8. Amortized Analysis

2018年10月30日　　19:28

Define $\sigma = (\sigma_1, \dots, \sigma_m)$ be a sequence of operations on data structure D
Define t_i time to execute σ_i

Sequence complexity $C(m) = \max\{\sum_1^m t_i : any\ sequence\ of\ \sigma\ of\ m\ operations\}$
Amortized complexity $A(m) = \frac{C(m)}{m}$: time each operation takes in average

Example consider a binary counter which keeps a number $x\ mod\ 2^k$ by increment
Using a data structure $A[0, \dots, k-1]$ to store x
$A \coloneqq [a_0, \dots, a_{k-1}], a_i \in \{0,1\}. x = \sum a_i 2^i$

```
increment(A)
    j = 0
    while j < k and A[j] == 1:
        A[j] = 0
        j ++
    if j == k:
        A[j] = 0
    A[j] = 1
```

Method 1 Aggregate
Consider the number of bits flipped in each increment (actual cost of steps)
$000 \to^1 001 \to^2 \to 010 \to^1 011 \to^3 100 \to^1 101 \to^2 110 \to^1 111 \to \cdots$
$A[0]$ flipped every time, $A[i]$ flipped every second time ...
$A[j]$ fliiped every 2^j time

$$C(m) = \#bit\ flips = \sum_0^{k-1} \#A[j]\ flip = \sum_0^{k-1} floor\left(\frac{m}{2^i}\right) \le m \sum_0^{k-1} \frac{1}{2^i} \le m \sum_0^{\infty} \frac{1}{2^i} = 2m$$
$A(m) = 2 \in \Theta(1)$

Method 2 Potential Function
Define D_0 be the initial state of the data structure
D_i be the state of the ds after σ_i

Define $\Phi(D_i)$ be the potential function, $a_i = t_i + \Phi(D_i) - \Phi(D_{i-1})$ be the amortized complexity

Proposition $\Phi(D_i) \ge 0 \Rightarrow \sum^m t_i \le \sum^m a_i + \Phi(D_0)$
proof $\sum^m t_i = \sum^m \big(a_i + \Phi(D_{i-1}) - \Phi(D_i)\big) = \sum^m a_i + \Phi(D_0) - \Phi(D_m)$

Back to binary counter example
Since the most step consumed operation is to flip 1 -> 0, say $0 \le j \le k$ times in each operation, and flip 0 -> 1 at most once.
Consider t_i, each operation flips at most $j + 1$ bits
Consider $\Phi(A_i) = \#1bit\ in\ A_i$
Then $\Phi(A_i) - \Phi(A_{i-1}) \le -j + 1$ Since j number of 1-bit at the end of A_i are flipped, and at most one 0-bit is flipped
$a_i = t_i + \Phi(A_i) - \Phi(A_{i-1}) \le (j + 1 - j + 1) = 2$

Dynamic Table
T: table (e.g. a hash table)

$T.size :=$ #slots
$T.num :=$ #items stored
$loadfactor\ d(T) := \dfrac{T.size}{T.num}$
Consider operation

```
insert(T, x)
    if d(T) == 1
        allocate T'
        T'.size = 2T.size
        copy T to T'
        insert x into T'
        T ⇐ T'
    insert x into T
```

Most time consumed operation is copy the stored item into T'.
It's trivial that the empty slots can only exists in the second half of the table after each insertion, hence $0.5 < d(T) \leq 1$
Consider $\Phi(T_i) = 2 \times \#occupied\ slots\ in\ T\left[\frac{n}{2}+1, \dots, n\right] = 2\left(T_i.num - \frac{T_i.size}{2}\right) = 2T_i.num - T_i.size$

Since each such slot will store one item and compensate for the future copying to T'.

Consider 2 cases

1. the if branch is not called, then $t_i = 1, \Phi_i - \Phi_{i-1} = 2, a_i = 1 + 2 = 3$
2. the if branch is called, then $t_i = T_{i-1}.size + 1, \Phi_i - \Phi_{i-1} = 2T_{i-1}.num + 2 - 2T_{i-1}.size - 2T_{i-1}.num - T_{i-1}.size = 2 - T_{i-1}.size$
   $a_i = T_{i-1}.size + 1 + 2 - T_{i-1}.size = 3$