# Цель работы

Изучение венгерского метода решения задачи о назначениях.

# Содержательная постановка задачи о назначениях

В распоряжении работодателя имеется N работ (заданий) и N исполнителей. Затраты на выполнение i-м работником j-й работы составляют Сij условных единиц. Каждый работник может выполнить только одно задание, при этом все задания должны быть выполнены. Необходимо составить такой план распределения работ между исполнителями, чтобы общие затраты были минимальны.

# Математическая постановка задачи о назначениях.

Для задачи максимизации:

Необходимо решить эквивалентную задачу минимизации:

Где С'ij = τ – Сij, τ – максимальный элемент матрицы назначений С.

# Исходные данные

# Краткое описание венгерского метода

1. В каждом столбце матрицы С находим наименьший элемент и вычитаем его из этого столбца.
2. В каждой строке полученной матрицы находим наименьший элемент и вычитаем его из этой строки. Получили матрицу С~.
3. Строим первоначальную систему независимых нулей: просматриваем матрицу С~ по столбцам сверху вниз слева направо в поисках первого нуля. Если в одной строке с найденным нулем нет 0\*, отмечаем \* найденный нуль. Обозначим через k число 0\* в полученной системе.
4. Если k=N, переход к п. 14, иначе – к п.5.
5. Выделим ‘+’ столбцы, содержащие 0\*.
6. Если среди невыделенных элементов есть 0, переход к п.8, иначе – к п.7.
7. Среди невыделенных элементов находим наименьший элемент h>0, вычитаем его из невыделенных строк и добавляем к выделенным столбцам. Среди полученных невыделенных элементов выбираем 0.
8. Отмечаем выбранный 0 как 0'.
9. Если в одной строке с выделенным 0' нет 0\*, переход к п.11, иначе – к п.10.
10. Выделяем ‘+’ строку с 0' и снимаем выделение со столбца, в котором стоит 0\* из той же строки. Переход к п. 6.
11. Строим непродолжаемую L-цепочку: от текущего 0' по столбцу к 0\*, по строке к 0' и т.д. Цепочка заканчивается на 0'.
12. В найденной L-цепочке 0\* заменяем на 0, 0' – на 0\*.
13. Снимаем все выделения. Переход к п. 4.
14. Записываем оптимальное решение: в позиции матрицы X, соответствующие позициям с 0\*матрицы С~, ставим 1, в остальные – 0.
15. Печать Хopt, f(Xopt).
16. Конец.

# Результаты расчетов

**Для задачи минимизации:**

Исходная матрица:

10 12 7 11 10

12 5 12 7 12

8 6 7 8 13

8 11 5 9 9

10 8 9 11 11

Начальная система независимых нулей:

1 6 1 3 0\*

4 0\* 7 0 3

0\* 1 2 1 4

0 6 0\* 2 0

0 1 2 2 0

Среди невыделенных элементов есть 0; Позиция (1,3)

1 6 1 3 0\*

4 0\* 7 0' 3

0\* 1 2 1 4

0 6 0\* 2 0

0 1 2 2 0

Среди невыделенных элементов есть 0; Позиция (2,1)

1 5 1 2 0\*

5 0\* 8 0' 4

0\* 0' 2 0 4

0 5 0\* 1 0

0 0 2 1 0

Среди невыделенных элементов есть 0; Позиция (3,0)

1 5 1 2 0\*

5 0\* 8 0' 4

0\* 0' 2 0 4

0' 5 0\* 1 0

0 0 2 1 0

Среди невыделенных элементов есть 0; Позиция (4,0)

1 5 1 2 0\*

5 0\* 8 0' 4

0\* 0' 2 0 4

0' 5 0\* 1 0

0' 0 2 1 0

L-цепочка: (4,0), (2,0), (2,1), (1,1), (1,3),

1 5 1 2 0\*

5 0 8 0\* 4

0 0\* 2 0 4

0 5 0\* 1 0

0\* 0 2 1 0

Матрица назначений:

0 0 0 0 1

0 0 0 1 0

0 1 0 0 0

0 0 1 0 0

1 0 0 0 0

Целевая функция: 38

**Для задачи максимизации:**

Исходная матрица:

10 12 7 11 10

12 5 12 7 12

8 6 7 8 13

8 11 5 9 9

10 8 9 11 11

Эквивалентная матрица

3 1 6 2 3

1 8 1 6 1

5 7 6 5 0

5 2 8 4 4

3 5 4 2 2

Начальная система независимых нулей:

2 0\* 5 0 3

0\* 7 0 4 1

4 6 5 3 0\*

3 0 6 1 3

2 4 3 0\* 2

Среди невыделенных элементов есть 0; Позиция (1,2)

2 0\* 5 0 3

0\* 7 0' 4 1

4 6 5 3 0\*

3 0 6 1 3

2 4 3 0\* 2

Среди невыделенных элементов есть 0; Позиция (0,0)

0' 0\* 3 0 3

0\* 9 0' 6 3

2 6 3 3 0\*

1 0 4 1 3

0 4 1 0\* 2

Среди невыделенных элементов есть 0; Позиция (4,0)

0' 0\* 3 0 3

0\* 9 0' 6 3

2 6 3 3 0\*

1 0 4 1 3

0' 4 1 0\* 2

Среди невыделенных элементов есть 0; Позиция (3,1)

0' 0\* 3 0 3

0\* 9 0' 6 3

2 6 3 3 0\*

1 0' 4 1 3

0' 4 1 0\* 2

L-цепочка: (3,1), (0,1), (0,0), (1,0), (1,2),

0\* 0 3 0 3

0 9 0\* 6 3

2 6 3 3 0\*

1 0\* 4 1 3

0 4 1 0\* 2

Матрица назначений:

1 0 0 0 0

0 0 1 0 0

0 0 0 0 1

0 1 0 0 0

0 0 0 1 0

Целевая функция: 57

# Листинг программы

class VengerMatrix

{

// Класс для представления матрицы стоимостей

struct Element

{

/// <summary>

/// Структура для представления элементов матрицы

/// </summary>

public int \_value; // значение

public char \_mark; // отметка

public Element(int val, char mark = ' ')

{

\_value = val;

\_mark = mark;

}

}

private Element[][] \_c; // матрица стоимостей

private int[] \_markedRows, \_markedColumns; // номера выделенных строк и столбцов

// properties

public int N { get; set; } // размерность матрицы

public int K { get; set; } // число нулей в СНН

// Constructors

public VengerMatrix(int n)

{

N = n;

K = 0;

\_c = new Element[n][];

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

\_c[i] = new Element[n];

\_markedRows = new int[n];

\_markedColumns = new int[n];

}

// Methods

// Получить минимаьный невыделенный элемент

private int getMinFromUnmarked()

{

int min = Int32.MaxValue;

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

{

if (\_markedColumns[j] == 1)

continue;

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

{

if (\_markedRows[i] == 1)

continue;

if (\_c[i][j].\_value < min)

min = \_c[i][j].\_value;

}

}

return min;

}

/// <summary>

/// Задать значения матрицы

/// </summary>

public void setMatrix(int[][] c)

{

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

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

\_c[i][j] = new Element(c[i][j]);

}

/// <summary>

/// Печать матрицы

/// </summary>

public String print()

{

String s = "";

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

{

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

s = s +\_c[i][j].\_value + \_c[i][j].\_mark + "\t";

s += "\n";

}

return s;

}

/// <summary>

/// вычисление целевой функции

/// </summary>

public int f(int[][] opt)

{

int sum = 0;

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

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

if (opt[i][j] == 1)

sum += \_c[i][j].\_value;

return sum;

}

/// <summary>

/// Получить копию матрицы

/// </summary>

public VengerMatrix Copy()

{

VengerMatrix m = new VengerMatrix(N);

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

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

m.\_c[i][j] = new Element(\_c[i][j].\_value, \_c[i][j].\_mark);

return m;

}

/// <summary>

/// Получить максимальный элемент матрицы

/// </summary>

public int getMax()

{

int max = -1;

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

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

if (\_c[i][j].\_value > max)

max = \_c[i][j].\_value;

return max;

}

/// <summary>

/// Вычесть все элементы матрицы из числа tau

/// </summary>

public void subFromTau(int tau)

{

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

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

\_c[i][j].\_value = tau - \_c[i][j].\_value;

}

/// <summary>

/// Получить минимальный элемент в столбце

/// </summary>

public int getMinInColumn(int j)

{

int min = Int32.MaxValue;

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

if (\_c[i][j].\_value < min)

min = \_c[i][j].\_value;

return min;

}

/// <summary>

/// Получить минимальный элемент в строке

/// </summary>

public int getMinInRow(int i)

{

int min = Int32.MaxValue;

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

if (\_c[i][j].\_value < min)

min = \_c[i][j].\_value;

return min;

}

/// <summary>

/// Вычитание одного и того же числа из строки матрицы

/// </summary>

public void subFromRow(int i, int value)

{

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

\_c[i][j].\_value -= value;

}

/// <summary>

/// Вычитание одного и того же числа из столбца матрицы

/// </summary>

public void subFromColumn(int j, int value)

{

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

\_c[i][j].\_value -= value;

}

/// <summary>

/// Выделить элемент матрицы

/// </summary>

public void markElement(int i, int j, char sym)

{

\_c[i][j].\_mark = sym;

}

/// <summary>

/// Проверка наличия в строке выделенного элемента

/// </summary>

public int hasMarkedElementInRow(int i, char sym)

{

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

if (\_c[i][j].\_mark == sym)

return j;

return -1;

}

/// <summary>

/// Проверка наличия в столбце выделенного элемента

/// </summary>

public int hasMarkedElementInColumn(int j, char sym)

{

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

if (\_c[i][j].\_mark == sym)

return i;

return -1;

}

/// <summary>

/// Получить эл-т матрицы (i, j)

/// </summary>

public int get(int i, int j)

{

return \_c[i][j].\_value;

}

/// <summary>

/// Выделить столбцы, содержащие 0\*

/// </summary>

public void markColumns()

{

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

if (hasMarkedElementInColumn(j, '\*') != -1)

\_markedColumns[j] = 1;

}

/// <summary>

/// Выделить строку с номером i

/// </summary>

public void markRow(int i)

{

\_markedRows[i] = 1;

}

/// <summary>

/// Снять выделение со столбца

/// </summary>

public void unmarkColumn(int j)

{

\_markedColumns[j] = 0;

}

/// <summary>

/// Снять выделение со всех строк и столбцов

/// </summary>

public void unmarkAll()

{

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

{

\_markedColumns[i] = 0;

\_markedRows[i] = 0;

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

if (\_c[i][j].\_mark == '\'')

\_c[i][j].\_mark = ' ';

}

}

/// <summary>

/// Проверка: "Среди невыделенных эл-тов есть нуль?"

/// </summary>

public bool hasUnmarkedNulls(out int i, out int j)

{

i = -1;

j = -1;

for (int l=0; l<N; l++)

{

if (\_markedColumns[l] == 1)

continue;

for (int k = 0; k < N; k++)

{

if (\_markedRows[k] == 1)

continue;

if (\_c[k][l].\_value == 0)

{

i = k;

j = l;

return true;

}

}

}

return false;

}

/// <summary>

/// Обновление матрицы в случае, когда среди невыделенных элементов не оказалось нуля

/// </summary>

public void update()

{

int min = getMinFromUnmarked();

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

{

// Вычесть из невыделенной строки

if (\_markedRows[i] != 1)

subFromRow(i, min);

// Добавить к выделенному столбцу

if (\_markedColumns[i] == 1)

subFromColumn(i, -min);

}

}

/// <summary>

/// Замена по L-цепочке

/// </summary>

public void replaceInChain(LChain chain)

{

for (int k = 0; k < chain.N; k++)

{

int i = -1, j = -1;

chain.get(k, out i, out j);

if (\_c[i][j].\_mark == '\*')

\_c[i][j].\_mark = ' ';

else

{

\_c[i][j].\_mark = '\*';

K++;

}

}

}

/// <summary>

/// Построение матрицы назначений

/// </summary>

public int[][] createOptimalMatrix()

{

int[][] opt = new int [N][];

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

{

opt[i] = new int[N];

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

if (\_c[i][j].\_mark == '\*')

opt[i][j] = 1;

else

opt[i][j] = 0;

}

return opt;

}

}

}