Lakehead
UNIVERSITY

COMP 4433: Algorithm Design and Analysis

Dr. Y. Gu

March 15, 2023 (Lecture 15)

Amortized Analysis (Cont.) (Chapter 17)

Topic II: Accounting Method

In the *accounting method* of amortized analysis, we assign differing charges to different operations, with some operations charged more or less than they actually cost.

We call the amount we charge an operation its amortized cost. When an operation's amortized cost exceeds its actual cost, we assign the difference to specific objects in the data structure as credit. Credit can help pay for later operations whose amortized cost is less than their actual cost.

Different operations may have different amortized costs. This method differs from aggregate analysis, in which all operations have the same amortized cost.

Topic II: Accounting Method

If we want to show that in the worst case the average cost per operation is small by analyzing with amortized costs, we must ensure that the total amortized cost of a sequence of operations provides an upper bound on the total actual cost of the sequence.

If we denote the actual cost of the i-th operation by c_i and the amortized cost of the i-th operation by $\hat{c}_i$, we require

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

for all sequence of n operations. The credit stored in the data structure is $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$.

The total credit associated with the data structure must be non-negative at all times.

Topic II: Accounting Method

Now we look at the example of stack operations. The actual costs are

Push 1,
Pop 1,
Multipop min(s,k),

where k is the argument supplied to Multipop and s is the stack size when it is called. Now we assign the following amortized costs:

Push 2,
Pop 0,
Multipop 0.

Topic II: Accounting Method

When we operating a Push, we pay amortized costs 2. But the actual cost is 1. So after a push the data structure has 1 credit which can used for Pop later. Since we cannot pop an empty stack, we always have some credits when we perform Pop or Multipop.

In the whole procedure, the number of credits stored in the stack equals to the number of elements in the stack. So the amount of credits is always nonnegative. Thus for any sequence of n operations, the total amortized cost is an upper bound on the total actual costs. Since the total amortized cost is $O(n)$, so is the actual cost.

Topic II: Accounting Method

Using a similar method for the incrementing a binary counter procedure, we assign amortized cost of 2 to set a bit to 1. When a bit is set, we pay 1 for the actual setting of the bit, and leave 1 credit to be used later. When the bit is flipped back to 0, we use that credit. Now we can determine the amortized cost of Increment.

The cost of resetting the bits within the while loop is paid when the bits are reset. The procedure sets at most one bit, therefore the cost of Increment is at most 2. The number of 1s in the counter is the number of credit, which is never negative. Thus, for n Increment operations, the total amortized cost is $O(n)$, which bounds the total actual cost.

Topic III: Potential Method

We will perform n operations, starting with an initial data structure D_0. For each i = 1, 2, . . . , n, we let c_i be the actual cost of the i^{th} operation and D_i be the data structure that results after applying the i^{th} operation to data structure D_{i-1}.

A **potential function** Φ maps each data structure D_i to a real number $\Phi(D_i)$, which is the potential (or potential energy) associated with data structure D_i.

The amortized cost $\hat{c}_i$ of the i^{th} operation with respect to potential function Φ is defined by

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}).$$

Topic III: Potential Method

The amortized cost of each operation is therefore its actual cost plus the change in potential due to the operation. The total amortized cost of the n operations is

$$\begin{aligned}\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0) \qquad (1)\end{aligned}$$

If we can define a potential function Φ so that $\Phi(D_n) \geq \Phi(D_0)$, then the total amortized cost in (1) gives an upper bound on the total actual cost.

Topic III: Potential Method

In practice, we do not always know how many operations might be performed. Therefore, if we require that $\Phi(D_i) \geq \Phi(D_0)$ for all i, then we guarantee, as in the accounting method, that we pay in advance.

We usually just define $\Phi(D_0)$ be 0 and then show that $\Phi(D_i) \geq 0$ for all i.

The amortized costs depend on the choice of the potential function Φ. Different potential functions may yield different amortized costsyet still be upper bounds on the actual costs.

Topic III: Potential Method

Consider the stack operations Push, Pop and Multipop. We define the potential function Φ on a stack to be the number of objects in the stack. For the empty stack D_0 with which we start, we have $\Phi(D_0) = 0$. Since the number of objects in the stack is never negative, the stack D_i that results after the i-th operation has nonnegative potential, and thus $\Phi(D_i) \geq \Phi(D_0)$. The total amortized cost of n operations with respect to Φ therefore represents an upper bound on the actual cost.

Suppose that the i-th operation on the stack is Multipop, which causes $k' = \min(s,k)$ objects to be popped off the stack.

Topic III: Potential Method

Let us now compute the amortized costs of the various stack operations. If the ith operation on a stack containing s objects is a PUSH operation, then the potential difference is

$$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) &= (s+1) - s \\ &= 1 .\end{aligned}$$

the amortized cost of this PUSH operation is

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + 1 \\ &= 2 .\end{aligned}$$

Suppose that the i-th operation on the stack is Multipop, which causes k′= min(s,k) objects to be popped off the stack.

Topic III: Potential Method

The actual cost of the operation is k′, and the potential difference is

$$\Phi(D_i) - \Phi(D_{i-1}) = -k'.$$

Thus, the amortized cost of the MULTIPOP operation is

$$\begin{aligned} \hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= k' - k' \\ &= 0. \end{aligned}$$

Similarly, the amortized cost of an ordinary Pop operation is 0. The total amortized cost of a sequence of n operations is $O(n)$. Since we have already argued that $\Phi(D_i) \geq \Phi(D_0)$, the total amortized cost of n operations is an upper bound on the total actual cost. The worst-case cost of n operations is therefore $O(n)$.

Topic III: Potential Method

Now we consider the increment a binary counter. We define the potential of the counter after the i-th Increment operation to be b_i, the number of 1s in the counter after the i-th operation.

Suppose that the i-th operation resets t_i bits. The actual cost of the operation is therefore at most t_{i+1}, since in addition to resetting t_i bits, it sets at most one bit to 1. If $b_i = 0$, then the i-th operation resets all k bits, and so $b_{i-1} = t_i = k$. If $b_i > 0$, then $b_i = b_{i-1} - t_i + 1$. In either case, $b_i \leq b_{i-1} - t_i + 1$, and the potential difference is

$$\begin{aligned} \Phi(D_i) - \Phi(D_{i-1}) &\leq (b_i - t_i + 1) - b_{i-1} \\ &= 1 - t_i. \end{aligned}$$

Topic III: Potential Method

The amortized cost is

$$\begin{aligned} \hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq (t_i + 1) + (1 - t_i) \\ &= 2. \end{aligned}$$

If the counter starts at zero, then $\Phi(D_0) = 0$. Since $\Phi(D_i) \geq 0$ for all i, the total amortized cost of a sequence of n Increment operations is an upper bound on the total actual cost, and so the worst-case cost of n Increment operations is $O(n)$.

Topic III: Potential Method

The potential method gives us a way to analyze the counter even when it does not start at zero. If the counter starts with b_0 1s, and after n Increment operations it has b_n 1s, where $0 \leq b_0, b_n \leq k$ (Recall that k is the number of bits in the counter.). From $c_i = \hat{c}_i - \Phi(D_i) + \Phi(D_{i-1})$, we have

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c}_i - \Phi(D_n) + \Phi(D_0).$$

We have $\hat{c}_i \leq 2$ for all $1 \leq i \leq n$. So the total actual cost of n Increment operations is

$$\begin{aligned} c_i &\leq \sum_{i=1}^{n} 2 - b_n + b_0 \\ &= 2n - b_n + b_0. \end{aligned}$$

Topic III: Potential Method

Note in particular that since $b_0 \leq k$, as long as $k = O(n)$, the total actual cost is O(n).

In other words, if we execute at least $n = \Omega(k)$ Increment operations, the total actual cost is O(n), no matter what initial value the counter contains.

Topic IV: Dynamic Table

In some software environments, we shall assume that our software environment provides a memory-management system that can allocate and free blocks of storage on request. Thus, upon inserting an item into a full table, we can expand the table by allocating a new table with more slots than the old table had. Because we always need the table to reside in contiguous memory, we must allocate a new array for the larger table and then copy items from the old table into the new table. A common heuristic allocates a new table with twice as many slots as the old one. If the only table operations are insertions, then the load factor of the table is always at least 1/2, and thus the amount of wasted space never exceeds half the total space in the table.

Topic IV: Dynamic Table

```
TABLE-INSERT(T, x)
 1  if T.size == 0
 2       allocate T.table with 1 slot
 3       T.size = 1
 4  if T.num == T.size
 5       allocate new-table with 2 · T.size slots
 6       insert all items in T.table into new-table
 7       free T.table
 8       T.table = new-table
 9       T.size = 2 · T.size
10  insert x into T.table
11  T.num = T.num + 1
```

Topic IV: Dynamic Table

To consider the running time of the above procedure, we assign cost 1 to each elementary insertion. We assume that the allocating table uses constant time. In this way, the main overhead for the second if block is dominated by the cost of transferring items in line 6.

Let us analyze a sequence of n Table-Insert operations on an initially empty table. If the current table has room for the new item (or if this is the first operation), then $c_i = 1$. If the current table is full, then $c_i = i$: the cost is 1 for the elementary insertion in line 10 plus $i - 1$ for the items that we must copy from the old table to the new table in line 6. If we perform n operations, the worst-case cost of an operation is O(n), which leads to an upper bound of $O(n^2)$ on the total running time for n operations.

Topic IV: Dynamic Table

But, the i-th operation causes an expansion only when i − 1 is an exact power of 2. The amortized cost of an operation is in fact O(1) as we can show using aggregate analysis. The cost of the i-th operation is

$$c_i = \begin{cases} i & \text{if } i-1 \text{ is an exact power of 2,} \\ 1 & \text{otherwise.} \end{cases}$$

The total cost of n Table-Insert operations is therefore

$$\begin{aligned} \sum_{i=1}^{n} c_i &\le n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j \\ &< n + 2n \\ &= 3n. \end{aligned}$$

Since the total cost of n Table-Insert operations is bounded by 3n, the amortized cost of a single operation is at most 3.

Topic IV: Dynamic Table

By using the accounting method, we can gain some feeling for why the amortized cost of a Table-Insert operation should be 3. We assign 3 as amortized cost for each operation. Suppose that the size of the table is m immediately after an expansion. Then the table holds $m/2$ items, and it contains no credit. We charge 3 for each insertion. The elementary insertion that occurs immediately costs 1. We place another 1 as credit on the item inserted. We place the third 1 as credit on one of the m/2 items already in the table. The table will not fill again until we have inserted another $m/2 - 1$ items, and thus, by the time the table contains m items and is full, we will have placed 1 credit on each item to pay to reinsert it during the expansion.

Topic IV: Dynamic Table

We can use the potential method to analyze a sequence of n Table-Insert operations. We define the potential function Φ that is 0 immediately after an expansion but builds to the table size by the time the table is full. So we may define

$$\Phi(T) = 2 \cdot T.num - T.size.$$

Immediately after an expansion, we have $T.num = T.size/2$, and thus $\Phi(T) = 0$. Immediately before an expansion, we have $T.num = T.size$, and thus $\Phi(T) = T.num$, as desired. The initial value of the potential is 0, and since the table is always at least half full, $T.num \geq T.size/2$, which implies that $\Phi(T)$ is always nonnegative. Thus, the sum of the amortized costs of n Table-Insert operations gives an upper bound on the sum of the actual costs.

Topic IV: Dynamic Table

To analyze the amortized cost of the ith operation, we let num_i denote the number of items stored in the table after the i-th operation, $size_i$ denote the total size of the table after the i-th operation, and Φ_i denote the potential after the ith operation. Initially, we have $num_0 = 0$, $size_0 = 0$, and $\Phi_0 = 0$.
If the i-th operation does not trigger an expansion, then we have $size_i = size_{i-1}$ and the amortized cost of the operation is

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2num_i - size_i) - (2num_{i-1} - size_{i-1}) \\
&= 1 + (2num_i - size_i) - (2(num_i - 1) - size_{i-1}) \\
&= 3.
\end{aligned}$$

Topic IV: Dynamic Table

If the ith operation does trigger an expansion, then we have $size_i = 2size_i - 1$ and $size_{i-1} = num_{i-1} = num_i - 1$, which implies that $size_i = 2(num_i - 1)$. Thus, the amortized cost of the operation is

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= num_i + (2num_i - size_i) - (2num_{i-1} - size_{i-1}) \\
&= num_i + (2num_i - 2(num_i - 1)) - (2(num_i - 1) - num_i - 1)) \\
&= num_i + 2 - (num_i - 1) \\
&= 3.
\end{aligned}$$

Topic IV: Dynamic Table

To implement a Table-Delete operation, when the number of items in the table drops too low, we allocate a new, smaller table and then copy the items from the old table into the new one. We can then free the storage for the old table by returning it to the memory-management system.

We may halve the size when a deleting an item would cause the table to become less than half full. This strategy would guarantee that the load factor of the table never drops below 1/2, however, it can cause the amortized cost of an operation to be quite large.

Topic IV: Dynamic Table

Consider the following scenario. We perform n operations on a table T , where n is an exact power of 2. The first n/2 operations are insertions, which by our previous analysis cost a total of Θ(n). At the end of this sequence of insertions, T.num = T.size = n/2. For the second n/2 operations, we perform the following sequence:
insert, delete, delete, insert, insert, delete, delete, insert, insert, . . .

The first insertion causes the table to expand to size n. The two following deletions cause the table to contract back to size n/2.

Topic IV: Dynamic Table

To use the potential method to analysis the cost of a sequence of n Table-Insert and Table-Delete operations, we need to define the load factor α(T) = T.num/T.size. For an empty table, T.num = T.size = 0, we define α(T) = 1. Then we always have T.num = α(T) · T.size no matter the table is empty or not. We shall use the potential function

$$\Phi(T) = \begin{cases} 2 \cdot T.num - T.size & \text{if } \alpha(T) \geq 1/2, \\ T.size/2 - T.num & \text{if } \alpha(T) < 1/2. \end{cases} \quad (2)$$

Topic IV: Dynamic Table

Note that the potential of an empty table is 0 and that the potential is never negative. Thus the total amortized cost of a sequence of operations with respect to Φ provides an upper bound on the actual cost of the sequence.

- When the load factor is 1/2, the potential is 0.
- When the load factor is 1, $\Phi(T) = T.num$, and thus the potential can pay for an expansion if an item is inserted.
- When the load factor is 1/4, we have $T.size = 4T.num$, which implies $\Phi(T) = T.num$, and thus the potential can pay for a contraction if an item is deleted.

Topic IV: Dynamic Table

To analyze a sequence of n insert and delete operations, we let c_i denote the actual cost of the ith operation, $\hat{c}_i$ denote its amortized cost with respect to Φ, num_i denote the number of items stored in the table after the ith operation, $size_i$ denote the total size of the table after the ith operation, α_i denote the load factor of the table after the i-th operation, and Φ_i denote the potential after the i-th operation. Initially, $num_0 = 0$, $size_0 = 0$, $\alpha_0 = 1$, and $\Phi_0 = 0$.

Topic IV: Dynamic Table

We start with the case in which the ith operation is Table-Insert. If $\alpha_{i-1} < 1/2$, the table cannot expand as a result of the operation, since the table expands only when $\alpha_{i-1} = 1$. If $\alpha_i < 1/2$ as well, then the amortized cost of the ith operation is

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\
&= 1 + (size_i/2 - num_i) - (size_i/2 - num_i - 1)) \\
&= 0
\end{aligned}$$

Topic IV: Dynamic Table

If $\alpha_{i-1} < 1/2$ but $\alpha_i \geq 1/2$, then

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (size_{i-1}/2 - num_{i-1}) \\
&= 1 + (2(num_{i-1} + 1) - size_{i-1}) - (size_{i-1}/2 - num_{i-1}) \\
&= 3 \cdot num_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&= 3\alpha_{i-1} size_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&< \frac{3}{2} size_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&= 3
\end{aligned}
$$

Thus the amortized cost of an insert operation is at most 3.

Topic IV: Dynamic Table

When the ith operation is delete, then $num_i = num_{i-1} - 1$. If $\alpha_{i-1} < 1/2$, then we must consider whether the operation causes the table to contract. If it does not, then $size_i = size_{i-1}$ and the amortized cost of the operation is

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\
&= 1 + (size_i/2 - num_i) - (size_i/2 - (num_i + 1)) \\
&= 2
\end{aligned}$$

If $\alpha_{i-1} < 1/2$ and the ith operation does trigger a contraction, then the actual cost of the oeration is $c_i = num_i + 1$, since we delete omn item and move num_i items.

Topic IV: Dynamic Table

We have $size_i/2 = size_{i-1}/4 = num_{i-1} = num_i + 1$, and the amortized cost of the operation is

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= (num_i + 1) + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\
&= (num_i + 1) + ((num_i + 1) - num_i) - ((2 \cdot num_i + 2) - (num_i + 1)) \\
&= 1 .
\end{aligned}$$

When the ith operation is Table-Delete and $\alpha_{i-1} \geq 1/2$, the amortized cost is also bounded above by a constant. In summary, since the amortized cost of each operation is bounded above by a constant, the actual time for any sequence of n operations on a dynamic table is O(n).

After Class

After class reading: Part IV 17.2-17.4