НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ ТА СПЕЦІАЛІЗОВАНИХ КОМП’ЮТЕРНИХ СИСТЕМ

**Лабораторна робота №4  
з дисципліни «Системне програмне забезпечення»**

Виконав  
студент 4-го курсу  
групи КВ-41  
Курач Віктор

Київ – 2017

**Постановка задачі**

Скласти програму реалізації угорського методу для неоднорідної обчислювальної системи.

**Опис алгоритму**

1. У кожному рядку знайти мінімальний елемент та відняти його від усіх елементів рядка.
2. Виконати дії аналогічні п. 1 для стовпців.
3. Знайти рядок, у якому рівно один нуль. Обрамити його квадратом та назвати позначеним. У стовпці, де стоїть цей нуль, усі інші нулі закреслити й надалі не розглядати.
4. Повторити п. 3, якщо це можливо.
5. Виконати дії аналогічні п. 3 для стовпців.
6. Повторити п. 5, якщо це можливо.
7. Якщо залишилися непозначені нулі, позначити будь-який із них, а в рядку та стовпці, де він стоїть, усі інші нулі закреслити.
8. Якщо кожен рядок і кожен стовпчик містять по одному нулю, розв’язок знайдено. В іншому випадку через усі позначені нулі провести мінімальну кількість вертикальних і горизонтальних прямих, що перетинаються.
9. Серед незакреслених прямими чисел знайти найменше. Відняти його від усіх незакреслених чисел та додати до всіх чисел, що знаходяться на перетинах прямих.
10. З отриманою матрицею виконати перетворення починаючи з п. 3.

**Приклад роботи**

10 14 11 15

3 20 19 17

7 15 6 9

13 8 6 9

0 4 1 5

0 17 16 14

1 9 0 3

7 2 0 3

0 2 1 2

0 15 16 11

1 7 0 0

7 0 0 0

[0] 2 1 2

x 15 16 11

1 7 0 0

7 0 0 0

[0] 2 1 2

x 15 16 11

1 7 [0] x

7 [0] x x

[0] 2 1 2

x 15 16 11

1 7 [0] x

7 [0] x x

[0] 2 1 2

x 15 16 11

{1} {7} {[0]} {x}

{7} {[0]} {x} {x}

{[0]} 2 1 2

{x} 15 16 11

{1} {7} {[0]} {x}

{7} {[0]} {x} {x}

{[0]} 1 0 1

{x} 14 15 10

{2} {7} {[0]} {x}

{8} {[0]} {x} {x}

0 1 0 1

0 14 15 10

2 7 0 0

8 0 0 0

x 1 [0] 1

[0] 14 15 10

2 7 x [0]

8 [0] x x

x 1 [0] 1

[0] 14 15 10

2 7 x [0]

8 [0] x x

x 1 [0] 1

[0] 14 15 10

2 7 x [0]

8 [0] x x

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

**Лістинг**

def print\_matrix(matrix):  
 for row in matrix:  
 for elem in row:  
 print("% 5s " % str(elem), end="")  
 print()  
 print()  
  
  
*# Step 0: initial*def convert\_to\_string(lst):  
 res = []  
 for elem in lst:  
 res.append(str(elem))  
 return res  
  
  
*# Steps 1 & 2: subtraction of least values from rows and columns*def convert\_matrix\_to\_string(matrix):  
 res = []  
 for row in matrix:  
 res.append(convert\_to\_string(row))  
 return res  
  
  
def convert\_to\_int(lst):  
 res = []  
 for elem in lst:  
 res.append(int(elem))  
 return res  
  
  
def fin\_min\_in\_row\_or\_column(lst):  
 return min(convert\_to\_int(lst))  
  
  
def get\_column(matrix, j):  
 column = []  
 for i in range(0, len(matrix)):  
 column.append(matrix[i][j])  
 return column

def subtract\_from\_rows(matrix):  
 for i in range(0, len(matrix)):  
 min\_elem = fin\_min\_in\_row\_or\_column(matrix[i])  
 for j in range(0, len(matrix[i])):  
 matrix[i][j] = str(int(matrix[i][j]) - min\_elem)  
 return matrix  
  
  
def subtract\_from\_columns(matrix):  
 for j in range(0, len(matrix[0])):  
 min\_elem = fin\_min\_in\_row\_or\_column(get\_column(matrix, j))  
 for i in range(0, len(matrix)):  
 matrix[i][j] = str(int(matrix[i][j]) - min\_elem)  
 return matrix  
  
  
*# Steps 3 & 4: marking one zero in each row and column and cross out others*def cycled\_inc(k, dim):  
 if k == dim - 1:  
 return 0  
 return k + 1

def contains\_one\_zero(lst, marked=False):  
 res = -1  
 for i in range(0, len(lst)):  
 if (lst[i] == "0" or marked and lst[i] == "[0]") and res == -1:  
 res = i  
 elif lst[i] == "0" or marked and lst[i] == "[0]":  
 return -1  
 return res  
  
  
def cross\_zeros\_in\_column(matrix, j, i\_of\_zero):  
 for i in range(0, len(matrix)):  
 if matrix[i][j] == "0" and i != i\_of\_zero:  
 matrix[i][j] = "x"  
 return matrix  
  
  
def mark\_zeros\_in\_rows(matrix):  
 row\_dim = len(matrix)  
 need\_to\_search = row\_dim - 1  
 i = 0  
 while True:  
 j\_of\_zero = contains\_one\_zero(matrix[i])  
 if j\_of\_zero != -1:  
 matrix[i][j\_of\_zero] = "[0]"  
 matrix = cross\_zeros\_in\_column(matrix, j\_of\_zero, i)  
 i = cycled\_inc(i, row\_dim)  
 need\_to\_search += 1  
 elif need\_to\_search == 0:  
 break  
 else:  
 need\_to\_search -= 1  
 i = cycled\_inc(i, row\_dim)  
 return matrix

*# I know about DRY, but let's not complicate things now*def cross\_zeros\_in\_row(matrix, i, j\_of\_zero):  
 for j in range(0, len(matrix[0])):  
 if matrix[i][j] == "0" and j != j\_of\_zero:  
 matrix[i][j] = "x"  
 return matrix  
  
  
def mark\_zeros\_in\_columns(matrix):  
 col\_dim = len(matrix[0])  
 need\_to\_search = col\_dim - 1  
 j = 0  
 while True:  
 i\_of\_zero = contains\_one\_zero(get\_column(matrix, j))  
 if i\_of\_zero != -1:  
 matrix[i\_of\_zero][j] = "[0]"  
 matrix = cross\_zeros\_in\_row(matrix, i\_of\_zero, j)  
 j = cycled\_inc(j, col\_dim)  
 need\_to\_search += 1  
 elif need\_to\_search == 0:  
 break  
 else:  
 need\_to\_search -= 1  
 j = cycled\_inc(j, col\_dim)  
 return matrix  
  
*# Step 5: mark one unmarked zero, if there are any, and cross out neighbouring ones*def mark\_any\_zero(matrix):  
 for i in range(0, len(matrix)):  
 for j in range(0, len(matrix[0])):  
 if matrix[i][j] == "0":  
 matrix[i][j] = "[0]"  
 matrix = cross\_zeros\_in\_column(matrix, j, i)  
 matrix = cross\_zeros\_in\_row(matrix, i, j)  
 return matrix  
  
  
*# Steps 6 & 7: stroke lines and rows, containing marked and crossed zeros*def has\_marked\_and\_crossed(lst):  
 has\_marked = False  
 has\_crossed = False  
 for elem in lst:  
 if elem == "[0]":  
 has\_marked = True  
 elif elem == "x":  
 has\_crossed = True  
 return has\_marked and has\_crossed  
  
  
def is\_stroked(elem):  
 return elem[0] == "{" and elem[-1] == "}"  
  
  
def stroke(elem):  
 return "{" + elem + "}"  
  
def stroke\_rows(matrix):  
 for i in range(0, len(matrix)):  
 if has\_marked\_and\_crossed(matrix[i]):  
 for j in range(0, len(matrix[i])):  
 if not is\_stroked(matrix[i][j]):  
 matrix[i][j] = stroke(matrix[i][j])  
 return matrix  
  
  
def stroke\_columns(matrix):  
 for j in range(0, len(matrix[0])):  
 if has\_marked\_and\_crossed(get\_column(matrix, j)):  
 for i in range(0, len(matrix)):  
 if not is\_stroked(matrix[i][j]):  
 matrix[i][j] = stroke(matrix[i][j])  
 return matrix  
  
*# Step 8: subtract min value from not stroked numbers, add it to intersections of strokes*def find\_min\_in\_not\_stroked(matrix):  
 res = float('inf')  
 for row in matrix:  
 for elem in row:  
 if elem not in ["x", "[0]"] and not is\_stroked(elem) and int(elem) < res:  
 res = int(elem)  
 return res  
def is\_row\_or\_column\_stroked(lst):  
 res = True  
 for elem in lst:  
 res &= is\_stroked(elem)  
 return res  
  
  
def get\_stroked\_rows(matrix):  
 res = []  
 for i in range(0, len(matrix)):  
 if is\_row\_or\_column\_stroked(matrix[i]):  
 res.append(i)  
 return res  
  
  
def get\_stroked\_columns(matrix):  
 res = []  
 for j in range(0, len(matrix[0])):  
 if is\_row\_or\_column\_stroked(get\_column(matrix, j)):  
 res.append(j)  
 return res  
  
  
def remove\_stroke(elem):  
 return elem[1:-1]

def update\_partly\_stroked(matrix):  
 min\_not\_stoked = find\_min\_in\_not\_stroked(matrix)  
 stroked\_rows = get\_stroked\_rows(matrix)  
 stroked\_columns = get\_stroked\_columns(matrix)  
 for i in range(0, len(matrix)):  
 for j in range(0, len(matrix[0])):  
 if not is\_stroked(matrix[i][j]):  
 matrix[i][j] = str(int(matrix[i][j]) - min\_not\_stoked)  
 elif i in stroked\_rows and j in stroked\_columns:  
 matrix[i][j] = stroke(str(int(remove\_stroke(matrix[i][j])) + min\_not\_stoked))  
 return matrix  
  
  
*# Step 9: remove strokes, marks and crosses, and try to get result*def remove\_strokes\_in\_matrix(matrix):  
 for i in range(0, len(matrix)):  
 for j in range(0, len(matrix[0])):  
 if is\_stroked(matrix[i][j]):  
 matrix[i][j] = remove\_stroke(matrix[i][j])  
 return matrix  
  
  
def prettify\_null(elem):  
 if elem in ["[0]", "x"]:  
 return "0"  
 return elem  
  
  
def prettify\_matrix(matrix):  
 pretty = remove\_strokes\_in\_matrix(matrix)  
 for i in range(0, len(pretty)):  
 for j in range(0, len(pretty[0])):  
 pretty[i][j] = prettify\_null(pretty[i][j])  
 return pretty  
*# Result: matrix containing single 1 in each row and columns*def have\_result(matrix):  
 for i in range(0, len(matrix)):  
 zero\_index = contains\_one\_zero(matrix[i], marked=True)  
 if zero\_index == -1:  
 return False  
 for j in range(0, len(matrix[0])):  
 zero\_index = contains\_one\_zero(get\_column(matrix, j), marked=True)  
 if zero\_index == -1:  
 return False  
 return True  
  
  
def create\_result(matrix):  
 res = []  
 for row in matrix:  
 new\_row = []  
 for elem in row:  
 if elem == "[0]":  
 new\_row.append(1)  
 else:  
 new\_row.append(0)  
 res.append(new\_row)  
 return res

def hungarian\_algorithm(matrix, debug=False):  
 def maybe\_print():  
 if debug:  
 print\_matrix(matrix)  
 steps\_012 = [convert\_matrix\_to\_string, subtract\_from\_rows, subtract\_from\_columns]  
 for f in steps\_012:  
 f(matrix)  
 maybe\_print()  
 while True:  
 steps\_345 = [mark\_zeros\_in\_rows, mark\_zeros\_in\_columns, mark\_any\_zero]  
 for f in steps\_345:  
 matrix = f(matrix)  
 maybe\_print()  
 if have\_result(matrix):  
 return create\_result(matrix)  
 steps\_6789 = [stroke\_rows, stroke\_columns, update\_partly\_stroked, prettify\_matrix]  
 for f in steps\_6789:  
 matrix = f(matrix)  
 maybe\_print()  
  
  
test\_matrix4 = [[10, 14, 11, 15],  
 [3, 20, 19, 17],  
 [7, 15, 6, 9],  
 [13, 8, 6, 9]]  
print\_matrix(hungarian\_algorithm(test\_matrix4, True))