Федеральное агентство связи

Ордена Трудового Красного Знамени

федеральное государственное бюджетное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Математическая кибернетика и информационные технологии»

Лабораторные работы по дисциплине

«Структуры и алгоритмы обработки данных»

Лабораторная работа №3

# «Методы поиска подстроки в строке»

Выполнила студентка группы БСТ2002

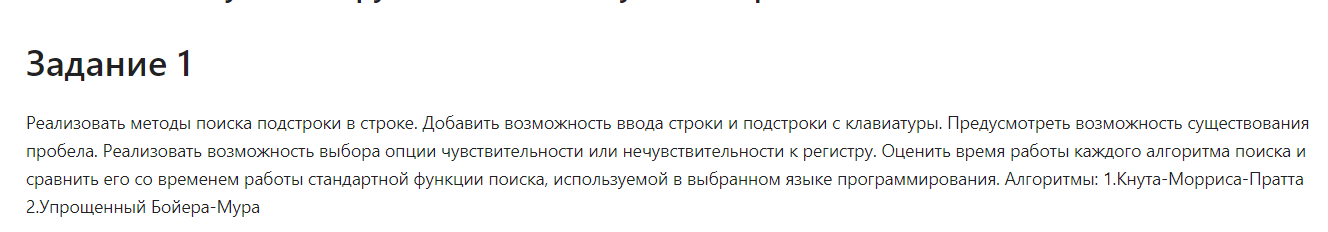
Сергеева А.А.

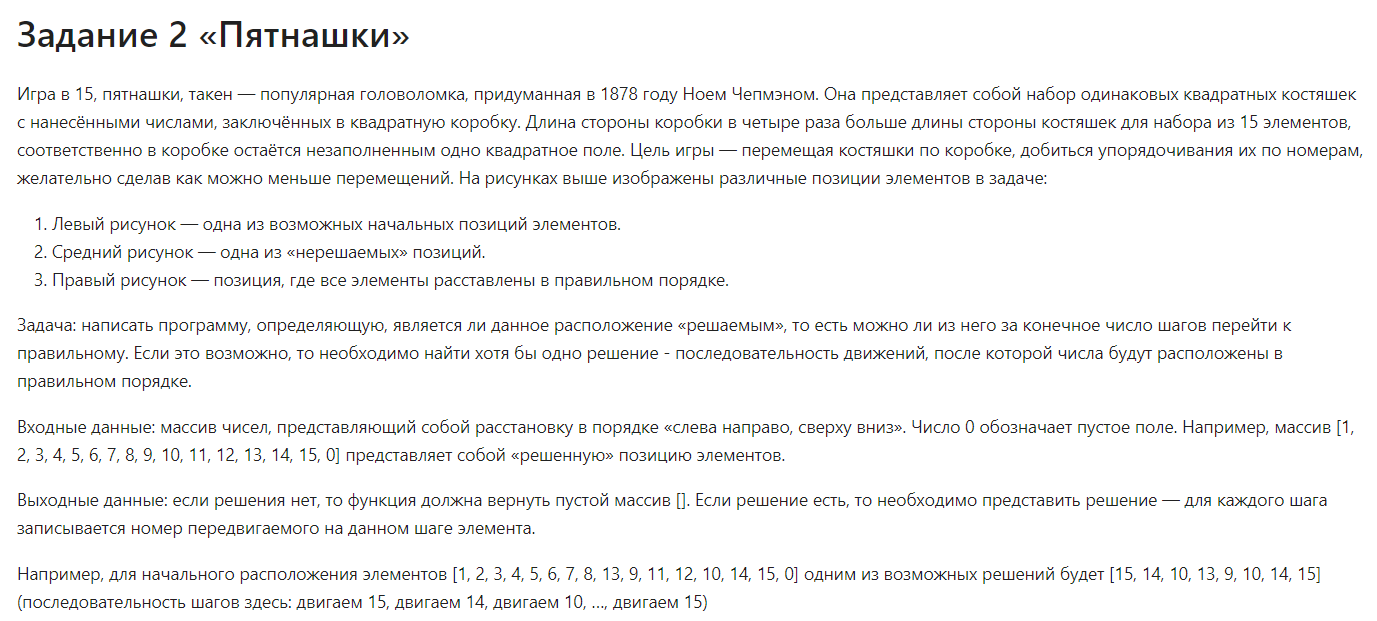
Вариант №16

Проверил: Аршинов Е.А.

Москва 2022

**Задание**





**Ход работы**

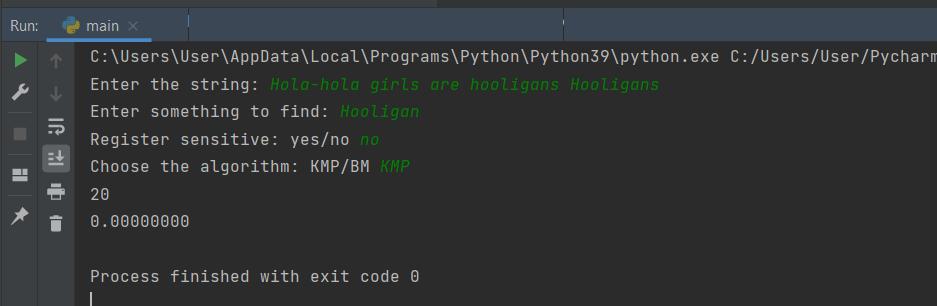
Задание 1

Алгоритм Кнута-Морриса-Пратта состоит в том, что изначально у нас есть строка и подстрока, которую надо найти. Объединим их в одну строку так, что вначале будет подстрока, потом поставим любой символ разделитель, не содержащийся в строке (например, \*) и в конец допишем всю стоку. Посчитаем на ней значение [префикс-функции](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F). Благодаря разделительному символу, выполняется то, что после него префикс будет равен или меньше нашей подстроки. Если в какой-то позиции i выполняется условие, что префикс равен подстроке, то мы нашли нашу подстроку в строке.

Код программы:

def solution(n, m):  
 n = m + '$' + n  
  
 mass = [0 for i in range(len(n))]  
  
 for i in range(2\*len(m)+1, len(n)):  
 max\_pr = -1  
 for j in range(1, i-1):  
 if n[:j] == n[i-j:i] and len(n[:j]) > max\_pr:  
 max\_pr = len(n[:j])  
 mass[i] = max\_pr  
  
 for i in range(len(mass)):  
 if mass[i] == len(m):  
 return i - len(m) \* 2 - 1

Результат работы программы:

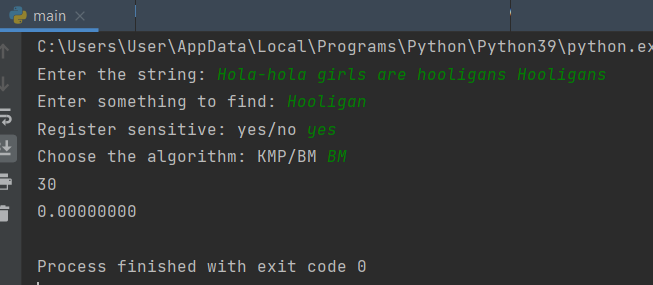


Алгоритм Бойера-Мура состоит в том, что мы создаем словарь со всеми допустимыми символами в стоке. Всем значениям присваиваем длину подстроки. Далее в подстроке мы присваиваем новые значения символам. Переопределенные значения символов присваиваем в словарь. Затем исходная строка и шаблон совмещаются по началу, сравнение ведется по последнему символу. Если последние символы совпадают, то сравнение идет по предпоследнему символу и так далее. Если не совпадает, то сдвигаемся на то количество шагов, которое равно не совпавшему символу в строке.

Код программы:

def solution(n, m):  
 # Creating & filling  
 mass = [0 for i in range(len(m))]  
 for i in range(len(m) - 1, -1, -1):  
 for j in range(len(m) - 1, i, -1):  
 if m[i] == m[j]:  
 mass[i] = mass[j]  
 break  
 else:  
 mass[i] = len(m) - i - 1  
  
 # Create an alphabetical table (key = symbol, value = number)  
 a = {i: len(m) for i in set(n)}  
 for key, val in a.items():  
 for i in range(len(m)):  
 if key == m[i]:  
 a[key] = mass[i]  
 break  
  
 # Algorithm itself  
 el = len(m)-1  
 while el < len(n):  
 old\_el = el  
 for i in range(len(m)-1, -1, -1):  
 if m[i] == n[el]:  
 el -= 1  
 else:  
 el += a[n[el]]  
 break  
 if old\_el - el == len(m):  
 return el + 1

Результат работы программы:



Задание 2

Создаем класс Position, где содержатся конструктор с текущей, начальной и конечной позицией; метод str, отвечающий за строковое представление объекта; метод lt, который реализует проверку на «меньше чем» для экземпляров класса.

Далее импортируем модуль PriorityQueue (конструктор для очереди с приоритетом).

Определяем размер поля. Функция shifts - генератор сдвигов. Там мы находим индекс нуля, находим его позицию в поле 4х4, где i это номер строки, а j номер столбца. В зависимости от местоположения нуля, смотрим варианты, куда его можно сдвинуть: вверх, вниз, вправо, влево. Считаем индексы новых позиций нуля, выводим новое состояние поля, где 0 сдвинут в одном из возможных направлений.

С помощью функции parityOfPairs проверяем четность неправильных пар.

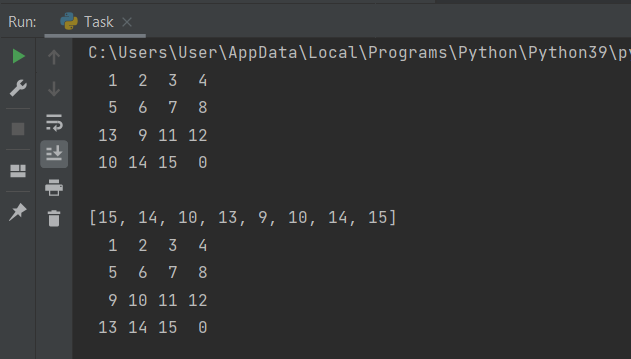
В функции fifteenGame устанавливаем кортеж с нашим «правильным и нужным» списком. Далее если количество неправильных пар четное, то такая позиция нерешаема. В противном случае создаем из начальной позиции кортеж, создаем экземпляр класса, с кортежем стартовой позиции в качестве параметра, длиной пути от начала до текущей точки и длиной от текущей позиции до конца. Выводим начальную позицию. Создаем экземпляр класса приоритетной очереди. Заносим в нее пару — это кортеж стартовой позиции, и вес данной вершины. Создаем множество посещенных вершин. Создаем словарь, в котором будем хранить позиции, где для каждой будет определена ее предыдущая позиция. В цикле пока позиция не будет равна решению получаем приоритетную позицию. В цикле for смотрим для каждого варианта передвижения нуля в одном состоянии поля. Если такой вариант передвижения не находится в списке посещенных, то рассчитаем расстояние до терминального состояния, это количество цифр, стоящих не на своих местах. То мы заносим его в очередь, с состоянием в качестве параметра, длиной пути от старта до текущего состояния и длиной до финиша, а в словарь добавляем новый ключ, то есть эту позицию, его значением будет предыдущая позиция. Добавляем этот вариант перемещения в посещенный.

Берем из словаря родителя текущего состояния поля. Запоминаем индекс нуля из текущего состояния и по этому индексу находим элемент, который сдвинули в предыдущем состоянии. Предыдущее состояние делаем текущим. Разворачиваем путь, чтобы получить путь от первого до последнего сдвига. Выводим нашу позицию.

Код программы:

class Position:  
 def \_\_init\_\_(self, position, start\_distance,finish\_distance):  
 self.position = position  
 self.start\_distance = start\_distance  
 self.finish\_distance = finish\_distance  
  
 def \_\_str\_\_(self):  
 return '\n'.join((N\*'{:3}').format(\*[i%(N\*N) for i in self.position[i:]]) for i in range(0, N\*N, N))  
  
 # Переопределяем метод less then для работы PriorityQueue  
 def \_\_lt\_\_(self, other):  
 return self.start\_distance+self.finish\_distance < other.start\_distance+other.finish\_distance  
  
  
from queue import PriorityQueue  
  
N = 4  
  
def shifts(position):  
 zeroPosition = position.index(0)  
 i, j = divmod(zeroPosition, N)  
 displacement = []  
 if i > 0: displacement.append(-N) # вверх  
 if i < N - 1: displacement.append(N) # вниз  
 if j > 0: displacement.append(-1) # влево  
 if j < N - 1: displacement.append(1) # вправо  
 for offset in displacement:  
 swap = zeroPosition + offset  
 yield tuple(position[swap] if x==zeroPosition else position[zeroPosition] if x==swap else position[x] for x in range(N\*N))  
  
def parityOfPairs(state):  
 countOfPairs = 0  
 for i in range(len(state)-1):  
 if state[i] > state[i+1]:  
 countOfPairs +=1  
 return countOfPairs % 2  
  
  
def fifteenGame(startState):  
 terminalState = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0)  
 if parityOfPairs(startState)==0:  
 print("Нет решений")  
 else:  
 startState= tuple(startState)  
 p = Position(startState, 0,0)  
 print(p)  
 print()  
 fieldStates= PriorityQueue()  
 fieldStates.put(p)  
 closePoints = set([p])  
 parents = {p.position: None}  
  
 while p.position != terminalState:  
 p =fieldStates.get()  
  
 for k in shifts(p.position):  
 count= 0  
 if k not in closePoints:  
 for m in range(len(k)):  
 if k[m] != terminalState[m]:  
 count+=1  
  
 fieldStates.put(Position(k, p.start\_distance +1,p.finish\_distance+count))  
 parents[k] = p  
 closePoints.add(k)  
  
 path = []  
 x = p  
 previous = p  
 while p.position != startState:  
 p = parents[p.position]  
 number = p.position[previous.position.index(0)]  
 path.append(number)  
 previous = p  
 path.reverse()  
  
 print(path)  
 print(x)  
  
startState = [1, 2, 3, 4, 5, 6, 7, 8, 13, 9,11,12,10,14,15,0]  
fifteenGame(startState)

Результат работы программы:



**Вывод**

Изучили основные алгоритмы поиска подстроки в строке. А также реализовал алгоритм решения игры "Пятнашки".