Домашнее задание № 2

Гунаев Руслан, 776 группа

27 февраля 2019 г.

1.

Из курса ТФС мы знаем, что язык

$$L = SelfReject = \{< M > | M \text{ отвергает } < M >\}$$

, где M – машина Тьюринга, а $< M >$ – ее описание, не является разрешимым, а значит он заведомо не принадлежит классу P.

2.

Так как языки L_1, L_2 принадлежат классу P, то для любого слова ω можно за полиномиальное время проверить принадлежность или не принадлежность соответствующим языкам.

1)

Покажем, что класс P замкнут относительно объединения.

Возьмем слово x, за полиномиальное время проверим принадлежность языкам, если хотя для одного языка ответ будет да, то слово принадлежит объединению, заметим, что мы затратили полиномиальное время.

2)

Покажем, что класс P замкнут относительно пересечения.

Проделаем аналогичные действия, но теперь если слово принадлежит сразу обоим языкам, то слово принадлежит и пересечению, а значит затратим снова полиномиальное время, иначе ответ нет.

3)

Покажем, что класс P замкнут относительно дополнения.

Проверим приналежность слова языку, если принадлежит, то будем выводить ответ нет, если не принадлежит – да. Итого так же полиномиальное время.

4)

Проверим для конкатенации языков.

Так как оба языка принадлежат классу полиномиальных языков, то проверка для любого слова принадлежности занимает полиномиальное время.

На вход подается некоторое слово ω, $|\omega| = n$. Будем делить наше слово на два слова так, что $\omega = \omega_1\omega_2$. Сначала $\omega_1 = \varepsilon, \omega_2 = \omega$. Алгоритм будет работать, пока либо не получим ответ да, либо пока $\omega_1 \neq \omega$. На каждом шаге нашего алгоритма будем проверять принадлежность ω_1 языку L_1 и ω_2 языку L_2. Алгоритм отвечает да, если оба подслова принадлежат соответственным языкам. Заметим, что в худшем случае, мы совершим $2n$ проверок принадлежности. Итого алгоритм займет время $O(n^{1+c})$, а значит тоже является полиномиальным.

5)

Проверим аналогично для итерации языка.

Заметим, что если входное слово можно разбить на подцепочки, такие, что каждая из них принадлежит языку, то входное слово будет принадлежать итерации. Пусть на вход подается слово ω.

Запустим цикл по длине слова(i). Создадим массив M, в котором будут лежать позиции слова, где могут заканчиваться слова, лежащие в языке.

Внутри этого цикла запустим еще один цикл, который будет пробегаться по нашему массиву и брать каждый раз элемент j.

Если $\omega[j, i] \in L$, тогда i добавим в M. Это и будет очередная подцепочка.

Заметим, что если для какого-то $k \in M$ и $i == n$ верно , что $\omega[k, i] \in L$, то это означает, что данное в условии слово разбивается на непересекающие подцепочки, каждая из которых принадлежит языку, а это означает, что слово принадлежит итерации языка. Иначе слово не принадлежит итерации.

Заметим, что в данном алгоритме присутствует вложенный цикл, на каждом шаге которого мы проверяли принадлежность некоторой подцепочки языку. В худшем случае массив M на каждом шаге будет состоять максимум из n элементов, а значит итоговая асимптотика данного алгоритма $O(n^{c+2})$.

3.

Построим аналогично 4 задаче НКА по РВ. Далее будем проверять принадлежит ли входное слово полученному НКА.

Будем идти по слову и каждой букве сопоставлять множество состояний, в которые можно попасть из попасть из предыщих.

Пусть мы находимся в начале слова и в начальном состоянии НКА. Прочитав эту букву можем попасть в новые состоянии(в том числе по пустому переходу). Далее находясь в этом множестве будем строить новое множество для следующей буквы, состоящее из новых состояний, куда мы можем попасть по этой букве. Таким образом, для каждой буквы слова в результате работы этого алгоритма получим множества из состояний автомата. Заметим, что мощность каждого такого множества не превосходит количесвто состояний автомата. Если в множестве, соответствующему последней букве есть конечное состояние, тогда слово принимается автоматом, иначе нет.

Оценим сложность алгоритма:

Для построения НКА потратим линейное время. В силу оценки на мощность каждого множества, получим оценку на время, всего букв $|\omega|$, а значит

$$F(n) = O(|\omega| + |\omega| \cdot |r|) = O(|\omega| \cdot |r|)$$

4.

В первую очередь построим два НКА по данным РВ. Далее применим алгоритм прямого произведения двух автоматов. Далее запустим алгоритм поиска в глубину для получившегося автомата, он будет иметь одно конечное состояние в силу того, что каждый из двух полученных НКА будет иметь по одному финальному состоянию, а в результате произведения так же получим автомат только с одним финальным состоянием, так как нам нужно пересечение автоматов.

Запустим на итоговом автомате алгоритм поиска в глубину для начальной вершины(в данном случае при обходе в глубину мы не будем смотреть по каким символам происходит переход от одной вершины до другой, будет достаточно перехода хотя бы по одному символу алфавита или же перехода по пустой цепочке, если же есть переходы сразу по нескольким символам из одной вершины в другую, будем считать, что он один), если в результате этого алгоритма мы сможем добраться из начальной вершины до конечной, то язык, порождаемый этим автоматом не пуст, иначе пуст.

Алгоритм построения НКА по РВ и оценка его сложности.

Данный алгоритм подразумевает собой разделение регулярного выражения на элементарные подвыражения, такие как части выражения с конкатенацией, объединением или итерацией, после выполнения алго-

ритма для каждой части следует их соединение $\varepsilon-$переходом. Точное построение автомата можно найти в книге Серебрякова В.А на странице 55.

Если мы видим конкатенацию, то просто добавляем новое состояние и соответствующий переход.

Если видим объединение, то добавляем новое состояние и соединяем переходом по пустой цепочке с каждым из тех частей автомата, которые нужно объединить.

В случае итерации строятся два новых состояние, одно на данном этапе будет начальным, другое конечным, из начального идет пустой переход в конечное, из начального состояния идет пустой переход к началу нужного участка автомата, из конца этого участка идет пустой переход в конечное состояние, так же конец и начало соединияются пустый переходом.

Добавление нового состояния или перехода означает действие алгоритма, таким образом, данный алгоритм занимает линейное время.

Прямое произведение автоматов и его оценка.

Получили два НКА с одним финальным состоянием. Описания автоматов также ограничены длиной входных регулярных выражений. В результате произведения мы получим $|N_1| \cdot |N_2|$ вершин, где N_1, N_2 количество состояний в автоматах. Построим переходы.

$$\delta(< q1, q2 >, c) = \{< \delta_1(q1, c), \delta_2(q2, c) >, < \delta_1(q1, \varepsilon), \delta_2(q2, c) >, < \delta_1(q1, c), \delta_2(q2, \varepsilon) >\}$$

Где $\delta, \delta_1, \delta_2$ – функции переходов в новом, первом, втором автоматах соответственно, $c \in \Sigma \cup \{\varepsilon\}$. Видим, что построение данного автомата занимает время $O(|r_1| \cdot |r_2|)$

Итоговая асимптотика: $O(|r_1| \cdot |r_2|)$.

5.

Пусть на вход подается некоторое число n. Запустим алгоритм бинарного поиска. Нам нужно сказать, представимо ли число $2n = p$ в виде произведения двух последовательных чисел. Есть отрезок от 1 до $2n$. На каждом шаге цикла проверяем $\lfloor p/2 \rfloor (\lfloor p/2 \rfloor + 1) ? 2n$, если первое больше, то делаем аналогично на отрезке $[\lfloor p/2 \rfloor; 2n]$, иначе на $[1; \lfloor p/2 \rfloor]$ Понятно, что такой алгоритм займет $O(\log n)$, а значит он полиномиальный от длины входного числа.

Таким образом, язык принадлежит классу P.

6.

1)

Язык графов, содержащих цикл.

Дан граф (V, E).

Воспользуемся алгоритмом поиска в глубину,который работает за $O(|V|+|E|)$. Модифицируем его следующим образом.

Каждый раз, обрабатывая новую вершину, проверяем является ли она смежной с какой-нибудь уже обработанной вершиной. Если да, то мы нашли цикл. Если же в графе был хотя бы один цикл, то мы совершенно точно его найдём, так как алгоритм поиска в глубину строит остовное дерево, а если мы можем добавить в дерево новое ребро, отличное от всех уже существующих, соединив две его вершины, то мы замкнём цикл. Можно дать грубую оценку сложности алгоритма, используя то, что проверок на ацикличность будет неболее, чем количество вершин в квадрате. Итоговая оценка $O(|V|^2(|V| + |E|))$.

2)

Язык двудольных графов.

Покрасим одну вершину произвольно, дальше будем красить соседние с уже покрашенными согласно условию, пока не придём к противоречию или не покрасим всю компоненту связности. В последнем случае продолжим тем же алгоритмом, пока все компоненты не кончатся. Если в какой-то момент случилось противоречие, значит граф содержит нечётный цикл, а тогда он не является двудольным.

3)

Язык связных графов.

Дан граф (V, E).

Будем от каждой вершины графа запускать алгоритм поиска в глубину. Модифицируем этот алгоритм так, что после завершения алгоритма он будет выдавать количество посещенных вершин. Если нашлась вершина, для которой количество вершин равно $|V|$, то граф связный.

Язык принадлежит классу P, так как асимптотика алгоритма равняется $O(|V|(|V| + |E|))$

4)

Язык деревьев.

Дерево — это связный и ациклический граф. Заметим, что данный язык представляет из себя пересечение двух языков. Языка связных графов и языка ациклических графов, который есть дополнением графов, содержащих циклов. А значит, в силу второго задания, данный язык принадлежит классу P.

7.

Для начала воспользуемся условием того, что все значения положительные, таким образом, для умножения и сложения чисел нам достаточно будет умножать и складывать только последние k цифр этих чисел. Соответсвенно в памяти будут храниться не сами числа, а только их последние цифры.

Заведем матрицу и вектор соответсвтущий каждому элементу последовательности.

$$A = \begin{pmatrix} a_1 & a_2 & \dots & a_{k-1} & a_k \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}, \mathbf{F_n} = \begin{pmatrix} F_n \\ F_{n-1} \\ \vdots \\ F_{n-k+1} \end{pmatrix}$$

Заметим, что

$$\mathbf{F_{n+1}} = A\mathbf{F_n} \Rightarrow \mathbf{F_{n+t}} = A^t\mathbf{F_n}$$

$$\mathbf{F_k} = \begin{pmatrix} F_k \\ F_{k-1} \\ \vdots \\ F_1 \end{pmatrix}$$

А значит,

$$\mathbf{F_{2^n}} = A^{2^n-k}\mathbf{F_k}$$

Оценим асимптотику алгоритма. В первую очередь мы должны возвести матрицу в степень. Очевидно, что возведение матрицы размером $k \times k$ в степень $p = 2^n - k$, где каждый элемент будет длиной не больше k, займет по времени

$$O(\log p \cdot k^{\log_2 7} \cdot k^{\log_2 3}) = O\left(\frac{n \cdot k^{\log_2 21}}{\log k}\right).$$

Возведение в степень будет аналогично обычному возведению числа в степень, за исключением того, что будут умножаться матрицы, для этого используем алгоритм Штрассена. Далее умножим полученную матрицу на $\mathbf{F_k}$ и выведем $\mathbf{F_k}[0]$. Это займет также полиномиальное время, а именно $O(k^{\log_2 21})$.

Итоговая асимптотика

$$T(n,k) = O\left(\frac{n \cdot k^{\log_2 21}}{\log k} + k^{\log_2 21}\right)$$

8.

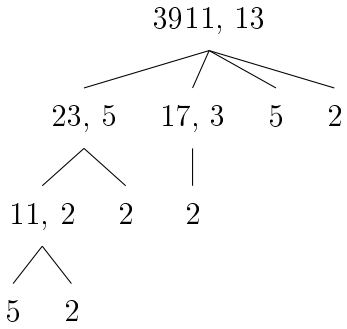
В качестве сертификата можем привести множество, которое состоит из непересекающих подслов данного слова, которые в конкатенации дают все слово и каждое из них принадлежит языку, это мы можем проверить за полином, так как $L \in \mathbf{NP}$.

Пример: пусть слово $\omega = \omega_1\omega_2...\omega_k$, тогда в качестве сертификата возьмем множество $\Omega = \{\omega_1, ..., \omega_k\}$ Заметим, что сертификат линейный, так как в сумме эти подслова дают само слово, а значит $L^* \in \mathbf{NP}$.

9.

1)

Построим сертификат простоты числа $p = 3911, g = 13$.



$$5^{22} \equiv 1(mod 23)$$
$$3^{16} \equiv 1(mod 17)$$
$$2^{10} \equiv 1(mod 11)$$

2)

Покажем, что сертификат действительно является полиномиальным. Приведем грубую оценку.

Сертификатом является дерево. В каждой вершине дерева может стоять число и образующий элемент. Двоичная запись числа равняется $O(\log p)$. В корне дерева стоит исходное число, далее из корня исходят вершины к потомкам, количество которых равно $O(\log p)$. Высота дерева также равна $O(\log p)$. Общая длина всех чисел на каждом уровне дерева не превосходит длину исходного слова, так как $\log p \geqslant \log k_1 k_2 ... k_t = \log k_1 + ... \log k_t$. Итого получим оценку на длину сертификата $O(\log^2 p)$.

3)

В силу того, что сертификатом является дерево, для его обхода будем использовать алгоритм поиска в ширину. Сначала мы находимся в корне дерева, алгоримт должен проверить дейстительно ли элемент g является образующим. Если да, то спускаемся к потомкам и для каждого из них делаем аналогичную проверку.