1 Задача

Дано регулярное выражение α и слово w. Нужно найти длину самого длинного подслова w, принадлежащего $L(\alpha)$.

Разрешены символы: $\{'a','b','c','1','.','+','*'\}$.

2 Решение

Будем решать данную задачу методом динамического программирования. Для каждого регулярного подвыражения выражения α (получающегося при его естественном разборе) будем поддерживать информацию о том, какими подсловами слова w может быть это подвыражение.

Определим действия для каждого подвыражения (по умолчаю все подслова выразить нельзя):

2.1 $\alpha \in \Sigma$

Пройдёмся по всем символам слова. Если этот символ совпадает с α, то это подслово мы можем выразить.

2.2 $\alpha = 1$

Можем выразить только пустое подслово.

2.3 $\alpha = lr+$

Подслово можно выразить, если оно выражается из l или r.

2.4 $\alpha = lr.$

Подслово sw можно выразить, если его можно разбить $sw = sw_l + sw_r$ (возможно, одно из них пустое), и при этом sw_l можно выразить из l, а sw_r можно выразить из r.

2.5 $\alpha = v*$

Подслово можно выразить, если оно разбивается на несколько, возможно пустых, подслов, каждое из которых выражается из v.

3 Реализация

Создадим класс DPVSubWord, который будет хранить в массиве информацию о возможности вывода подслова:

$c[i][j] = true \Leftrightarrow$ подслово, начинающееся в позиции i и имеющее последний символ в позиции j выводится из подвыражения

Этот класс будет иметь несколько конструкторов:

`DPVSubWord(const std::string& word, char ch)`

Инициалзируется от регулярного подвыражения состоящего только из одного символа - символа алфавита или '1'

`DPVSubWord(const DPVSubWord& L, const DPVSubWord& R, char operation)`

Пересчитывает состояния за $O(|w|^3)$ состояния из подвыражений L и R по операции operation.

`DPVSubWord(const DPVSubWord& V, char operation)`

Пересчитывает состояния за $O(|w|^3)$ состояния из подвыражения V по операции operation (пока что такая одна).

Пересчёт также осуществляется методом динамического программирования. Скопируем переходы из V, обьявим, что пустое слово выводится, после чего выполним код:

```
for (size_t i = 0; i < c.size(); ++i) {
    for (size_t j = i; j < c.size(); ++j) {
        for (size_t k = i; k + 1 <= j; ++k) {
            if (c[i][k] && c[k + 1][j]) {
                c[i][j] = true;
                mx = max(mx, {j - i + 1, i});
            }
        }
    }
}
```

Нетрудно заметить здесь алгоритм Флойда-Уоршелла в некоторой форме.

В итоге, обьявим функцию:

```
template<class DPV>
std::pair<int, int> regSubDP(const std::string& reg,
        const std::string& word)  {
```

Она для шаблонного класса вида $DPVSubWord$ выполняет пересчёт, разбирая польскую нотацию и вызывая нужные конструкторы в нужном порядке. В итоге она вернёт пару: длина максимального подслова, его начальный индекс.

Итого, алгоритм работает за $O(|w|^3 * |\alpha|)$, т.к. число вызванных конструкторов есть длина регулярного выражения, а самый медленный из них выполняет $O(|w|^3)$ действий. Впрочем, если количество операций '.' и '*' есть h, то время работы есть $O(|w|^2 * (\alpha - h) + |w|^3 * h)$.

4 Замечание

В коде можно найти класс $DPVSubWordSubReg$, который аналогичным образом решает задачу: Нужно найти длину самого длинного подслова w, являющегося подсловом слова, принадлежащего $L(\alpha)$. Возможно, он содержит баги.