**Практикум по форальным языкам** **Фомин Артем**
Вариант 11 Группа 795

## Задача

Даны $\alpha$, буква $x$ и натуральное число $k$. Вывести длину кратчайшего слова из языка $L$, содержащего ровно $k$ букв $x$.

## Идея решения

Возьмем дерево разбора данного регулярного выражения. Для каждой вершины будем хранить массив $d$. $d[m]$ – наименьшая длинна слова, соответствующего регулярному выражению в вершине, содержащего ровно $m$ букв $x$. $d[m] = inf$, если нет слов с $m$ буквами $x$, подходящими под регулярное выражение. Тогда пойдем в дереве от листьев к корню, находя $d$ для каждой вершины через $d$ потомков. Тогда ответ на задачу будет храниться в корне, который соответствует всему регулярному выражению, в $d[k]$ – наименьшей длине слова, содержащего ровно $k$ букв $x$.

## Алгоритм

Читаем выражение посимвольно. Прочитали букву – считаем, что это регулярное выражение, кладем в стек. Массив $d$ для буквы $x$ – $d[1] = 1$, все остальное – $inf$. Массив $d$ для буквы, отличной от $x$, - $d[0] = 1$, все остальное – $inf$.

Прочитали операцию – достаем два элемента(один для звездочки) из стека, кладем в стек новый элемент, соответствующий применению операции к элементам, пересчитывая его массив $d$ следующим образом:

1. Плюс: пусть со стека взяли $a$ и $b$, новое выражение – $c$. Тогда $c$ задает слова, подходящие под $a$ или под $b$. Соответственно, $d_c[i] = min(d_a[i], d_b[i])$

2. Умножить: пусть со стека взяли $a$ и $b$, новое выражение – $c$. Тогда $c$ задает слова, являющиеся конкатенацией слов из $a$ и $b$. Получить слово с $i$ буквами $x$ можно, взяв слово с $m$ буквами $x$ из $a$ и $i^\smile m$ буквами $x$ из $b$. Значит, $d_c[i] = min(d_a[m] + d_b[i - m])$ для $m$ от 0 до $i$.

3. Звездочка: мы можем несколько раз повторить наше выражение. Во-первых, можно его вообще не писать, поэтому $d[0] = 0$. Далее с помощью динамического программирования пересчитаем массив $d$. На $k$-том шаге попробуем улучшить $d[2 \cdot k], d[3 \cdot k], \ldots,$ используя значение $d[k]$: $d[l \cdot k] = min(d[l \cdot k], d[k] \cdot l)$. Заметим, что на $k$-том шаге значение $d[k]$ уже пересчитано правильно, потому что если оно могло быть улучшено, это произошло на предыдущих итерациях.

## Реализация

Создадим класс `Expression`, соответствующий регулярному выражению. Но так как само выражение нам в алгоритме не интересно, будем хранить в классе только массив $d$ и $k$, чтобы понимать, до какого момента нужно заполнять массив $d$.

В классе перегрузим операторы сложения и умножения как описано в алгоритме для операций «плюс» и «умножить». Определим функцию `star()`, пересчитывающую массив $d$ в случае нахождения звездочки.

Определим конструктор, принимающий $k$, который заполняет массив $k + 1$ элементами(индексы от 0 до $k$ включительно), равными $inf$ каждый. Если буква равна $x$, то $d[1] = 1$, иначе $d[0] = 1$.

Также определим функцию `ans()`, которая будет давать ответ на вопрос задачи – наименьшую длину слова, содержащего ровно $k$ букв $x$. Или $inf$, если такого слова не существует.

Теперь будем читать выражение. Нашли букву – создаем новый `Expression(k)`, правильно задаем $d$, в зависимости от того, нашли мы $x$ или другую букву. Кладем ее в стек. Нашли операцию – достаем нужное количество элементов из стека, кладем на верх стека `Expression`, равный операции, примененной к этим элементам. В конце работы в стеке останется один `Expression`, вызываем для него `Expression.ans()`, получаем ответ.

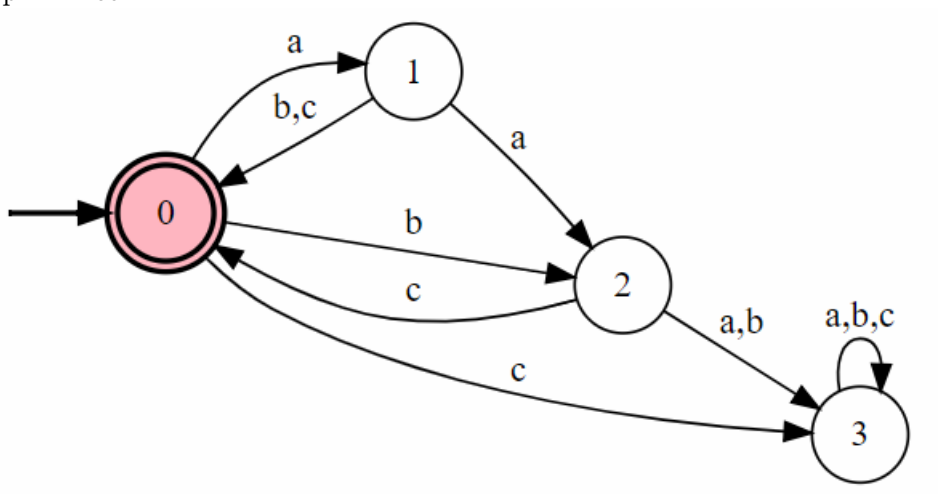
## Время работы

- Создание нового элемента – $O(1)$
- Применить плюс – $O(k)$
- Применить умножить – $O(k^2)$
- Применить звездочку – $O(k^2)$

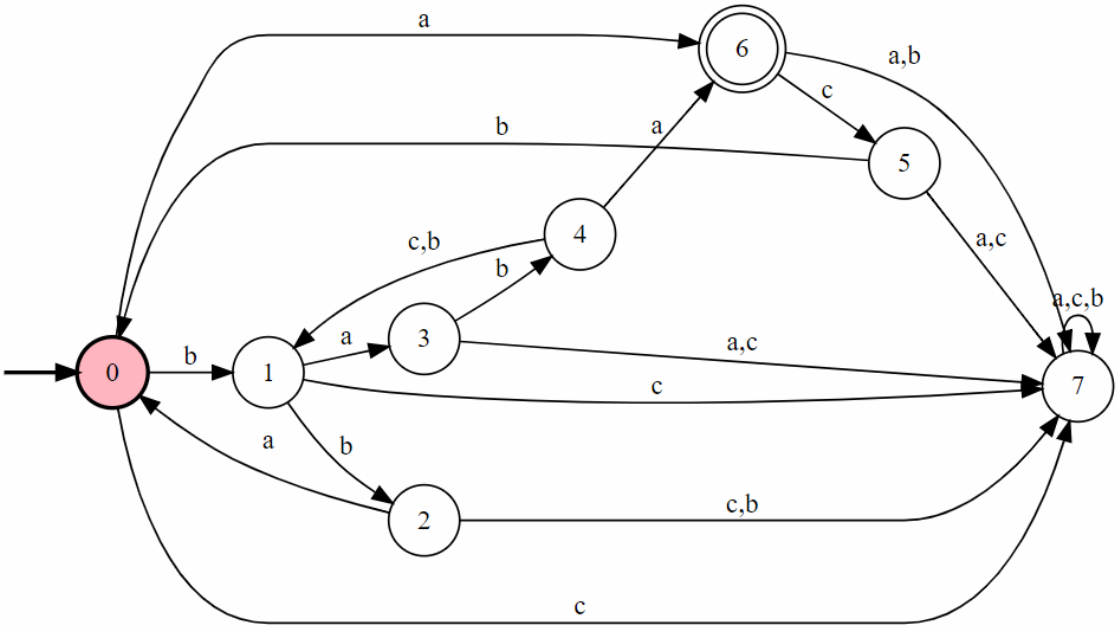
Пусть длина регулярного выражения $N$. Тогда операций в нем может быть $O(N)$, все из которых могут оказаться умножениями или звездочками. Тогда в худшем случае время работы – $O(N \cdot k^2)$

## Автоматы

Первый тест



Второй тест




## Тесты лектора

В первом тесте у лектора неправильный ответ. Было успешно проверено руками, что правильный ответ действительно 4, как и указывает программа.

Есть функция test, в которой тесты проверяются assert'ами.

## Код

```
import sys


inf = 123456789

class Expression:
    # структура, хранящая все, что нам нужно знать про регулярные выражения
    # см. описание решения
    d = []
    k = 0
    def __init__(self, k = 0):
        self.k = k
        self.d = []
        for i in range(k + 1):
            self.d.append(inf)
    def __add__(self, other):
        ans = Expression(self.k)
        for i in range(self.k + 1):
            ans.d[i] = min(self.d[i], other.d[i])
        return ans
```

```
    def __mul__(self, other):
        ans = Expression(self.k)
        for i in range(self.k + 1):
            for m in range (i + 1):
                ans.d[i] = min(ans.d[i], self.d[m] + other.d[i - m])
        return ans
    def star(self):
        self.d[0] = 0
        for l in range(1, self.k):
            for i in range(1, self.k - l + 1):
                self.d[l + i] = min(self.d[l + i], self.d[l] + self.d[i])
    def ans(self):
        if self.d[self.k] < inf:
            return str(self.d[self.k])
        else:
            return "inf"


def solve(string, x, k):
    stack = []
    for c in string:
        # разбираем регулярное выражение с помощью стека
        if c == "+":
            # достаем два из стека, сумму кладем обратно
            exp1 = stack.pop()
            exp2 = stack.pop()
            stack.append(exp1 + exp2)
        elif c == ".":
            # достаем два из стека, произведение кладем обратно
            exp1 = stack.pop()
            exp2 = stack.pop()
            stack.append(exp1 * exp2)
        elif c == "*":
            # Достаем из стека, делаем звездочку, кладем обратно
            exp = stack.pop()
            exp.star()
            stack.append(exp)
        else:
            #прочитали символ, кладем на стек
            exp = Expression(k)
            if c == x:
                exp.d[1] = 1
            else:
                exp.d[0] = 1
            stack.append(exp)
    exp = stack.pop() # регулярное выражение целиком
    return exp.ans()

def main():
```

```
    input_file = sys.stdin # можно подставлять файлы
    output_file = sys.stdout

    str1 = input_file.read()
    str2 = str1.split(" ")
    string = str2[0] # регулярное выражение
    x = str2[1] # буква x
    k = int(str2[2]) # количество k

    output_file.write(solve(string, x, k)) # печатаем ответ
    input_file.close()
    output_file.close()

def test():
    assert solve("ab+c.aba.*.bac.+.+*", 'b', 2) == '4'
    assert solve("acb..bab.c.*.ab.ba.+.+*a.", 'b', 3) == '7'

if __name__ == "__main__":
    test()
    main()
```