

Лекция 11

Подготовил Савичев Игорь, КБ-301

30.11.15

Теория алгоритмов 2015

Повторение

В прошлый раз мы остановились на полиномиальной иерархии. И успели ввести одно определение.

Определение

$$(\Pi) \sum^k p = \{L : \exists \text{ ДМТ } M \ w \in L \Leftrightarrow \exists c_1 \forall c_2 \exists c_2 \forall c_3 \dots Q c_k : M(w, c_1, c_2, \dots, c_k) = 1 \}.$$

Кроме того, мы поняли, что можно ввести функциональный класс.

Определение

$$F \sum^k p = \{R : \exists \text{ ДМТ } M \ (x, y) \in R \Leftrightarrow \forall c_1 \exists c_2 \dots Q c_{k-1} : M(x, y, c_1, \dots, c_{k-1}) = 1 \}.$$

Поговорим немного об этих классах. Очевидно следующее утверждение:

Утверждение

Если $\exists k \sum^k p = \prod^k p \Rightarrow \sum^{k+1} p = \sum^k p$, то есть, если эти классы равны, то полиномиальная иерархия коллапсирует и дальше никаких классов нет.

Доказательство

Без доказательства.

Сформулируем ещё одну задачу.

Задача

$QSAT - k - \exists x_1^1 \dots x_{k_1}^1 \forall x_1^2 \dots x_{k_2}^2 \exists x_1^3 \dots x_{k_3}^3 \dots : F(x_1^1 \dots) = 1$

. Где Q - какой-то квантор.

SAT - Задача выполнимости булевых формул.

k - количество переменных.

Замечание

И из соображений, эквивалентных теореме Кука, мы можем доказать, что эта задача полна в каждом классе, которые мы привели в начале. Соответственно,

$\exists SAT - 2$ полна в $\sum^2 p$ и

$\forall SAT - 2$ полна в $\Pi^2 p$.

Введём ещё одно утверждение о коллапсирующей иерархии.

Утверждение

Если $\exists k \sum^k p = \prod^k p \Rightarrow PH = \sum^k p$,
где $PH = \cup_k \sum^k p \cup \prod^k p$.

Доказательство

Без доказательства.

Суть этого утверждения в том, что все эти вопросы о равенстве классов P и NP , они в этой иерархии до самого верха, то есть, если $P = NP$ вся иерархия сваливается в P .

На этом мы закончим с полиномиальной иерархией и двинемся дальше.

SPACE-COMPLEXITY

Вторая сложность алгоритмов.

Мы до сих пор интересовались только временной сложностью алгоритмов: сколько тактов проработает данный алгоритм на каком-то входе.

Сейчас же введём вторую сложность алгоритмов - *ёмкостную сложность*

Для этого нам понадобится ввести модифицированную МТ.

Определение

МТ с двумя лентами.

- 1. read-only - на ней записано условие (входящие данные).*
- 2. обычная.*

И вот необходимый размер обычной ленты и есть ёмкостная сложность.

То есть, сколько ячеек будет затрачено на этой обычной ленте.

Соответственно, мы можем выделить следующие классы.

Определение

$DSPACE(f)$ - множество алгоритмов с ёмкостной сложностью $O(f(n))$.

Определение

$NSPACE(f)$ - множество алгоритмов с ёмкостной сложностью $O(f(n))$ на НМТ.

И раз у нас появляются $DSPACE$ и $NSPACE$, мы хотим ввести классы сложности.

Какие классы нам интересны ?

Определение

$L = DSPACE(\log)$ - класс алгоритмов, которые работают за логарифмическое время.

Определение

$NL = NSPACE(\log)$ - класс алгоритмов, которые работают за логарифмическое время.

Определение

$PSPACE = \cup_k DSPACE(n^k)$

Определение

$NPSPACE = \cup_k NSPACE(n^k)$

Теперь мы можем ввести строгое определение ёмкостной сложности.

Определение

Ёмкостная сложность - это точная верхняя граница количества дополнительной памяти необходимой для работы МТ в зависимости от размера входа.

То есть, это какая-то функция, которая на размер входа выдаёт точную верхнюю границу.

Как между собой соотносятся P и $PSPACE$?

Если мы вспомним, что такое P и $PSPACE$, то ответ придёт довольно легко.

Замечание

P - это те алгоритмы, которые работают за полиномиальное время.

Доказательство

Сложно за полиномиальное время "изгадить" больше ячеек памяти, чем полином.

Поэтому очевидно, что $P \subseteq PSPACE$.

Также очевидно более сложное условие.

Замечание

$$DTIME(f) \subseteq DPSACE(F).$$

Доказательство

Нужно просто понять, что если вы работаете k шагов, то больше чем k ячеек вам не понадобится.

Другой вопрос сложнее: что больше $PSPACE$ или $EXPTIME$?

Замечание

$PSPACE \subseteq EXPTIME$.

Доказательство

У нас есть некоторое ограничение на работу памяти.

Пусть мы берём в нашу МТ слово w , а это значит, что мы можем использовать места не более чем $|w|^k$, где k - некоторое число.

Теперь нужно вспомнить, что такое конфигурация МТ ? Это то:

1. Что записано на ленте (обычной).
2. Состояния, которых конечное число.
3. И две позиции.

А всего таких конфигураций может быть очень много.

Таким образом, если у нас полиномиальное ограничение на размер памяти, то у нас есть экспоненциальное ограничение на время.

Замечание

$$DTIME(f) \supseteq DSPACE(\log(f)).$$

Сейчас мы можем нарисовать грубую иерархию, чуть позднее мы её уточним.

Предположение

$$L \subseteq P \subseteq PSPACE \subseteq EXPTIME.$$

Где-то здесь потерялись NL , $NPSPACE$ и PH .

Сейчас мы постараемся на все эти вопросы ответить.

Теорема Савича (Savich)

Теорема

$$PSPACE = NPSPACE$$

Доказательство

Пусть \exists алгоритм поиска пути в графе G из s в t , занимающий $\log^2(G)$ (кол-ва вершин) памяти.

Мы имеем алгоритм $A(s, t, k)$ - \exists путь из s в t из k вершин.

1. if $(k = 0)$ { if $(s = t)$ ret 1; else ret 0; }
2. if $(k = 1)$ { if $(s = t) \in E$ ret 1 else ret 0; }
3. forEach $(v \in V \text{ s.t. } s \neq v \neq t)$ if $(A(s, v, \lfloor \frac{k}{2} \rfloor) \&\& A(v, t, \lfloor \frac{k}{2} \rfloor))$ ret 1; else ret 0;

То есть, когда мы ищем путь из s в t , мы пытаемся проверить, что для какой-то вершины в графе есть обе половины пути.

А когда мы найдём эту вершину, то это и будет означать, что существует такой путь.

Итог

Что из этого практически следует ?

Наблюдение

Допустим, что у нас есть какая-то НМТ, память которой ограничена неким числом k .

Число её конфигураций - 2^k .

Весь граф переходов этой НМТ имеет экспоненциальный размер, а как мы помним: Задача о том, остановится ли данная НМТ - это в точности задача поиска пути в пространстве её конфигураций.

А эту задачу мы умеем решать за k^2 .

Введём ещё одну полную задачу:

Задача

По w проверить $\exists c_1 \forall c_2 \dots Q c_k M(w, c_1, c_2, \dots, c_k) = 1$ и $k < P(w)$.

Это выиграшная стратегия в играх, которые не могут расти бесконечно. (например, шашки).