算法设计与分析—进阶篇

第六讲 平摊分析

哈尔滨工业大学
王宏志
wangzh@hit.edu.cn
http://homepage.hit.edu.cn/pages/wang/

本讲内容

6.1 平摊分析原理

6.2聚集方法

6.3 会计方法

6.4 势能方法

6.5 动态表操作的平摊分析

?

各个操作的代价？

对一个数据结构要执行一系列操作：

有的代价很高

有的代价一般

有的代价很低

分析中，执行一

均而得

将总的代价平摊到每个操作上

平摊代价

不涉及概率

不同于平均情况分析

平摊分析的方法

- 聚集方法
- 会计方法
- 势能方法

本讲内容

6.1 平摊分析原理

6.2聚集方法

6.3 会计方法

6.4 势能方法

6.5 动态表操作的平摊分析

聚集分析法-原理

对数据结构共有n个
最坏情况下：
操作1： t_1
操作2： t_2
。
。
。
操作n： t_n

操作序列中的每个操作被赋予相同的代价，不管操作的类型

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

平摊代价：
T(n)/n

平摊分析实例1-栈操作

普通栈操作

PUSH(S,x)：将对象压入栈S

POP(S)：弹出并返回S的顶端元素

时间代价:

- 两个操作的运行时间都是O(1)
- 我们可把每个操作的代价视为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θ(n)

平摊分析实例1-栈操作

新的栈操作

操作MULTIPOP(S,k)：

执行一次While循环要调用一次POP

实现算法

输入：栈S，k

输出：返回S顶端k个对象

```
MULTIPOP(S,k)
1 While not STACK-EMPTY(S) and k≠0 Do
  2         POP(S);
  3         k←k-1
```

MULTIPOP的总代价即为min(s,k)

平摊分析实例1-栈操作

初始为空的栈上的n个

由PUSH、POP和M

操作1: t_1
操作2: t_2
。
。
。
操作n: t_n

!

Note：分析过程没有使用任何的概率！

$T(n)<=2n$

于是：最坏情况下这样的一个操作序列的时间复杂度最多为O(n)

平摊分析实例2-二进计数器

1. 问题定义

实现一个由 0 开始向上计数的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

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

平摊分析实例2-二进计数器

- **2. 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 mod 2^k

```
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
```

平摊分析实例2-二进计数器

- **3.**初始为零的计数器上n个**INCREMENT**操作的分析

每隔8次发生一次改变

共$\lfloor n/8 \rfloor$

Counter

N	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	3
3	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	4
4	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	7
5	0	0	0	[illegible]	[illegible]	[illegible]	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11

总共发生的改变为：

$\Sigma\lfloor n/2^i \rfloor$ $(i=0,2,\ldots,[\log_2 n]$

$<2n$

本讲内容

6.1 平摊分析原理

6.2聚集方法

6.3 会计方法

6.4 势能方法

6.5 动态表操作的平摊分析

会计方法

平摊代价可能比实际代价大，也可能比实际代价小

一个操作序列中有不同类型的操

于是：我们在各种操作上定义平摊代价使得任意操作序列上存款总量是非负的，将操作序列上平摊代价求和即可得到这个操作序列的复杂度上界

会计方法实例 1 — 栈操作

1. 各栈操作的实际代价：

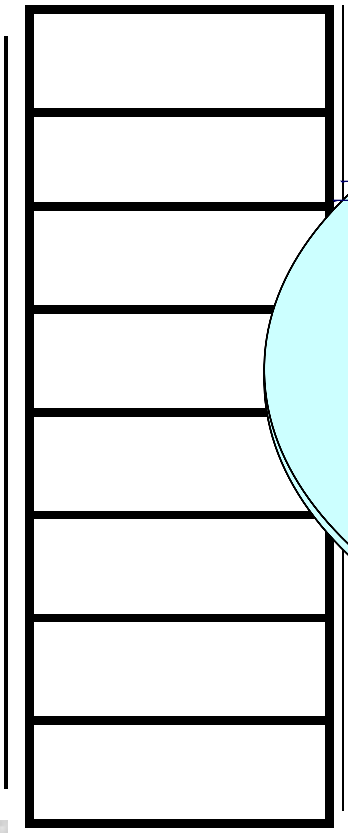
PUSH	1,
POP	1,
MULTIPOP	min(k,s)

2. 各栈操作的平摊代价：

PUSH	2,
POP	0,
MULTIPOP	0,

会计方法实例 1 — 栈操作

3. 栈操作序列代价分析



于是所有操作的平摊代价总和就是所有操作序列代价总和的上界

长度为n的操作序列中:

PUSH操作的个数<=n

于是:

平摊代价的总和<=2n

=?

会计方法实例 2 –二进计数器

1. 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 mod 2^k

```
    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
```

会计方法实例2－二进计数器

初始为零的计数器上n个INCREMENT操作的分析

任何操作序列，存款余额是计数器中1的个数，非负

因此，所有的翻转操作的平摊代价的和是这个操作

序列代价的上界

本讲内容

6.1 平摊分析原理

6.2聚集方法

6.3 会计方法

6.4 势能方法

6.5 动态表操作的平摊分析

势能分析—基本原理

在会计方法中，如果操作的平摊代价比实际代价大，我们将余额与具体的数据对象关联

如果我们将这些余额都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

势能分析—基本原理

势能的定义：

对一个初始数据结构D_0执行n个操作

对操作i:

实际代价c_i 将数据结构D_{i-1} 变为D_i

势函数ϕ将每个数据结构D_i映射为一个实数$\phi(D_i)$

平摊代价c'_i定义为：$c'_i = c_i + \phi(D_i) - \phi(D_{i-1})$

势能分析—基本原理

- n个操作的总的平摊代价为：

$$\sum_{i=1}^{n} c'_i \ldots$$

$$= \sum_{i=1}^{n} c_i + \ldots \Phi(D_0)$$

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

于是势函数φ满足$\phi(D_n) \geq \phi(D_0)$，则总的平摊代价就是总的实际代价的一个上界

势能方法实例1 — 栈操作

$\phi(D)$=栈D中对象的个数

初始栈D_0为，$\phi(D_0)=0$

因为栈中的对象数始终非负，第i个超作之后的栈D_l满足$\phi(D_i)\geq 0=\phi(D_0)$

于是：　以ϕ表示的n个操作的平摊代价的总和就表示了实际代价的一个上界

势能方法实例1 — 栈操作

作用于包含s个对象的栈上的栈操作的平摊代价

平摊分析：

每个栈操作的平摊代价都是O(1)

n个操作序列的总平摊代价就是O(n)

因为$\phi(D_i) \ge \phi(D_0)$， n个操作的总平摊代价即为总的实际代价的一个上界，即n个操作的最坏情况代价为O(n)

势能方法实例 2 –二进计数器

- ·$\phi(D)$=计数器D中1的个数
- ·计数器初始状态D_0中1的个数为0，$\phi(D_0)=0$
- ·因为栈中的对象数始终非负，第i个超作之后的栈D_i满 足$\phi(D_i)\geq 0=\phi(D_0)$
- ·于是：n个操作的平摊代价的总和就表示了实际代价的一个上界

势能方法实例 2 -二进计数器

第i次INCREMENT操作的平摊代价

计数器初始状态为0时的平摊分析 ：

每个栈操作的平摊代价都是O(1)

n个操作序列的总平摊代价就是O(n)

因为$\phi(D_i)\geq\phi(D_0)$， n个操作的总平摊代价即为总的实际代价的一个上界，即n个操作的最坏情况代价为O(n)

势能方法实例 2 -二进计数器

- 开始时不为零的计数器上n个INCREMENT操作的分析

设：开始时有b_0个1　　$0 \le b_0$

在n次INCREMENT操作之后有b_n个1

因为$\phi(D_0)=b_0$，
作的总的实际代价为

如果我们执行了至少$n=\Omega(k)$次INCREMENT操作，则无论计数器中包含什么样的初始值，总的实际代价都是O(n)

$i=1$　　$i=1$

正是势能法，给我们这样的分析带来了方便!

本讲内容

6.1 平摊分析原理

6.2聚集方法

6.3 会计方法

6.4 势能方法

6.5 动态表操作的平摊分析

动态表

- 动态表的概念
- 本节的目的：
  - 研究表的动态扩张和收缩的问题
  - 利用平摊分析证明插入和删除操作的平摊代价为O(1)，即使当它们引起了表的扩张和收缩时具有较大的实际代价
  - 研究如何保证一动态表中未用的空间始终不超过整个空间的一部分

动态表-基本术语

- 动态表支持的操作

  ·TABLE-INSERT：将

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

设T表示一个表:

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

算法：TABLE—INSERT(T, x)

```
If  size[T]=0
Then  allocate table[T] with 1 slot;
      size[T]←1;
If  num[T]=size[T] Then
      allocate new table with 2×size[T] slots;
      insert all items in table[T] into new-table;
      free table[T];
      table[T]←new-table;
      size[T]←2×size[T];
Insert x into table[T];
num[T]←num[T]+1
```

开销由size[T]决定

动态表-表的扩张

初始为空的表上n次TABLE-INSERT操作的代价分析-粗略分析

```
算法：TABLE—INSERT(T,x)
...
        insert all items in table[T] into new-table;
        free table[T];
        table[T]←new-table;
        size[T]←2×size[T];
    Insert x into table[T];
    num[T]←num[T]+1
```

这个界不精确，因为执行n次TABLE—INSERT操作的过程中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

第i次操作的代价C_i:

如果i=1 $c_i=1$

如果表有空间 $c_i=1$

如果表是满的 $c_i=i$

如果以共有n次操作：

情况下

每次进行n次操作

总的代价上界为n^2

动态表-表的扩张

初始为空的表上n次TABLE-INSERT操作的代价分析-聚集分析

```
算法：TABLE—INSERT(T, x)
    If  size[T]=0
    Then  allocate table[T] with 1 slot;
        size[T]←1;
    If  num[T]=size[T] Then
        allocate new table with 2×size[T] slots;
        insert all items in table[T] into new-table;
        free table[T];
        table[T]←new-table;
        size[T]←2×size[T];
  Insert x into table[T];
  num[T]←num[T]+1
```

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

动态表-表的扩张

初始为空的表上n次TABLE-INSERT操作的代价分析-会计法分析

```
算法：TABLE—INSERT(T, x)
1
                                     h 1 slot;

4         If  num[T]=size[T] Th
5             allocate new table with 2×size[T
6             insert all items in table[T] into
7             free table[T];
8             table[T]←new-table;
9             size[T]←2×size[T];
10      Insert x into table[T];
11      num[T]←num[T]+1
```

任何时候，存款总和非负

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

初始为空的表上n次TABLE-INSERT操作的平摊代价总和为3n

动态表-表的扩张

初始为空的表上n次TABLE-INSERT操作的代价分析-势能法分析

```
算法：TABLE—INSERT(T, x)
If  size[T]=0
Then  allocate table[T] with 1 slo
      size[T]←1;
If  num[T]=size[T
      allocate new
      insert all item
      free table[T];
      table[T]←new-table;
      size[T]←2×size[T];
Insert x into table[T];
num[T]←num[T]+1
```

由

n次TA

$\phi(T)=2*num[T]-size[T]$

第i次操作的平摊代价：

如果发生扩张： $c'_i = 3$

否则 $c'_i = 3$

初始为空的表上n次TABLE-INSERT操作的平摊代价总和为3n

动态表-表的扩张和收缩

表的扩张

表的收缩

理想情况下，我们希望表满足：

表具有一定的丰满度

表的操作序列的复杂度是线性的

动态表-表的扩张和收缩

表的收缩策略

上面的收缩策略可以改善，允许装载因子低于1/2

方法：当向满的表中插入一项时，还是将表扩大一倍

但当删除一项而引起表不足1/4满时，我们就将表缩小为原来的一半

这样，扩张和收缩过程都使得表的装载因子变为1/2
但是，表的装载因子的下界是1/4

动态表–表的扩张和收缩

- 由n个TABLE—INSERT和TABLE-DELETE操作构成的序列的代价的分析-势能法

第次操作的平摊代价：$c'_i = c_i + \phi(T_i) - \phi(T_{i-1})$

第i次操作是TABLE—INSERT：未扩张 $c'_i \le 3$

第i次操作是TA $c'_i \le 3$

所以作用于一动态表上的
n个操作的实际时间为O(n)

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 $c'_i \le 3$