Санкт-Петербургский государственный университет

Факультет прикладной математики - процессов управления

# Проект по КА

Разработал:

студент 233 группы Докиенко Д.А.

Проверил:

д.ф-м.н., проф. Матросов А.В.

Санкт-Петербург

2018

**1. Постановка задачи**

В ходе данной работы необходимо:

1. Разработать ДКА, распознающий цепочки заданного языка.
2. Разработать регулярное выражение, задающее данный язык.
3. Представить программу (код и исполняемый файл), которая во входной цепочке, принадлежащей заданному алфавиту, ищет все подцепочки языка.
4. Представить описание работы программы пункта c)
5. Разработать тестовые кейсы для программы пункта c)

4 вариант задания:

Алфавит:

Язык:

Строка начинается с префикса и заканчивается суффиксом , между которыми располагается последовательность символов в которой любая пара одинаковых символов разделяется символом .

Пример:

**2. ДКА и регулярное выражение**

Таблица переходов ДКА:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Регулярное выражение:

Т.к. в алфавите присутствует символ , для обозначения метасимвола (знак или) регулярных выражений используем символ

**3. Программа**

Данная программа написана на языке С++.

Код по файлам:

main.cpp

#include <iostream>

#include <string>

#include "Parser.h"

int main() {

// Читаем введенную пользователем в консоль строку

std::string procString = "";

std::cout << "Enter string:" << std::endl;

std::getline(std::cin, procString);

// Создаём объект класса parser и выводим найденные подстроки

// а также и количество

Parser parser(procString);

std::cout << "Found " << parser.getRelevantSubstringsNum() << " relevant substrings" << std::endl;

parser.printRelevantSubsrings();

// Пауза после вывода

system("pause");

return 0;

}

Parser.h

#ifndef PARSER\_H

#define PARSER\_H

#include <string>

#include <vector>

#include <utility>

#include <iostream>

class Parser {

public:

// Конструктор. Принимает обрабатываемую строку

// Вызывает метод searchSubstrings()

Parser(std::string procString);

// Вывод найденных релевантных подстрок в поток вывода

// в формате "substring\_pos: substring"

// где substring\_pos - позиция префикса + данной подстроки

// На вход подаётся поток вывода. По умолчанию - cout

std::ostream &printRelevantSubsrings(std::ostream &out = std::cout);

// Получить количество релевантных построк

size\_t getRelevantSubstringsNum();

private:

// Обрабатываемая строка

std::string procString\_;

// Номера позиций префикса + и суффикса + релевантных подстрок

std::vector<std::pair<size\_t, size\_t>> relevantSubstrings\_;

// Количество релевантных подстрок

size\_t relevantSubstringsNum\_;

// Поиск релевантных подстрок. Запускается в конструкторе

void searchSubstrings();

// Поиск плюсов, входящих в строку

// Возвращает количество найденных плюсов

// В аргумент-вектор записывает позиции найденных плюсов

size\_t searchPluses(std::vector<size\_t> &stringPluses);

// Проверка подстроки на соответсвие правилам языка

// Принимаем позиции первого + (префикса) и второго + (суффикса)

// Возвращаем соответствует строка или нет

bool checkSubstring(size\_t firstPlus, size\_t secondPlus);

};

#endif // PARSER\_H

Parser.cpp

#include "Parser.h"

Parser::Parser(std::string procString) : procString\_(procString), relevantSubstringsNum\_(0)

{

searchSubstrings();

}

std::ostream &Parser::printRelevantSubsrings(std::ostream &out) {

for (size\_t i = 0; i < relevantSubstringsNum\_; i++) {

out << relevantSubstrings\_[i].first + 1 << ": ";

out << procString\_.substr(relevantSubstrings\_[i].first, relevantSubstrings\_[i].second - relevantSubstrings\_[i].first + 1);

out << std::endl;

}

return out;

}

size\_t Parser::getRelevantSubstringsNum() {

return relevantSubstringsNum\_;

}

void Parser::searchSubstrings() {

// Ищем плюсы в подстроке

// Записываем их количество в plusesNum, а позиции в stringPluses

std::vector<size\_t> stringPluses;

size\_t plusesNum = searchPluses(stringPluses);

// Рассматриваем каждую подстроку от + до + (включительно)

// Если соответсвует - добавляем в relevantSubstrings\_

// и увеличиваем количество релевантных подстрок relevantSubstringsNum\_

for (size\_t i = 1; i < plusesNum; i++) {

if (checkSubstring(stringPluses[i - 1], stringPluses[i]) == true) {

relevantSubstrings\_.push\_back(std::pair<size\_t, size\_t>(stringPluses[i - 1], stringPluses[i]));

relevantSubstringsNum\_++;

}

}

}

size\_t Parser::searchPluses(std::vector<size\_t> &stringPluses) {

// Посимвольный разбор строки с счетчиком количества плюсов

size\_t plusesCounter = 0;

for (size\_t i = 0, stringSize = procString\_.length(); i < stringSize; i++)

if (procString\_[i] == '+') {

stringPluses.push\_back(i);

plusesCounter++;

}

return plusesCounter;

}

bool Parser::checkSubstring(size\_t firstPlus, size\_t secondPlus) {

// Упрощенная версия ДКА (см. Отчет по проекту КА)

// Имеет только два состояния, показывающие какой был предыдущий символ

// Состояния, отржающие предыдущий символ для знака минус упразднены

// вместо них используется срвнение символов

bool prevA = false, prevB = false;

// Читаем посимвольно подстроку кроме префикса + и суффикса +

// Ищем ошибки, а если их нет - возвращаем true

for (size\_t i = firstPlus + 1; i < secondPlus; i++)

switch (procString\_[i]) {

case 'a':

// Ошибка повторения символа a без разделяющего -

if (prevA == true) return false;

prevA = true;

prevB = false;

break;

case 'b':

// Ошибка повторения символа b без разделяющего -

if (prevB == true) return false;

prevB = true;

prevA = false;

break;

case '-':

// Ошибка идущих подряд минусов, или минуса, стоящего первым

if (prevA == false && prevB == false) return false;

// Ошибка минуса, стоящего между неодинаковыми символами

if (procString\_[i - 1] != procString\_[i + 1]) return false;

prevA = false;

prevB = false;

break;

default:

// Ошибка не принадлежащих алфавиту символов

return false;

}

return true;

}

4. Описание работы

Основная часть программы – класс Parser. Через него осуществляется разбор введенной строки и вывод найденных релевантных подстрок на консоль. Его описание содержится в Parser.h и Parser.cpp

Программа состоит из 3 файлов:

main.cpp:

Содержит функцию int main() – точку входа в программу. Она предназначена для взаимодействия с пользователем и работы с объектом класса Parser.

Parser.h:

Заголовочный файл, содержащий объявление класса Parser.

Содержит объявление следующих членов данных:

* string procString\_

Обрабатываемая строка.

* vector<pair<size\_t, size\_t>> relevantSubstrings\_

Массив пар значений позиций суффикса и префикса +, задающих подстроку, соответствующую языку.

* size\_t relevantSubstringsNum\_

Количество найденных подстрок, соответствующих заданному языку.

Parser.cpp

Содержит определение методов класса Parser:

* Parser(string)

Конструктор класса, принимающий строку, которую необходимо обработать в рамках данного задания.

* ostream &printRelevantSubstrings(ostream &)

Предназначена для вывода найденных подстрок.

* size\_t getRelevantSubstringsNum()

Предназначена для получения количества подстрок, соответствующих данному языку.

* void searchSubstrings()

Основной метод класса, который с помощью других методов осуществляет поиск подходящих подстрок.

* size\_t searchPluses(vector &)

Ищет позиции всех плюсов в строке, записывая их в переданный в качестве аргумента массив. Возвращает количество плюсов.

* bool checkSubstring(size\_t, size\_t)

Проверяет подстроку на соответствие заданному языку. Получает в качестве аргументов позиции префикса и суффикса , а возвращает true, если подстрока соответствует языку и false в обратном случае.

Ход работы программы:

1. Считываем строку, которую необходимо обработать в функции

int main().

1. Создаём объект класса Parser, передавая в качестве аргумента в конструктор Parser(string) эту строку.
2. В конструкторе вызывается функция void searchSubstrings().
3. В ходе её работы ищем все плюсы в введенной строке с помощью функции size\_t searchPluses(vector &). Затем проверяем каждую подстроку, начинающуюся и заканчивающуюся , используя

bool checkSubstring(size\_t, size\_t) ( не может быть вложен в другие ). Сохраняем подходящие подстроки в relevantSubstrings\_

1. С помощью функции size\_t getRelevantSubstringsNum() и

ostream &printRelevantSubstrings(ostream &) выводим полученный результат в форматированном виде на консоль

5. Тестовые кейсы

|  |  |
| --- | --- |
|  | Found 0 relevant substrings |
|  | Found 1 relevant substrings  1: |
|  | Found 0 relevant substrings |
|  | Found 0 relevant substrings |
|  | Found 0 relevant substrings |
|  | Found 1 relevant substrings  3: |
|  | Found 1 relevant substrings  1: |
|  | Found 0 relevant substrings |
|  | Found 3 relevant substrings  7:  13:  14: |
|  | Found 1 relevant substrings  7: |
|  | Found 2 relevant substrings  1:  15: |
|  | Found 4 relevant substrings  1: 2: 3:  4: |