

                float multiplyValue = pivotColVals[i];
                B[i] = B[i] - (multiplyValue * B[pivotRow]);

            }
        }
        //the least coefficient of the constraints of the objective function
        float multiplyValue = C[pivotColumn];
        //process the C array
        for (int i = 0; i < C.size(); i++) {
            C[i] = C[i] - (multiplyValue * rowNew[i]);

        }


        //replacing the pivot row in the new calculated A array
        for (int i = 0; i < cols; i++) {
            A[pivotRow][i] = rowNew[i];
        }


    }

    //print the current A array
    void print() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                cout << A[i][j] << " ";
            }
            cout << "" << endl;
        }
        cout << "" << endl << endl;
    }
```

```
//find the least coefficients of constraints in the objective function's position
int findPivotColumn() {

    int location = 0;
    float minm = C[0];



    for (int i = 1; i < C.size(); i++) {
        if (C[i] < minm) {
            minm = C[i];
            location = i;
        }
    }

    return location;

}

//find the row with the pivot value.The least value item's row in the B array
int findPivotRow(int pivotColumn) {
    float* positiveValues = new float[rows];
    std::vector<float> result(rows, 0);
    //float result[rows];
    int negativeValueCount = 0;

    for (int i = 0; i < rows; i++) {
        if (A[i][pivotColumn] > 0) {
            positiveValues[i] = A[i][pivotColumn];
        }
        else {
            positiveValues[i] = 0;
            negativeValueCount += 1;
        }
    }
    //checking the unbound condition if all the values are negative ones
    if (negativeValueCount == rows) {
        isUnbounded = true;
    }
    else {
        for (int i = 0; i < rows; i++) {
            float value = positiveValues[i];
            if (value > 0) {
                result[i] = B[i] / value;

            }
            else {
                result[i] = 0;
            }
        }
    }
    //find the minimum's location of the smallest item of the B array
    float minimum = 99999999;
    int location = 0;
    for (int i = 0; i < rows; i++) {
        if (result[i] > 0) {
            if (result[i] < minimum) {
                minimum = result[i];

                location = i;
            }
        }

    }
```

```
        return location;

    }

    void CalculateSimplex() {
        bool end = false;

        cout << "initial array(Not optimal)" << endl;
        print();

        cout << " " << endl;
        cout << "final array(Optimal solution)" << endl;


        while (!end) {

            bool result = simplexAlgorithmCalculataion();

            if (result == true) {

                end = true;


            }
        }
        cout << "Answers for the Constraints of variables" << endl;
        int iterator = 1;
        for (int i = 0; i < A.size(); i++) {  //every basic column has the values, get it form B
array
            int count0 = 0;
            int index = 0;
            for (int j = 0; j < rows; j++) {
                if (A[j][i] == 0.0) {
                    count0 += 1;
                }
                else if (A[j][i] == 1) {
                    index = j;
                }


            }

            if (count0 == rows - 1) {

                cout << "variable" << iterator << ": " << B[index] << endl;
                iterator++;//every basic column has the values, get it form B array
            }
            else {
                cout << "variable" << iterator << ": " << 0 << endl;
                iterator++;

            }

        }


        cout << "" << endl;
        cout << "maximum value: " << maximum << endl;  //print the maximum values




    }
```

```
};

int main()
{

    int colSizeA = 8;  //should initialise columns size in A
    int rowSizeA = 4;  //should initialise columns row in A[][] vector

    float C[] = { -40,-45,-75,-30,0,0,0,0 };  //should initialis the c arry here
    float B[] = { 870,750,752,350 };  // should initialis the b array here



    float a[4][8] = {    //should intialis the A[][] array here
                   { 60,  50,  0, 0, 1, 0, 0, 0},
                   { 0,  50,  100, 0, 0, 1, 0, 0},
                   { 0,  0,  100, 42, 0, 0, 1, 0},
                   {0, 0, 0, 42, 0, 0 ,0, 1}
    };


    std::vector <std::vector<float> > vec2D(rowSizeA, std::vector<float>(colSizeA, 0));

    std::vector<float> b(rowSizeA, 0);
    std::vector<float> c(colSizeA, 0);




    for (int i = 0; i < rowSizeA; i++) {         //make a vector from given array
        for (int j = 0; j < colSizeA; j++) {
            vec2D[i][j] = a[i][j];
        }
    }






    for (int i = 0; i < rowSizeA; i++) {
        b[i] = B[i];
    }

    for (int i = 0; i < colSizeA; i++) {
        c[i] = C[i];
    }


    // hear the make the class parameters with A[m][n] vector b[] vector and c[] vector
    Simplex simplex(vec2D, b, c);
    simplex.CalculateSimplex();


    return 0;
}
```

Rezultatul executiei programului:



Concluzie:

In urma efectuarii acestei lucrari am aplicat in practica algoritmul Simplex pentru a rezolva o problema din viata reala de 4 variabile. Rezolvand problema prin metoda data, am observat ca metoda simplex este o metoda complexa, dar eficienta in rezolvarea problemelor de 4 variabile sau mai mult, in comparatie cu metoda grafica, care face imposibila rezolvarea, din cauza faptului ca nu este posibil de gasit un punct in spatiul 3D, cu atat mai mult in spatii cu mai multe dimensiuni. Ruland programul am constatat ca rezultatele programului sunt aceleasi cu cele obtinute rezolvand pe foaie.