Tony Kovanen

DAA Week IV

1. The worst case cost of a single operation is O(k) where k is the number of items pushed into the stack by the operation multipush or O(n), where n is the number of pops in multipop: O(n + k). This makes the amortized cost of each individual operation O(n + k)/n, which means that the O(1) bound that we could see earlier does no hold any longer.
2. Every ith operation has the cost of O(1) of or O(i), iff i is an exact power of two. This corresponds to a situation of eagerly dynamically expanding a table when the next insert operations would overflow it. We know that when an insertion brings the last element to the table, it will expand and copy all the values to a new table, making the worst case running time of all these sequential operations O(n). O(n)/n = O(1), which is the amortized cost of a single operation.
3. Let's consider this problem again as eager dynamic expansion of a table. Let's define a cost ci of each operation as 1, and an additional cost of 2 for each insertion, so ^ci=3. This cost can be divided as follows. 1 token is used to pay for the insertion operation itself. 1 token is used to pay for the moving of copying of the value itself to a newly allocated table when the table is expanded. 1 token is used to do the same for one old value in the table.

   If we now do any sequence of insertions on the table, we always have >= 0 tokens in bank, 0 when expansion has just taken place, otherwise more than 0 tokens, in fact 2 tokens per value added after last expansion. This means that this is the total cost of the sequence of operations, making the amortized cost of any n operations of the cost defined O(1).
4. Let's define our potential function Φ as Φ = 2i – 2^{log2(i)} and once more look at this problem as eager dynamic table expansion. ^ci = ci + Φ(Di) - Φ(D_{i-1}) = (i when i is an exact power of 2, otherwise i = 1) + 2i – 2^{log2(i)} - 2(i-1)i – 2^{log2(i-1)} = (i when i is an exact power of 2, otherwise i = 1) + 2 - 2i + 2^{log2(i – 1)} – 2^{log2(i)}.

   If i is an exact power of 2, then ^ci = i + 2 - 2i + 2^{log2(i – 1)} – 2^{log2(i)} = i + 2 – 2(i-1) + (i-1) = 3

   Else ^ci = 1 + 2 - 2i + 2^{log2(i – 1)} – 2^{log2(i)} = 3.

   This means that since in any case when the data structure (table) is accessed (insert), the amortized cost per operation is 3. This would implicate that the sequence of n operations of inserts, thus any number i of operations, would be O(1), altogether O(n).
5. The push operation has an additional cost of 1 to prepare all of the values in the stack for a possible pop or multipop. This means that in the worst case, only push operations are executed, and 2n will the the total cost of these operations, where n is the number of operations.
6. -