Жуковский Анатолий, группа 696

Формальные языки и трансляции

Практикум №1

Регулярные выражения

В каждой задаче нужно реализовать на языке C++ или Python некоторый алгоритм обработки регулярных выражений. В каждой задаче аргументами являются строка в алфавите $\{a, b, c, 1, ., +, *\}$, а также некоторые дополнительные параметры. Если задача предполагает ответ "да/нет", то необходимо вывести YES в случае положительного ответа и NO — в случае отрицательного. В случае, если ответ является целым числом или словом, необходимо вывести это число или слово. В случае, если таких числа или слова не существует, необходимо вывести INF. В случае, если входная строка не является корректным регулярным выражением в обратной польской записи, необходимо выдать сообщение об ошибке. Дополнительные случаи оговорены непосредственно при формулировке задачи. В дальнейшем предполагается, что первым компонентом входа является регулярное выражение α в обратной польской записи, задающее язык L. Условия задач:

6. Даны α, буква x и натуральное число k. Вывести длину кратчайшего слова из языка L, содержащего префикс x^k.

Техническое задание

Реализовать на языке C++ или Python некоторый алгоритм обработки регулярных выражений. Входные данные подаются на *stdin*, выходные ожидаются на *stdout*. Аргументами программы являются строка в алфавите $\{a, b, c, 1, ., +, *\}$. Шаблон входных данных (разделяются пробелами):

1. Регулярное выражение α в обратной польской записи, задающее язык L
2. Буква x
3. Натуральное число k

Задача: вывести длину $length_{answer}$ кратчайшего слова из языка L, содержащего префикс x^k. **Доп. условия:** Если такого слова не существует, необходимо вывести «INF». Если входная строка не является корректным регулярным выражением в обратной польской записи, необходимо выдать сообщение об ошибке «$ERROR$». Шаблон выходных данных (один из вариантов):

- $length_{answer}$
- «INF»
- «$ERROR$»

Запуск решения и компиляция

Makefile проекта находится в папке **$/source/CmakeFiles/**. После сборки проекта выходной файл называется **source**. Входные данные принимаются в **stdin**, выходные подаются на **stdout**.

Формат входных данных: **regular-expression-no-spaces symbol-x number-k**.

Подробная инструкция по сборке и запуску проекта:

1. cd source/CmakeFiles
2. make
3. ./source

Общая идея решения

По регулярному выражению строится конечный недетерминированый автомат с однобуквенными переходами. Затем от начального состояния автомата запускается обход в глубину и вычисление на каждом шагу длины префикса вида x^l, $l <= k$, после чего по достижению $l = k$ запускается обход в ширину с поиском терминальных вершин. По приходу в терминальную вершину длина получившегося слова является кандидатом на ответ (и сравнивается с остальными претендентами).

Комментарии о реализации

Код включает в себя 2 основных и 1 побочный классы.

Первый класс "CAutomata"отвечает, как не странно, за построение автоматов. Для каждого вида операций есть свой конструктор. Автомат задаваемый примитивом вроде единственного символа (a, b или c) или же единицы (1) создается через специальный конструктор, принимающий данные символы **CAutomata::CAutomata(char symbol)**. Сумма или конъюнкция двух автоматов получается через конструктор **CAutomata::CAutomata (CAutomata *first, CAutomata *second, char operation)**, звезда клини через конструктор **CAutomata::CAutomata(CAutomata *old_automata, char operation)**.

Второй основной класс "CSolver"отвечает за парсинг регулярного выражения (и построения автомата на ходу) и запуск по данному автомату специфического для данной задачи DFS. Для запуска решения достаточно создать экземпляр классы CSolver и в качестве параметров его конструктора передать регулярное выражение, символ X и длину префикса k из условия. Теперь немного подробнее об устройстве класса CSolver:

- ParseExpression() вычитывает регулярное выражение уже сохранённое внутри экземпляра класса и по мере вычитывания на стеке строит автомат. В случае, если приходит не валидный символ, операция, или же под конец выполнения программы на стеке остается более одного автомата - программа выводит на экран $ERROR$ и заверается с кодом выхода от 1 до 4 (в зависимости от места, на котором она упала).

- CustomDFS() запускает обход в глубину по автомату, ранее построенному в ParseExpression. Для того чтобы обрабатывать циклы вводится понятие "оттенка серого вершины"(аналогия с покраской вершин в серый цвет в алгоритмах обходов графов и т.п.). В данном случае оттенков всего k (где k - длина префикса из условия). Т.е. вершина не будет обработана более k раз. Это гарантирует тот факт что мы не теряем никакие из решений и в то же время программа работает не слишком долго. В тот момент как на каком-то шаге мы насчитали префикс вида x^l длины k мы запускаем CustomDFS() от этой вершины.

- CustomBFS() запускает обход в ширину от данной вершины и ищет терминальные состояния. Как только ему встречается терминальное состояние он завершает данную итерацию обхода в шриину и выдвигает своего кандидата на ответ к задаче. В остальном это обычный бфс.