Київський національний університет імені Тараса Шевченка

Факультет комп'ютерних наук та кібернетики

Лабораторна робота №2

З дисципліни “Системне програмування”

за темою “Теорія автоматів”

Виконала:

студентка 3-го курсу групи ТТП-32

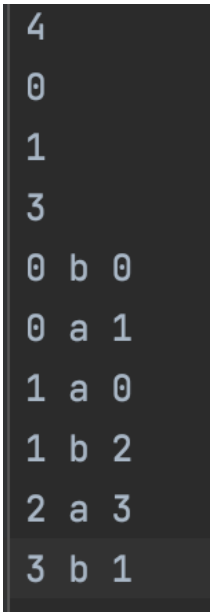
спеціальності "Інформатика"

Черечеча Катерина Сергіївна

Київ – 2023

Умова завдання: знайти всі слова (без періодичних фрагментів), що сприймаються заданим скінченним автоматом.

Отже, маємо текстовий файл, який містить скінченний автомат без виходів наступного вигляду:



де перша строка - кількість станів,
друга - початковий стан,
третя - кількість кінцевих станів
четверта - кількість приймаючих станів
усі наступні строки - переходи

Слова з періодичними фрагментами.
Оскільки кожен по-різному розуміє “періодичні фрагменти”, варто уточнити, що саме мається на увазі у виконаному завданні.
Під періодичним фрагментом розуміємо слово (string), яке за n кількість кроків можемо утворити з його складів (substring).
Наприклад, слово ababab має періодичні фрагменти, бо три повторення складу ab утворить слово ababab. Так само слово aabaab, babbab і т.д.
Слово ababba не містить періодичних фрагментів, бо ми його не утворимо з його складів за n кроків.

Результат роботи програми:

```
Слова без періодичних фрагментів, які приймає автомат:
Слово aba приймається автоматом
Слово aaaba приймається автоматом
Слово bbaba приймається автоматом
Слово baaaba приймається автоматом
Слово aababa приймається автоматом
Слово bbbaba приймається автоматом
Слово ababba приймається автоматом
Слово aaaaaba приймається автоматом
Слово bbaaaba приймається автоматом
Слово baababa приймається автоматом
Слово aabbaba приймається автоматом
Слово bbbbaba приймається автоматом
Слово bababba приймається автоматом
```

Реалізація програми:

```
struct Automaton {
    int states;
    int startState;
    std::vector<int> acceptingStates;
    std::unordered_map<int, std::unordered_map<char, int>> transitions;
};
```

Структура Automaton - це і є певний автомат, який задається текстовим файлом, які було наведено раніше.

Оскільки це текстовий файл, нам потрібно його зчитати. Його структура була описана вище.

```
Automaton readAutomatonFile(const std::string &fileName) {
    Automaton automaton;
    std::ifstream file(fileName);

    if (!file || file.peek() == std::ifstream::traits_type::eof()) {
        std::cerr << "Не вдалося відкрити файл" << std::endl;
        exit(1);
    }

    file >> automaton.states;
    file >> automaton.startState;

    int finalStates;
    file >> finalStates;
    automaton.acceptingStates.resize(finalStates);
    for (int i = 0; i < finalStates; ++i) {
        file >> automaton.acceptingStates[i];
    }

    int from, to;
    char transitionSymbol;
    while (file >> from >> transitionSymbol >> to) {
        automaton.transitions[from][transitionSymbol] = to;
    }

    file.close();
    return automaton;
}
```

Робимо це за допомогою функції readAutomatonFile.
На початку обов'язково визначаємо чи взагалі у нас є файл та чи не пустий він. У випадку якоїсь проблеми, у консоль буде виведено помилку "Не вдалося відкрити файл" та програма завершиться.

Далі зчитуємо кількість станів, початковий та заключний стани.

`std::vector<int> acceptingStates;`
використовуємо для збереження станів автомата, і за допомогою циклу for записуємо їх у цей вектор.

Також нам треба зчитати переходи, це і є наступний крок. Змінні from і to використовуються для зберігання початкового та кінцевого станів певного переходу. Змінна transitionSymbol зберігає символ переходу між станами. За допомогою циклу while ми зчитуємо усі наявні переходи. transitions - це двовимірна мапа, де ключами є пари (from, transitionSymbol), а значеннями є to.

Далі генеруємо слова та за допомогою функції acceptedWords перевіряємо, чи приймаються вони автоматом.

```
std::vector<std::string> generateWords(const Automaton &automaton) {
    std::vector<std::string> acceptedWords;

    for (int len = 1; len <= 7; ++len) {
        for (int i = 0; i < (1 << len); ++i) {
            std::string word;
            for (int j = 0; j < len; ++j) {
                if (i & (1 << j)) word += 'b';
                else word += 'a';
            }
            if (acceptWord( fa: automaton,  word: word)) {
                acceptedWords.push_back(word);
            }
        }
    }

    return acceptedWords;
}
```

```
bool acceptWord(Automaton fa, std::string word) {
    int currentState = fa.startState;
    for (const char& ch : word) {
        if (fa.transitions.at(currentState).find(ch) == fa.transitions.at(currentState).end()) {
            return false;
        }

        currentState = fa.transitions.at(currentState).at(ch);
    }


    for (int state : fa.acceptingStates) {
        if (currentState == state)
        {
            return !hasPeriodicFragment(word);
        }
    }
    return false;
}
```

Нам потрібно пройтися по символам кожного слова, і по переходам перевірити, чи воно приймається заданим автоматом. Метод find шукає символ у мапі переходів, і якщо він відсутній, повертає false, що означає, що слово не приймається.
В нашій мапі переходів ми беремо ключ для поточного стану і завдяки цьому визначаємо стан, в який ми перейдемо.
Після завершення цього циклу перевіряємо, чи є він кінцевим, якщо так, то ми викликаємо функцію hasPeriodicFragment, щоб перевірити слово на наявність періодичних фрагментів, якщо ні, функція повертає false - цей автомат не приймається.
Перевірка відбувається наступним чином:

```
std::vector<int> computePrefixFunction(const char* pattern) {
    int m = 0;
    while (pattern[m] != '\0') {
        ++m;
    }

    std::vector<int> prefixFunction(m,  x: 0);
    int k = 0;
    for (int q = 1; q < m; ++q) {
        while (k > 0 && pattern[k] != pattern[q]) {
            k = prefixFunction[k - 1];
        }
        if (pattern[k] == pattern[q]) {
            k++;
        }
        prefixFunction[q] = k;
    }
    return prefixFunction;
}
```

```cpp
bool hasPeriodicFragment(const char* word) {
    int n = 0;
    while (word[n] != '\0') {
        ++n;
    }

    std::vector<int> prefixFunction = computePrefixFunction(word);

    // Періодичний фрагмент буде присутній, якщо останній елемент префікс-функції ділить n на (n - (останній елемент))
    int lastPrefix = prefixFunction[n - 1];
    return lastPrefix > 0 && n % (n - lastPrefix) == 0;
}
```

Визначаємо довжину слова. Ініціалізуємо вектор prefixFunction нулями за довжиною. Далі проходимося по кожному символу слова, і зрівнюємо його з іншими наступними символами. При кожному співпадінні, збільшується індекс k. В кінці у функцію hasPeriodicFragment ми передаємо вектор, з якого повинні взяти останній елемент, якщо він більше нуля, і при діленні розміру слова на різницю розміру і останнього елементу не отримуємо остачі, то він має періодичну фрагментацію (ну і навпаки).
Якщо ж слово має періодичну фрагментацію, то воно не приймається і, відповідно, не виводиться.

[Посилання на GitHub]