МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №6

по дисциплине: Исследование операций

тема: Нахождение седловой точки в смешанных стратегиях для матричной игры с нулевой суммой

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович
Проверил:
Вирченко Юрий Петрович

Белгород 2024 г.

Цель работы: Освоить методы нахождения седловой точки в смешанных стратегиях с помощью построения пары двойственных задач ЛП.

Задания

1. Изучить основные понятия теории матричных игр двух игроков с нулевой суммой, анализ игры в чистых стратегиях, понятие смешанной стратегии и седловой точки в смешанных стратегиях, а также метод нахождения седловой точки в смешанных стратегиях с помощью построения пары двойственных задач ЛП.

2. Составить и отладить программу для нахождения седловой точки игры с помощью решения пары симметрично двойственных задач ЛП.

3. Для подготовки тестовых данных решить вручную одну из следующую задачу.

3.

$$\begin{pmatrix} 8 & 5 & 7 & 6 \\ 9 & 8 & 10 & 7 \\ 12 & 6 & 4 & 3 \\ 7 & 13 & 5 & 2 \end{pmatrix}$$

Ручной расчет

Рассмотрим игру двух лиц, интересы которых противоположны. Такие игры называют *антагонистическими играми* двух лиц. В этом случае выигрыш одного игрока равен проигрышу второго, и можно описать только одного из игроков.

Предполагается, что каждый игрок может выбрать только одно из конечного множества своих действий. Выбор действия называют *выбором стратегии* игрока.

Если каждый из игроков выбрал свою стратегию, то эту пару стратегий называют *ситуацией игры.* Следует заметить, каждый игрок знает, какую стратегию выбрал его противник, т.е. имеет *полную информацию* о результате выбора противника.

Чистой стратегией игрока I является выбор одной из n строк матрицы выигрышей A, а чистой стратегией игрока II является выбор одного из столбцов этой же матрицы.

1. Проверяем, имеет ли платежная матрица седловую точку.

Если да, то выписываем решение игры в чистых стратегиях.

Считаем, что игрок I выбирает свою стратегию так, чтобы получить максимальный свой выигрыш, а игрок II выбирает свою стратегию так, чтобы минимизировать выигрыш игрока I.

Игроки	B_1	B_2	B_3	B_4	$a = \min(A_i)$
A_1	8	5	7	6	5
A_2	9	8	10	7	7
A_3	12	6	4	3	3
A_4	7	13	5	2	2
$b = \max(B_i)$	12	13	10	7	

Находим гарантированный выигрыш, определяемый нижней ценой игры $a = \max(a_i) = 7$, которая указывает на максимальную чистую стратегию A_2.Верхняя цена игры $b = \min(b_j) = 7$.

Седловая точка (2, 4) указывает решение на пару альтернатив (A2,B4).Цена игры равна 7.

2. Проверяем платежную матрицу на доминирующие строки и доминирующие столбцы.

Иногда на основании простого рассмотрения матрицы игры можно сказать, что некоторые чистые стратегии могут войти в оптимальнуюсмешанную стратегию лишь с нулевой вероятностью.

Говорят, что *i-я* стратегия 1-го игрока доминирует его *k-ю* стратегию, еслиa$_{ij}$ ≥ a$_{kj}$ для всех *j Э N* и хотя бы для одного *j* $a_{ij} > a_{kj}$. В этом случае говорят также, что *i-я* стратегия (или строка) – доминирующая, *k-я* – доминируемая.

Говорят, что *j-я* стратегия 2-го игрока доминирует его *l-ю* стратегию, еслидля всех *j Э M* $a_{ij} \le a_{il}$ и хотя бы для одного i $a_{ij} < a_{il}$. В этом случае *j-*

ю стратегию (столбец) называют доминирующей, *l-ю* – доминируемой. Стратегия A_2 доминирует над стратегией A_1 (все элементы строки 2 больше или равны значениям 1-ой строки), следовательно, исключаем 1-ую строку матрицы. Вероятность $p_1 = 0$.

9	8	10	7
12	6	4	3
7	13	5	2

С позиции проигрышей игрока В стратегия B_4 доминирует над стратегией B_1 (все элементы столбца 4 меньше элементов столбца 1), следовательно, исключаем 1-й столбец матрицы. Вероятность $q_1 = 0$.

С позиции проигрышей игрока В стратегия B_4 доминирует над стратегией B_2 (все элементы столбца 4 меньше элементов столбца 2), следовательно, исключаем 2-й столбец матрицы. Вероятность $q_2 = 0$.

10	7
4	3
5	2

Стратегия A_2 доминирует над стратегией A_3 (все элементы строки 2 больше или равны значениям 3-ой строки), следовательно, исключаем 3-ую строку матрицы. Вероятность $p_3 = 0$.

10	7
5	2

Мы свели игру 4 x 4 к игре 2 x 2.

3. Находим решение игры в смешанных стратегиях.

Математические модели пары двойственных задач линейного программирования можно записать так: найти минимум функции F(x) при ограничениях (для игрока II):

$10x_1+5x_2 \geq 1$

$7x_1+2x_2 \geq 1$

$F(x) = x_1+x_2 \rightarrow \min$

найти максимум функции Z(y) при ограничениях (для игрока I):$10y_1+7y_2 \leq 1$

$5y_1+2y_2 \leq 1$

$Z(y) = y_1+y_2 \rightarrow \max$

Решим прямую задачу линейного программирования симплексныметодом, с использованием симплексной таблицы.

Определим максимальное значение целевой функции $Z(Y) = y_1+y_2$ при следующих условиях - ограничений.

$10y_1+7y_2\leq1$

$5y_1+2y_2\leq1$

Для построения первого опорного плана систему неравенств приведем к системе уравнений путем введения дополнительных переменных (*переходк канонической форме*).

$10y_1+7y_2+y_3 = 1$

$5y_1+2y_2+y_4 = 1$

Решим систему уравнений относительно базисных переменных: y_3, y_4 Полагая, что *свободные переменные* равны 0, получим первый опорныйплан:

Y0 = (0,0,1,1)

Базис	B	y_1	y_2	y_3	y_4
y_3	1	10	7	1	0
y_4	1	5	2	0	1
Z(Y0)	0	-1	-1	0	0

Переходим к основному алгоритму симплекс-метода.

Итерация №0.

Текущий опорный план неоптимален, так как в индексной строке находятся отрицательные коэффициенты.

В качестве ведущего выберем столбец, соответствующий переменной y2, так как это наибольший коэффициент по модулю.

Вычислим значения Di по строкам как частное от деления: bi / ai2 и из них выберем наименьшее:

min (1 : 7 , 1 : 2) = $^1/_7$

Следовательно, 1-ая строка является ведущей.

Разрешающий элемент равен (7) и находится на пересечении ведущегостолбца и ведущей строки.

Базис	B	y_1	y_2	y_3	y_4	min
y_3	1	10	7	1	0	1/7
y_4	1	5	2	0	1	1/2
Z(Y1)	0	-1	-1	0	0	

Формируем следующую часть симплексной таблицы. Вместо переменнойy_3 в план 1 войдет переменная y_2.

Получаем новую симплекс-таблицу:

Базис	B	y_1	y_2	y_3	y_4
y_2	1/7	10/7	1	1/7	0
y_4	5/7	15/7	0	-2/7	1
Z(Y1)	1/7	3/7	0	1/7	0

Конец итераций: индексная строка не содержит отрицательных элементов - найден оптимальный план

Среди значений индексной строки нет отрицательных. Поэтому эта таблица определяет оптимальный план задачи.

Окончательный вариант симплекс-таблицы:

Базис	B	y_1	y_2	y_3	y_4
y_2	1/7	10/7	1	1/7	0
y_4	5/7	15/7	0	-2/7	1
Z(Y2)	1/7	3/7	0	1/7	0

Оптимальный план можно записать так:

$y_1 = 0$, $y_2 = {}^1/_7$

$Z(Y) = 1*0 + 1*{}^1/_7 = {}^1/_7$

Используя последнюю итерацию прямой задачи найдем, оптимальныйплан двойственной задачи.

$x_1={}^1/_7$, $x_2=0$

Это же решение можно получить, применив теоремы двойственности.Из теоремы двойственности следует, что $X = C*A^{-1}$.

Составим матрицу A из компонентов векторов, входящих в оптимальныйбазис.

$A = (A_2, A_4) =$

7	0
2	1

Определив обратную матрицу $D = A^{-1}$ через алгебраические дополнения,получим:

$D = A^{-1} =$

1/7	0
-2/7	1

Как видно из последнего плана симплексной таблицы, обратная матрицаA^{-1} расположена в столбцах дополнительных переменных.

Тогда $X = C*A^{-1}$ =

$$(1, 0) \times \begin{array}{|c|c|} \hline 1/7 & 0 \\ \hline -2/7 & 1 \\ \hline \end{array} = (1/7;0)$$

Оптимальный план двойственной задачи равен:

$x_1 = {}^1/_7$, $x_2 = 0$

$F(X) = 1*{}^1/_7+1*0 = {}^1/_7$

Цена игры будет равна $g = 1/F(x)$, а вероятности применения стратегийигроков:

$q_i = g*y_i$; $p_i = g*x_i$.

Цена игры: $g = 1 : {}^1/_7 = 7p_1 = 7*{}^1/_7 = 1$

$p_2 = 7*0 = 0$

Оптимальная смешанная стратегия игрока I: P = (1; 0)

$q_1 = 7*0 = 0$

$q_2 = 7*{}^1/_7 = 1$

Оптимальная смешанная стратегия игрока II: Q = (0; 1)Цена игры: v=7

4. Проверим правильность решения игры с помощью **критерия оптимальности стратегии**.

$\sum a_{ij}q_j \leq v$

$\sum a_{ij}p_i \geq v$

$M(P_1;Q) = (10*0) + (7*1) = 7 = v$

$M(P_2;Q) = (5*0) + (2*1) = 2 \leq v$

$M(P;Q_1) = (10*1) + (5*0) = 10 \geq v$

$M(P;Q_2) = (7*1) + (2*0) = 7 = v$

Все неравенства выполняются как равенства или строгие неравенства, следовательно, решение игры найдено верно.

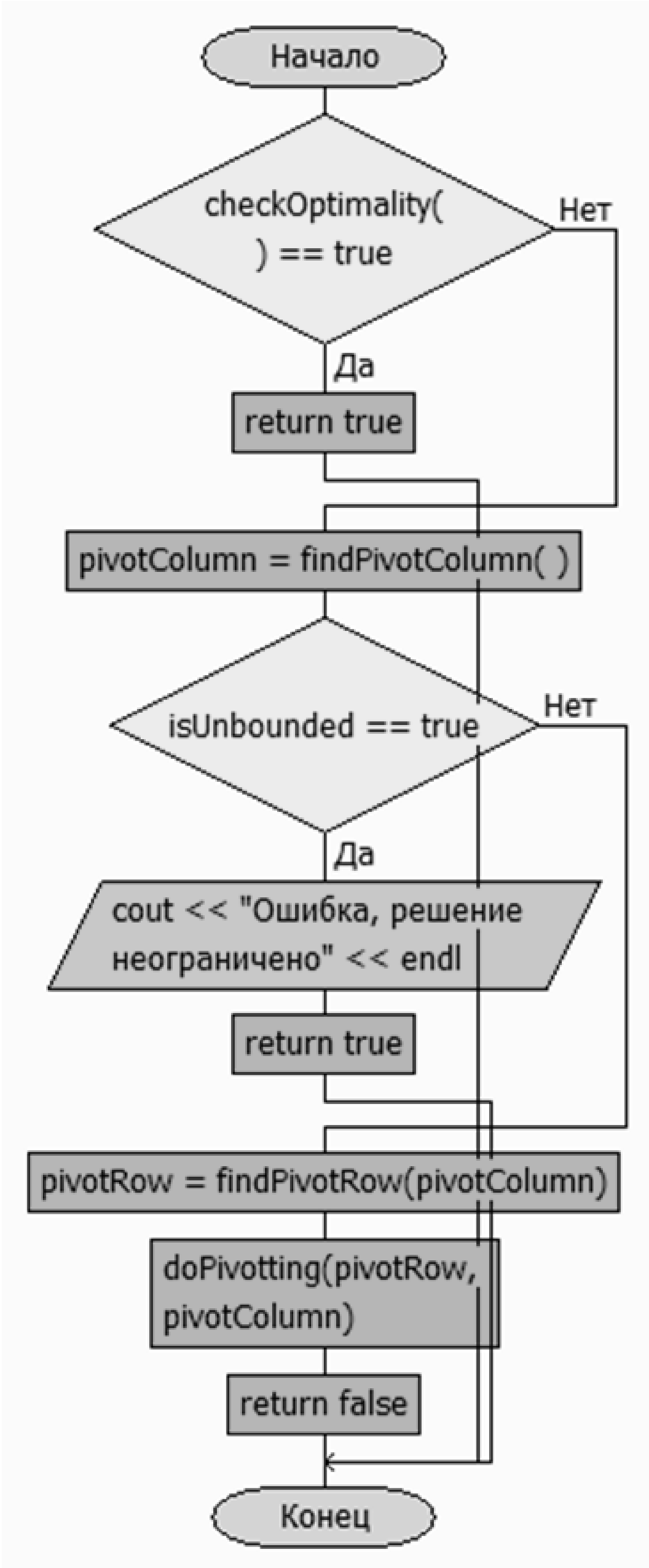
Поскольку из исходной матрицы были удалены строки и столбцы, то найденные векторы вероятности можно записать в виде:

P(0,1,0,0)

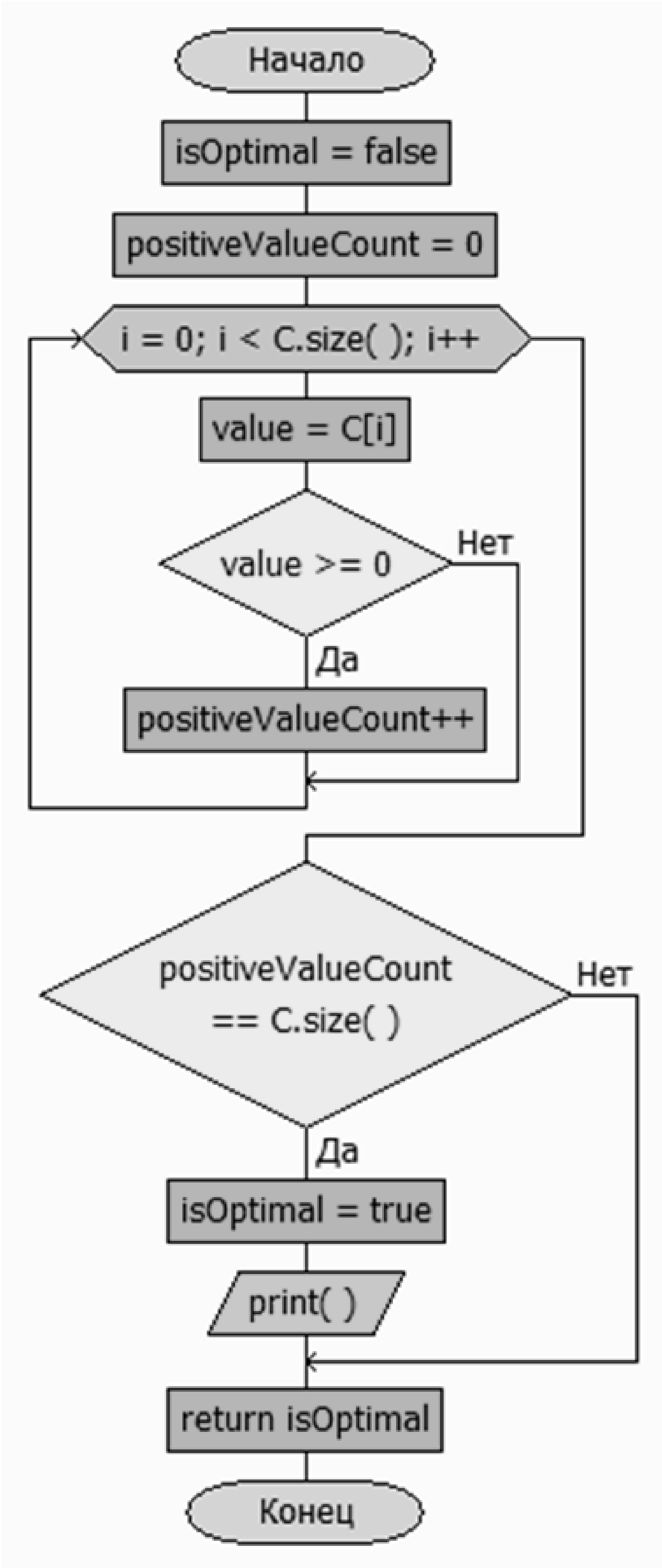
Q(0,0,0,1)

Блок – схемы:

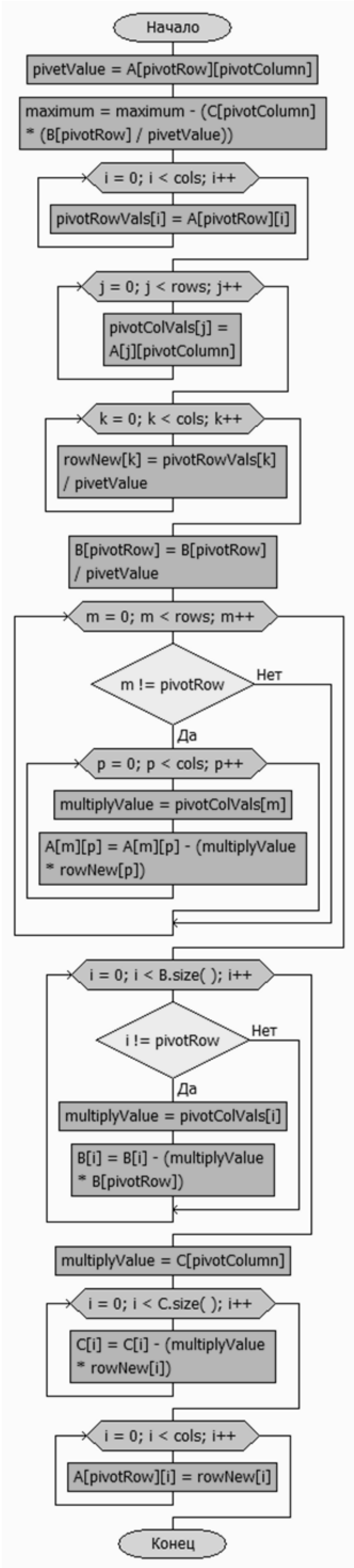
Функция simplexAlgorithmCalculataion



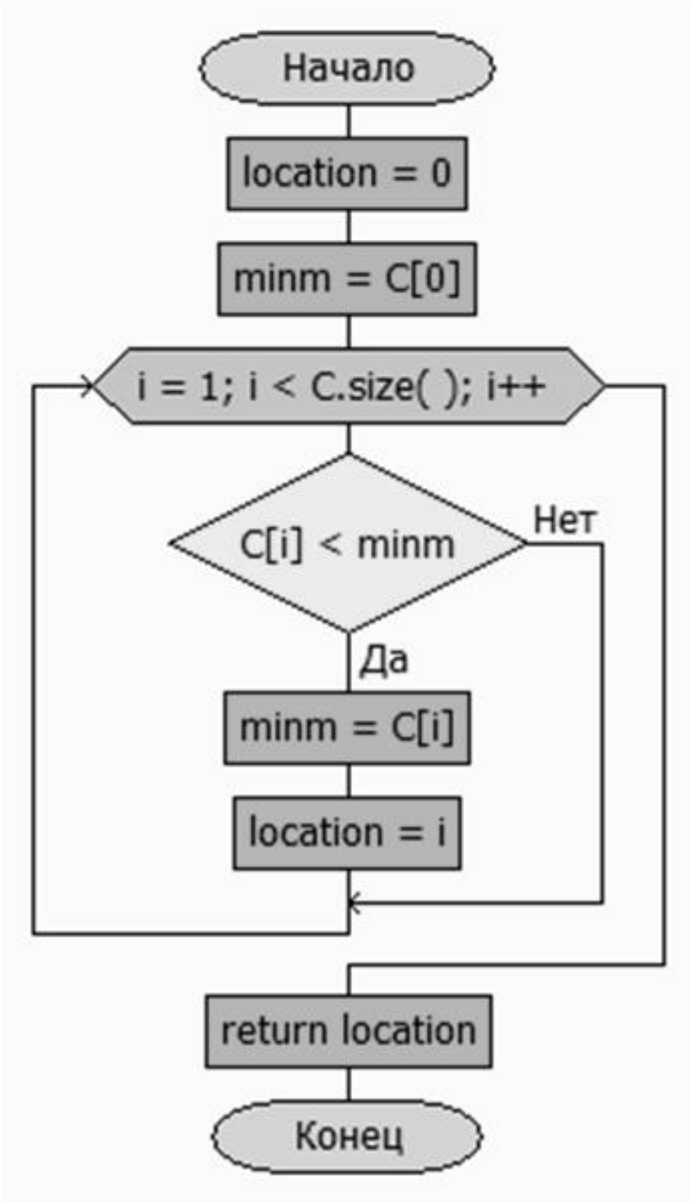
Функция CheckOptimality:



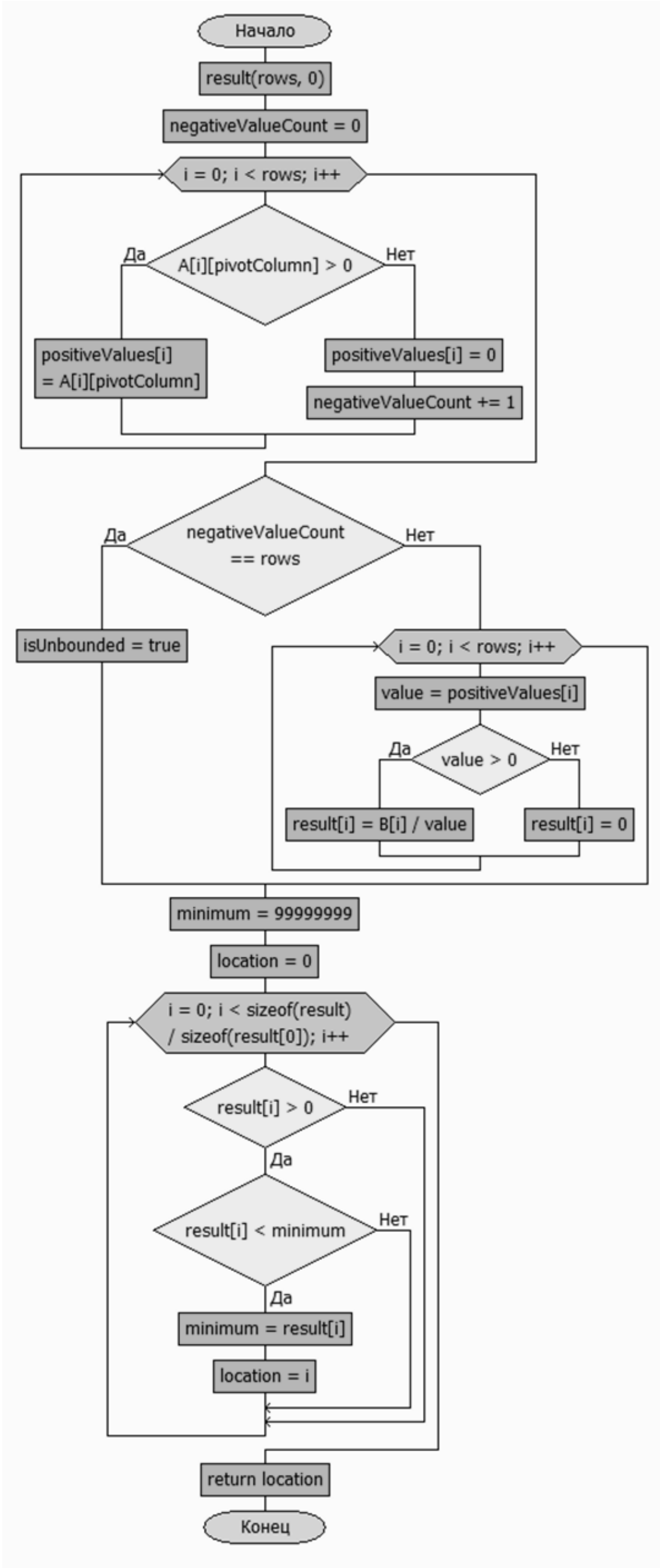
Функция doPivotting:



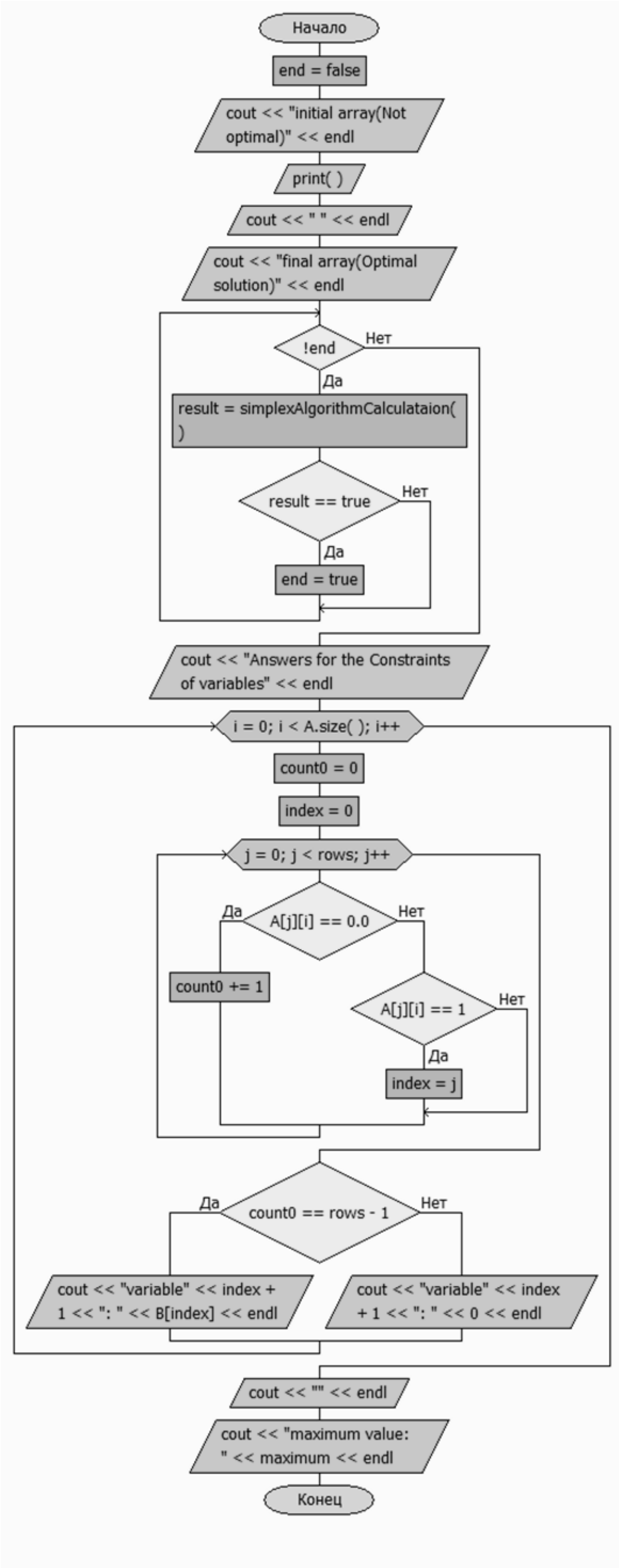
Функция findPivotColumn:



Функция findPivotRow:



Функция CalculateSimplex:



Код программы:

```
class Simplex:
    def __init__(self, matrix, b, c):  # Инициализация класса и его
атрибутов
        self.maximum = 0  # Инициализация максимального значения
        self.isUnbounded = False  # Инициализация флага неограниченности
        self.rows = len(matrix)  # Количество строк в матрице
        self.cols = len(matrix[0])  # Количество столбцов в матрице
        self.A = [row[:] for row in matrix]  # Копирование матрицы
        self.B = b[:]  # Копирование массива констант ограничений
        self.C = c[:]  # Копирование массива коэффициентов целевой функции

    def simplexAlgorithmCalculataion(self):  # Функция для выполнения
итерации метода симплекс
        if self.checkOptimality():  # Проверка на оптимальность таблицы
            return True
        pivotColumn = self.findPivotColumn()  # Находим столбец для
опорного элемента
        if self.isUnbounded:  # Проверяем на условие неограниченности
            print("Error unbounded")  # Вывод сообщения об ошибке
            return True
        pivotRow = self.findPivotRow(pivotColumn)  # Находим строку для
опорного элемента
        self.doPivotting(pivotRow, pivotColumn)  # Производим обновление
таблицы
        return False

    def checkOptimality(self):  # Функция для проверки оптимальности
таблицы
        isOptimal = False  # Инициализация флага оптимальности
        positiveValueCount = sum(1 for value in self.C if value >= 0)  #
Подсчет положительных значений в массиве C
        if positiveValueCount == len(self.C):  # Если все коэффициенты
неотрицательны, то таблица оптимальна
            isOptimal = True  # Устанавливаем флаг оптимальности
            self.print()  # Вывод текущего состояния таблицы
        return isOptimal  # Возвращаем флаг оптимальности

    def doPivotting(self, pivotRow, pivotColumn):  # Функция для обновления
таблицы после выбора опорного элемента
        pivotValue = self.A[pivotRow][pivotColumn]  # Получаем значение
опорного элемента
        pivotRowVals = self.A[pivotRow][:]  # Копируем значения строки с
опорным элементом
        pivotColVals = [row[pivotColumn] for row in self.A]  # Получаем
значения столбца с опорным элементом
        rowNew = [val / pivotValue for val in pivotRowVals]  # Обновляем
значения обновленной строки
        self.maximum -= self.C[pivotColumn] * (self.B[pivotRow] /
pivotValue)  # Обновляем максимальное значение
        for i in range(self.cols):  # Обновляем значения обновленной строки
            self.A[pivotRow][i] = rowNew[i]
        self.B[pivotRow] /= pivotValue  # Обновляем значение вектора B
        for i in range(self.rows):  # Обновляем остальные значения в
матрице A
            if i != pivotRow:
                multiplyValue = pivotColVals[i]
                self.B[i] -= multiplyValue * self.B[pivotRow]
```

```
                for p in range(self.cols):
                    self.A[i][p] -= multiplyValue * rowNew[p]
        multiplyValue = self.C[pivotColumn]  # Получаем коэффициент для
обновления вектора C
        for i in range(self.cols):  # Обновляем значения вектора C
            self.C[i] -= multiplyValue * rowNew[i]

    def print(self):  # Функция для вывода текущего состояния таблицы
        for row in self.A:  # Цикл по строкам
            print(' '.join(str(val) for val in row))  # Вывод элемента
матрицы
        print()

    def findPivotColumn(self):  # Функция для поиска столбца с наименьшим
коэффициентом целевой функции
        location = 0  # Переменная для хранения индекса столбца
        minm = self.C[0]  # Переменная для хранения минимального
коэффициента
        for i in range(1, len(self.C)):  # Цикл по коэффициентам целевой
функции
            if self.C[i] < minm:  # Поиск минимального коэффициента
                minm = self.C[i]
                location = i
        return location  # Возвращаем индекс столбца

    def findPivotRow(self, pivotColumn):  # Функция для поиска строки с
опорным элементом
        positiveValues = [row[pivotColumn] if row[pivotColumn] > 0 else 0
for row in
                          self.A]  # Массив для хранения положительных
значений
        result = [self.B[i] / positiveValues[i] if positiveValues[i] > 0
else 0 for i in
                  range(self.rows)]  # Вектор для хранения результатов
        if all(val == 0 for val in positiveValues):  # Проверка условия
неограниченности
            self.isUnbounded = True
        else:
            minimum = min((val, i) for i, val in enumerate(result) if val >
0)[
                1]  # Поиск минимального значения в векторе результатов
            return minimum  # Возвращаем индекс строки с опорным элементом

    def CalculateSimplex(self):  # Функция для выполнения метода симплекс
        end = False  # Флаг завершения алгоритма
        print("Первый опорный план:")  # Вывод сообщения о начальном
состоянии таблицы
        self.print()  # Вывод начального состояния таблицы
        print()
        print("Оптимальный план:")  # Вывод сообщения о конечном состоянии
таблицы
        while not end:  # Цикл выполнения итераций алгоритма
            result = self.simplexAlgorithmCalculataion()  # Выполняем
итерацию алгоритма
            if result:  # Проверяем флаг окончания алгоритма
                end = True  # Завершаем цикл
        print("Базисные переменные:")  # Выводим результаты
        for i in range(len(self.A[0])):  # Цикл по строкам матрицы
            count0 = 0  # Счётчик нулевых элементов
            index = 0  # Индекс для переменной
            for j in range(self.rows):  # Цикл по строкам
```

```
                if self.A[j][i] == 0.0:  # Проверка на ноль
                    count0 += 1
                elif self.A[j][i] == 1:  # Проверка на единицу
                    index = j  # Получаем индекс переменной
            if count0 == self.rows - 1:  # Проверка на количество нулевых
элементов
                print(f"x{index + 1}: {self.B[index]}")  # Выводим значение
переменной
            else:
                print(f"x{index + 1}: 0")  # Выводим ноль
        print()
        print(f"Значение целевой функции: {self.maximum}")  # Выводим
максимальное значение целевой функции

colSizeA = 4 # Задаём размер столбцов матрицы А
rowSizeA = 2 # Задаём размер строк матрицы А

C = [-1, -1, 0, 0]

simplex = Simplex(a, B, C)
simplex.CalculateSimplex()
```

Результат работы программы:

```
Первый опорный план:
10 7 1 0
5 2 0 1



Оптимальный план:
1.4285714285714286 1.0 0.14285714285714288 0.0
2.142857142857143 0.0 -0.2857142857142857 1.0

Базисные переменные:
x1: 0
x1: 0.14285714285714288
x1: 0
x2: 0.7142857142857143

Значение целевой функции: 0.14285714285714288

Индексы седловых точек: 2 4
Цена игры: 7

Process finished with exit code 0
```

Вывод: в рамках лабораторной работы, студентам предстоит погрузиться в теорию игр и изучить основы смешанных стратегий. Они будут формализовывать игровую ситуацию в виде Линейной Программы и разрабатывать соответствующую двойственную задачу. С использованием методов Линейного Программирования они исследуют стратегические возможности и точно определят седловую точку в рассматриваемых ситуациях. Получение практических навыков в решении таких игровых задач будет полезно студентам для применения усвоенных методов в различных областях, где требуется анализ и оптимизация стратегий.