Hashing

Hashing

Hash 的目标：使搜索代价降低到 $O(1+\alpha)$ 其中 α 随输入规模变大增长得很慢

BF Approch：Direct-address table，考虑所有可能的 key，实际上不可行

> CLRS 3rd edition P.254
>
> Direct addressing is a simple technique that works well when the universe U of keys is reasonably small. Suppose that an application needs a dynamic set in which each element has a key drawn from the universe $U = \{0, 1, \ldots, m-1\}$ where m is not too large. We shall assume that no two elements have the same key.

Basic idea

建立一个 hash table，对每个 key 使用 hash 函数算出其在表中的地址。由于 key 的空间大于表的大小，根据鸽笼原理，必定会出现 collision，即两个不同的 key 映射到表中同一个地址，此时就需要 collision handling

Collision Handling

设 hash table 的大小为 m，插入元素个数为 n

定义哈希表的 load factor $\alpha = \dfrac{n}{m}$

closed address analysis

基本思想：表中每一个地址对应一个链表，映射到相同地址的 key 存入同一个链表（头插）

最差情况下搜索的代价会达到 $\Theta(n)$，现分析平均情况

假设：simple uniform hashing

> CLRS 3rd edition P.259

> ..., but for now we shall assume that any given element is equally likely to hash into any of the m slots, independently of where any other element has hashed to. We call this the assumption of **simple uniform hashing**.

即假设 hash table 中每个地址的链的长度为 $\frac{n}{m}$

Unsuccessful search

不在表中的元素根据假设被等可能地映射到 m 个地址，故代价为

$$\Theta(1+n/m)$$

Successful search

重要：为每个表中的元素标号，x_i 即为第 i 个插入表中的元素

> 这种假设的思想很重要，类比基于指标随机变量分析快速排序的平均情况时间复杂度时将输入元素按照大小次序标号而非输入次序

- 对任意 x_i ，被搜索的概率为 $\frac{1}{n}$
- 对于 x_i ，成功搜索代价为 $1+t$ ，其中 t 为在 x_i 之后被插入 x_i 同一链表的元素个数（链表采用头插法）

即代价为

$$\frac{1}{n}\sum_{i=1}^{n}(1+t)$$

而根据之前假设，每个元素等可能映射到 m 个地址，故代价为

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right)\\
=&\ 1+\frac{1}{nm}\sum_{i=1}^{n}(n-i)\\
=&\ 1+\frac{1}{nm}\sum_{i=1}^{n-1}i\\
=&\ 1+\frac{n-1}{2m}\\
=&\ 1+\frac{\alpha}{2}-\frac{\alpha}{2n}\\
=&\ \Theta(1+\alpha)
\end{aligned}$$

open address analysis

基本思想：所有元素存储在 hash table 中，α 不超过 1

发生 collision 时使用新函数探查下一个地址，直至不发生 collision

探查序列遍历表中所有位置，可看作 m 个元素的一种排列

假设：uniform hashing

> CLRS 3rd edition P.271
>
> In our analysis, we assume uniform hashing: the probe sequence of each key is equally likely to be any of the $m!$ permutations of $\langle 0, 1, \ldots, m-1 \rangle$.

即假设每个 key 选取 probe 序列为 $m!$ 种排列中的任意一种是等概率的

Probe

- Linear probing：

$$h(k,i) = (h'(k) + i) \bmod m \quad (i = 0, 1, \ldots, m-1)$$

- Quadratic probing:

$$h(k,i) = \left(h'(k) + c_1 i + c_2 i^2\right) \bmod m \quad (i = 0, 1, \ldots, m-1)$$

- Double hashing:

$$h(k,i) = (h_1(k) + i h_2(k)) \bmod m \quad (i = 0, 1, \ldots, m-1)$$

对于 linear probing 和 quadratic probing，均只有 m 种序列，取决于 probe 的起始元素

Unsuccessful search

基于假设，第一个探查位置被占据的概率为 $\frac{n}{m}$，第 j 个探查位置被占据的概率为 $\frac{n-j+1}{m-j+1}$

probe 次数至少为 i 次的概率为

$$\frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdot \cdots \cdot \frac{n-i+2}{m-i+2} \leqslant \left(\frac{n}{m}\right)^{i-1} = \alpha^{i-1}$$

故平均 probe 次数为

$$\sum_{i=1}^{\infty} \alpha^{i-1} = \sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha}$$

> 设有随机变量 $X = \{0, 1, 2, \ldots\}$

$$
\begin{aligned}
E[x] &= \sum_{i=0}^{\infty} i \cdot \Pr\{X = i\} \\
&= \sum_{i=0}^{\infty} i \cdot (\Pr\{X \geqslant i\} - \Pr\{X \geqslant i+1\}) \\
&= \sum_{i=0}^{\infty} i \cdot \Pr\{X \geqslant i\} - \sum_{i=1}^{\infty} (i-1) \cdot \Pr\{X \geqslant i\} \\
&= \sum_{i=1}^{\infty} \Pr\{X \geqslant i\}
\end{aligned}
$$

Successful search

思想：检索第 $i+1$ 个被插入的元素代价与表中仅有 i 个元素时插入元素的代价是相等的（Unsuccessful search）

在表中仅有 i 个元素时，$\alpha = \dfrac{i}{m}$ ，由上文结论，插入代价为 $\dfrac{1}{1-\frac{i}{m}} = \dfrac{m}{m-i}$

故代价为

$$
\begin{aligned}
\frac{1}{n} \sum_{i=0}^{n-1} \frac{m}{m-i} &= \frac{m}{n} \sum_{i=0}^{n-1} \frac{1}{m-i} \\
&= \frac{1}{\alpha} \sum_{i=m-n+1}^{m} \frac{1}{i} \\
&\leqslant \frac{1}{\alpha} \int_{m-n}^{m} \frac{dx}{x} \\
&= \frac{1}{\alpha} \ln \frac{m}{m-n} \\
&= \frac{1}{\alpha} \ln \frac{1}{1-\alpha}
\end{aligned}
$$

成功搜索代价至多为 $\dfrac{1}{\alpha} \ln \dfrac{1}{1-\alpha}$

Hash Function

好的 hash function 可以尽可能达到 simple uniform hashing

- the division method：$h(k) = k \bmod m$
- the multiplication method：$h(k) = \lfloor m(kA \bmod 1) \rfloor_{(0<A<1)}$

但是没有一个函数能避免 $\Theta(n)$ 的 worst-case

further-reading：Gonnet and Baeza-Yates: Handbook of Algorithms and Data Structures, Addison-Wesley, 1991.

Amortized Analysis

> The problem of “unusually expensive” individual operation

e.g.， Array doubling：当数组满时重新分配一个原来大小两倍的数组，并将元素依次复制到新数组中

n 次插入的代价：显然不是 $O(n^2)$

$$c_i = \begin{cases} i & \text{if } i-1 \text{ is exactly the power of 2} \\ 1 & \text{otherwise} \end{cases}$$

$$\text{So the total cost is: } \sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j < n + 2n = 3n$$

关键：**expensive operation 和 usual operation 之间的关系**

> CLRS 3rd edition P.451
>
> Amortized analysis differs from average-case analysis in that probability is not involved; an amortized analysis guarantees the average performance of each operation in the worst case.

Aggregate analysis

思想：一个 n 个操作的序列总的最坏情况时间复杂度为 $T(n)$，则对每个操作而言 amortized cost 为 $T(n)/n$

e.g.，MULTIPOP

在栈的两种操作 POP 和 PUSH （代价均为 1）之外新增 MULTIPOP(k) 操作，即从栈顶弹出 k 个元素，若栈中元素个数不足 k 则全部弹出

分析：n 次包含 POP, PUSH, MULTIPOP 的操作序列最坏情况时间复杂度为 $O(n)$，因为每个元素最多被 POP 一次，而总的 POP 次数 （POP 和 MULTIPOP）等于 PUSH 次数，最多为 n，故总的时间复杂度为 $O(n)$，根据 aggregate analysis，每次操作平均时间复杂度为

$$O(n)/n = O(1)$$

e.g.，Incrementing a binary counter

统计反转 bit 的代价，对于第 i 位（最低位为 0），增加 n 次，翻转 $\lfloor n/2^i \rfloor$ 次，总的代价为

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

每次操作平均情况时间复杂度为

$$O(n)/n = O(1)$$

The accounting method

为每个操作分配 amortized cost，当 amortized cost 多于 actually cost 时，多出来的部分为 credit，而当 amortized cost 少于 actually cost 时，则观察 credit 是否能补足

> CLRS 3rd edition P.456
>
> In the accounting method of amortized analysis, we assign differing charges to different operations, with some operations charged more or less than they actually cost. We call the amount we charge an operation its amortized cost. When an operation's amortized cost exceeds its actual cost, we assign the difference to specific objects in the data structure as credit. Credit can help pay for later operations whose amortized cost is less than their actual cost.

必须确保序列总的 amortized cost 是总的 actually cost 的上界

> HY："不能欠钱"
>
> CLRS 3rd edition P.456
>
> If we denote the actual cost of the i^{th} operation by c_i and the amortized cost of the i^{th} operation by $\hat{c_i}$, we require
>
> $$\sum_{i=1}^{n}\hat{c}_i \geqslant \sum_{i=1}^{n}c_i$$

e.g.，MULTIPOP

Operation	Amortized cost	Actually cost	Credit
POP	0	1	-1
PUSH	2	1	1
MULTIPOP	0	min{k, s}	-min{k, s}

e.g.，Incrementing a binary counter

Operation	Amortized cost	Actually cost	Credit
set a bit to 0	0	1	-1
set a bit to 1	2	1	1

两个例子均易于验证

The potential method

> HY：“平砍攒能量，攒够了发波”
>
> CLRS 3rd edition P.459
>
> ..., the potential method of amortized analysis represents the prepaid work as “potential energy,” or just “potential,” which can be released to pay for future operations.

初始的结构 D_0 ，设第 i 次操作的 actually cost 为 c_i，D_i 为第 i 次操作作用于 D_{i-1} 的结果

potential function Φ 建立起 D_i 和实数的映射，即 D_i 拥有的 “potential”，则可定义第 i 次操作的 amortized cost $\hat{c}_i$ 为

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

即 amortized cost = actually cost + change in potential

$$\begin{aligned}\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} \left(c_i + \Phi\left(D_i\right) - \Phi\left(D_{i-1}\right)\right) \\ &= \sum_{i=1}^{n} c_i + \Phi\left(D_n\right) - \Phi\left(D_0\right)\end{aligned}$$

需要 $\Phi\left(D_n\right) \geqslant \Phi\left(D_0\right)$，实践中由于不知道操作次数，一般定义

$$\begin{gathered}\Phi(D_0) = 0 \\ \Phi\left(D_i\right) \geqslant 0 \text{ for all } i\end{gathered}$$

可理解为 accounting method 中当前 credit 的总和

不同的 Φ 将得出不同的 actually cost 的上界

Incrementing a binary counter: 可定义 $\Phi(D_i) = b_i$，b_i 为 counter 中 1 的个数