性能优化 实验报告

16307130006 陈幸豪

1. 实验原理

现在有一块连续内存堆：

口口口口口口口口口口口口口口口口口口口口口口口口口口口口口……

我们要自己在这块内存上实现malloc,free和realloc函数，并获得最优性能。此外，程序刚开始运行时，可用init函数做初始化。

* 1. malloc(size)

由于我们假想的用户使用malloc函数时，要求的是一块**连续的**内存。例如用户要求6单位空间，且当前内存占用情况如下（黑色表示被占用）：

p0 p1 p2 p3

口口口■■■■■■口口口口口■■■■■口口口口口口■■口口

则malloc的唯一选择是返回p2。同时，malloc不能自行把已占用的内存移位，因为用户（在不知情的情况下）还会用到原先申请过的内存的地址。

* 1. free(\*ptr)

告知内存分配器，ptr所指的内存不需要再用到了。但用户只给出这段内存的首地址，而不会给出这段内存的长度。

* 1. realloc(\*ptr,size)

要求废弃ptr所指的内存空间，并重新找一块大小为size的内存空间。用户同样不会给出ptr所指内存的长度。可以认为这个操作类似于先free再malloc。但重新malloc后的内存块应保持原先的内容，即我们需要用memcpy把原先内容放入新内存块。

* 1. malloclab的实质

迅速查找出一块能满足malloc要求的内存！当然我们可以每次都在最右端（地址最大的一端）新开内存，但这样的空间效率极低。应当试着在之前的空闲内存中寻找。

在具体实现中，我们要完成以下操作：

* + 1. 增加内存堆的空间
    2. 输入一块被分配过的内存首地址，求这块内存大小
    3. 迅速找一块足够大的连续内存空间（为此要维护一个数据结构）
  1. 红黑树

教材上喜欢用链表来查找内存块，然而只用链表的速度并不是最快的。可以用红黑树来查找。红黑树是涂了颜色的二叉树，例子如下：

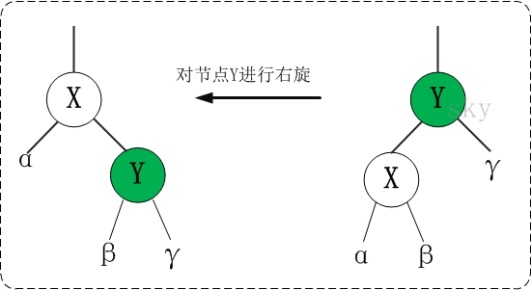
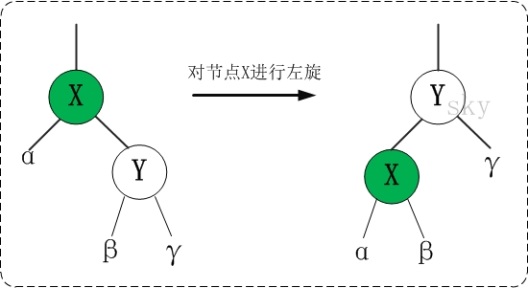


红黑树有以下性质：

* + 1. 每个节点或者是黑色，或者是红色。
    2. 根节点是黑色。
    3. 每个叶子节点是黑色。 [这里叶子节点，是指为空的叶子节点！]
    4. 如果一个节点是红色的，则它的子节点必须是黑色的。
    5. 从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点。
    6. （在malloclab中）是二叉平衡查找树

在malloclab中，主要会对红黑树做以下操作：

1. 左旋和右旋



1. 插入和删除节点

把红黑树当成二叉平衡查找树来插入。新插入的节点先标记为红色。插入和删除节点后的红黑树几乎不可能还是红黑树。因此用旋转操作来维持红黑树性质。

1. 重新涂色

同样是为了维持红黑树的性质。

1. 查找

类似于平衡二叉搜索树的查找。左子树的所有数小于等于左孩子的数。可以证明旋转操作并不会改变二叉平衡查找树的性质。

红黑树的详细操作可参见<https://www.cnblogs.com/skywang12345/p/3245399.html>。涂色的目的是保持二叉平衡查找树的高效率。

1. 主要策略
   1. malloc

首先去红黑树查找一块大于等于所需空间的最小空间。如果没找到（所有空间都小于所需空间），则新开空间（升高堆顶）。

* 1. free

如果被free的块前后也是空着的，则应当与前后块合并。被free的块不急于放入红黑树，而是放在临时链表List（网上大佬们又叫blob）里。当需要malloc时，才把List里所有空闲记录插入红黑树。

* 1. realloc

虽然可以写成先free再malloc（这样对节约空间有一定好处），但这样会导致大量的memcpy，消耗时间。总的来说，单纯的先free再malloc，最后memcpy是不合算的。可以略做修改使得realloc后的首地址尽量不变。当所需size缩小时，直接返回原始首地址。当所需size增大，但当前块之后（更高地址方向）有足够多的空闲块，则仍然返回原始首地址。