CSCI 136
Data Structures &
Advanced Programming

Lecture 12

Spring 2020

Bill and Dan

Administrative Details

- Lab 4 Due Monday
  - Coordinate with your partner to program *in person*
- Colloquium Today at 2:30 in Wege
  - Nate Derbinski – **Adventures in Hybrid Architectures for Intelligent Systems**
- Upcoming power outages will affect our labs
  - Look for announcements on Piazza and plan accordingly
  - We have no control over this… but we can try to minimize impact with planning

Last Time

- Other List implementations
  - Compared Linked Lists (single, double, circular) with Vectors
  - No clear winner in accross-the-board performance
    - Linked lists are a "pay as you go" recursively defined structure
    - Vectors are random access, but have "bursty" add costs
      - Cost to add depends on (hidden) internal array: hard to predict

Today's Plan

- List Tradeoffs: Revisit Vector Growth
  - Additive Growth: $O(n^2)$
  - Multiplicative Growtn: $O(n)$
- Prove these costs using induction
  - Mathematical cousin to recursion
- Use induction to reason about code

Vectors: Add Method Complexity

Suppose we grow the Vector's array by a fixed amount `d`.

How long does it take to add `n` items to an empty Vector?

- The array will be copied each time its capacity needs to exceed a multiple of d
  - At sizes `0d, 1d, 2d, … , (n/d)d.`
- Copying an array of size `k*d` takes `c*k*d` steps for some constant `c`, giving a total of:

$$\sum_{k=1}^{n/d} ckd \quad = cd \sum_{k=1}^{n/d} k \;= cd \left(\frac{n}{d}\right)\left(\frac{n}{d} + 1\right)/2 \;= O(n^2)$$

Vectors: Add Method Complexity

As a concrete example, let's choose our fixed growth amount, d, to be 1.

- The array will be copied each time its capacity needs to exceed a multiple of 1
  - Grow at sizes `0, 1, 2, … , n-1.`

We can use induction to prove that growing by 1 will result in $O(n^2)$ work.

Induction

- The mathematical cousin of recursion is induction
- Induction is a proof technique

Induction proofs have three key components:

- Base case(s): show the claim is true for all base cases
- Assume: assume the claim is true for some problem size
- Show: using your assumption, show that you can you can prove your claim for the next problem size

Mathematical Induction

- Additive Growth: Prove that for every n ≥ 0

$$P_n : \sum_{i=0}^{n} i = 0 + 1 + \ldots + n = \frac{n(n+1)}{2}$$

- Proof by induction:
  - Base case: P_n is true for n = 0
  - Assume: P_n is true for some n≥0
  - Show: If P_n is true for some n≥0, then P_{n+1} is true.

    (Using a smaller version of the problem, we solve a larger version)

Mathematical Induction

$$P_n : \sum_{i=0}^{n} i = 0 + 1 + \ ... + n = \frac{n(n+1)}{2}$$

- Prove the base case: P_n is true for $n = 0$
  - Just check it! Substitute 0 into the equation.

    $0 = 0(1)/2$

- Assume the inductive hypothesis: P_n is true for some $n \geq 0$
- Then use assumption to show that P_{n+1} is true.

Write out P_{n+1} and target equality

$$P_{n+1}: 0 + 1 + \ ... + n + (n+1) = \frac{(n+1)\big((n+1)+1\big)}{2} = \frac{(n+1)(n+2)}{2}$$

This is P_n!

$$\frac{n(n+1)}{2} + (n+1) = \frac{n(n+1) + 2n+2)}{2} = \frac{n^2+3n+2}{2} = \frac{(n+1)(n+2)}{2}$$

- First equality holds by assumed truth of P_n!

Vectors: Add Method Complexity

Suppose we instead grow the Vector's array by doubling.

How long does it take to add `n` items to an empty Vector?

- The array will be copied each time its capacity needs to exceed a power of 2
  - At sizes `0, 1, 2, 4, 8 …, n/2`
- The total number of elements are copied when `n` elements are added is:
  - `1 + 2 + 4 + ... + n/2 = n-1 = O(n)`

- Very cool! Let's show this is the case using induction.

Mathematical Induction

- Prove: $$\sum_{i=0}^{n} 2^i = 2^0 + 2^1 + 2^2 + \ldots + 2^n = 2^{n+1} - 1$$
  - Base case: P_n is true for n = 0
  - Assume: P_n is true for some n≥0
  - Show: If P_n is true for some n≥0, then P_{n+1} is true.

- (Practice Problem) Prove:

$$0^3 + 1^3 + \ldots + n^3 = (0 + 1 + \ldots + n)^2$$

What about Recursion?

- What does induction have to do with recursion?
  - Same form!
    - Base case
    - Inductive case that uses simpler form of problem
- We can prove things about recursive functions using induction.
- Example: Let's use induction to prove that fact(n) requires n multiplications
  - Base case: fact(0) requires 0 multiplications
  - Assume: fact(n) requires n multiplications for some $n \geq 0$
  - Show: if fact(n) is true for some $n \geq 0$, then fact(n+1) is true

fact(n) requires n multiplications

- Prove that fact(n) requires n multiplications
  - Base case: n = 0 returns 1
    - 0 multiplications
  - Inductive Hypothesis: Assume true for all k<n, so fact(k) requires k multiplications.
  - Prove, from simpler cases, that the *n*th case holds
    - `fact(n)` performs `1` multiplication (`n*fact(n-1)`).
    - We know `fact(n-1)` requires `n-1` multiplications (by our I.H.)
    - `1+n-1 = n`
      - therefore `fact(n)` requires `n` multiplications.

Conclusions

- Induction is a valuable proof technique
  - We've shown that vector doubling is superior to fixed-growth allocations
    - Sped up our runtime to *amortized* O(1) adds!
  - We've proven things about recursive functions
    - Convinced ourselves that our runtime is good!

Counting fib() method calls

- Prove that `fib(n)` makes at least `fib(n)` calls to `fib()`
  - Base cases: n = 0: 1 call; n = 1; 1 call
  - Inductive Hypothesis: Assume that for some n≥2, fib`(n-1)` makes at least `fib(n-1)` calls to `fib()` and `fib(n-2)` makes at least `fib(n-2)` calls to `fib()`.
  - Claim: Then `fib(n)` makes at least `fib(n)` calls to `fib()`
    - 1 initial call: fib(n)
    - By induction: At least fib(n-1) calls for fib(n-1)
    - And as least fib(n-2) calls for fib(n-2)
    - Total: 1 + fib(n-1) + fib(n-2) > fib(n-1) + fib(n-2) = fib(n) calls
  - Note: Need two base cases!