МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

**Факультет информационных технологий и робототехники (ФИТР)**

**Отчёт по лабораторной работе №4**

По дисциплине: «Методы и алгоритмы принятия решений»

На тему: «РЕШЕНИЕ МАТРИЧНЫХ ИГР»

Вариант 9

**Выполнил:**  студент группы 10701118 ВоробейИ.А.

**Приняла ст. преподаватель:** Борисова И.М.

Минск 2020

**Цель работы:** изучить способы решения матричной игры в чистых стратегиях, используя принцип минимакса, а также решения игр в смешанных стратегиях с помощью ЗЛП.

**Постановка задачи:**

1.Разработать приложение для решения матричной игры в чистых стратегиях, используя принципы минимакса.



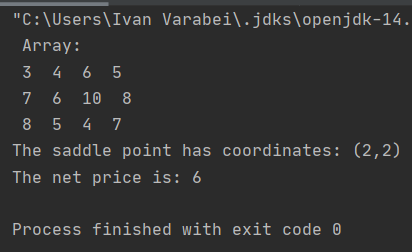
2.Решеть матричную игру в смешанных стратегиях с помощью ЗЛП в EXCEL.

****

**Приложения для решения матричной игры в чистых стратегиях.**

Я разработал программу, которая находит minimax и maxmin и их положение в матрице, после чего сравнивает между собой. Если они равны, то выводятся координаты седловой точки и сама седловая точка.

Результат выполнения программы:



Седловая точка (2, 2) указывает решение на пару альтернатив (A2,B2).

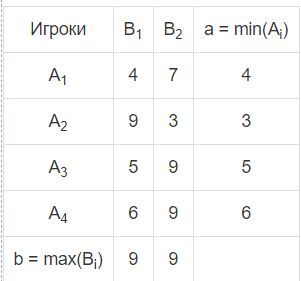
**Ответ:** Найденная седловая точка означает максимальный выигрыш первого игрока и минимальный проигрыш второго, а положение этого числа в платежной матрице – задействованные в ней стратегии первого и второго игроков.

Листинг программы представлен в конце отчета.

**Решить матричную игру в смешанных стратегиях с помощью ЗЛП.**

1. Проверяем, имеет ли платежная матрица седловую точку. Если да, то выписываем решение игры в чистых стратегиях.

Считаем, что игрок 1 выбирает свою стратегию так, чтобы получить максимальный свой выигрыш, а игрок 2 выбирает свою стратегию так, чтобы минимизировать выигрыш игрока 1.



1. Определяем гарантированный выигрыш, определяемый нижней ценой игры a = max(a) = 6, которая указывает на максимальную чистую стратегию А4. Верхняя цена игры b = min(b) = 9.

Что свидетельствует об отсутствии седловой точки, так как a ≠ b, тогда цена игры находится в пределах 6 ≤ y ≤ 9. Находим решение игры в смешанных стратегиях.

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

Стратегия A3 доминирует над стратегией A1 (все элементы строки 3 больше или равны значениям 1-ой строки), следовательно, исключаем 1-ую строку матрицы. Вероятность p1 = 0.

Стратегия A4 доминирует над стратегией A3 (все элементы строки 4 больше или равны значениям 3-ой строки), следовательно, исключаем 3-ую строку матрицы. Вероятность p3 = 0.



Cвели игру 4 x 2 к игре 2 x 2.

Так как игроки выбирают свои чистые стратегии случайным образом, то выигрыш игрока I будет случайной величиной. В этом случае игрок I должен выбрать свои смешанные стратегии так, чтобы получить максимальный средний выигрыш.

1. Находим решение игры в смешанных стратегиях

Запишем систему уравнений.

Для игрока I:

|  |
| --- |
| 9y1+3y2 ≤ 1  6y1+9y2 ≤ 1  Z(y) = y1+y2 → max |

Для игрока II :

|  |
| --- |
| 9x1+6x2 ≥ 1  3x1+9x2 ≥ 1  F(x) = x1+x2 → min |

1. Решаем прямую задачу линейного программирования симплекс методом.

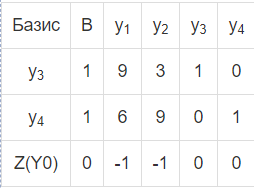
Определяем максимальное значение целевой функции:

|  |
| --- |
| Z(Y) = y1+y2 |

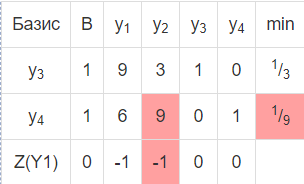
При ограничениях:

|  |
| --- |
| 9y1+3y2≤1  6y1+9y2≤1 |

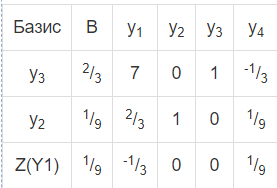
1. Для построения опорного плана систему неравенств приведем к системе уравнений путем введения дополнительных переменных и решим систему.
2. Первый опорный план



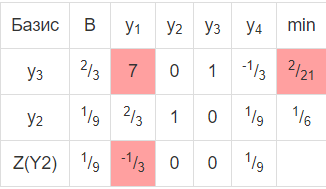
1. Так как текущий опорный план не оптимален, в качестве ведущего столбца выберем столбец y2, так как это наибольший коэффициент по модулю.



1. Получаем новую симплекс-таблицу.



1. Так как полученный план также не оптимален, повторно применяем пункт 8.



1. Формируем следующую часть симплексной таблицы. Вместо переменной y3 в план 2 войдет переменная y1.



1. Найден оптимальный план.

|  |
| --- |
| y1 = 2/21  y2 = 1/21  Z(y) = 1/7 |

1. Цена игры будет равна g= 1/Z(y), а вероятности применения стратегий:

|  |
| --- |
| qi = g\*yi; pi = g\*xi.  *Цена игры:* g = 1 : 1/7 = 7  p1 = 7\*1/21 = 1/3  p2 = 7\*2/21 = 2/3 |

1. Оптимальная смешанная стратегия игрока I:

|  |
| --- |
| P = (1/3; 2/3)  q1 = 7\*2/21 = 2/3  q2 = 7\*1/21 = 1/3 |

Оптимальная смешанная стратегия игрока II:

|  |
| --- |
| Q = (2/3; 1/3) |

Цена игры: v=7.

**Выводы:** в результате выполнения лабораторной работы разработана программа для решения матричной игры в чистых стратегиях, используя принципы минимакса, в результате выполнения которой выводится информация о стратегиях, верхней и нижней цене игры, а также чистая цена игры. А также решил матричную игру в смешанных стратегиях с помощью ЗЛП в EXCEL. Изначально убрав все заведомо ложные стратегии, сократив размер матрицы, далее воспользовавшись и найдя промежуток цены игры, вычислив верхнюю и нижнюю ее пределы. Далее, решая задачу в EXCEL, был выведен оптимальный план, найдены оптимальные стратегии для каждого игрока и цена игры.

**Листинг**

Calculator.java

public class Calculator {  
 public static void main(String[] args) {  
 System.*out*.println(" Array:");  
 int[][] table = {{3, 4, 6, 5},  
 {7, 6, 10, 8},  
 {8 ,5, 4, 7}};  
 int rowAmount = table.length;  
 int columnAmount = table[0].length;  
 int[] minA;  
 int[] minB;  
 minA = *searchA*(table, rowAmount, columnAmount);  
 minB = *searchB*(table, rowAmount, columnAmount);  
 for (int i = 0; i < rowAmount; i++) {  
 for (int j = 0; j < columnAmount; j++) {  
 System.*out*.print(" " + table[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
 System.*out*.println("The saddle point has coordinates: (" + (minA[1] + 1) + "," +  
 (minB[1] + 1) + ") \nThe net price is: " + minA[0]);  
 }  
  
 public static int[] searchA(int[][] array, int rowAmount , int columnAmount) {  
 int min = array[0][0];  
 int[] arr = new int[columnAmount];  
 int[] minCord = {0, 0};  
 for (int i = 0; i < rowAmount; i++) {  
 for (int j = 0; j < columnAmount; j++) {  
 if (array[i][j] < min && j + 1 != 4) {  
 min = array[i][j];  
 arr[i] = min;  
 }  
 if (j + 1 == 4 && i + 1 != rowAmount) {  
 arr[i] = min;  
 min = array[i + 1][0];  
 }  
 }  
 }  
 for (int i = 0; i < columnAmount; i++) {  
 if (arr[i] > min) {  
 min = arr[i];  
 minCord[0] = min;  
 minCord[1] = i;  
 }  
 }  
 return minCord;  
 }  
  
 public static int[] searchB(int[][] array, int rowAmount , int columnAmount) {  
 int max = array[0][0];  
 int[] arr = new int[columnAmount];  
 int[] minCord = {0, 0};  
 for (int j = 0; j < rowAmount; j++) {  
 for (int i = 0; i < columnAmount; i++) {  
 if (j != columnAmount && i != rowAmount && array[i][j] > max ) {  
 max = array[i][j];  
 arr[j] = max;  
 }  
 if (i + 1 == 4 && j + 1 != columnAmount) {  
 arr[j] = max;  
 max = array[0][j + 1];  
 } else if (i + 1 == columnAmount)  
 arr[j] = max;  
 }  
 }  
 for (int i = 0; i < columnAmount; i++) {  
 if (arr[i] < max) {  
 max = arr[i];  
 minCord[0] = max;  
 minCord[1] = i;  
 }  
 }  
 return minCord;  
 }  
}