CMPT307: Tutorial 1

Week 6-3

Xian Qiu

Simon Fraser University

xianq@sfu.ca

Basics

- asymptotic notations: $O, \Omega, \Theta, o, \omega$
- basic formulas (section 3.2):

$$\log_a n = \frac{\log_b n}{\log_b a}, \quad c^{\log_a n} = n^{\log_a c}, \quad H_n = \sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1)$$

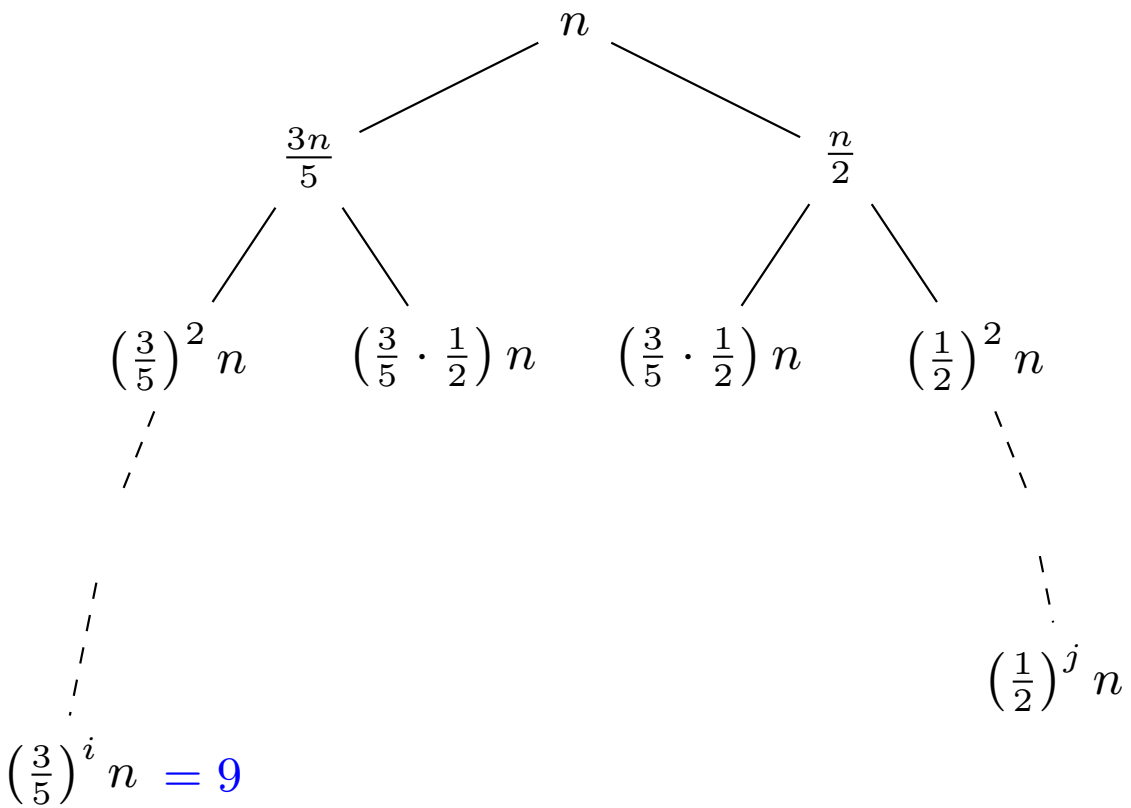
- recurrences: master theorem (section 4.5)
- estimation of tree height

Exercise: tree height

Consider the recursion tree of the following recurrence. What is the tree height?

$$T(n) = \begin{cases} T(3n/5) + T(n/2) + \Theta(1), & n > 9 \\ \Theta(1), & n \leq 9 \end{cases}$$

Basics



$$\left(\frac{5}{3}\right)^i = \frac{n}{9} \Rightarrow h = i = \log_{5/3}\frac{n}{9} = O(\log n)$$

Exercise

Find a lower bound on the tree height h of a k-ary tree of n nodes.

- $\#$ nodes at depth i is at most k^i
- total $\#$ nodes $= n$

$$n \leq 1 + k + k^2 \cdots + k^h = \frac{k^{h+1} - 1}{k - 1}$$

$$(k-1)n + 1 \leq k^{h+1}$$

$$h \geq \log_k[(k-1)n + 1] - 1$$

Algorithms

correctness and efficiency

algorithms for dictionary operations

- ▷ correctness often trivial
- ▷ except for the deletion and insertion for red-black trees

divide-and-conquer

- ▷ merge sort (section 2.3)
- ▷ maximum subarray, matrix multiplication (chapter 4)
- ▷ quicksort (chapter 7)
- ▷ selection in linear time (chapter 9)

Inductive Proofs

- running time of INORDER-TREE-WALK of binary search trees (chapter 12)
- substitution method for recurrences (section 4.3)
- correctness of insertion sort and merge sort (chapter 2)
- correctness of the insertion and deletion for red-black trees (chapter 13)
- RANDOMIZE-IN-PLACE computes a uniform random permutation (chapter 5)
- Q7 of H2
- Q4 of H3

Inductive Proofs

Paradigm of Inductive Proof

1. claim holds for initial step, say $i = 0$;
2. make inductive hypothesis for $i - 1$ steps with $i \geq 1$;
3. show that the claim holds for the ith step.

- ▷ to prove the correctness of an algorithm, it is often convenient to prove a loop invariant first by induction
- ▷ then the correctness follows from the loop invariant

Example: Insertion Sort

INSERTION-SORT(A)

```
1 for j = 2 to n do
2     key = A[j];
      // insert A[j] to sorted sequence A[1..j − 1]
3     i = j − 1;
4     while i > 0 and A[i] > key do
5         A[i + 1] = A[i];
6         i = i − 1;
7     A[i + 1] = key;
```

Loop invariant

At the start of each iteration, $A[1..j-1]$ contains the orignal elements of $A[1..j-1]$ in sorted order.

Example: Insertion Sort

- ▷ at termination, we have $j = n + 1$ thus insertion sort yield sorted sequence of A

  now it suffices to prove the loop invariant

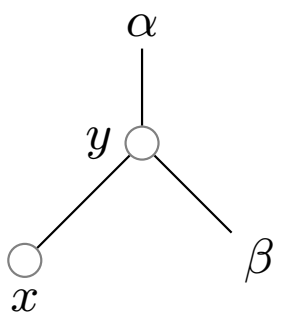
- ▷ initialization: $j = 2$ and $A[1]$ is sorted
- ▷ inductive hypothesis: assume loop invariant holds for $2, \ldots, j$, now consider $j + 1$:
  - ∗ $A[1..j]$ is sorted (by inductive assumption)
  - ∗ the algorithm will insert $A[j + 1]$ to $A[1..j]$, thus $A[1..j + 1]$ will be sorted
  - ∗ clearly, $A[1..j + 1]$ contains the original elements of $A[1..j + 1]$

Proofs by Contradiction

Example: Q1 of H2

Let T be a binary search tree whose keys are distinct. Let x be a leaf node and y be its parent. Show that y is a successor of x or x is a successor of y.

- ▷ assume x is a left child (similar for the other case)

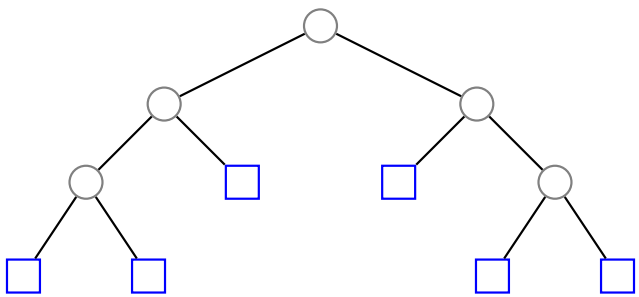


- ▷ to show y is a successor of x
- ▷ assume y is not and let z be the successor of x
- ▷ $z \neq y$, hence z is in α or β and there will be contradiction

Proofs by Contradiction

Example

Given a binary tree with n nodes, show that we can add $n + 1$ nodes such that they are all leaves.

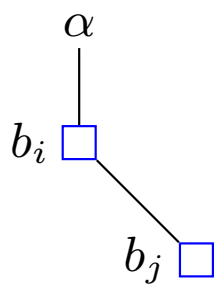


Proofs by Contradiction

- assign the n nodes with values $a_1 < a_2 < \ldots < a_n$ such that it yields the resulting binary search tree is the same as before
- now insert values $b_0, \ldots, b_n$ with

$$b_{i-1} < a_i < b_i, \quad \text{for } i = 1, \ldots, n$$

- now prove that $b_0, \ldots, b_n$ are all leaves
- assume b_i is not leave, then it has a child, say b_j



- no a_k between b_i and b_j for any k, contradiction!
  (b_j is a successor of b_i or the other way around)

Probabilistic Analysis

- average searching time for using hash table: hash with chaining and open addressing (chapter 11)
- chapter 5
- running time of QUICKSORT (chapter 7)
- running time of BUCKETSORT (chapter 8)
- running time of RANDOMIZED-SELECT (chapter 9)
- Q5 of H1

Example

Throw n times a fair dice, which takes values in $\{1, \ldots, 6\}$. What is the total sum in expectation?

$$S := \text{total sum}$$

$$X_{ij} = \mathrm{I}\{\text{get value } j \text{ at time } i\}$$

$$S = \sum_{i=1}^{n} \sum_{j=1}^{6} X_{ij} \cdot j$$

$$\mathbb{E}[S] = \sum_{i=1}^{n} \sum_{j=1}^{6} \mathbb{E}[X_{ij}] \cdot j = \sum_{i=1}^{n} \sum_{j=1}^{6} \frac{j}{6} = \frac{7n}{2}$$