数据结构: Homework 2

张霄　2019030045

October 22, 2023

Problem 1

当 $n < 2^k$ 时，作用于初始为 0 的计数器 n 次，对于 $i > \lfloor \log n \rfloor$ 的位来说始终没有影响，但是对 $i \leq \lfloor \log n \rfloor$ 的位置来说，$A[0]$ 会变化 n 次，$A[1]$ 会变化 $\lfloor n/2 \rfloor$ 次，以此类推，$A[i]$ 会变化 $\lfloor n/2^i \rfloor$ 次，因此总的代价满足：

$$\sum_{i=0}^{\lfloor \log n \rfloor} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

所以从 0 开始调用 n 次 INCREMENT 操作的代价是 $\mathcal{O}(n)$ 的，那么平均代价就是：

$$\mathcal{O}(n)/n = \mathcal{O}(1)$$

当 $n \geq 2^k$ 时，所有 k 都会被占满，$A[0]$ 会变化 2^k 次，$A[1]$ 会变化 2^{k-1} 次，以此类推，$A[k]$ 会变化 1 次，因此总的代价满足：

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1 \leq 2n - 1 = \mathcal{O}(n)$$

平均代价为：

$$\mathcal{O}(n)/n = \mathcal{O}(1)$$

Problem 4

先对这 n 个长度进行排序，这个过程需要 $\mathcal{O}(n \log n)$ 复杂度。之后想要计算个数仍旧需要遍历，可以使用二分查找的方法。假设我们排序的结果是从小到大，存在数组 A 中 (下标从 0 到 $n-1$)，那么用一个 for 循环，从下标为 $n-1$ 的木棒开始到下标为 2 的木棒截止。假设我们现在循环到了下标为 i 的木棒，$left$ 指向 index 为 0 的位置，$right$ 指向 index 为 $i-1$ 的位置。接着判断条件 $A[left] + A[right] > A[i]$，如果满足条件那么就给我们的结果加上 $right - left$，同时让 $right$ 向左移一位；如果不满足条件我们就让 $left$ 向右移一位，直到 $left \geq right$ 为止。

Algorithm 1 Binary Search

A[n] ▷ Put n numbers in an array A
$Arrays.sort(A)$ ▷ Sort array A from small to large
$nums = 0$ ▷ The number of triangles that can be formed
for i $from$ $n-1$ to 2 **do**
　$left = 0, right = i - 1$
　while $left < right$ **do**
　　if $A[left] + A[right] > A[i]$ **then**

```
            nums += right − left
            right = right − 1
        else
            left = left + 1
        end if
    end while
end for
return nums
```

最终得到的 $nums$ 就是总的三角形数，该部分复杂度为 $\mathcal{O}(n^2)$。总的复杂度为

$$\mathcal{O}(n\log n)+\mathcal{O}(n^2)=\mathcal{O}(n^2)$$

Problem 5

容量递增策略 T* oldElem = _elem; _elem = new T[_capacity += INCREMENT]
最坏情况下在初始容量为 0 的空向量中连续插入 $n = m * I >> 2$ 个元素，而无删除操作，因此在第 $1, I+1, 2I+1, \cdots$ 次插入时需要扩容。每次扩容复制原向量的时间成本依次为 $\mathcal{O}(0), \mathcal{O}(I), \mathcal{O}(2I), \cdots, \mathcal{O}((m-1)I)$
累计扩容时间为

$$\mathcal{O}(0)+\mathcal{O}(I)+\cdots+\mathcal{O}((m-1)I)=\mathcal{O}(n^2)$$

分摊扩容时间为

$$\mathcal{O}(n^2)/n=\mathcal{O}(n)$$

空间利用率考虑向量实际规模与其内部数组容量的比值，即装填因子)，显然当 n 足够大的时候，装填因子趋近于 100%。

容量倍增策略 T* oldElem = _elem; _elem = new T[_capacity <<= 1];
最坏情况在初始容量为 1 的满向量中，连续插入 $n = 2^m >> 2$ 个元素而无删除操作，在第 $1, 2, 4, 8, 16, \cdots$ 次插入时需要扩容。每次扩容的时间成本为 $1, 2, 4, 8, 16, \cdots, 2^{m-1}, 2^m = n$
累计扩容时间为

$$\mathcal{O}(1)+\mathcal{O}(2)+\mathcal{O}(4)+\cdots+\mathcal{O}(2^{m-1})+\mathcal{O}(2^m)=\mathcal{O}(2^{m+1}-1)=\mathcal{O}(n)$$

分摊扩容时间为

$$\mathcal{O}(n)/n=\mathcal{O}(1)$$

空间利用率大于 50%

综上可以看出，容量倍增的时间成本较容量递增更小，但空间利用率更大，可以认为容量倍增是是在空间上适当牺牲换取时间上的巨大收益。