Dynamic Array

init: capacity = 1, length = 0

add(5): capacity = 1, length = 1

add(7): capacity = 2, length = 2

add(3): capacity = 4, length = 3

add(2):

add(4): capacity = 8, length = 5

aggregate method

- analyze the cost of single operations

  what is the cost of i-th add operation?

  - case 1: if we do not need to allocate new memory
    
    \[ t(i) \text{ takes } \frac{1}{0(1)} \text{ unit of time.} \]
    
    \[ (i \neq 2^k + 1 \text{ for any } k) \]

  - case 2: if we need to allocate new memory
    
    \[ i = 2^k + 1 \text{ for } k = 0, 1, 2, \ldots \]
    
    - need to allocate an array of size \(2^{k+1}\) (doubling capacity)
    
    - copy \(2^k\) numbers from old array to new array
    
    - add the new number as the \(2^k + 1\) element.
    
    \[ t(i) \text{ takes } \frac{2^k + 1}{0(2^k)} \text{ unit of time.} \]

  \[ t(i) = \begin{cases} 
    \frac{2^k + 1}{0(2^k)} & \text{if } i = 2^k + 1 \text{ (case 2)} \\
    1 & \text{otherwise (case 1)} 
  \end{cases} \]
 amortized cost = \left\{ \begin{array}{ll} \frac{n}{2} + t(i) & \text{otherwise (case 1)} \\ \sum_{i=1}^{n} t(i) = n + \sum_{k=0}^{\frac{\lfloor \log_2(n-1) \rfloor}{k}} 2^k \\ = n + 2^{\lfloor \log_2(n-1) \rfloor + 1} - 1 \\ \leq n + 2n \cdot 2^{\lfloor \log_2(n-1) \rfloor} \leq n-1 < n \\ = 3n \\ \text{total cost of } n \text{ operations } \leq 3n \\ \text{amortized cost } = \frac{\text{total cost}}{n} \leq 3 = O(1) \end{array} \right.

charging method

\[
\begin{array}{cccccccc}
  i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
  t(i) & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 9 \\
\end{array}
\]

question: how many cheap operations are there between two expensive operations \(2^{k-1}, 2^k\)

\[
2^k - (2^{k-1}) - 1 = 2^{k-1} - 1
\]

question: how much time does each operation need to pay

\[
\frac{2^{k+1}}{2^{k-1}} \approx 2
\]
\[ \approx \frac{2^k+1}{2^{k-1}-1} \approx 2 \]

Plan: each cheap operation Pay\(1\) for itself
Pay\(2\) to the bank.

For expensive operation \(i = 2^k + 1\)
Use \(2 \times (2^{k-1} - 1)\) from bank
Pay \(3\) for itself

\[ 2 \times (2^{k-1} - 1) + 3 = 2^{k+1} = f(i) \]

Amortized running time \(\leq 3\)

Potential argument

\[ \Phi (\text{dynamic array}) = 2 \times \text{length} - \text{capacity} + 1 \]

Normal operation \(i\)

\[ X_i \rightarrow X_{i+1} \]

\(X\): state of data structure
\(X\) has two elements \(X\): length \(X\): capacity

\[ \begin{align*}
X_i & : X_{i+1} \\
\text{length} & : \text{length} + 1 \\
\text{capacity} & : \text{capacity}
\end{align*} \]

\[ \Phi (X_i) + 2 = \Phi (X_{i+1}) \]

Amortized cost = real cost - \[\Phi (X_i) + \Phi (X_{i+1})\]

Expensive operation

\(i = 2^k + 1\)

\[ \begin{align*}
X_i & : X_{i+1} \\
\text{length} & : 2^k \quad 2^{k+1} \\
\text{capacity} & : 2^k \quad 2^{k+1} \\
\Phi & : 2^{k+1} \quad 3
\end{align*} \]