算法设计与分析

平摊分析

主要内容

- 平摊分析的概念
- 平摊分析的三种方法
  - 聚集分析
  - 记账法
  - 势能法
- 动态表及其上的平摊分析

平摊分析

- 在平摊分析中，执行一系列数据结构操作所需要的时间是通过执行的所有操作求平均而得出的。平摊分析可以用来证明在一系列操作中，通过对所有操作求平均之后，即使其中单一的操作具有较大的代价，平均代价还是很小的。
- 平摊分析不牵涉到概率, 平摊分析保证在最坏情况下，每个操作具有的平均性能。

平摊分析——聚集分析

栈S上的三种操作

- **PUSH(*S*,*x*)**：将*x*压入*S*
  - 运行时间**O(1)**
- **POP(*S*)**：弹出栈顶
  - 运行时间**O(1)**
- **MULTIPOP(*S*,*k*)**：弹出栈顶*k*个对象

```
MULTIPOP(S, k)
while not STACK-EMPTY(S) and k ≠ 0
    do POP(S)
        k → k - 1
```

  - 运行时间**O(min(*k*,*s*))**，***s***为栈中对象个数

n个栈操作的平摊时间

- 设有n个栈操作（**PUSH**、**POP**、**MULTIPOP**）的序列，作用于初始为空的栈S。
- 总的运行时间的界是什么？
  - 每个操作都可能是**MULTIPOP**
  - 每个**MULTIPOP** 的运行时间是$O(\min(k, s))=O(n)$
  - 总的运行时间的上界为$O(n^2)$
  - 这是一个紧的上界吗？

n个栈操作的平摊时间

- 只有PUSH操作增加栈S中的对象个数
- 所有POP和MULTIPOP 弹出的对象数不会弹出多于PUSH入栈的对象数
- 故总的运行时间为$O(n)$
  - 所有PUSH入栈的对象数为$O(n)$
  - 所有POP和MULTIPOP 弹出的对象数也为$O(n)$
- 每个PUSH、 POP和MULTIPOP 操作的平摊时间
  - $O(n)/n = O(1)$
- **聚集法：**通过总运行时间求平均得到平摊时间，不需要对操作序列的概率分布做假设。

二进制计数器

- 计数器$A[0 \,..\, k-1]$ 表示为k位二进制位的数组

$$x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$$

- 操作INCREMENT实现计数器加一
  - 运行时间$O(k)$
  - n个INCREMENT操作序列的运行时间 $O(nk)$（紧吗？）

```
INCREMENT(A)
i ← 0
while i < length[A] and A[i] = 1
    do A[i] ← 0
       i ← i + 1
if i < length[A]
    then A[i] ← 1
```

INCREMENT操作的平摊时间

- *n*个INCREMENT操作序列的运行时间应为：$O(n)$
- 观察INCREMENT 操作序列
  - 每次操作，A[0]都反转；
  - 每两次操作，A[1]反转；
  - 每2^i次操作，A[*i*]反转；
- 于是，总反转次数为

$$\sum_{i=0}^{\lfloor \log(n) \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor < n\sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

- 总时间$O(n)$
- 每个操作平摊时间$O(n)$ /n= $O(1)$

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

平摊分析——记账法

记账法

- 对不同的操作赋予不同的费用，某些操作的费用比它们的实际代价或多或少。
- 我们对一个操作的收费的数量称为平摊代价。
  - 当一个操作的平摊代价超过了它的实际代价时，两者的差值就被当作存款，并赋予数据结构中的一些特定对象，可以用来补偿那些平摊代价低于其实际代价的操作。
  - 记账法与聚集分析的区别：
    聚集分析中的所有操作都具有相同的平摊代价。
- 数据结构中存储的总存款等于总的平摊代价和总的实际代价之差。注意：总存款不能是负的。

n个栈操作的平摊时间

操作	实际代价	平摊代价
PUSH	**1**	**2**
POP	**1**	**0**
MULTIPOP	**min(*k*,*s*)**	**0**

- 对**PUSH**操作多收费，多出来的**1**元钱放在入栈的对象上，待**POP**和**MULTIPOP**把该对象弹出栈时，恰好每个对象上的**1**元前抵消了操作的实际代价
- 因为栈S中对象数不可能为负，即存款不会小于**0**
  - 保证n个操作平摊时间之和是n个操作实际时间的上界

二进制计数器上的记账法分析

- 两个问题
  - 如何对INCREMENT收费？ （平摊代价）
  - 收的费用放在哪里？（如何平摊）

二进制计数器上的记账法分析

- 注意到，每次**INCREMENT**只会把一个**0**反转为**1**，但可能把多个**1**反转为**0**。
  - **INCREMENT**的实际代价为反转的次数
  - **INCREMENT**平摊代价为**2**，用于把**0**反转为**1**，并把剩余的**1**元存放在反转成**1**的位上。
  - 把**1**反转成**0**的实际代价由存放在**1**上的**1**元钱支付
- ***A*[1..*k*-1]**数组中，**1**的个数（=存款）不可能为负

练习题

- 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$O(n)$

平摊分析——势能法

势能法（potential method）

- 不是将已预付的费用作为存在数据结构特定对象中存款来表示，而是表示成一 种“势能” 或“势”，它在需要时可以释放出来，以支付后面操作的额外开销。
- 势是与整个数据结构而不是其中的个别对象发生联系的。（区别于记账法）

势函数Φ

- 对一个初始数据结构D_0执行n个操作。对每个i，设c_i为每个操作的实际代价， D_i为对数据结构D_{i-1}执行第i个操作的结果。
- 势函数Φ将每个数据结构D_i映射为一个实数$\Phi(D_i)$, 即与D_i相联系的势。
- 第i个操作的平摊代价a_i根据势函数Φ定义为

$$a_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

- 即每个操作的平摊代价为其实际代价c_i加上由于该操作所增加的势。n个操作的总的平摊代价为

$$\sum a_i = \sum c_i + \Phi(D_n) - \Phi(D_0)$$

势函数Φ

- 如果势函数Φ使得对所有n，有$\Phi(D_n) \geq \Phi(D_0)$，
  则总平摊代价$\sum a_i$就是总实际代价$\sum c_i$的一个上界
  - 通常为了方便起见会定义$\Phi(D_0) = 0$（不是必须的）
  - 正的势差存储势
    - 如果第i个操作的势差$\Phi(D_i) - \Phi(D_{i-1})$是正的，则平摊代价$a_i$表示对第$i$个操作多收了费，同时数据结构的势也随之增加了。
  - 负的势差不足收费
    - 如果势差是负值，则平摊代价就表示对第i个操作的补足收费，这是通过减少势来支付该操作的实际代价。
  - 平摊代价依赖于所选择的势函数Φ。
    - 不同的势函数可能会产生不同的平摊代价，但它们都是实际代价的上界。最佳势函数的选择取决于所需的时间界。

栈操作——势能法分析

- 定义势函数为栈中对象的个数
  - 开始时要处理是空栈D_0，所以$\Phi(D_0)=0$
  - 栈中的对象数始终非负，所以$\Phi(D_i) \geq 0= \Phi(D_0)$
  - 以Φ表示的n个操作的平摊代价的总和，那么Φ就是总的实际代价的一个上界
- 对于作用于一个包含s个对象的栈上的第i个操作
- 如果**PUSH**，则
  - 势差为$\Phi(D_i) - \Phi(D_{i-1}) = (s+1) - s = 1$
  - 平摊代价为$a_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1+1=2$

栈操作——势能法分析

- 如果是**MULTIPOP(S,k)**
  - 该操作的实际代价为$c_i = \min(s, k)$
  - 势差为$\Phi(D_i) - \Phi(D_{i-1}) = -\min(s, k)$
  - 平摊代价为

    $a_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = \min(s, k) - \min(s, k) = 0$
- 类似的，如果是**POP**，平摊代价也为**0**
- 三种栈操作每种的平摊代价都是$O(1)$， 这样包含n个操作的序列的总平摊代价就是$O(n)$。已经证明n个操作的平摊代价的总和是总的实际代价的一个上界。所以n个操作的最坏情况代价为$O(n)$

二进制计数器——势能法分析

- 计数器***A*[0 .. *k* -1]** 表示为*k*位二进制位的数组

$$x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$$

- 操作INCREMENT实现计数器加一
- 如何定义势？

```
INCREMENT(A)
i ← 0
while i < length[A] and A[i] = 1
    do A[i] ← 0
       i ← i + 1
if i < length[A]
    then A[i] ← 1
```

二进制计数器——势能法分析

- 定义势函数为数组$A[0...k\text{-}1]$中1的个数

  计算INCREMENT的平摊代价
  - 设第i次操作把t_i个1反转为0，则实际代价为$t_i +1$
  - 设第i次操作后数组中1的个数为b_i
    - 如果$b_i = 0$，则第i次操作反转了k个1，有$b_{i\text{-}1}=t_i=k$；
    - 如果$b_i > 0$，则$b_i = b_{i\text{-}1} - t_i + 1$。
    - 两种情形下都有：$b_i \le b_{i\text{-}1} - t_i + 1$
- 势差为 $\Phi(D_i)\text{-}\Phi(D_{i\text{-}1}) \le (b_{i\text{-}1} - t_i + 1) - b_{i\text{-}1} = 1 - t_i$
- 平摊开销为 $a_i = c_i +\Phi(D_i)\text{-}\Phi(D_{i\text{-}1}) \le (t_i+1)+(1\text{-}t_i) = 2$

二进制计数器——初值非零

- 设计数器中初始有b_0个1，n次INCREMENT操作后有b_n个1。（$0 \le b_0, b_n \le k$， k为计数器位数）

  对所有的i， $a_i \le 2$

  $\Phi(D_n) = b_n$， $\Phi(D_0) = b_0$

  n次INCREMENT操作总代价

  $\sum c_i = \sum a_i - \Phi(D_n) + \Phi(D_0) \le \sum 2 - b_n + b_0 = 2n - b_n + b_0$
- 只要$k \le O(n)$，则总代价就为$O(n)$
- 这里并没有要求$\Phi(D_n) \ge \Phi(D_0)$，也没有要求$\Phi(D_0) = 0$

练习题

- 考虑普通二叉最小堆上最坏运行时间为$O(\log n)$的操作INSERT和EXTRACT-MIN。请给出势函数Φ，使得INSERT操作的平摊代价为$O(\log n)$，EXTRACT-MIN的平摊代价为$O(1)$。
- 说明如何用两个普通的栈来实现一个队列，使得每个ENQUEUE和DEQUEUE操作的平摊代价都为$O(1)$。

动态表及其上的平摊分析

动态表

- 在有些应用中，在开始的时候无法预知在表中要存储多少个对象。这就希望能根据对象的多少调整所需要的存储空间（多退少补）
- 表的动态扩张和收缩——动态表
  - 用平摊分析证明插入、删除操作的平摊代价是$O(1)$
  - 动态表的具体结构可以是堆、栈、散列表等等
  - 假设我们用数组来存储动态表（不是链表）
  - 通过插入扩张和删除搜索
    - 保证未使用空间总是不多于整个分配空间的一定比例
- 先看只有插入操作的情形，再拓展也有删除操作

插入算法

```
Insert(T, x)
if size[T]=0
    then 给table[T]分配一个槽的空间
        size[T]←1
if num[T]=size[T]
    then 分配一个有2*size[T]个槽的空间的新表
        将table[T]中所有的项插入到新表中
        释放table[T]
        table[T]指向新表的存储块地址
        size[T]←2*size[T]
将x插入table[T]
num[T]←num[T]+1
```

一次插入操作的代价

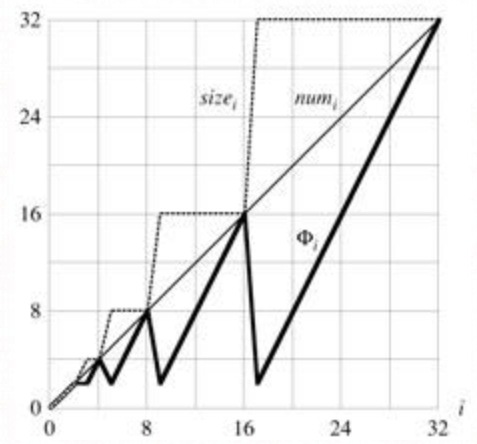
- 简单分析，一次插入操作的代价为$O(n)$
  - 于是n次插入操作总代价为$O(n^2)$
- 聚集分析，一次插入操作的代价为
  - 当i-1是2的幂时：$c_i = i$
  - 其他时候：$c_i = 1$
  - 总代价 $$\sum_{i=0}^{n} c_i \le n + \sum_{i=0}^{\lfloor \log(n) \rfloor} 2^i \le n + 2\mathrm{n} = 3n$$
  - 平摊代价为$3n/n = 3$
- 记账法，插入收费3，其中2元钱分别赋给表中最后插入的对象和一个已经没有存款的对象

当需要扩张表时，每个对象被复制一次，恰好用完存款

表扩张时插入操作的平摊代价

- 势函数： $\Phi(T) = 2num[T] - size[T]$
- 当第i个操作不触发表扩张，则

$$
\begin{aligned}
a_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 1 + (2 \cdot num_i - size_i) - (2(num_i - 1) - size_i) \\
&= 3
\end{aligned}
$$



- 当第i个操作触发表扩张，则

$$
\begin{aligned}
a_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= num_i + (2 \cdot num_i - 2(num_i - 1)) - (2(num_i - 1) - (num_i - 1)) \\
&= num_i + 2 - (num_i - 1) = 3
\end{aligned}
$$

表扩张和收缩时的平摊代价

- 插入/删除操作：满时扩张、小于**1/4**时收缩
- 势函数
  - $\Phi(T) = \max(2num[T] - size[T], size[T]/2 - num[T])$
- 插入操作的平摊代价都小于**3**
  - 根据是否引起扩张来分别计算
- 删除操作的平摊代价都小于**2**
  - 根据是否引起收缩来分别计算
- 总平摊代价为$O(n)$

- 插入/删除操作：满时扩张、小于**1/3**时收缩，势函数？

更多平摊分析经典问题

- Splay trees
- Redblack trees
- Fibonacci heaps
- Disjoint sets
- Maximum flow
- Hash table
- Scapegoat trees

平摊分析小结

- 平摊分析的概念
- 平摊分析的三种方法
  - 聚集分析
  - 记账法
  - 势能法
- 动态表及其上的平摊分析
- 参考资料：算法导论第17章