Лекция 4

Сложность недетерминированных вычислений по памяти

(Конспект: Д. Ицыксон)

4.1 Определения недетерминированной машины Тьюринга и ее сложности по памяти

1. *Недетерминированная* машина Тьюринга отличается от детерминированной тем, что функция перехода $\delta(q, c)$ по состоянию q и символу c выдает теперь *множество* троек $\{(q_1, c_1, d_1), (q_2, c_2, d_2), \ldots\}$, где q_i — состояние, c_i — символ, а d_i — направление движения. Таким образом, из каждой конфигурации есть много способов перейти в следующую. Образуется целое дерево вычислений. Соответственно, машина принимает слово тогда и только тогда, когда есть ветвь вычислений, приводящая к принимающему состоянию. Время работы — длина максимальной ветви. Память — максимальная память по всем ветвям.

2. *Недетерминированная* машина Тьюринга — это обыкновенная детерминированная машина с дополнительной лентой (подсказкой), которая доступна только для чтения (readonly). Такая машина принимает слово x, если существует подсказка, при которой машина, получая на дополнительную ленту эту подсказку, а на вход — x, останавливается в принимающем состоянии. Время работы — максимум по всем подсказкам. Для памяти тоже самое, только для

рабочих лент. Память на дополнительной ленте не учитываем — ведь подсказку мы не обязаны даже читать полностью, а писать туда вообще не разрешается (неформально, подсказка соответствует недетерминированному выбору между (q_i, c_i, d_i) на каждом шаге — т.е. количество символов в ней соответствует времени, а не памяти). Оказывается, существенно то, как разрешается считывать символы с ленты подсказки:

(a) *offline*-модель: подсказку можно читать, как любую ленту;

(b) *online*-модель: подсказку можно читать только слева направо.

Обозначим соответствующие классы языков $\mathbf{NSpace}_{\text{off}}(f(n))$ и $\mathbf{NSpace}_{\text{on}}(f(n))$ соответственно.

Очевидно, что определение 1 эквивалентно online-модели.

Offline непригодна для измерений, так как она по памяти экспоненциально сильнее online.

Утверждение 4.1. $\mathbf{NSpace}_{\text{on}}(2^{f(n)}) \subseteq \mathbf{NSpace}_{\text{off}}(O(f(n)))$.

Доказательство. Пусть $L \in \mathbf{NSpace}_{\text{on}}(2^{f(n)})$, M — соответствующая ему (online) НМТ.

Для любого слова x из L существует вычисление машины M, приводящее к принимающему состоянию. Запишем это принимающее вычисление как подсказку для offline машины. В строчку: начальная конфигурация, далее все конфигурации подряд, и, наконец, принимающая конфигурация (конфигурация включает в себя все необходимое для описания «состояния», в котором находится машина: текущее состояние, положение *всех* головок (двоичное число!), содержимое *рабочих* лент).

Теперь осталось проверить:

1. Правильность синтаксиса подсказки — память $O(1)$.

2. Первая конфигурация — действительно начальная (в частности, входная лента содержит x), а последняя — действительно принимающая — память $O(1)$.

3. Следующая конфигурация правильно получается из предыдущей:

   (a) состояние машины и положение головок изменились в соответствии с функцией δ — память $O(f(n))$ (на счетчик для возвращения к нужному месту предыдущей/последующей конфигурации — где она начинается, мы может найти и так), ибо положение головки — это число, не превосходящее $2^{f(n)}$;

(b) содержимое рабочих лент изменилось в соответствии с функцией δ — для этого нам нужно еще и хранить и увеличивать на единицу счетчик сравниваемой позиции (чтобы сравнивать *одинаковые* позиции конфигураций, а затем возвращаться к *следующей* позиции предыдущей конфигурации), снова память $O(f(n))$.

□

В дальнейшим мы используем только online-модель.

4.2 Факты о сложности недетерминированных вычислений по памяти

Лемма 4.1. *Узнать, есть ли в графе путь между двумя вершинами, можно, используя памяти не более квадрата логарифма числа вершин, т.е.* `Reachability` $\in \mathbf{DSpace}(O(\log^2 n))$.

Доказательство. Определим трехместный предикат $\mathrm{PATH}(x, y, i)$, который означает, что есть путь из x в y длины не более 2^i. (Тогда $\mathrm{PATH}(x, y, \lceil\log n\rceil)$ — искомый ответ.) Очевидно,

$$\mathrm{PATH}(x, y, i) \iff \exists z\ (\mathrm{PATH}(x, z, i-1) \wedge \mathrm{PATH}(z, y, i-1)).$$

Пользуясь этим соотношением, получаем следующий рекурсивный алгоритм.

Шаг рекурсии: пусть в самом конце ленты написано (x, y, i) — тогда, стерев эту тройку, перебираем все z; для каждого из них

- пишем $(x, z, i-1)$ в конец ленты и совершаем рекурсивный вызов,
- если этот вызов закончился успешно, пишем в конце ленты $(z, y, i-1)$ и совершаем еще один рекурсивный вызов,
- если оба рекурсивных вызова закончились успешно, то успешно завершаем вызов.

Если же нужного z не было найдено, то завершаем вызов неудачно.

Рекурсия заканчивается, когда $x = y$ — тогда вызов успешен, либо когда $i = 0$ — тогда надо просто проверить, что в графе есть ребро (x, y).

Ясно, что при работе $\mathrm{PATH}(x, y, \lceil\log n\rceil)$ длина записи на ленте не превосходит $3 \cdot \lfloor\log n\rfloor \cdot$ длину чисел $= O(\log^2 n)$. □

Теорема 4.1. $\mathbf{NSpace}(f) \subseteq \mathbf{DSpace}(f^2)$ *для space constructible функции* $f(n) = \Omega(\log n)$.

Доказательство. Рассмотрим граф, заданный на конфигурациях НМТ, ограниченной по памяти $f(n)$. Нам интересно, есть ли путь из начальной конфигурации в принимающую. Воспользуемся алгоритмом из леммы. Сколько вершин в графе? Их не более $c^{f(n)}$, поскольку ячеек используется не более $f(n)$. Следовательно, по лемме 4.1 достаточно памяти $O(f^2)$. (Заметим, что сам граф нам хранить не надо, так как выяснить, можно ли перейти из конфигурации x в конфигурацию y, можно с памятью $O(f(n))$ — мы это делали в утверждении 4.1.) □

Определение 4.1.

$$\mathbf{PSPACE} = \bigcup_{k\in\mathbb{N}} \mathbf{DSpace}(O(n^k)).$$

$$\mathbf{NPSPACE} = \bigcup_{k\in\mathbb{N}} \mathbf{NSpace}(O(n^k)).$$

Следствие 4.1. $\mathbf{PSPACE} = \mathbf{NPSPACE}$.

Из этого следствия вытекает также, что $\mathbf{NPSPACE} = \mathbf{co\text{-}NPSPACE}$, но из теоремы не вытекает, что $\mathbf{NSpace}(f(n)) = \mathbf{co\text{-}NSpace}(O(f(n)))$. Ниже мы докажем это более тонкое утверждение. Начнем со вспомогательной леммы.

Лемма 4.2. *Размер множества вершин некоторого графа* $G = (V, E)$, *достижимых из заданной вершины* $x \in V$, *можно вычислить на НМТ, используя память* $O(\log |V|)$. *При этом можно их все перечислить.*

Доказательство. Обозначим $S(k)$ — это множество вершин, до которых есть путь из x длины не более k. (Тогда $|S(n)|$ — это требуемый результат.)

Мы будем вычислять $|S(k)|$ индуктивно, используя только $|S(k-1)|$. База индукции: $|S(0)| = 1$.

Пройдемся один раз по всем вершинам $u \in V$ и подсчитаем l — количество вершин $u \in S(k)$.

Как мы определим, что $u \in S(k)$? А вот как: перебираем все вершины v и ищем среди них такую, что $v \in S(k-1) \wedge (v, u) \in E$. (Заодно проверяем, что количество $v \in S(k-1)$ равно $|S(k-1)|$; если не равно, то вычисление оказалось неудачным.)

Вы спросите, откуда же мы узнаем, что $v \in S(k-1)$? На это у нас есть недетерминизм машины. Чтобы это узнать, берем с ленты подсказки одно за другим числа $w_1, w_2, \ldots, w_{k-2}$, которые мы подозреваем в том, что они последовательные вершины на искомом пути длины $\leq k-1$ (соседние вершины в пути могут совпадать) — проверить, что $\forall i \in [0\,..\,k-2]$ $((w_i, w_{i+1}) \in E \vee w_i = w_{i+1})$, нетрудно (здесь w_0 обозначает x, а $w_{k-1} = v$). Если хотя бы одна проверка не прошла, то считаем, что $v \notin S(k-1)$. Таки образом мы заново подсчитываем количество $v \in S(k-1)$; если оно не сошлось с ожидаемым, объявим вычисление неудачным. □

Теорема 4.2. $\mathbf{NSpace}(f(n)) = \mathbf{co\text{-}NSpace}(O(f(n)))$ *для любой space constructible функции* $f(n) = \Omega(\log n)$.

Доказательство. Пусть язык $L \in \mathbf{NSpace}(f(n))$ принимается НМТ M, ограниченной по памяти $f(n)$. Покажем, что существует машина $\bar{M}$, ограниченная по памяти $f(n)$, решающая язык $\bar{L}$. $\bar{M}$ запускает недетерминированный алгоритм перечисления достижимых вершин в графе конфигураций M из начальной конфигурации и проверяет дополнительно на каждом шаге, не является ли вершина принимающим состоянием. Если алгоритм сработал успешно (подсказка правильная), и есть путь в принимающее состояние, то он отвергает; если нет пути, то принимает. (Если подсказка была неправильная, то ветвь — неудачная; впрочем, это как раз и значит, что в этом случае — тоже отвергает.)

```
          /\
         /  \
        /   /\
       /   |  \
      /    |   \
     /     |    \
    /      |     \
неудача  в лемме   в лемме
в лемме    OK,       OK,
 => 0   нашли q_Y  не нашли q_Y
          => 0       => 1
```

□