Strictly Fibonacci heap

许臻佳、杨卓林、曾凡高

April 8, 2016

Abstract

Strict Fibonacci heap [1]是一种时间复杂度优秀的可合并堆。其除了delete-min操作之外，所有的基本操作均可在O(1)的最坏复杂度内完成。唯一遗憾的是由于维护结构性质的繁琐，以及其难写易错的代码，不仅在现实中应用不广，算法的常数也使得其在竞争中几乎优势全无。抛却这一切，Strict Fibonacci heap通过结构上的精细构思，达到了目前所有的堆都难以企及的优秀复杂度。而这优秀的理论复杂度，也使其成为一种真正的理想意义上的数据结构。

Contents

1 结构

1.1 简述

- 整个堆形成一个树的结构,默认大根堆。
- 兄弟节点之间用双向链表连接。
- 特别注意的是，出于某种原因（之后会讲），堆中不能有相同元素（处理方法：元素以$pair < key, id >$存储）。

1.2 性质

为了维持整个堆操作复杂度的稳定，Tarjan通过一些方法，严格的维护整个堆的某些性质。不妨先令$R = 2\log n + 6$

- 对于一个active节点，它的第i右的active儿子，其rank值加loss值至少是i-1。
- Active roots的总数量最多是R+1。
- 总loss值最多为R+1。
- 根的儿子数最多为R+3，对于一个非根节点passive或者loss值为正的active节点，假设它在Q中的位置是p，其儿子数最多为2lg(n-p)+9. 对于一个loss值为0 的active节点，其儿子数最多为2log(n-p)+10。
- 所有点的rank都小于等于R。

1.3 节点

- 节点分为两种类型active和passive。
- 一个active节点的父亲如果是passive节点，那么它又被称为active root。
- 如果一个passive节点，他的儿子节点都是passive的，那么他被定义为linkable。
- 每个active node存2个值。rank：它有几个active儿子节点。loss: 和斐波那契堆一样，每失去一个儿子loss值加一（active root的loss一定为0）。

1.4 存储

- 整个堆维护了一个Q队列，存除了根以外的所有元素，具体作用后面讲。
- 对于每一个节点，他的active儿子节点总是放在左边，passive儿子节点总是放在右边。
- 对于root，顺序严格保证是：active，passive，linkable。

2 基本操作

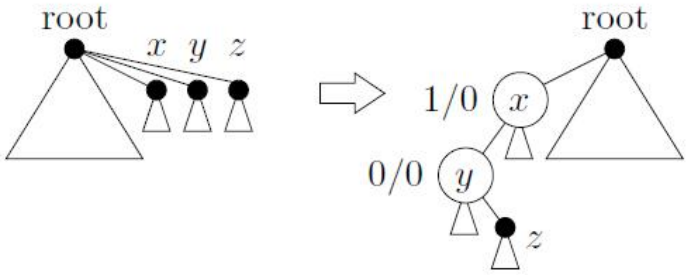
在插入、删除等一系列操作后之前将的一系列性质会不满足，因此需要以下操作维护堆得性质。

2.1 A操作

目标 减少active root的数量。

条件 x,y都是active root,而且他们的rank值相同。

具体操作 首先比较x,y的key值，不妨设$key_x < key_y$。y接到x上，x的rank加一，如果x的儿子最右边的节点是passive 的，直接接到堆顶元素的儿子链表中。

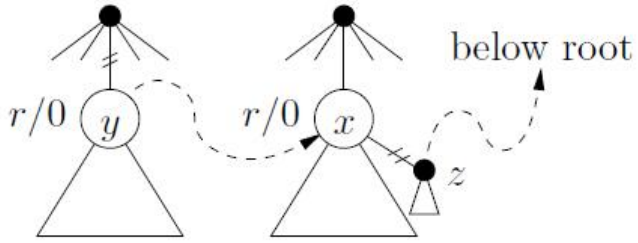


2.2 R操作

目标 减少根的儿子个数。

条件 x,y,z分别是根的最右边三个linkable儿子。

具体操作 首先比较x,y,z的key值，不妨设$key_x < key_y < key_z$。把x,y变成active,然后y连到x，z连到y，把x节点放到根的儿子链表的最左边。x,y的loss值清零，$rank_x = 1, rank_y = 0$,x成为新的active root。

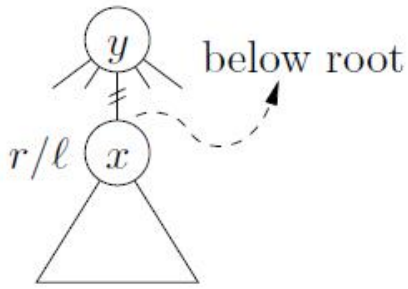


2.3 L1操作

目标 减少总loss。

条件 存在一个active节点x，x的loss大于等于2。

具体操作 x直接连到根，成为新的active root, loss值清零。

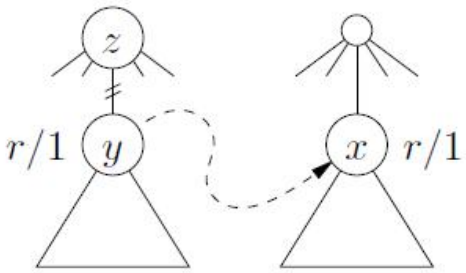


2.4 L2操作

目标 减少总loss。

条件 有两个active节点x,y，他们rank值相同，而且loss值都为1。

具体操作 首先比较x,y的key值，不妨设$key_x < key_y$。y连到x，$rank_x$加一，x,y的loss清零。y的父亲的儿子数和rank值都减一，如果z不是active root, z的loss减一。



2.5 操作比较

Table 1: Effect of the different transformations

	Root degree	Total loss	Active roots	Key comparisons
Active root reduction (A)	$\leq +1$	0	-1	$+1$
Root degree reduction (R)	-2	0	$+1$	$+3$
Loss reduction	$\leq +1$	≤ -1	$\leq +1$	$\leq +1$
– one-node	$+1$	≤ -1	$+1$	0
– two-node	0	-1	0	$+1$
(A) + (R)	≤ -1	0	0	$+4$
2×(A) + (R)	≤ 0	0	-1	$+5$
3×(A) + 2×(R)	≤ -1	0	-1	$+9$

Table 2: The changes caused by the different heap operations

	Root degree	Total loss	Active roots
decrease-key	$\leq 1+1+6-8$	$\leq 1-1+0+0$	$\leq 1+1-6+4$
meld	$\leq 1+0+1-2$	$\leq 0+0+0+0$	$\leq 0+0-1+1$
delete-min	$\leq (2\lg n+12+4)+1$	$\leq 0-1$	$\leq R+1$

这些操作各有利弊，但经过一系列组合，就会有非常好的效果。

3 堆操作

3.1 decrease key

- 如果减小后$rank_x < rank_root$，则swap(x,root)。
- 将x提为root的直接儿子。
- 做6次A操作，4次R操作。

3.2 delete min

- 先从root的儿子中找到key最小的替代品x。
- 如果x是active的，那么把x变成passive，所有x的active儿子变成active root。
- 把其他儿子接到x的儿子链表上去，并注意维护这个链表的active先，linkable后的顺序。
- 把x从当前堆的Q中删除，把原root删掉。
- 做两次：取出Q的队头y，如果y的最右边两个儿子是passive 的，连到x。

- 然后不断做L操作(先L1，再L2)。
- 不断轮流做A操作和R操作直到不能做为止。

3.3 merge

- 不妨设$size_x < size_y$。
- 先把x堆的所有元素全部设为passive（之后会讲如何O（1）完成）。
- 假设堆顶元素较小的是u，另一个是v。
- v连向u, 新的Q为Qx+v+Qy，这里+指顺次连接。
- 不断轮流做A操作和R操作直到不能做为止。

4 实现技巧

我们建立一个rank list，代表一个rank值宏节点链表，链表从表头到开始以0, 1, 2, 3编号，表示对应rank值宏节点。

为了保证四种操作的执行复杂度，我们把所有可能进行操纵的节点用一个链表维护起来，并从左至右分为四个部分，总称为fix list，其中part1, part2存放那些active roots, part3, part4存放那些loss不为0的active节点。

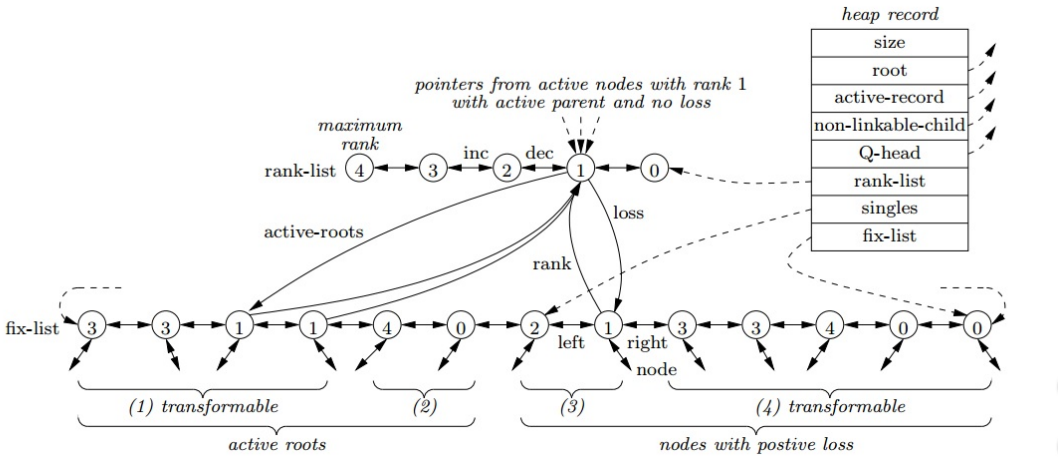
可以发现part1,2与part3,4不会有交(loss不为0不可能是active roots)

- part1: 存放那些rank值不落单的active roots, 保证rank值相同的在链表中一定是连续一段。其中每个节点都指向rank list对应rank的节点，而对应rank的节点只指向这连续一段rank相同的节点中最左边的一个。
- part2: 存放那些rank值落单的active roots，每个节点都指向rank list对应rank的节点，对应rank的节点也会指向它。可以发现对于rank list的一个节点，它指向part1,part2的指针只会有一个(要么一段连续，要么落单)，这个指针称为active root指针。
- part3: 存放那些rank值落单，loss值还为1的节点，每个节点都指向rank list对应rank的节点，对应rank的节点也会指向它。
- part4：存放那些rank值不落单的active节点，或者rank值落单但是loss值大于等于2的节点。仍然保证rank值相同的在链表中一定是连续一段。每个节点都指向rank list对应rank的节点，而对应rank 的节点只指向这连续一段rank相同的节点最右边的一个。

同样可以发现，对于rank list的一个节点，它指向part3, part4的指针只会有一个，这个指针称为loss指针。

对于之前堆操作的各种变化，可能有新的active root节点加入，也可能会有删除，loss值的增减也会影响到part3, part4的链表的变化和维护。很多时候，这个链表需要很精细的维护。而之前的A操作和L操作，便可以变为直接取定链表两端的节点，进行操作。

对于一个节点，设立fix和rank指针，如果它是passive节点，那么其fix指针和rank指针均为null, 如果其不属于可能进行操作的节点，它会连向它对应的rank list里的节点，fix指针为null, 否则连向fix list里的节点，rank指针为null。



References

[1] Gerth Stolting Brodal,George Lagogiannis,and Robert E. Tarjan. **" Strict Fibonacci Heaps "** *Proceedings of the 44th symposium on Theory of Computing - STOC '12. p. 1177*