P vs. NP

Класс P и сведение

Сейчас мы переходим в теоретический материал, который является основой классификации алгоритмов. Нас совершенно не интересует эффективность, кроме как разделение на «алгоритмы, которые работают за полиномиальное время» и все остальные.

Договоримся, что мы определяем «алгоритм», как детерминированную машину Тьюринга. И будем использовать тезис Чёрча-Тьюринга о том, что любая вычислимая функция является вычислимой на машине Тьюринга. Также условимся, что мы рассматриваем только задачи принятия решения (принадлежит ли слово)

Определение 1. *Задача $A \in \mathsf{P}$ (формально говоря язык), если существует машина Тьюринга, проверяющая принадлежность слова языку, время работы $t_A(x)$ которой ограничено полиномом от $P(|x|)$, который является фиксированным для всех входов.*

Также, до определения класса NP, определим, что значит язык A сводится к языку B за полиномиальное время.

Определение 2. *Язык A **сводится** к языку B за полиномиальное время, если существует функция f, вычислимая за полиномиальное время, такая, что $w \in A \iff f(w) \in B$. Обозначение: $A \leqslant_p B$.*

Докажем основное утверждение о сводимости:

Утверждение 1. *Пусть $A \leqslant_p B$ и $B \in \mathsf{P}$ (существует полиномиальный алгоритм решения задачи). Тогда $A \in \mathsf{P}$.*

Доказательство. Достаточно построить алгоритм, показывающий, что для языка A тоже существует полиномиальное решение. Пусть у нас имеются МТ M_A, допускающая язык A, и M_B, допускающая язык B. Поскольку $A \leqslant_p B$, то $\exists$ полиномиальная f, удовлетворяющая определению выше. Построим алгоритм вычисления M_A:

Require: слово w
  вычислить $f(w)$
  return $M_B\big(f(w)\big)$

Поскольку сама функция f является вычислимой за полиномиальное время, то вычисление $f(w)$ потратит полиномиальное время. Кроме того, поскольку $B \in \mathsf{P}$, то вычисление $M_B\big(f(w)\big)$ также займёт полиномиальное время (от размера $f(w)$, но $f(w)$ также вычисляется за полином)! Отсюда $M_A(w)$ также допускает слово w за полиномальное время, что по определению означает, что $A \in \mathsf{P}$. □

NP класс

Определение 3. *Язык L лежит в классе NP, если существует функция от двух аргументов $A(x, y)$ — **алгоритм верификации** — с полиномиальной сложностью от x такая, что x лежит в L тогда и только тогда, когда для него существует y (его принято называть **сертификатом**) такой, что $A(x, y) = 1$. При этом сертификат должен быть полиномиально зависим от размера x.*

Ясно, что $\mathsf{P} \subseteq \mathsf{NP}$, так как за алгоритм верификации можно взять просто полиномиальный алгоритм принадлежности слова языку.

Также введём понятие coNP класса:

Определение 4.

$$\mathsf{coNP} = \{\overline{L} = \Sigma^* \setminus L \mid L \in \mathsf{NP}\}$$

Другими словами, coNP — множество всех языков, которые могут верифицировать **не**принадлежность слово языку. Например, такая задача — «Нет ли гамильтонового цикла в графе?». Неизвестно, лежит ли она в NP, но она точно лежит в coNP (оставляем упражнение читателю, как предъявить верификатор). Также ясно, что $\mathsf{P} \subseteq \mathsf{coNP}$.

Оставим тоже следующую лемму, как упражнение читателю (думаем, что 2 определения должны быть интуитивно понятны):

Лемма 1. $\mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME}$

NPC классы и теорема Кука-Левина

Определение 5. *Язык A— NP-трудный, если* $\forall\ B \in \mathsf{NP} : B \leqslant_p A$.

Определение 6. *Язык $A \in \mathsf{NPC}$ (NP-complete или Nondeterministic Polynomial Complete), если $A \in \mathsf{NP}$ и* $\forall\ B \in \mathsf{NP} \implies B \leqslant_p A$.

Другими словами, NPC — это в точности те языки, которые являются сами NP и NP-трудными одновременно.

Докажем следующие простые утверждения:

Лемма 2. *Пусть $L \in \mathsf{NPC}$. Тогда, если $L \in \mathsf{P}$, то $\mathsf{P} = \mathsf{NP}$.*

Доказательство. Пусть $L' \in \mathsf{NP}$. Так как $L \in \mathsf{NPC}$, то $L' \leqslant_p L$. Но, поскольку $L \in \mathsf{P}$, то $L' \in \mathsf{P}$, откуда $\mathsf{P} = \mathsf{NP}$. □

Лемма 3. *Если $B \in \mathsf{NPC}$ и $B \leqslant_p C \in \mathsf{NP}$, то $C \in \mathsf{NPC}$.*

Доказательство. Проверим оба условия, входящих в определение NP-полного класса.

1. $C \in \mathsf{NP}$ – по условию.
2. Пусть $L' \in \mathsf{NP}$. Так как $B \in \mathsf{NPC}$, то $L' \leqslant_p B$, что по определению означает, что существует некая функция $f_{L'B}$ (которая сводит одну задачу к другой), вычислимая за полиномиальное время. Кроме того, $B \leqslant_p C$, что по определению означает, что существует некая функция f_{BC}, вычислимая за полиномиальное время.

   Но это означает, что $L' \leqslant_p C$, так как существует функция $f_{L'C}$, такая, что $f_{L'C}(w) = f_{BC}\big(f_{L'B}(w)\big)$ – она также является вычислимой за полиномиальное время.

   Следовательно, $\forall\ L' \in \mathsf{NP} : L' \leqslant_p C$, откуда C – NP-трудный по определению.

Оба условия выполняются, значит, $C \in \mathsf{NPC}$. □

Но всё равно должны же остаться вопросы по тому, зачем всё это надо и как вообще доказывать, что задача трудна, то есть лежит в классе NP. Ведь должна же быть какая-то «первая» задача из NPC. Если вы решаете задачу и вдруг понимаете, что она эквивалентна какой-то задаче из NPC, то возможно стоит либо перечитать задачу (и понять, каким условием надо точно пользоваться), либо решить и получить \$1 000 000.

Одна из первых NPC задач была задача SAT (от англ. Satisfiability). Можем считать, что нам дана булева формула в конъюнктивно нормальной форме. Сейчас мы покажем, что любая NP задача сводится к этой. Вообще, так как если мы хотим сводить любую NP задачу к SAT, то какие-то обычные рассуждения вряд ли пройдут. Действительно, мы будем стараться моделировать работу МТ через КНФ.

Теорема 1 (Теорема Кука-Левина)**.** $SAT \in \mathsf{NPC}$.

Доказательство. Сначала покажем, что $SAT \in \mathsf{NP}$. Действительно, по входу булевой формулы и верификатору (значение переменных $x_1, \ldots, x_n$) мы с лёгкостью проверим выполнимость формулы. Поэтому есть линейный алгоритм верификации.

Пусть $\mathcal{P} \in \mathsf{NP}$. Надо понять, как полиномиально свести эту задачу к SAT. По определению класса NP существует полином p и алгоритм $\mathcal{P}'$, который верифицирует задачу $\mathcal{P}$, причём верификатор размера не больше, чем $|p(|x|)|$.

Пусть

$$\Phi : \{0, \ldots, N\} \times \overline{A} \to \{-1, \ldots, N\} \times \overline{A} \times \{-1, 0, 1\}$$

МТ, работающая за полиномиальное время, отвечающая за задачу $\mathcal{P}'$ с алфавитом A. Добавим пустой символ к A, чтобы легче было работать. Пусть работа этой МТ ограничена полиномом q, то есть $\mathrm{time}(\Phi, x\#c) \leqslant q(|x\#c|)$. Сейчас мы соорудим набор дизъюнктов (или клауз) $Z(x)$, который ограничен полиномом $Q \leftarrow q(|x\#c|)$, такой, что $Z(x)$ выполняется тогда и только тогда, когда слово принадлежит языку $\mathcal{P}$.

Ясно, что мы не уйдём за пределы ленты от $-Q$ до Q.

Теперь создадим несколько переменных:

- переменные $v_{ij\sigma}$ для всех $0 \leqslant i \leqslant Q$, $-Q \leqslant j \leqslant Q$ и $\sigma \in \overline{A}$. Верно ли, что в момент i (после i шагов выполнения МТ) позиция с номером j содержала символ σ.
- переменные w_{ijn} для всех $0 \leqslant i \leqslant Q$, $-Q \leqslant j \leqslant Q$ и $-1 \leqslant n \leqslant N$. Верно ли, что в момент i на позиции j МТ была в состоянии n. За минус 1 отвечает финальное состояние.

Поэтому, если у нас есть конфигурация МТ на i-ом шаге с позицией $\pi^{(i)}$, символами $s^{(i)}$ и состоянием $n^{(i)}$, тогда мы должны выставить $v_{ij\sigma} := true$ тогда и только тогда, когда $s_j^{(i)} = \sigma$ (s_j — позиция на j-ом месте ленты МТ). И мы должны выставить $w_{ijn} := true$ тоже тогда и только тогда, когда $\pi^{(i)} = j$ и $n^{(i)} = n$.

Сейчас мы предъявим набор клауз $Z(x)$, который будет выполняться тогда и только тогда, когда существует строка c полиномиального размера, что МТ Φ на входе $x\#c$ выдаёт единицу.

В каждый момент времени на каждой позиции стоит строго один символ:

- дизъюнкт $(v_{ij\sigma} \mid \sigma \in \overline{A})$ по всем $0 \leqslant i \leqslant Q$ и $-Q \leqslant j \leqslant Q$.
- $(\overline{v_{ij\sigma}}, \overline{v_{ij\tau}})$ по всем $0 \leqslant i \leqslant Q$, и $-Q \leqslant j \leqslant Q$, и $\sigma \neq \tau \in \overline{A}$.

В каждый момент времени единственная позиция в строке сканируется и единственная инструкция выполняется:

- $(w_{ijn} \mid -Q \leqslant j \leqslant Q, -1 \leqslant n \leqslant N)$ для всех $0 \leqslant i \leqslant Q$.
- $(\overline{w_{ijn}}, \overline{w_{ij'n'}})$ по всем $0 \leqslant i \leqslant Q$ и $-Q \leqslant j, j' \leqslant Q$ и $-1 \leqslant n, n' \leqslant N$ с условием, что $(j, n) \neq (j', n')$.

Алгоритм корректно начинает свою работу:

- (v_{0,j,x_j}) по всем $1 \leqslant j \leqslant |x|$.
- $(v_{0,|x|+1,\#})$ — разделяющий символ.
- $(v_{0,|x|+1+j,0}, v_{0,|x|+1+j,1})$ по всем $1 \leqslant j \leqslant p(|x|)$.
- $(v_{0,j,\sqcup})$ по всем $-Q \leqslant j \leqslant 0$ и $|x| + 2 + p(|x|) \leqslant j \leqslant Q$ — пустые символы не с входа.
- $(w_{0,1,0})$ — начальное положение головки.

Алгоритм работает корректно:

- $(\overline{v_{ij\sigma}}, \overline{w_{ijn}}, v_{i+1,j,\tau})$ и $(\overline{v_{ij\sigma}}, \overline{w_{ijn}}, w_{i+1,j+\delta,m})$ по всем $0 \leqslant i < Q, -Q \leqslant j \leqslant Q; \sigma, \tau \in \overline{A}, \delta \in \{-1, 0, 1\}$, где $\Phi(n, \sigma) = (m, \tau, \delta)$ (переход по состоянию n, символу σ в какое-то состояние m, символ τ пишем на j-ом месте и сдвиг на $-1, 0$ или 1).

Когда алгоритм достигает финального состояния (минус 1 в нашем случае), алгоритм останавливается:

- $(\overline{w_{i,j,-1}}, w_{i+1,j,-1})$ и $(\overline{w_{i,j,-1}}, \overline{v_{i,j,\sigma}}, v_{i+1,j,\sigma})$ по всем $0 \leqslant i < Q; -Q \leqslant j \leqslant Q$ и $\sigma \in \overline{A}$ — как раз, если достигли состояния -1, тогда все состояния в каждое время должны быть минус один и символ меняться не должен.

Позиции, которые не были просмотрены, должны остаться неизменными:

- $(\overline{v_{ij\sigma}}, \overline{w_{ij'n}}, v_{i+1,k,\sigma})$ по всем $0 \leqslant i \leqslant Q; \sigma \in \overline{A}; -1 \leqslant n \leqslant N$ и $-Q \leqslant j, j' \leqslant Q$ при условии, что $j \neq j'$.

Алгоритм выводит единицу:

- $(v_{Q,1,1})$ и $(v_{Q,2,\sqcup})$ — на первой позиции стоит единица, а потом пробел (можно считать, что только один, можно, что все последующие, это не играет на роль полиномиальности МТ и полиномиальности размера КНФ).

Заметим, что размер $Z(x)$ будет не больше, чем $\mathcal{O}(Q^3 \log Q)$, существует $\mathcal{O}(Q^3)$ литералов и нужно $\mathcal{O}(\log Q)$ памяти, чтобы закодировать индексы.

Осталось показать, что $Z(x)$ выполняется тогда только тогда, когда x принадлежит языку.

Если $Z(x)$ выполняется, то пусть T — это набор переменных, удовлетворяющим всем клаузам. Поставим $c_j = 1$ по всем j для которых $T(v_{o,|x|+1+j,1}) = true$ и $c_j = 0$ в ином случае. Сверху мы описали работу МТ Φ на входе $x\#c$. Поэтому мы можем заключить, что $\Phi(x\#c) = 1$. Так как Φ — алгоритм верификации, то это значит, что x принадлежит языку.

Пусть x принадлежит языку. Тогда пусть c — сертификат проверки для x. Тогда пусть конфигурация МТ на входе $x\#c$ на i-ом шаге пусть будет равна $(n^{(i)}, s^{(i)}, \pi^{(i)})$. Тогда давайте поставим $T(v_{i,j,\sigma}) = true$ тогда и только тогда, когда $s_j^{(i)} = \sigma$ и $T(w_{i,j,n}) = true$ тогда и только тогда, когда $\pi^{(i)} = j, n^{(i)} = n$ по всем $i \leqslant m$. Для бóльших i выставим $T(v_{i,j,\sigma}) := T(v_{i-1,j,\sigma})$ и $T(w_{i,j,n}) := T(w_{i-1,j,n})$ по всевозможным j, n, σ. Тогда можно убедиться, что формула выполняется, что завершает наше доказательство. □