平摊分析

算法设计与分析小班课

2022 年 4 月

注意: 以下内容主要是给课本作一些注解并补充习题, 可能不能准确地代表课程内容和考核的方向, 也不能替代阅读课本、听课、完成作业题、做往年题等活动; 其中的内容没有经过课程主管老师的审核, 也可能存在错误. 不过出现的所有错误都由作者本人负责.

1 三种方法

平摊分析考虑的是一段操作在最坏情况下的时间复杂度, 它在这段操作中设法给每个操作赋予平摊代价, 使得这些代价都比较小. 三种方法中的第一种就是直接求和算一下平均值. 第二种和第三种是同一个思想的一体两面——多退少补, 比如消耗时间高的操作出现之前一定有许多较快的操作, 所以可以把这些时间消耗分配到其他操作上. 在记账法中, 我们直接对每个操作赋予平摊代价; 而在势能法中, 平摊代价等于势能变化加上实际代价, 势能下降用来补偿较慢的操作. 前者是局部的, 后者是全局的, 但总体是类似的. 使用不同的分析方法可能导出不同的平摊代价, 即分析结果不唯一.

以下是一些简单的问题, 一般来说使用任何一种平摊分析方法都是可以的.

1.1 聚集法

问题 1 设对某个数据结构的一段操作序列中, 第 i 个操作的情况为: 若 $i = 2^k$, 操作代价为 i, 否则操作代价为 1. 用聚集法给出平摊分析.

解 直接计算:

$$\sum_{i=1}^{n} c_i \le (2^1 + 2^2 + \cdots + 2^{\lceil \log_2 n \rceil}) + n = O(n),$$

所以平摊复杂度为 $O(1)$. □

问题 2 假设在基础的栈的操作中增加一个操作 MultiPush, 每次可以推入任意 k 个元素, 请问平摊复杂度是否还是 $O(1)$?

解 考察之前的证明过程可以发现此情况下已经无法如此论证. 事实上, 只要每次推入 k 个再弹出 k 个元素, 则平摊复杂度为 $\Omega(k)$. □

1.2 记账法

问题 3 假设一个栈的最大元素个数为 k. 在 Push, Pop, MultiPop 的基础上, 我们每进行 k 次操作后都对栈中全部元素复制一次. 请用记账法进行平摊分析.

解 有两个高代价的操作: 多次弹出和复制. 这两个操作的代价都取决于栈中有多少元素, 因此要摊到推入中去; 不过注意到一个元素或者要被复制, 或者要被 MultiPop, 所以仍然可以令 Push 的代价为 2, 多出来的部分留给两个高代价的操作. 因为复制最多 k 个元素而且间隔 k 次操作, 所以记账法中无欠款, 所以平摊复杂度为 $O(1)$. □

问题 4 设对某个数据结构的一段操作序列中, 第 i 个操作的情况为: 若 $i = 2^k$, 操作代价为 i, 否则操作代价为 1. 用记账法给出平摊分析.

解 我们给每个操作赋予 3 的平摊代价. 归纳地证明在前 2^i 个操作结束后, 账面非负. 归纳步: 由于 2^i 之后, 2^{i+1} 之前的 $2^i - 1$ 个操作累计多出 $2(2^i - 1)$, 这在 2^{i+1} 次操作后, 收入 3, 支出 2^{i+1}, 得到净值为 1, 因此平摊复杂度为 $O(1)$. □

问题 5 假设我们希望不仅能使一个计数器增值, 也能使之复位至零. 请说明如何将计数器实现为一个位数组, 使得对一个初始为零的计数器, 任一包含 n 个 Increment 和 Reset 操作的序列的时间为 $O(n)$. 设计这样的计数器并用记账法分析.

解 为了降低 Reset 的开销, 需要增加一个 $flag$ 变量指明现在的最高位 (初始化为 0), Reset 时从那里开始置 0 即可.

算法 1 Increment

```
1: i ← 0
2: while a[i] == 1 and i < ℓ(a) do
3:     a[i] ←
4:     i ← i + 1
5:     flag ← max(i, flag)
6: end while
7: if i < ℓ(a) then
8:     a[i] ← 1
9:     flag ← max(i, flag)
10: end if
```

算法 2 Reset

```
1: for i ← 0 to flag do
2:     a[i] ← 0
3: end for
4: flag ← 0
```

每次将 0 设为 1 都赋予 3 的平摊代价, 其中额外的部分备以抵消被置 0 的代价 (进位和 Reset 两部分). 每次 Increment 的代价最多为 3. 另外每次调整 $flag$ 都赋予 2 的平摊代价, 支付修改代价并备以抵消 Reset 时重设它的代价, 不论是 Increment 还是 Reset 每次也在这上面至多消耗 2. 因此 n 步总开销 $O(n)$. □

1.3 势能法 势能法可较好地给出数据结构中不同操作的分析——特别是它们的平摊代价并非常数的时候, 这样可能要涉及到对数据结构状态的分析. 选定势函数可以用凑

的方法. 大体上说, 势函数需要保证在数据结构即将发生大开销操作时充分大 (操作前后的势降恰好能补充开销为宜), 同时恒非负, 主要是确保那些临界状态的值可行.

问题 6 设对某个数据结构的一段操作序列中, 第 i 个操作的情况为: 若 $i = 2^k$, 操作代价为 i, 否则操作代价为 1. 用势能法给出平摊分析.

解 这使用势能法没有前面的方法直接, 我们可以从记账法中导出下面的势能函数: 如果 $i = 2^k$, 则令 $\Phi(D_i) = k + 3$, 否则取 $\Phi(D_i) = \Phi(D_{2^{\lfloor \log_2 i \rfloor}}) + 2(i - 2^{\lfloor \log_2 i \rfloor})$. 则

$$\Phi(D_i) - \Phi(D_{i-1}) = \begin{cases} k + 3 - (k - 1 + 3) - 2(2^k - 1 - 2^{k-1}) = 3 - 2^k & (i = 2^k), \\ 2 \quad (\text{否则}). \end{cases}$$

因此两类操作的均摊复杂度均为 $3 = O(1)$. □

问题 7 考虑一个使用二叉树结构实现的最小堆, 它支持 Insert 和 ExtractMin (取出并删除根结点元素) 两个操作, 二者的最坏时间复杂度均为 $O(\log n)$. 给出一个使用势能法的平摊分析, 使得 Insert 操作的平摊代价为 $O(\log n)$ (n 为当前堆中元素个数), 而 ExtractMin 的平摊代价为 $O(1)$.

解 考虑非负势能函数 $Cn \log n$, 其中 C 为 ExtractMin 最坏情况下复杂度之前的常数, n 为当前堆中元素个数, 那么插入的平摊代价为 $O(\log n) + (n + 1)\log(n + 1) - n \log n = O(\log n) + n \log(1 + 1/n) = O(\log n)$; 而取最小时堆变小, 所以平摊代价为 $C \log n - C \log(n + 1) - n \log(1 + 1/n) = (n - C) \log(1 + 1/n) = O(1)$. □

问题 8 说明如何用两个普通的栈来实现一个队列, 使得 Enqueue 和 Dequeue 操作的平摊代价都为 $O(1)$.

解 将两个栈编号为 1 和 2. 进队操作: 直接将元素推入 1 中. 出队操作: 如果 2 不是空的, 就从 2 中弹出一个元素, 否则把整个 1 中的元素逐个弹出再推入 2 中, 再从 2 中弹出一个元素.

定义势函数为栈 1 中的元素个数的两倍. 则它是一个非负函数.

▷ **进队** 平摊代价为 $1 + 2 = 3$.

▷ **出队** 如果 2 是空的, 则平摊代价为 1. 设栈 1 中的元素个数为 n, 实际代价为 $2n + 1$, 势函数变化为 $-2n$, 所以平摊代价为 1. □

问题 9 请设计数据结构来维护一组 n 个不同整数的集合 S, 要求支持以下两种操作:

(i) Insert(x, S): 将 x 加入 S;

(ii) RemoveBottomHalf(S): 从 S 中移除最小的 $\left\lceil \frac{n}{2} \right\rceil$ 个整数.

描述你的算法并给出最坏情况下两个操作的时间复杂度. 进一步采用势能法进行平摊分析, 得到两种操作的平摊时间: 请选择合适的分析策略, 使得 (i) 的平摊代价为 $O(1)$, (ii) 的平摊代价为零.

解 使用动态表的结构: 将元素存入数组中, 如果插入时不够存了就把容量扩展两倍. 在删除最小的 $\left\lceil \frac{n}{2} \right\rceil$ 个数时, 首先找出中位数 m (采用 $O(n)$ 的办法), 然后把旧表中所有大于 m 的数复制到新的和原来表容量相同的表中. 最坏插入操作代价 $O(n)$, 删除操作代价也为 $O(n)$. 以下设删除时操作需要时间前的常数为 k.

定义势函数为 $\Phi(S) = 3kn - c$, 其中 n 为表中数字数量, c 为表容量, 则由于表至多空闲一半, $\Phi(S) \geq \Phi(\varnothing) = 0$.

▷ **插入** 如果不扩张表则平摊代价为 $1 + (3k(n+1) - c) - (3kn - c) = 4 + k$. 如果扩张了, 则当前 $c = n$, 故平摊代价为 $n + 1 + (3k(n+1) - 2c) - (3kn - c) = n + 4 - n + k = 4 + k$.

▷ **删除** 找中位数和复制整张表至多需要代价 kn, 所以平摊代价为 $1.5kn + (1.5kn - c) - (3kn - c) = 0$. □

除了在数据结构中的应用之外, 在线算法中也常常用到平摊分析的技巧. 虽然这个话题是后半学期的内容, 为了演示势能法的更多应用, 我们现在就展示一个例子 (虽然这个例子颇有一种拿 "势能" 当幌子的感觉, 但思想是很像的).

考虑这样一个问题: 有 n 个自称专家的人将在接下来的 $1, 2, \ldots, T$ 天内对股市的涨跌 $y_t \in \{0, 1\}$ 进行预测, 他们的预测记为 $\hat{y}_{it}$; 李雷希望参考他们的预测来给出自己的预测 $\hat{y}_t$. 记李雷和每个专家的错误次数分别为

$$L = \sum_{t=1}^{T} \mathbf{1}_{y_t \neq \hat{y}_t}, \quad L_i = \sum_{t=1}^{T} \mathbf{1}_{y_t \neq \hat{y}_{it}}.$$

李雷应该如何决策使得 L 和最好的专家的错误次数 $L^* = \min_i L_i$ 尽可能接近?

以下展示一种叫作 weighted majority vote 的算法. 思想为: 为每个专家维护权 w_i, 选择总权和大的预测作为李雷的预测; 若预测错了, 就降低那些弄错了的专家的权重.

算法 3 WeightedMajorityVote

输入: $\beta \in (0, 1)$

$w_{1i} \leftarrow 1\ (\forall i)$
for $t \leftarrow 1$ to T **do**
  $y_t \leftarrow \begin{cases} 0 & \left(\text{如果 } \sum_{\hat{y}_{it}=0} w_{it} > \sum_{\hat{y}_{it}=1} w_{it}\right), \\ 1 & (\text{否则}). \end{cases}$
  if $\hat{y}_t = y_t$ **then**
    $w_{i,t+1} \leftarrow w_{it}\ (\forall i)$
  else
    $w_{i,t+1} \leftarrow \begin{cases} \beta w_{it} & (\hat{y}_{it} \neq y_t), \\ w_{it} & (\text{否则}). \end{cases}$
  end if
end for

定理 10 ([LW94, 推论 2.1]) *若用以上算法, 则*

$$L \le \frac{L^* \log 1/\beta + \log n}{\log(2/(1+\beta))}.$$

注意 $\frac{\log 1/\beta}{\log(2/(1+\beta))} \ge 2$ 而其在 $(1,2)$ 处有可去间断, 所以此定理大致意思是说 $L \approx$ 常数 $\times$ L^* + 常数. 也就是说错误率在 $T \to \infty$ 的时候李雷和最好的专家不会差超过一倍.

证明 定义势函数 $W_t = \sum_{i=1}^n w_{it}$. 当 t 时刻预测错误时, 至少一半的专家都错了, 所以此时

$$W_{t+1} \le \frac{1+\beta}{2} W_t, \quad \text{这导出} \quad W_{T+1} \le \left(\frac{1+\beta}{2}\right)^L n.$$

另一方面, $W_{T+1} \ge w_{i,T+1} \ge \beta^{L_i}$ $(\forall i)$, 结合两个不等式就算出了定理中的界. □

2 平摊分析的补充例题

问题 11 在快速 Fourier 变换的实现中需要对一个线性数组作如下变换: 把下标 i 的元素和下标 j 的元素交换, 其中 i, j 的 k 位二进制表示刚好相反 (这时我们记 $j = \mathrm{rev}_k(i)$). 例如 1 的四位二进制表示是 0001_2, 对应于 1000_2.

我们可以如下方式获得 $0, 1, 2, \ldots$ 的对应二进制反序: 从 $a = 0$ 开始, 我们每次计算 $\mathrm{rev}_k(\mathrm{rev}_k(a) + 1)$ 并迭代. 例如 $k = 4$ 时会得到

$$0000_2, 1000_2, 0100_2, 1100_2, 0010_2, 1010_2, \ldots$$

就是 $0, 1, 2, \ldots$ 的对应二进制反序. 更本质地说, 我们把二进制表示的最左侧视为最低位, 每次加一. 模拟计算机中加法器的实现, 我们可以给出如下算法:

算法 4 BitReversedIncrement

输入: a

```
mask ← 10⋯00
while mask AND a ≠ 0 do
    a ← a XOR mask
    m 右移一位
end while
a ← a OR mask
```

假设计算机可以在 1 的时间内计算与、或、非和移位操作, 请对以上生成 $0, 1, 2, \ldots, n$ 的对应二进制反序的过程作平摊分析.

解 这就是课上讲过的 INCREMENT 问题, 只不过现在把二进制倒过来了. 因此平摊代价为 $O(1)$. □

问题 12 考虑以下同时支持查找和插入的动态数据结构. 对于 n 个元素的表, 我们维护 $\lceil \log_2(n+1) \rceil$ 个排好序的表. 这些表的大小由 n 的二进制表示 $n_{k-1}, \ldots, n_0$ 决定, 第 i 个表的大小为 2^i; 如果 $n_i = 0$, 则它是空的, 否则是全满的.

(1) 表中如何进行搜索操作? 分析其最坏时间复杂度.

(2) 表中如何进行插入操作? 分析其最坏时间复杂度并进行平摊分析.

(3) 表中如何进行删除操作? 分析其最坏时间复杂度.

解 (1) 很容易, 我们直接逐个查表并二分即可, 时间复杂度为 $O(\log^2 n)$.

考虑 (2). 当新进入一个元素的时候, 相当于 $n \leftarrow n+1$, 那些参与二进制位翻转 (进位) 的表要进行合并操作. 比如 $n = 7 = 111_2$, 新元素独立成表后, 所有的表要归并成一个表; 又如 $n = 2 = 10_2$, 则新元素独立成表后无需再归并. 具体地说, 令 $list \leftarrow \{$新元素$\}$, 依次检查第 i 个表:

(i) 如果它是空的, 则就把 $list$ 放到这里;

(ii) 否则, 将它和 $list$ 归并后得到新的 $list$, 继续检查.

最坏情况下要将所有元素合并成一个表, 这需要的时间是 $O(n)$. 对于平摊分析, 我们注意到每插入 2^i 个元素之后, 第 i 个表才会发生一次代价 2^i 的归并, 利用聚集法, 总代价为

$$O\left(\sum_{i=0}^{\lceil \log_2(n+1) \rceil - 1} 2^i \times \frac{n}{2^i}\right) = O(n \log n),$$

所以平摊代价为 $O(\log n)$.

(3) 我们先在各个表中查找要删除的元素, 然后在最小的满的表中找一个元素替到该元素被删除的位置上. 接下来最小的满表少了一个元素, 将它按次序拆成更小的表即可. 这在最坏情况下需要 $O(n)$ 时间 (拆了最大的表). □

问题 13 二叉搜索树应当尽量平衡. 考虑一般的二叉搜索树, 我们用 $x.size$ 表示以 x 为根的子树中的结点个数. 另外用 $x.left$ 和 $x.right$ 表达 x 的两个孩子结点. 给定 $\alpha \in \left[\frac{1}{2}, 1\right)$, 称 x 是 **α-平衡**的, 如果 $x.left.size \le \alpha \cdot x.size, x.right.size \le \alpha \cdot x.size$. 若每个结点都是 α-平衡的, 那么就称整棵树是 α-平衡的.

(1) 设计一个算法重建以 x 为根的子树, 使得它是 $\frac{1}{2}$-平衡的 (即完美平衡的).

(2) 证明: 在 n 结点、α-平衡的二叉搜索树中进行搜索操作的代价是 $O(\log n)$.

设 $\alpha > \frac{1}{2}$. 我们对二叉搜索树进行正常的插入和删除操作, 不过一旦某个子树不是 α-平衡的, 那么我们将重建之, 使得其是 $\frac{1}{2}$-平衡的.

以下我们将用势能法对数据结构进行平摊分析. 设 x 是搜索树 T 中的结点, 定义不平衡度

$$\Delta(x) = |x.left.size - x.right.size|$$

以及势函数

$$\Phi(T) = c \sum_{x \in T : \Delta(x) \geq 2} \Delta(x),$$

其中 $c = c(\alpha)$ 是一个充分大的常数.

(3) 证明: $\Delta(T) \geq 0$, 其中 T 完美平衡时势函数为零.

(4) 假设重建 m 个结点的树的实际代价是 Km, 请选择一个合适的 c 来确保重建的平摊代价为 $O(1)$.

(5) 基于对前面问题的回答, 证明: 在 n 结点、α-平衡的二叉搜索树中进行插入和删除操作的平摊代价都是 $O(\log n)$.

解 (1) 是简单的, 我们直接遍历得到子树中排序的序列, 然后选定中位数作为根, 递归地重建树, 这总之需要线性的时间.

对 (2), 搜索需要的时间满足 $T(n) = T(\alpha n) + 1$, 所以 $T(n) = O(\log n)$. 因为搜索中每次都会进展一层, 所以搜索树的深度其实也是 $O(\log n)$.

(3) 是显然的 (对于完美平衡的树, $\Delta(x) \leq 1$). 考虑 (4), 我们取 $c = \frac{K}{2\alpha - 1}$, 那么重建以 x 为根的子树的平摊代价为

$$Km + 0 - \Phi(D_{i-1}) \leq Km - c\Delta(x) \leq Km - cm(2\alpha - 1) \leq 0.$$

在 (2)、(4) 的基础上, 插入和删除时, 首先要用 $O(\log n)$ 的时间找到元素, 随后进行树的重建. 由于最多需要重建插入或删除元素到根路径上的所有子树, 而深度最多 $O(\log n)$, 因此平摊的重建代价为 $O(\log n)$, 由此可见插入和删除操作的平摊代价都是 $O(\log n)$. □

参考文献

平摊分析这个词语来源于财务会计, 是指对除固定资产之外, 其他可以长期使用的经营性资产按照其使用年限每年分摊购置成本的会计处理办法. Tarjan 于 1985 年总结了前人的记账、势能等等分析方法, 将其称为 amortized analysis.

选入的习题基本上来源于 [Cor+09, 第 17 章]. 在线算法用势能法分析的例子是作者在王立威老师的机器学习课程上学到的.

[Cor+09] T. H. Cormen et al. *Introduction to Algorithms*. MIT press, 2009. ISBN: 978-0-26-203384-8.

[LW94] N. Littlestone and M. K. Warmuth. “The weighted majority algorithm”. 刊于: *Information and Computation* 108.2 (1994), pp. 212–261.

编写: WC

E-mail: wchang@pku.edu.cn