第六章

Amortized Analysis

(自学)

骆吉洲
计算机科学与工程系

提要

6.1 *Elements of Amortized Analysis*
6.2 *Aggregate Analysis*
6.3 *The Accounting Method*
6.4 *The Potential Method*
6.5 *Dynamic Arrays*

参考资料

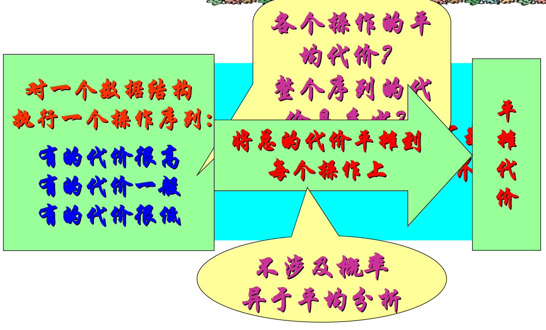
Chapter 6 Amortized Analysid

Pages 185 - 194

6.1 *Elements of Amortized Analysis*

- *Amortized Analysis*的基本思想
- *Amortized Analysis*方法

Amortized Analysis的基本思想



Amortized Analysis的基本思想

- Aggregate Analysis方法 *(每个操作的代价)*
  - 为每个操作都赋予相同的平摊代价
  - 确定n个操作的上界$T(n)$, 每个操作平摊$T(n)/n$
- Accounting方法 *(整个操作序列的代价)*
  - 不同类型操作赋予不同的平摊代价
  - 某些操作在数据结构的特殊对象上"预付"代价
- Potential方法 *(整个操作序列的代价)*
  - 不同类型操作赋予不同的平摊代价
  - "预付"的代价作为整个数据结构的"能量"

HIT
CS&E

6.2 Aggregate Analysis

- 聚集方法的原理
- 聚集方法的实例之一
- 聚集方法的实例之二

HIT
CS&E

聚集方法的原理

数据结构上共有n个操作，最坏情况下：

操作1: Cost=t_1
操作2: Cost=t_2
.
.
.
操作n: Cost=t_n

$T(n)=\sum_{i=1}^{n} t_i$

平摊代价：$T(n)/n$

每个操作被赋予相同代价，不管操作类型

HIT
CS&E

聚集方法实例之一：栈操作系列

- 普通栈操作及其时间代价
  - Push(S, x): 将对象压x入栈S
  - Pop(S)：弹出并返回S的顶端元素
  - 两个操作的运行时间都是$O(1)$
  - 可把每个操作的实际代价视为1
  - n个Push和Pop操作系列的总代价是n
  - n个操作的实际运行时间为$\theta(n)$

HIT
CS&E

- 新的普通栈操作及其时间代价
  - 操作Multipop(S, k):
    去掉S的k个顶对象，当$|S|<k$时弹出整个栈
  - 实现算法

    Multipop(S, k)
    1 While not STACK-EMPTY(S) and $k\neq0$ Do
    2 Pop(S);
    3 $k \leftarrow k-1$.
  - Multipop(S, k)的实际代价 (设Pop的代价为1)
    - Multipop的代价为$min(|S|, k)$

HIT
CS&E

- 初始栈为空的n个栈操作序列的分析
  - n个栈操作序列由Push、Pop和Multipop组成
  - 粗略分析
    - 最坏情况下，每个操作都是Multi
    - 每个Multipop的代价最坏是$O(n)$
    - 操作系列的最坏代价为$T(n) = O(n^2)$
    - 平摊代价为$T(n)/n = O(n)$
  - 精细分析
    - 一个对象在每次被压入栈后至多被弹出一次
    - 在非空栈上调用Pop的次数(包括在Multipop内的调用)至多为Push执行的次数，即至多为n
    - 最坏情况下操作序列的代价为$T(n)\leq 2n = O(n)$
    - 平摊代价=$T(n)/n=O(1)$

分析太粗糙！！！

n-1个push
1个multipop

HIT
CS&E

聚集方法实例之二：二进计数器

- 问题定义：由0开始计数的k位二进计数器

  输入：k位二进制变量x，初始值为0

  输出：$x+1$ mod 2^k

  数据结构：

  $A[0..k-1]$作为计数器，存储x

  x的最低位在$A[0]$中，最高位在$A[k-1]$中

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

HIT CS&E

- **计数器加1算法**

输入：$A[0..k-1]$，存储二进制数 x

输出：$A[0..k-1]$，存储二进制数 $x+1 \bmod 2^k$

```
INCREMENT(A)
i←0
while i<k and A[i]=1 Do
      A[i]←0;
      i←i+1;
If i<k Then A[i]←1
```

- **初始为零的计数器上 n 个INCREMENT操作分析**

Counter Value N	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	每个操作 Cost
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	2
3	0	0	0	0	0	0	1	1	1
4	0	0	0	0	0	1	0	0	3
5	0	0	0	0	0	1	0	1	1
6	0	0	0	0	0	1	1	0	2
7	0	0	0	0	0	1	1	1	1
8	0	0	0	0	1	0	0	0	4
9	0	0	0	0	1	0	0	1	1
10	0	0	0	0	1	0	1	0	2
11	0	0	0	0	1	0	1	1	1
12	0	0	0	0	1	1	0	0	3
13	0	0	0	0	1	1	0	1	1
14	0	0	0	0	1	1	1	0	2
15	0	0	0	0	1	1	1	1	1
16	0	0	0	1	0	0	0	0	5

HIT CS&E

- **粗略分析**
  - 每个Increment的时间代价最多 $O(k)$
  - n 个Increment序列的时间代价最多 $O(kn)$
  - n 个Increment平摊代价为 $O(k)$
  - 例如上例中：$k*n=8*16=128$
- **精细分析**

HIT CS&E

操作Cost＝O(发生改变的位数)

Counter Value N	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

HIT CS&E

- **精细分析**
  - $A[0]$ 每1次操作发生一次改变，总次数为 n
  - $A[1]$ 每2次操作发生一次改变，总次数为 $n/2$
  - $A[2]$ 每4次操作发生一次改变，总次数为 $n/2^2$
  - $A[3]$ 每8次操作发生一次改变，总次数为 $n/2^3$
  - 一般地
    - 对于 i=0, 1, …., lg$n$, $A[i]$ 改变次数为 $n/2^i$
    - 对于 i>lg$n$, $A[i]$ 不发生改变

      （因为 n 个操作结果为 n，仅涉及 $A[0]$ 至 $A[\lg n]$ 位）
  - $T(n)= \Sigma_{0\le i\le \lg n}\, n/2^i < n\, \Sigma_{0\le i\le\infty} 1/2^i = 2n = O(n)$
  - 每个Increment操作的平摊代价为 $O(1)$

HIT CS&E

6.3 The Accounting Method

- Accounting方法的原理
- Accounting方法的实例之一
- Accoutning方法的实例之二

HIT CS&E

Accounting方法的原理

- Accounting方法
  - 目的是分析n个操作序列的复杂性上界
  - 一个操作序列中有不同类型的操作
  - 不同类型的操作的操作代价各不相同
  - 于是我们为每种操作分配不同的平摊代价
    - 平摊代价可能比实际代价大，也可能比实际代价小

数据结构中存储的Credit在任何时候都必须非负，即$\sum_{1\le i\le n} \alpha_i - \sum_{1\le i\le n} c_i \ge 0$永远成立

  - 平摊代价的选择规则：
    - 设c_i和α_i是操作i的实际代价和平摊代价
    - $\sum_{1\le i\le n}\alpha_i \ge \sum_{1\le i\le n} c_i$ 必须对于任意n个操作序列都成立

HIT CS&E

栈操作序列的分析

- 各栈操作的实际代价
  - Cost(PUSH)=*1*
  - Cost(POP)=*1*
  - Cost(MULTIPOP)=$min\{k, s\}$
- 各栈操作的平摊代价
  - Cost(PUSH)=2
    - 一个*1*用来支付PUSH的开销,
    - 另一个*1*存储在压入栈的元素上, 预支POP的开销
  - Cost(POP)=*0*
  - Cost(MULTIPOP)=*0*
- 平摊代价满足
  - $\sum_{1\le i\le n}\alpha_i - \sum_{1\le i\le n} c_i \ge 0$ 对于任意n个操作序列都成立
    - 因为在n个操作序列中, POP个数(包括MULTIPOP中的POP)不大于PUSH个数.
- n个栈操作序列的总平摊代价
  - $O(n)$

HIT CS&E

二进制计数器Increment操作序列分析

- Increment操作的平摊代价
  - 每次一位被置*1*时，付*2*美元
    - *1*美元用于置*1*的开销
    - *1*美元存储在该"*1*"位上，用于支付其被置*0*时的开销
    - 置*0*操作无需再付款
  - Cost(Increment)=*2*
- 平摊代价满足
  - $\sum_{1\le i\le n}\alpha_i \ge \sum_{1\le i\le n} c_i$ 对于任意n个操作序列都成立，因为从前面的分析可知$\sum_{1\le i\le n} c_i < 2n$
- n个Increment操作序列的总平摊代价
  - $O(n)$

HIT CS&E

6.3 The Potential Method

- Potential方法的原理
- Potential方法的实例之一
- Potential方法的实例之二

HIT CS&E

Potential方法的原理

- Potential方法
  - 目的是分析n个操作系列的复杂性上界
  - 在会计方法中，如果操作的平摊代价比实际代价大，我们将余额与数据结构的数据对象相关联
  - Potential方法把余额与整个数据结构关联，所有的这样的余额之和，构成**数据结构的势能**
    - 如果操作的平摊代价大于操作的实际代价，**势能增加**
    - 如果操作的平摊代价小于操作的实际代价，要用数据结构的势能来支付实际代价，**势能减少**

- 数据结构势能的定义
  - 考虑在初始数据结构D_0上执行n个操作
  - 对于操作i
    - 操作i的实际代价为c_i
    - 操作i将数据结构D_{i-1}变为D_i
    - 数据结构D_i的势能是一个实数$\phi(D_i)$，ϕ是一个正函数
    - 操作i的平摊代价：$\alpha_i = c_i + \phi(D_i) - \phi(D_{i-1})$
  - n个操作的总平摊代价（必须是实际代价的上界）

$$\sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} (c_i + \phi(D_i) - \phi(D_{i-1}))$$
$$= \sum_{i=1}^{n} c_i + \phi(D_n) - \phi(D_0)$$

  - 关键是ϕ的定义
    - 保证$\phi(D_n) \ge \phi(D_0)$，使总平摊代价是总实际代价的上界
    - 如果对于所有i，$\phi(D_i) \ge \phi(D_0)$，可以保证$\phi(D_n) \ge \phi(D_0)$
    - 实际可以定义$\phi(D_0) = 0$，$\phi(D_i) \ge 0$

栈操作序列的分析

- 栈的势能定义
  - $\phi(D_m)$定义为栈D_m中对象的个数，于是
    - $\phi(D_0)=0$，D_0是空栈
    - $\phi(D_i)\ge 0=\phi(D_0)$，因为栈中对象个数不会小于0
    - n个操作的总平摊代价是实际代价的上界
  - 栈操作的平摊代价（设栈D_{i-1}中具有s个对象）
    - PUSH: $\alpha_i=c_i+\phi(D_i)-\phi(D_{i-1})=1+(s+1)-s=2$
    - POP: $\alpha_i=c_i+\phi(D_i)-\phi(D_{i-1})=1+(s-1)-s=0$
    - MULTIPOP(S, k): 设$k'=min(k,s)$

      $\alpha_i=c_i+\phi(D_i)-\phi(D_{i-1})=k'+(s-k')-s=k'-k'=0$
  - n个栈操作序列的平摊代价是$O(n)$

二进制计数器操作序列分析

- 计数器的势能定义
  - $\phi(D_m)$定义为第m个操作后计数器中1的个数b_m
    - $\phi(D_0)=0$，D_0中1的个数为0
    - $\phi(D_i)\ge 0=\phi(D_0)$，因为计数器中1的个数不会小于0
    - 于是，n个操作的总平摊代价是实际代价的上界
  - INCREMENT操作的平摊代价
    - 第i个INCREMENT操作将 t_i个1置0，实际代价为t_i+1
    - 计算操作i的平摊代价$\alpha_i=c_i+\phi(D_i)-\phi(D_{i-1})$
      - If $b_i=0$, 操作i resets所有k位，所以$b_{i-1}=t_i=k$
      - If $b_i>0$, 则$b_i=b_{i-1}-t_i+1$
      - 于是 $b_i\le b_{i-1}-t_i+1$

        $\phi(D_i)-\phi(D_{i-1})=b_i-b_{i-1}\le b_{i-1}-t_i+1-b_{i-1}=1-t_i$
      - 平摊代价$\alpha_i=c_i+\phi(D_i)-\phi(D_{i-1})\le(t_i+1)+(1-t_i)=2$
  - n个操作序列的总平摊代价是$O(n)$

6.5 Dynamic Arrays

- 动态表的概念
- 动态表的扩张与收缩
- 仅含扩张操作的动态表平摊分析
- 一般的动态表平摊分析

动态表—基本概念

- 动态表支持的操作
  - ·TABLE-INSERT:将某一元素插入表中
  - ·TABLE-DELETE:将一个元素从表中删除
- 数据结构:用一个(一组)数组来实现动态表
- 非空表T的装载因子$\alpha(T)=T$存储的对象数/表大小
  - 空表的大小为0，装载因子为1
  - 如果动态表的装载因子以一个常数为下界，则表中未使用的空间就始终不会超过整个空间的一个常数部分

动态表的表示

设T表示一个动态表:
- $table[T]$是一个指向表示表的存储块的指针
- $num[T]$包含了表中的项数
- $size[T]$是T的大小
- 开始时，$num[T]=size[T]=0$

动态表的扩张

- 插入一个数组元素时,完成的操作包括
  - 分配一个包含比原表更多的槽的新表
  - 再将原表中的各项复制到新表中去
- 常用的启发式技术是分配一个比原表大一倍的新表，
  - 只对表执行插入操作,则表的装载因子总是至少为1/2
  - 浪费掉的空间就始终不会超过表总空间的一半

HIT CS&E

扩张算法

```
算法: TABLE—INSERT(T, x)                    /*复杂的插入操作*/
1     If size[T]=0 Then                      /*开销为常数*/
2         获取一个大小为1的表 table[T];
3         size[T]←1;
4     If num[T]=size[T] Then                 /*开销取决于size[T]*/
5         获取一个大小为 2×size[T]的新表new-table;
6         将 table[T]中元素插入new-table;    /*简单插入操作*/
7         释放 table[T];
8         table[T]←new-table;
9         size[T]←2×size[T];
10    将 x插入table[T];
11    num[T]←num[T]+1
```

HIT CS&E

初始为空的表上n次插入操作的代价分析

聚集分析-粗略分析

- 考察第i次操作的代价C_i
  - 如果i=1, C_i=1;
  - 如果$num[T]<size[T]$, C_i=1;
  - 如果$num[T]=size[T]$, $C_i=i$;
- 共有n次操作
  - 最坏情况下,每次进行n次操作,总的代价上界为n^2
- 这个界不精确
  - n次TABLE—INSERT操作并不常常包括扩张表的代价
  - 仅当i-1为2地整数幂时第i次操作才会引起一次表地扩张

聚集分析-精细分析

- 第i次操作的代价C_i
  - 如果$i=2^m$, $C_i=i$; 否则C_i=1
- n次TABLE—INSERT操作的总代价为

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

★ 每一操作的平摊代价为$3n/n=3$

HIT CS&E

初始为空的表上n次插入操作的代价分析

会计方法

- 每次执行TABLE—INSERT平摊代价为3
  - 1支付第11步中的基本插入操作的实际代价
  - 1作为自身的存款
  - 1存入表中第一个没有存款的数据上
- 当发生表的扩张时，数据的复制的代价由数据上的存款来支付
- 任何时候，存款总和非负
- 初始为空的表上n次TABLE-INSERT操作的平摊代价总和为$3n$

HIT CS&E

初始为空的表上n次插入操作的代价分析

势能法分析

?势怎么定义才能使得表满发生扩张时势能能支付扩张的代价

- 如果势能函数满足
  - 刚扩充完,$\phi(T)$=0
  - 表满时 $\phi(T)=size(T)$
- $\phi(T)=2*num[T]-size[T]$
  - $num[T]\ge size[T]/2$, $\phi(T)\ge 0$
  - n次TABLE－INSERT操作的总的平摊代价是总的实际代价的一个上界
- 第i次操作的平摊代价
  - 如果发生扩张,c_i=3
  - 如果未发生扩张, c_i=3
- 初始为空的表上n次插入操作的代价的上界为$3n$

HIT CS&E

动态表的扩张与收缩

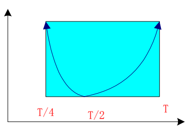
- 表的扩张
- 表的收缩
  - 表具有一定的丰满度
  - 表的操作序列的复杂度是线性的
- 表的收缩策略
  - 表的装载因子小于1/2时，收缩表为原表的一半
  - $n=2^k$,考察下面的一个长度为n的操作序列:
    - 前$n/2$个操作是插入，后跟I D D I I D D I I…
    - 每次扩张和收缩的代价为O(n)，共有O(n)扩张或收缩
    - 总代价为$O(n^2)$，而每一次操作的平摊代价为$O(n)$--每个操作的平摊代价太高
- 改进的收缩策略(允许装载因子低于1/2)
  - 满表中插入数据项时，将表扩大一倍
  - 删除数据项引起表不足1/4满时，将表缩小为原表的一半
  - 扩张和收缩过程都使得表的装载因子变为1/2
  - 表的装载因子的下界是1/4

HIT CS&E

动态表上n次(插入、删除)操作的代价分析

势能函数的定义

- 操作序列过程,势能总是非负的
  - 保证一列操作的总平摊代价即为其实际代价的一个上界
- 表的扩张和收缩过程要消耗大量的势
- 势能函数应满足
  - $num(T)=size(T)/2$时，势最小
  - 当$num(T)$减小时，势增加直到收缩
  - 当$num(T)$增加时，势增加直到扩充



- 势能函数特征的细化
  - 当装载因子为1/2时，势为0
  - 装载因子为1时，有$num[T]=size[T]$，即$\phi(T)=num[T]$。这样当因插入一项而引起一次扩张时，就可用势来支付其代价
  - 当装载因子为1/4时，$size[T]=4\cdot num[T]$。即$\phi(T)=num[T]$。因而当删除某项引起一次收缩时就可用势来支付其代价

$$\Phi(T)=\begin{cases} 2\cdot num[T]-size[T] & \alpha(T)\ge 1/2 \\ size[T]/2-num[T] & \alpha(T)<1/2 \end{cases}$$

HIT CS&E

平摊代价的计算

- 第i次操作的平摊代价：$c'_i = c_i + \phi(T_i) - \phi(T_i - 1)$
  - 第i次操作是TABLE—INSERT：未扩张 $c'_i \le 3$
  - 第i次操作是TABLE—INSERT：扩张 $c'_i \le 3$
  - 第i次操作是TABLE—DELETE：未收缩 $c'_i \le 3$
  - 第i次操作是TABLE—DELETE：收缩 $c'_i \le 3$
- 所以作用于一动态表上的n个操作的实际时间为$O(n)$

HIT CS&E

summary