Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Домашняя работа №1 по курсу "Анализ алгоритмов"

Тема Графовые представления

Студент Иммореева М.А.

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Строганов Д.В.

Москва — 2023 г.

Оглавление

1 Выполнение задания

1.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования Python. Выбор обусловлен наличием библиотек для измерения времени, наличием инструментов для работы с параллельными потоками. Были использованы библиотеки threading, multiprocessing, os.

1.2 Программный код

В листингах 1.1, 1.2, 1.3 приведены реализации алгоритмов.

Листинг 1.1 – Алгоритм Бойера–Мура поиска подстроки в строке.

```
def bouer_moor():
    text = input()                                          #text
    pattern = input()                                       #pattern
    bad_char = bad_char_heuristic(pattern)                  #1
    good_suffix = good_suffix_heuristic(pattern)            #2
    i = 0                                                   #3
    indexes = []                                            #4
    while i <= len(text) - len(pattern):                    #5
        j = len(pattern) - 1                                #6
        while j >= 0 and pattern[j] == text[i + j]:         #7
            j -= 1                                          #8
        if j < 0:                                           #9
            indexes.append(i)                               #10
            i += good_suffix[0]                             #11
        else:                                               #12
            x = j - bad_char.get(text[i + j], -1)           #13
            y = good_suffix[j + 1]                          #14
            i += max(x, y)                                  #15
    return indexes
```

Листинг 1.2 – Эвристика хорошего суффикса

```
def good_suffix_heuristic(pattern):
    good_suffix = [-1] * (len(pattern) + 1)
    border = [0] * (len(pattern) + 1)
    i = len(pattern)
    j = len(pattern) + 1
    border[i] = j
    while i > 0:
        while j <= len(pattern) and pattern[i - 1] != pattern[j -
            1]:
            if good_suffix[j] == 0:
                good_suffix[j] = j - i
            j = border[j]
        i -= 1
        j -= 1
        border[i] = j
    j = border[0]
    for i in range(len(pattern) + 1):
        if good_suffix[i] == -1:
            good_suffix[i] = j
        if i == j:
            j = border[j]
    return good_suffix
```

Листинг 1.3 – Эвристика плохого символа

```
def bad_char_heuristic(pattern):
    bad_char = {}
    for i in range(len(pattern)):
        bad_char[pattern[i]] = i
    return bad_char
```

1.3 Графовые представления

На рисунке 1.1 представлен операционный граф для алгоритма Бойера–Мура поиска подстроки в строке. На рисунке 1.2 представлен информационный граф для этого же алгоритма. На рисунке 1.3 представлен граф операционной истории, на рисунке 1.3 представлен граф информационной истории.

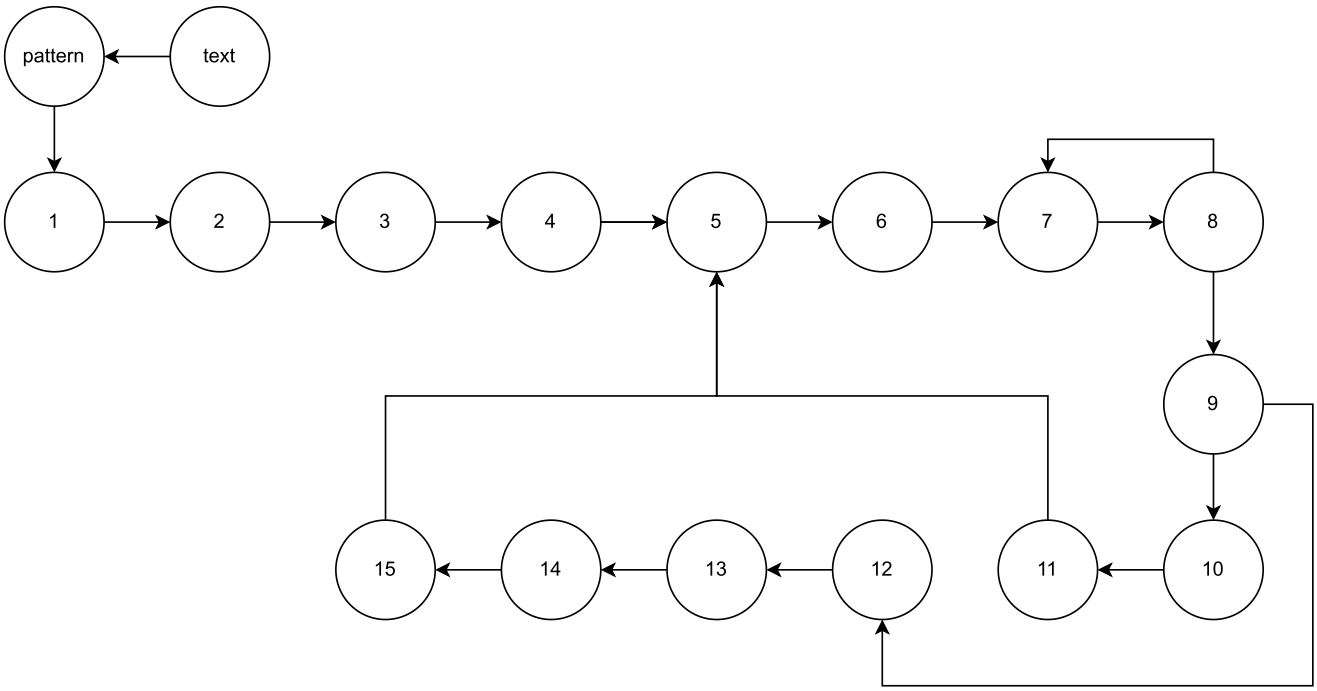

Рисунок 1.1 – Операционный граф

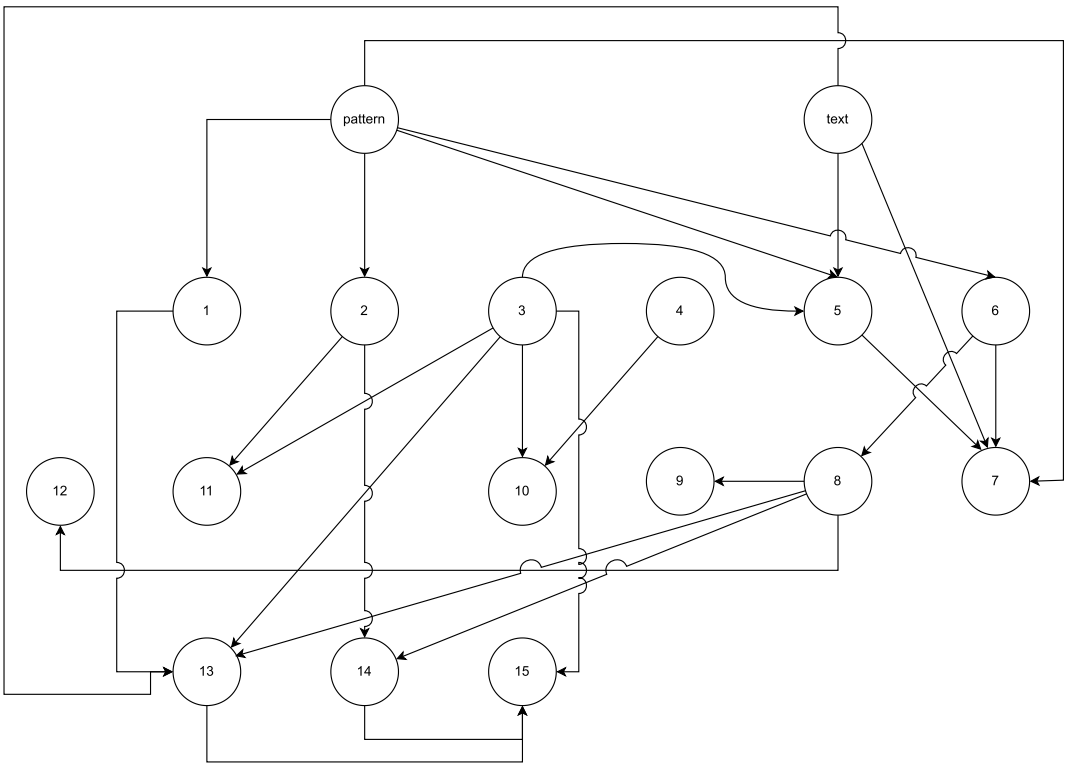

Рисунок 1.2 – Информационный граф

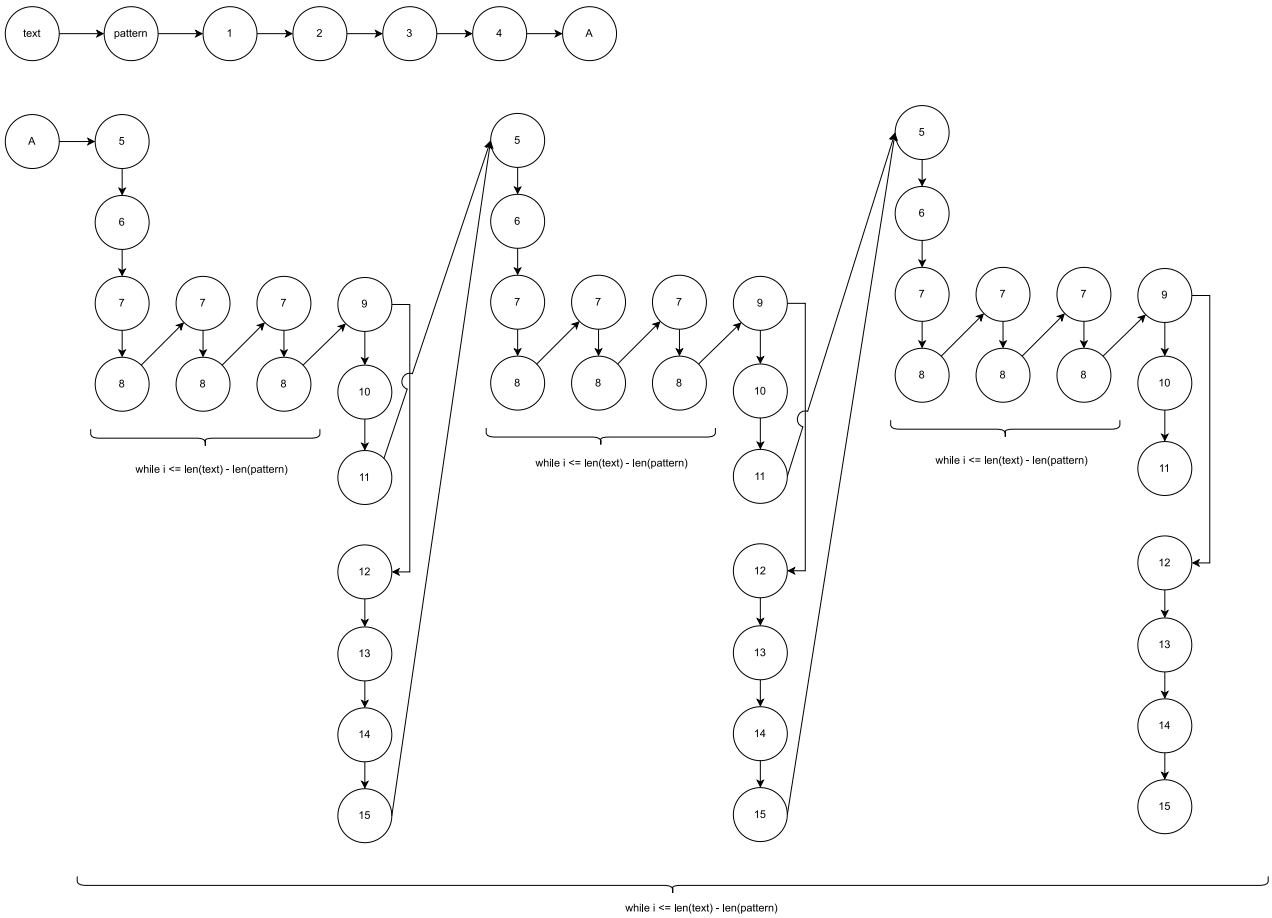

Рисунок 1.3 – Граф операционной истории

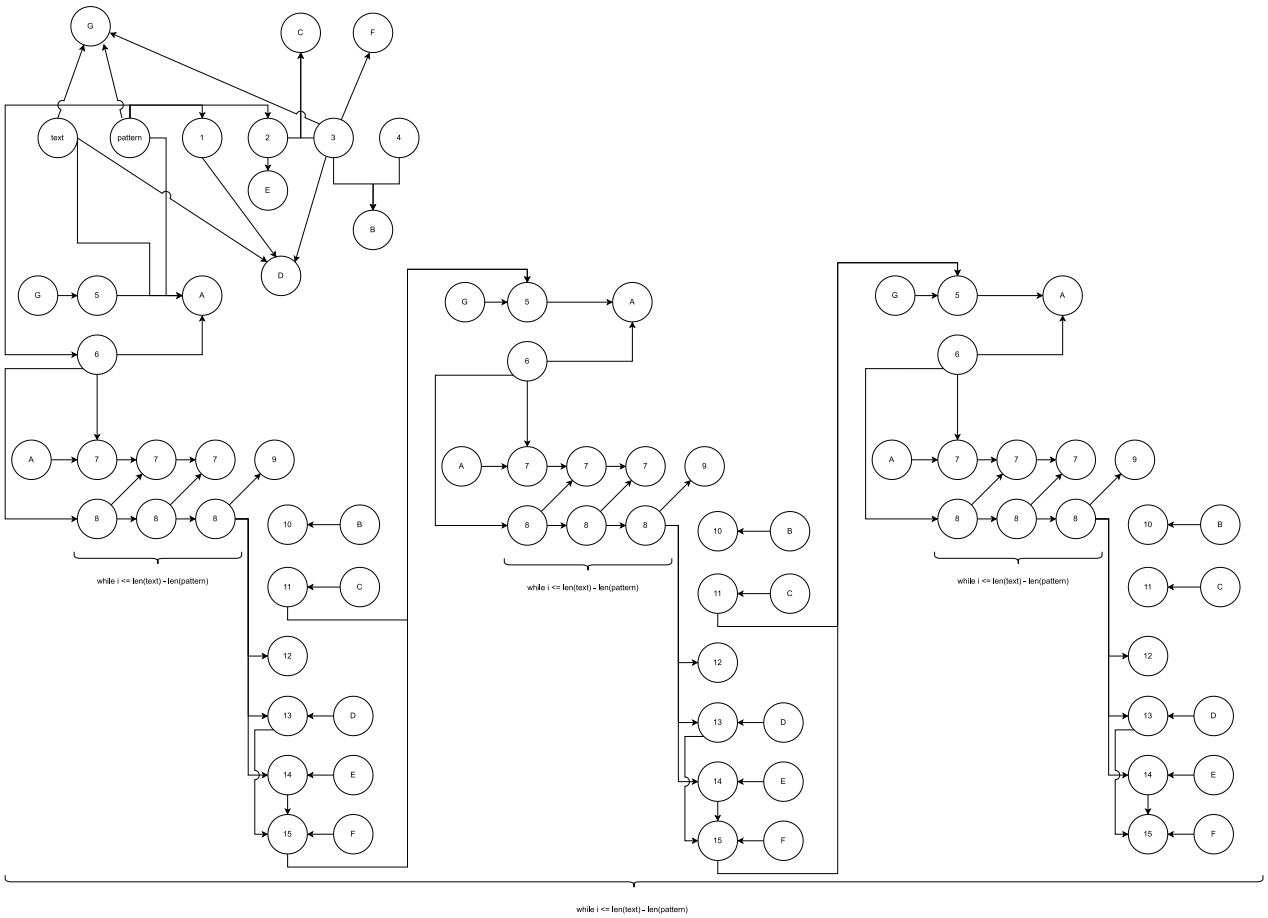

Рисунок 1.4 – Граф информационной истории

1.4 Параллельный алгоритм Бойера–Мура

Так как при работе алгоритма отступ после каждого сравнения может быть рассчитан только после нахождения несоответствия в тексте, был выбран подход разделения текста, в котором происходит поиск паттерна, на равные части. Вычисление результата для каждой части будет независим от вычисления результата других частей. При ситуации, где разделение текста происходит на паттерне, выбран подход, где параллельный процесс с собственной частью может получить доступ к следующей за ней части текста ровно на размер строки паттерна.

Список использованных источников

[1] Волосова А. В. Параллельные методы и алгоритмы. – Москва: МАДИ, 2020.

[2] time — Time access and conversions // Python.org URL: https://docs.python.org/3/library/time.html (дата обращения: 12.11.2023).

[3] Definitive Guide: Threading in Python Tutorial, May 2020 URL: https://www.datacamp.com/tutorial/threading-in-python (дата обращения: 12.11.2023).