Data structure [A10]

김종규, PhD

2017-05-08

Review

- Types of traversal: preorder, inorder, postorder
  - $\equiv$ prefix, infix, postfix

Outline

- Stack and its limitation
- Iterative algorithm
- Binary search tree operations
  - Review: inorder traversal
  - iterative inorder traversal
  - minimum, maximum
- Adding parent pointer (to be discussed in more detail)
  - successor, predecessor

$\longrightarrow$ $O(n \lg n)$ sorting

Stack

- Abstract data structure
- 배열에 기반한 자료구조로 구현
  - 장점: 간단하다, 빠르다 $O(1)$
  - 단점: 최대 크기에 제약이 있다
- Linked list 에 기반한 자료구조로 구현
  - 장점: 간단하다, 빠르다 $O(1)$
  - 단점: 최대 크기에 제약이 없다

$\longrightarrow$ 모든 stack 은 linked list 로 구현되는가?

Limitations caused by data structure

```
def s(n):
    if n == 0:
        return 0
    else:
        return n + s(n-1)
>>> print(s(10))

55
>>> print(s(1000))
...

RecursionError: maximum recursion depth exceeded ...
```

Function stack in programming languages

- 프로그래밍 언어는 함수의 호출에 stack 을 사용
- 대부분 배열 (array) 에 기반
- 따라서 depth 가 배열의 크기에 제한된다

  $\longrightarrow$ 더 깊은 depth 를 제공하려면?

  $\longrightarrow$ linked list 에 기반한 stack 을 사용한다
- Stack 과 같은 간단한 ADT 도 상황에 따라 적합한 자료구조가 제공되어야 한다.

Inorder traversal (Java)

```
public void inorder(Node tree) {
  if (tree.left != null)
    inorder(tree.left);
  System.out.print(" " + tree.val);
  if (tree.right != null)
    inorder(tree.right);
}
```

$\longrightarrow$ What if `tree == null`

Inorder traversal (simplified)

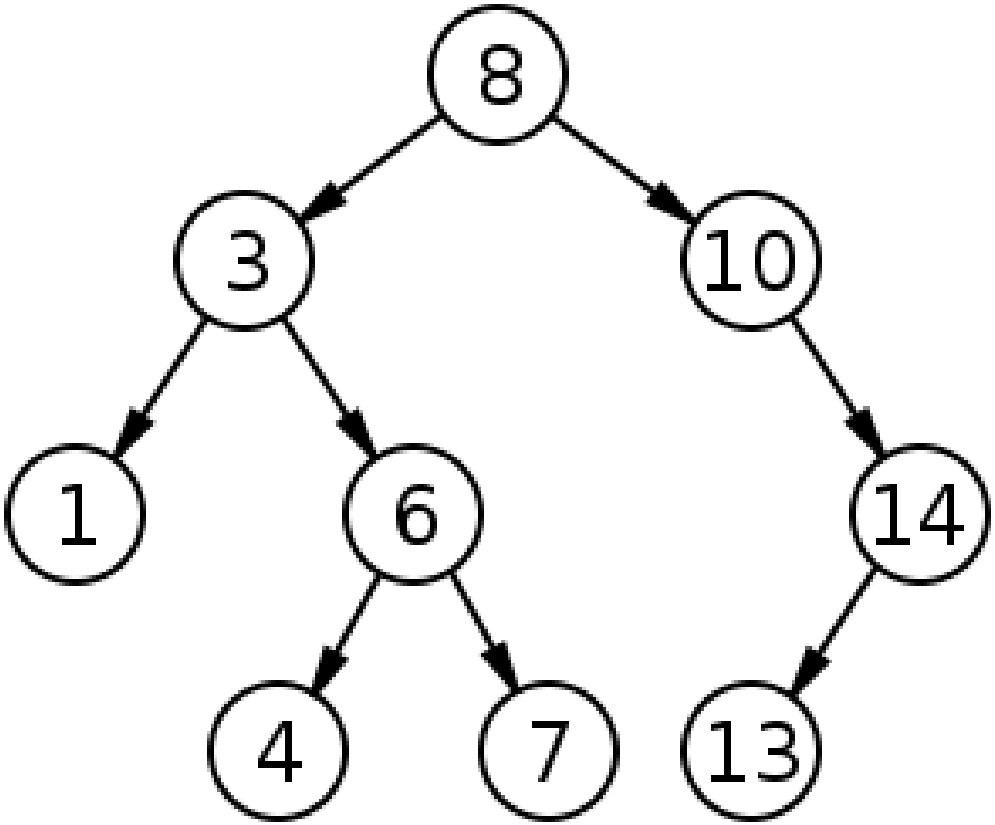

그림: Binary search tree

Inorder traversal (simplified)

```
public void inorder(Node tree) {
  if (tree == null)
    return;
  else {
    inorder(tree.left);
    System.out.print(" " + tree.val);
    inorder(tree.right);
  }
}
```

$\longrightarrow$ Make non-recursive, i.e., iterative

Designing inorder traversal

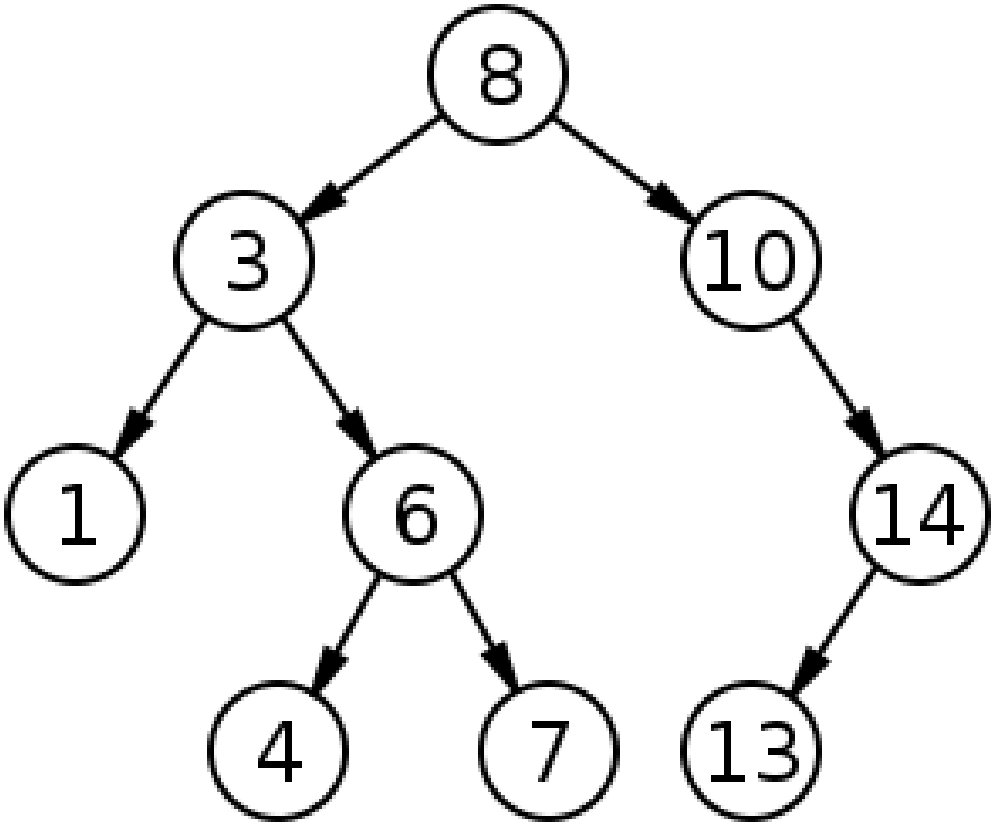

그림: Binary search tree

$\longrightarrow$ Need a stack

Inorder traversal (iterative, Java)

```
public void inorder_iter(Node tree) {
  while(!stk.is_empty() || tree != null) {
    if (tree != null) {
      stk.push(tree);
      tree = tree.left;
    }
    else {
      tree = stk.pop();
      System.out.print(" " + tree.val);
      tree = tree.right;
    }
  }
}
```

Successor

- What is the value next to 3?

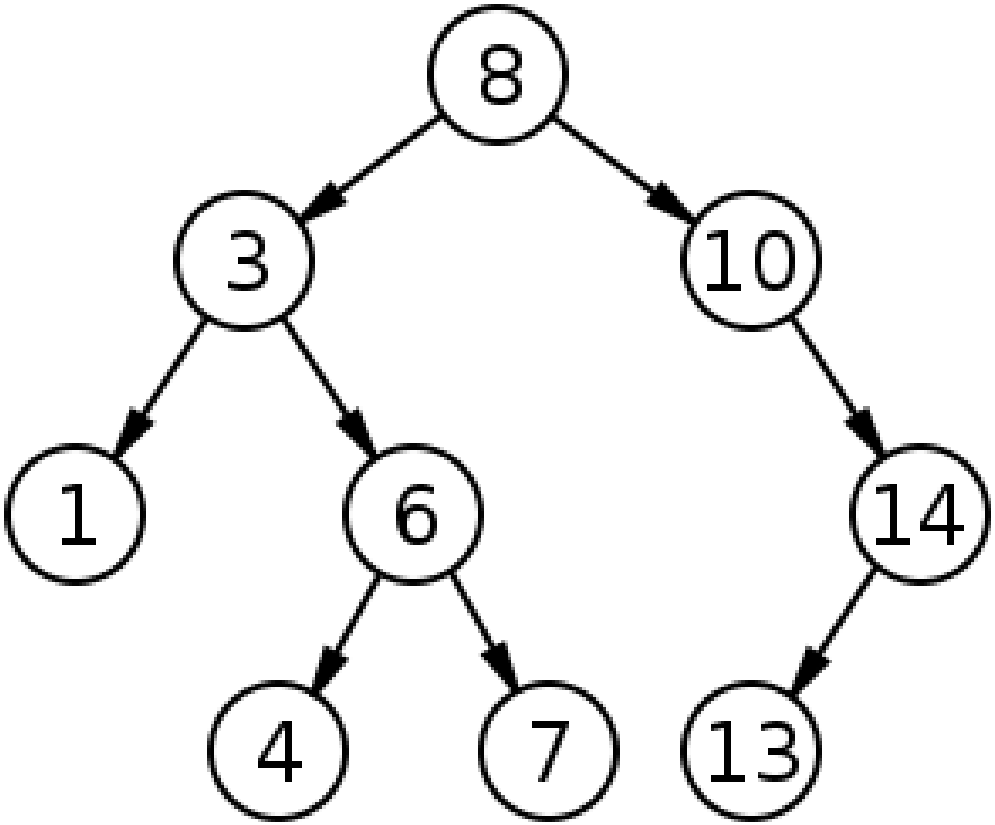

그림: Successor

Successor

- What is the value next to 7?

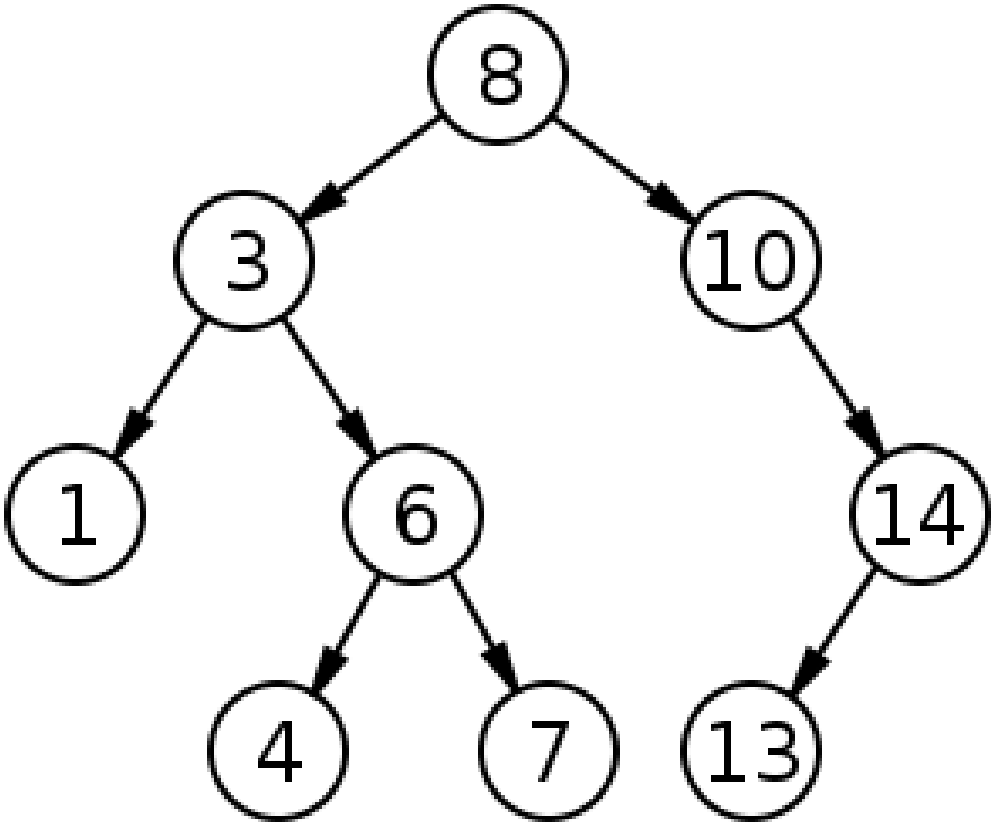

그림: Successor

Assignments

- 9 주차 강의자료 (2/2) 에 지시된 내용을 수행하여 5/11 (목) 23:59 까지 blackboard 에 제출

Iterative algorithms

- Recursion 은 매우 강력한 알고리즘 기술도구이다.
- 그러나 일부 프로그래밍 언어에서는 recursion 을 지원하는데 한계가 있다
  - Recursion 은 프로그래밍 언어의 stack 을 이용한다.
  - 대부분의 프로그래밍 언어에서는 stack 의 자료구조로 배열에 기반한 방식을 사용한다.
  - tree 의 insert, inorder traversal 등에서 자료구조의 한계에 도달할 수 있다

Tree search tree

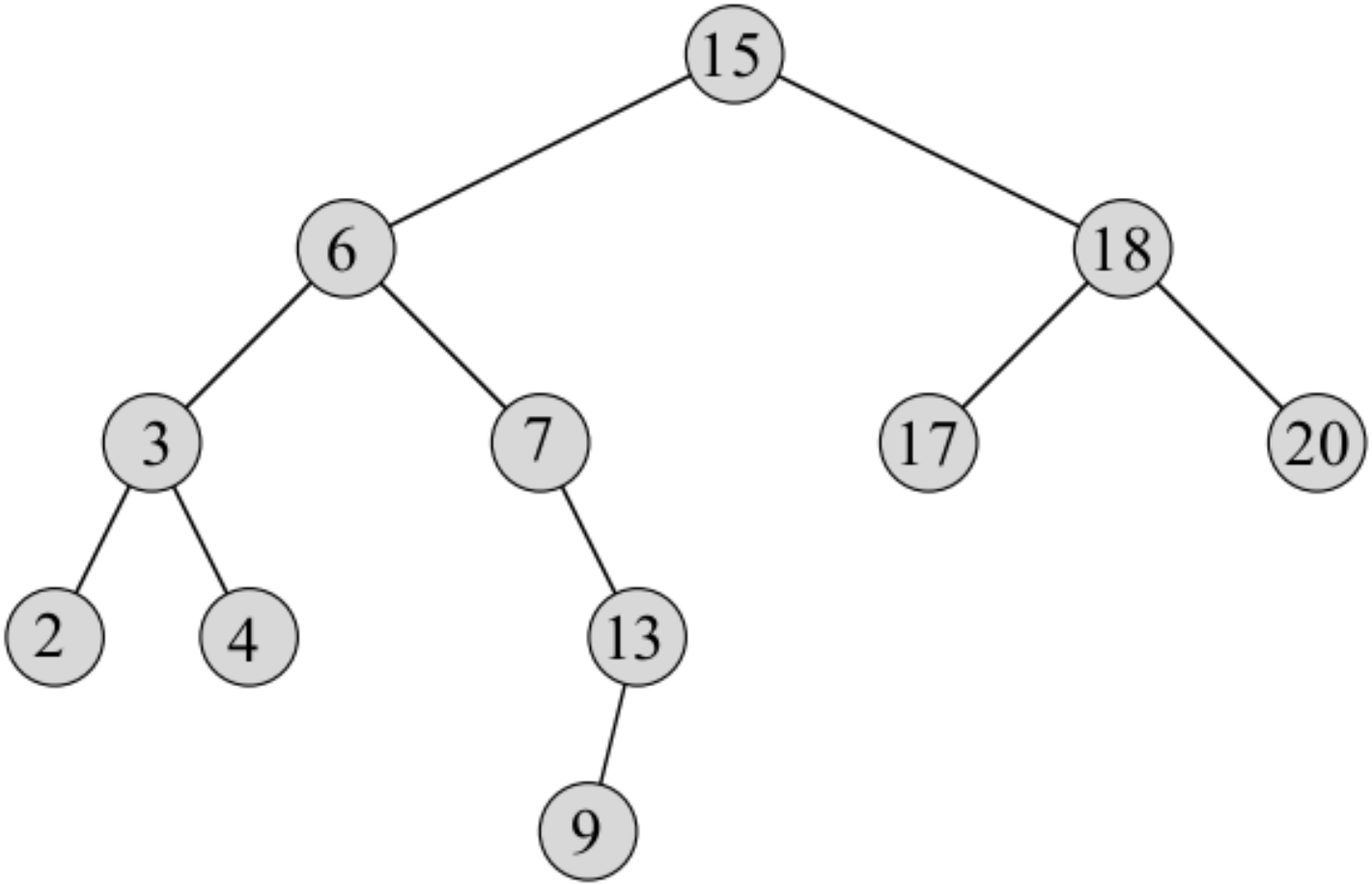

그림: Binary search tree concept

Tree search tree

```
TREE-SEARCH(x, k)
1   if x == NIL or k == x.key
2       return x
3   if k < x.key
4       return TREE-SEARCH(x.left, k)
5   else return TREE-SEARCH(x.right, k)
```

그림: Search

Tree search tree

```
TREE-MINIMUM(x)
1   while x.left ≠ NIL
2       x = x.left
3   return x
```

그림: Minimum

Tree search tree

```
TREE-MAXIMUM(x)
1  while x.right ≠ NIL
2      x = x.right
3  return x
```

그림: Maximum

Tree search tree

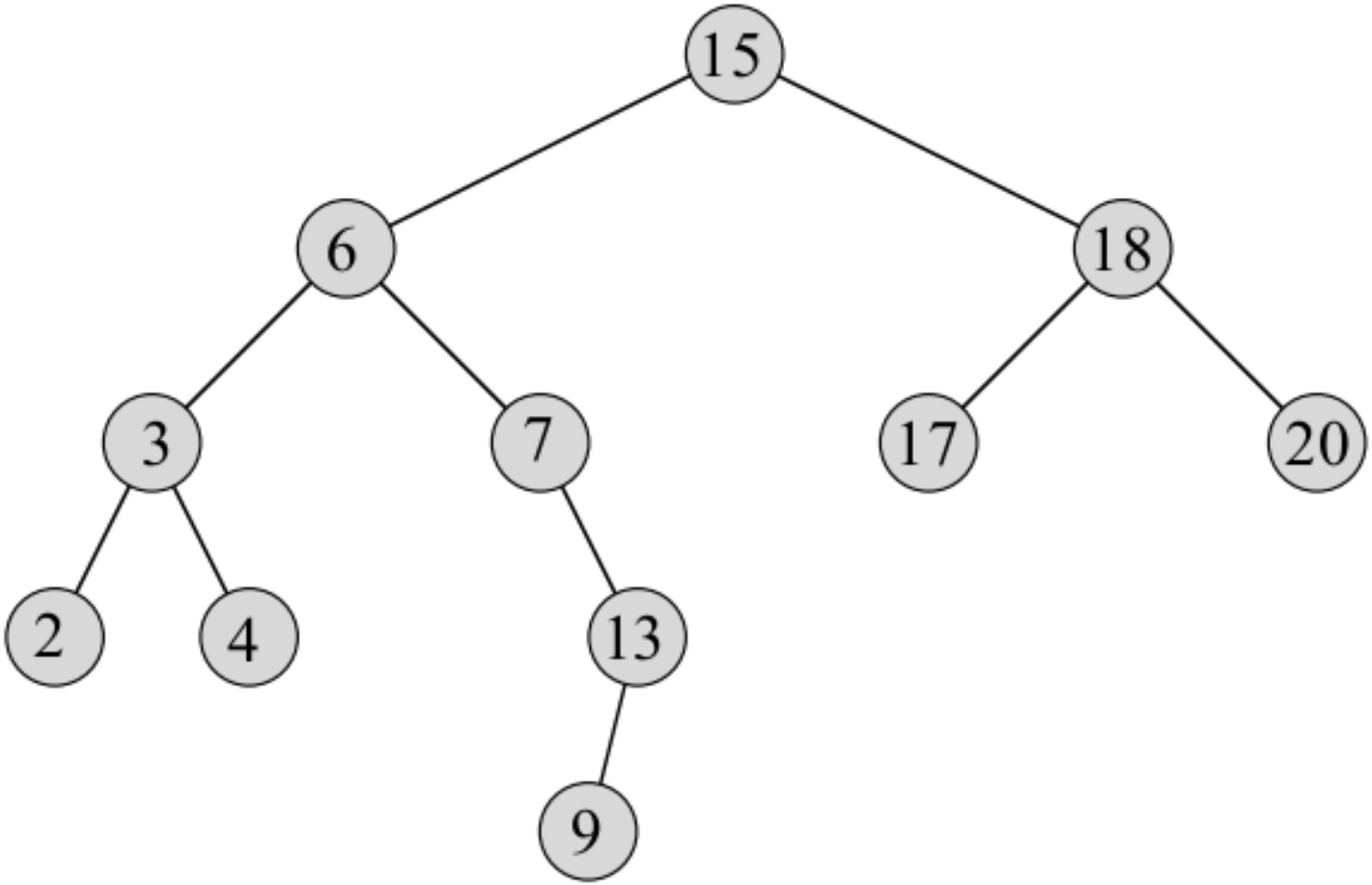

그림: Binary search tree concept

Tree search tree

```
TREE-SUCCESSOR(x)
1  if x.right ≠ NIL
2        return TREE-MINIMUM(x.right)
3  y = x.p
4  while y ≠ NIL and x == y.right
5        x = y
6        y = y.p
7  return y
```

그림: Successor

Tree search tree

```
TREE-INSERT(T, z)
 1   y = NIL
 2   x = T.root
 3   while x ≠ NIL
 4         y = x
 5         if z.key < x.key
 6               x = x.left
 7         else x = x.right
 8   z.p = y
 9   if y == NIL
10         T.root = z          // tree T was empty
11   elseif z.key < y.key
12         y.left = z
13   else y.right = z
```

그림: Insert

Wrap-up

- Although most programming languages provide stack ADT to support function call, its data structure is implemented using array
- Sometimes we need to convert recursive algorithms to iterative algorithms just to overcome the limitation of the stack data structure
- It is quite straightforward to convert recursive algorithm to iterative algorithm by explicityly maintaining user defined stack
- To find the successor and predecessor in BSTs, we introduced parent pointer to BST node.