Практическое задание. Описание алгоритма.

Иванов Максим, Б05-223

Октябрь 2023

1 Проверка регулярного выражения на корректность

Сначала программа проверяет регулярное выражение на корректность. Это происходит в методе `is_regex_valid` класса `Regex`. Проверка проходит "симуляцией" парсинга строки:

1. Если очередной символ – буква, кладём её на стек.
2. Если символ – операция, берём одну (если звезда Клини, ноль) букву со стека. Только одну, потому что после выполнения операции над двумя операндами мы кладём результат обратно на стек. Следовательно, так как у нас "симуляция" парсинга, нет надобности удалять 2 числа.
3. Каждый раз мы проверяем, что на стеке лежит необходимое число операндов для совершения операции. Если это не так, вызываем исключение.
4. Если в конце на стеке лежит больше 1 символа, то у нас изначально было слишком много операндов – вызываем исключение.

Асимптотика: проходимся один раз по строке, что-то добавляем и убираем со стека конечное число раз, поэтому алгоритм работает за $O(n)$, где (здесь и дальше) n – длина регулярного выражения.

2 Построение НКА из регулярного выражения

НКА реализован следующим образом: массив состояний (вершин) хранится как поле класса автомата. Массивы переходов – поле каждого состояния.

НКА строится рекурсивно. Алгоритм почти не отличается от алгоритма из пункта 1 за исключением того, что при нахождении символа операции мы выполняем эту операцию и рекурсивно создаём автомат – об этом ниже. В стеке храним пары индексов создаваемых частей автомата: стартовая и конечная вершины.

База рекурсии: создаются 2 состояния, второе – завершающее. Переход – буква или пустое слово. В дальнейшем автомат строится аналогично алгоритму из этой статьи. Алгоритм корректен, потому что он повторяет рекурсивное правило построения НКА из регулярного выражения.

Асимптотика: аналогично пункту 1 алгоритм работает за $O(n)$.

3 Проверка, есть ли в языке слова, содержащие ровно k букв x

Для проверки воспользуемся алгоритмом обхода графа в глубину.

Если в момент, когда мы пришли в состояние, мы оказались в терминальной вершине, набрав необходимое число букв, это значит, что в языке существует нужное слово.

Возможна другая ситуация: мы ещё не набрали необходимое количество букв. Тогда пробуем перейти по переходам в другие состояния. Проверяем, что, если у нас есть возможность перейти по переходу с нужной буквой, букв не станет больше, чем надо. Иначе идём по переходам с другими буквами.

Также возможно, что количество букв меньше необходимого, но есть цикл, который может увеличить их количество. Для этого в коде есть массив `dfs_counter`. В элементе `dfs_counter[i]` хранится число необходимых букв, которые мы прошли на момент прохода вершины, или -1, если вершина не была посещена. Таким образом мы можем сравнить количество необходимых букв после прохода цикла с количеством букв до прохода.

Мы покрыли все возможности прохода по переходам, а значит, алгоритм корректен.

Асимптотика: DFS работает за $O(m+k)$, где m – число переходов, k – число вершин. В процессе построения НКА добавляются ϵ-переходы, их количество будет ограничено полиномом. В итоге весь алгоритм работает за полином.