15-122 : Principles of Imperative Computation, Spring 2014

Written Homework 6

Due before class: Thursday, February 27, 2014

Name:ACM1501 U201514716 罗海 min(LaTex 不支持我名字中的字)

Andrew ID: _____

Recitation: _____

The written portion of this week's homework will give you some practice working with amortized analysis. You can either type up your solutions or write them *neatly* by hand, and you should submit your work in class on the due date just before lecture begins. Please remember to *staple* your written homework before submission.

Question	Points	Score
1	3	
2	7	
Total:	10	

You must do this assignment in one of two ways and bring the stapled printout to the handin box on Thursday:

1) Write your answers *neatly* on a printout of this PDF.

2) Use the TeX template at `http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15122-s14/www/theory6.tgz`

1. **Amortized Analysis**

Consider a k-bit counter where it costs 2^k tokens to flip the kth bit. The rightmost bit (i.e bit 0) costs 1 token to flip. The next bit to its left (bit 1) costs 2 tokens to flip. The next bit to its left (bit 2) costs 4 tokens to flip. And so on. We wish to analyze the time it takes to perform $n = 2^k$ increments of this k-bit counter.

In the worst case, the maximum cost in tokens of an increment occurs when the counter is currently set to all ones and we add 1 to the counter which causes all bits to flip. In this case, the total cost of an increment is:

$$1 + 2 + 2^2 + 2^3 + \ldots + 2^{k-1}$$

(1) (a) Show that in the worst case, the total cost in tokens to increment the counter is $O(n)$.

Solution:
最坏情况下，所有位都是 1，那么共需要

$$1 + 2 + 2^2 + 2^3 + \ldots + 2^{k-1} = 2^k - 1$$

即 $n = 2^k - 1$ 个 tokens，即 $O(n)$

Now, we will use amortized analysis to show that although the worst case for an increment is $O(n)$, the amortized cost of an increment over n increments is asymptotically less than this.

(1) (b) How many tokens are required to flip bit 0 through n increments? (THINK: How often does bit 0 flip when we increment n times?)

Solution:
翻转 n 次需要 d 对第 0 位加 n 次，即需要翻转 n 次，需要 n 个 token

How many tokens are required to flip bit 1 through n increments? (THINK: How often does bit 1 flip when we increment n times?)

Solution:
每加两次对第 1 位翻转一次，需要翻转 n/2 次，每次花费 2tokens，需要 n 个 tokens

(1) (c) Based on your answers to the previous questions, how many tokens are required to increment this counter through n increments using big O notation?

Solution:
共有 n 位，每一位需要 n 个 token，共 $O(nlogn)$

Based on your answer above, what is the amortized cost of a single increment using big O notation?

Solution: $O(logn)$

2. **A New Implementation of Queues**

Recall the interface for `stack` that stores elements of the type `elem`:

```
/* Stack Interface */
stack stack_new();              /* O(1) */
bool stack_empty(stack S);      /* O(1) */
void push(elem e, stack S);     /* O(1) */
elem pop(stack S)               /* O(1) */
  //@requires !stack_empty(S);
  ;
```

We wish to implement a queue using two stacks, called `instack` and `outstack`. For example, the picture below shows a queue with five elements and how they might be stored in the two stacks. (Note that in this example, there are 6 ways to store the elements of the queue using the two stacks. This picture is only one possible way to store the queue shown below.)

(left) An abstract queue with A at the front and E at the back.

(right) A representation of this queue using two stacks.

Based on the picture above, when we enqueue an element from our queue, we push it on top of the `instack`. When we dequeue an element from our queue, we pop the top element of the `outstack`. Of course, an issue occurs if we try to dequeue a non-empty queue and our `outstack` is empty. In this case, we need to move all of the elements from the `instack` to the `outstack` first before we pop the top element of the `outstack`.

We can define our queue data structure as follows in C0:

```
struct queue {
    stack instack;
    stack outstack;
};
typedef struct queue* queue;

queue new_queue()
{
    queue Q = alloc(struct queue);
    Q->instack = stack_new();
    Q->outstack = stack_new();
    return Q;
}
```

(4) (a) Write the queue function `queue_empty` that returns true if both stacks are empty.

Solution:

```
bool queue_empty(queue Q)
{
    return stack_empty(Q->instack) && stack_empty(Q->outstack);
}
```

Write the queue function `enqueue` based on the description of the data structure above.

Solution:

```
void enqueue(elem e, queue Q)
{
    push(Q->instack,e);
}
```

Write the queue function `dequeue` based on the description of the data structure above.

Solution:

```
elem dequeue(queue Q)
\\@requires !queue_empty(Q);
{
    if (stack_empty(Q->outstack)) {
        while(!stack_empty(Q->instack)) {
            push(Q->outstack, pop(Q->instack));
        }
    }
    pop(Q->outstack);
}
```

(1) (b) We now determine the runtime complexity of the `enqueue` and `dequeue` operations. Let q represent the total number of elements in the queue.

What is the worst-case runtime complexity of each of the following queue operations based on the description of the data structure implementation given above? Write ONE sentence that explains each answer.

Solution:
enqueue: $O(1)$
dequeue: $O(m)$, 其中 m 是最坏情况下队列中的元素个数

(2) (c) Using amortized analysis, we can show that the worst-case complexity of a valid sequence of n queue operations, where each operation is either an enqueue or dequeue operation, is $O(n)$. This means that the amortized cost per operation is $O(1)$, even though a single operation might require more than constant time.

Assume that each push and each pop operation requires one token to pay for the operation.

How many tokens in total are required to enqueue an element? State for what purpose each token is used.

Solution: (Your answer for the number of tokens should be a constant integer since the amortized cost should be $O(1)$.)
每次入队需要 3 个 token, 其中入队元素的压栈操作需要 1 个，预留给未来将该元素出队的前置操作的 token 分别为 1 个出 instack 以及一个进 outstack

How many tokens are required to dequeue an element? When the `outstack` is empty and we move elements from `instack` to `outstack`, why does this not require additional tokens?

Solution: (Your answer for the number of tokens should be a constant integer since the amortized cost should be $O(1)$.)
由于出队前置操作已在入队时考虑，故此时出队 token 为 1