Тверской Государственный технический университет

Кафедра: Программного обеспечения и вычислительной

техники

Лабораторная работа № 5

Выполнил: Студент второго курса

Группы Б.ПИН.РИС 18.05

Федотов Ярослав Всеволодович

Тверь 2019

**Постановка задачи**

Реализовать стандартный алгоритм перемножения матриц на с. 307 для матриц A и B. Учесть ограничения алгоритма.

**Алгоритм решения**

Функция matrix\_multiply(a,b):

columns = int(a.shape[1])

rows = int(b.shape[0])

c = np.zeros((rows, columns))

Для i in range(len(a)):

Для j in range(len(b[0])):

if(c[i][j] == 0):

Для k in range(len(b)):

c[i][j] += a[i][k] \* b[k][j]

вернуть c

**Текст программы**

if(self.ui.radioButtonStayAlgoMultiptication.isChecked()):

          if(columns != rows):

            msg = QtWidgets.QMessageBox()

            msg.setWindowTitle("Ошибка!")

            msg.setText("Умножить нельзя")

            msg.setIcon(msg.Warning)

            msg.exec()

          else:

              C = Matrix\_Multiply(A,B)

          self.ui.plainTextEditResultMultiplicationMatrix.setPlainText(str(C))

def Matrix\_Multiply(A,B):

    columns = int(A.shape[1])

    rows = int(B.shape[0])

    C = np.zeros((rows, columns))

    for i in range(len(A)):

        for j in range(len(B[0])):

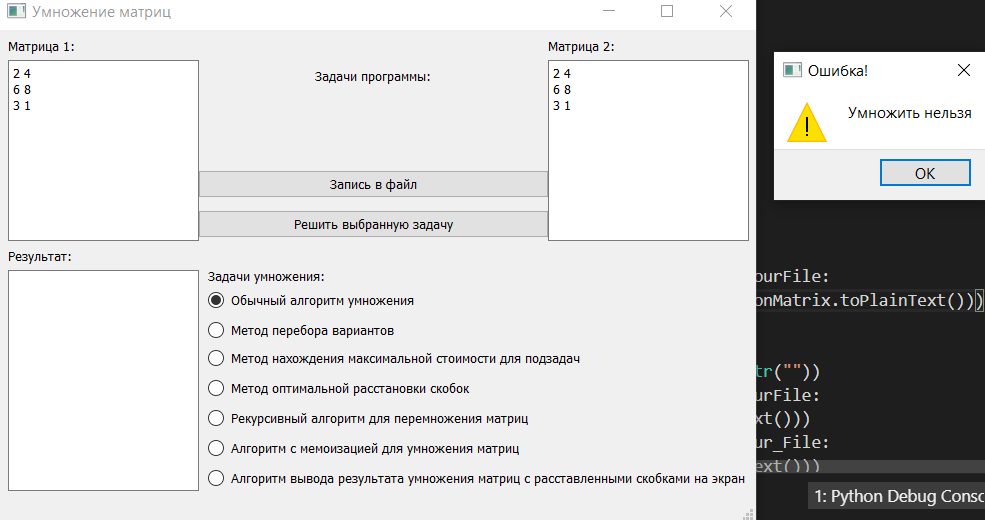
                if(C[i][j] == 0):

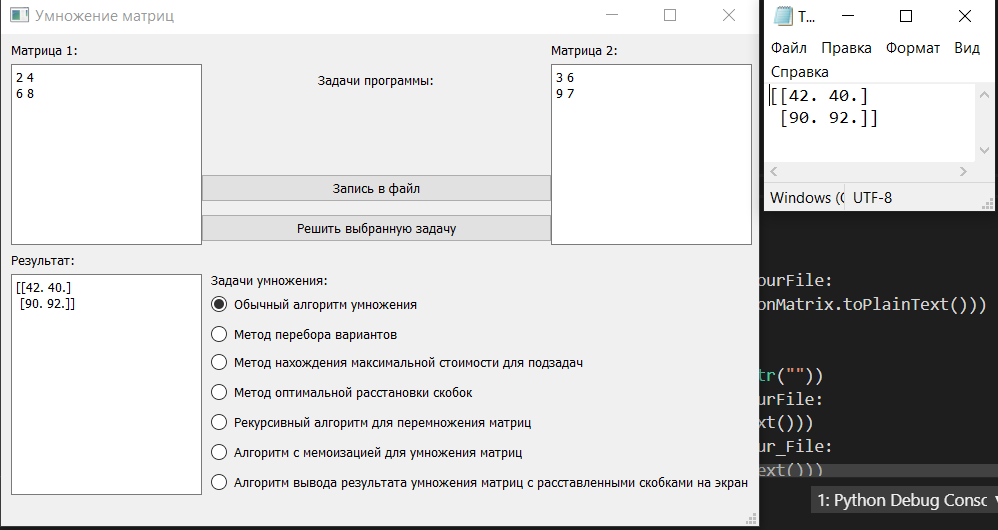
                    for k in range(len(B)):

                        C[i][j] += A[i][k] \* B[k][j]

    return C

**Скриншоты выполнения программы**





Реализовать задачу умножения матриц методом перебора вариантов.

**Алгоритм решения**

Для i in range(len(a)):

Для j in range(len(b[0])):

if(c[i][j] == 0):

Для k in range(len(b)):

c[i][j] += a[i][k] \* b[k][j]

self.ui.plaintexteditresultmultiplicationmatrix.setplaintext(str(c))

**Текст программы**

if(self.ui.radioButtonPerebor.isChecked()):

            for i in range(len(A)):

                for j in range(len(B[0])):

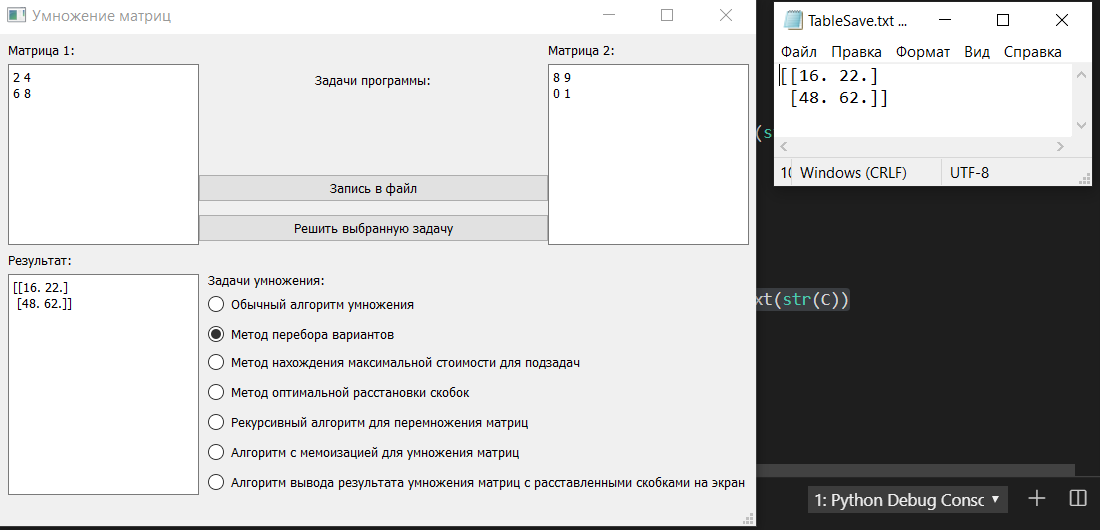
                    if(C[i][j] == 0):

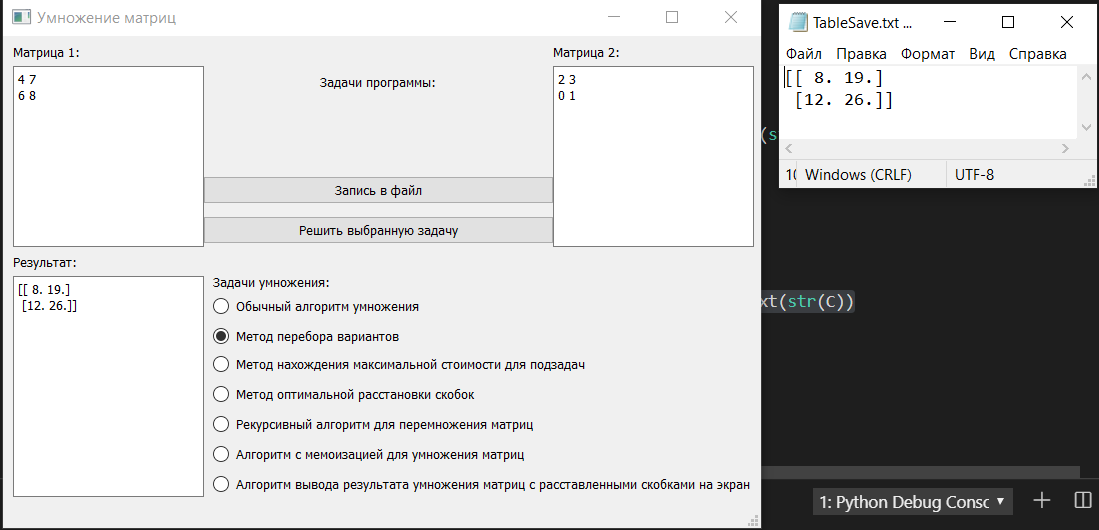
                        for k in range(len(B)):

                            C[i][j] += A[i][k] \* B[k][j]

            self.ui.plainTextEditResultMultiplicationMatrix.setPlainText(str(C))

**Скриншоты выполнения программы**





Реализовать алгоритм поиска оптимального числа умножений заданных матриц (нахождения максимальной стоимости для подзадач) (с. 310).

**Алгоритм решения**

if(self.ui.radiobuttonmaxbuypodtasks.ischecked()):

n = len(a)

Для i in range(n):

c[i][i] = 0

Для l in range(2,n):

Для i in range(1,n-l+1):

j=i+l-1

c[i][j] = 10000

Для k in range(i,j-1):

q = c[i][k]+c[k+1][j]

if(q < c[i][j]):

c[i][j] = q

b[i][j] = k

Для i in range(len(a)):

Для j in range(len(b[0])):

if(c[i][j] == 0):

Для k in range(len(b)):

c[i][j] += a[i][k] \* b[k][j]

self.ui.plaintexteditresultmultiplicationmatrix.appendplaintext(str(c))

self.ui.plaintexteditresultmultiplicationmatrix.appendplaintext("\n"+str(b))

**Текст программы**

 if(self.ui.radioButtonMaxBuyPodTasks.isChecked()):

            n = len(A)

            for i in range(n):

                C[i][i] = 0

            for l in range(2,n):

                for i in range(1,n-l+1):

                    j=i+l-1

                    C[i][j] = 10000

                    for k in range(i,j-1):

                        q = C[i][k]+C[k+1][j]

                        if(q < C[i][j]):

                            C[i][j] = q

                            B[i][j] = k

            for i in range(len(A)):

                for j in range(len(B[0])):

                    if(C[i][j] == 0):

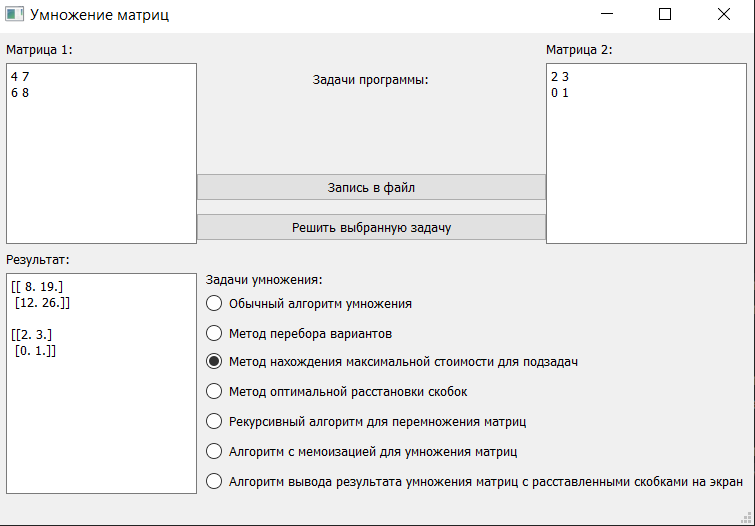
                        for k in range(len(B)):

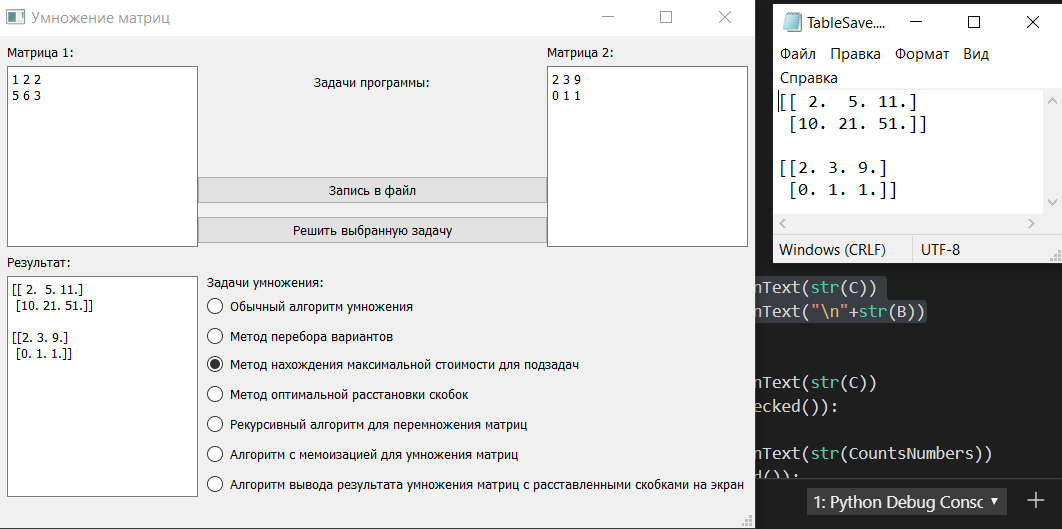
                            C[i][j] += A[i][k] \* B[k][j]

            self.ui.plainTextEditResultMultiplicationMatrix.appendPlainText(str(C))

            self.ui.plainTextEditResultMultiplicationMatrix.appendPlainText("\n"+str(B))

**Скриншоты выполнения программы**





Реализовать алгоритм оптимальной расстановки скобок на с. 312.

**Алгоритм решения**

Функция matrix\_chain\_multiply(a, s, i, j):

Если j > i:

x = matrix\_chain\_multiply(a, s, i, s[i][j])

y = matrix\_chain\_multiply(a, s, s[i][j]+1, j)

вернуть matrix\_multiply(x,y)

else:

вернуть a

**Текст программы**

 if(self.ui.radioButtonOptimalRastBackets.isChecked()):

            C = Matrix\_Chain\_Multiply(A,B,0,0)

            self.ui.plainTextEditResultMultiplicationMatrix.appendPlainText(str(C))

def Matrix\_Chain\_Multiply(A, s, i, j):

        if j > i:

            X = Matrix\_Chain\_Multiply(A, s, i, s[i][j])

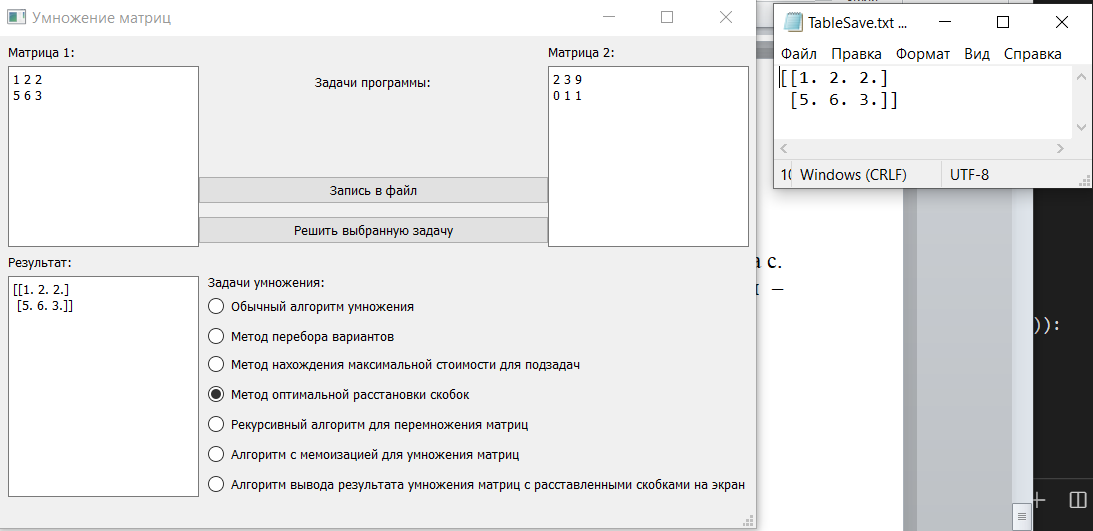
            Y = Matrix\_Chain\_Multiply(A, s, s[i][j]+1, j)

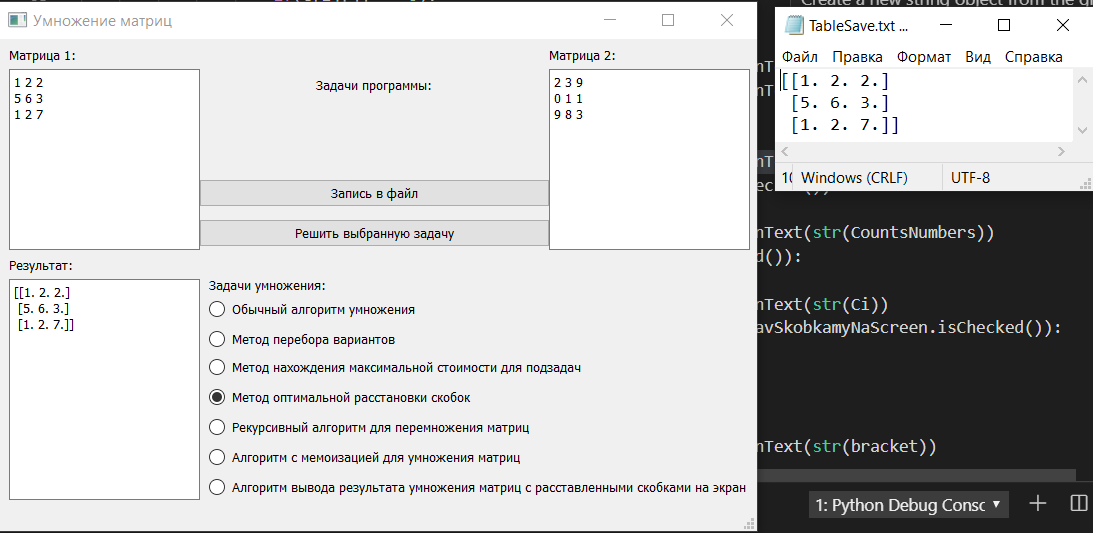
            return Matrix\_Multiply(X,Y)

        else:

            return A

**Скриншоты выполнения программы**





Реализовать рекурсивный алгоритм для перемножения матриц, разобранный на с. 313-315. Ответить на вопрос: «Что эффективней по времени выполнения – рекурсивный алгоритм или метод перебора вариантов?»

**Алгоритм решения**

Функция recursive\_matrix\_chain(m1,m2):

s=0 #сумма

t=[] #временная матрица

m3=[] # конечная матрица

Если len(m2)!=len(m1[0]):

msg = qtwidgets.qmessagebox()

msg.setwindowtitle("ошибка!")

msg.settext("умножить нельзя")

msg.seticon(msg.warning)

msg.exec()

else:

r1=len(m1) #количество строк в первой матрице

c1=len(m1[0]) #количество столбцов в 1

r2=c1 #и строк во 2ой матрице

c2=len(m2[0]) # количество столбцов во 2ой матрице

Для z in range(0,r1):

Для j in range(0,c2):

Для i in range(0,c1):

s=s+m1[z][i]\*m2[i][j]

t.append(s)

s=0

m3.append(t)

t=[]

вернуть m3

**Текст программы**

if(self.ui.radioButtonRekyrsyvniyAlgoMultiplicationMatrix.isChecked()):

            CountsNumbers = Recursive\_Matrix\_Chain(A,B)

            self.ui.plainTextEditResultMultiplicationMatrix.appendPlainText(str(CountsNumbers))

def Recursive\_Matrix\_Chain(m1,m2):

    s=0     #сумма

    t=[]    #временная матрица

    m3=[] # конечная матрица

    if len(m2)!=len(m1[0]):

         msg = QtWidgets.QMessageBox()

         msg.setWindowTitle("Ошибка!")

         msg.setText("Умножить нельзя")

         msg.setIcon(msg.Warning)

         msg.exec()

    else:

        r1=len(m1) #количество строк в первой матрице

        c1=len(m1[0]) #Количество столбцов в 1

        r2=c1           #и строк во 2ой матрице

        c2=len(m2[0])  # количество столбцов во 2ой матрице

        for z in range(0,r1):

            for j in range(0,c2):

                for i in range(0,c1):

                   s=s+m1[z][i]\*m2[i][j]

                t.append(s)

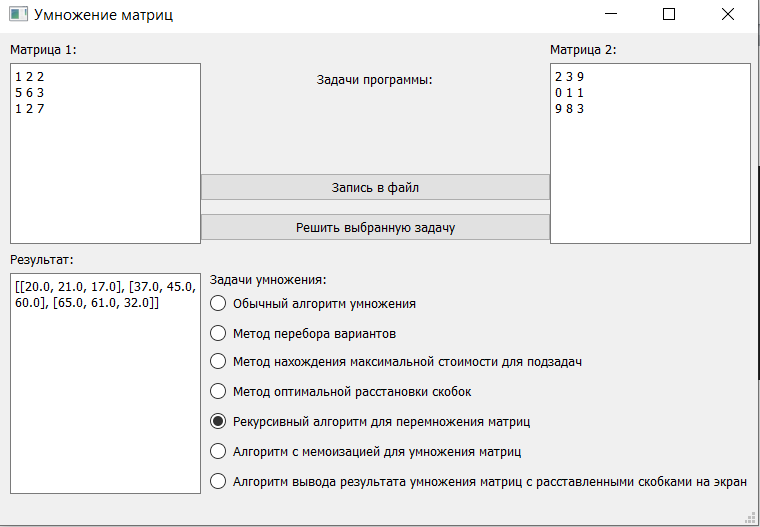
                s=0

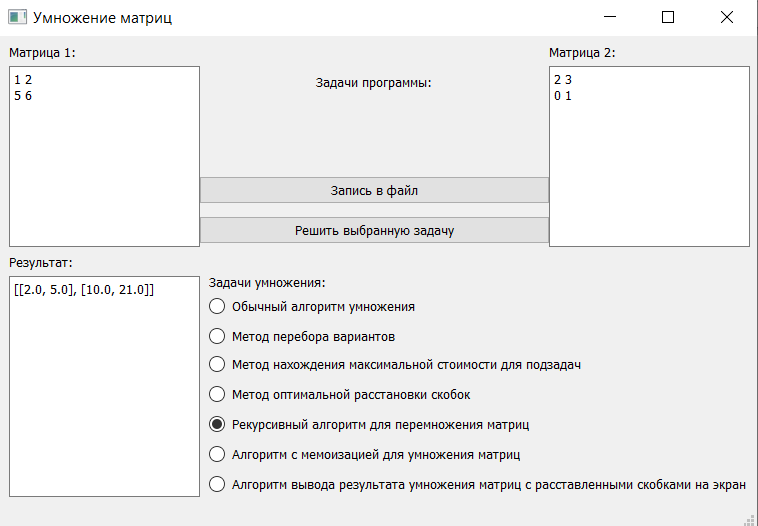
            m3.append(t)

            t=[]

    return m3

**Скриншоты выполнения программы**





Реализовать алгоритм с мемоизацией для умножения матриц, разобранный на с. 316-317.

**Алгоритм решения**

Функция memomization\_matrix(a,b):

res =[[0for x in range(a.shape[1])] Для y in range(b.shape[1])]

Для i in range(len(a)):

Для j in range(len(b[0])):

Для k in range(len(b)):

# resulted matrix

res[i][j] +=a[i][k] \*b[k][j]

вернуть res

**Текст программы**

if(self.ui.radioButtonMemoizationMultiplicationMatrix.isChecked()):

            Ci = Memomization\_Matrix(A,B)

            self.ui.plainTextEditResultMultiplicationMatrix.appendPlainText(str(Ci))

def Memomization\_Matrix(A,B):

    res =[[0for x in range(A.shape[1])] for y in range(B.shape[1])]

    for i in range(len(A)):

        for j in range(len(B[0])):

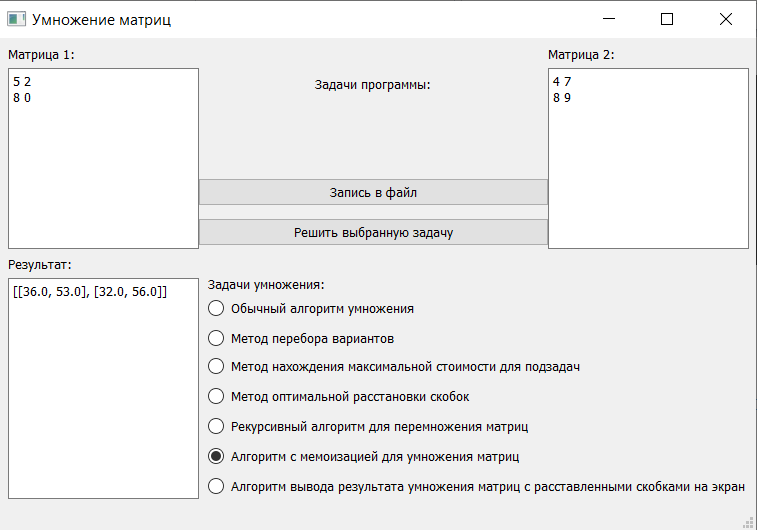
            for k in range(len(B)):

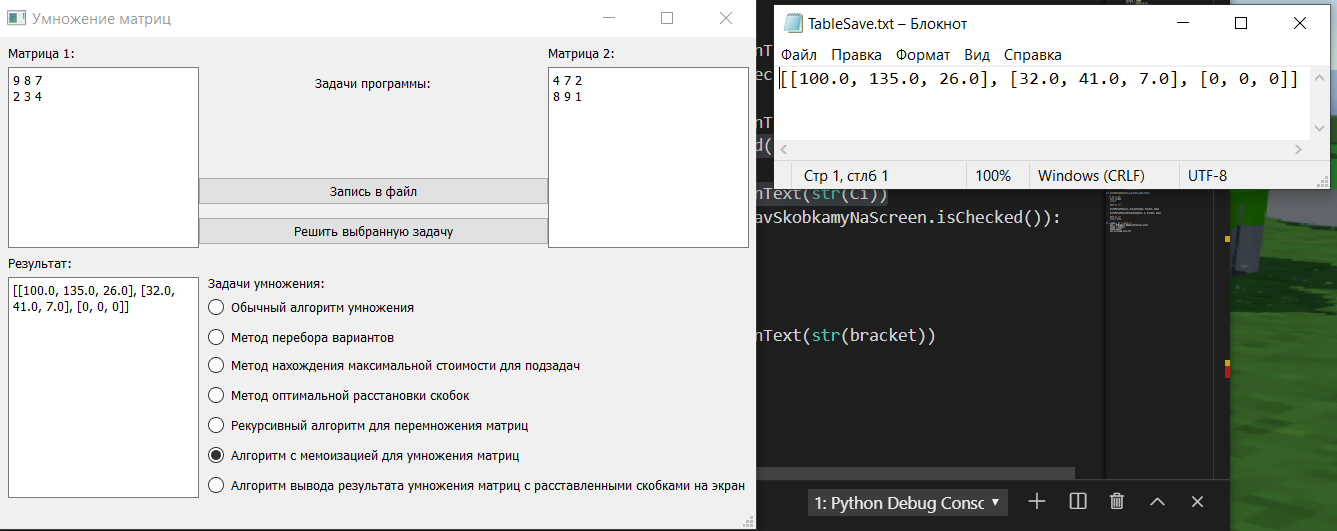
                # resulted matrix

                res[i][j] +=A[i][k] \*B[k][j]

    return res

**Скриншоты выполнения программы**





Разработать алгоритм вывода результата умножения матриц с расставленными скобками на экран или в файл.

**Алгоритм решения**

Функция matrixchainorder(p,n):

m = np.zeros((n,n))

bracket = np.zeros((n,n))

Для i in range(1,n):

m[i][i] = 0

Для l in range(2,n):

Для i in range(1,n-l+1):

j = i+l-1

m[i][j] = 100000

Для k in range(i,j-1):

q = m[i][k] + m[k+1][j] + p[i-1]\*p[k]\*p[j]

if(q < m[i][j]):

m[i][j] = q

bracket[i][j] = k

Функция printparenthesis(i,j,bracket,name,strl):

Если (i == j):

strl +=name

вернуть

strl += "("

printparenthesis(i, bracket[i][j], bracket, name)

printparenthesis(bracket[i][j]+1, j, bracket, name)

strl += ")"

вернуть strl

**Текст программы**

if(self.ui.radioButtonAlgoInputResultMultiplicationMatrixRasstavSkobkamyNaScreen.isChecked()):

            n = len(A)

            bracket = str("")

            matrixChainOrder(A,n)

            bracket = printParenthesis(1,n-1,A,'A',bracket)

            self.ui.plainTextEditResultMultiplicationMatrix.appendPlainText(str(bracket))

def matrixChainOrder(p,n):

    m = np.zeros((n,n))

    bracket = np.zeros((n,n))

    for i in range(1,n):

        m[i][i] = 0

    for L in range(2,n):

        for i in range(1,n-L+1):

            j = i+L-1

            m[i][j] = 100000

            for k in range(i,j-1):

                q = m[i][k] + m[k+1][j] + p[i-1]\*p[k]\*p[j]

                if(q < m[i][j]):

                    m[i][j] = q

                    bracket[i][j] = k

def printParenthesis(i,j,bracket,name,strl):

   if (i == j):

    strl +=name

    return

    strl += "("

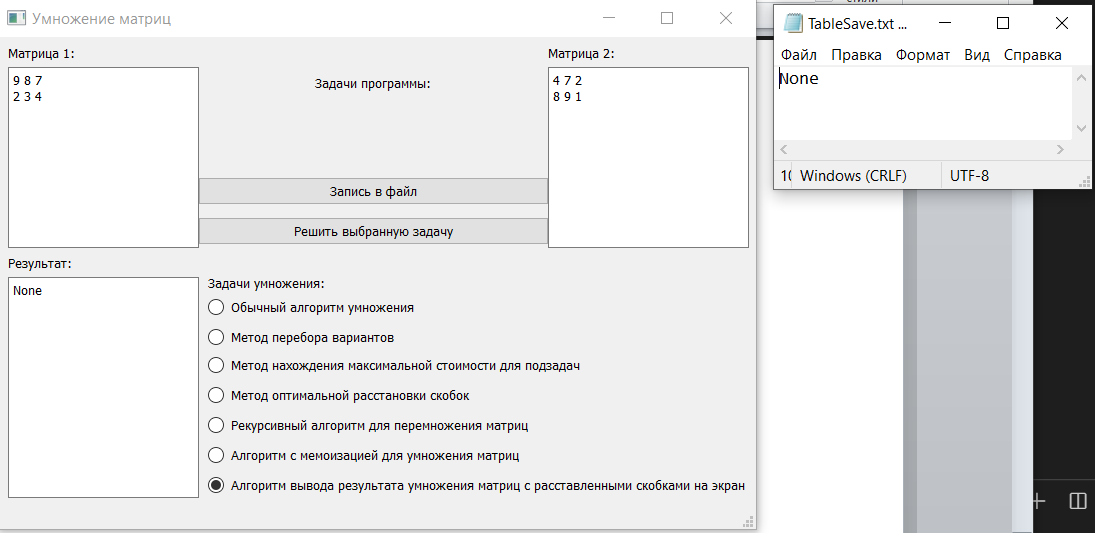
    printParenthesis(i, bracket[i][j], bracket, name)

    printParenthesis(bracket[i][j]+1, j, bracket, name)

    strl += ")"

    return strl

**Скриншоты выполнения программы**



**Ответы на вопросы**

Почему данная задача относиться к динамическому программированию, тут всё предельно просто, так как эта задача использует циклы в цикле в своем решении ,которые выполняют одну и тоже под задачу, а также вариации задачи(последующие типы решение этой задачи, но разными способами): использование рекурсии вызов функции из функций, сохранение в файл через поток W – чтение и запись (если файл создан) все эти факторы указывают на то, что данная задача относиться к задачам динамического программирования.

Чем отличаются методы решения задачи снизу вверх от сверху вниз.

Отличие состоит лишь в количестве методов, которые будут использоваться для решения задачи.