Юрай Громкович,
Аблаев Фарид Мансурович

АЛГОРИТМЫ И СЛУЧАЙНОСТЬ УРОК 6. ОНЛАЙН АЛГОРИТМЫ И КОНКУРЕНТНЫЙ АНАЛИЗ

В этой статье мы продолжим наш разговор об онлайн проблемах. Мы покажем, как использование случайности при разработке онлайн алгоритмов может помочь улучшить качество решения проблемы.

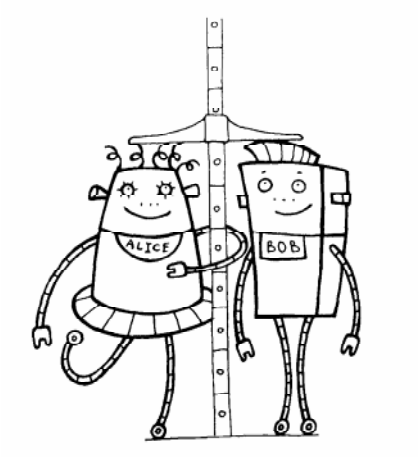
В прошлой статье мы подробно разобрали задачу кэширования [1]. Мы рассмотрели детерминированные стратегии. Сейчас определим численную характеристику качества алгоритма, решающего онлайн проблему.

Повторим некоторые важные определения.

Определение 2. Рассмотрим некоторую онлайн проблему P и онлайн алгоритм A, решающий эту проблему. Алгоритм A будем называть *c-конкурентным*, если существует неотрицательная константа α такая, что для любого $I \in \mathcal{I}$ справедливо соотношение:

$$cost(A(I)) \le c \cdot cost(Opt(I)) + \alpha,$$

где Opt – это оптимальный (идеально возможный) алгоритм для этой задачи, то есть алгоритм, порождающий оптимальный выходной набор. Значение c мы будем также называть *конкурентным отношением* алгоритма A.



Значение c мы будем также называть конкурентным отношением алгоритма A.

Конкурентное отношение алгоритма A на входном наборе I мы также будем обозначать $comp(A(I))$. Отметим, что наличие константы α позволяет алгоритмам, стоимость решений которых отличается от стоимости решений оптимального алгоритма лишь на константу, становиться 1-конкурентными.

Обсудим, какой алгоритм будем считать оптимальным и обозначать Opt. Это алгоритм, который заранее знает весь входной набор. Например, в варианте поездки на горнолыжный курорт алгоритм Opt заранее совершенно точно знает сколько подходящих дней для катания на лыжах будет за время нашего отпуска. Понятно, что в действительности существование такого Opt алгоритма чисто гипотетическое. Алгоритм Opt обладает мощной информацией (некоторые люди сказали бы, что алгоритм Opt обладает возможностями всемогущего чародея и предсказателя), которой не обладает A, и это та дилемма, с которой мы встречаемся при построении онлайн алгоритмов.

При таком соглашении об алгоритме *Opt*, говоря обыденным языком (содержательно), конкурентное отношение дает нам численную характеристику, «цены незнания будущего».

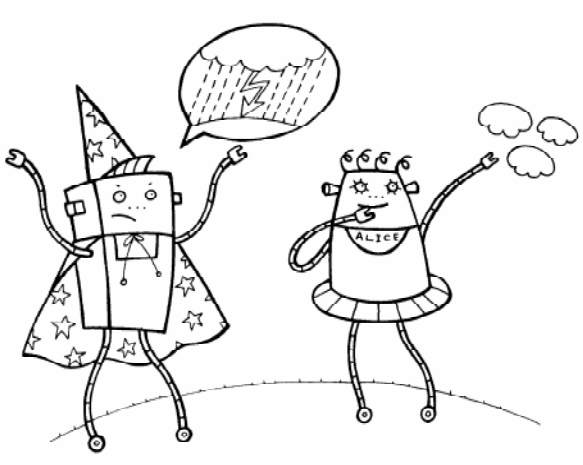
В следующем разделе рассмотрим онлайн задачу кэширования, для которой вероятностный подход по построению алгоритма, ее решающую, дает хорошие результаты.

Задача кэширования (Пейджинга). Одна из очень важных онлайн задач – это задача кэширования или задача пейджинга. Суть ее состоит в следующем.

Физическая память компьютера многослойна. Центральный процессор в первую очередь оперирует своими регистрами, работа с которыми происходит очень быстро, но сама эта память очень дорога. Далее идет кэш процессора, который дешевле и медленнее, далее – оперативная память, следующая память самая дешевая самая медленная – это память, располагающаяся на жестком диске. Мы сфокусируемся на одном уровне иерархии, когда у нас есть медленная физическая память и быстрая, но дорогая память кэша.

Для более эффективного использования памяти она разбивается на страницы (или ячейки, page). Далее процессор и программы оперируют только страницами, а не отдельными байтами. В случае, если пользователь запрашивает некоторую страницу и она уже находится в кэше, то процессор сразу же берет ее из кэша и отдает пользователю. Если же ее в кэше не оказывается, то происходит так называемый «обмен страницами»: процессор выбирает некоторую ячейку из кэша и записывает данные запрошенной страницы в эту ячейку, после чего отдает данные пользователю. Данные же, поверх которых была записана новая страница, пропадают из кэша. Основная задача алгоритма – это определить, поверх какой ячейки стоит записывать новую страницу. Пусть кэш может хранить в себе не более чем k страниц. Обычно кэш заполнен, поэтому можно считать, что в кэше всегда ровно k страниц.

Опишем задачу формально.



...алгоритм Opt обладает возможностями всемогущего чародея и предсказателя, которой не обладает A...

Определение 3. Рассмотрим последовательность целых чисел, представляющих запросы страниц памяти $I = (x_1, ..., x_n)$, $x_i > 0$. Онлайн алгоритм A обслуживает буфер (содержимое кэша) $B = \{b_1, ..., b_K\}$ K целых чисел, где K – фиксированная константа, известная A. До первого запроса буфер инициализируется следующим значением $B = \{1, ..., K\}$. Если A принимает запрос $x_i \in B$, то тогда выдает ответ $y_i = 0$. Если же $x_i \notin B$, то происходит «обмен страницами», и A должен определить заменяемую страницу b_j, при этом $B := B \setminus \{b_j\} \cup \{x_i\}$ и $y_i = b_j$. Стоимость решения I равно количеству обменов страниц, то есть $cost(A(I)) = |\{y_i : y_i > 0\}|$.

Рассмотрим вероятностный алгоритм решения задачи кэширования. Называется он MARK. Подробнее о нем можно прочитать в статье авторов этого алгоритма [2].

АЛГОРИТМ MARK

Будем обрабатывать последовательность запросов, разделяя процесс обработки на фазы. Первая фаза начинается с первого запроса к странице памяти. Заканчивается фаза, когда мы получим запросы на k различных страниц. Следующий запрос относится уже ко второй фазе выполнения нашего алгоритма. Вторая фаза заканчивается тогда, когда мы получим запросы к k различным страницам во второй фазе. И так далее.

В начале каждой фазы будем помечать все страницы в кэше 0. При поступлении запроса на страницу, находящуюся в кэше, будем помечать ее 1. При поступлении запроса на страницу, которой нет в кэше, выберем случайным образом одну из помеченных 0 страниц и заменим ее на запрошенную, новую страницу пометим 1. В конце фазы все страницы в кэше будут помечены 1. В начале новой новой фазы мы обнулим все метки в кэше и повторим наши действия.

Приведем пример. Рассмотрим задачу кэширования со следующими параметрами: размер кэша $k = 3$, кэш перед началом работы алгоритма заполнен значениями $\{1^0, 2^0, 3^0\}$, размер физической памяти $m = 5$. Пометки страниц будем делать верхним индексом над значением в кэше.

Рассмотрим следующий входной набор $I = (1, 2, 4, 2, 4, 2, 3, 1, 5)$. Будем рассматривать состояние кэша в процессе работы алгоритма:

1. $x_1 = 1$. Значение 1 есть в кэше, пометим его 1. Состояние кэша после первого шага работы алгоритма $\{1^1, 2^0, 3^0\}$.

2. $x_2 = 2$. Значение 2 есть в кэше, пометим его 1. Состояние кэша после второго шага работы алгоритма $\{1^1, 2^1, 3^0\}$.

3. $x_3 = 4$. Значения 4 нет в кэше. Выберем случайное значение из кэша среди значений, помеченных 0. Такое значение только одно – 3. Его мы и заменим. Состояние кэша после третьего шага работы алгоритма $\{1^1, 2^1, 4^1\}$.

Заметим,что мы запросили $k = 3$ разных значений: 1, 2, 4. Это значит, что первая фаза работы алгоритма завершена. Сбросим все пометки страниц на 0. Состояние кэша будет $\{1^0, 2^0, 4^0\}$.

Далее начнется вторая фаза.

4. $x_4 = 2$. Значение 2 есть в кэше, пометим его 1. Состояние кэша после четвертого шага работы алгоритма $\{1^0, 2^1, 4^0\}$.

5. $x_5 = 4$. Значение 4 есть в кэше, пометим его 1. Состояние кэша после пятого шага работы алгоритма $\{1^0, 2^1, 4^1\}$.

6. $x_6 = 2$. Значение 2 есть в кэше, оно уже помечено 1. Состояние кэша после шестого шага работы алгоритма $\{1^0, 2^1, 4^1\}$.

7. $x_7 = 3$. Значения 3 нет в кэше. Выберем случайное значение из кэша среди значений, помеченных 0. Такое значение только одно – 1. Его мы и заменим. Состояние кэша после седьмого шага работы алгоритма $\{3^1, 2^1, 4^1\}$.

Заметим, что мы снова запросили $k = 3$ разных значений: 2, 4, 3. Это значит, что вторая фаза работы алгоритма завершена. Сбросим все пометки страниц на 0. Состояние кэша будет $\{3^0, 2^0, 4^0\}$.

Далее начнется третья фаза.

8. $x_8 = 1$. Значения 1 нет в кэше. Выберем случайное значение из кэша среди значений, помеченных 0. Допустим, это значение 4. Заменим его. Состояние кэша после восьмого шага работы алгоритма $\{3^0, 2^0, 1^1\}$.

9. $x_9 = 5$. Значения 5 нет в кэше. Выберем случайное значение из кэша среди значений, помеченных 0. Допустим, это значение 3. Заменим его. Состояние кэша после восьмого шага работы алгоритма $\{5^1, 2^0, 1^1\}$.

Входная последовательность завершилась, алгоритм завершает свою работу. Стоимость решения – 4.

Проделайте небольшое упражнение – посчитайте стоимость оптимального решения (3).

Вот такой несложный, в общем-то, вероятностный алгоритм решения задачи кэширования. Попробуем оценить качество этого алгоритма в терминах конкурентного соотношения.

Нам понадобится знание о так называемых «гармонических числах». k-ым гармоническим числом будем называть сумму $\sum_{i=1}^{k} \frac{1}{i}$ и обозначать H_k. Пиблизительное значение $H_k \sim \ln k$.

Теперь мы можем сформулировать теорему о коэффициенте оптимальности алгоритма MARK.

Теорема 1. $cr_{MARK} \le 2 H_k$.

Хороший результат, не правда ли? Использование вероятности позволило нам снова существенно улучшить качество решения.

Чуть позже мы приведем доказательство этой теоремы.

Рассмотрим генератор случайных чисел, порождающий числа от 1 до d – значения некоторой случайной величины X. Рассмотрим множество элементарных событий $\Omega = \{1, 2, ..., d\}$. Пусть значение i порождается с вероятностью p_i. Тогда матожиданием или средним значением случайной величины будем называть

$$\mathrm{E}[X] = \sum_{i=1}^{d} i \cdot \text{вероятность}(X = i) = \sum_{i=1}^{d} i * p_i .$$

Определим модель вероятностного онлайн алгоритма. Разрешим нашему онлайн алгоритму R пользоваться генератором случайных чисел. Можем формально определить это так: при вычислении очередного y_i алгоритм может использовать значения на дополнительно ленте ϕ, содержащей случайные числа. То есть $y_i = f_i(\phi, x_1, x_2, ..., x_i)$. При таком определении алгоритма на одной и той же входной последовательности разные запуски нашего алгоритма могут дать различный результат. Множество возможных результатов будет составлять множество элементарных исходов. Будем обозначать $cost(R^{\phi}(I))$ случайную величину, определяющую стоимость решения алгоритма R на входной последовательности I.

Через $\mathrm{E}[cost(R^{\phi}(I))]$ определим среднее значение (матожидание) стоимости решения алгоритма R на I.

Определение 4. Алгоритм R будем называть *c-конкурентным* в ожидании, если существует неотрицательная константа α такая, что для любого $I \in \mathcal{I}$ справедливо соотношение:

$$\mathrm{E}[cost(R^{\phi}(I))] \le c \cdot cost(Opt(I)) + \alpha .$$

Вернемся к доказательству теоремы.

Теорема 1. $cr_{MARK} \le 2 H_k$.

Доказательство: Будем анализировать поведение нашего алгоритма в i-ой фазе.

Страницы, запрошенные в фазе i, которые не запрашивались в предыдущей фазе, будем называть *новыми*.

Соответственно, страницы, запрошенные в фазе i, которые запрашивались в предыдущей фазе, будем называть *старыми*.

Обозначим m_i – число новых страниц на фазе i.

Заметим, что в начале фазы i кэш содержит те страницы памяти, которые были запрошены в предыдущей фазе. Значит, при запросе на новую страницу любой алгоритм будет производить замену. Он заменит одну из старых страниц. Далее при поступлении запроса на эту старую замененную страницу, алгоритму снова придется производить замену.

Давайте попробуем оценить, сколько раз наш алгоритм может ошибиться в фазе i при запросе к старым станицам.

Допустим, что в фазе i уже были запрошены $j-1$ различных старых и l различных новых страниц памяти. До конца текущей фазы будут запрошены еще $k-(l+j-1)$ различных страниц. Среди этих запросов могут быть запросы на старые страницы, которые были заменены при запросе новых. Таких запросов не больше l (так как мы заменили l страниц в кэше). Алгоритму придется делать замены при запросах этих старых страниц (при запросах других страниц замен не будет). Значит, вероятность ошибки при запросе старой страницы,которой нет в кэше в данный момент, будет

$$\frac{l}{k-(j-1+l)} .$$

Вспомним, что $l \le m_i$:

$$\frac{l}{k-(j-1+l)} \le \frac{m_i}{k-(j-1+l)} .$$

Общее количество замен в фазе не превосходит

$$m_i + \sum_{j=1}^{k-m_i} \frac{m_i}{k-m_i-j+1} =$$

$$= m_i \cdot (1 + \sum_{j=1}^{k-m_i} \frac{1}{k-m_i-j+1}) \le m_i \cdot (2 \cdot H_k)$$

при $k \ge 1$.

Стоимость решения

$$cost_{MARK} \le \sum_i m_i \cdot (2 \cdot H_k)$$

даже в худшем случае, а значит, и в среднем.

Рассмотрим оптимальный алгоритм. Рассмотрим ситуацию между фазами $i-1$ и i. В каждой фазе оптимальный алгоритм бу-

дет ошибаться не менее чем m_i раз, так как у него в кэше будут только старые страницы:

$$cost_{OPT} \geq \sum_i m_i .$$

Из предыдущих оценок следует

$$cr_{MARK} = \frac{cost_{MARK}}{cost_{OPT}} \leq 2H_k .$$

Литература

1. *Hromkovic J.* Algorithmic adventures. From Knowledge to Magic. Springer, 2009.

2. *Fiat A., Karp R.M., McGeoch L.A., Sleator D.D., Young N.E.* Competitive paging algorithms // Journal of Algorithms 12. P. 685–699.

Юрай Громкович,
Professor of Computer Science, Swiss Federal Institute of Technology, Zürich,

Аблаев Фарид Мансурович,
доктор физико-математических наук, профессор, заведующий кафедрой теоретической киберенетики Казанского федерального университета.