CS4311
Design and Analysis of Algorithms

Lecture 16: Amortized Analysis III

About this lecture

- Application of Amortization
  - Managing a Dynamic Table

Dynamic Table

- Sometimes, we may not know in advance the #objects to be stored in a table
- We may allocate space for the table, say with malloc(), at the beginning
- When more objects are inserted, the space allocated before may not be enough

Dynamic Table

- Then, the table must be reallocated with a larger size, say with realloc(), and all objects from original table must be copied over into the new, larger table
- Similarly, if many objects are deleted, we may want to reallocate the table with a smaller size (to save space)
- Any good choice for the reallocation size?

Load Factor

- For a table T, we define the load factor, denoted by LF(T), to be the ratio of #items stored in T and the size of T
  - That is, LF is between 0 and 1, and the fuller the T, the larger its LF

- To measure how good the space usage in a reallocation scheme, we can look at the load factor it guarantees

Load Factor

- Obviously, we have a reallocation scheme that guarantees a load factor of 1:
  - Rebuild table for each indel
- However, n indels can cost $\Theta(n^2)$ time
- In general, we want to trade some space for time efficiency
  - Can we ensure any n indels cost $\Theta(n)$ time, and a not-too-bad load factor?

Handling Insertion

- Suppose we have only insertion operations
- Our reallocation scheme is as follows:

> If T is full after insertion of an item,
> we expand T by doubling its size

- It is clear that at any time,

  LF(T) is at least 0.5

- Question: How about the insertion cost?

Handling Insertion

- Observe that the more the items stored, the closer the next expansion will come
- Let num(T) = #items currently stored in T
- Let size(T) = size of T
- To reflect this observation, we define a potential function Φ such that

$$\Phi(T) = 2\ \mathrm{num}(T) - \mathrm{size}(T)$$

Handling Insertion

- The function Φ has some nice properties:
  - Immediately before an expansion,

$$\Phi(T) = \mathrm{num}(T)$$

➔ this provides enough cost to copy items into new table

  - Immediately after an expansion,

$$\Phi(T) = 0$$

➔ this resets everything, and simplify the analysis

  - Its value is always non-negative

Amortized Cost of Insertion

- Now, what will be amortized insertion cost?
- Notation
  - c_i = actual cost of i^{th} operation
  - α_i = amortized cost of i^{th} operation
  - num_i = #items in T after i^{th} operation
  - $size_i$ = size of T after i^{th} operation
  - Φ_i = $\Phi(T)$ after i^{th} operation
- There are two cases …

Case 1: No Expansion

- If i^{th} insertion does not cause expansion:

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

Case 2: With Expansion

- If i^{th} insertion causes an expansion:

$$\alpha_i = c_i + \Phi_i - \Phi_{i-1}$$
$$= num_i + (2num_i - size_i) - (2num_{i-1} - size_{i-1})$$
$$= num_i + 0 - (2(num_i - 1) - num_i)$$
$$= 2$$

Conclusion:

amortized cost for insertion = O(1)

Handling Insertion & Deletion

- Suppose we have both insertion & deletion
- Can we still maintain a reallocation scheme with LF(T) is at least 0.5 ?
- To do so,
  - when table is full, we need to expand as usual, AND
  - when table is just below half-full, we need to contract immediately

Handling Insertion & Deletion

- Will the following scheme work?

If T is full after insertion of an item, we expand T by doubling its size

If T is below half-full after deletion, we contract T by halving its size

- In worst-case, n indels cost $\Theta(n^2)$ time

Slight Modification

- The previous scheme fails because we are too greedy … (contracting too early)
- Let us modify our scheme slightly:

If T is full after insertion of an item,
we expand T by doubling its size

If T is only (1/4)-full after deletion,
we contract T by halving its size

- At any time, LF(T) is at least 0.25

Handling Insertion & Deletion

- Now, using this scheme,
  - If table is more than (1/2)-full, we should start worrying about the next expansion ➔ watch for insertion
  - If table is less than (1/2)-full, we should start worrying about the next contraction ➔ watch for deletion
- This gives us some intuition of how to define the potential function

New Potential Function

- Our new potential function Φ is a bit strange (it has two parts):
  - If table is at least half full:

$$\Phi(T) = 2\ \mathrm{num}(T) - \mathrm{size}(T)$$

  - If table is less than half full:

$$\Phi(T) = \mathrm{size}(T)/2 - \mathrm{num}(T)$$

- Can you compute the amortized cost for each operation?

Nice Properties

- The function Φ has some nice properties:
  - Immediately before a resize,

$$\Phi(T) = \mathrm{num}(T)$$

➔ this provides enough cost to copy items into new table

  - At half-full or immediately after resize,

$$\Phi(T) = 0$$

➔ this resets everything, and simplify the analysis

  - Its value is always non-negative

Amortized Insertion Cost

- If i^{th} operation = insertion
- If it causes an expansion:

$$\alpha_i = \text{same as before} = 2$$

- If it does not cause expansion:
  - if T at least half full,

$$\alpha_i = \text{same as before} = 3$$

  - if T less than half full,

$$\alpha_i = c_i + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1})$$
$$= 1 + (-1) = 0$$

Amortized Deletion Cost

- If i^{th} operation = deletion
- If it causes a contraction:

$$\alpha_i = c_i + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1})$$
$$= num_i + 0 - (num_i - 1) = 1$$

- If it does not cause a contraction:
  - if T less than half full,

$$\alpha_i = c_i + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1})$$
$$= 1 + 1 = 2$$

Amortized Deletion Cost (cont)

- If it does not cause a contraction:
  - if T at least half full,

$\alpha_i = c_i + (2num_i - size_i) - (2num_{i-1} - size_{i-1})$

$= 1 - 2 = -1$

Conclusion

- The amortized insertion or deletion cost in our new scheme = O(1)
- Meaning:
  Any n operations in total cost O(n) time
- Remark: There can be other reallocation schemes with O(1) load factor and O(1) amortized cost (Try to think about it !)