14강

알고리즘과 자료구조

트리2

서울과학기술대학교 신일훈 교수

한국방송통신대학교 프리임칼리지

학습목표

1. 이진탐색트리의 구현을 이해한다.
2. 이진탐색트리의 단점을 이해하고 이를 보완하기 위한 방법에 대해 이해한다.

01

이진탐색트리 구현 – 노드정의

1. 이진탐색트리 구현 – 노드 정의

- 클래스 BSTNode 정의
  - 트리를 구성하는 하나의 노드에 해당
  - 멤버 변수
    - left: 왼쪽 자식 노드를 가리키는 링크
    - right: 오른쪽 자식 노드를 가리키는 링크
    - key
    - value

1. 이진탐색트리 구현 – 노드 정의

- 클래스 BSTNode 정의
  - 트리를 구성하는 하나의 노드에 해당
  - 메서드
    - 생성자

1. 이진탐색트리 구현 – 노드 정의

클래스 BSTNode 정의

```
class BSTNode:
    def __init__(self, key, value, left=None, right=None):
        self.key = key
        self.value = value
        self.left = left
        self.right = right
```

02

이진탐색트리 구현 – 1단계

2. 이진탐색트리 구현 – 1단계

- 클래스 BST 정의
  - 이진탐색트리를 표현
  - 멤버 변수
    - root: 최상위 노드를 가리키는 링크

2. 이진탐색트리 구현 – 1단계

- 클래스 BST 정의
  - 이진탐색트리를 표현
  - 메서드
    - insert()
    - delete()
    - search()
    - get_min()
    - get_max()
    - print()
    - ...

2. 이진탐색트리 구현 – 1단계

클래스 BST 정의 (생성자)

```
class BST:
    def __init__(self):
        self.root = None
```

2. 이진탐색트리 구현 – 1단계

클래스 BST 정의 (insert() 의사코드)

1. 최상위 노드가 None이면 (트리가 비어 있는 상태), 최상위 노드로 삽입하고 종료
2. 최상위 노드부터 시작하여 노드의 키와 탐색 키를 비교
   - 추가할 키가 더 작고 왼쪽 자식 노드가 존재하면 왼쪽 자식 노드에 대해 2번 작업 반복
   - 추가할 키가 더 크고 오른쪽 자식 노드가 존재하면 오른쪽 자식 노드에 대해 2번 작업 반복
   - 키가 동일하면 충돌 또는 update 케이스. 실패로 처리 또는 값 수정 후 종료
   - 키가 다른데 반복할 자식 노드가 존재하지 않으면, 키에 따라 왼쪽 자식 또는 오른쪽 자식 노드로 삽입

2. 이진탐색트리 구현 – 1단계

클래스 BST 정의 (insert())

```
class BST :
    def insert(self, key, value):
        node = BSTNode(key, value)
        if (self.root == None):
            self.root = node
            return True
```

2. 이진탐색트리 구현 – 1단계

클래스 BST 정의 (insert())

```
class BST :
    def insert(self, key, value):
        ...
        target = self.root
        while (True):
            if (target.key == key):
                # 중복.. value를 수정하도록 구현할 수도 있음
                del(node)
                return False
```

2. 이진탐색트리 구현 – 1단계

클래스 BST 정의 (insert())

```
class BST :
    def insert(self, key, value):
        ...
        while (target):
            ...
            elif (target.key > key): # 왼쪽으로 진행
                if (target.left == None):
                    target.left = node
                    return True
                target = target.left
```

2. 이진탐색트리 구현 – 1단계

클래스 BST 정의 (insert())

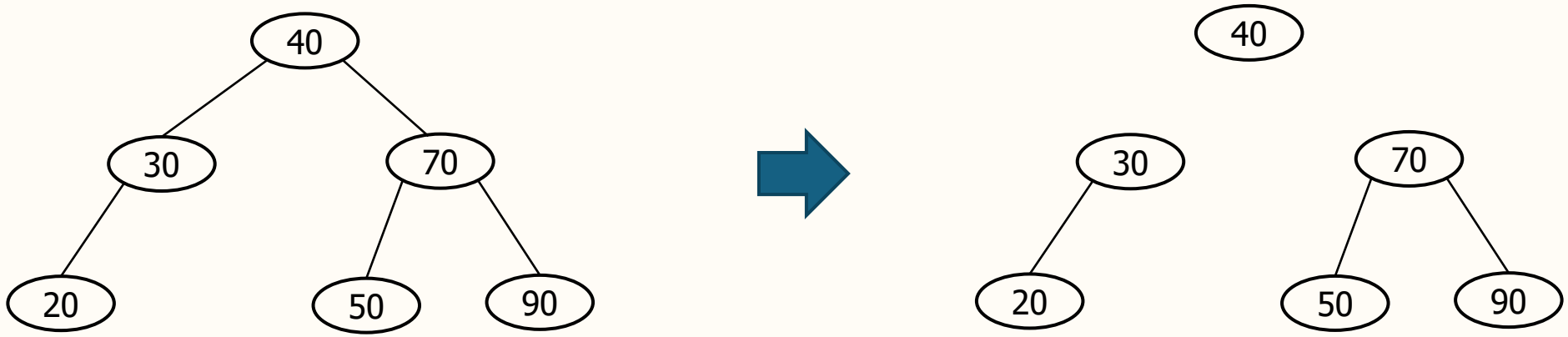
```
class BST :
    def insert(self, key, value):
        ...
        while (target):
            ...
            elif (target.key < key): # 오른쪽으로 진행
                if (target.right == None):
                    target.right = node
                    return True
                target = target.right
```

2. 이진탐색트리 구현 – 1단계

- 클래스 BST 정의
  - print()
    - 중위순회 방식으로 구현 (LNR)
      - print(left subtree)
      - node의 key 출력
      - print(right subtree)
    - 오름차순으로 정렬되어 출력

2. 이진탐색트리 구현 – 1단계

클래스 BST 정의 (중위순회 아이디어)



2. 이진탐색트리 구현 – 1단계

클래스 BST 정의 (print())

```
class BST :
    def print(self):
        self.doLNRPrint(self.root)
        print()
```

2. 이진탐색트리 구현 – 1단계

클래스 BST 정의 (print())

```
class BST :
    def doLNRPrint(self, node):
        if (node != None) :
            self.doLNRPrint(node.left)
            print(node.key, end=' ')
            self.doLNRPrint(node.right)
```

2. 이진탐색트리 구현 – 1단계

BST 1차 테스트

```
if __name__ == '__main__':
    myBST = BST()
    myBST.insert(5, "a")
    myBST.insert(7, "b")
    myBST.insert(3, "c")
    myBST.insert(1, "d")
    myBST.insert(9, "e")
    myBST.insert(15, "f")
    myBST.print()
    myBST.insert(8, "g")
    myBST.print()
```

2. 이진탐색트리 구현 – 1단계

BST 1차 테스트 실행

```
1   3   5   7   9   15
1   3   5   7   8   9   15
```

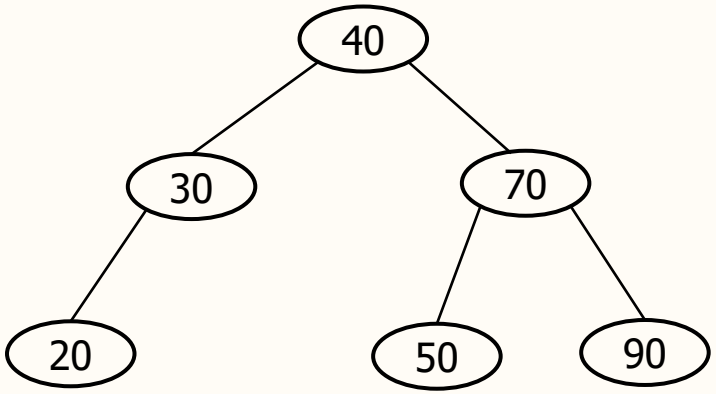
03

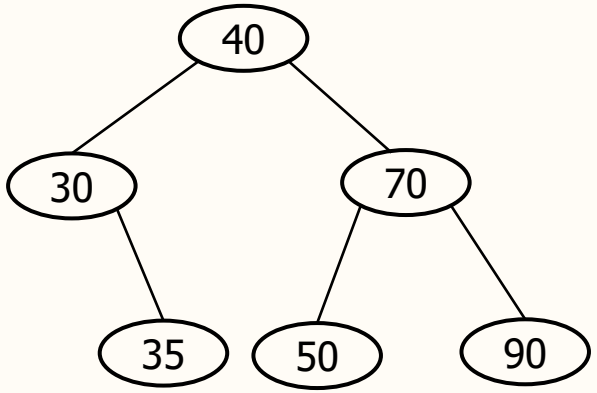
이진탐색트리 구현 – 2단계

3. 이진탐색트리 구현 – 2단계

클래스 BST 정의

- get_min()
  - 왼쪽 자식이 없을 때까지 반복해서 왼쪽 자식 노드를 따라가면 됨.
  - 더 이상 왼쪽 자식이 없으면 해당 노드가 최소값 노드임.





3. 이진탐색트리 구현 – 2단계

클래스 BST 정의 (get_min() - 잘못된 구현1)

```
class BST :
    def get_min(self):
        target = self.root
        while (target):
            target = target.left
        return target
```

3. 이진탐색트리 구현 – 2단계

클래스 BST 정의 (get_min() - 잘못된 구현2)

```
class BST :
    def get_min(self):
        target = self.root
        while (target.left):
            target = target.left
        return target
```

3. 이진탐색트리 구현 – 2단계

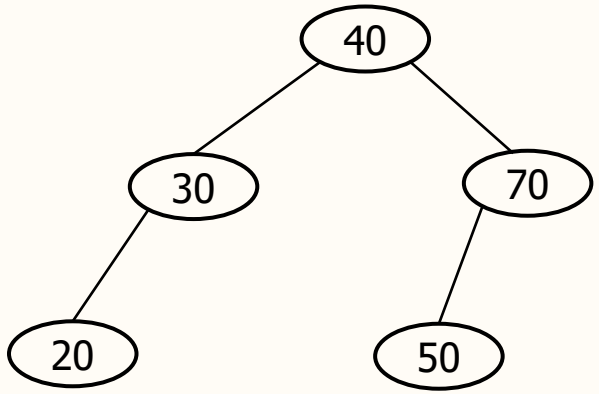
클래스 BST 정의 (get_min() - 올바른 구현)

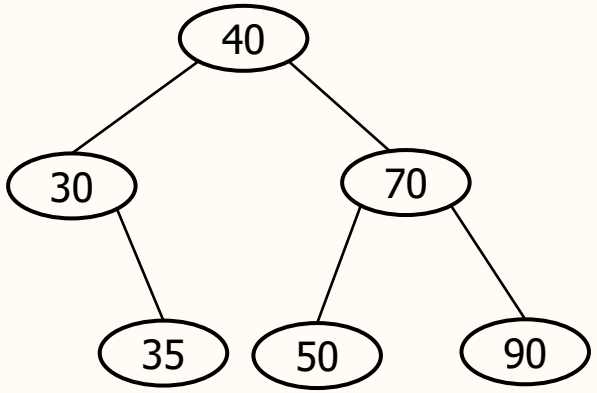
```
class BST :
    def get_min(self):
        target = self.root
        while (target):
            if (target.left == None):
                break
            target = target.left
        return target
```

3. 이진탐색트리 구현 – 2단계

클래스 BST 정의

- get_max()
  - 오른쪽 자식이 없을 때까지 반복해서 오른쪽 자식 노드를 따라가면 됨.
  - 더 이상 오른쪽 자식이 없으면 해당 노드가 최대값 노드임.





3. 이진탐색트리 구현 – 2단계

클래스 BST 정의 (get_max())

```
class BST :
    def get_max(self):
        target = self.root
        while (target):
            if (target.right == None):
                break
            target = target.right
        return target
```

3. 이진탐색트리 구현 – 2단계

BST 2차 테스트

```
if __name__ == '__main__':
    myBST = BST()
    myBST.insert(5, "a")
    myBST.insert(7, "b")
    myBST.insert(3, "c")
    myBST.insert(1, "d")
    myBST.insert(9, "e")
    myBST.insert(15, "f")
    myBST.insert(8, "g")
    myBST.print()
    print(myBST.get_min().key)
    print(myBST.get_max().key)
```

3. 이진탐색트리 구현 – 2단계

BST 2차 테스트 실행

```
1   3   5   7   8   9   15
1
15
```

04

이진탐색트리 구현 – 3단계

4. 이진탐색트리 구현 – 3단계

- 클래스 BST 정의
  - search()
    1. 최상위 노드부터 시작
    2. 노드의 키와 탐색 키를 비교
       - 탐색 키가 작으면 왼쪽 자식 노드에 대해 2를 반복
         - 왼쪽 자식 노드가 없으면 찾는 노드가 없음 & 종료
       - 탐색 키가 크면 오른쪽 자식 노드에 대해 2를 반복
         - 오른쪽 자식 노드가 없으면 찾는 노드가 없음 & 종료
       - 키가 동일하면 탐색을 종료함 (발견)

4. 이진탐색트리 구현 – 3단계

클래스 BST 정의 (search())

```
class BST :
    def search(self, key):
        target = self.root
        while (target):
```

4. 이진탐색트리 구현 – 3단계

클래스 BST 정의 (search())

```
class BST :
    def search(self, key):
        ...
            if (target.key == key):
                return target
            elif (target.key > key):
                target = target.left
            else:
                target = target.right
        return None
```

4. 이진탐색트리 구현 – 3단계

BST 3차 테스트

```
if __name__ == '__main__':
    myBST = BST()
    myBST.insert(5, "a")
    myBST.insert(7, "b")
    myBST.insert(3, "c")
    myBST.insert(1, "d")
    myBST.insert(9, "e")
    myBST.insert(15, "f")
    myBST.insert(8, "g")
    myBST.print()
```

4. 이진탐색트리 구현 – 3단계

BST 3차 테스트

```
...
node = myBST.search(7)
if (node != None) :
    print(node.value)
else:
    print('fail')

node = myBST.search(6)
if (node != None) :
    print(node.value)
else:
    print('fail')
```

4. 이진탐색트리 구현 – 3단계

BST 3차 테스트 실행

```
1   3   5   7   8   9   15
b
fail
```

05

이진탐색트리 구현 – 4단계

5. 이진탐색트리 구현 – 4단계

- 클래스 BST 정의
  - delete()
    1. 타겟이 리프 노드인 경우 (자식이 없는 경우)
       - 부모 노드와 타겟 노드를 연결하는 링크 제거
    2. 타겟이 하나의 자식 노드를 갖는 경우
       - 타겟의 부모와 타겟의 자식 노드를 직접 연결 (손자가 자식으로 바뀌는 격)
    3. 타겟이 두 개의 자식 노드를 갖는 경우
       - 타겟의 자리를 다른 후계자로 대체
       - 후계자는 타겟의 왼쪽 서브 트리 중 키가 가장 큰 노드 또는 오른쪽 서브 트리 중 키가 가장 작은 노드
       - 후계자 노드 삭제 (후계자 노드는 1번 또는 2번 경우이므로 해당 작업 실행함)

5. 이진탐색트리 구현 – 4단계

클래스 BST 정의 (delete())

```
class BST :
    def delete(self, key):
        target = self.root
        parent = None
```

5. 이진탐색트리 구현 – 4단계

클래스 BST 정의 (delete())

```
class BST :
    def delete(self, key):
        ...
        while (target):
            if (target.key != key):
                parent = target
                if (target.key > key):
                    target = target.left
                else:
                    target = target.right
```

5. 이진탐색트리 구현 – 4단계

클래스 BST 정의 (delete())

```
class BST :
    def delete(self, key):
        ...
        while (target):
            ...
            else:
```

5. 이진탐색트리 구현 – 4단계

클래스 BST 정의 (delete())

```
if (target.left == None and target.right == None) :
    # leaf node
    self.do_delete(parent, target, None)
    return target
elif (target.left != None and target.right == None) :
    self.do_delete(parent, target, target.left)
    return target
elif (target.left == None and target.right != None) :
    self.do_delete(parent, target, target.right)
    return target
```

5. 이진탐색트리 구현 – 4단계

클래스 BST 정의 (delete())

```
class BST :
    def do_delete(self, parent, target, child) :
        if (parent == None) :
            self.root = child
        elif (parent.left == target) :
            parent.left = child
        elif (parent.right == target) :
            parent.right = child
```

5. 이진탐색트리 구현 – 4단계

BST 4차 테스트

```
if __name__ == '__main__':
    myBST = BST()
    myBST.insert(20, "a")
    myBST.insert(24, "b")
    myBST.insert(28, "c")
    myBST.insert(26, "d")
    myBST.print()
```

5. 이진탐색트리 구현 – 4단계

BST 4차 테스트

```
if __name__ == '__main__':
    ...
    myBST.delete(28)
    myBST.print()
    myBST.delete(20)
    myBST.print()
    myBST.delete(26)
    myBST.print()
    myBST.delete(24)
    myBST.print()
    myBST.insert(20, "a")
    myBST.print()
```

5. 이진탐색트리 구현 – 4단계

BST 4차 테스트 실행

```
20   24   26   28
20   24   26
24   26
24

20
```

06

기타트리

6. 기타 트리

AVL 트리

- 이진탐색트리는 균형이 무너진 skewed tree가 될 수 있음.
  - 탐색 시간 증가..
- 이 문제를 해결하기 위해 나온 것이 AVL 트리
  - 균형이 깨지면 회전 연산을 통해 균형을 유지.
  - 모든 노드 n이 다음의 조건을 만족하는 이진탐색트리
    - n의 왼쪽 서브트리의 높이와 오른쪽 서브트리의 높이 차이가 1 이하임
  - 삽입이나 삭제로 인해 높이 차이가 2가 되면 회전 연산 수행하여 균형 유지

6. 기타 트리

- 기타
  - RB트리
  - B트리
  - B*트리
  - B+트리
  - …

정리하기

- 이진탐색트리의 구현
- 기타 트리

15 강

다음시간안내 ▶▶▶

그래프