Hash 函数的设计优化

天津南开中学 李羽修

Hash 的应用

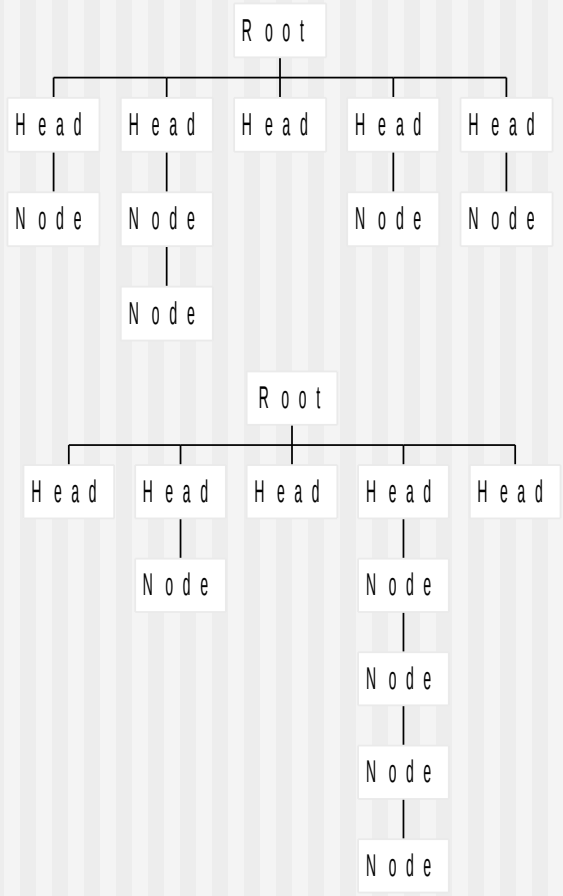
- 字典
- 编译器
- 搜索引擎

从 Hash 表到 Hash 函数

- 既然 Hash 的应用如此广泛，一个好的 Hash 函数则显得尤为重要。
- 在 Hash 函数的帮助下， Hash 表可以处理各种各样的数据
- 整数、实数、字符串、排列组合……

Hash 函数优劣的评价

- 解决冲突是 Hash 表的关键
- 冲突越少， Hash 表的效率就越高
- 数据分布越均匀，冲突越少
- Hash 函数的随机性越好，数据分布越均匀



整数的 Hash 函数

- 最常见的是直接取余法
  - h(k) = k mod M， M 为 Hash 表的容量
  - 为了保证随机性，应该尽量使 k 的每一位都对 h(k) 产生影响
  - M 应选取不太接近 2 的幂的素数
  - 例如 k = 100, M = 12, 那么 h(k) = 4

整数的 Hash 函数

- 把关键字 k 平方，中间几位受 k 的每一位影响最大
- 由此想到平方取中法
  - 把关键字 k 平方，取出中间的 r 位作为 h(k) 的值
  - 此时 $M = 2^r$

整数的 Hash 函数

- 例子:
  - r = 4, k = 100
  - k = $(1100100)_2$
  - k^2 = $(10011\ 1000\ 10000)_2$
  - h(k) = $(1000)_2$ = 8

整数的 Hash 函数

- 另一种方法：利用无理数
- 乘积取整法
  - 用 k 乘以一个 (0,1) 中的无理数 A
  - 取出小数部分，乘以 M
  - 再取出整数部分作为 h(k)
  - 例： k = 100, A = 0.61803..., M = 12
  - h(k) = 9

整数的 Hash 函数

- 比较这三种方法：
  - 直接取余法：
    - 实现容易
    - 效果受 M 影响大
  - 乘积取整法
    - M 可以任取，效果好
    - 速度奇慢
  - 平方取中法
    - 速度快
    - 不容易推广
- 结论：一般的竞赛应用，直接取余法足矣

其他类型数据的 Hash 函数

- 我们可以把其他类型数据转化为整数，然后再利用整数的 Hash 函数将其转化为 Hash 函数值
- 实数还需要转化吗?
- 不需要。范围不太离谱的话可以利用乘积取整法；范围离谱的话……
- 必杀技 (自创)：用整数指针指向实数内存，读出来！然后……

字符串的 Hash 函数

- 字符串信息量巨大，无法直接定址
- 如果能充分采集利用每个字符，随机性可以非常好
- 以 MD5 和 SHA1 为代表的杂凑函数很难找到碰撞
- 下面主要介绍字符串和排列的 Hash 函数

字符串的 Hash 函数

- 首先，不管是字符串还是整数，在计算机中的表示都是二进制序列
- 可以把字符串看成 256 进制的大整数，套用直接取余法
- M 选得好的话，效果还是可以接受的

字符串的 Hash 函数

- 例子： str = “IOI2005”, M = 23
- str = 0x494F4932303035, M = 0x17
- h(str) = str % M = 13
- h(“NOI2005”) = 1
- h(“IOI2004”) = 12
- 出现了扎堆，这是直接取余法的必然现象

字符串的 Hash 函数

- 常用的函数还有很多，大多是用位运算实现
- 竞赛中要找到一个实现复杂度 vs 运行效果的平衡点
- SDBMHash 是一个很好的选择

字符串的 Hash 函数

- 初始时 hash = 0
- 从左到右遍历每一个字符
- for each ch in str
  hash = hash * 65599 + ch
- 去掉符号位返回

字符串的 Hash 函数

- 例子： str = “IOI2005”
  - hash　　ch
  - 00 00 00 00　　‘I’ (49)
  - 00 00 00 49　　‘O’ (4F)
  - 00 49 12 46　　‘I’ (49)
  - 24 41 7F 83　　‘2’ (32)
- 还需要继续往下观察吗?
- 不需要了。 Hash 函数值已经“面目全非”了

字符串的 Hash 函数

- 比较一下相似字符串之 SDBMHash 函数值
  - SDBMHash(“IOI2005”) = 1988023814
  - SDBMHash(“NOI2005”) = 626359947
  - SDBMHash(“IOI2004”) = 1988023813
- 很遗憾，又出现了扎堆
- 解决方法：在字符串后面附加一个长度信息 (借鉴 MD5)
- 但是使用这种方法要牺牲一点点速度

字符串的 Hash 函数

- 重新比较：
  - SDBMHash(“IOI2005”) = 2134682497
  - SDBMHash(“IOI2004”) = 2134616898
  - SDBMHash(“NOI2005”) = 781526076
- 只要 M 不太接近 65599，这个效果基本可以差强人意了

排列的 Hash 函数

- 这里的讨论已经不仅仅局限在“ hashing”，而是推广到“ numerize”，也就是和自然数建立一一对应关系
- direct-address，状态压缩 DP...
- 下面我们研究如何把 n 个元素的 n! 个全排列与 1~n! 之间的自然数建立一一对应

Episode: 关于进位制 (1)

- 数的 p 进制我们已经很熟悉了
  - 每一位逢 p 进一
  - $m=a_0p^0+a_1p^1+a_2p^2+\ldots$
  - $0 \le a_i \le p-1$
- 可不可以第一位逢 2 进一，第二位逢 3 进一，第三位逢 4 进一……?
- 当然可以

Episode: 关于进位制 (1)

- 我们知道：
  - $n!-1 = (n-1)(n-1)! + (n-2)(n-2)! + \ldots + 1*1! + 0*0!$
  - $p^n-1 = (p-1)p^{n-1} + (p-1)p^{n-2} + (p-1)p^{n-3} + \ldots + (p-1)p^1 + (p-1)p^0$
- 何等的相似！
- 由此我们可以定义“变进制数”：
- $m=a_0 0!+a_1 1!+a_2 2!+\ldots$，其中 $0 \le a_i \le i$
- $a_0 0!$ 多余，可以省略

排列的 Hash 函数

- 回到主题
- 为了方便起见，n 个元素我们设为 1,2,…,n
- 对于一个排列，我们数一下元素 i 的右边比 i 小的元素有几个，记为 a_{i-1}
- 显然满足 $0 \le a_i \le i$，因为“个数”不会是负数；同时比 i+1 小的元素不会超过 i 个
- a_i 的序列不就是一个“变进制数”嘛！

排列的 Hash 函数

- “ 变进制数”的本质就是自然数，我们习惯于把它转化为十进制或二进制表示 (使用定义式)
- 同样，我们可以用类似“除 p 取余法”的方法把十进制数转化成“变进制数” (第一位除 2 取余，第二位除 3 取余.....)
- 然后再把“变进制数”转化成全排列也很容易
- 于是全排列和自然数的一一对应关系已经完整的建立起来了

排列的 Hash 函数

- 更一般的排列怎么处理?
- 回想一下 A(n,m) 的公式是怎么来的?
- 根据乘法原理，第一次有 n 个元素可选，第二次有 n-1 个元素可选……第 m 次有 n-m+1 个元素可选
- A(n,m) = n(n-1)(n-2)...(n-m+1)
- 第 i 次选取的时候有 n-i+1 种可能
- 把这个序号记为 a_{m-i+1}，可以知道 $0 \le a_{m-i+1} \le n-i$
- 即 $0 \le a_i \le n-m+i-1$，其中 i=1,2,...,m

Episode: 关于进位制 (2)

- 既然这样，如果让数的第一位是 n-m+1 进制，第二位是 n-m+2 进制，……，第 m 位是 n 进制的话可不可以?
- 当然可以
- 注意到

$$
\begin{aligned}
A(n, m) - 1 = {} & (n-m)A(n-m, 0) \\
& + (n-m+1)A(n-m+1, 1) \\
& + (n-m+2)A(n-m+2, 2) \\
& + \ldots \\
& + (n-1)A(n-1, m-1)
\end{aligned}
$$

Episode: 关于进位制 (2)

- 由此我们可以定义“ m-n 变进制数”:
- $k = a_m A(n-1, m-1) + a_{m-1} A(n-2, m-2)$
  $+ a_{m-2} A(n-3, m-3) + \ldots$
  $+ a_2 A(n-m+1, 1) + a_1 A(n-m, 0)$
- 其中 $0 \le a_i \le n-m+i-1$ ， $i=1,2,\ldots,m$

排列的 Hash 函数

- 再次回到主题
- 我们在生成排列的过程中很自然的生成了一个“ m-n 变进制数”：只要在生成的过程中维护一个线性表，记下每次的选择就可以了；亦可以相同的方式进行逆向转换
- 同时，“ m-n 变进制数”与自然数之间的互换也很容易
- 一般性的排列与自然数之间的一一对应也已经建立起来了

总结

- 应用： Hash 的用处很广泛，不仅仅局限在处理整数
- 方法：对现有方法的推广 (256 进制数，变进制数等)
- 启示：有时候仅仅关心时间复杂度是不够的， O(1) 的算法速度差距也会很大
- 问题：关于组合的 Hash 函数，我还没有想到好的方法，欢迎大家讨论

谢谢大家！

Thank you.