Рябых Владислав, Б05-905

Алгоритмы и модели вычислений.
Домашнее задание № 5

Задача 1. *(i)* Докажите, что в Σ_2 лежит язык булевых формул от двух наборов переменных $\varphi(x_1, \dots, x_n, y_1 \dots y_n) = \varphi(\vec{x}, \vec{y})$ таких, что при некоторых значениях $\vec{x}$ они справедливы вне зависимости от значений $y_1, \dots, y_n$.
(ii) Придумайте какую-нибудь свою задачу из класса Σ_3 (или Π_3, на ваш вкус).
(iii) Докажите, что $\Sigma_k \subset \Sigma_{k+1} \cap \Pi_{k+1}$.
(iv) Докажите, что $\mathcal{NP} \subset \mathcal{PSPACE} \subset \mathcal{EXPTIME}$.

Решение. 1. Пусть L — язык из условия задачи. Заметим, что принадлежность булевой формулы языку можно переписать в виде: $\varphi(\vec{x}, \vec{y}) \in L \Leftrightarrow \exists \vec{x}\ \forall \vec{y} \hookrightarrow \varphi(\vec{x}, \vec{y}) = 1$. А это в свою очередь сильно напоминает условие принадлежности языка классу Σ_2: $\Sigma_2 = \{L | input \in L \Leftrightarrow \exists x\ \forall y \hookrightarrow R(input, x, y) = 1\}$, где R вычислима за $poly(|input|)\}$.

   В нашем случае $input = (\vec{x}, \vec{y})$, вычисление значения булевой функции $\varphi(\vec{x}, \vec{y})$ при заданном значении производится за $poly(|\vec{x}| + |\vec{y}|) = poly(|input|)$. Таким образом, наш язык L лежит в Σ_2.

2. Как мы доказываем в следующем пункте, имеют место следующие вложения: $\mathcal{P} = \Sigma_0 \subseteq \Sigma_1 \subseteq \Sigma_2 \subseteq \Sigma_3$, так что можем взять любую задачу из $\mathcal{P}$: мне очень нравится задача поиска k-й порядковой статистики в заданном массиве: $kth - order - statistics = \{(a, k, x) \mid a_k = x\}$, которая, как известно из курса основных алгоритмов, решается за $O(|a|)$.

3.
   - Докажем сначала, что $\Sigma_k \subseteq \Sigma_{k+1}$: возьмём произвольный язык $L \in \Sigma_k$. $x \in L \Leftrightarrow \exists y_1\ \forall y_2\ \dots \exists(\forall) y_k \hookrightarrow R(x, y_1, y_2, \dots, y_k) = 1$. Тогда мы можем добавить в R фиктивную переменную y_{k+1}, которая, соответственно, никак не будет влиять на работу R, тогда мы можем записать аналогичную формулу, из которой уже будет следовать, что $L \in \Sigma_{k+1}$: $x \in L \Leftrightarrow \exists y_1\ \forall y_2\ \dots \exists(\forall) y_k\ \forall(\exists) y_{k+1} \hookrightarrow R(x, y_1, y_2, \dots, y_k, y_{k+1}) = 1$.
   - Теперь $\Sigma_k \subseteq \Pi_{k+1}$: возьмём произвольный язык $L \in \Sigma_k$. $x \in L \Leftrightarrow \exists y_1\ \forall y_2\ \dots \exists(\forall) y_k \hookrightarrow R(x, y_1, y_2, \dots, y_k) = 1$. Если в прошлом пункте мы добавляли фиктивную переменную y_{k+1} в «конец», то теперь добавим фиктивную переменную y_0 в «начало», она никак не будет влиять на работу R, таким образом мы можем записать аналогичную формулу, из которой уже будет следовать, что $L \in \Pi_{k+1}$: $x \in L \Leftrightarrow \forall y_0\ \exists y_1\ \forall y_2\ \dots \exists(\forall) y_k \hookrightarrow R(x, y_0, y_1, y_2, \dots, y_k) = 1$.

4.
   - сначала докажем, что $\mathcal{NP} \subseteq \mathcal{PSPACE}$: возьмём произвольный язык $L \in \mathcal{NP}$, по определению $x \in L \Leftrightarrow \exists y : R(x, y) = 1$, причём R вычислима за $poly(|x|)$. Если мы можем за полиномиальную от длины входа память перебрать все сертификаты, то таким образом мы докажем доказать, что $L \in \mathcal{PSPACE}$.

     Так как длина всех сертификатов не превышает длину некоторого полинома от x, то существует такой полином $p(|x|)$, что $\forall y \hookrightarrow y \leqslant p(|x|)$. Тогда определим работу МТ следующим образом: первые $p(|x|)$ ячеек выделяются под сертификат, а следующие n — под вычисление $R(x,y)$ (причём R вычислима за полином, так что n тоже не больше некоторого полинома от $|x|$ и таким образом мы выделяем $p(|x|) + n = poly(|x|)$ ячеек). Также заметим, что всего возможных сертификатов конечное количество: если m — количество символов в алфавите МТ, то всего сертификатов $1 + m + m^2 + \ldots + m^{p(|x|)}$. Таким образом наша МТ выдаёт 1, если сертификат подходит, то есть $R(x, y) = 1$. Иначе МТ стирает текущий сертификат, записывает в «начало» своей ленты следующий и продолжает свою работу по поиску подходящего сертификата. Если же МТ перебрала все сертификаты и не нашла подходящий, то она останавливается и выдаёт 0. Таким образом мы за полиномиальную память проверяем наличие сертификата, то есть принадлежность входа x к языку L, причём в таком случае по построенной нами распознающей МТ также верно, что $L \in \mathcal{PSPACE} \Rightarrow \mathcal{NP} \subseteq \mathcal{PSPACE}$.

- $\mathcal{PSPACE} \subseteq \mathcal{EXPTIME}$: рассмотрим количество возможных конфигураций МТ, распознающей некоторый язык $L \in \mathcal{PSPACE}$ за полиномиальную от длины входа память. Пусть $p(|x|) = poly(|x|)$ — верхняя оценка используемой памяти. Тогда верхней оценкой количества всех конфигураций МТ будет $p(|x|) \cdot |x| \cdot q \cdot m^{p(|x|)}$, где q — количество различных состояний МТ, m — количество символов в алфавите МТ. МТ не может сделать больше шагов, потому что в этом случае она бы зациклилась и не остановилась. Таким образом, всего МТ делает $\leqslant p(|x|) \cdot |x| \cdot q \cdot m^{p(|x|)} = poly(|x|) \cdot m^{p(|x|)} \leqslant m^{poly(|x|)} \cdot m^{p(|x|)} = m^{poly(|x|)}$ — экспоненциальное время, так как m — константа. Таким образом произвольный язык $L \in \mathcal{PSPACE}$ распознаётся за экспоненциальное время, а следовательно $\mathcal{PSPACE} \subseteq \mathcal{EXPTIME}$.

Таким образом получаем: $\mathcal{NP} \subseteq \mathcal{PSPACE} \subseteq \mathcal{EXPTIME}$.

Задача *2.* Покажите, как свести следующую задачу к вычислению некоторого перманента: найти количество перестановок n элементов, в которых части элементов (с номерами $i_1, i_2, \ldots i_k$) запрещено занимать позиции $j_1, \ldots j_k$ соответственно.

Решение. Для того, чтобы свести задачу к вычислению некоторого перманента, нам для начала нужно определить матрицу, перманент которой мы собираемся считать: возьмём булеву матрицу $A = (a_{ij})$ размера $n \times n$, причём $a_{ij} = 1 \Leftrightarrow i$-му элементу разрешено стоять на j-й позиции. Тогда исходным числом перестановок и будет $perm\ A = \sum\limits_{\sigma \in S_n} a_{1\sigma(1)} a_{2\sigma(2)} \ldots a_{n\sigma(n)}$, так как если комбинация разрешена по условию, то все $a_{k\sigma(k)}$ обращаются в 1 и перманент учитывает эту перестановку в своей сумме, а если комбинация запрещена, то $\exists k : a_{k\sigma(k)} = 0$ и перманент данную перестановку не учитывает.

Задача *3.* Докажите, что если всякий $\mathcal{NP}$-трудный язык является $\mathcal{PSPACE}$-трудным, то $\mathcal{PSPACE} = \mathcal{NP}$.

Решение. То что $\mathcal{NP} \subseteq \mathcal{PSPACE}$ мы доказали в 4м пункте первой задачи. Докажем теперь, что если всякий $\mathcal{NP}$-трудный язык является $\mathcal{PSPACE}$-трудным, то выполняется и обратное включение.

Рассмотрим произвольный язык $A \in \mathcal{NP}_c$. $A \in \mathcal{NP} \cap \mathcal{NP}_h \Rightarrow A \in \mathcal{PSPACE} \cap \mathcal{PSPACE}_h \Rightarrow A \in \mathcal{PSPACE}_c$. Тогда любую задачу из $\mathcal{PSPACE}$ можно за полиномиальное время свести к задаче из $\mathcal{NP}$ и, собственно, за полиномиальное время на недетерминированной МТ решить её. То есть таким образом любая задача из $\mathcal{PSPACE}$ решается за полиномиальное время на недетерминированной МТ, а следовательно, лежит в $\mathcal{NP}$. Поэтому $\mathcal{PSPACE} \subseteq \mathcal{NP}$ и обратное включение доказано.

Задача 4. Докажите, что следующие языки лежат в $\mathcal{L}$:
(*i*) $\{a\#b\#c | c = a + b\}$ (a, b, c — числа в двоичной записи).
(*ii*) $\{a\#b\#c | c = a \cdot b\}$ (a, b, c — числа в двоичной записи).
(*iii*) $UCYCLE = \{G \mid \text{В неориентированном графе } G \text{ есть цикл}\}$

Решение.

1. Вроде как было на ТФС, причём на самом деле для этой задачи достаточно константной памяти. Можно хранить i — указатель разряда, который мы складываем в данный момент времени (идём при этом с конца), бит $shift$, отвечающий за перенос и, собственно, результат последнего сложения двух разрядов $res = a_i + b_i + shift$, который мы на каждом шаге будем сравнивать с c_i. Если существует разряд, для которого равенство не выполняется — выводим 0 и завершаем работу. Если были обработаны все разряды и МТ при этом не остановилась — выводим 1.

2. Вроде бы тоже было, но тут я уже не уверен. Однако алгоритм примерно тот же. Будем хранить i — указатель разряда, который мы складываем в данный момент времени (идём при этом с конца), число (а не бит, как в предыдущем пункте) $shift$, отвечающий за перенос и, собственно, результат последнего сложения двух разрядов $res = a_0b_i + a_1b_{i-1} + \ldots + a_ib_0 + shift$, последний разряд которого мы на каждом шаге будем сравнивать с c_i, а всё, кроме последнего разряда — отправлять в перенос. Если существует разряд, для которого равенство не выполняется — выводим 0 и завершаем работу. Если были обработаны все разряды и МТ при этом не остановилась — выводим 1.

3. По теореме Рейнгольда $UPATH \in L$. Таким образом мы можем перебрать всевозможные упорядоченные пары вершин (u, v) такие, что $(u, v) \in E$. Делать это будем следующим образом: пусть $G = (V, E)$ — наш исходный граф. Возьмём граф $\tilde{G} = (V, \tilde{E})$, где $\tilde{E} = E \setminus (u, v)$. Проверим наличие пути между из вершины v в вершину u (можем это сделать за логарифмическую память по упомянутой выше теореме Рейнгольда). И заметим, что если в $\tilde{G}$ найден путь из v в u, то в исходном графе G есть цикл, так как добавляется ребро (u, v), дополняющее найденный путь до цикла.

Задача 5. Сертификатное определение $\mathcal{NL}$: $A \in \mathcal{NL}$ тогда и только тогда, когда для некоторой детерминированной машины M выполнена эквивалентность: $x \in A \Leftrightarrow \exists s : \ M(x, s) = 1$. При этом длина s должна быть полиномиальна от длины x, машина получает s на отдельной ленте, по которой может двигаться только слева направо, а количество ячеек, занятых на рабочей ленте, должно быть логарифмическим.
Вопрос задачи: а какой класс получится, если в предыдущем определении разрешить машине двигаться по сертификатной ленте в обе стороны?

Решение. Пусть получается некоторый класс $\mathcal{A}$. Из определения очевидно, что $\mathcal{A} \subseteq \mathcal{NP}$, так как мы по сути берём определение класса $\mathcal{NP}$, но накладываем дополнительное ограничение в виде того, что *количество ячеек, занятых на рабочей ленте, должно быть логарифмическим.*
Докажем, что на самом деле верно и обратное включение. Для этого построим на нашей МТ верификатор для $\mathcal{NP}_c$ задачи 3КНФ, который будет при этом работать за логарифмическую память. Будем хранить всего 2 значения: значение текущего дизъюнкта, значение рассматриваемого в данный момент литерала. Таким образом, взяв сертификатом выполняющий набор, мы без проблем проверяем истинность формулы с помощью того, что дойдя до некоторого литерала, можем без проблем «пробежаться по ленте назад» и посмотреть его значение. Таким образом $\mathcal{NP} \subseteq \mathcal{A}$.
Итого имеем: $A = \mathcal{NP}$.

Задача 6. Докажите, что логарифмическая сводимость транзитивна, причем если $B \in \mathcal{L}$ и $A \leqslant_L B$, то $A \in \mathcal{L}$, а если $B \in \mathcal{NL}$ и $A \leqslant_L B$, то $A \in \mathcal{NL}$.

Решение.

1. Транзитивность: если $A \leqslant_L B$ и $B \leqslant_L C$, то $\exists f, g$, вычислимые за логарифмическую память, такие что $x \in A \Leftrightarrow f(x) \in B \Leftrightarrow g(f(x)) \in C$. Заметим, что функция $g(f(x))$ вычисляется за $\log(|f(x)|) = \log(\log(|x|)) = \log(|x|)$, так как логарифм от логарифма также является логарифмом. Таким образом $x \in A \Leftrightarrow g(f(x)) \in C \Rightarrow A \leqslant_L C$

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

   Построим теперь детерминированную МТ M, распознающую язык A за полином: на входе x сначала моделируется работа M_1, то есть за логарифмическую память 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 $\log(|x|) + \log(|f(x)|) = \log(|x|) + \log(\log(|x|)) = \log(|x|)$, так как логарифм от логарифма также является логарифмом и сумма логарифмов есть логарифм. Таким образом язык A распознаётся за логарифмическую память от длины входа на детерминированной МТ M, следовательно $A \in \mathcal{L}$

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

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

Построим теперь недетерминированную МТ M, распознающую язык A за полином: на входе x сначала моделируется работа M_1, то есть за логарифмическую память 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 $\log(|x|)+\log(|f(x)|) = \log(|x|) + \log(\log(|x|)) = \log(|x|)$, так как логарифм от логарифма также является логарифмом и сумма логарифмов есть логарифм. Таким образом язык A распознаётся за логарифмическую память от длины входа на недетерминированной МТ M, следовательно $A \in \mathcal{NL}$