

Министерство цифрового развития связи и массовых коммуникаций РФ

Государственное бюджетное образовательное учреждение высшего
образования

Ордена Трудового Красного Знамени

«Московский технический университет связи и информатики»

Кафедра «Математическая кибернетика и информационные технологии»

дисциплина «Структуры и алгоритмы обработки данных»

Отчет по лабораторной работе №3

«Методы поиска подстроки в строке»

Подготовил: студент группы

БВТ1903 Саввин Д.И.

Проверил: Кутейников И.А.

Москва

2021

Оглавление.

1. ЦЕЛЬ РАБОТЫ	3
2. ВЫПОЛНЕНИЕ	4
3. ВЫВОД	14

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.

Алгоритмы:

1. Кнута-Морриса-Пратта
2. Упрощенный Бойера-Мура

Задание 2 «Пятнашки».

Задача: написать программу, определяющую, является ли данное расположение «решаемым», то есть можно ли из него за конечное число шагов перейти к правильному. Если это возможно, то необходимо найти хотя бы одно решение - последовательность движений, после которой числа будут расположены в правильном порядке.

Входные данные: массив чисел, представляющий собой расстановку в порядке «слева направо, сверху вниз». Число 0 обозначает пустое поле. Например, массив [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0] представляет собой «решенную» позицию элементов.

Выходные данные: если решения нет, то функция должна вернуть пустой массив []. Если решение есть, то необходимо представить решение — для каждого шага записывается номер передвигаемого на данном шаге элемента.

2. ВЫПОЛНЕНИЕ

Ниже представлен код методов поиска Кнута-Морриса-Пратта и Бойера-Мура.

Cnoot.py – метод Кнута-Морриса-Пратта.

```
# Класс алгоритма поиска КМП
class Cnoot:
    def __init__(self):
        self.reg = True

    # Метод поиска наибольшего суффикса во входящей строке
    def find_biggest_suffix(self, st):
        j = 0
        i = 1
        p = [0]*len(st)

        while i < len(st):
            if st[j] == st[i]:
                p[i] = j+1
                i += 1
                j += 1
            else:
                if j == 0:
                    p[i] = 0
                    i += 1
                else:
                    j = p[j-1]

        return p

    # Метод находит подстроку в строке и возвращает результат поиска
    def find_string(self, main_string, f_string):
        if self.reg == True:
            main_string = str.lower(main_string)
            f_string = str.lower(f_string)

        i = 0
        j = 0
        n = len(main_string)
        m = len(f_string)
        p = self.find_biggest_suffix(f_string)

        while i < n:
            if main_string[i] == f_string[j]:
                i += 1
                j += 1
                if j == m:
                    print('Образ найден.')
                    break
            else:
                if j > 0:
                    j = p[j-1]
                else:
                    i+=1

        if i == n:
            print('Образ не найден.')
            break
```

Boyer.py – метод Бойера-Мура.

```
# Класс алгоритма поиска Бойера-Мура
class Boyer:
    def __init__(self):
        self.reg = True

    # Метод записывающий все символы входящей строки в словарь
    def s_table(self, st):
        S = set()
        M = len(st)
        d = {}

        for i in range(M - 2, -1, -1): # итерации с предпоследнего символа
            if st[i] not in S: # если символ еще не добавлен в таблицу
                d[st[i]] = M - i - 1
                S.add(st[i])

        if st[M-1] not in S:
            d[st[M-1]] = M

        d['*'] = M
        return d

    # Метод находит подстроку в строке, возвращает результат поиска и индекс
    # вхождения
    def find_string(self, main_s, st):
        if self.reg == True:
            main_s = str.lower(main_s)
            st = str.lower(st)

        N = len(main_s)
        M = len(st)
        d = self.s_table(st)

        if N >= M:
            i = M-1

            while i < N:
                k = 0
                j = 0
                flBreak = False
                for j in range(M-1, -1, -1):
                    if main_s[i-k] != st[j]:
                        if j == M-1:
                            off = d[main_s[i]] if d.get(main_s[i], False)
                        else:
                            off = d[st[j]]

                        i += off
                        flBreak = True
                        break

                k += 1

                if not flBreak:
                    print("Исходная строка: \"\" + main_s + "\"\n\" + "Искомая
строка: \"\" + st + "\"\n\" + f"Образ найден поиндексу: {i-k+1}")
                    break
                else:
                    print('Образ не найден.')
            else:
                print('Образ не найден.')
```

На рисунках 1-4 представлены результаты тестирования обоих алгоритмов поиска.

```
main x
C:\Users\den4ik\PycharmProjects\SIA0D_3\venv\Scripts\python.exe C:/Users/den4ik/PycharmProjects/SIA0D_3/main.py
Методы поиска:
1. Кнута-Морриса-Пратта
2. Бойера-Мура
1
Учитывать регистр при поиске?
1. Да
2. Нет
2
Введите основную строку:
Р03 два три четыре пять
Введите подстроку:
четыре
Образ найден.

Process finished with exit code 0
```

Рис. 1 – Результат тестирования алгоритма поиска.

```
main x
C:\Users\den4ik\PycharmProjects\SIA0D_3\venv\Scripts\python.exe C:/Users/den4ik/PycharmProjects/SIA0D_3/main.py
Методы поиска:
1. Кнута-Морриса-Пратта
2. Бойера-Мура
1
Учитывать регистр при поиске?
1. Да
2. Нет
1
Введите основную строку:
Р03 два три четыре пять
Введите подстроку:
четыре
Образ не найден.

Process finished with exit code 0
|
```

Рис. 2 – Результат тестирования алгоритма поиска.

```
main x
C:\Users\den4ik\PycharmProjects\SIA0D_3\venv\Scripts\python.exe C:/Users/den4ik/PycharmProjects/SIA0D_3/main.py
Методы поиска:
1. Кнута-Морриса-Пратта
2. Бойера-Мура
2
Учитывать регистр при поиске?
1. Да
2. Нет
2
Введите основную строку:
Есть игра но нет правил
Введите подстроку:
нет
Исходная строка: "есть игра но нет правил"
Искомая строка: "нет"
Образ найден по индексу: 13

Process finished with exit code 0
```

Рис. 3 – Результат тестирования алгоритма поиска.

```
C:\Users\den4ik\PycharmProjects\SIA0D_3\venv\Scripts\python.exe C:/Users/den4ik/PycharmProjects/SIA0D_3/main.py
Методы поиска:
1. Кнута-Морриса-Пратта
2. Бойера-Мура
2
Учитывать регистр при поиске?
1. Да
2. Нет
2
Введите основную строку:
Есть игра но нет правил
Введите подстроку:
no
Образ не найден.

Process finished with exit code 0
|
```

Рис. 4 – Результат тестирования алгоритма поиска.

На рисунках ниже представлены результаты оценки времени работы алгоритмов КМП, БМ и стандартной функции поиска предусмотренным Python.

Задание 2.

Ниже представлен код скриптов a_star.py и solver15.py

a_star.py

```
from heapq import heappop, heappush

def a_star(start_chain, goal_node):
    move_history = []
    node_hash = {}
    chains_queue = []
    heappush(chains_queue, start_chain)
    while chains_queue:
        cur_chain = heappop(chains_queue)
        cur_node = cur_chain.last_node()
        print(str(cur_chain))
        if cur_node == goal_node:
            for i in range(0, len(cur_chain.history)-1):
                for j in range(0, 16):
                    if not (cur_chain.history[i].board_state[j] ==
cur_chain.history[i+1].board_state[j]):
                        if not (cur_chain.history[i+1].board_state[j] == 0):
move_history.append(cur_chain.history[i+1].board_state[j])
                            break
            return move_history
        node_hash[cur_node] = cur_chain.g()
        for chain in cur_chain.get_neighbours():
            if chain.last_node() in node_hash:
                if chain.g() >= node_hash[chain.last_node()]:
                    continue
                node_hash[chain.last_node()] = chain.g()
            heappush(chains_queue, chain)

    return []
```

solver15.py

```
import time

import a_star
from math import sqrt

def manh_dst_matrix(a, b, n):
    """Find manhattan distance between 'a' and 'b' in matrix of size 'n'
    """
    return abs(a % n - b % n) + abs(a // n - b // n)

class chain15:
    def __str__(self):
        i = 0
        sstr = ""
        while i < self.size ** 2:
            sstr += str(self.board_state[i]) + " "
            if i % self.size == 3:
                sstr += "\n"
            i += 1
```

```

        return sstr

    def __init__(self, board_state, history=[], last_moved=[]):
        self.board_state = list(board_state)
        self.size = int(sqrt(len(board_state)))
        self.history = history
        self.quad_size = int(self.size * self.size)
        self.last_moved = last_moved

    def manh_dst(self):
        dst = 0
        for i in range(0, self.quad_size):
            dst += manh_dst_matrix((self.board_state[i] - 1) %
self.quad_size, i, self.size)
        return int(dst)

    def last_node(self):
        """Must be hashable value (list not hashable :( )
        """
        return str(self.board_state)

    def linear_conflict(self):
        conflict_count = 0
        # ToDo some heuristic :)
        return 2 * conflict_count

    def last_move(self):
        if self.board_state[-1] == self.quad_size - 1 or self.board_state[-1]
== self.quad_size - self.size:
            return 0
        return 2

    def corner_tiles(self):
        conflict_count = 0
        # upper left corner
        if self.board_state[0] != 1:
            if self.board_state[1] == 2 or self.board_state[self.size] ==
self.size + 1:
                conflict_count += 1
        # upper right corner
        if self.board_state[self.size - 1] != self.size:
            if self.board_state[self.size - 2] == self.size - 1 or
self.board_state[self.size * 2 - 1] == self.size * 2:
                conflict_count += 1
        # lower left corner
        if self.board_state[self.quad_size - self.size] != self.quad_size -
self.size + 1:
            if self.board_state[self.quad_size - self.size * 2] ==
self.quad_size - self.size * 2 + 1 or \
                self.board_state[self.quad_size - self.size + 1] ==
self.quad_size - self.size + 2:
                conflict_count += 1

        return 2 * conflict_count

    def simple_heur(self):
        dst = 0
        for i in range(0, int(self.quad_size)):
            if (self.board_state[i] - 1) != i:
                dst += 1
        return dst

    def h(self):
        return self.manh_dst() + self.last_move()

```

```

def g(self):
    return len(self.history)

def f(self):
    return self.h() + self.g()

def __lt__(self, other):
    return self.f() < other.f()

def get_neighbours(self):
    neighs = []
    zero_coord = self.board_state.index(0)

    # look at neighbours
    if zero_coord + 1 < self.size ** 2 and manh_dst_matrix(zero_coord,
zero_coord + 1, self.size) == 1:
        new_state = self.board_state.copy() # !!!!!!! видимо это число
сдвигается
        new_state[zero_coord], new_state[zero_coord + 1] =
new_state[zero_coord + 1], new_state[zero_coord]
        neighs.append(chain15(new_state, self.history + [self],
self.last_moved + [new_state[zero_coord]]))

    if zero_coord - 1 >= 0 and manh_dst_matrix(zero_coord, zero_coord -
1, self.size) == 1:
        new_state = self.board_state.copy()
        new_state[zero_coord], new_state[zero_coord - 1] =
new_state[zero_coord - 1], new_state[zero_coord]
        neighs.append(chain15(new_state, self.history + [self],
self.last_moved + [new_state[zero_coord]]))

    if zero_coord + self.size < self.size ** 2 and
manh_dst_matrix(zero_coord, zero_coord + self.size,
self.size) == 1:
        new_state = self.board_state.copy()
        new_state[zero_coord], new_state[zero_coord + self.size] =
new_state[zero_coord + self.size], new_state[
zero_coord]
        neighs.append(chain15(new_state, self.history + [self],
self.last_moved + [new_state[zero_coord]]))

    if zero_coord - self.size >= 0 and manh_dst_matrix(zero_coord,
zero_coord - self.size, self.size) == 1:
        new_state = self.board_state.copy()
        new_state[zero_coord], new_state[zero_coord - self.size] =
new_state[zero_coord - self.size], new_state[
zero_coord]
        neighs.append(chain15(new_state, self.history + [self],
self.last_moved + [new_state[zero_coord]]))

    # # Debug
    # for i in neighs:
    #     print(i.last_node() + " " + str(i.f()))
    # print("-----")
    # time.sleep(1)
    return neighs

# Метод проверяющий существует ли решение
def check_is_soluting(self, list):
    sum = 0
    ryad = 1
    d = 3

```

```

    for i in range(0, 16):
        if list[i] == 0:
            index = i
    while i > d:
        ryad += 1
        d += 4
    for i in range(0, 16):
        for j in range(i, 16):
            if not (list[j] == 0):
                if list[i] > list[j]:
                    sum += 1
    sum += ryad
    if sum % 2 == 0:
        return True
    else:
        return False

if __name__ == '__main__':
    start = chain15((5, 11, 12, 7, 1, 4, 10, 0, 2, 9, 3, 15, 13, 8, 6, 14))
    start_time = time.time()
    # start = chain15((5, 1, 9, 3, 11, 13, 6, 8, 14, 10, 4, 15, 0, 12, 7, 2))
    end = chain15((1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0))
    if start.check_is_solving(start.board_state):
        if not (str(start) == str(end)):
            result = a_star.a_star(start, end.last_node())
            end_time = (time.time() - start_time) / 60
            print(result)
            print(
                'Времени затрачено: ' + str(int(end_time)) + ' мин. ' +
                str(int((end_time - int(end_time)) * 60)) + ' сек.')
            print('Потребовалось ' + str(len(result)) + ' ходов для
решения.')
        else:
            print(str(start) + '\nРешение не требуется.')
    else:
        print([])

```

На рисунке 8 представлен результат работы программы со входной цепью: [5, 11, 12, 7, 1, 4, 10, 0, 2, 9, 3, 15, 13, 8, 6, 14], на рисунке 9 со входной цепью: [5, 1, 9, 3, 11, 13, 6, 8, 14, 10, 4, 15, 0, 12, 7, 2], и на рисунке 10 со входной цепью: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 14, 0].

```
solver15.py
ct
a_star.py
solver15.py
__name__ == '__main__'

solver15
1 2 3 4
5 6 0 7 |
9 10 11 8
13 14 15 12

1 2 3 4
5 6 7 0
9 10 11 8
13 14 15 12

1 2 3 4
5 6 7 8
9 10 11 0
13 14 15 12

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 0

[10, 12, 11, 4, 1, 2, 9, 8, 6, 14, 15, 10, 12, 11, 4, 1, 2, 5, 1, 2, 11, 3, 8, 6, 14, 8, 10, 12, 7, 4, 3, 11, 6, 10, 8, 15, 12, 8, 11, 7, 8]
Времени затрачено: 0 мин. 42 сек.
Потребовалось 41 ходов для решения.

Process finished with exit code 0
```

Рис. 8 – Результат работы решения пятнашек.

```
solver15
5 6 7 4
9 10 11 8
13 14 15 12

1 2 3 4
5 6 7 0
9 10 11 8
13 14 15 12

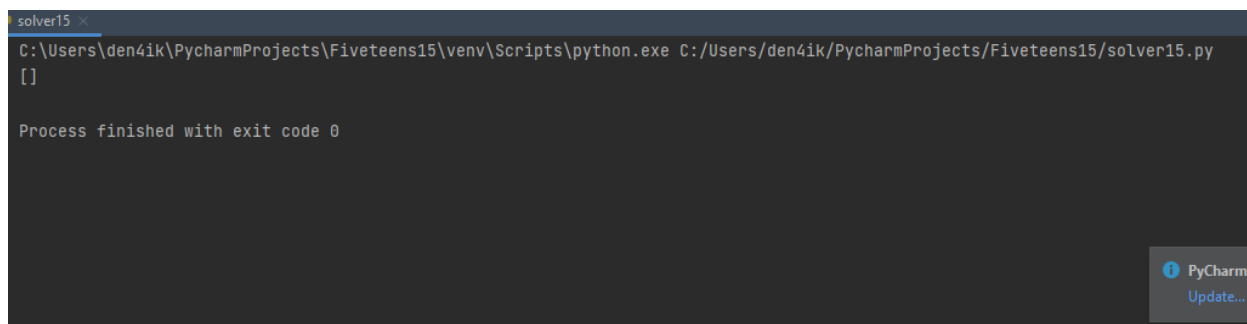
1 2 3 4
5 6 7 8
9 10 11 0
13 14 15 12

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 0

[14, 11, 13, 6, 9, 1, 6, 9, 4, 7, 12, 10, 7, 15, 2, 12, 15, 2, 8, 4, 2, 7, 11, 13, 9, 6, 1, 2, 7, 11, 10, 14, 13, 9, 5, 1, 2, 3, 4, 8]
Времени затрачено: 3 мин. 25 сек.
Потребовалось 40 ходов для решения.

Process finished with exit code 0
```

Рис. 9 – Результат работы решения пятнашек.



```
solver15 x
C:\Users\den4ik\PycharmProjects\Fiveteens15\venv\Scripts\python.exe C:/Users/den4ik/PycharmProjects/Fiveteens15/solver15.py
[]

Process finished with exit code 0
```

Рис. 10 – Результат работы решения пятнашек.

3. ВЫВОД

В ходе данной лабораторной работы были получены практические навыки по применению алгоритмов Кнута-Морриса-Пратта и Бойера-Мура, а также использование алгоритма A^* для решения «Пятнашек».