ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 7

«Жадные алгоритмы»

Выполнил работу

Ковров Евгений

Академическая группа №J3110

Принято

Должность, звание Вершинин Владислав

Санкт-Петербург

2024

1. Введение

Цель: реализовать алгоритм сопоставления строки с шаблоном.

Задачи:

1. Постановка задачи
2. Алгоритм
3. Реализация
4. Тестирование
5. Оптимизация

Теоретическая подготовка

Основной подход, использованный для решения задачи – жадный алгоритм. Данный подход удобен, так как решение задачи в общем случае предполагает локальный поиск максимального количества символов строки, подходящих под символ «\*» из шаблона.

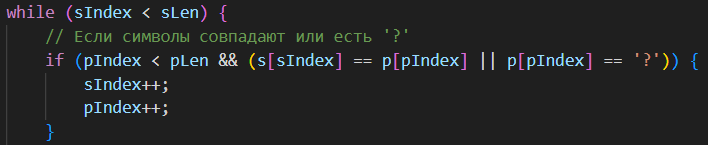
1. Реализация

Постановка задачи: Дана строка, содержащая только маленькие буквы английского алфавита, и шаблон, содержащий те же буквы и символы «?» и «\*». Символ «?» означает любую букву, символ «\*» - любую последовательность (в том числе пустую). При данный условиях необходимо определить, сопоставима ли строка с шаблоном.

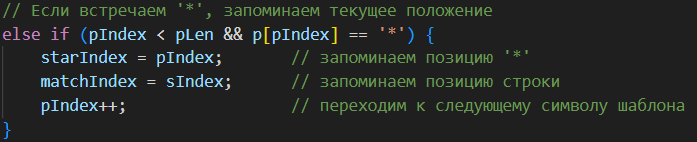
Алгоритм**:**

1. Проходим циклом по всем индексам строки
2. Сравниваем символы с текущими индексами в строке и в шаблоне
3. Если символы совпадают (в т.ч. «?»), увеличиваем оба индекса
4. Если встречается «\*» - запоминаем её индекс и увеличиваем индекс шаблона
5. Если символы не совпадают, но до этого в шаблоне была «\*» - возвращаемся к индексу после нее в шаблоне и начинаем с её индекса в строке
6. Если ничего не подходит, досрочно возвращаем отрицательный ответ
7. После завершения основного цикла проверяем, что в шаблоне осталось либо ничего, либо только «\*»
8. Возвращаем результат проверки «полноты» совпадения

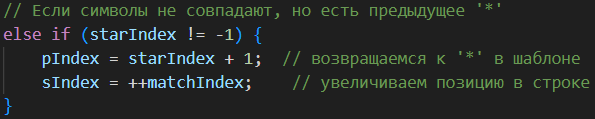
Реализация:

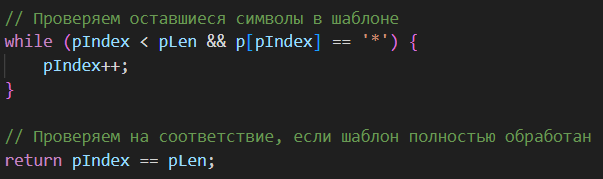


Изображение №1 – Проверка совпадения символов в цикле



Изображение №2 – Если встретили «\*»

  
Изображение № 3 – Если «\*» была до этого, и символы не совпадают

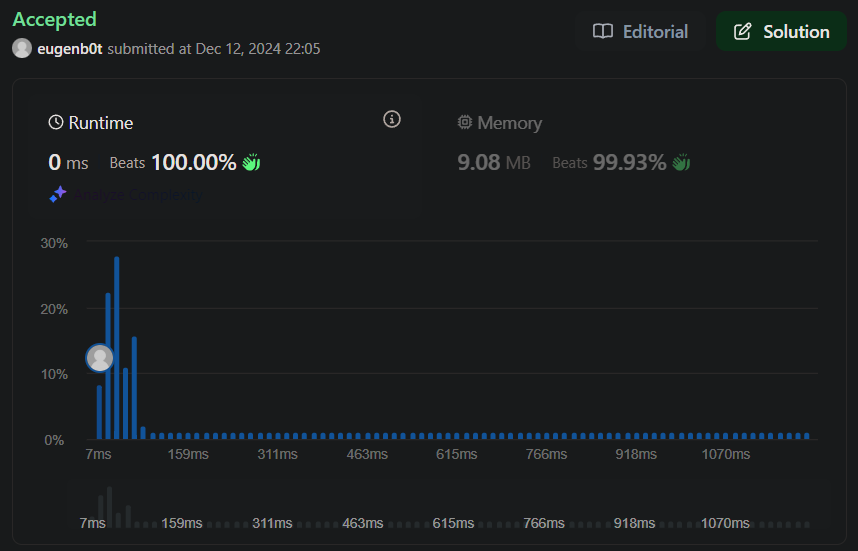


Изображение №4 – Проверка остатка шаблона и проверка полноты совпадения

Экспериментальная часть

Тестирование:

Так как задача была взята с leetcode, она была протестирована тестами на платформе (изображение № 5).



Изображение № 5 – Отчёт leetcode

Подсчёт памяти:

Дополнительная память для сложных структур не используется. Программа в принципе оперирует 6 int переменными. Поэтому память составляет O(1).

Подсчёт асимптотики:

Алгоритм использует один проход по строке и один проход по шаблону, что приводит к тому, что каждый символ строки и символ шаблона будет обработан в рамках основных условий. При этом, в худшем случае алгоритм может выполнять сравнение символов несколько раз из-за встречи символа **\***, который может соответствовать нулевому или множеству символов.



Изображение № 6 – Перебор всех символов строки



Изображение № 7 – Перебор всех оставшихся символов шаблона

Итоговая теоретическая сложность – O(N + M), где N – длина строки, M – длина шаблона.

Путем тестирования алгоритма на около рандомных тестах, были получены результаты, представленные в таблице 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сумма | 2 | 4 | 6 | 7 | 8 | 11 | 12 | 15 | 18 | 19 | 19 | 22 | 22 | 22 | 23 | 23 |
| Время | 2.2E-06 | 3E-07 | 4E-07 | 8E-07 | 7E-07 | 4E-07 | 5E-07 | 7E-07 | 5E-07 | 7E-07 | 8E-07 | 9E-07 | 6E-07 | 9E-07 | 1.1E-06 | 5E-07 |
| Сумма | 27 | 29 | 31 | 31 | 31 | 34 | 34 | 36 | 36 | 37 | 37 | 38 | 40 | 40 | 43 | 45 |
| Время | 9E-07 | 4E-07 | 7E-07 | 0.000001 | 9E-07 | 1.4E-06 | 1.2E-06 | 1.6E-06 | 1.2E-06 | 5E-07 | 7E-07 | 4E-07 | 1.2E-06 | 7E-07 | 5E-07 | 1.1E-06 |
| Сумма | 46 | 46 | 48 | 48 | 50 | 51 | 51 | 52 | 52 | 54 | 54 | 55 | 56 | 62 | 77 | 77 |
| Время | 0.000001 | 1.2E-06 | 5E-07 | 1.2E-06 | 1.1E-06 | 1.3E-06 | 1.1E-06 | 5E-07 | 1.3E-06 | 0.000002 | 3E-07 | 0.000002 | 0.000002 | 1.3E-06 | 1.1E-06 | 7E-07 |

Таблица №1 – Зависимость времени работы алгоритма от суммы N + M

На основе данных из таблицы был построен график (изображение №8).

Изображение №8 – График зависимости времени от суммы N + M

Очевидно, что график времени довольно хаотичен, однако прослеживается некоторый восходящий тренд, примерно совпадающий с графиком N + M.

В целом, теоретическая оценка сложности данного алгоритма является очень грубой, так как количество переборов в процессе обработки может быть как очень большим, так и нулевым. То же самое касается завершающего перебора оставшихся символов шаблона. По сути, асимптотика может колебаться в диапазоне от O(N) до O(N ^ 2), поэтому теоретическая оценка является лишь неким усреднением.

Заключение

В ходе выполнения работы мною был реализован алгоритм сопоставления строки с шаблоном. Цель работы была достигнута путём тестирования на строках и шаблонах различной длины и состава. Полученные результаты не совпадают с теоретическими оценками сложности алгоритма, однако тому есть логическое обоснование.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения асимптотики и добавление большего количества преждевременных выходов в качестве исключений.

ПРИЛОЖЕНИЕ А

Листинг кода файла wildcard\_matching.cpp

#include <iostream>

#include <string>

#include <chrono>

bool isMatch(const std::string& s, const std::string& p) {

    int sLen = s.length();

    int pLen = p.length();

    int sIndex = 0, pIndex = 0;

    int starIndex = -1, matchIndex = -1;

    while (sIndex < sLen) { // проход по символам строки - O(N), где N - длина строки

        // Если символы совпадают или есть '?'

        if (pIndex < pLen && (s[sIndex] == p[pIndex] || p[pIndex] == '?')) {

            sIndex++;

            pIndex++;

        }

        // Если встречаем '\*', запоминаем текущее положение

        else if (pIndex < pLen && p[pIndex] == '\*') {

            starIndex = pIndex;       // запоминаем позицию '\*'

            matchIndex = sIndex;      // запоминаем позицию строки

            pIndex++;                 // переходим к следующему символу шаблона

        }

        // Если символы не совпадают, но есть предыдущее '\*'

        else if (starIndex != -1) {

            pIndex = starIndex + 1;  // возвращаемся к '\*' в шаблоне

            sIndex = ++matchIndex;    // увеличиваем позицию в строке

        }

        // Если ничего не подходит, возвращаем false

        else {

            return false;

        }

    }

    // Проверяем оставшиеся символы в шаблоне

    while (pIndex < pLen && p[pIndex] == '\*') { // проход по оставщимся символам шаблона - O(M) в худшем случае, где M - длина шаблона

        pIndex++;

    }

    // Проверяем на соответствие, если шаблон полностью обработан

    return pIndex == pLen;

    // Итоговая сложность:

    // О(N) + O(M) = O(N + M)

}

int main() {

    struct Test {

        std::string s;

        std::string p;

        bool expected;

    };

    Test tests[] = {

        {"a", "a", true},                        // Длина 1

        {"ab", "ab", true},                      // Длина 2

        {"abc", "a?c", true},                   // Длина 3

        {"abcd", "a\*d", true},                  // Длина 4

        {"abcde", "a\*e", true},                 // Длина 5

        {"abcdef", "a?c?e", true},               // Длина 6

        {"abcdefg", "\*defg", true},             // Длина 7

        {"abcdefgh", "a\*d?f?h", true},          // Длина 8

        {"abcdefghi", "abcdefgh\*", true},       // Длина 9

        {"abcdefghij", "a?c\*g?i?j", true},     // Длина 10

        {"abcdefghijk", "a\*b\*c\*d\*e\*k", true},  // Длина 11

        {"abcdefghijkl", "a\*c\*e\*g\*i\*k", true}, // Длина 12

        {"abcdefghijklm", "a?c?e\*i\*m", true},  // Длина 13

        {"abcdefghijklmn", "a\*e\*n", true},     // Длина 14

        {"abcdefghijklmno", "abc\*efg\*", true}, // Длина 15

        {"abcdefghijklmnop", "a\*d\*?g", true},  // Длина 16

        {"abcdefghijklmnopq", "ab?c\*e\*i\*k\*q", true}, // Длина 17

        {"abcdefghijklmnopqr", "a\*?k\*r", true}, // Длина 18

        {"abcdefghijklmnopqrs", "a\*e\*?g\*s", true}, // Длина 19

        {"abcdefghijklmnopqrst", "a?b\*c\*e?g\*h\*i\*b\*t", true}, // Длина 20

        {"abcdefghijklmnopqrstu", "abcdef\*g\*?", true}, // Длина 21

        {"abcdefghijklmnopqrstuv", "ab\*c\*d\*?u", true}, // Длина 22

        {"abcdefghijklmnopqrstuvw", "ab\*cd\*?w", true}, // Длина 23

        {"abcdefghijklmnopqrstuvwx", "a\*b\*c\*d\*?x", true}, // Длина 24

        {"abcdefghijklmnopqrstuvwxy", "?bcdefgh?j?y", true}, // Длина 25

        {"abcdefghijklmnopqrstuvwxyz", "a\*b\*c\*d\*e\*f\*g\*h\*i\*j\*k\*l\*m\*n\*o\*p\*q\*r\*s\*t\*u\*v\*w\*x\*y\*z", true}, // Длина 26

        {"abcdefghijklmnopqrstuvwxyzabc", "a\*?c\*e\*?a", true}, // Длина 27

        {"abcdefghijklmnopqrstuvwxyzabcd", "ab?\*cd", true}, // Длина 28

        {"abcdefghijklmnopqrstuvwxyzabcde", "abc\*d?e", true}, // Длина 29

        {"abcdefghijklmnopqrstuvwxyzabcdef", "abcdef\*g\*e", true}, // Длина 30

        {"abcdefghijklmnopqrstuvwxyzabcdefg", "abcde?g", true}, // Длина 31

        {"abcdefghijklmnopqrstuvwxyzabcdefgh", "abcd\*hi\*", true}, // Длина 32

        {"abcdefghijklmnopqrstuvwxyzabcdefghi", "abcdefgh?i", true}, // Длина 33

        {"abcdefghijklmnopqrstuvwxyzabcdefghij", "a?c\*e\*g\*i\*j", true}, // Длина 34

        {"abcdefghijklmnopqrstuvwxyzabcdefghijk", "a?c?g\*h\*j\*k", true}, // Длина 35

        {"abcdefghijklmnopqrstuvwxyzabcdefghijkl", "a\*e\*c\*?k\*?", true}, // Длина 36

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklm", "ab\*?e\*g\*", true}, // Длина 37

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmn", "aaaa\*?b\*?m", true}, // Длина 38

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmno", "abcd\*?g\*n", true}, // Длина 39

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmnop", "a?bc\*?f\*mnop", true}, // Длина 40

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopq", "abcdefg?\*q", true}, // Длина 41

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqr", "abcd\*f\*g?j", true}, // Длина 42

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrs", "ab?\*g\*?" , true}, // Длина 43

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrst", "a\*b\*c\*d\*?t", true}, // Длина 44

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuv", "a?b\*c\*d\*u", true}, // Длина 45

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvw", "abcd\*?w?v", true}, // Длина 46

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwx", "\*xyz", true},  // Длина 47

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxy", "abc\*d\*?y", true}, // Длина 48

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz", "abc\*dv\*gh\*o\*?", true}, // Длина 49

        {"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz0", "abcdefghijklmnopqrstuvwxyz\*", true} // Длина 50

    };

    for (const auto& test : tests) {

        auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

        bool result = isMatch(test.s, test.p);

        auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

        std::chrono::duration<double> duration = end - start;

        // std::cout << "Тест: s = \"" << test.s << "\", p = \"" << test.p << "\" - "

        //           << (result == test.expected ? "Совпадает!" : "Не совпадает.") << std::endl;

        // std::cout << duration.count() << std::endl;

        std::cout << test.p.length() << std::endl;

    }

    return 0;

}