МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **ІКНІ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 12
З дисципліни: *"Алгоритми та структури даних"*
На тему: *"Алгоритм пошуку Бойєра Мура"*

Лектор:
доц. каф. ПЗ
Коротєєва Т.О.

Виконав:
ст. гр. ПЗ – 22
Ясногородський Н.В.

Прийняв:
асист. каф. ПЗ
Франко А.В.

« ____ » _____ 2022 р.
∑= _____ _____.

Львів – 2022

Тема роботи: Алгоритм пошуку Бойєра Мура

Мета роботи: Навчитися застосовувати алгоритм пошуку Бойєра Мура при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму

Теоретичні відомості

Нажаль співпадіння зустрічаються значно рідше, ніж неспівпадіння. Тому виграш від використання алгоритму КМП в більшості випадків незначний. Інший алгоритму Бойера-Мура базується на наступній схемі: порівняння символів починається з кінця взірця, а не з початку. Нехай для кожного символу x взірця dx - відстань від самого правого у взірці входження x до кінця взірця. Припустимо, знайдено неспівпадіння між взірцем та текстом. Тоді взірець можна зразу посунути вправо на dx позицій що є більше або рівне 1. Якщо x у взірці взагалі не зустрічається, то посунути взірець можна зразу на всю його довжину m.

В даному алгоритмі розглядається поняття стоп-символа - це є символ в тексті, який є першим неспівпадінням тексту і взірця при порівнянні справа (з кінця взірця). Розглянемо три можливих ситуації:

1. 1. Стоп-символ у взірці взагалі не зустрічається, тоді зсув дорівнює довжині взірця m.

2. 2. Крайня права позиція k входження стоп-символа у взірці є меншою від його позиції j у тексті. Тоді взірець можна зсунути вправо на k-j позицій так, щоб стоп-символ у взірці і тексті опинились один під одним.

3. 3. Крайня права позиція k входження стоп-символа у взірці є більшою від його позиції j у тексті. Тоді зсув дорівнює 1.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

15. Дано текст що складається з n стрічок. Визначити чи є серед даного тексту цифри, якщо є вилучити їх з тексту. Знайти задане слово в тексті.

ВИКОНАННЯ РОБОТИ

Код програми:

```
"""
15. Дано текст що складається з n стрічок.
Визначити чи є серед даного тексту цифри, якщо є вилучити їх з тексту.
Знайти задане слово в тексті.

Виміряти час пошуку,
а також час (окремо) додаткового опрацювання тексту та/чи взірця,
якщо таке є.
"""

from lab12.boyer_moore import search

TEXT = """Lorem Ipsum is simply dummy text of the printing and typesetting industry.
Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.
It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged.
It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages,
and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum."""

TEXT_2 = "cabcbbb"
SEARCH_PATTERN_2 = "cabca"
```

```
# txt = "ABAAABCD"
# pat = "ABC"


def get_input_text(_search_pattern):
    text = []
    for char in TEXT_2:
        if not char.isnumeric():
            text.append(char)
    text_str = "".join(text)
    print(f"Text: \n{text_str}\n")
    if _search_pattern:
        return text_str, _search_pattern
    search_pattern = input("Please enter search pattern: ")
    return text_str, search_pattern


if __name__ == "__main__":
    # while True:
    text, search_pattern = get_input_text(SEARCH_PATTERN_2)
    search(text, search_pattern)
    print()

NO_OF_CHARS = 256


def bad_char_heuristic(string, size):
    """
    The preprocessing function for
    Boyer Moore's bad character heuristic
    """
```

```
    # Initialize all occurrence as -1
    bad_chars = [-1] * NO_OF_CHARS

    # Fill the actual value of last occurrence
    for i in range(size):
        bad_chars[ord(string[i])] = i

    # return initialized list
    print(f"bad chars: {bad_chars}")
    return bad_chars


def search(txt, pat):
    """
    A pattern searching function that uses Bad Character
    Heuristic of Boyer Moore Algorithm
    """
    m = len(pat)
    n = len(txt)

    # create the bad character list by calling
    # the preprocessing function badCharHeuristic()
    # for given pattern
    bad_chars = bad_char_heuristic(pat, m)

    # s is shift of the pattern with respect to text
    s = 0
    while s <= n - m:
        j = m - 1
```

```
# Keep reducing index j of pattern while
# characters of pattern and text are matching
# at this shift s
while j >= 0 and pat[j] == txt[s + j]:
    j -= 1

# If the pattern is present at current shift,
# then index j will become -1 after the above loop
if j < 0:
    print("Pattern occur at shift = {}".format(s))

    """
    Shift the pattern so that the next character in text
        aligns with the last occurrence of it in pattern.
    The condition s+m < n is necessary for the case when
    pattern occurs at the end of text
    """
    s += m - bad_chars[ord(txt[s + m])] if s + m < n else 1
else:
    """
    Shift the pattern so that the bad character in text
    aligns with the last occurrence of it in pattern. The
    max function is used to make sure that we get a positive
    shift. We may get a negative shift if the last occurrence
    of bad character in pattern is on the right side of the
    current character.
    """
    s += max(1, j - bad_chars[ord(txt[s + j])])
```

ПРОТОКОЛ РОБОТИ

```
❯ poetry run python -m lab12.main
Text:
cabcbbbcabca

bad_chars=defaultdict(<function bad_char_map.<locals>.<lambda> at 0x1008cc820>, {'c': 3, 'a': 4, 'b': 2})

text and search_pattern:
cabcbbbcabca
----*-------
cabca

shift=2
text and search_pattern:
cabcbbbcabca
------*-----
  cabca

shift=4
char match
text and search_pattern:
cabcbbbcabca
--------*---
    cabca

char match
text and search_pattern:
cabcbbbcabca
-------*----
    cabca

char match
text and search_pattern:
cabcbbbcabca
------*-----
    cabca

shift=5
text and search_pattern:
cabcbbbcabca
---------*--
     cabca
```

ВИСНОВКИ

Під час виконання лабораторної роботи я навчився застосовувати алгоритм пошуку Бойєра Мура при розв’язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначив складність алгоритму.