COMPSCI 330 Lecture 14 Amortized Analysis

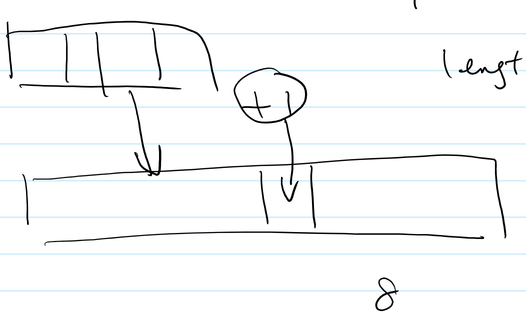
Wednesday, October 19, 2016 2:59 PM

- Basic Concept
- Example: Dynamic Array
- Techniques: Aggregate, accounting (charging), potential

- "amortize": paying off debt/mortgage
- idea: Certain steps in algorithm may be very expensive
  if these steps don't happen often, total running time is bounded.
- problem: dynamic array
  recall: java has "arrayList", vector is a growing array
  vector supports "append" operation: add 1 element to end of vec.
  Goal: - do not want to waste a lot of space
  - make sure append operation is not very slow.

- Solution: initially 1 element, space of size 1
  append( )
  if length $= 2^i$ (length $= 1, 2, 4, 8, \ldots$)
  allocate new space of size 2^{i+1}
  copy the current 2^i elements to the new space
  put new element in (2^i+1) location
  free the old space
  else
  put new element into first free space.

- Example



length $4 \to 5$

- Space: If there are n elements in array, it has $\le 2n$ space.
- running time: append operation can take $O(n)$ time
  naive analysis: if we do n append operations

it can take $O(n^2)$ time.

- Aggregate : take the sum of running times

step 2^i takes 2^i operations

other steps take 1 operation

$$T(n) = \underbrace{\sum_{i=0}^{\lfloor \log_2 n \rfloor} 2^i}_{\text{"heavy" operation}} + \underbrace{\sum_{\substack{i \neq 2^j \\ i=1}}^{n} 1}_{\text{"light" operations}}$$

$$\leq 2n + n \leq 3n$$

Simple, but not very general.

- accounting (charging)

idea: save "money" for light operations, pay money for heavy operations

observation: "heavy" operation comes at time 2^i

before that, $\underbrace{2^{i-1}+1, 2^{i-1}+2, \ldots, 2^i-1}_{2^{i-1}-1 \text{ light operations}}$ are light operations

if we save 2 time units per light operation, can "charge" 2^i-2 to these light operation, and the remaining 2 is charged to the current operation.

- For every operation: time paid + time saved ≤ 3

Total runtime $\leq 3n$

- potential argument

Keep a potential function Φ, $\Phi \geq 0$

amortized cost for an operation = actual cost − current potential + new potential.

$= \text{actual cost} - (\Phi_{current} - \Phi_{new})$

$\Phi = 2n - m$

$= \text{actual cost} + (\Phi_{current} - \Phi_{new})$

$$\Phi = \underset{\uparrow\ \#\text{elements}}{2n} - \underset{\uparrow\ \#\text{of space}}{m}$$

"heavy" step: before $n = 2^i$ $m = 2^i$ $\Phi_{current} = 2^i$

after $n = 2^i + 1$ $m = 2^{i+1}$ $\Phi_{new} = 2$

actual cost $= 2^i$

amortized cost $= 2$

"light" step before n m

after $n+1$ m $\Phi_{current} - \Phi_{new} = -2$

actual cost $= 1$

amortized cost $= 3$

Claim: $\sum$ actual cost $= \sum$ amortized cost $+ \Phi_{init} - \Phi_{end}$

Proof: Suppose alg had k steps.

Let Φ_i be the potential function after step i

for step i actual cost $=$ amortized cost $+ (\Phi_{i-1} - \Phi_i)$

take the sum: Φ terms cancel

$$\sum \text{actual cost} = \sum \text{amortized cost} + (\Phi_0 - \cancel{\Phi_1}) + (\cancel{\Phi_1} - \cancel{\Phi_2}) + \cdots + (\cancel{\Phi_{k-1}} - \Phi_k)$$

$$= \sum \text{amortized cost} + \Phi_0 - \Phi_k$$

therefore $\sum$ actual cost $= \sum$ amortized cost $+ \Phi_{init} - \Phi_{end}$ $\square$

For dynamic array $\Phi_{init} = 1$

$$\sum \text{actual cost} \underset{\uparrow}{\leq} \sum \text{amortized cost} + 1 = O(n)$$

because $\Phi_{end} \geq 0$

- Simple examples we've seen that uses the idea of "amortize"
  - DFS $O(n+m)$
  - merge sort $O(n)$