Амортизационный анализ

Содержание

Основные определения

Определение:

Амортизационный анализ (англ. *amortized analysis*) — метод подсчёта времени, требуемого для выполнения последовательности операций над структурой данных. При этом время усредняется по всем выполняемым операциям, и анализируется средняя производительность операций в худшем случае.

Такой анализ чаще всего используется, чтобы показать, что даже если некоторые из операций последовательности являются дорогостоящими, то при усреднении по всем операциям средняя их стоимость будет небольшой за счёт низкой частоты встречаемости. Подчеркнём, что оценка, даваемая амортизационным анализом, не является вероятностной: это оценка среднего времени выполнения операций для худшего случая.

Определение:

Средняя амортизационная стоимость операций — величина a, находящаяся по формуле: $a = \frac{\sum\limits_{i=1}^{n} t_i}{n}$, где $t_1, t_2 \dots t_n$ — время выполнения операций $1, 2 \dots n$, совершённых над структурой данных.

Амортизационный анализ использует следующие методы:

1. Метод усреднения (метод группового анализа).
2. Метод потенциалов.
3. Метод предоплаты (метод бухгалтерского учёта).

Метод усреднения

В методе усреднения амортизационная стоимость операций определяется напрямую по формуле, указанной выше: суммарная стоимость всех операций алгоритма делится на их количество.

Примеры

Стек с multipop

Рассмотрим стек с операцией $\mathrm{multipop}(a)$ — извлечение из стека a элементов. В худшем случае она работает за $O(n)$ времени, если удаляются все элементы массива. Однако прежде чем удалить элемент, его нужно добавить в стек. Итак, если в стеке было не более n элементов, то в худшем случае с каждым из них могли быть произведены две операции — добавление в стек и извлечение из него. Например, если было n операций push — добавление в стек, стоимость каждой $O(1)$, и одна операция $\mathrm{multipop}(n)$, то суммарное время всех операций — $O(n)$, всего операций $n+1$, а значит, амортизационная стоимость операции — $O(1)$.

Распишем приведённые рассуждения более формально. Пусть n — количество операций, m — количество элементов, задействованных в этих операциях. Очевидно, $m \leqslant n$ Тогда:

$a = \frac{\sum\limits_{i=1}^{n} t_i}{n} = \frac{\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{m} t_{ij}}{n} = \frac{\sum\limits_{j=1}^{m}\sum\limits_{i=1}^{n} t_{ij}}{n}$, где t_{ij} — стоимость i-ой операции над j-ым элементом. Величина $\sum\limits_{i=1}^{n} t_{ij}$ не превосходит 2, так как над элементом можно совершить только две операции, стоимость которых равна 1 — добавление и удаление. Тогда:

$a \leqslant \frac{2m}{n} \leqslant 2$, так как $m \leqslant n$.

Таким образом, средняя амортизационная стоимость операций $a = O(1)$.

Двоичный счётчик

Рассмотрим также двоичный инкрементальный счётчик (единственная возможная операция — увеличить значение на единицу). Пусть результат увеличения счётчика — n, тогда в худшем случае необходимо изменить значения $1 + \lfloor \log n \rfloor$ бит, и стоимость n операций составит $O(n \log n)$. Теперь воспользуемся для анализа методом усреднения. Каждый следующий бит изменяет своё значение в n, $\frac{n}{2}$, $\frac{n}{4}$ … операциях. Общая стоимость:

$$\sum_{i=0}^{\lfloor \log n \rfloor} \frac{n}{2^i} < 2n = O(n)$$

В итоге амортизационная стоимость одной операции — $O(1)$.

Метод потенциалов

Теорема (О методе потенциалов):

Введём для каждого состояния структуры данных величину Φ — потенциал. Изначально потенциал равен Φ_0, а после выполнения i-й операции — Φ_i. Стоимость i-й операции обозначим $a_i = t_i + \Phi_i - \Phi_{i-1}$. Пусть n — количество операций, m — размер структуры данных. Тогда средняя амортизационная стоимость операций $a = O(f(n, m))$, если выполнены два условия:

1. Для любого $i : a_i = O(f(n, m))$
2. Для любого $i : \Phi_i = O(n \cdot f(n, m))$

Доказательство:

$\triangleright$

$$a = \frac{\sum\limits_{i=1}^{n} t_i}{n} = \frac{\sum\limits_{i=1}^{n} a_i + \sum\limits_{i=0}^{n-1} \Phi_i - \sum\limits_{i=1}^{n} \Phi_i}{n} = \frac{n \cdot O(f(n,m)) + \Phi_0 - \Phi_n}{n} = O(f(n, m))$$

$\triangleleft$

Примеры

Стек с multipop

В качестве примера вновь рассмотрим стек с операцией $\text{multipop}(a)$. Пусть потенциал — это количество элементов в стеке. Тогда:

1. Амортизационная стоимость операций:
   - $a_{push} = 1 + 1 = 2$, так как время выполнения операции $\text{push} - 1$, и изменение потенциала — тоже 1.
   - $a_{pop} = 1 - 1 = 0$, так как время выполнения операции $\text{pop} - 1$, а изменение потенциала — -1.
   - $a_{multipop} = k - k = 0$, так как время выполнения операции $\text{multipop}(k) - k$, а изменение потенциала — $-k$.
2. Для любого $i : \Phi_i = O(n)$, так как элементов в стеке не может быть больше n

Таким образом, $f(n, m) = 1$, а значит, средняя амортизационная стоимость операций $a = O(1)$.

Динамические хэш-таблицы

Рассмотрим хэш-таблицы, использующие цепочки в виде списков для разрешения коллизий. Для того, чтобы поиск элемента в цепочке не начал слишком сильно влиять на производительность, введём величину α — фактор загруженности (load factor) нашей таблицы. Пусть в нашей таблице размером m хранится n элементов, тогда $\alpha = \frac{n}{m}$. Введём значение α_{max}, при превышении которого таблица будет пересоздана с увеличением размера в 2 раза, и все элементы будут перераспределены по-новому (rehashing). Из-за этого сложность операции add в худшем случае составит $O(n)$.

Для анализа структуры введём следующую потенциальную функцию: $\Phi = 2n - \alpha_{max} m$

Рассмотрим время работы каждой из операций $\text{add}, \ \text{remove}, \ \text{find}$:

1. add: n увеличивается на единицу. Следовательно, возникают два случая:
   - $\alpha < \alpha_{max} : \alpha_i = 1 + 2 \cdot (n + 1) - \alpha_{max} m - (2n - \alpha_{max} m) = 3$, так как время выполнения операции $\text{add} - 1$
   - $\alpha = \alpha_{max} : a_i = 1 + \alpha_{max} m + 2 \cdot (\alpha_{max} m + 1) - 2\alpha_{max} m - 2\alpha_{max} m + \alpha_{max} m = 3$, так как таблица увеличивается в размере, поэтому время работы операции add составит $1 + \alpha_{max} m$, потому что сейчас в таблице $n = \alpha_{max} m$ элементов.
2. find:
   - Если элементы распределяются по таблице достаточно равномерно, то время поиска элемента в

списке — $O(1)$, потенциал не изменяется, следовательно и реальная, и амортизированная сложности — 1.
    - В случае, если все элементы оказываются размещены в одном списке, время поиска элемента достигает $O(n)$. Это время может быть улучшено до $O(\log n)$, если вместо списков использовать сбалансированные деревья поиска (эта модификация была добавлена в Java 8 для стандартной коллекции HashSet).
3. remove: n уменьшается на единицу. Тогда амортизационное время работы с учётом изменения потенциала составит:
    - $a_{remove} = 1 + 2 \cdot (n - 1) - \alpha_{max}m - (2n - \alpha_{max}m) = -1$

Так как $\Phi_i = 2n - \alpha_{max}m = O(n)$, поэтому если $f(n, m) = 1$, то средняя амортизационная стоимость модифицирующих операций составит $a = O(1)$.

Метод предоплаты

Представим, что использование определённого количества времени равносильно использованию определённого количества монет (плата за выполнение каждой операции). В методе предоплаты каждому типу операций присваивается своя учётная стоимость. Эта стоимость может быть больше фактической, в таком случае лишние монеты используются как резерв для выполнения других операций в будущем, а может быть меньше, тогда гарантируется, что текущего накопленного резерва достаточно для выполнения операции. Для доказательства оценки средней амортизационной стоимости $O(f(n, m))$ нужно построить учётные стоимости так, что для каждой операции она будет составлять $O(f(n, m))$. Тогда для последовательности из n операций суммарно будет затрачено $n \cdot O(f(n, m))$ монет, следовательно, средняя амортизационная стоимость операций будет $a = \frac{\sum_{i=1}^{n} t_i}{n} = \frac{n \cdot O(f(n,m))}{n} = O(f(n, m))$.

Примеры

Стек с multipop

При выполнении операции push будем использовать две монеты — одну для самой операции, а вторую — в качестве резерва. Тогда для операций pop и multipop учётную стоимость можно принять равной нулю и использовать для удаления элемента монету, оставшуюся после операции push.

Таким образом, для каждой операции требуется $O(1)$ монет, значит, средняя амортизационная стоимость операций $a = O(1)$.

Источники информации

- Wikipedia — Amortized analysis
- Томас Кормен. Алгоритмы. Построение и анализ. - Санкт-Петербург, 2005. стр. 483-491.
