Лабораторная работа №3

по дисциплине «Структура и алгоритмы и обработки данных»

на тему:

«Методы поиска подстроки в строке»

Выполнили: студ. гр. БСТ1902

Козлов М. С.

Вариант №7

Москва 2021

1. Цель работы

Задание 1 Реализовать методы поиска подстроки в строке. Добавить возможность ввода строки и подстроки с клавиатуры. Предусмотреть возможность существования пробела. Реализовать возможность выбора опции чувствительности или нечувствительности к регистру. Оценить время работы каждого алгоритма поиска и сравнить его со временем работы стандартной функции поиска, используемой в выбранном языке программирования.

Задание 2 «Пятнашки» Игра в 15, пятнашки, такен — популярная головоломка, придуманная в 1878 году Ноем Чепмэном. Она представляет собой набор одинаковых квадратных костяшек с нанесёнными числами, заключённых в квадратную коробку. Длина стороны коробки в четыре раза больше длины стороны костяшек для набора из 15 элементов, соответственно в коробке остаётся незаполненным одно квадратное поле. Цель игры — перемещая костяшки по коробке, добиться упорядочи

Ход выполнения лабораторной работы

1.1 Задание 1

Код программы:

Кнута-Морриса-Пратта

```
    public class KMP
    {
        public static int FindSubstring(string pattern, string text, bool
checkCase = true)
        {
            if (!checkCase)
            {
                pattern = pattern.ToLower();
                text = text.ToLower();
            }

            int[] prefix = GetPrefix(pattern);
            int index = 0;
```

```
            for (int i = 0; i < text.Length; i++)
            {
                while (index > 0 && pattern[index] != text[i])
                {
                    index = prefix[index - 1];
                }

                if (pattern[index] == text[i]) index++;
                if (index == pattern.Length)
                {
                    return i - index + 1;
                }
            }

            return -1;
        }

        private static int[] GetPrefix(string text)
        {
            var result = new int[text.Length];
            result[0] = 0;
            int index = 0;

            for (int i = 1; i < text.Length; i++)
            {
                while (index >= 0 && text[index] != text[i])
                {
                    index--;
                }
                index++;
                result[i] = index;
            }

            return result;
        }
    }
```

Упрощенный Бойера-Мура

```
   public class BoyerMoore
    {
        public static int FindSubstring(string pattern, string text, bool
checkCase = true)
        {
            if (!checkCase)
            {
                pattern = pattern.ToLower();
                text = text.ToLower();
            }

            var offsetTable = new int[256];

            if (pattern.Length > text.Length)
            {
                return -1;
            }

            for (int t = 0; t < offsetTable.Length; t++)
            {
                offsetTable[t] = pattern.Length;
```

```
            }

            for (int t = 0; t < pattern.Length - 1; t++)
            {
                offsetTable[pattern[t]] = pattern.Length - t - 1;
            }

            int i = pattern.Length - 1;
            int j = i;
            int k = i;

            while (j >= 0 && i <= text.Length - 1 )
            {
                j = pattern.Length - 1;
                k = i;

                while (j >= 0 && text[k] == pattern[j])
                {
                    k--;
                    j--;
                }

                i += (char)offsetTable[text[i]];
            }

            if (k >= text.Length - pattern.Length)
            {
                return -1;
            }

            return k + 1;
        }
    }
```

Результат работы программы для строки длинной 100000

```
Ответ: 94056
KMP - результат: 94056; время поиска: 1,316 мс.
BoyerMoore - результат: 94056; время поиска: 0,5872 мс.
```

2.2 Задание 2

Код программы:

```
public class GameController
{
    public static void Run(Game start)
    {
        if (!ValidateGame(start))
        {
            Console.WriteLine("Нет решений");
            return;
        }
```

```
        var target = new Game(new[,]
        {
           {1, 2, 3, 4},
           {5, 6, 7, 8},
           {9, 10, 11, 12},
           {13, 14, 15, 0},
        });

        Dictionary<Game, Game> path = new Dictionary<Game, Game>();
        path[start] = null;
        var queue = new Queue<Game>();
        queue.Enqueue(start);
        while (queue.Count != 0)
        {
            var game = queue.Dequeue();

            var nextGames = game
                .AllAdjacentGames()
                .Where(g => !path.ContainsKey(g));
            foreach (var nextGame in nextGames)
            {
                path[nextGame] = game;
                queue.Enqueue(nextGame);
            }
            if (path.ContainsKey(target)) break;
        }
        while (target != null)
        {
            target.Print();
            target = path[target];
        }
    }

    private static bool ValidateGame(Game game)
    {
        int sum = 0;
        int position = 0;
        var array = new List<int>(16);

        foreach (int item in game.Data)
        {
            array.Add(item);
        }

        while (position != 16)
        {
            for (int i = position + 1; i < array.Count; i++)
            {
                if (array[i] < array[position] && array[i] != 0)
                {
                    sum++;
                }
            }

            position++;
        }

        sum += Array.FindIndex(array.ToArray(), (x) => (x == 0)) % 4;

        return sum % 2 == 0;
    }
}
```

```
    public class Game
    {
        class Point
        {
            public int X { get; set; }
            public int Y { get; set; }
        }

        const int Size = 4;
        public int[,] Data { get; set; }
        public Game(int[,] data)
        {
            Data = data;
        }
        public Game(Game original)
            : this((int[,])original.Data.Clone())
        {

        }

        IEnumerable<Point> Rectangle(int xmin, int xmax, int ymin, int ymax)
        {
            for (int x = xmin; x <= xmax; x++)
                for (int y = ymin; y <= ymax; y++)
                    yield return new Point { X = x, Y = y };
        }

        IEnumerable<Point> GamePoints
        {
            get
            {
                return Rectangle(0, Size - 1, 0, Size - 1);
            }
        }

        public Game Move(int dx, int dy)
        {
            var point = GamePoints
                .Where(p => Data[p.X, p.Y] == 0)
                .Where(p => p.X + dx >= 0 && p.X + dx < Size && p.Y + dy >= 0
&& p.Y + dy < Size)
                .FirstOrDefault();
            if (point == null) return null;

            var newGame = new Game(this);
            newGame.Data[point.X, point.Y] = Data[point.X + dx, point.Y +
dy];
            newGame.Data[point.X + dx, point.Y + dy] = Data[point.X,
point.Y];
            return newGame;
        }

        public IEnumerable<Game> AllAdjacentGames()
        {
            return Rectangle(-1, 1, -1, 1)
                .Where(point => point.X == 0 || point.Y == 0)
                .Select(point => Move(point.X, point.Y))
                .Where(game => game != null);
        }

        public override bool Equals(object obj)
        {
```

```
            var game = obj as Game;
            return GamePoints
                .All(point => Data[point.X, point.Y] == game.Data[point.X,
point.Y]);
        }

        public override int GetHashCode()
        {
            return GamePoints
                .Select(point => Data[point.X, point.Y])
                .Aggregate((sum, val) => sum * 97 + val);
        }

        public void Print()
        {
            var str = GamePoints
                .GroupBy(z => z.X)
                .Select(row => row
                .Select(point => Data[point.X, point.Y].ToString())
                .Aggregate((a, b) => a + " " + b))
                .Aggregate((a, b) => a + "\n" + b);
            Console.WriteLine(str);
            Console.WriteLine();
        }
    }
```

Результат работы программы