МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

до лабораторної роботи №11

на тему: «Алгоритм Кнута, Моріса і Прата пошуку в стрічці.»

з дисципліни: «Алгоритми і структури даних»

Лектор:
доц. кафедри ПЗ
Коротєєва Т. О.

Виконав:
ст. гр. ПЗ-22
Чаус О. М.

Прийняв:
асист. кафедри ПЗ
Франко А. В.

« ____ » _____ 2022 р.

∑= _____ ……:

Львів-2022

Тема роботи: Алгоритм Кнута, Моріса і Прата пошуку в стрічці.
Мета роботи: Навчитися застосовувати алгоритм КМП для пошуку у стрічці при розв'язанні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму.

Теоретичні відомості

Алгоритм Кнута — Морріса — Пратта (скорочено алгоритм КМП) — один із алгоритмів пошуку рядка, що шукає входження слова W у рядку S, використовуючи просте спостереження, що коли відбувається невідповідність, то слово містить у собі достатньо інформації для того, щоб визначити, де наступне входження може початися, таким чином пропускаючи кількаразову перевірку попередньо порівняних символів.
Алгоритм, що винайшли Дональд Кнут та Вон Пратт, а також незалежно від них Джеймс Морріс, опубліковано у спільній статті у 1977 році.
КМП витрачає небагато часу (за порядком розміру W[], O(n)) на попереднє обчислення таблиці, і потім використовує таблицю для швидкого пошуку рядка за час O(k).
Алгоритм передбачає побудову таблиці T, яка показує де ми маємо почати нове зіставлення в разі невдачі поточного. Записи T утворені так, що якщо маємо збіг від S[m], що зазнав невдачі при порівнянні S[m + i] з W[i], тоді наступний можливий збіг почнеться з індексу m + i - T[i] в S (T[i] це кількість повернень, які ми маємо зробити після невдачі). Це має два наслідки: перший, T[0] = -1, показує, що якщо W[0] це не збіг, ми не можемо повернутись і повинні просто перевірити наступний символ; і другий, хоча наступний можливий збіг почнеться з індексу m + i - T[i], ми не маємо насправді перевіряти будь-який з символів T[i] після цього, так що ми продовжуємо пошук з W[T[i]].
Через те, що дві складові алгоритму мають складності, відповідно, O(k) і O(n), складність всього алгоритму становить O(n + k).
Складності залишаються незмінними, попри те, скільки зразків, що повторюються в W або S.

Індивідуальне завдання

Задано два тексти. В першому тексті вилучити слово що найчастіше зустрічається і знайти його входження в другий текст відповідним алгоритмом пошуку.

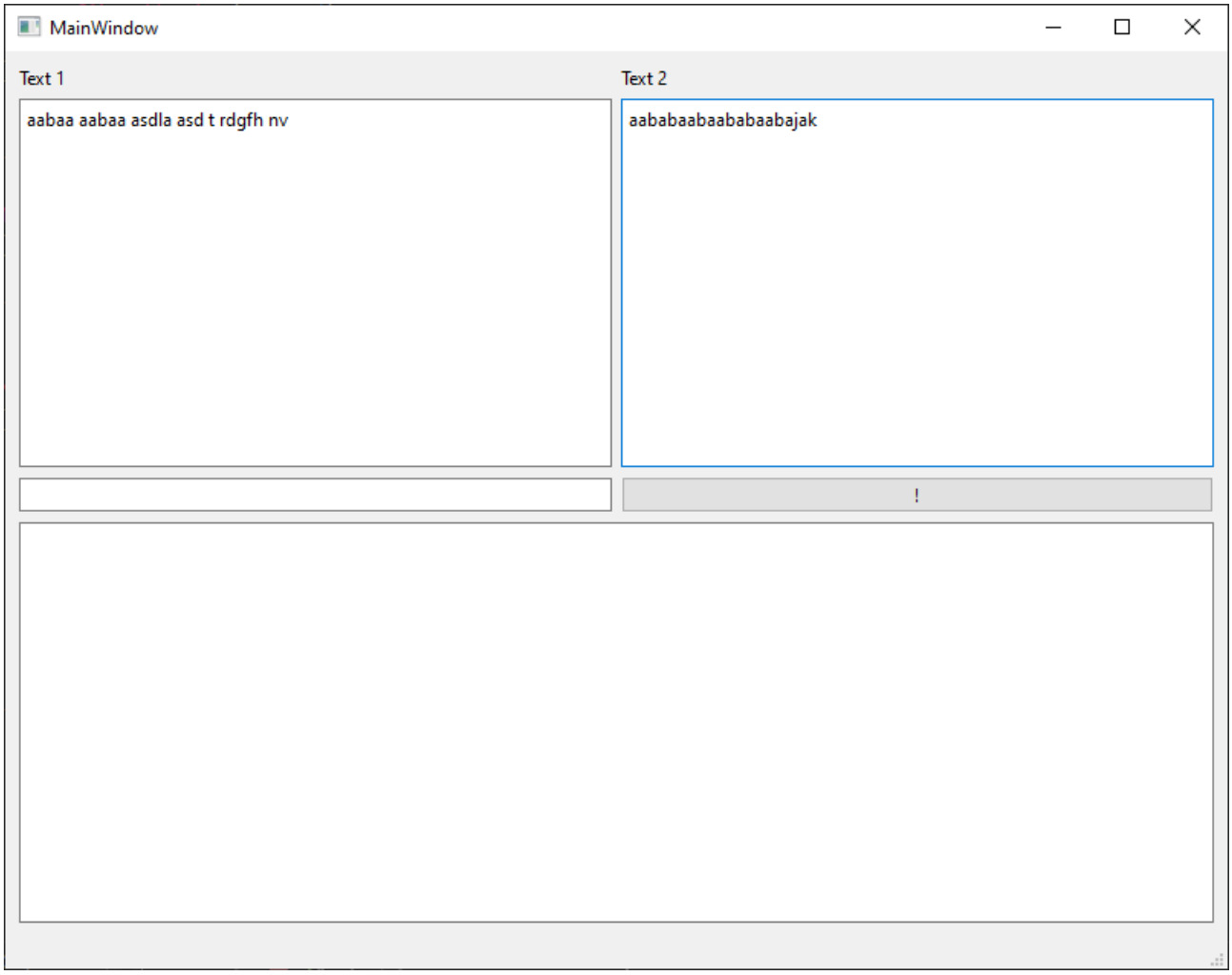
Код програми

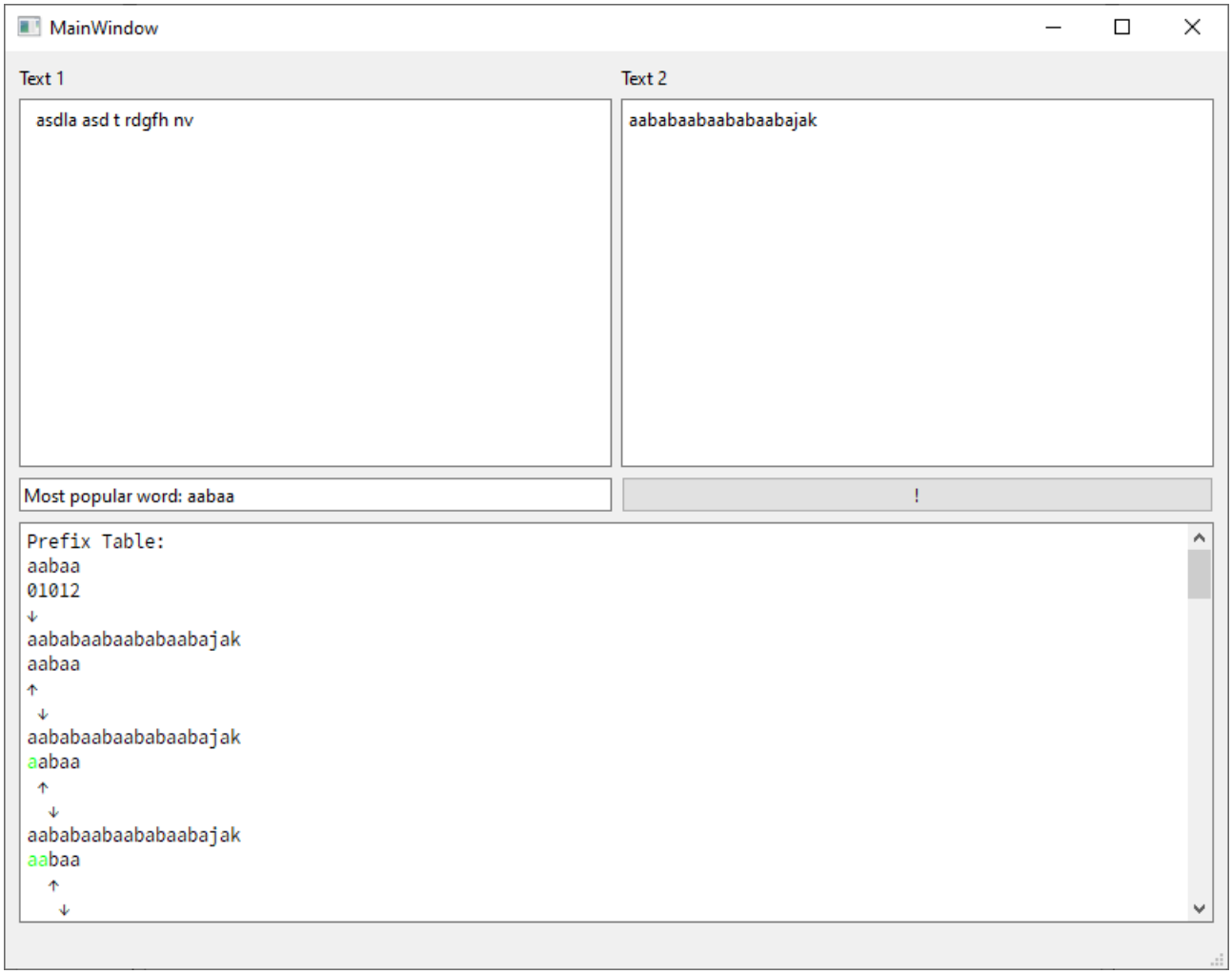
```
std::string delete_most_common(std::string &txt) {
    const std::string delimeters {" .,!;?"};
    std::vector<std::string> words {split(txt, delimeters)};
    std::unordered_map<std::string, std::size_t> words_count;
    for(const auto &x : words)
        words_count[x]++;
    auto max = std::max_element(words_count.cbegin(), words_count.cend(),
[](const auto &a, const auto &b) {
        return a.second < b.second;
    });
    auto indices = kmp_search(txt, max->first);
    std::size_t deleted {0};
    std::size_t max_size {max->first.size()};
    for(const int idx : indices) {
        std::size_t str_idx {idx - deleted * max_size};
        if ((str_idx == 0 ? true :
             std::find(delimeters.cbegin(), delimeters.cend(), txt[str_idx - 1])
!= delimeters.cend()) &&
```

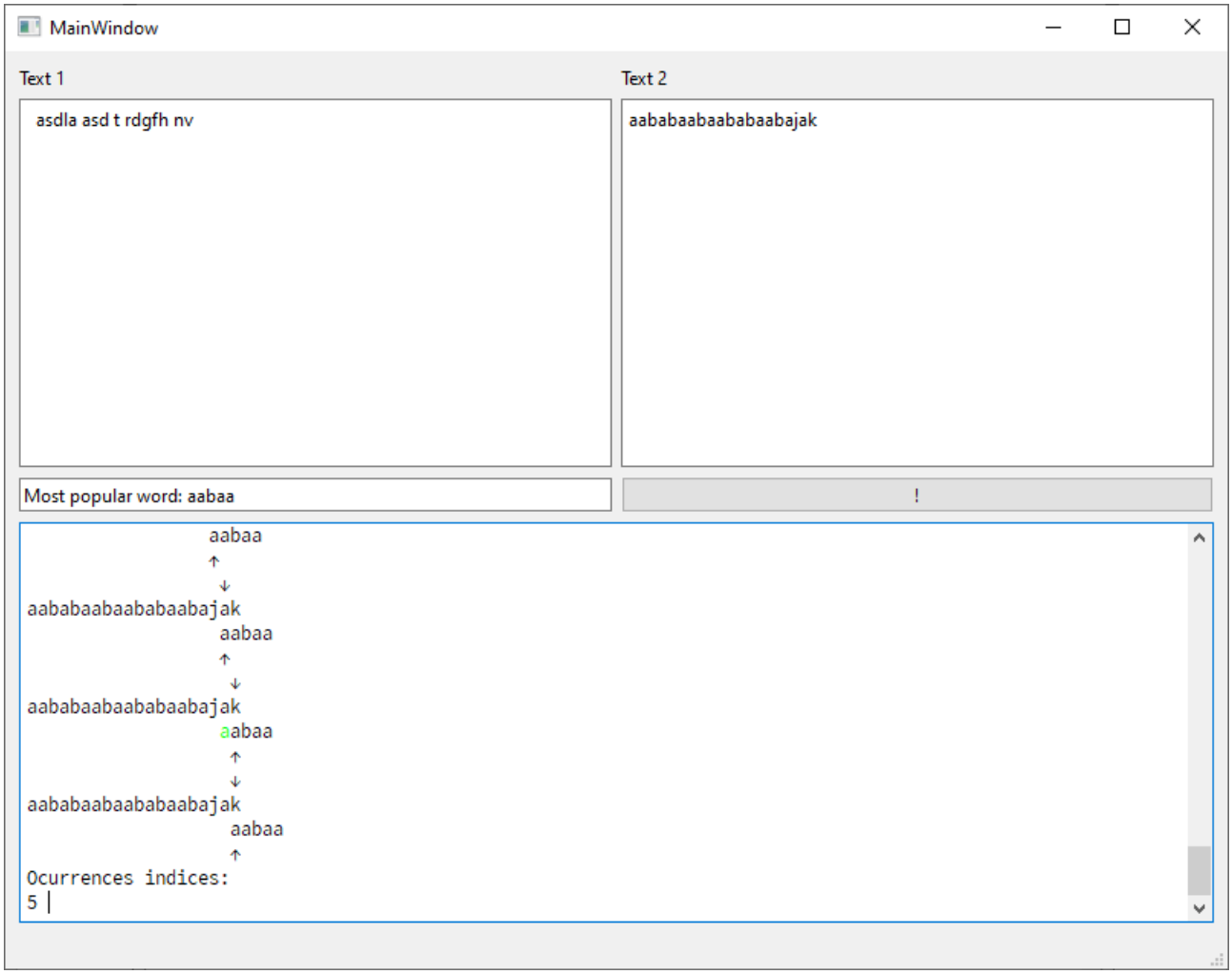
```
                str_idx + max_size == txt.size() ? true :
                std::find(delimeters.cbegin(), delimeters.cend(), txt[str_idx +
max_size]) != delimeters.cend()) {
            txt.erase(txt.begin() + str_idx, txt.begin() + str_idx + max_size);
            deleted++;
        }
    }
    return max->first;
}

std::vector<std::size_t> kmp_search(const std::string txt, const std::string
wrd) {
    std::vector<std::size_t> res;
    const auto wrd_size = wrd.size();
    const auto txt_size = txt.size();
    // find prefix table
    std::vector<int> prefix_tbl (wrd_size, 0);
    std::size_t i = 0, j = 1;
    while (j < wrd_size) {
        if (wrd[i] == wrd[j]) {
            prefix_tbl[j] = i + 1;
            i++;
            j++;
        }
        else if (i) //go back
            i = prefix_tbl[i - 1];
        else { //first element is always 0
            prefix_tbl[j] = 0;
            j++;
        }
    }
    //search in string
    i = 0, j = 0;
    while (i < txt_size) {
        if (txt[i] == wrd[j]) {
            i++;
            j++;
        }
        else if (j)
            j = prefix_tbl[j - 1];
        else {
            i++;
        }
        if (j == wrd_size) {
            res.push_back(i - j);
            j = prefix_tbl[j - 1];
        }
    }
    return res;
```

Зображення програми


MainWindow
Text 1
aabaa aabaa asdla asd t rdgfh nv
Text 2
aababaabaababaabajak
!


MainWindow
Text 1
asdla asd t rdgfh nv
Text 2
aababaabaababaabajak
Most popular word: aabaa
!
Prefix Table:
aabaa
01012
↓
aababaabaababaabajak
aabaa
↑
↓
aababaabaababaabajak
aabaa
↑
↓
aababaabaababaabajak
aabaa
↑
↓



Висновок:

навчився застосовувати алгоритм КМП для пошуку у стрічці при розв’язанні задач та перевірив його ефективність на різних масивах даних. Дослідив складність алгоритму $O(n+m)$