Lab05-Amortized Analysis

Algorithm: Analysis and Theory (X033533), Xiaofeng Gao, Spring 2018.

$*$ If there is any problem, please contact TA Mingding Liao.

$*$

1. A **multistack** consists of an infinite series of stacks $S_0, S_1, S_2, \cdots$, where the i^{th} stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first pop all the elements off S_i and push them onto stack S_{i+1} to make room. (Thus, if S_{i+1} is already full, we first recursively move all its members to S_{i+2} .) An illustrative example is shown in Figure 1. Moving a single element from one stack to the next takes $O(1)$ time. If we push a new element, <u>we always intend to push it in stack S_0</u>.

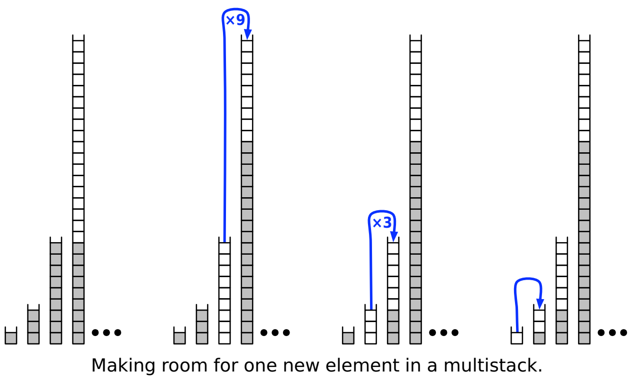

图 1: An example of making room for one new element in a multistack.

   (a) In the worst case, how long does it take to push a new element onto a multistack containing n elements?

   (b) Prove that the amortized cost of a push operation is $O(\log n)$ by *Aggregation Analysis*.

   (c) (Optional Subquestion with Bonus) Prove that the amortized cost of a push operation is $O(\log n)$ by *Potential Method*.

Solution :(a)

A multistack usually consists of an infinite series of stacks S_0,S_1... S_{t-1} where the i^{th} stack s_j can hold upto to 3 elements. Whenever a user attempts to push an element onto a full stack S, we first POP all the elements off S_i and push them back onto stack S_{i+1} to make room.Thus S_{i+1} is already full, we recursively move all its members to S_i+2. Moving a single element from one stack to the next takes $O(1)$ time. For instance, here is how we would insert a new element into a multistack that already contains 15 elements. Let's take an example in order to explain this problem.

<u>Note:</u> This is only to show how this system works exactly.

Suppose we push 15 elements (let the elements be 1...15) onto an empty infinite stack and then POP 7 elements out. List the contents of stacks S_0,S_1,S_2,S_3,S_4 and ,S_5 in correct order after these 22 operations. After pushing in 15 elements, the first 4 stacks are all full, and the contents of the stacks would be:

S_0 =15 S_1 =13,14 S_2 =10,9,12,11 S_3 =3,4,1,2,7,8,5,6

Where in each stack, the numbers are listed according to the order of from the bottom of the stack to the top. All other stacks S_i with i>3 are empty. After popping out 7 elements, the stack becomes:

S_0 =8 S_1 =7,o S_2 =6,5,o,o S_3 =3,4,1,2,o,o,o,o
where "o" is used to represent an empty entry. All other stacks are empty.
When the new element is being PUSH, the elements that are present in the multistack are also PUSH and POP. because of this we have to push and pop all the elements in the multistack immediately. So in Worst case scenario we can mathematically assume that a new pushing element is $O(n)$.

(b)
Let us first push in n elements such that all the stacks from S_0 to S_i are full. Now we repeat doing "push,pop". When stacks from S_0 to S_i are full and we operate a push, this push costs $2n+1$ time and the status of the stacks would make S_0 full and S_1 to S_{i+1} as half full. Now what we do is to POP them cost $2n+1$ time according to the prior analysis that we made, we can repeat this for an infinite number of times. So the worst case continues and the cost of these operations grows faster than the number of operations, which shows that POP and PUSH on the infinite stack cannot be achieved in amortized Constant time. Since the stack contains the 3^i elements so for the N elements in the stack , the $log_3 n$ completely fill all the stacks.
This shows that pushing the elements moves it to a later stack which is in our case is $O(logn)$ times.
Thus number of operations of PUSH performed would cost $O(n\frac{logn}{n}) = O(logn)$

(c)
Another way of proving it is to use
$\Phi(D_i) = 2\sum_{j=0}^{t-1} S_i\ (S_j)\ (log_3 n - j)$ where $S_i\ (S_j)$
is the number of elements stack S_j contains in D_i
Now we can start by computing the amortized cost of moving the elements of a full stack onto the next stack. The strategy is to define a potential function Φ which maps a state D to a scalar-valued potential $\Phi(D)$ Given a sequence of n operations with actual costs $c_1...c_n$ which transform the stacks from its initial state S_0 through states $S_1...S_n$. which potentially will leads to the conclusion that amortized cost after potential analysis proves to be $O(logn)$.
Note: For more details, Please visit this site.
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-desig
lecture-notes/MIT6_046JS12_lec11.pdf
https://tinyurl.com/go6lcfp

2. A factory needs to deliver a kind of product in 2 months. Suppose that for month i ($i = 1 or 2$): the contract requires the factory to deliver d_i products; the selling price for a product is s_i; the capitalized cost for a product is c_i; the working time needed for a product is t_i. In month i, the normal working time is no more than T_i, and it is allowed to do extra work, but the extra working time is no more than T_i', and each product produced in extra working time has an extra c_i' in its capitalized cost. If the products are stored (not delivered) in month i, the storage cost p_i is required to pay for each stored product.

   Please design a production plan in the form of linear programming, which maximizes the overall profit under all possible constraints mentioned above.

   (a) Please add some necessary explanations on your objective function and constraints, and finally write your LP in *standard* form.
   (b) Transform your LP into its dual form.

Solution: (a)
production in i month: a_i , $a_i^{'}$

Product in extra working day in month i: a_i , $a_i^{'}$ for sake of simplicity

Capital Cost for month i : $c_i a_i + (c_i + c_i^{'})a_i^{'}$
Storage cost A: $(a_1 + a_1^{'} - d_1)p_1 = y_1 p_1$ Month A
Storage Cost B: $(a_1 + a_1^{'} - d_1 + a_2 + a_2^{'} - d_2)p_2 = y_2 p_2$ MONTH B
MIN [-1]$[-c_1 a_1 - (c_1 + c_1^{'})a_1^{'} + c_2 a_2 + (c_2 + c_2^{'})a_2^{'}]$ + $[p_1 y_1 + p_2 y_2]$
Main objective is to maximize the profit, to increase the profit the function should maximize total sale of the product and total cost of the production. $a_1 + a_1^{'} - y_1 = d_1$
$a_1 + a_1^{'} + a_2 + a_2^{'} - y_2 = d_1 + d_2$
$t_1 a_1 \geq T_1$
$t_1 a_2 \geq T_1$
$t_2 a_1^{'} \geq T_2$
$t_2 a_2^{i} \geq T_2$
For the sake of simplicity we can consider as term i for both the months A and B
$t_i a_i \geq T_i$
$t_i a_i^{'} \geq T_i^{'}$
a_i, $a_i^{'}$,$y_i \geq 0$ Converted the Profit maximization to Cost minimze problem in order to get more profit
Reason: Profit Maximization has fixed value
Standard LP: MAX [-1]$[c_1 a_1 + c_1 a_1^{'} + c_1^{'} a_1^{'} + c_2 a_2 + c_2 a_2^{'} + c_2^{'} a_2^{'} - p_1 y_1 - p_2 y_2]$

$a_1 + a_1^{'} - y_1 \geq d_1$
$a_1 + a_1^{'} - y_1 \geq -d_1$
$a_1 + a_1^{'} + a_2 + a_2^{'} - y_2 \geq d_1 + d_2$
$a_1 + a_1^{'} + a_2 + a_2^{'} - y_2 \geq -d_1 - d_2$
For the sake of simplicity we can consider as term i for both the months A and B
$t_i a_i \geq T_i$
$t_i a_i^{'} \geq T_i^{'}$
a_i,$a_i^{'}$,$y_i \geq 0$
Dual Form LP: MIN [-1]$[-d_1 y_1 + d_1 y_2 - (d_1 + d_2)y_3 + (d_1 + d_2)y_4]$ [sum(T_i,y_i)]

MIN [-1]$[-d_1 y_1 + d_1 y_2 - (d_1 + d_2)y_3 + (d_1 + d_2)y_4$ - $(T_1 + T_1^{'})y_5]$
OR
MIN [-1]$[-d_1 y_1 + d_1 y_2 - (d_1 + d_2)y_3 + (d_1 + d_2)y_4$ - $T_1 y_5 - T_2 y_6 - T_1^{'} y_7 - T_1^{'} y_8]$

$y_1 + y_2 + y_3 + y_4 + t y_5 \geq -c_{-1}$
$y_1 + y_2 + y_3 + y_4 + t y_6 \geq -c_{-1}$
$y_3 + y_4 + t y_7 \geq -c_{-1}^{'}$
$y_3 + y_4 + y_8 \geq -c_{-2}$
$(-1)(y_1 + y_2) \geq -c_{-2}$
$(-1)(y_3 + y_3 - t y_8) \geq -c_{-2}^{'}$
y_1,y_2,y_3,y_4,y_5,y_6,y_7,$y_8 \geq 0$