Randomized Algs. & Amortized Analysis

CS3230: Design and Analysis of Algorithms

Roger Zimmermann

National University of Singapore

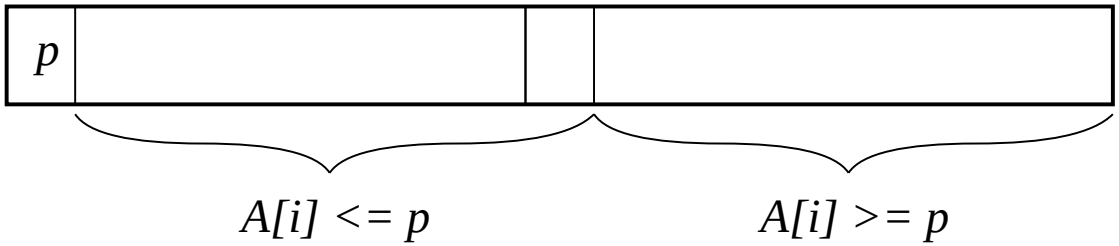
Spring 2017

Randomized Algorithms & Amortized Analysis

- Randomized algorithms
  - A randomized algorithm is an algorithm that employs a degree of randomness as part of its logic.
  - Typically uniformly random bits are used as auxiliary input to guide the behavior of the algorithm to achieve good performance in the average case, and avoiding the worst case.
- Amortized analysis
  - Amortized analysis is a method for analyzing a given algorithm's time complexity when looking at the worst-case run time per operation can be too pessimistic.

Randomized Quicksort

- Recall from Lecture 4 on Quicksort:
  - Select a pivot (partitioning element) – e.g., the first element.



- Worst case analysis:
  - The worst case occurs when all the partitions divide their subarrays into an empty array and an array with one fewer element than the given subarray.
  - This happens when the array is already sorted: $C(n) \in \Theta(n^2)$.

Randomized Quicksort

- Goal: Want to avoid (or reduce the chance of) worst case.
- Idea: Use randomization to improve the performance of Quicksort against those worst-case instances.
- How:
  - Replace procedure `Partition()` with `Randomized-Partition()`:

  $\Rightarrow$ Randomly pick one element in the sequence, swap it with the first element (i.e., use it as pivot), and then call `Partition()`.

Randomized Quicksort

- `Randomized-Partition(`A, p, q`)`, which works on $A[p \ldots q]$.
  1. $r =$ `Random(`p, q`)` (pick an integer between p and q uniformly at random) and exchange $A[p]$ with $A[r]$.
  2. return `Partition(`A, p, q`)`.

- The remaining parts of the algorithm stay the same.

Randomized Quicksort

- THEOREM (Randomized Quicksort)

  Given any input sequence A of length n, `Randomized-Quicksort(`$A[1 \ldots n]$`)` has expected running time $O(n \log n)$.

- This of course implies that the worst-case expected running time of `Randomized-Quicksort` is $O(n \log n)$.
- Proof is quite involved and lengthy, and not given here.

Randomized Algorithms

- Intuition: Randomization is used to, with high probability, avoid the worst case runtime (class of Las Vegas algorithms).

- Note: There are also probabilistic algorithms, which, depending on the random input, have a chance of producing an incorrect result or fail to produce a result either by signalling a failure or failing to terminate (class of Monte Carlo algorithms).
  - Ex.: Monte Carlo simulation (e.g., for numerical integration).

Amortized Analysis

- This technique is most commonly the case with data structures, which have state that persists between operations.
- The basic idea is that a worst case operation can alter the state in such a way that the worst case cannot occur again for a long time, thus “amortizing” its cost.
- The technique was first formally introduced by Robert Tarjan in his 1985 paper *Amortized Computational Complexity*.

Amortized Analysis – Ex. 1

- Ex.: **Dynamic Array** (e.g., in Java)
- Let's say we start with an array A of size 4. Then it would take constant time to push 4 elements into A.
- Pushing element 5 into A will take longer because a new array would have to be allocated (say of size 8), the 4 original elements copied, and finally element 5 pushed into it.
- Usually pushing an element takes $O(1)$ time, except when the array needs to be doubled. Then it takes $O(n)$ time.
- If we do this for large n's we will find that on average it still takes $O(1)$ time to add an element.

Amortized Analysis – Ex. 1

ALGORITHM $Array - Insert$

IF array does not exist: initialize array size to $m = 1$.
IF (number of elements in array $= m$)
    Generate new array of size $2m$.
    Re-insert m old elements into new array.
    Deallocate old array.
ENDIF
Insert x into array.

Amortized Analysis – Ex. 1

- Sequence of n insert operations takes $O(n)$ time.
- Let c_i be the cost of the i^{th} insert:

$$c_i = \begin{cases} i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{otherwise} \end{cases}$$

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\log_2 n} 2^j = n + (2n-1) \leq 3n$$

- Therefore, n insertions will on average take $O(n)$ time, even though some insertions might take much longer than constant time.

Amortized Analysis – Ex. 2

- Ex.: **AVL Trees**
- Recall that when performing element insertions, in some cases AVL trees need to perform rotations, i.e., execute a rebalancing of the structure.
- Rotations only happen occasionally, not with every insertion operation.
- Therefore, rotations can be “amortized” across all the insertions.
- The benefit of performing the rotations is that the search for a key will be faster on average.

Amortized Analysis – Other Examples

- **Splay Trees**. A sequence of M operations on an n-node splay tree takes $O(M \log n)$ time. Splay trees are efficient when multiple searches are performed for the same element, because it is moved to the top of the tree through *tree rotations*.
- **Red-Black Trees**. Suppose we color each edge of a binary search tree either red or black. The color is conveniently stored in the lower node of the edge. Such a edge-colored tree is a red-black tree if

  (1) there are no two consecutive red edges on any descending path and every maximal such path ends with a black edge;

  (2) all maximal descending paths have the same number of black edges.