摊还分析

王侦 关注
2017.12.05 01:20:13 字数 4,768 阅读 1,873

0.目录

在摊还分析中，求数据结构的一个操作序列中所执行的所有操作的平均时间，来评价操作的代价。
摊还分析不同于评价情况分析，它并不涉及概率。

摊还分析最常用的三种技术：

- 聚合分析：这种方法用来确定一个n个操作序列的总代价的上界T(n)，因此每个操作的平均代价为T(n)/n
- 核算法：用来分析每个操作的摊还代价。核算法将序列中某些较早的操作的“余额”作为“预付信用”存储起来，与数据结构中的特定对象相关联。在操作序列中随后的部分，存储的信用即可用来为拿些缴费少于实际代价的操作支付差额
- 势能法：也是分析每个操作的摊还代价，也是通过较早的操作的余额来补偿稍后操作的差额。势能法将信用作为数据结构的势能存储起来，且将势能作为一个整体存储，而不是将信用与数据结构中单个对象关联分开存储。

利用聚合分析，我们证明对所有n，一个n个操作的序列最坏情况下花费的总时间为T(n)。因此，在最坏情况下，每个操作的平均代价，或摊还代价为T(n)/n。
此摊还代价是适用于每个操作的，即使序列中有多种类型的操作也是如此。

1.2 聚合分析的实例

1.2.1 栈操作

基本的栈操作，时间复杂性均为O(1)：

- PUSH(S, x)：将对象x压入栈S中
- POP(S)：将栈S的栈顶对象弹出，并返回该对象。对空栈调用POP产生错误

新增一个栈操作MULTIPOP(S, k)：删除栈S栈顶的k个对象，如果栈中对象数少于k，则将整个栈的内容都弹出。

```
MULTIPOP(S, k)
1  while not STACK-EMPTY(S) and k>0
2      POP(S)
3      k=k-1
```

一个包含s个对象展现执行MULTIPOP(S, k)，总代价为min(s, k)。

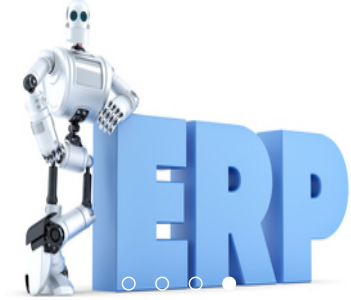
聚合分析：
考虑整个序列的n个操作，在一个空栈上执行n个PUSH、POP和MULTIPOP的操作序列，代价至多O(n)。
因为将一个对象压入栈后，至多将其弹出一次，对一个非空的栈，可以执行的POP操作的次数（包括MULTIPOP中调用POP次数）最多与PUSH操作的次数相当，即最多n次。因此，对任意的n值，任意一个有n个PUSH、POP和MULTIPOP的操作序列，最多花费O(n)时间。
因此三种栈操作的摊还代价为O(1)。

1.2.2 二进制计数器递增

k位二进制计数器递增的问题，计数器的初值为0.
用一个位数组A[0..k-1]作为计数器，其中A.length = k。当计数器中保存的二进制值为x时，x的最低位保存在A[0]，而最高位保存在A[k-1]中。因此：

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

```
INCREMENT(A)
1  i=0
2  while i < A.length and A[i]==1
3      A[i]=0
4      i=i+1
5  if i < A.length
```

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

最坏情况：INCREMENT执行一次花费Θ(k)时间。

聚合分析：
每次调用INCREMENT是A[0]确实都会翻转；
A[1]每两次调用翻转一次；
A[2]每四次调用才翻转一次；
...
因此，对于一个初值为0的计数器，在执行一个由n个INCREMENT操作组成的序列的过程中，进行翻转的总数为：

$$\sum_{i=0}^{k-1}\left\lfloor\frac{n}{2^i}\right\rfloor < n\sum_{i=0}^{\infty}\frac{1}{2^i} = 2n$$

因此，对一个初值为0的计数器，执行一个n个INCREMENT操作的序列的最坏情况时间为O(n)。每个操作的平均代价，即摊还代价为O(1)。

1.2.3 动态表

对某些应用程序，无法预知将会有多少个对象存储在表中。
不够用时，必须为其分配更大的空间，并将所有对象从原表中复制到新的空间中；
若从表中删除了很多对象，为其重新分配一个更小的内存空间就是值得的。

使用摊还分析证明，虽然插入和删除操作可能会引起扩张或收缩，从而有较高的实际代价，但它们的摊还代价都是O(1)，并保证动态表中的空闲空间相对于总空间的比例永远不超过一个常量分数。

装载因子α：表中存储的数据项数量 除以 表的规模（槽的数量）。

1）表扩张——只允许插入数据项
一个常用的分配新表的启发式策略：为新表分配2倍于旧表的槽。如果只允许插入操作，那么装载因子总是保持在1/2以上，因此，浪费的空间永远不会超过总空间的一半。
如下，T对应表：
T.table保存指向表的存储空间的指针；
T.num保存表中的数据项数量
T.size保存表的规模（槽数）

推荐阅读
笔试的题目（栈和队列）
阅读 221
第三章 检测理论
阅读 169
iOS知识汇总
阅读 964
散列表-HashMap
阅读 298
Redis底层数据结构
阅读 133

```
2        allocate T.table with 1 slot
3        T.size =1
4    if T.num==T.size
5        allocate new-table with 2 • T.size slots
6        insert all items in T.table into new-table
7        free T.table
```

运行时间分析：
将每次基本插入操作的代价设定为1，然后用基本插入操作的次数来描述TABLE-INSERT的运行时间。

最坏情况分析：
第i个操作的代价ci。如果当前表满，会发生一次扩张，则ci = i：基本插入操作代价1，加上从旧表到新表的复制代价i-1。

聚合分析：
执行n个TABLE-INSERT操作，扩张动作是很少的，仅当i-1恰为2的幂时，第i个操作才会引起一次扩张。

$$c_i = \begin{cases} i & \text{若 } i-1 \text{ 恰为 2 的幂} \\ 1 & \text{其他} \end{cases}$$

因此，n 个 TABLE-INSERT 操作的总代价为

$$\sum_{i=1}^{n} c_i \leqslant n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j < n + 2n = 3n$$

因此单一操作的摊还代价至多为3

2）表扩张和收缩——既允许插入也允许删除
见势能法

2.核算法

2.1 核算法的核心要点（信用与特定对象相关联）

- 用核算法进行摊还分析，对不同操作赋予不同费用，赋予某些操作的费用可能多于或少于其实际代价。将赋予一个操作的费用称为它的摊还代价。
- 信用：当一个操作的摊还代价超出其实际代价时，将差额存入数据结构中的特定对象，存入的差额称为信用。
  对于后续操作中摊还代价小于实际代价的情况，信用可以用来支付差额。因此，将一个操作的摊还代价分解为其实际代价和信用（存入的或用掉的）
  不同的操作可能有不同的摊还代价，不同于聚合分析。
- 必须小心地选择操作的摊还代价。若希望通过分析摊还代价来证明每个操作的平均代价的最坏情况很小，就应确保操作序列的总摊还代价给出了序列总真实代价的上界。这种关系必须对所有操作序列都成立。

$$\sum_{i=1}^{n} \hat{c}_i \geqslant \sum_{i=1}^{n} c_i$$

推荐阅读
笔试的题目（栈和队列）
阅读 221
第三章 检测理论
阅读 169
iOS知识汇总
阅读 964
散列表-HashMap
阅读 298
Redis底层数据结构
阅读 133

$$\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$$

数据结构中存储的信用为两者的差值，信用必须一直为非负值。

2.2 核算法的实例

2.2.1 栈操作

1）操作的实际代价为：

PUSH	1
POP	1
MULTIPOP	$\min(k, s)$

2）为这些操作赋予如下摊还代价：

PUSH	2
POP	0
MULTIPOP	0

为什么这样赋值（特点——信用与特定的对象绑定在一起，例如这里每个元素都绑定一个预存的信用）：
为每个入栈的元素存储对应的一个信用（一个实际代价+一个信用=总数为2），作为POP和MULTIPOP弹出时使用。因此，对任意n个PUSH、POP、MULTIPOP操作组成的序列，总摊还代价为总实际代价的上界。
由于总摊还代价为O(n)，因此总实际代价也是。

2.2.2 二进制计数器递增

1）操作的实际代价
置位 1
复位 1

2）为这些操作赋予如下摊还代价：
置位 2
复位 0

为什么这样赋值：
进行置位时，用1支付置位操作的实际代价，并将另外1存为信用，用来支付将来复位操作的代价。
在任何时候，计数器中任何为1的位都存有1的信用，这样对于复位操作，就无需缴纳任何费用。

INCREMENT的摊还代价：
INCREMENT过程至多置位一次，因此，其摊还代价最多为2

推荐阅读
笔试的题目（栈和队列）
阅读 221
第三章 检测理论
阅读 169
iOS知识汇总
阅读 964
散列表-HashMap
阅读 298
Redis底层数据结构
阅读 133

2.2.3 动态表

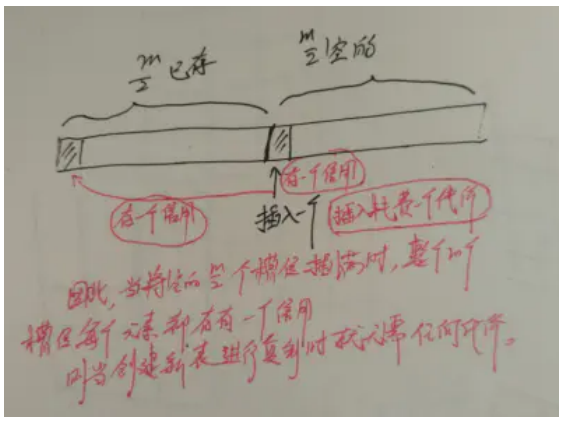
装载因子α：表中存储的数据项数量 除以 表的规模（槽的数量）。

```
TABLE-INSERT(T, x)
 1  if T.size==0
 2      allocate T.table with 1 slot
 3      T.size =1
 4  if T.num==T.size
 5      allocate new-table with 2 • T.size slots
 6      insert all items in T.table into new-table
 7      free T.table
 8      T.table=new-table
 9      T.size =2 • T.size
10  insert x into T.table
11  T.num=T.num+1
```

1）表扩张——只允许插入数据项

为每次插入赋予摊还代价为3，为什么赋值为3：

假定表的规模在一次扩张后边为m，则表中保存了m/2个数据项，且它没有存储任何信用。为每次插入操作付3的代价，3的分配如下：



2）表扩张和收缩——既允许插入也允许删除

见势能法

2.2.4 二项队列的插入分析

类似于二进制计数器

推荐阅读

笔试的题目（栈和队列）
阅读 221

第三章 检测理论
阅读 169

iOS知识汇总
阅读 964

散列表-HashMap
阅读 298

Redis底层数据结构
阅读 133

为这些操作赋予如下摊还代价：
插入 2
链接 0

为什么这样赋值：
进行插入时，用1支付置位操作的实际代价，并将另外1存为信用，用来支付将来该位置上链接操作的代价。
在任何时候，二项队列中任何有树的位都存有1的信用，这样对于链接操作，就无需缴纳任何费用。

插入的摊还代价：
插入过程至多置位一次，因此，其摊还代价最多为2.
二项队列中树的个数永远不会为负，因此，任何时刻信用都是非负。因此，对于n个插入操作，总摊还代价为O(n)，为总实际代价的上界。

3.势能法

3.1 势能法的核心要点（势能与整个数据结构相关联，而不是特定对象）

- 势能法摊还分析将预付代价表示为势能，将势能释放即可用来支付未来操作的代价。将势能与整个数据结构而不是特定对象相关联
- 势能法工作方式

势能法工作方式如下。我们将对一个初始数据结构 D_0 执行 n 个操作。对每个 $i=1, 2, \cdots, n$，令 c_i 为第 i 个操作的实际代价，令 D_i 为在数据结构 D_{i-1} 上执行第 i 个操作得到的结果数据结构。**势函数** Φ 将每个数据结构 D_i 映射到一个实数 $\Phi(D_i)$，此值即为关联到数据结构 D_i 的**势**。第 i 个操作的**摊还代价** $\hat{c}_i$ 用势函数 Φ 定义为：

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \tag{17.2}$$

每个操作的摊还代价等于其实际代价加上此操作引起的势能变换。n个操作总的摊还代价为：

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

如果能定义一个势函数 Φ，使得 $\Phi(D_n) \geqslant \Phi(D_0)$，则总摊还代价 $\sum_{i=1}^{n} \hat{c}_i$ 给出了总实际代价 $\sum_{i=1}^{n} c_i$ 的一个上界。实际中，我们不是总能知道将要执行多少个操作。因此，如果对所有 i，我们要求 $\Phi(D_i) \geqslant \Phi(D_0)$，则可以像核算法一样保证总能提前支付。我们通常将 $\Phi(D_0)$ 简单定义为 0，然后说明对所有 i，有 $\Phi(D_i) \geqslant 0$。（处理的一种简单方法参见练习 17.3-1，其中 $\Phi(D_0) \neq 0$。）

直觉上，如果第 i 个操作的势差 $\Phi(D_i) - \Phi(D_{i-1})$ 是正的，则摊还代价 $\hat{c}_i$ 表示第 i 个操作多付费了，数据结构的势增加。如果势差为负，则摊还代价表示第 i 个操作少付费了，势减少用于支付操作的实际代价。

推荐阅读
笔试的题目（栈和队列）
阅读 221
第三章 检测理论
阅读 169
iOS知识汇总
阅读 964
散列表-HashMap
阅读 298
Redis底层数据结构
阅读 133