Самоорганизующиеся структуры данных

Евтушевская-Коновалова Иоанна

June 2025

1

Мы рассматриваем самоорганизующиеся структуры данных, точнее, мы будем рассматривать самоорганизующиеся списки, и их применения для решения некоторых задач. Итак, у нас есть список из элементов, в котором каждый элемент содержит указатель на следующий элемент. Таким образом, чтоб найти некоторый элемент, необходимо сделать линейный проход по списку и найти элемент. Множество элементов списка обозначим S. В процессе алгоритм в зависимости от полученных данных список может изменяться: можно выбрать любой элемент и переставить его на произвольную позицию в списке.

2

Будем решать такую задачу, известную как list update problem: Есть последовательность запросов следующего вида:
Access(x). Доступ к элементу x из S.(В таком случае, если элемент находится на позиции с номером k, необходимо пройти по списку по всем элементам, предшествующим x, то есть затраты на обработку такого запроса - k)
Insert(x). Вставка элемента x в S.(В таком случае, если элемент находится на позиции с номером k, необходимо пройти по списку и проверить, что элемента нет в списке, а затем вставить элемент, то есть затраты на обработку такого запроса - n, где n - длина списка)
Delete(x). Удаление элемента x из S.(в таком случае, если элемент находится на позиции с номером k, необходимо пройти по списку по всем элементам, предшествующим x, и затем удалить x, то есть затраты на обработку такого запроса - k)
Не умаляя общности, можно считать, что все запросы - это запросы типа Access. Мы будем оценивать, насколько хорош наш онлайн-алгоритм, сравнивая его время работы с потенциальным временем работы онлайн-алгоритм. Будем считать, что существует некоторый оптимальный онлайн-алгоритм OPT.

Обозначим время работы некоторого алгоритма A на последовательности запросов σ как $C_A(\sigma)$, время, затраченное на обработку запроса в момент t обозначим $C_A(t)$. Считаем, что переставлять элементы в списке мы можем "бесплатно без увеличения затрат на время работы. Определение: алгоритм называется c-конкурентным, если верна оценка:
$C_A(\sigma) < c \cdot C_{OPT} + a$

3 Детерминированные алгоритмы

Рассмотрим для начала детерминированные алгоритмы.
Move-to-front: алгоритм на каждом шагу перемещает запрошенный элемент в начало списка.
Transpose: алгоритм на каждом шагу заменяет запрошенный элемент на непосредственно предшествующий элемент в списке.
Frequency-count: алгоритм ведет подсчет частоты для каждого элемента в списке. Всякий раз, когда запрашивается элемент, увеличиваем его количество на 1. На каждом шагу переставляем список так, чтобы

элементы всегда появлялись в порядке возрастания частоты их появления.
Оказывается, что алгоритмы Frequency-count и Transpose не являются c-конкурентными ни для какой константы c. Далее докажем, что алгоритм Move-to-front является 2-конкурентным.

4 Лемма 1

Алгоритм Move-to-front является 2-конкурентным.

5 Доказательство

Рассмотрим последовательность запросов $\sigma = \sigma(1)\sigma(2)\ldots$ На каждом шаге будем получать и сравнивать два списка: полученный в результате работы OPT, и в результате работы Move-to-front. Будем называть инверсией упорядоченную пару элементов , y, такую, что что после обслуживания запроса в момент времени t в этих списках они расположены в разном порядке, например, в в списке, полученном в результате работы OPT, x идёт впереди y, а в списке, полученном в результате работы Move-to-front, наоборот.
Введем функцию потенциала $\Phi(t)$, значение которой будет равна количеству инверсий после обслуживания запроса в момент времени t.
Сделаем некоторые наблюдения, позволяющие нам получить неравенства на Ф. Пусть в момент времени t запрошен элемент . Пусть k - число элементов, стоящих впереди него в обоих наших списках, а l - число элементов, предшествующих в списке, соответствующем Move-to-front, но находящихся после в списке, соответствующему OPT.
Тогда ясно, что $C_{MTF} = k + l + 1, C_{OPT} > k + 1$
Докажем, что для всех t выполнено неравенство ниже. Ясно, что суммируя неравенство по всем t, получаем искомую оценку.
$C_{MTF}(t) + \Phi(t) - \Phi(t-1) < 2C_{OPT}(t)$
Действительно, переставив по алгоритму MTF в начало списка, мы потеряем хотя бы l инверсий, и создадим не больше k, тогда: $\Phi(t) - \Phi(t-1) < k - l$
Значит, $C_{MTF} + \Phi(t) - \Phi(t-1) < C_{MTF} + k - l = 2k + 1 < 2C_{OPT}(t) - 1$
Таким образом, неравенство доказано, а значит, получена искомая оценка.

6 Вероятностный алгоритм

Теперь рассмотрим вероятностный алгоритм, зависящий от параметра p. Оказывается, он тоже является 2-конкурентным, но оценка будет несколько сложнее.

Алгоритм Timestamp(p):
При заданной последовательности запросов σ каждый запрос $\sigma(t)$ обрабатывается следующим образом.
Рассмотрим, $\sigma(t)$ - запрос к элементу .
С вероятностью p выполняется шаг (a).

(a). перемещается в начало списка.
С вероятностью 1-p выполняется шаг (b)

(b). Для каждого элемента и момента времени t будем хранить $v_t(x)$, определенное следующим образом: выделим промежуток времени от последнего запроса на до t. Рассмотрим все элементы, которые либо на были запрошены в этот промежуток времени, либо запрошены лишь однажды и обслужены шагом (b). $v_t(x)$ будет указывать на ближайший к началу списку из этих элементов, если такие есть, или на , если таких элементов не нашлось. Таким образом, если в момент t запрошен впервые, алгоритм не меняет его позицию, иначе вставляем перед $v_t(x)$.

Понятно, что при $p = 1$ это будет просто Move-to-front.
Определение c-конкурентности в данном случае будет следующим. Вероятностный алгоритм называется c-конкурентным, если выполнено неравенство, оценивающе матожидание времени работы:
$E[C_{TS}(\sigma)] < c \cdot C_{OPT}(\sigma)$

7 Теорема

Алгоритм Timestapm(p) является c-конкурентным при $c = max(2 - p, 1 + p(2 - p))$
Сначала сформулируем несколько лемм и кратко опишем их доказательство.

8 Лемма 2

Пусть x и y - два различных элемента. Предположим, что запрашивается в момент времени t_0 и в момент времени t, $t_0 < t$, и что y не запрашивается в течение интервала $[t_0; t]$. Тогда сразу после обслуживания σ(t), предшествует y в списке Timestapm(p), и это не меняется до следующего запроса к y.

8.1 Доказательство

В случае, если запрос в момент t обслуживается шагом (a), лемма, очевидно, верна. Пусть был использован шаг (b), тогда необходимо вставить перед всеми элементами, которые не запрашиваются на промежутке или запрашиваются на промежутке ровно один раз и обслуживаются шагом (b) (это ясно из определения $v_t(x)$. Очевидно, y удовлетворяет условиям выше, значит, после вставки перед $v_t(x)$ будет предшествовать y. Понятно также, что y не может переместиться вперёд, пока не будет обслужены запрос на y. Таким образом, всё доказано.

9 Лемма 3

Пусть t некоторый момент времени из промежутка [1, m], $= \sigma(t)$ запрашивается ровно один раз в промежутке [1, t-1], выполнены два условия ниже. Пусть $t^{'} < t$ - предыдущий запрос к .
a) Если y был запрошен хотя бы дважды в промежутке $[t^{'}, t - 1]$, то y предществует $v_t(x)$ в списке в момент t.
b) Если y был запрошен ровно один раз в промежутке $[t^{'}, t - 1]$, и обслуживался шагом a), то y предществует $v_t(x)$ в списке в момент t.

9.1 Доказательство

Пусть t_0 - самый первый момент, когда утверждение a) неверно, $t_0^{'}$ последний момент до t, когда был запрошен, $z = v_{t_0}(x)$. Пусть y не предшествует z. t_y момент последнего запроса к y в промежутке $[t_0^{'}$, t_0-1]. Покажем, что после обработки $\sigma(t_y)$ y предшествует x. Если был использован шаг a), это очевидно, предположим, что использовался шаг b). Из определения $v_t(x)$ ясно, что z запрашивается лишь однажды на промежутке $[t_0^{'}, t_0 - 1]$, и использовался шаг b). Тогда z не может предшествовать $v_{t_y}(y)$. Тогда y был вставлен куда-то перед z, значит, y предшествует z после обработки $\sigma(t_y)$. По предположению, z предшествует y в момент t_0, элемент z должен быть запрошен в некоторый момент t_z в промежутке $[t_y+1$, t_0-1] и вставлен перед y. Значит, y не предшествует $v_{t_z}(z)$ в момент t_z, поскольку запрос в этот момент обррабатывался шагом b). Заметим, что в момент t_z, y был запрошен хотя бы дважды после последнего запроса к z. Это означает, что мы взяли не первый момент, когда условие a) не было выполнено. Противоречие.
Выполнение условия b) проверяется аналогично.

10 Доказательство теоремы

Перейдем к оценке матожидания, немного преобразуем его.
Введем обозначение $C_{TS}(t,x)$. $C_{TS} = 1$, если в момент t x предшествует элементу, запрошенному в $\sigma(t)$.
Тогда можно переписать матожидание так:
$E[C_{TS}(t,x)] = E[\sum_{i=1}^{m} \sum_{x \in S} C_{TS}(t,x)] = E[\sum_{x \in S} \sum_{y \in S} \sum_{t \in [1,m], \sigma(t)=y}]$
Аналогично можно переписать и C_{OPT}. Теперь можно оценивать отдельно сумму слагаемых, относящихся к конкретной паре элементов x, y . Заметим, что для оценки этих слагаемых можно упростить ситуацию, а именно, рассматривать последовательность запросов σ_{xy}, и применять к списку, состоящему только из этих двух элементов, поскольку нас интересует только взаимное расположение x и y. σ_{xy} разобьем на фазы P_j так: если $\sigma_{xy}(i-1) = \sigma_{xy}(i) \neq \sigma_{xy}(i+1)$, фаза заканчивается на запросе i.
Тогда могут быть фазы четырех типов, каждый из которых оценим отдельно.
1) $P_j = x^h, h > 1$
2) $P_j = xy^h, h > 1$
3) $P_j = (xy)^{h_1} x^{h_2}, h_1 > 0, h_2 > 1$
4) $P_j = (xy)^{h_1} y^{h_2}, h_1 > 0, h_2 > 1$
Используя леммы 2 и 3, можно получить следующие оценки:
Для P_j типа 1: $E[C_{TS}(P_j)] < (2-p)C_{OPT}(P_j)$
Для P_j типа 2: $E[C_{TS}(P_j)] < (1+p(2-p))C_{OPT}(P_j)$
Для P_j типа 3 и 4: $E[C_{TS}(P_j)] < max(2-p, 1+p(2-p))C_{OPT}(P_j)$
Просуммировав эти оценки по всем фазам, и по всем парам элементов x, y, получим искомую оценку, таким образом, теорема доказана.