Interactive Memory Management 解题报告

杭州学军中学 金策

1 试题来源

Andrew Stankevich Contest 46, XIV Open Cup Onsite
Petrozavodsk Summer Training Camp, August 31, 2014

2 试题大意

这是一道交互题。

你有一个长度为n的内存条，可以看做一个数组$M[1..n]$。一个大小为s的块需要占据s个连续格子。多个块不能同时共用某个格子。

你需要实现分配内存、释放内存的请求。中途你可以将已分配的块挪动位置。

每次让你分配内存时，checker会告诉你一个参数s，表示需要分配的内存块的大小，其中$s \in \{1, 2, 3\}$。你可以先执行不超过2次移动操作，然后选择一个包含s个连续空位的地方，塞入这个大小为s的块。你需要把塞入的位置告诉checker。

每次让你释放内存时，checker也会给你一个参数s，表示第s次分配操作时分配的内存块被删除。之后，你允许做不超过2次移动操作。

移动操作是指你自己选择将当前某个内存块移动到另一个有空位的地方，而且新位置和旧位置包含的格子也不允许有重叠。做移动操作时你也需要将旧位置和新位置告诉checker。

数据保证任意时刻至多只有$n - 1$个格子被分配。你需要实现一个策略，使得总是能够完成分配的请求。

3 数据规模

$n \leq 100000$，请求的数量不超过30000。

4 算法介绍

这个题每次分配或删除的块大小都不超过3，不是很大；而每次允许的移动操作数也不多。所以大概思路就是让当前的内存块形成一个比较好弄的形态，使得每次加、删之后都可以通过一点点微调保持这个形态。

如果只有两种块大小的时候（即$s \in \{a, b\}$），做法还是比较简单的。我们可以让a的块全部挤在左边，b的块全部挤在右边。这样每次加入的时候就直接加在顶端，删除的时候就把顶端的块填过来，于是可以保证每时每刻都能满足。

现在有3种块$1, 2, 3$。也可以仿照这个思路，把最大的3放在右边，左边放1和2。

3的那边的处理方法和刚才完全一样，所以现在关键是如何处理$1, 2$。

根据题目描述，我们每时每刻都有一个格子可以空着，不需要全部利用。另外又注意到$2 = 1 + 1$。所以可以尝试这样子维护：从左边开始每连续两个格子作为一个小段，并规定一个小段内要么填入一个2，要么填入若干个1，不允许有某个2跨过两个小段。另外我们强制要求至多只有一个小段是半满的（即恰好只填入了一个1），且所有非空小段都是挤在左边（即中间不可以出现全空小段），所以至多空出了一个格子。

接下来看一下怎么维护，分几种情况来研究。

4.1 分配请求

首先来看插入操作。

4.1.1 情况一

如果当前没有半满的小段，那么我们在顶端新开辟一个小段，新插入的格子就放在这里面。如果插入的是1则出现一个半满小段，如果是2则什么也没有发生。

4.1.2 情况二

如果当前有一个半满的小段，而待插入的块是1，那么正好可以填进这个洞里面；如果待插入的是2，那么也可以在顶端开一个新的小段填进去，这样当前还是只有一个半满小段。

4.2 释放请求

接着来看稍微麻烦一些的删除操作。

4.2.1 情况一

当前没有半满的小段。

那么删除一个1会导致出现一个半满小段，然后什么也不用做就行了。

要删除一个2的话，如果这个2刚好在顶端那么也没有关系，如果在中间的话，就会出现一个全空小段，这是不行的，所以要把顶端的小段内的东西补进来，这样至多进行两次移动操作。

4.2.2 情况二

当前有一个半满的小段。

如果要删除一个2，有可能会在中间多出一个全空小段。这时我们看一下顶端的小段的情况，如果顶端是全满的，那么就把它全部填过来；如果不是，那么它就是那个唯一的半满小段，所以把那个1填过来就行了，于是此时依然只有一个半满小段。

现在考虑要删除的是一个1的情况。我们仍然要结合顶端小段进行考虑，如果顶端是半满的，那么把它填过来就好（如果删的不是它自己的话）。所以现在假设顶端是全满的，那么删除一个1可能会导致半满小段变成一个全空小段，或者新增一个半满小段。如果变出一个全空小段，那么把顶端搬过来就行。如果新增一个半满小段，那么现在有两个半满小段，我们看一下顶端小段的具体构成：如果是两个1，那么直接分别填进来就行；如果是一个2，我们可以把两个半满小段的1移到一起，变成一个全满小段和一个全空小段，再把顶上的2移进那个全空小段。

于是这道题目就得到了解决。

5 总结

这个交互题的大致思路是很明确的，就是尝试去维护一个形态（类似于维护一个数据结构，使得每次插入删除后进行比较小的微调可以保持它的性质）。如果找到了一个合适的形态，那么剩下的工作就是比较简单的分类讨论了，但是具体什么样的形态可以维护并不是非常显然的，需要一些灵感以及几次尝试、失败的过程才能发现。