Типы сводимостей. Сводимость по Карпу и по Тьюрингу

Общая информация про сводимости

Решая задачи мы обычно находимся не в вакууме – кроме задачи, которую мы решаем, вокруг есть много задач, которые уже были исследованы кем-то другим. И этим стоит пользоваться – может быть, после некоторых манипуляций с нашей задачей мы обнаружим, что для её решения нам нужно уметь решать некоторую другую, которая может оказаться для нас проще. Например:

1. Мы хотим найти k-й по величине элемент в массиве. Для этого можно отсортировать массив, затем вернуть его k-й элемент, что будет сведением данной задачи к задаче сортировки.

2. Мы хотим найти в графе цикл наименьшей длины (так называемый *обхват*). Чтобы решить данную задачу, мы можем перебрать две соседние вершины в этом цикле, затем удалить ребро между ними и вызвать алгоритм, находящий кратчайший путь между этими вершинами в новом графе. Так мы сведём задачу нахождения цикла наименьшей длины к поиску кратчайшего пути между двумя вершинами.

Но в нашем курсе сведение применяется не столько для того, чтобы решать какие-то задачи, сколько для того, чтобы оценивать, насколько какие-то задачи сложны. Действительно, если мы свели задачу A к задаче B, то с одной стороны это будет значить, что задача A не сложнее B, а с другой – что задача B не проще задачи A. В этом смысле сводимость можно рассматривать как отношение порядка. Теоретическая значимость этого в том, что в некоторых случаях можно выделить внутри класса самые сложные задачи, такие, что если мы решим их, то сможем свести к ним любые другие задачи из этого класса.

При этом ранее мы говорили о сводимости только на интуитивном уровне. Если мы захотим работать с чем-то более формальным, окажется, что уже было введено множество различных типов сводимостей, с которыми можно ознакомиться здесь. Какую именно сводимость стоит использовать – вопрос, ответ на который в первую очередь зависит от того, в каком классе задач мы работаем. Обычно мы ожидаем, что класс будет замкнутым относительного выбранного вида сводимости. Это значит, что если мы работаем в классе $\mathcal{A}$ и некоторая (произвольная) задача L_1 сводится к задаче $L_2 \in \mathcal{A}$, то L_1 также принадлежит $\mathcal{A}$. Или, проще говоря, за пределами $\mathcal{A}$ нет задачи, которая, согласно данной сводимости окажется проще какой-то задачи из $\mathcal{A}$. Формально:

$$\text{класс } \mathcal{A} \text{ – замкнут по } \leq_* \iff \forall L_1 \notin \mathcal{A}, L_2 \in \mathcal{A} \hookrightarrow L_1 \not\leq_* L_2$$

Для рассматриваемых нами классов таким свойством обладают полиномиальные сводимости.

1. (3б) Рассмотрим следующие функциональные задачи:

1. Задача сортировки. Даны различные целые числа $a_1, \dots, a_n$. Нужно найти перестановку $p_1, \dots, p_n$ такую что $a_{p_1} < a_{p_2} \iff p_1 < p_2$. То есть, перестановка задаёт порядок чисел от наименьшего к наибольшему.

2. Задача о выпуклой оболочке. На плоскости дан набор из n целых точек $(x_1, y_1), \dots, (x_n, y_n)$, координаты точек попарно различны. Нужно найти их выпуклую оболочку, то есть, наименьшее по включению выпуклое множество, содержащее все эти точки.

   Известно, что таким множеством будет многоугольник, опирающийся на некоторый поднабор этих точек и содержащий в себе или на своей границе все точки множества. Поэтому выходом к задаче должен быть набор чисел $p_1, \dots, p_m$ такой, что вершинами этого многоугольника в порядке обхода против часовой стрелке являются точки именно с такими номерами. При этом точка p_1 является лексикографически минимальной в этом наборе, то есть, она обладает наименьшей x-координатой, а среди точек с такой x-координатой – наименьшей y-координатой.

Вам нужно построить сведение первой задачи ко второй в следующей модели сведения:

Пусть есть две функции над словами из алфавита Σ, $\mathcal{A} : \Sigma^* \to \Sigma^*$ и $\mathcal{B} : \Sigma^* \to \Sigma^*$. Сведением функции $\mathcal{A}$ к функции $\mathcal{B}$ мы будем называть такую вычислимую за линейное время функцию $f : \Sigma^* \to \Sigma^*$ что $\mathcal{A}(x) = \mathcal{B}(f(x))$.

Считайте, что предикат $a \le b$ и функции $a + b$ и a^2 от целых чисел вычисляются за $O(1)$.

Результат данной задачи показывает, что задача сортировки не более сложна, чем задача нахождения выпуклой оболочки. В частности, если в данной модели верно, что сортировка требует $\Omega(n \log n)$ операций, то построенное сведение будет показывать, что это верно и про задачу нахождения выпуклой оболочки.

1*. (0.5б) Предложите алгоритм, работающий за время $O(n \log n)$, находящий выпуклую оболочку заданного набора из n точек в двумерном пространстве.

Полиномиальные сводимости

Основные две полиномиальные сводимости – это сводимости по Куку и по Карпу.

1. Говорят, что задача A сводится к задаче B *по Куку* (то же самое – *по Тьюрингу*, обозначение – $A \le_T B$) если существует полиномиальный алгоритм, решающий задачу A при условии, что он может за $O(1)$ обращаться за решением задачи B к "оракулу". Эта сводимость кажется интуитивной, но её применение влечёт ряд неприятных последствий. Например, язык L всегда сводится по Тьюрингу к дополнительному языку $\overline{L}$, следовательно эта сводимость, например, не позволяет различать классы $\mathbf{NP}$ и $\mathbf{co-NP}$.

2. Основная сводимость, которую мы будем рассматривать в курсе – сводимость *по Карпу*. Мы говорим, что язык L_1 сводится к языку L_2 по Карпу ($L_1 \le_m L_2$) если есть полиномиально вычислимая функция f, преобразующая слова L_1 в слова L_2. Формально:

$$L_1 \le_m L_2 \iff \exists \underset{\text{poly}}{f} (\cdot) : w \in \Sigma^* \iff L_1(w) = L_2(f(w))$$

   Здесь под $L_i(w)$ подразумевается предикат принадлежности слова w языку L_i.

   Для этой сводимости используется обозначение $\le_m$, так как она также называется *many-to-one* – одному входу задачи L_2 может соответствовать несколько разных входов L_1. На это стоит обратить особое внимание – функция f, упомянутая в определении, не обязана быть взаимно однозначной. Но даже если она такой будет, из этого ещё не будет следовать сводимость L_2 к L_1, так как вопрос о том, является ли полиномиально вычислимой функция, обратная к полиномиально вычислимой, является открытым.

   Из $L_1 \le_m L_2$ можно делать следующие выводы:

   (a) $L_2 \in \mathbf{P} \implies L_1 \in \mathbf{P}$

   (b) $L_1 \notin \mathbf{P} \implies L_2 \notin \mathbf{P}$

   (c) $L_2 \in \mathbf{NP} \implies L_1 \in \mathbf{NP}$

Трудные и полные задачи

Как было сказано, с помощью механизма сводимости мы можем выделить самые сложные задачи в классе. Пусть у нас есть класс сложности задач $\mathcal{A}$ и некоторая задача L. Эту задачу называют:

1. $\mathcal{A}$-трудной ($\mathcal{A}$ – **hard**) если любая задача из $\mathcal{A}$ сводится к ней: $\forall L' \in \mathcal{A} \hookrightarrow L' \le_m L$.

2. $\mathcal{A}$-полной если она $\mathcal{A}$-трудна и при этом сама принадлежит $\mathcal{A}$. Обозначение: $\mathcal{A}$ – **complete**.

Рассмотрим класс $\mathbf{P}$. Какие задачи в нём будут полными? Казалось бы, тут работает тот же аргумент, что при доказательстве принадлежности $\mathbf{P}$ классу $\mathbf{NP}$ – почему бы нам при сведении L_1 к L_2 не встроить алгоритм решающий L_1 в f? Этот аргумент здесь будет работать. Почти.

Есть два языка, для которых он не сработает. По определению сводимости мы хотим, чтобы предикаты принадлежности $L_1(w)$ и $L_2(f(w))$ были равны. Но для этого нужно, чтобы хотя бы их множества значений совпадали, а для языков $A = \varnothing$ и $B = \Sigma^*$ предикаты могут принимать только значения 0 и 1 соответственно, поэтому как минимум друг к другу их сводить нельзя, а значит, полными в классе $\mathbf{P}$ они не являются.

Для любых же $L_1 \in \mathbf{P}$ и $L_2 \notin \{\varnothing, \Sigma^*\}$ мы можем построить сводимость $L_1 \leq_m L_2$. Пусть есть слова $w_0, w_1 \in \Sigma^*$ такие что $L_2(w_0) = 0$ и $L_2(w_1) = 1$. Тогда мы можем рассмотреть в качестве нашей функции $f(w) = w_{L_1(w)}$. Эта функция вычисляется за полиномиальное время если $L_1 \in \mathbf{P}$[1], таким образом $\mathbf{P} - \mathbf{complete} = \mathbf{P} \setminus \{\varnothing, \Sigma^*\}$[2].

NP-полные задачи

В зависимости от выбранного типа сводимости, полных задач для класса может не существовать вообще. К счастью, в классе **NP** они есть. Основу для теории **NP**-полных задач положила **теорема Кука-Левина:** Задача о выполнимости булевой формы в конъюнктивной нормальной форме (КНФ) является **NP**-полной.

Эту задачу также называют `SAT` (*satisfiablity*). В доказательстве этой теоремы произвольная задача из **NP** в явном виде сводится к `SAT`. Теорема была независимо доказано Стивеном Куком и Леонидом Левиным в 1971. Затем уже в 1972 Карп опубликовал статью, в которой было введено понятие сводимости и получен список из 21 $\mathbf{NP} - \mathbf{complete}$ задачи. Вот некоторые из этих задач:

1. `SAT`. Есть ли набор значений, на котором данная формула в КНФ равна 1.
2. `CLIQUE` и `INDEP`. Есть ли в данном графе *клика/независимое множество* (множество попарно смежных/несмежных вершин соответственно) заданного размера k.
3. `VERTEX-COVER`. Есть ли в данном графе *вершинное покрытие* (множество вершин таких, что каждое ребро графа инцидентно хотя бы одной из них) заданного размера k.
4. `HAMPATH` и `HAMCYC`. Есть ли в данном графе *гамильтонов* (проходящий по каждой вершине ровно один раз) *путь/цикл* соответственно.
5. `COLOR`. Верно ли, что граф можно раскрасить в k цветов так, чтобы никакие смежные вершины не имели одинаковый цвет.
6. `KNAPSACK`. Задача о рюкзаке. Верно ли, что из набора чисел $a_1, \ldots, a_n$ можно выбрать поднабор, чья сумма будет равна m?

Полиномиальный алгоритм для любой из этих задач решил бы вопрос $\mathbf{P} = \mathbf{NP}$, так как любая задача из **NP** может быть сведена по Карпу к любой из них.

2. (2б) Покажите, что $L \in \mathbf{NP} - \mathbf{complete} \iff \overline{L} \in \mathbf{co} - \mathbf{NP} - \mathbf{complete}$.

3. (1б) Постройте сводимости, показывающие, что `CLIQUE` $\leq_m$ `INDEP` и `INDEP` $\leq_m$ `CLIQUE`.

4. (3б) Постройте сводимости, показывающие, что `HAMPATH` $\leq_m$ `HAMCYC` и `HAMCYC` $\leq_m$ `HAMPATH`.

[1] Здесь может показаться, что мы должны знать w_0 и w_1, чтобы построить такую функцию. Но в определении требовалось только существование такой функции и, раз w_0 и w_1 существуют, то существует и соответствующая функция. Она будет полиномиальной в силу того, что $|w_0| + |w_1|$ – некоторая константа.

[2] Вообще говоря, рассматривать **P**-полноту с данной сводимостью не очень содержательно, поэтому в данном классе полноту обычно рассматривают по другим типам сводимостей. Подробнее почитать можно здесь.

Примеры сводимостей

SAT $\le_m$ 3-SAT

3-SAT – это версия задачи о выполнимости, в которой в каждом конъюнкте может быть не более 3 переменных. Рассмотрим произвольную формулу в КНФ. Без ограничения общности будем считать, что её первый конъюнкт содержит больше трёх переменных:

$$(x_1^{k_1} \vee x_2^{k_2} \vee \cdots \vee x_m^{k_m}) \wedge (\dots) \wedge \cdots \wedge (\dots)$$

Мы хотим уменьшить число переменных, логичным действием в данном случае будет попытаться взять любую дизъюнкцию и заменить её одной переменной, например, мы можем ввести новую переменную $y = x_1^{k_1} \vee x_2^{k_2}$. Тогда первый конъюнкт примет вид $(y \vee x_3^{k_3} \vee \cdots \vee x_m^{k_m})$ и будет иметь на одну переменную меньшей в своей записи. Однако нам нужно добавить в нашу формулу условие, что $y = x_1^{k_1} \vee x_2^{k_2}$. Для этого можно, воспринимая данное выражение как булеву формулу, записать её в виде КНФ и просто приписать к исходной формуле. КНФ можно легко получить из таблицы истинности для функции $y =^? a \vee b$:

y	a	b	f(y, a, b)
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Данные действия стоит продолжать, пока все конъюнкты не будут иметь размер меньше 4.

5. (1б) Постройте для данной функции ДНФ и КНФ и покажите, что построенное преобразование действительно является полиномиальной сводимостью.

6. (2б) Расставьте и обоснуйте **P**, **NP – complete** и **co – NP – complete** в таблице:

Задача / Запись	Выполнимость	Тавтологичность
КНФ		
ДНФ		

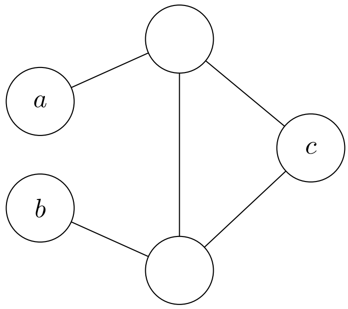
- Выполнимость – проверить, что есть набор переменных, на котором формула равна 1.
- Тавтологичность – проверить, что на всех наборах переменных формула равна 1.

3-SAT $\le_m$ 3-COLOR

3-COLOR – частный случай COLOR при $k = 3$. Дан граф, нужно проверить, можно ли раскрасить его вершины в три цвета, чтобы никакое ребро не соединяло вершины одного цвета. Построим сводимость 3-SAT $\le_m$ 3-COLOR. Первым делом добавим в наш граф три попарно соединённые вершины – эти вершины будут иметь три различных цвета, назовём их белым, чёрным и серым.

Теперь пусть в нашей КНФ n переменных и m конъюнктов. Сопоставим каждой переменной x две вершины x_0 и x_1 и соединим их друг с другом, а также с серой вершиной. Таким образом, каждой переменной будет соответствовать либо белая, либо чёрная вершина x_0 и окрашенная в противоположный цвет вершина x_1.

Рассмотрим теперь каждый конъюнкт в отдельности. Нам нужно, чтобы хотя бы одна вершина, упомянутая в нём была чёрной. Рассмотрим граф следующего вида:



Можно проверить, что при раскраске в 3 цвета вершина c должна быть раскрашена либо в цвет a, либо в цвет b. Без ограничения общности можно считать, что в каждом конъюнкте имеется ровно три *различные* переменные.

7. (1б) Обоснуйте это, построив сводимость `3-SAT` $\leq_m$ `EXACTLY-3-SAT`, где `EXACTLY-3-SAT` – язык выполнимых 3-КНФ, в каждом из конъюнктов которых содержится три различные переменные.

Теперь для конъюнкта $(a \vee b \vee c)$ Мы можем “навесить” данную конструкцию на a и b с выходом в вершине d, а затем на вершины c и d с выходом, закреплённым на чёрной вершине. Таким образом, раскрасить этот подграф будет возможно только если хотя бы одна из $\{a, b, c\}$ будет чёрного цвета.

8. (1б) Покажите, что данное преобразование действительно является искомой сводимостью.

9. (2б) Покажите, что если `3-COLOR` $\in$ **P**, то за полиномиальное время можно не только определить, что граф допускает раскраску вершин в три цвета, но и найти какую-то 3-раскраску если она существует.

10. (8б) Вам известно, что $\{$`SAT`, `COLOR`$\} \subset$ **NP – complete**. За каждый оставшийся пункт из списка выше, про который вы докажете, что он **NP – complete**, вы получите 2 балла.

При доказательстве допустимо обратиться к различным источникам, где такие сводимости уже построены. Однако настоятельно рекомендуется сперва попытаться построить сводимость самостоятельно. В случае, если вы пользуетесь готовыми сводимостями, обязательно пропустите их через себя, разберитесь, почему они корректны и опишите своими словами, при возможности попытайтесь их упростить.

Полиномиальные частные случаи

`2-COLOR`

Покажем, что раскрашиваемость в два цвета можно проверить за полиномиальное время. Без ограничения общности считаем, что граф связен. Рассмотрим в нём произвольное любое дерево. С точностью до инверсии цветов, оно может быть покрашено в два цвета единственным образом, мы можем рассмотреть эту покраску и просто проверить, что рёбра вне дерева с ней согласуется.

11. (1б) Докажите, что граф можно раскрасить в два цвета тогда и только тогда, когда он не содержит циклов нечётной длины. Предложите полиномиальный алгоритм, находящий в графе любой цикл нечётной длины, если он есть.

`2-SAT`

Дана формула в КНФ, в каждом конъюнкте которой не больше двух переменных. Рассмотрим следующий алгоритм: Возьмём любую переменную x и попытаемся заменить её на 0. После этого конъюнкты вида $(x \vee y)$ превратятся в просто y, а $(\overline{x} \vee y)$ – в 1. В первом случае это будет значит, что мы узнали, что $y = 1$, поэтому сделаем аналогичную процедуру и для этой переменной. В какой-то момент мы либо определим значения нескольких переменных и останемся с 2-КНФ, которая затрагивает только оставшиеся, либо придём к противоречию. Утверждается, что в первом случае выполнимость новой формулы эквивалентна выполнимости исходной, а во втором – выполнимость формулы, которую мы бы получили подстановкой $x = 1$, эквивалентна исходной. Таким образом, мы в любом случае единожды запустим рекурсивную процедуры от формулы с меньшим числом переменных.

12. (2б) Обоснуйте корректность данного алгоритма и оцените время его работы.

Альтернативное решение: Вместо каждого конъюнкта $(a \vee b)$ можно рассмотреть пару импликаций $(\overline{a} \to b)(\overline{b} \to a)$. Таким образом, мы можем построить ориентированный граф, в котором каждой переменной будет соответствовать две вершины, а импликациям – дуга между вершинами. Утверждается, что исходная формула выполнима тогда и только тогда, когда нет такой переменной, что есть путь по импликациям как из x в $\overline{x}$, так и обратно, то есть, что обе соответствующие переменные лежат в одной и той же компоненте сильной связности.

13. (2б) Обоснуйте корректность данного алгоритма и оцените время его работы. Предложите, как с его помощью за полиномиальное время не только проверить формулу на выполнимость, но и найти какой-нибудь выполняющий набор переменных.

2*. (0.5б) Предложите алгоритм с линейным по входу временем работы, который по данной `2-SAT` найдёт набор переменных, на котором она истинна или сообщит, что такого набора нет.