Array Resizing

Assume we insert into an array (unit time per insertion) when array is full, next insert:

1. Allocate new array with twice the capacity
2. Copy all items into new array (unit time per item copied)

Assume

Time = # items copied.

\( O(1) \) amortized time per operation, i.e.,

\( n \) insertions takes \( O(n) \) time, starting with empty list.

Potential method: \( n \) operations \( a_1, \ldots, a_n \) performed on some data structure \( S \).
Let \( c_i \) be the actual cost of \( a_i \). Let \( S_0 \) be initial state of \( S \), and let \( S_i \) be the state of \( S \) after \( a_i \).

\( S_0, S_1, \ldots, S_n \)
We define a function
\[ \Phi(S_i) \quad (i=0, ..., n) \]
\[ \in \mathbb{R} \]
\( \Phi(S_i) \) depends on
the state \( S_i \)
the potential function.
Such that:
\[ \forall i: \Phi(S_i) \geq 0 \quad \text{and} \quad \Phi(S_0) = 0 \]
Define the amortized
cost of op \( a_i \) as
\[ \hat{c}_i := c_i + \Phi(S_i) - \Phi(S_{i-1}) \]
net change in potential

Trick: define \( \Phi \) so
that \( \hat{c}_i \) is low for all \( i \).
\[ T = \text{total time for } a_1, ..., a_n. \]
\[ T = \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} c_i + \Phi(S_n) \]
\[ = \sum_{i=1}^{n} [c_i + \Phi(S_i) - \Phi(S_{i-1})] \]
\[ = \sum_{i=1}^{n} \hat{c}_i \]
so total time is \( \leq \) total
amortized time.

\[ \therefore \text{good bound on amortized time} \]
\[ \text{guarantees good bound on actual time} \]
Trick: choose good $\Phi$ to keep all $\hat{C}_i$ low.

Back to current example.

A array with capacity $A[1..s]$ holding $n$ items currently ($n \leq s$) ($s$ power of 2).

Define: $\alpha = \frac{n}{s}$ (\(\alpha\) = load factor)

$$
\Phi = \begin{cases} 
2n-s & \text{if } \alpha \geq \frac{1}{2} \\
0 & \text{otherwise}
\end{cases}
$$

after any resize, array is exactly half full.

Amortized cost per operation:

insert:

$$
\hat{C}_i = C_i + \Phi(s'_i) - \Phi(s_i)
$$

$$
= 1 + (2(n+1) - s) - (2n - s)
$$

$$
= 1 + 2 = 3
$$

array resizing:

$n = s$

$s \rightarrow 2s$

$n \rightarrow n$

$$
\hat{C}_i = C_i + \Delta \Phi
$$

$$
= n + (2n - 2s) - (2n - s)
$$

$$
= n - s = 0
$$
Deletion (1 time)

Bad strategy:
  when \( n < \frac{s}{2} \)
  resize \( s_{\text{new}} \rightarrow \frac{s_{\text{old}}}{2} \)
  cost of resize still
  = \# items (whether down or up)

Good strategy:
  halve the size of the array when \( \alpha \) falls below \( \frac{1}{4} \).

\( n = \# \text{ items in array} \)
\( s = \text{capacity of array} \)
\( \alpha := \frac{n}{s} \) (load factor)
\( \alpha \geq \frac{1}{4} \) always (except at beginning)

\[ \phi := \begin{cases} 2n - s & \text{if } \alpha \geq \frac{1}{2} \\ \frac{s}{2} - n & \text{if } \alpha < \frac{1}{2} \end{cases} \]
Binary counter:
  cost 1 per bit change

\[ 01111111 \]

\( \Phi = \# \) of rightmost contiguous 1's

\( \Phi = 0 \) initially

\( \Phi \geq 0 \) always

\( \Phi_{\text{old}} = \text{i} \quad 0111\ldots1 \) (increment)

\( \Phi_{\text{new}} = 10\ldots0 \) if \( i > 0 \).

amortized cost:

\[ = (i + 1) + \Phi_{\text{new}} - \Phi_{\text{old}} \]

\[ = (i + 1) + 0 - i = 1 \]

\( i = 0: \)

\[ \underbrace{11111110}_i \quad \underbrace{0}_{i+1} \quad \Phi = 0 \]

\[ \underbrace{\ldots1111}_{i+1} \quad \Phi = i + 1 \]

amortized cost:

\[ = 1 + (i + 1) - 0 = i + 2 \neq O(1) \]

Not a good choice of potential function.

Better choice:

\[ \Phi = \text{total \# of 1's in the bit sequence. (Hamming weight)} \]
increment/decrement

each "bit" \( \in \{-1, 0, 1, 2\} \)

Exercise: Show that this is \( O(1) \) amortized time per op.

Hint: Potential method

where

\( \Phi = \# \text{ digits that are } -1 \text{ or } 2 \).

Stack with multipop

\( \Phi = \text{size of stack} \).

Amort. anal. not appropriate in some cases:

- real-time, critical systems.
- shared structures among users (fairness).

Binomial Heaps

Implement mergeable

Heaps:

- \( \text{binary heaps} \)

\[ \begin{align*}
\text{insert} & \quad \text{ExtractMin} \\
\text{FindMin} & \quad \text{DecreaseKey} \\
\text{Union}(H_1, H_2) & \quad \text{merges } H_1, H_2 \\
\text{into one heap} & \\
\text{\( O(\log n) \)} & \quad \text{\( H_1, H_2 \) destroyed}
\end{align*} \]