问题求解（二）作业（第十周）

161180162 许致明

2018 年 5 月 13 日

MA 第二章

2.6

对于 PUSH 操作，操作前 $|elemes| < max$，则操作后 $|elemes| \leq max$ 满足不变式 3；

操作后，当前的时间戳 c' 满足 $c' > c$，其中 c 为栈中元素的时间戳，满足不变式 2；

进行 PUSH 操作后，加入了一个元素 a，且当前时间戳增大，故满足不变式 1。

TC 第十章

10.1-4

```
ENQUEUE(Q, x)
1  if Q.head == Q.tail + 1, or
   Q.head == 1 and Q.tail = Q.length
2      error "overflow"
3  Q[Q.tail] = x
4  if Q.tail == Q.length
5      Q.tail = 1
6  else
7      Q.tail = Q.head + 1
```

```
DEQUEUE(Q, x)
1  if Q.tail == Q.head
2      error "underflow"
3  x = Q[Q.tail]
4  if Q.head == Q.length
5      Q.head = 1
6  else
7      Q.head = Q.head + 1
8  return x
```

10.1-5

```
HEAD-ENQUEUE(Q, x)
1  Q.[Q.head] = x
2  if Q.head == 1
3      Q.head = Q.length
4  else
5      Q.head = Q.head − 1
```

```
TAIL-ENQUEUE(Q, x)
1  Q.[Q.tail] = x
2  if Q.tail == Q.length
3      Q.tail = 1
4  else
5      Q.tail = Q.tail + 1
```

```
HEAD-DEQUEUE(Q, x)
1  x = Q.[Q.head]
2  if Q.head == Q.length
3      Q.head = 1
4  else
5      Q.head = Q.head + 1
6  return x
```

```
TAIL-DEQUEUE(Q, x)
1  x = Q.[Q.tail]
2  if Q.tail == 1
3      Q.tail = Q.length
4  else
5      Q.tail = Q.tail − 1
6  return x
```

10.1-6

将进队的操作转化为压栈（假定存入栈 A），出队时，先将此元素前（在栈的上面）的所有元素弹出，放入另一个栈中（假定为栈 B），再将此元素弹出。最后把 B 中的元素再按顺序全部出栈压入 A 中。此种实现入队操作复杂度为 $O(1)$，出队操作复杂度为 $O(n)$，n 为队列中所有元素的个数。

10.2-1

```
INSERT(L, x) // O(1)
1  x.next = L.head
2  L.head = x
```

```
DELETE(L, x) // 至少为线性时间
1  p = L.head
2  while p ≠ x and p ≠ NULL
3      p = p.next
4  if p == NULL
5      error "No such element x"
6  if p.prev ≠ NULL
7      p.prev.next = p.next
8  else
9      L.head = p.next
10 if p.next ≠ NULL
11     p.next.prev = p.prev
12 return p
```

10.2-2

```
PUSH(L, x)
1  x.next = L.head
2  L.head = x
```

```
POP(L, x)
1  x = L.head
2  L.head = x.next
3  return x
```

10.2-3

加入哨兵：$L.head \to L.\text{NULL}$（哨兵）$\to L.end$

```
DEQUEUE(L, x)
1  x = L.head
2  L.head = x.next
3  return x
```

```
ENQUEUE(L, x)
1  x.next = L.NULL.next
2  L.NULL.next = x
```

10.2-6

设 L_1 为包含了 S_1 中所有元素的双向链表，L_2 为包含了 S_2 中所有元素的双向链表。

```
UNION(L1, L2)
1  L1.NULL.prev.next = L2.NULL.next
2  L2.NULL.next.prev = L1.NULL.prev
3  L1.NULL.prev = L2.NULL.prev
4  L2.NULL.prev.next = L1.NULL.prev
5
```

10.3-4

m 指示已经分配的元素个数。

```
ALLOCATE-OBJECT(A)
1  if m == A.length
2      error "out of space"
3  else
4      m = m + 1
5      x = m
6  return x
```

```
FREE-OBJECT(A, x)
1  if m == 0
2      error "underflow"
3  elseif x == m
4      m = m − 1
5  else
6      SWAP(x, m)
7      m = m-1
```

10.3-5

```
COMPACTIFY-LIST(L, F)
 1  p1 = 1, p2 = A.length
 2  while p1 < p2
 3      while key[p1] ≠ NULL
 4          p1 = p1 + 1
 5      while key[p2] == NULL
 6          p2 = p2 − 1
 7      if p1 < p2
 8          SWAP(key[p1], key[p2])
 9          for i = 1 to A.length
10              if next[i] == p2
11                  next[i] == p1
12              if prev[i] == p2
13                  prev[i] == p1
14              if next[i] == p1 and key[i] == NULL
15                  next[i] == p2
```

使用循环不变量：p_1 前存放的都是数字，p_2 后全部为空，可以证明算法的正确性。

10.4-2

```
PRINT-TREE(T)
1  if T ≠ NULL
2      print T.key
3      PRINT-TREE(T.left)
4      PRINT-TREE(T.right)
```

10.4-3

```
PRINT-TREE-STACK(T)
1  PUSH(S, T)
2  while S ≠ NULL
3      tmp = POP(S)
4      print tmp.key
5      if tmp.left ≠ NULL
6          PUSH(S, tmp.left)
7      if tmp.right ≠ NULL
8          PUSH(S, tmp.right)
```

10.4-4

```
PRINT-TREE-ARBITRARY(T)
1  PUSH(S, T)
2  while S ≠ NULL
3      tmp = POP(S)
4      print tmp.key
5      if tmp.left-child ≠ NULL
6          PUSH(S, tmp.left-child)
7      if tmp.right-sibling ≠ NULL
8          PUSH(S, tmp.right-sibling)
```

10-3

(a) 若算法二在 $1-t$ 的循环中已经返回，则与算法一等价，此时迭代的总次数为 i；

否则，在循环结束后，必有 $key[i] < key[j]$，且 $key[j] < k$。则接下来的 8、9 行运行中进行搜索，可以得到正确的结果。次数算法二的总迭代次数大于 t。

(b) 若在 $1-t$ 的循环中返回，则运行时间为 $O(t)$；

否则，在 **for** 循环结束后的过程中，期望运行时间为 $E[X_t]$。因此，总的期望运行时间为 $O(t + E[X_t])$。

(c)

$$E[X_t] = \sum_{i=0}^{d} iP(X_t = i) = \sum_{i=1}^{d} P(X_t \geq i)$$
$$\leq \sum_{i=1}^{n} P(X_t \geq i) \leq \sum_{i=1}^{n} \left(1 - \frac{r}{n}\right)^t$$

(d)

$$\sum_{i=0}^{n-1} r^t \le \int_0^n x^t \,\mathrm{d}x = \frac{n^{t+1}}{t+1}$$

(e)

$$\begin{aligned} E[X_t] &= \sum_{i=0}^{n} \left(1 - \frac{i}{n}\right)^t = \frac{1}{n^t} \sum_{i=0}^{n-1} i^t \\ &\le \frac{1}{n^t} \cdot \frac{n^{t+1}}{t+1} = \frac{n}{t+1} \end{aligned}$$

(f)

$$O(t + E[X_t]) = O\left(t + \frac{n}{t+1}\right) = O\left(t + \frac{n}{t}\right)$$

(g)

$$t + \frac{n}{t} \ge 2\sqrt{t \cdot \frac{n}{t}} = 2\sqrt{n}$$

故当 $t = \sqrt{n}$ 时，期望运行时间取最小，为 $O(\sqrt{n})$。

(h) 当所有元素均相同，且不是所要找的元素时，渐进运行时间无法降低。