Классы сложности

Введя понятие времени работы алгоритма, мы сталкиваемся с естественной необходимостью классификации задач в зависимости от “сложности” их решения. Для этого используется понятие *классов сложности*, которые объединяют в себе задачи со схожими требованиями ко времени, памяти или каким-либо другим вычислительным ресурсам.

Задачи разрешимости

Знакомство с классами сложности мы начнём с основных классов для задач разрешимости (*decision problems*), то есть, таких, которые подразумевают ответ “да” или “нет”. Например:

1. Верно ли, что число m простое?
2. Верно ли, что в графе G есть гамильтонов путь?
3. Верно ли, что система уравнений $Ax = b$ совместна?

Помимо задач разрешимости есть другие типы задач, для которых определены свои сложностные классы, например, функциональные задачи (*function problems*), заключающиеся в вычислении некоторых функций и оптимизационные задачи (*optimization problems*), заключающиеся в поиске наилучшего в каком-то смысле ответа. Последние обычно можно также поставить в виде задач разрешимости. Например, *задача Коммивояжёра* заключается в поиске гамильтонова цикла наименьшего веса. Это оптимизационная задача, но мы можем рассмотреть её постановку в виде задачи разрешимости: “Верно ли, что в графе есть гамильтонов цикл веса не больше w?”.

Формально можно сказать, что задача разрешимости заключается в вычислении предиката:

$$P(\cdot) : \Sigma^* \to \{0, 1\}$$

Здесь Σ – конечный алфавит, а Σ^* – множество всех конечных слов, которые можно составить над этим алфавитом. То есть, любую задачу разрешимости можно поставить как задачу проверки принадлежности слов некоторому языку L:

$$P(w) = 1 \iff w \in L = \{w \in \Sigma^* | P(w) = 1\}$$

Класс P

Первый пример класса сложности, который мы рассмотрим – это класс полиномиальных задач **P** (*polynomial*). Полиномиальными называются алгоритмы, которые могут быть выполнены на машине Тьюринга за число тактов, асимптотически ограниченное многочленом от размера входных данных, то есть, $T(w) = O(|w|^c)$ для некоторого c. Соответственно, говорят, что язык L (и соответствующий ему предикат P) принадлежат классу **P** если существует полиномиальный алгоритм $\mathcal{A}$, вычисляющий предикат принадлежности слов языку L.

Исторически полиномиальные алгоритмы называются быстрыми, в противовес “медленным” переборным алгоритмам, которые обычно не полиномиальны. Важность этого класса заключается в том, что такое определение является универсальным для моделей вычисления, которые сводятся друг к другу за полиномиальное время. Если точнее – мы определили класс **P** для детерминированной машины Тьюринга, но аналогичным образом его можно определить для RAM-машины или для реализации ассемблера. При этом код на ассемблере можно преобразовать в машину Тьюринга с полиномиальным замедлением, а любой современный язык может быть скомпилирован в ассемблер с полиномиальным замедлением. Это означает, что классы полиномиальных задач, определённые на машине Тьюринга, RAM-машине и на любом современном языке программирования, совпадают.

Кроме того, если алгоритм будет фиксированное число раз обращаться к полиномиальным алгоритмам, он сам при этом будет оставаться полиномиальным, что подчёркивает ещё одну важную особенность многочленов – алгебраическую замкнутость.

1. (1б) Приведите пример языка, заведомо не принадлежащего $\mathbf{P}$.

2. (3б) Докажите, что если $L_1 \in \mathbf{P}$ и $L_2 \in \mathbf{P}$, то:

1. $L_1 \cup L_2 \in \mathbf{P}$, $\quad L_1 \cap L_2 \in \mathbf{P}$, $\quad L_1 L_2 \in \mathbf{P}$
2. $\overline{L_1} = \Sigma^* \setminus L_1 \in \mathbf{P}$, $\quad L_1^* = \bigcup\limits_{k=0}^{\infty} L_1^k \in \mathbf{P}$

3. (2б) Дано некоторое регулярное выражение r и слово w. Предложите полиномиальный алгоритм, проверяющий слово w на принадлежность языку, порождённому выражением r.

В данной задаче и r, и w являются частью входа, т.е., размер входа равен $|r| + |w|$.

4. (3б) Даны два регулярных выражения r_1 и r_2. Предложите полиномиальный алгоритм, проверяющий, что пересечение заданных ими языков не пусто и оцените время его работы.

5. (1б) Покажите, что язык чисел, которые могут быть представлены в виде $k(k+1)/2$ для некоторого целого числа k, принадлежит классу $\mathbf{P}$.

6. (2б) Покажите, что следующие языки принадлежат классу $\mathbf{P}$:

1. Язык графов, содержащих цикл
2. Язык двудольных графов
3. Язык связных графов
4. Язык деревьев

7. (3б) Даны целые числа $a_1, \ldots, a_k$ и $F_1, \ldots, F_k$. Рассмотрим последовательность, определённую как $F_n = a_1 F_{n-1} + \cdots + a_k F_{n-k}$ для $n > k$. Предложите полиномиальный алгоритм, вычисляющий последние k цифр числа F_{2^k} и оцените время его работы.

1*. (2б) Дана контекстно-свободная грамматика и ДКА. Предложите полиномиальный алгоритм, проверяющий, что пересечение заданных ими языков не пусто и оцените время его работы.

2*. (5б) Рассмотрим бинарную операцию $\circ : \{1, \ldots, n\} \times \{1, \ldots, n\} \to \{1, \ldots, n\}$. Назовём её *степенью ассоциативности* число троек i, j, k таких что:

$$i \circ (j \circ k) = (i \circ j) \circ k$$

Даны числа n и x. Предложите полиномиальный по n алгоритм, который построит операцию, имеющую степень ассоциативности равную x или сообщит, что её не существует.

Класс NP

Говоря о полиномиальных алгоритмах, мы неявно подразумевали, что исполняются они на *детерминированной* машине Тьюринга. Задачи же, которые разрешимы за полиномиальное время на *недетерминированной* машине Тьюринга образуют класс **NP** (*non-deterministic polynomial*).

С определением времени работы алгоритма на недетерминированной машине Тьюринга могут быть некоторые трудности, т.к. не совсем понятно, в чём его стоит считать. Как мы помним, недетерминированная машина Тьюринга принимает некоторое слово если *существует* такой набор переходов, который приведёт её в конечное состояние. Поэтому, говоря о времени работы недетерминированной машине Тьюринга, мы будем подразумевать наименьшее возможное число тактов, после которого машина может принять данное слово.

Такая интерпретация позволяет сформулировать определение для задач из класса **NP** в терминах детерминированной машины. Если есть набор переходов, позволяющий недетерминированной машине отработать за полиномиальное время, мы можем рассмотреть детерминированную машину, которая будет иммитировать действия недетерминированной, при этом в качестве входа дать ей не только слово, но также инструкцию, указывающую какие переходы нужно делать на каждом шаге работы недетерминированной машины.

Это, наконец, позволит нам избавиться от разговора о недетерминированных машинах совсем. Мы будем говорить, что язык L принадлежит классу **NP** если существует такой бинарный предикат $\mathcal{A}(\cdot,\cdot) \in \mathbf{P}$, что если предоставить ему в качестве входа слово $w \in L$ и полиномиально ограниченный *сертификат* $s(w)$, то $\mathcal{A}$ сможет проверить, что w принадлежит языку L:

$$L \in \mathbf{NP} \iff \exists \mathcal{A}(\cdot,\cdot) \in \mathbf{P}\ \exists c : L = \left\{ w \;\middle|\; \exists s, |s| < |w|^c : \mathcal{A}(w,s) = 1 \right\}$$

Неформально $s(w)$ – это доказательство того, что w принадлежит L. Примеры задач из **NP**:

1. Любая задача из класса **P**. Мы можем рассмотреть для таких задач тот же алгоритм и для каждого слова давать пустой сертификат, алгоритм и без него справится.

2. Задача о гамильтоновом пути. В качестве сертификата мы можем предоставить этот путь.

3. Задача о выполнимости КНФ (SAT, *satisfiability problem*). Дана конъюнктивная нормальная форма, нужно проверить, есть ли набор переменных, на котором эта формула принимает значение 1. Сертификатом в данном случае будет выступать соответствующий набор.

4. Язык составных чисел. Сертификат – любой нетривиальный делитель числа.

8. (1б) Покажите, что $L \in \mathbf{NP} \implies L^* \in \mathbf{NP}$.

Класс co-NP

В силу того, что вердикты 0 и 1 у машины Тьюринга неравноценны (чтобы выдать 1 нам нужно найти один любой подходящий сертификат, а чтобы выдать 0 нужно убедиться, что на всех сертификатах мы не получим 1), мы не можем гарантировать, что для дополнения исходного языка $\overline{L}$ будет существовать недетерминированная машина, распознающая его на тех же условиях. Это приводит нас к третьему основному сложностному классу, $\mathbf{co} - \mathbf{NP}$ (от *complementary*).

Класс $\mathbf{co} - \mathbf{NP}$ состоит из дополнений языков из **NP**. Формально:

$$L \in \mathbf{co} - \mathbf{NP} \iff \exists \mathcal{A}(\cdot,\cdot) \in \mathbf{P}\ \exists c : L = \left\{ w \;\middle|\; \forall s, |s| < |w|^c : \mathcal{A}(w,s) = 0 \right\}$$

Примеры задач, принадлежащих классу $\mathbf{co} - \mathbf{NP}$:

1. Любой язык L из класса **P**. В качестве $\mathcal{A}$ можно взять алгоритм, распознающий $\overline{L}$.

2. Язык простых чисел.

3. Язык графов без гамильтоновых циклов.

3*. (1б) Рассмотрим некоторый предикат от двух переменных $f(x,y) : \mathbb{N} \times \mathbb{N} \to \{0,1\}$, который можно вычислить за полином от суммы длин x и y. Верно ли, что задача проверки предиката $g(x) = f(x, y_0)$ на общезначимость для данного на вход y_0 обязательно лежит в $\mathbf{co} - \mathbf{NP}$?

Отношения между основными классами

О сложностных классах есть несколько простых и решённых вопросов и намного больше простых и не решённых. Мы знаем, что $\mathbf{P} \subset \mathbf{NP}$ и $\mathbf{P} \subset \mathbf{co} - \mathbf{NP}$. В большей степени этим наши знания насчёт взаимоотношений этих классов заканчиваются. А вот несколько нерешённых:

1. $\mathbf{P} =^? \mathbf{NP}$. Хотя точного решения этой задачи нет, есть некоторые естественные основания полагать, что $\mathbf{P} \neq \mathbf{NP}$ и по умолчанию мы будем придерживаться этой гипотезы в задачах, если не будет указано обратное.

2. $\mathbf{NP} =^? \mathbf{co} - \mathbf{NP}$. Ещё один открытый вопрос. В настоящее время принято считать по умолчанию, что это не верно и $\mathbf{NP} \neq \mathbf{co} - \mathbf{NP}$.

3. $\mathbf{NP} \cap \mathbf{co} - \mathbf{NP} =^? \mathbf{P}$. Стоит отметить, что если язык принадлежит $\mathbf{NP}$ и $\mathbf{co} - \mathbf{NP}$ одновременно, то это, как правило, считается указанием на то, что он не является $\mathbf{NP}$-полным (мы поговорим об этом понятии на следующей неделе), так как в противном случае выполнялось бы предыдущее равенство. Так, долгое время было известно, что язык простых чисел принадлежал обоим классам и в 2002 году был опубликован детерминированный полиномиальный алгоритм, проверяющий простоту чисел.

Принадлежность языка простых чисел $\mathbf{NP} \cap \mathbf{co} - \mathbf{NP}$

Как мы заметили выше, язык составных чисел принадлежит классу $\mathbf{NP}$, откуда следует, что язык простых чисел `PRIME` по определению принадлежит $\mathbf{co} - \mathbf{NP}$. Построим для него сертификат простоты. Из курса алгебры известно, что число p простое если и только если у него есть образующий элемент порядка $p - 1$:

$$p \in \mathtt{PRIME} \iff \exists g : \mathrm{ord}(g) = p - 1$$

Его существование показывает, что мультипликативная группа остатков по модулю p содержит ровно $p-1$ элемент, из чего непосредственно следует простота p, так как мультипликативную группу составляют числа, взаимно простые с p. Но привести его нам будет недостаточно – нужно показать, что он действительно образующий, что эквивалентно тому, что первая после нулевой степень, в которой он равен 1 равна $p-1$. Рассмотрим a такое что $\gcd(a, m) = 1$. Если $a^b \equiv a^c \equiv 1 \pmod m$, то $a^{\gcd(b,c)} \equiv 1 \pmod m$, это следует из алгоритма Евклида.

Значит, если $g^{p-1} \equiv 1 \pmod p$ и $g^a \equiv 1 \pmod p$, то заменой $a' = \gcd(p-1, a)$ мы можем считать, что a – делитель $p - 1$. Но $g^a \equiv 1 \pmod p \implies g^{ab} \equiv 1 \pmod p$. Значит, мы можем вместо a взять некоторое число, которому до $p - 1$ недостаёт лишь одного простого множителя.

Если $p - 1 = p_1^{k_1} \dots p_m^{k_m}$, нам будет достаточно проверить лишь числа вида $\frac{p-1}{p_i}$, коих $O(\log p)$. Значит, мы можем включить числа p_i в наш сертификат. Но этого всё ещё не достаточно, чтобы показать простоту p. Для чисел p_i нам тоже нужно предоставить сертификат, уже их простоты, что позволит нам считать, что мы корректно нашли факторизацию числа $p - 1$. Это, наконец, завершит построение сертификата.

Фактически мы можем считать, что сертификат простоты представляет собой дерево, в узлах которого записаны простые числа p и образующие g, подтверждающие их простоту, а в потомках каждого узла находятся узлы, представляющие собой сертификаты простоты для чисел p_i из факторизации $p - 1$.

9. (4б)

1. Постройте NP-сертификат простоты для числа $p = 3911$, $g = 13$. Простыми в рекурсивном построении считаются только числа 2, 3, 5.

2. Докажите, что сертификат простоты, описанный выше является полиномиальным от длины двоичной записи простого числа p.

3. Подробно опишите полиномиальный алгоритм, который, получив на вход простое число и сертификат, проверит, что число простое.