Алгоритм Решения Задачи № 16

Задача: Даны α - регулярное выражение и слово $u \in \{a,\ b,\ c\}*$. Найти длину самого длинного подслова u, являющегося также подсловом некоторого слова в языке L.

Решение: основным юнитом решения является структура Language, которая хранит 4 верхнетреугольные булевые матрицы размера (N + 1)x(N + 1), где N = len(u).

1. has_prefix_(i, j) - содержит ли Language слово, для которого [i, j) (подслово u, начинающееся на i-том индексе и заканчивающееся на j-1-ом) является префиксом.
2. has_suffix_(i, j) - содержит ли Language слово, для которого [i, j) является суффиксом.
3. has_whole_(i, j) - содержит ли Language слово, которое совпадает с [i, j).
4. has_substr_(i, j) - содержит ли Language слово, для которого [i, j) является подсловом.

Во время парсинга регулярного выражения будем вычислять эти предикатные матрицы.

Когда рег.выражение состоит из одного символа, очевидно как заполнять эти матрицы. Они все единичные, если в рег.выражении только "1". Если регулярка не состоит из "1", то на главной диагонали всех матриц, кроме has_whole_ (ее диагональ нулевая, т.к. когда рег.выражение состоит не из "1", в L не будет пустого слова) лежат единички, а также если i-тый символ входного слова то (i, i + 1) элементы каждой матрицы -- единички.

Для удобства записи, пусть (has_substr_, has_whole, has_prefix_, has_suffix_) = (ss, w, p, s). Итак, мы поняли как выглядят предикатные матрицы для языков задаваемых рег.выражениями длины 1. Опишем, вычислять предикатные матрицы для объединения, произведения и итерации языков. Мы знаем, что:

1) Language(a + b) = Language(a) ∪ Language(b);
2) Language(a.b) = Language(a).Language(b);
3) Language(a*) = [Language(a)]*

Достаточно очевидно, что, если подслово [i, j) префикс некоторого слова в языке Language(a) ∪ Language(b), то оно является префиксом некоторго слова в Language(b) или в Language(a). В случае суффикса, совпадения [i, j) с некоторым словом в Language(a) ∪ Language(b) или подсловости рассуждения аналогичны. Значит, чтобы вычислить предикатные матрицы для языка Language(a) ∪ Language(b) достаточно просто поэлементо применять операцию ИЛИ для соответствующих матриц языков Language(a) и Language(b).

А также, очевидно, что:

$$[i,\ j)\ --\ \text{префикс некоторого слова в}\ Language(a).Language(b)\ <=>\ (\exists k \in N,\ i \leqslant k \leqslant j,\ Language(a).w(i,\ k) = true \wedge Language(b).p(k,\ j) = true) \tag{1}$$

Значит:

$$Language(a.b).p\ =\ Language(a).p\ +\ Language(a).w * Language(b).p \tag{2}$$

В умножении матриц переопределим оператор + как дизъюнкицю ИЛИ, а умножение двух булевых значений -- конъюнкция. Тогда, формула (2) действительно соответствует утверждению (1). Умножение матриц Language(a).w * Language(b).p означает, что мы рассматриваем все k, для всех i и j, что i⩽ k ⩽j и Language(a).w(i, k) = true ^ Language(b).p(k, j) = true.

Аналогичными рассуждениями выводим, что:

$$Language(a.b).s\ =\ Language(b).s\ +\ Language(a).s * Language(b).w \tag{3}$$

$$Language(a.b).w\ =\ Language(a).w * Language(b).w \tag{4}$$

$$Language(a.b).ss\ =\ Language(a).ss\ +\ Language(b).ss\ +\ +\ Language(a).s * Language(b).p \tag{5}$$

Итак, научились вычислять предикатные матрицы для объединения и умножения языков.

Осталось вычислить их для итерации языка.
рассмотрим Language(a*).w

$$Language\left(a^0\right).w = Language(1).w = E$$
$$Language\left(a^2\right).w = (Language(a).w)^2$$
$$.....$$
$$Language\left(a^n\right).w = (Language(a).w)^n$$
$$Language(a*).w = Language\left(a^0\right).w + Language(a).w + Language\left(a^2\right).w + ... +$$
$$+Language\left(a^n\right).w + ... = E + Language(a).w + (Language(a).w)^2 + ... +$$
$$+(Language(a).w)^n + ... \tag{6}$$

Если, рассмотреть предикатную матрицу A, как ориентированный граф, то согласно утверждению из теории графов A^n- все пути длины n. Значит матрица в формуле (6) имеет все пути длины от 0, до максимально возможной, т.е. это транзитивное замыкание графа Language(a).w. Это значит, чтобы вычислить матрицу Language(a*).w достаточно транзитивно замкнуть Language(a).w. Для вычисления остальных матриц воспользуемся Language(a*).w. Если, слово - префикс некоторого слова в Language(a*), то некоторый префикс этого слова может целиком лежать в языке $Language(1) \cup Language(a) \cup Language\left(a^2\right) \cup ... = Language(a*)$, а оставшая часть этого слова является префиксом какого-то слова в Language(a). Из этого получаем естественную формулу для Language(a*).p:

$$Language(a*).p = Language(a*).w * Language(a*).p$$

Аналогичными рассуждениями выводим формулы для s и ss:

$$Language(a*).s = Language(a).s * Language(a*).w$$
$$Language(a*).ss = Language(a).s * Language(a*).w * Language(a).p + Language(a).ss$$

Итак, научились вычислять предикатные матрицы для любого рег.выражения. Теперь, используя эти матрицы посчитаем ответ задачи.
После того, как тривиальными операциями push/pop в стек распарсили входное рег.выражение и одновременно вычислили все 4 предикатные матрицы, итерируемся по L(α).ss и находим в ней значение true с макисамльной разницей индекса стоблца и индекса строки, т.к. эта разница и дает нам длину максимального подслова удовлетворяющего условию задачи.
Оценим работу нашего алгоритма. пусть N -- длина входного слова u и R -- длина входного рег.выражения α. Тогда, каждое каждое пермножение матриц делается за O((N + 1)(N + 1)(N + 1)) = O(N^3). Сложение за O(N^2), последння итерация по матрице has_substr_ для посика ответа также за O(N^2), O(N^3 + N^2) = O(N^3), а такие операции выполняются максимально R раз ⟹ общая сложность алгоритм составляет O(R * N^3).