강의 6: 균형 이진 탐색 트리

강의 개요

- 균형을 이루는 것의 중요성
- AVL 트리
  - 정의와 균형
  - 회전
  - 삽입
- 다른 균형 트리들
- 일반적인 데이터 구조들
- 하한

복습: 이진 탐색 트리 (BSTs)

- 루트가 있는 이진 트리
- 각 노드는 다음을 가짐:
  - 키
  - 왼쪽 포인터
  - 오른쪽 포인터
  - 부모 포인터

그림 1. 참조

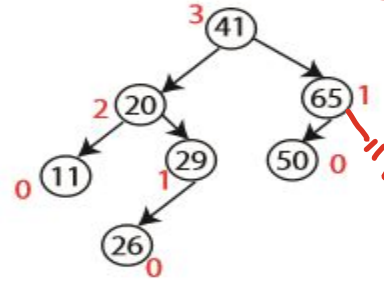


그림 1: 이진 탐색 트리에서 노드의 높이

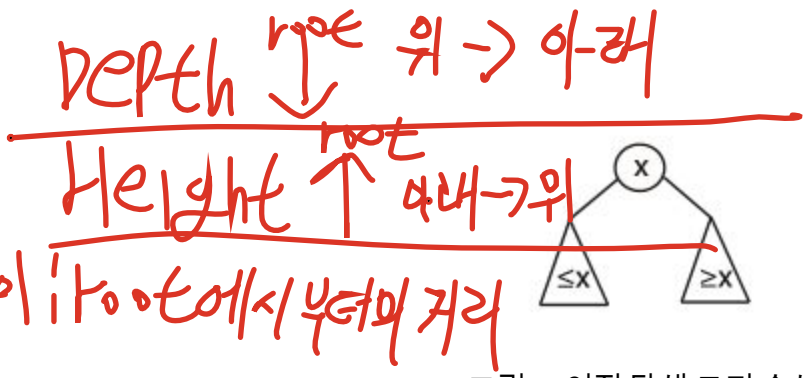


그림 2: 이진 탐색 트리 속성

• 이진 탐색 트리 속성 (그림 2. 참조).

• 노드의 높이 = 단말 노드까지의 가장 긴 하향 경로의 길이 (자세한 내용은 CLRS B.5 참조).

균형을 이루는 것의 중요성:

• 이진 탐색 트리는 삽입, 삭제, 최솟값, 최댓값, 다음으로 큰 값 찾기, 다음으로 작은 값 찾기 등을 O(h) 안에 실행할 수 있음. 이 때 h = 트리의 높이 (= 루트의 높이)

• h는 lg n과 n 사이에 있음: 그림 3.

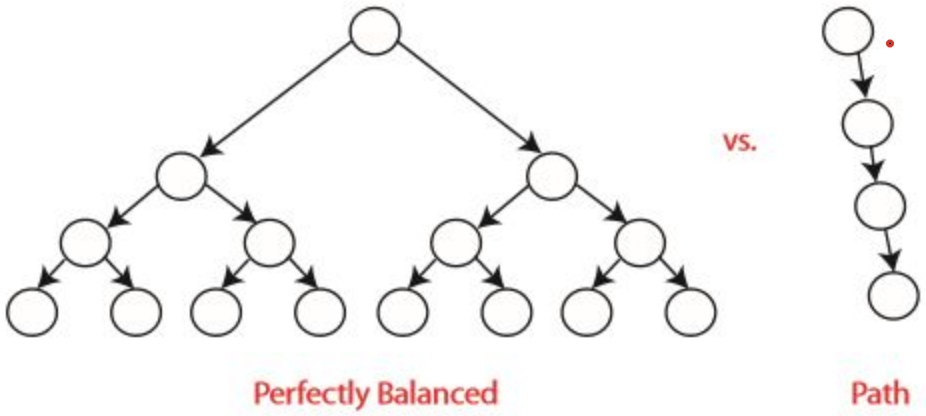


그림 3: 이진 탐색 트리 균형 맞추기

• 균형 이진 탐색 트리는 h = O(lgn)로 유지됨 ⇒ 모든 작업은 O(lgn) 시간 안에 실행

AVL 트리: Adel'son-Vel'skii & Landis 1962

모든 노드에 대해, 왼쪽과 오른쪽 자식들의 높이 차는 최대 ±1

- nil 트리는 -1의 높이로 취급.
- 각 노드는 자신의 높이를 저장. (자료 구조 확장) (서브 트리 크기와 유사) (높이 대신 높이의 차를 저장할 수도 있음.)

이 내용은 그림. 4 에 그려져 있음.

왼쪽 자식의 높이 k-1과 오른쪽 자식의 높이 k가 있을 때
둘이 차이가 최대 1이 되어야 한다.
|k - (k-1)| <= 1

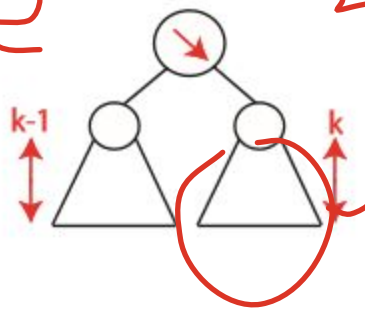


그림 4: AVL 트리 개념

균형:

Worst when every node differs by 1 — let N_h = (min.) # nodes in height-h AVL tree

$$\begin{aligned} \Longrightarrow N_h &= N_{h-1} + N_{h-2} + 1 \\ &> 2N_{h-2} \\ \Longrightarrow N_h &> 2^{h/2} \\ \Longrightarrow h &< 2 \lg N_h \end{aligned}$$

Alternatively:

$N_h > F_h$ (nth Fibonacci number)

- In fact $N_h = F_{n+1} - 1$ (simple induction)
- $F_h = \frac{\varphi_h}{\sqrt{5}}$ rounded to nearest integer where $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$ (golden ratio)
- $\Longrightarrow$ max. $h \approx \log_\varphi n \approx 1.440 \lg n$

AVL 삽입:

1. 간단한 이진 탐색 트리로 삽입
2. 트리를 따라 올라가면서 처리, AVL 속성 회복 (올라가면서 높이 값도 갱신).

각 단계:

- x가 AVL을 위반하는 가장 낮은 노드라고 가정
- x가 오른쪽-무거움이라고 가정 (왼쪽은 대칭임)
- if: x의 오른쪽 자식이 오른쪽-무거움이거나 균형이라면, 그림 5 의 단계를 수행

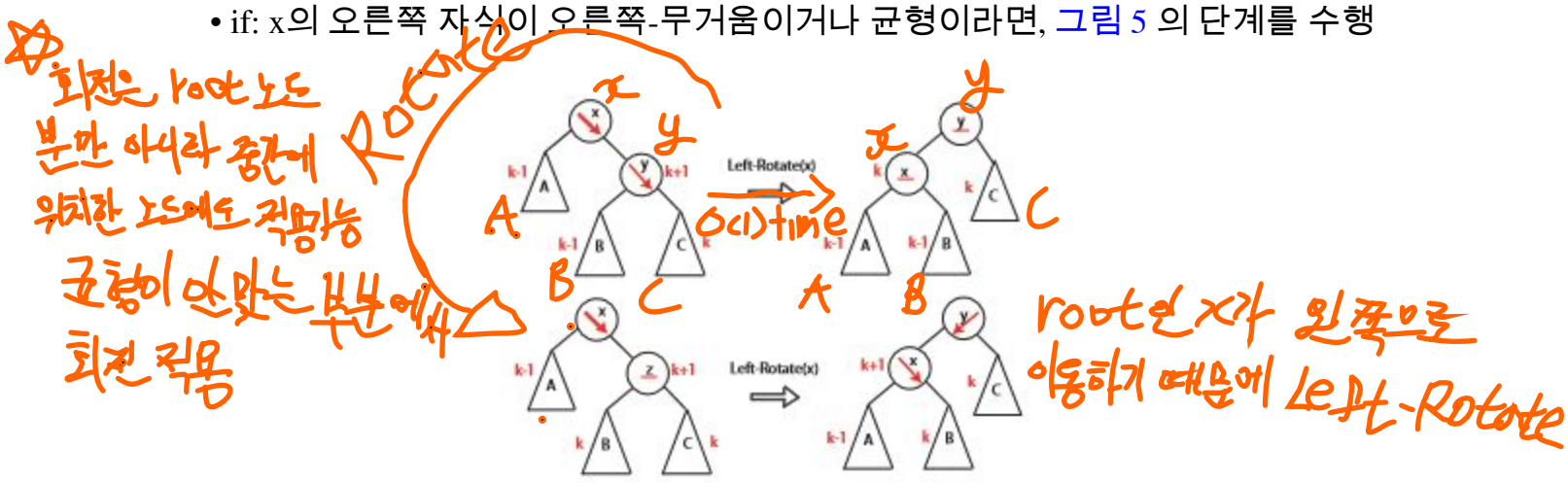


그림 5: AVL 삽입 균형 맞추기

- else: 그림 6 의 단계를 수행

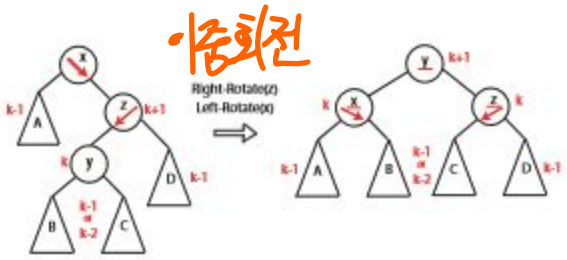


그림 6: AVL 삽입 균형 맞추기

- 그다음에 x의 조부모, 증조부모(...)까지 올라가며 계속함.

예제: AVL 삽입 과정의 구현 예제는 그림 7에 그려져 있음.

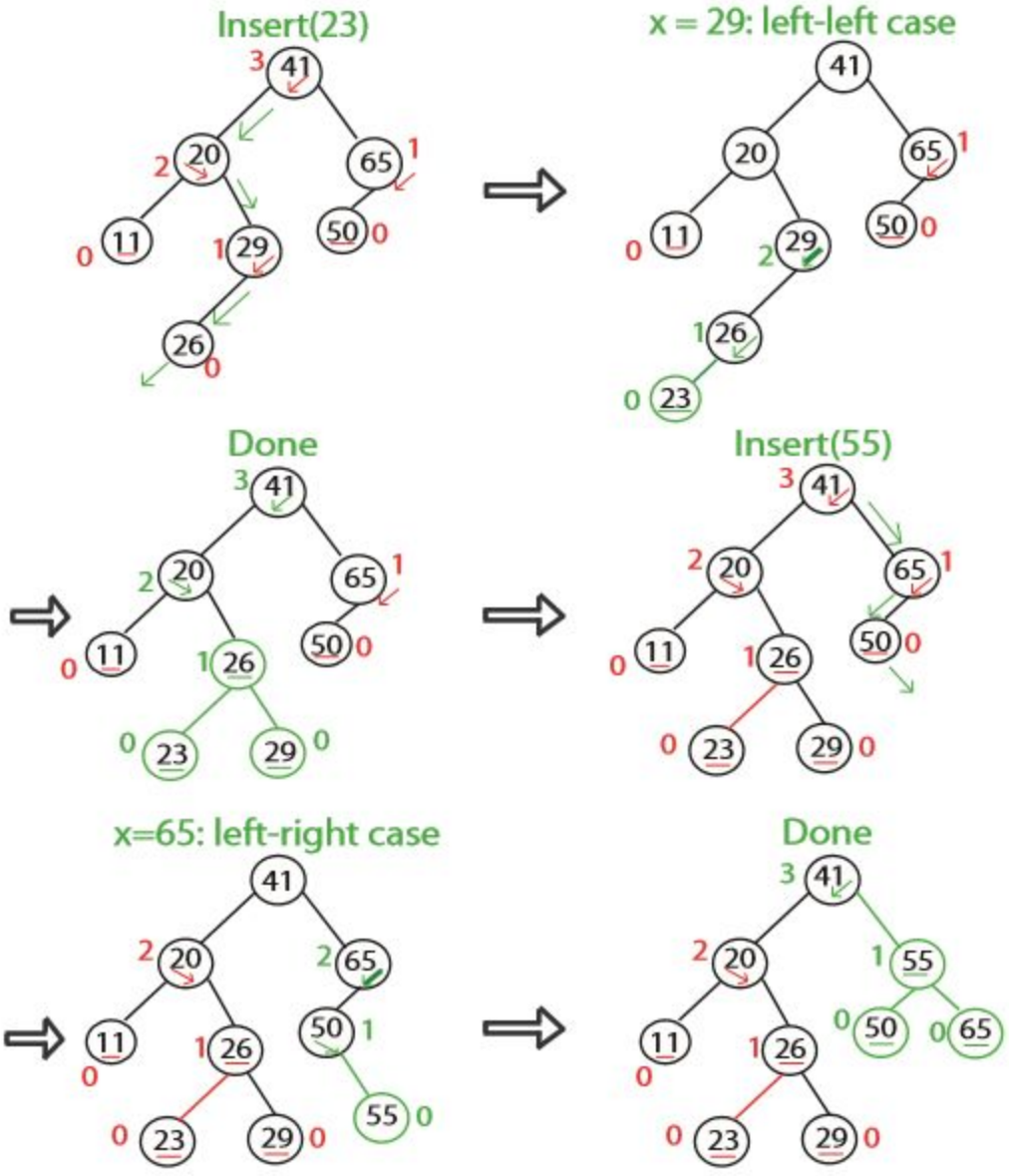


그림 7: AVL 트리 삽입 과정에 대한 삽화

댓글 1. 일반적으로, Insert를 하기 전에 여러 번의 회전이 필요함.

댓글 2. Delete(-min)도 비슷 — 더 어렵지만 가능은 함.

AVL 정렬:

- AVL 트리에 각 아이템 삽입 $\Theta(n \lg n)$
- 중위 순회 $\frac{\Theta(n)}{\Theta(n \lg n)}$

균형 탐색 트리:

많은 종류의 균형 탐색 트리가 있음.

AVL Trees Adel'son-Velsii and Landis 1962
B-Trees/2-3-4 Trees Bayer and McCreight 1972 (see CLRS 18)
BB[α] Trees Nievergelt and Reingold 1973
Red-black Trees CLRS Chapter 13
(A) — Splay-Trees Sleator and Tarjan 1985
(R) — Skip Lists Pugh 1989
(A) — Scapegoat Trees Galperin and Rivest 1993
(R) — Treaps Seidel and Aragon 1996

(R) = 높은 확률로 신속한 의사 결정을 내리기 위하여 난수 사용
(A) = "상환": 여러 작업에 대한 비용 합산 ⇒ 평균적으로 속도가 빠름
예를 들어, Splay Trees는 연구 주제 — 6.854 (고급 알고리즘) and 6.851 (고급 데이터 구조) 참조

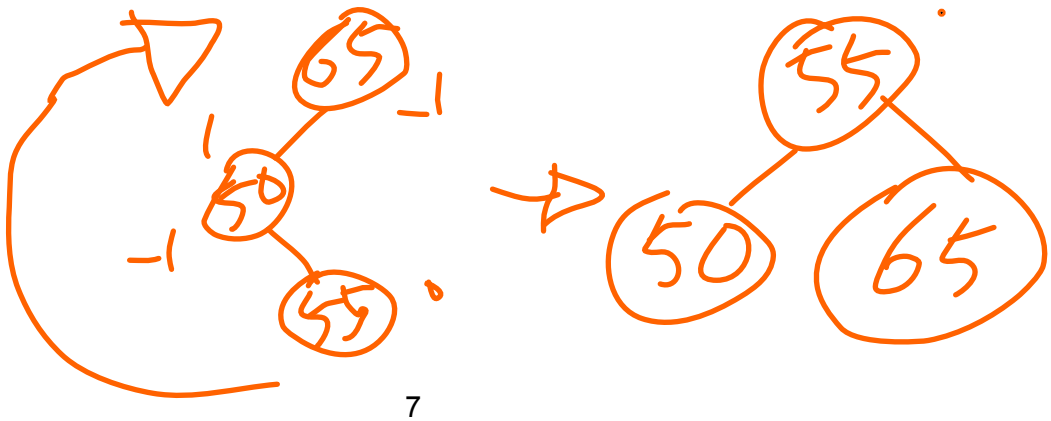
큰 그림:

추상 자료형 (ADT): 인터페이스 지정
vs.
자료 구조 (DS): 각 연산의 알고리즘

하나의 추상 자료형을 위한 여러 개의 자료 구조가 있음.
이 과목의 후반부에서 다루는 "힙" 우선순위 큐가 하나의 예임.

Priority Queue ADT	**heap**	**AVL tree**
Q = new-empty-queue()	$\Theta(1)$	$\Theta(1)$
Q.insert(x)	$\Theta(\lg n)$	$\Theta(\lg n)$
x = Q.deletemin()	$\Theta(\lg n)$	$\Theta(\lg n)$
x = Q.findmin()	$\Theta(1)$	$\Theta(\lg n) \to \Theta(1)$

Predecessor/Successor ADT	**heap**	**AVL tree**
S = new-empty()	$\Theta(1)$	$\Theta(1)$
S.insert(x)	$\Theta(\lg n)$	$\Theta(\lg n)$
S.delete(x)	$\Theta(\lg n)$	$\Theta(\lg n)$
y = S.predecessor(x) → next-smaller	$\Theta(n)$	$\Theta(\lg n)$
y = S.successor(x) → next-larger	$\Theta(n)$	$\Theta(\lg n)$



MIT OpenCourseWare
http://ocw.mit.edu

6.006 알고리즘 기초
가을 2011

본 자료 또는 이용 약관 인용에 대한 정보는 다음의 주소를 방문하십시오: http://ocw.mit.edu/terms.

☆AVL 트리의 최대높이구하기

$N_h = 1 + N_{h-1} + N_{h-2}$

$> 2(N_{h-2})$ → 노드가 최악일수록 N_{h-1}이 N_{h-2}보다 크기때문

$= \Theta(2^{h/2})$ → 2의 배수마다 2씩 곱하므로

$h < 2\log n$ → 뒤집은 후 log화