势能分析方法

1 概述

势能分析的关键是找到合适的势能函数。这也是很多势能分析问题的难点所在。我们可以把势能方法看做是记账方法的延伸。记账方法中使用 credit 记录均摊成本和实际成本之间的差异，势能方法则使用势能函数记录这个差异。函数的灵活性使得势能方法应用场合比记账方法更广。

在势能分析中，我们一般使用以下符号记录操作中数据结构和势能的变化。

D_0：初始数据结构；

D_i：执行第 i 步操作后的数据结构；

势能函数：$\Phi : D_i \to R$，反应操作后数据结构的势能；

c_i: 将 $D_i - 1$ 变换到 D_i 的实际成本；

$\hat{c}_i$: 将 $D_i - 1$ 变换到 D_i 的均摊成本，我们有：$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$；所以，总均摊成本为：$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$

在这个总均摊成本的计算中，如果我们能够保证 $\Phi(D_n) - \Phi(D_0) \geq 0$，我们就能保证总均摊成本是总实际成本的一个上限。势能方法的关键是找到合适的势能函数。一般一个好的势能函数设计总能使 $\Phi(D_0)$ 是这个序列中的最小值。实际使用中，我们一般定义 $\Phi(D_0) = 0$。这样，只要证明上式中 $\Phi(D_i) \geq 0$，就可以证明均摊成本是实际成本的一个上限。

2 Splay Trees

我们注意到 Splay 操作中，会将所访问的节点翻转到根节点。前面的样例中我们也注意到，翻转代价高的操作往往更大程度上的降低了树高（比如前面例子中，将节点 1 从叶节点位置翻转到根节点，大致将树高缩减为原来的一半）。所以我们考虑一个跟节点高度相关的（或类似的）势能函数。

在 Splay 操作中，几乎每个节点的高度都会改变，哪怕该节点为根节点的子树没有任何变化。如果我们直接使用节点高度作为势能函数，后续的数学计算与推导会变得非常复杂。所以我们希望有一个可以简化后续数学推导的势能函数。

一个可用的势能函数是树中所有节点的 rank 之和：

$$\Phi(T) = \sum_{i=1}^{n} \log S(i)$$

其中 $S(i)$ 指的是子树 i 中的节点数（包括节点 i）。我们用 $R(i)$ 表示节点 i 的 rank，$R(i) = \log S(i)$。

势能函数 $\Phi(T) = \sum_{i \in T} R(i)$。对于我们刚才学过的 3 种 splay 操作：zig、zig-zig、zig-zag，我们使用 R_2 表示操作后的势能，R_1 表示操作前势能。首先是 zig 操作，做了个单旋，成本为 1；从 PPT 上的 zig 操作示意图中可以看出，在整个操作中只有 X 和 P 的 rank 值有变化。所以我们有：

$$\hat{c}_i = 1 + R_2(X) - R_1(X) + R_2(P) - R_1(P)$$

由于节点 P 由根节点变为非根节点，我们有 $R_2(P) - R_1(P) \leq 0$，因此 $\hat{c}_i \leq 1 + R_2(X) - R_1(X)$。既然 $R_2(X) - R_1(X) \geq 0$，我们有 $\hat{c_{i-zig}} \leq 1 + 3(R_2(X) - R_1(X))$。

对于 zig-zag 操作，实际成本是两次旋转，为 2。因此，

$$\hat{c_i} = 2 + R_2(X) - R_1(X) + R_2(P) - R_1(P) + R_2(G) - R_1(G)$$

从左图中看到，操作前 G 是根节点，操作后 X 是根节点，他们的 rank 相同，因此，$\hat{c_i} = 2 - R_1(X) + R_2(P) - R_1(P) + R_2(G)$。同时，操作后我们可以看到 $S_2(P) + S_2(G) \leq S_2(X)$，由书上的 lemma 11.4，我们可以得出：$R_2(P) + R_2(G) \leq R_2(x) - 2$。故我们有：

$$\hat{c_i} \leq 2(R_2(X) - R_1(X))$$

zig-zig 的推导就留给同学们自己完成。

最后，给定一个伸展树上访问节点 X 的一系列 M 个 splay 操作（zig、zigzig、zigzag），其中最多只会有 1 个 zig。把他们都给加起来后，可得：

$$\sum_{i=1}^{M} \hat{c_i} \leq 3(R_M(X) - R_1(X)) + 1$$

注意到结束操作后，X 是新的根节点，所以 $R_M(X) = R_1(T)$，所以我们有

$$\sum_{i=1}^{n} \hat{c_i} \leq 3(R_1(T) - R_1(X)) + 1$$

很明显，均摊成本是 $O(\log n)$ 级别的。

3 Skew Heaps

斜堆合并的总步骤数就是两个堆的右路径节点数之和；但是斜堆右路径长度不受限，最坏情况可到 $O(n)$，所以我们需要通过均摊分析来确定 M 个操作序列的复杂度。

均摊分析的势能函数要反映斜堆合并操作的效果。考虑到合并成本为右路径总节点数，直接想法是通过右路径节点数目来定义势能函数（斜堆合并由空堆开始，因此该函数刚好从 0 开始且非负）。但该函数的问题是单调递增的，不能反映合并过程中的势能变化。

我们选择斜堆中的“重节点（heavy nodes）”数作为势能函数。如果一个节点的右子树总结点数占该节点为根的子树总结点数达一半及以上为重节点，反之为轻节点。注意到斜堆合并是由空堆开始，该势能函数满足：$\Phi(D_0) = 0$；随着合并的进展，中间任何步骤都有 $\Phi(D_0) \geq 0$。

假设要合并的两斜堆 H_1 和 H_2 右路径上的重节点数分别为 h_1 和 h_2。俩斜堆中其他重节点数目为 $\mathbf{h}$。则有：

$$\Phi(D_0) = h_1 + h_2 + \mathbf{h}$$

注意在合并过程中，除了右路径节点，其他节点不会发生轻重转变，因为它们的左右子树都没有变化。

因为合并成本为右路径总节点数，如果 H_1 和 H_2 右路径上的轻节点数分别为 l_1 和 l_2，则实际合并的总代价为：

$$\sum_{i=1}^{n} c_i = l_1 + l_2 + h_1 + h_2$$

合并后，只有右路径节点会出现轻重节点的改变：（1）右路径重节点合并后会肯定变成轻节点（左右调换后，左轻右重自然变成了左重右轻）；（2）右路径轻节点合并后有可能变成重节点。所以合并后重节点数目最多的情况就是所有右路径的轻节点都变重节点的情况。所以：$\Phi(D_N) \leq l_1 + l_2 + \mathbf{h}$。所以：$(\Phi(D_N) - \Phi(D_0)) \leq l_1 + l_2 - h_1 - h_2$。所以：

$$\sum_{i=1}^{n} \hat{c_i} = \sum_{i=1}^{n} c_i + (\Phi(D_N) - \Phi(D_0)) \leq 2(l_1 + l_2)$$

l_1 和 l_2 为要合并的俩斜堆右路径上轻节点数量。轻节点意味着节点的子树“左重右轻”（和前面讲过的左倾堆类似），因此 $l_1 + l_2 \leq \log N_1 + \log N_2$（$N_1$ 和 N_2 分别为俩斜堆的节点数）。

4 Binomial Queues

便宜的操作生成树，昂贵的操作移除树。考察操作成本与树数目变化之间的关系：要得到一颗 B_{c-1}，从 B_{c-2} 到 B_0 都得消失。需要 $c-1$ 的 linking 成本；加上插入成本 1，得到一颗 B_{c-1} 的成本为 c，消失的树数目为 $c-1$ 棵，所以该操作后，树的总数目减少了 $c-2$ 棵，即增加了 $2-c$ 棵。

势能函数：$\Phi_i=$ 第 i 步操作后，二项队列中的树的数目；

均摊成本：$\hat{c}_i = c_i + (\Phi_i - \Phi_{i-1}) = c + (2-c) = 2$

而且满足 $\Phi_0 = 0$，任何其他步骤都有 $\Phi_i \geq 0$

简直是我们碰到过的最简单的均摊分析。