1. 假设一个的栈中的大小永远不超过*k*，每执行*k*次操作，就将栈中所有元素拷贝出来. 该栈的操作代价包括Push、Pop和*k*次操作之后的元素拷贝(拷贝每个元素代价*O*(1)). 用会计方法分析该栈从空开始执行n次操作(包括Push和Pop)的代价.

|  |  |  |
| --- | --- | --- |
| 操作 | 实际代价 | 均摊代价 |
| PUSH | 1 | 3 |
| POP | 1 | 0 |
| COPY(k) | k | 0 |

定义PUSH操作均摊代价为3原因是用1来抵消实际开销，用1作为Credit预支所有POP的开销，用1作为Credit来预支COPY(k)的开销

因为POP操作次数不超过PUSH的操作次数，且COPY(k)的代价不超过当前栈的元素个数，即PUSH的操作次数，所以(1)式非负。所以操作总代价T(n)=O(n)。执行n次操作之后总平均代价为O(1)

2. 设计使用两个栈实现队列的方法，支持平摊代价为*O*(1)的ENQUEUE和DEQUEUE操作. 并用会计方法和Potential方法证明ENQUEUE和DEQUEUE操作的平摊代价为*O*(1).

栈1：每次ENQUEUE把元素PUSH入栈1

栈2：每次DEQUEUE时，如果栈2非空，从栈2 POP一个元素；如果栈2为空，把栈1中元素全都POP后依次PUSH进入栈2(按照POP的顺序)，最后一个元素不用入栈2，直接从栈1POP即可，此时顶端的元素一定是最前出现的元素，符合了FIFO队列的定义。

class Queue {   
private:   
 stack<int> s1, s2;   
public:   
 void enqueue(int a) {   
 s1.push(a); //入栈1  
 cout << "Enqueue " << a << "!\n";   
 }   
   
 void dequeue() {   
 int tmp = -1;   
 if (!s2.empty()) { //栈2非空  
 tmp = s2.top(); //栈2 POP  
 s2.pop();   
 } else { //栈2空  
 while (!s1.empty()) { //栈1元素全部入栈2  
 tmp = s1.top();   
 s1.pop();   
 if (!s1.empty()) s2.push(tmp); //如果到s1最后一个元素了可以直接出栈   
 }   
 }   
 cout << "Dequeue " << tmp << "!\n";   
 }   
};

分析：由上述定义知，每次ENQUEUE实际代价为1，下面分析DEQUEUE。

如果栈2非空，那么DEQUEUE实际代价为1；如果栈2为空，那么DEQUEUE实际代价=2×栈1元素个数+1

1. accounting分析：我们定义均摊代价如下

|  |  |  |
| --- | --- | --- |
| 操作 | 实际代价 | 均摊代价 |
| ENQUEUE | 1 | 3 |
| DEQUEUE | 1或2倍栈1元素个数-1 | 0 |

定义cost(Enqueue)为3，1用于抵消Enqueue的实际代价，2用于作为Credit抵消Dequeue的代价。由于DEQUEUE操作的栈1元素个数至多为2倍ENQUEUE入栈1的个数（出栈1、入栈2各1次），所以我们有

因此操作总代价为O(n)，平摊代价为O(1)

1. potential分析：

这里我们修改一下Dequeue操作，使得Dequeue操作：

在处理栈2空集情况时，我们把栈1左右的元素先POP出来，全部PUSH进入栈2再POP，相当于这一步开销比之前多了2.

class Queue {   
private:   
 stack<int> s1, s2;   
public:   
 void enqueue(int a) {   
 s1.push(a); //入栈1  
 cout << "Enqueue " << a << "!\n";   
 }   
   
 void dequeue() {   
 int tmp = -1;   
 if (!s2.empty()) { //栈2非空  
 tmp = s2.top(); //栈2 POP  
 s2.pop();   
 } else { //栈2空  
 while (!s1.empty()) { //栈1元素全部入栈2  
 tmp = s1.top();   
 s1.pop();   
 s2.push(tmp);   
 }  
 s2.pop(); //POP栈2最上面的一个元素  
 }   
 cout << "Dequeue " << tmp << "!\n";   
 }   
};

设当前栈1元素个数为*l*，栈2元素个数为*r*，我们定义势能。那么

* 对于ENQUEUE操作而言，栈1元素个数+1，那么
* 对于DEQUEUE操作而言，
  + 如果栈2非空，那么
  + 如果栈2为空，那么

综上我们有，操作总开销T(n)=O(n)，均摊代价为O(1)