1 Теория сложности

До сих пор мы обсуждали только то, что можно вычислить на МТ. Сейчас мы поговорим о том, насколько быстро ту или иную задачу можно решить на МТ. При этом не имеет значения, сколько та или иная задача будет вычисляться на МТ, так как мы не собираемся запускать её на МТ. Мы собираемся запускать её на реальной машине. Поэтому первое, что нам важно понять — как соотносится время работы МТ с временем работы реальной машины.

Расширенный принцип Чёрча-Тьюринга: Любой адекватный вычислитель работает с той же скоростью, что и МТ с разве лишь полиномиальным ускорением или замедлением.

Следствие 1. *ММ не является адекватным вычислителем.*

1.1 Задачи разрешения

Определение 1. *Сложность алгоритма — это функция* $f : \mathbb{N} \to \mathbb{N}$*, которая показывает точную верхнюю границу количества операций, необходимых МТ для анализа входа в зависимости от его длины.*

Вспомним терминологию:

$f = o(g)$ т.е. f — пренебрежительно мало по сравнению с g, т.е. $\frac{f}{g} \longrightarrow 0$

$f = O(g)$ т.е. $\exists M\ \forall n > N\ f(n) \leq M \cdot g(n)$

$f = \Theta(g)$ т.е. $\exists M_1, M_2\ \forall n > M_1, M_2\ M_2 \cdot g(n) \leq f(n) \leq M_1 \cdot g(n)$

Определение 2. $DTIME(f)$ *— множество задач разрешения, таких, что для них существует алгоритм, сложность которого имеет оценку* $O(f)$*.*

Определение 3. *Тогда можно определить* $P = \cup_{k=1}^{\infty} DTIME(n^k)$ *— класс сложности. Берём все задачи, которые решаются за линейное время, за квадратичное время, за кубическое и т.д., и объединяем их. Получаем класс задач, которые разрешимы за полиномиальное время.*

Определение 4. $EXPTIME = \cup_{k=1}^{\infty} DTIME(2^k)$

Определение 5. $EXPTIME2 = \cup_{k=1}^{\infty} DTIME(2^{2^k})$

Очевидно, что $DTIME(n^k) \leq DTIME(n^{k+1})$. Ответим на вопрос: действительно ли $\forall k\ \exists p$ — задача, такая, что: $p \in DTIME(n^k), p \notin DTIME(n^{k-1})$. Иначе говоря, действительно ли эта система расширяется?

1.2 Теорема о временной иерархии

Теорема 1. ***(Теорема о временной иерархии)***
$DTIME(\frac{o(f(n)}{\log(f(n))}) \subsetneqq DTIME(f(n))$
Более слабая формулировка:
$DTIME(f(n)) \subsetneqq DTIME(f^3(2n+1))$

Доказательство. Прежде всего определим язык $H_f = \{M,\ x \mid M$ останавливается на входе длины $|x| < f(x)$ и печатает «да»$\}$

$\Rightarrow$ Покажем, что $H_f \subseteq DTIME(f^3)$. У нас есть множество языков, которые представляют собой пары (машина, вход). Пары определены так: получив указанный вход, машина останавливается, если длина $|x| < f(x)$, и «да». То есть мы просто запускаем машину на этом входе и смотрим, остановилась ли она за правильное количество шагов. Она должна остановиться через $f(x)$. Для этого нам нужно смоделировать МТ, то есть запустить УМТ, выполнить на ней программу и ждать. УМТ накладывает дополнительные расходы — в данном случае моделирует за куб. Если машина успела остановиться, то пара (машина, вход) попадает в этот язык, если нет, то не попадает.

Покажем, что $H_f \notin DTIME(f(\frac{x}{2}))$, где x - длина входа.

Предположим обратное: $H_f \in DTIME(f(\frac{x}{2}))$. Это означает, что существует машина P, которая распознает этот язык, и распознает его за это время (по определению). То есть P разрешает язык H_f за $o(f(\frac{x}{2}))$.

Введем $P'(M) = P(M, M)$, которая работает за $f(x)$, так как удваивает вход. Дальше возьмем $P''(M) = \bar{P}'(M)$ — диагонализирующая машина, сложность работы та же.

Выясним, как работает $P''(P'')$. $P''(P'')$ работает за $f(x) \to (P'', P'') \in H_f$ (по определению). С другой стороны, $(P'', P'') \notin H_f$, так как $P''(P'') = 1 \leftrightarrow P(P'', P'') = 0$. Это означает, что данная машина не может остановиться через указанное время (не входит в этот язык). Получили противоречие, следовательно, наше предположение о неравенстве классов верно. H_f называется разделяющим языком для этих классов. □

Следствие 2. *Иерархия не схлопывается.*

Следствие 3. $P \subsetneqq EXPTIME$

Фактически мы доказали, что существуют задачи, которые можно решить алгоритмом сложности $O(n^{1000000})$ и нельзя решить за $O(n^{999999})$. Очевидно, что $O(n^{1000000})$ — это не очень хорошая сложность вычислений. В соответствии с законом Мура, вычислительная мощность современных машин вырастает вдвое

каждые два года, и поэтому мы можем решать всё больше полиномиальных задач, а для экспоненциальных можем добавлять только по единичке раз в два года. Однако существуют физические ограничения, при которых процессор должен превратиться в «шар плазмы», потому что та энергия, которая должна в нём умещаться, и та частота, на которой процессор должен работать, будут соответствовать «шару плазмы». Очевидно, этого не произойдёт, поэтому рост когда-нибудь закончится. Но надо понимать, что теория алгоритмов — это теория, а не практика, поэтому нельзя относиться к P как к классу «хороших» алгоритмов, так как в реальности «хорошие» алгоритмы имеют сложность не более, чем кубическую. На самом деле даже линейную, так как если мы запускаем алгоритм на современных объёмах данных (терабайты), то ничего, кроме алгоритма линейной или линеарифмической сложности, у нас не отработает. Исторически удавалось все полиномиальные задачи 4-5 степени постепенно снизить хотя бы до квадратичного алгоритма или квадратичного, умноженного на логарифм. Такие алгоритмы постепенно упрощались и в итоге упростились до чего-то разумного. Но теорема о временной иерархии говорит о том, что такое возможно не всегда. Иначе говоря есть задачи, которые находятся на каком-то этаже этой иерархии, и ниже они никогда не упадут.

Итак, первый вывод, который можно сделать из доказательства — что иерархия, как говорят, не схлапывается. Иерархия схлапывается, если она кончается на каком-то этапе (например, n, n^2, n^3, а дальше все алгоритмы кубические). В нашем случае иерархия не схлопнулась, она растёт, поднимаясь в бесконечность. Каждая новая степень в этом объединении даёт нам новые задачи, которые мы можем решать.

Второй вывод, несколько неожиданный, заключается в том, что $P \nsubseteq EXPTIME$. Начиная с некоторого n, экспонента будет всегда больше полинома, а это значит, что $DTIME(f^3(2n+1)) \subset DTIME(2^{f(n)})$. Начиная с какого-то n, экспонента даёт нам больше времени, она позволяет нам решать больше задач.

Данное утверждение верно для любой степени полинома, поэтому любая f всегда лежит внутри своей экспоненты. А раз любой полином лежит внутри экспоненты, то и весь класс P лежит строго внутри экспоненты, они не смешиваются. То есть существуют задачи, которые нельзя решить полиномиально.

Пример задачи вывода привести легко, например: вывести множество всех перестановок n-элементного множества. Очевидно, что нельзя за полиномиальное время распечатать экспоненциальное количество символов. Но для задачи разрешения это

не так очевидно, потому что задача разрешения выдаёт только 0 или 1. Это можно только доказать, и мы это доказали.

1.3 Класс NP

Определение 6. *NTIME(f) — это множество всех задач, которые разрешимы на НМТ за время O(f).*

Определение 7. *Класс* $NP = \cup_{k=1}^{\infty} NTIME(n^k)$ *— это класс задач, которые разрешимы за полиномиальное время на НМТ.*

Для класса NP точно так же можно доказать теорему о временной иерархии. Но центральная задача здесь заключается в следующем:

$P \stackrel{?}{=} NP$

Мы не знаем этого, однако нам как-то надо научиться сравнивать эти классы. Нам точно известно, что $P \subseteq NP$, потому что ДМТ — это частный случай НМТ. Также мы знаем, что $NP \subseteq EXPTIME$. Почему? ДМТ на каждом шаге может «раздвоиться». Допустим, она сделала всего полином шагов, так как это класс NP. На каждом из этих шагов она может «раздвоиться», поэтому она сделала 2 в степени полином шагов, а это экспоненциальное число шагов, значит, все возможные ветки можно перебрать за экспоненциальное время. Поэтому если мы начнём моделировать НМТ обходом в ширину графа её конфигураций, то мы закончим за экспоненциальное время. Также мы знаем, что $P \nsubseteq EXPTIME$, поэтому как минимум одно из этих неравенств строгое (но мы не знаем, какое):

$P \subseteq NP \subseteq EXPTIME$

Сейчас нам необходимо поисследовать дополнительно класс NP, но прежде всего желательно избавиться от приведённого определения класса NP, так как оно нелепое. Мы определяем класс NP через какие-то машины, которые мы сами не в полной мере себе представляем. Никто не любит НМТ, так как математически о них рассуждать неудобно, ведь неудобно рассматривать существование какого-то пути в графе конфигураций, так как мы никогда его не строим. А на пальцах о них рассуждать не получается, потому что какие-то интерпретации мало что имеют общего с действительностью. Поэтому мы хотим «изгнать» НМТ из определения класса NP.

Определение 8. *Класс* NP' *— это множество языков L:*

$\{L : \exists M_L,\ \forall w \in \Sigma^*\ \exists c_w \in \Sigma^* : M_L(w, c_w) = 1 \Leftrightarrow w \in L,\ M_L$ *работает за полином от* $|w|,\ |c_w| < P(|w|)\}$

Здесь M_L — ДМТ, c_w — так называемый сертификат (просто какое-то слово).

Для примера рассмотрим какую-нибудь NP-полную задачу, например, гамильтонов цикл. Задача формулируется так: найти цикл, который проходит по каждой вершине графа только один раз. Рассмотрим задачу разрешения: глядя на граф, понять, существует ли такой цикл. Искать его необязательно — надо просто определить, есть он или нет. Тогда «гамильтонов цикл» — это некоторый язык. Графы как-то кодируются в слова, и те слова, которые соответствуют графам, содержащим гамильтонов цикл, входят в этот язык.

Попробуем придумать, что для графа является сертификатом. Сертификат — это то, что убеждает нас, что слово принадлежит языку. Мы можем легко, глядя на слово и сертификат, понять, что слово принадлежит нашему языку. Для гамильтонова цикла сертификатом является сам цикл. То есть если взять граф и последовательность вершин, мы очень быстро убедимся, что эта последовательность вершин действительно образует гамильтонов цикл. Это можно сделать за полином. Но сертификат должен быть полиномиально ограничен (в нашем случае это так), и должна быть определена процедура сертификации. Граф — это в худшем случае квадрат от количества вершин, а последовательность вершин — это корень от размера графа, что в любом случае меньше полинома.

Ещё одна NP-полная задача — это разбиение: разбить множество чисел на группы так, чтобы их суммы совпали. Условием задачи является набор чисел. Сертификатом является какое-то решение, например, индексы элементов.

Такое определение говорит о том, что NP-полные задачи — это такие задачи, решение которых легко проверить. То есть если нам дали задачу и дали её решение, мы легко (за полиномиальное время) можем это решение проверить.

Теорема 2. $NP = NP'$

Доказательство. 1. Докажем, что $NP \subseteq NP'$ — это достаточно очевидно.

Для этого покажем, что $\forall L(L \in NP) \to (L \in NP')$. Раз L лежит в классе NP, значит существует НМТ, которая решает её за полиномиальное время. Нам нужно указать какой-то сертификат и построить анализатор M_L. В качестве сертификата мы можем взять информацию о том, куда повернула НМТ на каждой развилке, то есть сделать некий «путеводитель». Он будет иметь полиномиальную длину, потому что развилок столько же,

сколько тактов, а тактов не более, чем полином от входа, потому что задача принадлежит классу NP. Анализатор строится легко — нужно просто пройти по этим развилкам в соответствии с «путеводителем».

2. Докажем, что $NP' \subseteq NP$.

У нас имеется ДМТ. Мы пишем на её ленте слово и какой-то сертификат (он может быть длиннее, но имеет полиномиальную длину). После этого ДМТ (анализатор) скажет нам «да» (выдаст единицу). Используем это, чтобы решить исходную задачу: принадлежит ли слово языку. Слово принадлежит языку, когда существует сертификат, для которого ДМТ после этой процедуры выдаст положительный результат. Мы перебираем все сертификаты на ДМТ, делая полиномиальное число шагов. Каждый раз ДМТ «раздваивается» и пишет 0 или 1 в этот сертификат. Таким образом, мы элегантно генерируем все возможные сертификаты (экспоненциальное число) за полиномиальное число шагов, а далее запускаем для каждого из них проверку (детерминированную процедуру), и если хотя бы в одном из случаев проверка завершится успехом, значит слово принадлежит языку. □

Таким образом, мы убрали НМТ из определения NP. Зато теперь у нас в определении фигурирует сертификат, который должен существовать, необходимо найти его, но это всё-таки проще, чем работать с неинтуитивной НМТ.

1.4 Сводимость по Карпу

Далее обсудим связь сложности разных задач. Для этого введём ещё одно определение.

Определение 9. *Сводимость по Карпу:* $L_1 \underset{c}{\rightsquigarrow} L_2$, *если* $\exists f : L_1 \to L_2 : w \in L_1 \Leftrightarrow f(w) \in L_2$.

Здесь f — полиномиально вычислимая функция (т.е. функция, которую можно вычислить с помощью МТ за полином), которая сохраняет сложность, т.е. это полиномиально вычислимый гомеоморфизм.

Допустим, нам удалось найти функцию f, которая сводит L_1 к L_2 по Карпу:

$f : L_1 \underset{c}{\rightsquigarrow} L_2$

Допустим, мы можем быстро решать задачи из L_2. Это значит, что мы можем быстро решать задачи из L_1. Берём слово,

которое надо проверить, с помощью полиномиального алгоритма превращаем его в другое слово, принадлежность этого второго слова к L_2 определяем быстро. А эта принадлежность имеет место тогда и только тогда, когда имеет место принадлежность исходного слова, значит, мы научились быстро решать эту задачу. Также мы показываем, что L_1 — не более сложный язык, чем L_2.

Определение 10. *C-трудная задача* $M_c : \forall N \in C \; N \underset{c}{\rightsquigarrow} M$.

Иначе говоря, M_c — это самая трудная задача в своём классе: если мы умеем решать M, то мы умеем решать все задачи из класса C. M — это такая задача, к которой сводится по Карпу любая задача из C.

Определение 11. *C-полная задача — это задача* M_c*, являющаяся C-трудной, и* $M_c \in C$.

Пример: рассмотрим язык L_1, состоящий из одного слова 1: $L_1 = \{1\}$. Далее рассмотрим какой-то язык L_P из класса P. Построим сведение по Карпу L_P к этому языку. Берём слово языка L_P. У нас есть ДМТ, которая может за полином установить принадлежность этого слова к языку L_P, так как это полиномиальный язык. Так как $L_P \in P$, то для любого слова мы можем за полиномиальное время распознать, находится оно в этом языке или нет, и вывести 1 или 0 соответственно.

Теорема 3. $\forall L_P \in P \; L_P \underset{c}{\rightsquigarrow} L_1$

Доказательство. По определению сводимости по Карпу, необходимо найти функцию f:

$f : f(w) \in L_1 \Leftrightarrow w \in L_P$.

Так как L_1 — одноэлементный язык, то:

$f : f(w) = 1 \Leftrightarrow w \in L_P$.

Строим функцию f следующим образом:

$$f(w) = \begin{cases} 1, w \in L_P \\ 0, w \notin L_P \end{cases}$$

Эта функция должна быть полиномиально вычислимой. В данном случае она таковой является, потому что $L_P \in P$, а это значит, что существует МТ, которая именно эту задачу решает за полином. Поэтому f(w) — это полиномиально вычислимая функция, и любой язык из L_P сводится к L_1. Это означает, что L_1 — P-трудный. □

Понятно, что любой непустой язык является P-трудным, так как нам необходимо наличие в нём всего одного элемента. Поэтому класс P — это класс задач, которые все являются P-трудными и P-полными. Это связано с тем, что в определении сводимости по Карпу мы потребовали, чтобы f была полиномиально вычислимой. Для LOGSPACE-сводимости требование более жёсткое: f должна быть вычислимой на логарифмической памяти.

NP-трудный язык найти очень легко — можно взять какой-нибудь язык из EXPTIME. Найдём NP-полную задачу.

Определим язык {L: (M, x): M — полиномиально ограниченная ДМТ, x: $\exists y : M(x, y) = 1, |y| < P(x)$} Это — NP-трудная задача по определению NP'. Также очевидно, что это NP-полная задача, так как существует сертификат (y).

Таким образом, пример какой-то NP-полной задачи придумать очень легко, точно так же, как легко было найти пример какой-то проблемы, которая алгоритмически неразрешима. В классе NP ничего сложнее, чем NP-полная задача, быть не может. Но открытием было не то, что существуют NP-полные задачи, а то, что некоторые из них имеют очень простую формулировку и встречаются на практике, так же, как это было с диофантовыми уравнениями.