Рябых Владислав, Б05-905

Алгоритмы и модели вычислений. Домашнее задание № 2

Задача 1. Докажите следующие свойства полиномиальной сводимости:
(i) Рефлексивность: $A \leqslant_p A$; транзитивность: если $A \leqslant_p B$ и $B \leqslant_p C$, то $A \leqslant_p C$;
(ii) Если $B \in \mathcal{P}$ и $A \leqslant_p B$, то $A \in \mathcal{P}$;
(iii) Если $B \in \mathcal{NP}$ и $A \leqslant_p B$, то $A \in \mathcal{NP}$.

Решение. Для начала запишем определение полиномиальной сводимости: $A \leqslant_p B \Leftrightarrow \exists f$ — функция, вычислимая за полином от длины входа, такая что $x \in A \Leftrightarrow f(x) \in B$

1. - Рефлексивность очевидна: в качестве f берём тождественную функцию id, тогда $x \in A \Leftrightarrow id(x) = x \in A \Rightarrow A \leqslant_p A$
   - Транзитивность: если $A \leqslant_p B$ и $B \leqslant_p C$, то $\exists f, g$, вычислимые за полином, такие что $x \in A \Leftrightarrow f(x) \in B \Leftrightarrow g(f(x)) \in C$. Заметим, что функция $g(f(x))$ вычисляется за $poly(|f(x)|) = poly(poly(|x|)) = poly(|x|)$, так как полином от полинома также является полиномом. Таким образом $x \in A \Leftrightarrow g(f(x)) \in C \Rightarrow A \leqslant_p C$

2. - $A \leqslant_p B \Rightarrow \exists f$ — функция, вычислимая за полином, такая что $x \in A \Leftrightarrow f(x) \in B$; таким образом существует детерминированная МТ M_1, вычисляющая функцию $f(x)$ за $poly(|x|)$
   - $B \in \mathcal{P} \Rightarrow \exists M_2$ — детерминированная МТ, распознающая язык B за полином.

   Построим теперь детерминированную МТ M, распознающую язык A за полином: на входе x сначала моделируется работа M_1, то есть за полином от длины входа вычисляется функция $f(x)$, а затем моделируется работа M_2 на входе $f(x)$

   Таким образом детерминированная МТ M распознаёт язык A за $poly(|x|)+poly(|f(x)|) = poly(|x|) + poly(poly(|x|)) = poly(|x|)$, так как полином от полинома также является полиномом и сумма полиномов есть полином. Таким образом язык A распознаётся за полиномиальное от длины входа время детерминированной МТ M, следовательно $A \in \mathcal{P}$

3. Доказательство аналогично предыдущему пункту с точностью до замены детерминированной МТ на недетерминированную МТ, приведём его для полноты:

   - $A \leqslant_p B \Rightarrow \exists f$ — функция, вычислимая за полином, такая что $x \in A \Leftrightarrow f(x) \in B$; таким образом существует детерминированная МТ M_1, вычисляющая функцию $f(x)$ за $poly(|x|)$
   - $B \in \mathcal{P} \Rightarrow \exists M_2$ — недетерминированная МТ, распознающая язык B за полином.

   Построим теперь недетерминированную МТ M, распознающую язык A за полином: на входе x сначала моделируется работа M_1, то есть за полином от длины входа вычисляется функция $f(x)$, а затем моделируется работа M_2 на входе $f(x)$

Таким образом недетерминированная МТ M распознаёт язык A за $poly(|x|)+poly(|f(x)|) = poly(|x|) + poly(poly(|x|)) = poly(|x|)$, так как полином от полинома также является полиномом и сумма полиномов есть полином. Таким образом язык A распознаётся за полиномиальное от длины входа время недетерминированной МТ M, следовательно $A \in \mathcal{NP}$

Задача 2. Докажите, что следующие языки принадлежат классу $\mathcal{P}$. Считайте, что графы заданы матрицами смежности.
(i) Язык двудольных графов, содержащих не менее 2018 треугольников (троек попарно смежных вершин);
(ii) Язык несвязных графов без циклов;
(iii) Язык квадратных $\{0;1\}$-матриц порядка $n \geqslant 3000$, в которых есть квадратная подматрица порядка $n - 2018$, заполненная одними единицами.

Решение. 1. Выберем произвольную тройку вершин. Исходный граф двудолен, по принципу Дирихле в одной из долей окажется по крайней мере 2 вершины. Они не будут смежны в силу двудольности исходного графа и, следовательно, тройка вершин не будет образовывать треугольник. Таким образом мы доказали, что в двудольном графе в принципе не может быть треугольников, а следовательно язык из условия задачи пуст: $L = \varnothing$. А пустое множество, очевидно, лежит в $\mathcal{P}$: машине Тьюринга достаточно просто на любом входе x выдавать ответ 0, что даже при условии полного прочтения входа x (хотя это и необязательно), работает за $poly(|x|)$

2. Построим полиномиальный алгоритм: для начала нам необходимо найти все компоненты связности исходного графа — можем сделать это с помощью поиска в глубину: сначала запускаем DFS из произвольной вершины, помечаем её, поиск обходит все достижимые (также помечаем их) и находит тем самым первую компоненту связности, затем запускаем DFS из произвольной не помеченной раннее вершины и т.д, пока не будут найдены все компоненты связности исходного графа (если компонента связности всего одна, то граф связен и алгоритм останавливается, выводя 0)

   Теперь необходимо проверить на ацикличность каждую из компонент связности графа: с этим также справляется DFS — в каждой из компонент связности запускаем серию обходов DFS следующим образом: из каждой вершины, которая ещё никак не была помечена запускается поиск в глубину, который при входе в вершину красит её в красный цвет, а при выходе в синий. Таким образом, если в какой-то момент DFS пытается пройти в красную вершину (которая при этом не является предком вершины, в которой мы находимся в данный момент времени), то цикл найден и алгоритм останавливается, выводя 0. Если же оказалось, что все вершины помечены и не было найдено ни одного цикла, то граф является ацикличным, а так как алгоритм вообще дошёл до этой точки, то граф является ещё и несвязным, то есть алгоритм выводит 1

   Таким образом МТ, реализующая построенный алгоритм, распознаёт язык из условия задачи. При этом алгоритм (а следовательно и МТ) работает за полиномиальное время,

так как вся его работа заключается в использовании $O(|V|)$ поисков в глубину, которые сами по себе работают за линию. Следовательно язык из условия задачи $L \in \mathcal{P}$

3. Построим полиномиальный алгоритм: с помощью динамического программирования найдём k — порядок максимальной подматрицы, заполненный одними единицами, а потом просто сравним это число с $n - 2018$

   Пусть $A = (a_{ij}), i \in \overline{1 \dots n}, j \in \overline{1 \dots n}$ — матрица порядка n, поданная на вход. Для простоты добавим ей *pad* (обёртку) из нулей слева и сверху (то есть теперь A — матрица порядка $n + 1$, причём $a_{i,j} = 0$ при $i = 1 \vee j = 1$). Теперь будем преобразовывать её элементы следующим образом:

Algorithm 1 Преобразование матрицы A

```
for i ∈ 2...n+1 do
    for j ∈ 2...n+1 do
        if a_{i,j} ≠ 0 then
            a_{i,j} ← min(a_{i-1,j}, a_{i,j-1}, a_{i-1,j-1}) + 1
```

   После завершения алгоритма в элементе a_{ij} матрицы A содержится число, равное порядку наибольшей подматрицы из единичек, имеющей клетку i, j своим правым нижним углом.

   Приведём пример работы алгоритма:

$$\begin{matrix} & & \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{matrix} \xrightarrow{pad} \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{matrix} \xrightarrow{\text{alg } 1} \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{matrix}$$

   Таким образом, мы находим k — порядок максимальной подматрицы, заполненный одними единицами, а потом просто сравниваем это число с порядком $(n - 1) - 2018$, $(n - 1$ объясняется тем, что в ходе работы алгоритма мы увеличиваем порядок матрицы на 1).

   Алгоритм работает полиномиальное время, следовательно L — язык из условия задачи лежит в $\mathcal{P}$

Задача 3. Корректно ли следующее рассуждение? Язык $3 - \mathsf{COLOR}$ сводится к языку $2 - \mathsf{COLOR}$ следующим образом: добавим новую вершину и соединим её со всеми вершинами исходного графа. Тогда новый граф можно окрасить в 3 цвета тогда и только тогда, когда исходный можно было окрасить в 2 цвета.
При положительном ответе приведите обоснование записанной сводимости. В противном случае — укажите явное место ошибки.

Решение. Автор данного рассуждения пишет, что $3 - \mathsf{COLOR}$ сводится к языку $2 - \mathsf{COLOR}$, но для доказательства этого преобразовывает граф $G \in 2 - \mathsf{COLOR}$ так, чтобы $f(G) = \tilde{G}$ лежало в $3 - \mathsf{COLOR}$, то есть на самом деле доказывает не $3 - \mathsf{COLOR} \leqslant_p 2 - \mathsf{COLOR}$, а $2 - \mathsf{COLOR} \leqslant_p 3 - \mathsf{COLOR}$

Доказательство же того, что $2-\mathsf{COLOR} \leqslant_p 3-\mathsf{COLOR}$ не даёт нам ничего интересного, потому что мы сводим гораздо более простую задачу ($2-\mathsf{COLOR} \in \mathcal{P}$) к более сложной ($3-\mathsf{COLOR} \in \mathcal{NP}$).
Заметим, что из этой сводимости и свойства, доказанного в третьем пункте первой задачи следует, что $2-\mathsf{COLOR} \in \mathcal{NP}$, однако мы это и так знали в силу того, что $\mathcal{P} \subseteq \mathcal{NP}$

Задача 4. Докажите, что классы $\mathcal{P}$ и $\mathcal{NP}$ замкнуты относительно операции $*$ — звезды Клини (была в ТРЯПе). Для языка NP приведите также и сертификат принадлежности слова из Σ^* языку L^*, где $L \in \mathcal{NP}$.

Решение.
- $L \in \mathcal{P} \Leftrightarrow L$ распознаётся детерминированной МТ, работающей за полиномиальное время от длины входа. Тогда, так как полиномы замкнуты относительно элементарных операций, то языки вида L^n также распознаются детерминированной МТ, работающей за полиномиальное время от длины входа (т.е $L^n \in \mathcal{P}$), равно как и объединение языков, распознающихся за полином, распознаётся за полиномиальное время. Тогда имеем: $L^* = \bigcup\limits_{n=0}^{\infty} L^n \in \mathcal{P}$
- Доказательство аналогично с точностью до замены детерминированной МТ на недетерминированную. Приведём сертификат принадлежности s для $x \in \Sigma^*$ к $L^* \in \mathcal{NP}$. Пусть $s = w_1\#w_2\#\ldots\#w_n$, где $w_1w_2\ldots w_n = x, w_i \in L$ — разложение слова x по словам из языка L. Тогда мы можем за полином от $|x|$ проверить, что действительно $x = w_1w_2\ldots w_n$, следовательно $x \in L^n \subseteq L^* \in \mathcal{NP}$

Задача 5. Покажите, что язык разложения на множители

$$L_{factor} = \{(N,M) \in \mathbb{Z}^2 \mid 1 < M < N \text{ и } N \text{ имеет делитель } d, 1 < d \leqslant M\}$$

лежит в пересечении $\mathcal{NP} \cap co-\mathcal{NP}$.

Решение.
- Для начала покажем, что язык L_{factor} лежит в $\mathcal{NP}$. Для этого предъявим сертификат. Пусть $x = (n,m) \in \mathbb{Z}^2$. Сертификатом будет $d \in \mathbb{Z} : 1 \leqslant d \leqslant m$ — делитель числа n. Тогда мы с помощью деления в столбик за полиномиальное время от длины входа можем действительно убедиться в том, что $d \mid n$ и полиномиально проверить, что $1 \leqslant d \leqslant m$

- Теперь, чтобы показать, что язык L_{factor} лежит в $co-\mathcal{NP}$ необходимо для любого входа $x = (n,m) \in \mathbb{Z}^2$ такого что $x \notin L_{factor}$ подобрать такой сертификат s, чтобы мы могли полиномиально доказать $x \notin L_{factor}$. В качестве такого сертификата можно взять разложение n на простые множители: $s = p_1\#p_2\#\ldots\#p_k$, где все p_i различны (их кратности нас не интересуют), также заметим, что $k < n$. Теперь для доказательства того, что $x \notin L_{factor}$ необходимо и достаточно показать, что $\forall i \in \overline{1\ldots k} \hookrightarrow p_i > n$ — что делается за полиномиальное от длины входа время

Задача 6. Язык ГП состоит из описаний графов, имеющих гамильтонов путь. Язык ГЦ состоит из описаний графов, имеющих гамильтонов цикл (проходящий через все вершины, при-

чем все вершины в этом цикле, кроме первой и последней, попарно различны). Постройте явные полиномиальные сводимости ГЦ к ГП и ГП к ГЦ.

Решение. • ГЦ $\leqslant_p$ ГП: пусть на вход подаётся некоторый граф $G = (V, E)$, будем строить $f(G) = \tilde{G} = (\tilde{V}, \tilde{E})$. Пусть $V = (v_1, \ldots, v_n)$.

$\tilde{V} = V \cup \{v_{n+1}, v_{n+2}, v_{n+3}\}$; $\tilde{E} = E \cup E_{n+1} \cup E_{n+2} \cup E_{n+3}$, где

$E_{n+1} = \{(v_{n+1}, v_i) | (v_n, v_i) \in E\}$ — по рёбрам это копия вершины v_n

$E_{n+2} = \{(v_{n+2}, v_n)\}$; $E_{n+3} = \{(v_{n+3}, v_{n+1})\}$, таким образом $\deg(v_{n+2}) = \deg(v_{n+3}) = 1$

Заметим, что $f(G)$ работает за полином, так как она лишь дополняет вход 3 вершинами и $2 + \deg(v_n) \leqslant n + 1$ рёбрами. Докажем теперь, что G содержит ГЦ $\Leftrightarrow$ $\tilde{G}$ содержит ГП

- $\Rightarrow$: G содержит ГЦ, который, не умаляя общности, проходит вершины в порядке $v_1 \to v_2 \to \ldots \to v_n \to v_1$. Тогда покажем, что $\tilde{G}$ содержит ГП: начинаем путь с вершины v_{n+2}, затем идём в вершину v_n (такое ребро есть по построению), затем проходим по части ГЦ исходного графа G: $v_n \to v_1 \to \ldots \to v_{n-1}$, после чего направляем v_{n-1} в вершину v_{n+1} (ребро (v_{n-1}, v_n) есть в исходном графе, а следовательно $(v_{n-1}, v_{n+1}) \in \tilde{E}$ по построению), после чего идём по ребру (v_{n+1}, v_{n+3}), которое присутствует в $\tilde{G}$ по построению и тем самым завершаем построенный ГП: $v_{n+2} \to v_n \to v_1 \to \ldots \to v_{n-1} \to v_{n+1} \to v_{n+3}$
- $\Leftarrow$: $\tilde{G}$ содержит ГП. Так как вершины $\deg(v_{n+2}) = \deg(v_{n+3}) = 1$, то эти вершины обязательно являются концами этого пути, и тогда при их удалении вместе с инцидентными рёбрами в графе останется ГП, который теперь путь будет иметь своими концами вершины v_n и v_{n+1} (единственные, с которыми были связаны удалённые v_{n+2} и v_{n+3}). Допустим, что вершина v_{n+1} в этом ГП соединена ребром с некоторой вершиной u. Тогда по построению этой вершины v_{n+1} мы можем утверждать, что в исходном графе G есть ребро (v_n, u), после чего делаем финальный шаг: удаляем вершину v_{n+1} со всеми инцидентными ей рёбрами (и получаем тем самым обратно наш исходный граф G), после чего получаем ГП, имеющий концами v_n и u, добавляя ребро $(v_n, u) \in E$, получаем ГЦ в исходном графе G

• ГП $\leqslant_p$ ГЦ: пусть на вход подаётся некоторый граф $G = (V, E)$, будем строить $f(G) = \tilde{G} = (\tilde{V}, \tilde{E})$ следующим образом:

$\tilde{V} = V \cup \{u\}$; $\tilde{E} = E \cup \{(u, v_i) | v_i \in V\}$ — добавляем вершину, соединённую со всеми остальными

Заметим, что $f(G)$ работает за полином, так как она лишь дополняет вход 1 вершиной и $|V| = n$ рёбрами. Докажем теперь, что G содержит ГП $\Leftrightarrow$ $\tilde{G}$ содержит ГЦ

- $\Rightarrow$: G содержит ГП, возьмём его концы и соединим рёбрами с вершиной u — получим ГЦ в $\tilde{G}$

– $\Leftarrow$: $\tilde{G}$ содержит ГЦ, пусть в этом цикле вершина u соединена с v_p и v_q — разорвём цикл, удалив из него вершину u — получим ГП с концами v_p и v_q в исходном графе G

Задача 7. Регулярный язык L задан регулярным выражением. Постройте полиномиальный алгоритм проверки непринадлежности $w \notin L$. Вы должны определить, что вы понимаете под длиной входа, и выписать явную оценку трудоёмкости алгоритма.

Решение. Под длиной входа будем понимать $|RegExpr|$ — длину регулярного выражения.
Для начала по алгоритму из ТРЯПа (который, естественно, забыт, но должен быть упомянут для важности) строим полный ДКА по РВ, делается это за $O(n^3)$, где $n = |RegExpr|$.
Теперь будем считывать само слово w и моделировать на МТ работу ДКА, таким образом, считав всё слово, мы либо оказываемся в принимающем состоянии и, соответственно, $w \in L$, либо же этого не происходит и тогда мы определённо знаем, что $w \notin L$.
Оценим трудоёмкость алгоритма: переход из одного состояния ДКА в другое мы выполняем за 1 такт, так что на работу над входом тратится $O(|w|)$ тактов, построение ДКА по РВ, как было сказано ранее, выполняется за $O(n^3)$.
Таким образом, в ходе алгоритма совершается $O(n^3 + |w|)$ тактов работы МТ

Задача 9. Для языка $L \subset \Sigma^*$ определим язык $\mathsf{AND}(L) = (L\#)^* = \{w\# \mid w \in L\}^* \subset (\Sigma \cup \{\#\})^*$, где символ $\# \notin \Sigma$ — разделитель.
Верно ли, что если языки $L_1 \subset \Sigma_1^*$ и $L_2 \subset \Sigma_2^*$ таковы, что $L_1 \leqslant_P L_2$, то $\mathsf{AND}(L_1) \leqslant_P \mathsf{AND}(L_2)$?

Решение. 1) $L_1 \leqslant_p L_2 \Rightarrow \exists f : \Sigma^* \longrightarrow \Sigma^*$ такая, что $x \in L_1 \Leftrightarrow f(x) \in L_2$, причём f — полиномиально вычислима от длины входа
2) $x \in \mathsf{AND}(L_j) \Leftrightarrow \exists n \in \mathbb{N}_0 : x \in (L_j\#)^n \Leftrightarrow \exists n \in \mathbb{N}_0 : x = w_1\#w_2\#\ldots\#w_n\#, \forall i \in \overline{1\ldots n} \hookrightarrow w_i \in L_j$ (тут при $n = 0 \hookrightarrow x = \varepsilon$), то есть все слова, лежащие в языке $\mathsf{AND}(L_j)$, имеют вид $w_1\#w_2\#\ldots\#w_n\#, \forall i \in \overline{1\ldots n} \hookrightarrow w_i \in L_j$
Таким образом имеем:
$x = w_1\#w_2\#\ldots\#w_n\# \in \mathsf{AND}(L_1) \overset{2)}{\Longleftrightarrow} w_i \in L_1 \overset{1)}{\Longleftrightarrow} f(w_i) \in L_2 \overset{2)}{\Longleftrightarrow} f(w_1)\#f(w_2)\#\ldots\#f(w_n)\# \in \mathsf{AND}(L_2) \overset{3)}{\Longleftrightarrow} g(w_1\#w_2\#\ldots\#w_n\#) = f(w_1)\#f(w_2)\#\ldots\#f(w_n)\# \in \mathsf{AND}(L_2) \Leftrightarrow g(x) \in \mathsf{AND}(L_2)$
3) определим алгоритм, вычисляющий $g(x)$ следующим образом: сначала за линейное от длины входа время проверяем вход: если $x \neq \varepsilon \wedge x \neq w_1\#w_2\#\ldots\#w_n\#$, то сразу останавливаем вычисление функции и выдаём, что $g(x) \notin \mathsf{AND}(L_2)$. Если же вход «правильный», то для каждого w_i вызываем функцию $f(w_i)$ и записываем в выход $f(w_i)\#$, получая в конечном счёте на выходе $g(x) = f(w_1)\#f(w_2)\#\ldots\#f(w_n)\# \in \mathsf{AND}(L_2) \Leftrightarrow f(w_i) \in L_2 \Leftrightarrow w_i \in L_1$.
Заметим, что g — полиномиально вычислимая от длины входа функция, так как она сначала проверяет вход (что делается за линейное время), а затем $\leqslant |x|$ раз вызывает полиномиально вычислимую от длины w_i функцию f, что тоже делается за полиномиальное от длины входа время