Lec13 平摊分析、表的扩增、势能方法

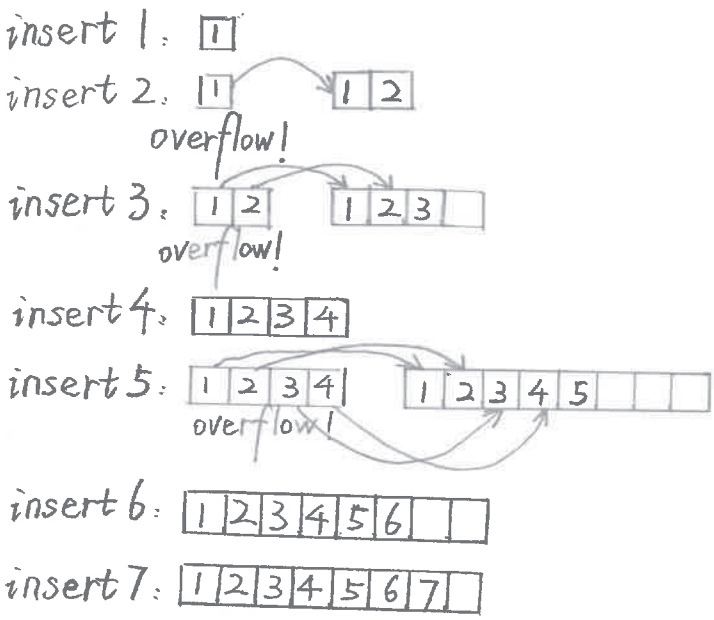
Amortized Analysis

analyze a sequence of operations to show that the average cost per operation is small, even though one or several operation(s) maybe expensive (no probability here, it's average performance in the worst case)

Example: dynamic tables

idea is that whenever the table gets too full (overflows), "grow" it

1. allocate a larger table
2. move the items from the old table to the new
3. free the old table



Analysis:

sequence of n insert operations,

the worst-case cost of 1 insert is $\Theta(n)$,

but the worst-case cost of n inserts is NOT $n \cdot \Theta(n) = \Theta(n^2)$!

因为不是所有的项都是最坏情况。

n inserts take $\Theta(n)$ time.

let c_i be the cost of the i-th insert, then $c_i = \begin{cases} i, & \text{if } i-1 = 2^x \\ 1, & \text{otherwise} \end{cases}$

i	1	2	3	4	5	6	7	8	9	10
$size_i$	1	2	4	4	8	8	8	8	16	16
c_i	1	2	3	1	5	1	1	1	9	1

so cost of n inserts $= \sum_{i=1}^{n} c_i = n + \sum_{j=0}^{\lfloor \lg(n-1) \rfloor} 2^j \leq 3n = \Theta(n)$

thus, average cost per insert is $\frac{\Theta(n)}{n} = \Theta(1)$

"尽管有时候要付出比较大的代价，但这个巨大的开销会被之前的操作平摊掉"

three types of amortized arguments

1. aggregate analysis (just saw，基本上就是要你分析 n 次操作一共花费多少时间)
2. accounting
3. potential

} more precise, because they allocate specific amortized costs to each operatio

Accounting Method

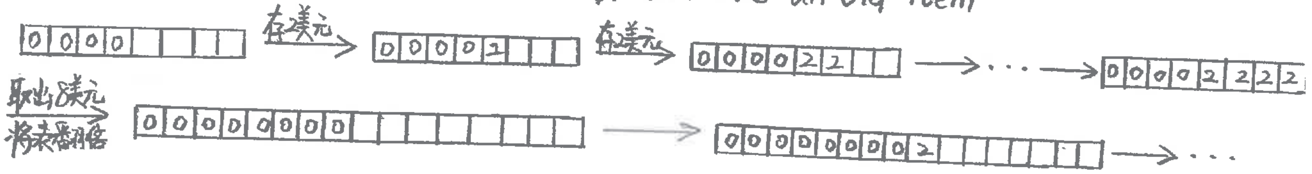
- charge the i-th operation a fictitous amortized cost $\hat{c}_i$ (虚构的)
  (\$1 pays for 1 unit of work)
- fee is consumed to perform the operation
- unused amount is stored in the bank for use by later operations
- bank balance must not go negative (不能借款)

must have $\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \quad , \forall n$

c_i ← true cost

Example: dynamic table

- charge $\hat{c}_i = \$3$ for the i-th insert,
  \$1 pays for an immediate insert, so \$2 gets stored (预存，用于将表翻倍)
- when the table doubles.
  \$1 to move a rescent item. \$1 to move an old item



"我总是能偿付所有表扩增的开销"

$$\underbrace{\sum_{i=1}^{n} \hat{c}_i}_{= 3n} \ge \sum_{i=1}^{n} c_i$$

i	1	2	3	4	5	6	7	8	9	10
$size_i$	1	2	4	4	8	8	8	8	16	16
$\hat{c}_i$	2	3	3	3	3	3	3	3	3	3
$bank_i$	1	2	2	4	2	4	6	8	2	4

收了3美元，花了4美元用于复制旧项，又花了1美元插入新项，$4+3-4-1=2$

这里也可以收3美元，这样之后都会多出1美元

注：也可以每次收4美元，5、6、7、…都是可行的。
但不能每次收2美元，这样余额会变为负数。

Potential Method

"what do you aspire, to be a bookkeeper or to be a physicist?"

"bank account" viewed as potential energy of dynamic set

framework:

- start with data structure D_0
- operation i transforms D_{i-1} into D_i
- cost of operation i is c_i
- define the potential function $\phi: \{D_i\} \to R$ such that

  $\phi(D_0) = 0$ and $\phi(D_i) \geq 0 \ \forall i$
- define amortized cost $\hat{c}_i$, $\hat{c}_i = c_i + \underbrace{\phi(D_i) - \phi(D_{i-1})}_{\text{change in potential} \cdot \Delta\phi_i}$

if $\Delta\phi_i > 0$, then $\hat{c}_i > c_i$.

// I charged more than it costs me to do the operation

operation i stores work in the data structure for later

if $\Delta\phi_i < 0$, then $\hat{c}_i < c_i$

data structure delivers up stored work to help pay for operation i

从势能的角度,与从记帐法的角度比较,用记帐方法来看,会先决定一个平摊代价,然后再分析一下银行存款,确保它不为负值。在某种程度上,在势能法里,会说"我的存款是这样子的",然后再分析一下哪个平摊代价才合适。

total amortized cost of n operations is

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

$$= \sum_{i=1}^{n} c_i + \underbrace{\phi(D_n)}_{\geq 0} - \underbrace{\phi(D_0)}_{=0}$$

$$\geq \sum_{i=1}^{n} c_i \quad \text{(telescope, 缩近)}$$

Example: table doubling

define $\phi(D_i) = 2i - 2^{\lceil \log_2 i \rceil}$, assume $\phi(D_0) = 0$, note $\phi(D_i) \geq 0 \ \forall i$

0	0	0	0	0	0		

$\phi(D_6) = 2 \times 6 - 2^{\lceil \log_2 6 \rceil} = 4$

amortized cost of the i-th insert

$$\hat{c_i} = c_i + \phi(D_i) - \phi(D_{i-1})$$

$$= \begin{cases} i, & \text{if } i-1 = 2^x \\ 1, & \text{otherwise} \end{cases} + 2i - 2^{\lceil \log_2 i \rceil} - 2(i-1) + 2^{\lceil \log_2 i-}$$

$$= \begin{cases} i, & \text{if } i-1 = 2^x \\ 1, & \text{otherwise} \end{cases} + 2 - 2^{\lceil \log_2 i \rceil} + 2^{\lceil \log_2 (i-1) \rceil}$$

$$= \begin{cases} i + 2 - 2^{\lceil \log_2 i \rceil} + 2^{\lceil \log_2 (i-1) \rceil}, & \text{if } i-1 = 2^x \\ 1 + 2 - 2^{\lceil \log_2 i \rceil} + 2^{\lceil \log_2 (i-1) \rceil}, & \text{otherwise} \end{cases}$$

$$= \begin{cases} i + 2 - 2(i-1) + (i-1), & \text{if } i-1 = 2^x \\ 1 + 2, & \text{otherwise} \end{cases}$$

$$= \begin{cases} 3, & \text{if } i-1 = 2^x \\ 3, & \text{otherwise} \end{cases}$$

$$= 3$$

therefore, the amortized cost is 3, for each insert

so, n inserts costs $\Theta(n)$ in the worst case

平摊分析的结论：

① amortized costs provide a clean abstraction for data structure performance
简洁的 抽象

只要你不关注实时表现，只关注聚集行为，这将是一个相当好的性能抽象概念。

即使有的时候代价会很大，但也会被平摊掉。

② any method can be used, but each has situations where it is arguably simplest or most precise

③ different potential functions, or accounting costs, may yield different bounds
产生