工程中的算法与数据结构

吕正华

2020.11

今天的话题

- 算法和算法分析
- Analytical Combinatorics
- 基数估计问题
- 哈希算法分析在多阶段聚合中的应用
- Pivotal Hash 和 Concrete Math
- 资料推荐

自我介绍

- VMware 资深软件工程师，开发 Greenplum 内核
- kainwen@gmail.com
- 个人主页: https://kainwen.com
- 2014 年毕业于电子系智能感知实验室, 工学硕士
- 2012 年第一次担任数据与算法助教

Table of Contents

算法分析什么?

- 算法的正确性分析
    - 要仔细研究证明，蕴含算法设计的思路
    - 学习一些形式化验证的工作和内容
    - 很多算法的证明技巧是归纳法: 循环不变范式
    - Software Foundations
- 算法的性能分析
    - 性能分析可以预测程序执行的时间
    - 性能分析可以比较不同算法的快慢
    - Analytical Combinatorics 的技术
    - Robert Sedgewick: Big O notation is harmful!

算法导论中的一个例子

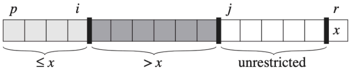


Figure 7.2 The four regions maintained by the procedure PARTITION on a subarray $A[p \,..\, r]$. The values in $A[p \,..\, i]$ are all less than or equal to x, the values in $A[i+1 \,..\, j-1]$ are all greater than x, and $A[r] = x$. The subarray $A[j \,..\, r-1]$ can take on any values.

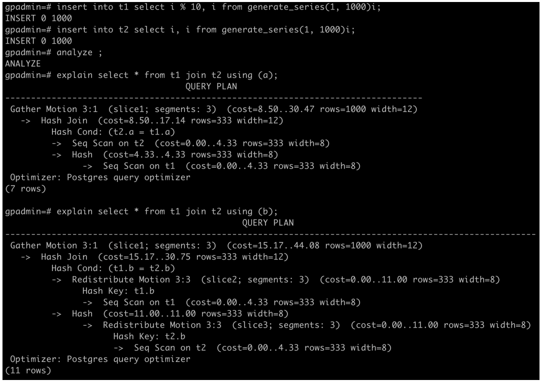
Initialization: Prior to the first iteration of the loop, $i = p - 1$ and $j = p$. Because no values lie between p and i and no values lie between $i+1$ and $j-1$, the first two conditions of the loop invariant are trivially satisfied. The assignment in line 1 satisfies the third condition.

Maintenance: As Figure 7.3 shows, we consider two cases, depending on the outcome of the test in line 4. Figure 7.3(a) shows what happens when $A[j] > x$; the only action in the loop is to increment j. After j is incremented, condition 2 holds for $A[j-1]$ and all other entries remain unchanged. Figure 7.3(b) shows what happens when $A[j] \leq x$; the loop increments i, swaps $A[i]$ and $A[j]$, and then increments j. Because of the swap, we now have that $A[i] \leq x$, and condition 1 is satisfied. Similarly, we also have that $A[j-1] > x$, since the item that was swapped into $A[j-1]$ is, by the loop invariant, greater than x.

Termination: At termination, $j = r$. Therefore, every entry in the array is in one of the three sets described by the invariant, and we have partitioned the values in the array into three sets: those less than or equal to x, those greater than x, and a singleton set containing x.

Figure: 算法导论证明快速排序划分正确性的技巧

SQL 查询计划代价的例子



```
gpadmin=# insert into t1 select i % 10, i from generate_series(1, 1000)i;
INSERT 0 1000
gpadmin=# insert into t2 select i, i from generate_series(1, 1000)i;
INSERT 0 1000
gpadmin=# analyze ;
ANALYZE
gpadmin=# explain select * from t1 join t2 using (a);
                                   QUERY PLAN
---------------------------------------------------------------------------------
 Gather Motion 3:1  (slice1; segments: 3)  (cost=8.50..30.47 rows=1000 width=12)
   ->  Hash Join  (cost=8.50..17.14 rows=333 width=12)
         Hash Cond: (t2.a = t1.a)
         ->  Seq Scan on t2  (cost=0.00..4.33 rows=333 width=8)
         ->  Hash  (cost=4.33..4.33 rows=333 width=8)
               ->  Seq Scan on t1  (cost=0.00..4.33 rows=333 width=8)
 Optimizer: Postgres query optimizer
(7 rows)

gpadmin=# explain select * from t1 join t2 using (b);
                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------
 Gather Motion 3:1  (slice1; segments: 3)  (cost=15.17..44.08 rows=1000 width=12)
   ->  Hash Join  (cost=15.17..30.75 rows=333 width=12)
         Hash Cond: (t1.b = t2.b)
         ->  Redistribute Motion 3:3  (slice2; segments: 3)  (cost=0.00..11.00 rows=333 width=8)
               Hash Key: t1.b
               ->  Seq Scan on t1  (cost=0.00..4.33 rows=333 width=8)
         ->  Hash  (cost=11.00..11.00 rows=333 width=8)
               ->  Redistribute Motion 3:3  (slice3; segments: 3)  (cost=0.00..11.00 rows=333 width=8)
                     Hash Key: t2.b
                     ->  Seq Scan on t2  (cost=0.00..4.33 rows=333 width=8)
 Optimizer: Postgres query optimizer
(11 rows)
```

Figure: Hash 表建在了不同的位置

Table of Contents

离散数学结构 Class

- $Class\mathcal{A}$ 是定义了 *size* 函数的集合
- 普通生成函数:$A(z) = \sum_e z^{size(e)} = \sum_N A_N z^N$
- 指数生成函数:$A(z) = \sum_e \frac{1}{N!} z^{size(e)} = \sum_N \frac{A_N}{N!} z^N$
- 这样一个数学结构的信息就被一个多项式编码了

例子：全排列

- 集合 $\mathcal{P}$ 是所有的全排列
- *size* 是全排列的长度
- $\mathcal{P} = \{1, 12, 21, 123, 132, 321, \dots\}$
- $P(z) = \sum_N \frac{N!}{N!} z^N = \frac{1}{1-z}$

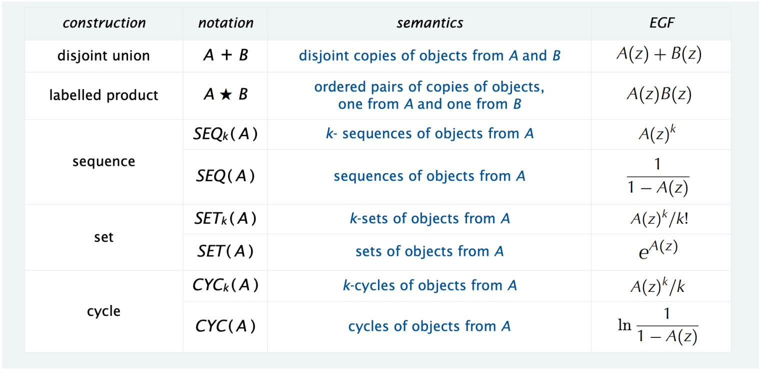
联想：信号与系统

- 离散时间信号处理: z 变换
- 研究问题的定义后，进一步研究各种性质: 时域移位、卷积等
- 序列的变换，数字域的滤波器的系统可以用延迟器和加法器实现
- Q: Analytical Combinatorics 研究的数学结构有没有类似性质？
- A: 有，Symbolic Method, 像搭积木和写程序

搭积木的例子：全排列和 Cycles

- $1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
- $2\ 5\ 4\ 1\ 8\ 7\ 6\ 3$
- $1 \to 2 \to 5 \to 8 \to 3 \to 4 \to 1, 6 \to 7 \to 6$
- 搭积木的方法：一个全排列就是一些圆排列的集合
- $\mathcal{P} = SET(CYC(\mathcal{I}))$
- $P(z) = e^{ln \frac{1}{1-z}} = \frac{1}{1-z}$
- 网传电子系挂科率很高的概率论考试第一题

Symbolic Method



construction	*notation*	*semantics*	*EGF*
disjoint union	$A + B$	disjoint copies of objects from A and B	$A(z) + B(z)$
labelled product	$A \star B$	ordered pairs of copies of objects, one from A and one from B	$A(z)B(z)$
sequence	$SEQ_k(A)$	k- sequences of objects from A	$A(z)^k$
	$SEQ(A)$	sequences of objects from A	$\frac{1}{1 - A(z)}$
set	$SET_k(A)$	k-sets of objects from A	$A(z)^k/k!$
	$SET(A)$	sets of objects from A	$e^{A(z)}$
cycle	$CYC_k(A)$	k-cycles of objects from A	$A(z)^k/k$
	$CYC(A)$	cycles of objects from A	$\ln \frac{1}{1 - A(z)}$

Figure: labelled 离散结构的变换总结 (图片引用自这里)

Table of Contents

Cardinality Estimate

- `select count(distinct a) from t`
- 在估计 Join 结果集大小和分析 HashTable 性能的时候极其重要
- Hyerloglog 算法分析
- Cardinality Estimate

Table of Contents

多阶段聚合

演示如何调优 TPCDS 性能的过程，并展示多阶段 agg。

- 搭建周期性运行的 Performance Pipeline
- 找到回归查询
- 阅读查询计划，分析是否由于查询计划引起
- 本地复现、调试

多阶段数学模型

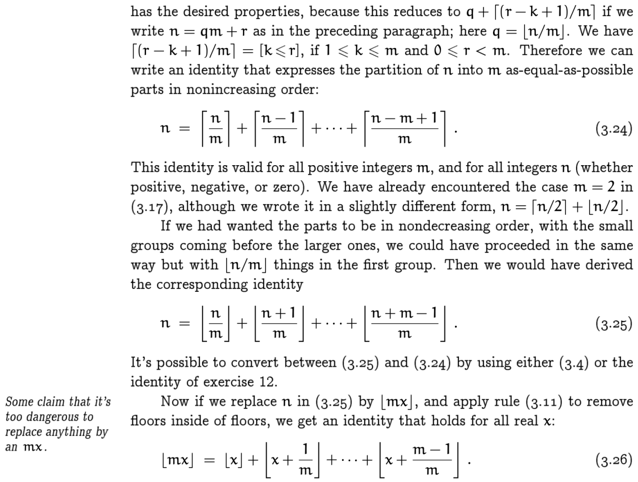
- 多阶段 agg 在第一个阶段可以消耗大量 tuple 的话，这样可以对抗过分倾斜的数据
- 多阶段 agg 的第一阶段有 spill 或者 streaming 的技术，当发生 spill 或者 streaming 时候，如果 group 数目太多，频繁 streaming，则第一阶段会产生过大开销
- 因此需要对多阶段 agg 的 spill 和 streaming 数量进行精准估计
- Coupon collector’s problem: sample without replacement
- Coupon collector’s problem：An infinite-series perspective

Table of Contents

一致性哈希

- 分布式系统中的一致性哈希：单调性，均匀性
- Q: 简单取模在什么时候是一致性哈希?
- Jump consistent Hash 算法：伪随机带来确定性，但可以用概率手段分析
- Heikki 老师的发现：Jump consistent hash performance
- 我的思考：
  - maglev 并不是好的算法，并不是单调的，构建算法复杂
  - 基于查表的最佳算法一定是：PivotalHash
  - 天下没有免费的午餐：Cache 命中问题

具体数学里的公式



has the desired properties, because this reduces to $q + \lceil (r-k+1)/m \rceil$ if we write $n = qm + r$ as in the preceding paragraph; here $q = \lfloor n/m \rfloor$. We have $\lceil (r-k+1)/m \rceil = [k \leqslant r]$, if $1 \leqslant k \leqslant m$ and $0 \leqslant r < m$. Therefore we can write an identity that expresses the partition of n into m as-equal-as-possible parts in nonincreasing order:

$$n = \left\lceil \frac{n}{m} \right\rceil + \left\lceil \frac{n-1}{m} \right\rceil + \cdots + \left\lceil \frac{n-m+1}{m} \right\rceil . \qquad (3.24)$$

This identity is valid for all positive integers m, and for all integers n (whether positive, negative, or zero). We have already encountered the case $m = 2$ in (3.17), although we wrote it in a slightly different form, $n = \lceil n/2 \rceil + \lfloor n/2 \rfloor$.

If we had wanted the parts to be in nondecreasing order, with the small groups coming before the larger ones, we could have proceeded in the same way but with $\lfloor n/m \rfloor$ things in the first group. Then we would have derived the corresponding identity

$$n = \left\lfloor \frac{n}{m} \right\rfloor + \left\lfloor \frac{n+1}{m} \right\rfloor + \cdots + \left\lfloor \frac{n+m-1}{m} \right\rfloor . \qquad (3.25)$$

It's possible to convert between (3.25) and (3.24) by using either (3.4) or the identity of exercise 12.

Some claim that it's too dangerous to replace anything by an mx.

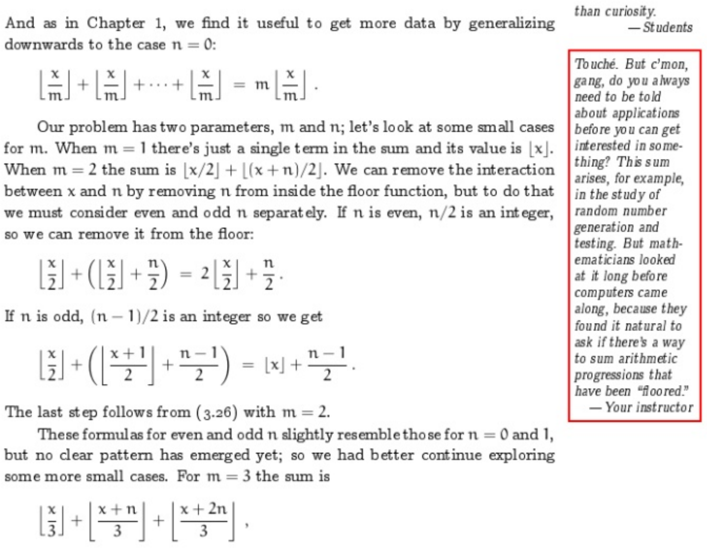
Now if we replace n in (3.25) by $\lfloor mx \rfloor$, and apply rule (3.11) to remove floors inside of floors, we get an identity that holds for all real x:

$$\lfloor mx \rfloor = \lfloor x \rfloor + \left\lfloor x + \frac{1}{m} \right\rfloor + \cdots + \left\lfloor x + \frac{m-1}{m} \right\rfloor . \qquad (3.26)$$

Figure: 具体数学里的公式

Table of Contents

具体数学里一个旁白



And as in Chapter 1, we find it useful to get more data by generalizing downwards to the case $n = 0$:

$$\left\lfloor\frac{x}{m}\right\rfloor + \left\lfloor\frac{x}{m}\right\rfloor + \cdots + \left\lfloor\frac{x}{m}\right\rfloor = m\left\lfloor\frac{x}{m}\right\rfloor .$$

Our problem has two parameters, m and n; let's look at some small cases for m. When $m = 1$ there's just a single term in the sum and its value is $\lfloor x \rfloor$. When $m = 2$ the sum is $\lfloor x/2 \rfloor + \lfloor (x+n)/2 \rfloor$. We can remove the interaction between x and n by removing n from inside the floor function, but to do that we must consider even and odd n separately. If n is even, $n/2$ is an integer, so we can remove it from the floor:

$$\left\lfloor\frac{x}{2}\right\rfloor + \left(\left\lfloor\frac{x}{2}\right\rfloor + \frac{n}{2}\right) = 2\left\lfloor\frac{x}{2}\right\rfloor + \frac{n}{2} .$$

If n is odd, $(n-1)/2$ is an integer so we get

$$\left\lfloor\frac{x}{2}\right\rfloor + \left(\left\lfloor\frac{x+1}{2}\right\rfloor + \frac{n-1}{2}\right) = \lfloor x \rfloor + \frac{n-1}{2} .$$

The last step follows from (3.26) with $m = 2$.

These formulas for even and odd n slightly resemble those for $n = 0$ and 1, but no clear pattern has emerged yet; so we had better continue exploring some more small cases. For $m = 3$ the sum is

$$\left\lfloor\frac{x}{3}\right\rfloor + \left\lfloor\frac{x+n}{3}\right\rfloor + \left\lfloor\frac{x+2n}{3}\right\rfloor ,$$

than curiosity.
— Students

Touché. But c'mon, gang, do you always need to be told about applications before you can get interested in something? This sum arises, for example, in the study of random number generation and testing. But mathematicians looked at it long before computers came along, because they found it natural to ask if there's a way to sum arithmetic progressions that have been "floored."
— Your instructor

Figure: 具体数学里的公式

资料推荐

- 书籍
  - Concrete Mathematics: A Foundation for Computer Science
  - C Interfaces and Implementations
  - Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne
- 课程
  - Prof. Robert Sedgewick's MOOCs
- 论文
- 电影：《知无涯者》

结束

谢谢！
Q&A