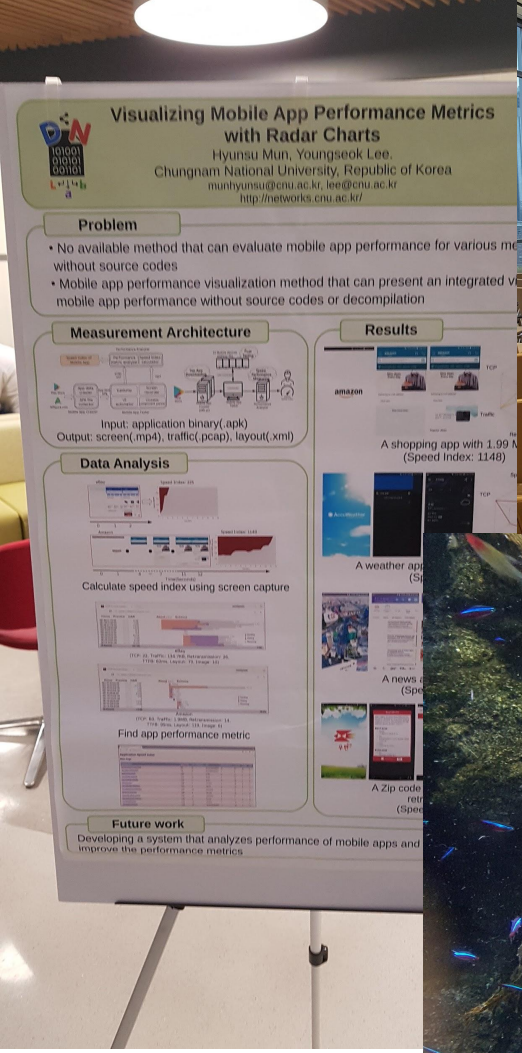
알고리즘 실습

181108 - Binary Search Tree

오늘의 목표

- 이진 탐색 트리(Binary Search Tree)

Feedback


Visualizing Mobile App Performance Metrics with Radar Charts
Hyunsu Mun, Youngseok Lee,
Chungnam National University, Republic of Korea
munhyunsu@cnu.ac.kr, lee@cnu.ac.kr
http://networks.cnu.ac.kr/
Problem
• No available method that can evaluate mobile app performance for various me
without source codes
• Mobile app performance visualization method that can present an integrated vi
mobile app performance without source codes or decompilation
Measurement Architecture
Input: application binary(.apk)
Output: screen(.mp4), traffic(.pcap), layout(.xml)
Data Analysis
Calculate speed index using screen capture
Find app performance metric
Results
A shopping app with 1.99 M
(Speed Index: 1148)
Future work
Developing a system that analyzes performance of mobile apps and
improve the performance metrics



Northeastern University

지난 시간: 트리 순회

- 제출률: 85.37%(35/41)
- 질문: 이메일 32건, 연구실 방문 1명

지지난 시간: 중간고사

- 채점 기준표
  - ~~부분 점수 인정 없음~~ -> 인정
- 점수
  - 평균: 38.41, 중간값: 39
  - 실습 점수표에 첨부

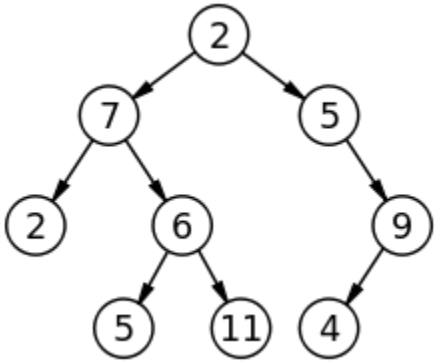
지지난 시간: 설문조사

- 중간 강의 평가: 지금 바로! 통합정보시스템으로 들어가서 중간 강의 평가
  - 학과 조교님 요청.
- 지지지난 시간 구글 설문지 이야기
  - 패키징을 해보고 싶음
  - 진도 속도 및 난이도 (교재)
  - 메일 답변 및 연구실 방문
  - 실습 방식 및 딜레이 감점

트리 표현(지난주 자료)

Tree

- 방향이 있게 연결된 노드들: 순환, 다중 연결 없음
  - https://en.wikipedia.org/wiki/Tree_(data_structure)



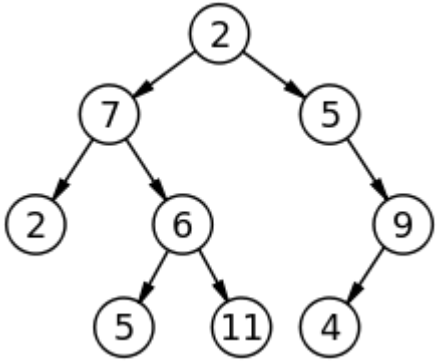
Tree 구성: 노드 클래스

- node.py
  - 생성자(초기화)
  - getter / setter
  - __str__(self) ?
    - JAVA에서의 toString() 같은 것

```python
class Node(object):
    def __init__(self, key=None, value=None, left=None, right=None):
        self._key = key
        self._value = value
        self._left = left
        self._right = right

    def set_key(self, key):
        self._key = key

    def get_key(self):
        return self._key

    def set_value(self, value):
        self._value = value

    def get_value(self):
        return self._value

    def set_left(self, left):
        self._left = left

    def get_left(self):
        return self._left

    def set_right(self, right):
        self._right = right

    def get_right(self):
        return self._right

    def __str__(self):
        return str({'key': self._key, 'value': self._value,
                    'left': self._left, 'right': self._right})
```

Tree 그리기

- 수동으로 그려보자
- value는 우선 무시




```python
def main():
    tree = Node(key=2)

    tree.set_left(Node(key=7))
    tree.set_right(Node(key=5))

    tree.get_left().set_left(Node(key=2))
    tree.get_left().set_right(Node(key=6))
    tree.get_right().set_right(Node(key=9))

    tree.get_left().get_right().set_left(Node(key=5))
    tree.get_left().get_right().set_right(Node(key=11))
    tree.get_right().get_right().set_left(Node(key=4))

    print(tree)


if __name__ == '__main__':
    main()
```

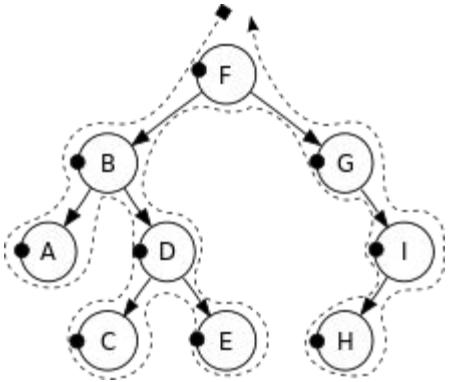
```
/home/harny/PycharmProjects/SCSC/venv/bin/python /home/harny/PycharmProjects/SCSC/week8/node.py
{'key': 2, 'value': None, 'left': <__main__.Node object at 0x7fde01b5c518>, 'right': <__main__.Node object at 0x7fde01bb9320>}
```

트리 순회(지난주 자료)

Pre Order

- 자식노드 순회 전에 처리
  - F, B, A, D, C, E, G, I, H.

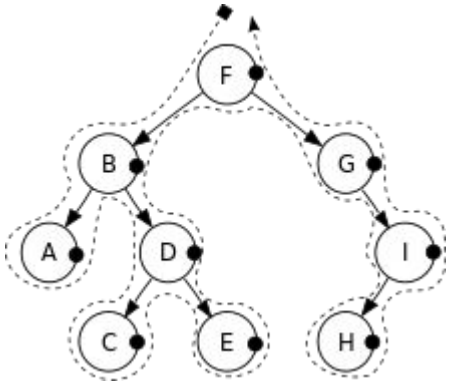
```
preorder(node)
  if (node = null)
    return
  visit(node)
  preorder(node.left)
  preorder(node.right)
```



In Order

- 자식노드 순회 중에 처리
  - A, B, C, D, E, F, G, H, I.

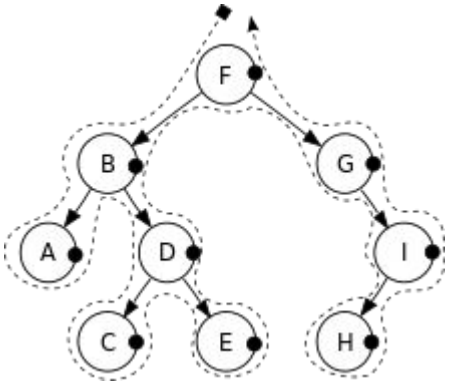
```
inorder(node)
  if (node = null)
    return
  inorder(node.left)
  visit(node)
  inorder(node.right)
```



Post Order

- 자식노드 순회 후에 처리
  - A, C, E, D, B, H, I, G, F.

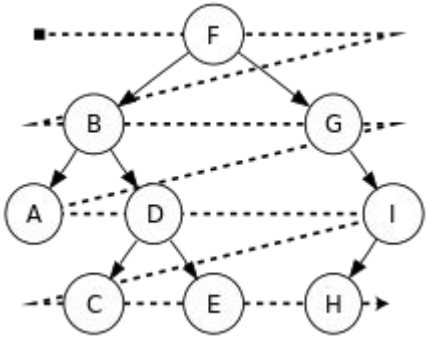
```
postorder(node)
  if (node = null)
    return
  postorder(node.left)
  postorder(node.right)
  visit(node)
```



Level Order

- 같은 depth끼리 왼쪽부터 출력
  - F, B, G, A, D, I, C, E, H.

```
levelorder(root)
  q ← empty queue
  q.enqueue(root)
  while (not q.isEmpty())
    node ← q.dequeue()
    visit(node)
    if (node.left ≠ null)
      q.enqueue(node.left)
    if (node.right ≠ null)
      q.enqueue(node.right)
```



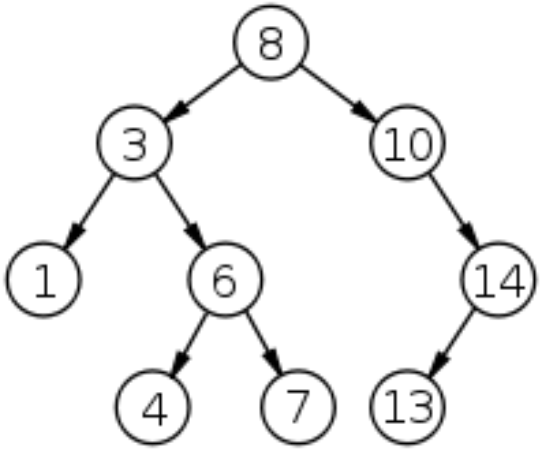
이진 탐색 트리

들어가기에 앞서서...

- 꼭! 알고 있어야 할 내용(모르면 BST 이해 및 구현에 많은 어려움을 겪음)
- 함수, 클래스, 멤버변수(필드), 멤버함수(메소드)
- 클래스(Class) vs 객체(Object) (vs 인스턴스(Instance))

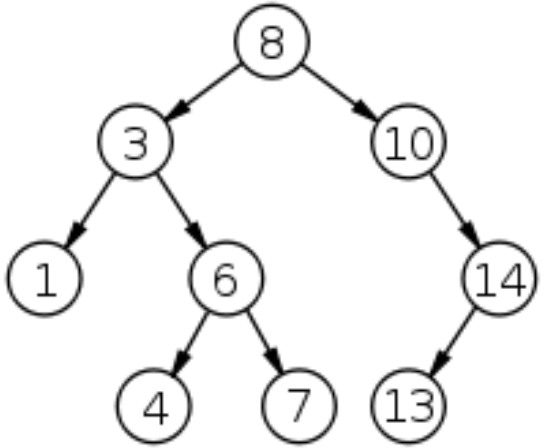
이진 탐색 트리

- Tree에서 빠른 탐색, 추가, 삭제를 하기 위해서 사용
- 보통 Key-Value 형식으로 저장
- https://en.wikipedia.org/wiki/Binary_search_tree



이진 탐색 트리 특징

- (보통) Key-Value 형식으로 저장
- 부모, 왼쪽 자식, 오른쪽 자식 노드를 가짐
  - parent, left_child, right_child
- 부모-자식 노드 Key값의 규칙이 있음
  - 왼쪽 자식의 Key 값은 부모의 Key 값보다 작음
  - 오른쪽 자식의 Key 값은 부모의 Key 값보다 큼



이진 탐색 트리의 멤버 변수

- root - 최상단 Node
- size - Node의 수

```python
from node import Node


class BinarySearchTree(object):
    def __init__(self):
        self._root = None
        self._size = 0

    def get_size(self):
        return self._size

    def get_root(self):
        return self._root
```

이진 탐색 트리 인터페이스

- insert_node(key, value)
- delete_node(key)
- search_by_key(key)
- get_size(): 트리 노드의 개수
- get_root(): BST 루트
- print_bst()

insert_node()

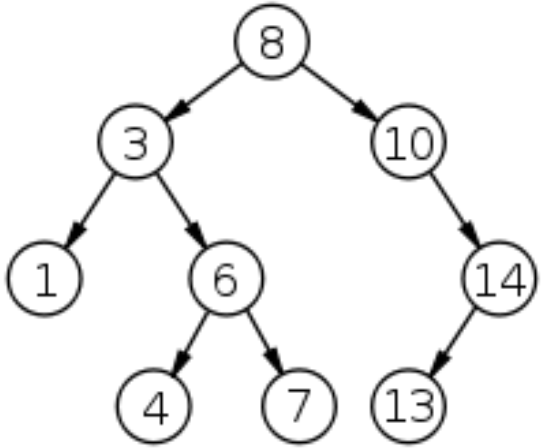
```
void insert(Node*& root, int key, int value) {
  if (!root)
    root = new Node(key, value);
  else if (key == root->key)
    root->value = value;
  else if (key < root->key)
    insert(root->left, key, value);
  else  // key > root->key
    insert(root->right, key, value);
}
```

- insertNode(key, value)
  - root가 None일 경우: root에 Node 생성
  - else: 알맞은 위치까지 이동한 후 Node 생성
  - size 늘리기!
  - 단, key의 중복은 고려하지 않는다. (키가 중복될 경우 value 덮어쓰기)
- 알맞은 위치까지 어떻게 이동?
  - recursive or loop

```
29      def insert_node(self, key, value):
30          if self._root is None:
31              self._root = Node(key=key, value=value, parent=self._root)
32              self._size = self._size+1
33          else:
34              self.__insert_node(current_node=self._root, key=key, value=value)
```

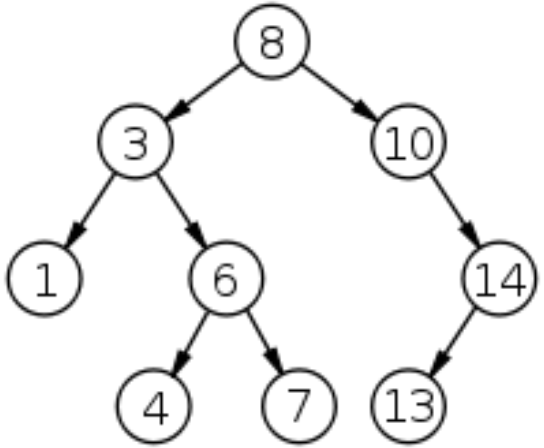
delete_node()

- deleteNode(key)
  - key를 가지는 Node를 찾아서 삭제
  - 삭제 후 자식 Node들 연결
- 남은 자식 Node들을 어떻게 연결?
  - case1) Left, Right Node가 모두 존재
  - case2) Left Node가 존재
  - case3) Right Node가 존재
  - case4) Left, Right Node가 모두 없음



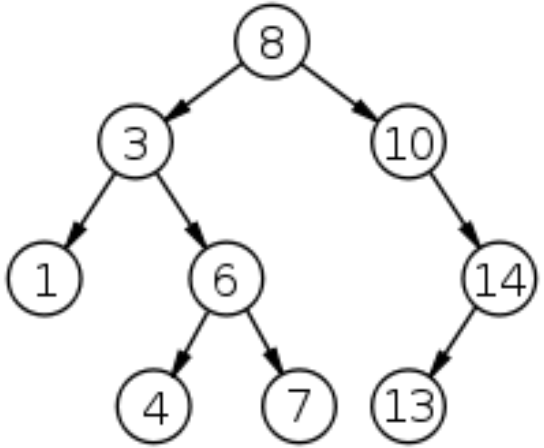
delete_node(): None

- Left, Right Node 없음
  - parent Node의 Left/Right Node에 None
  - size 줄이기



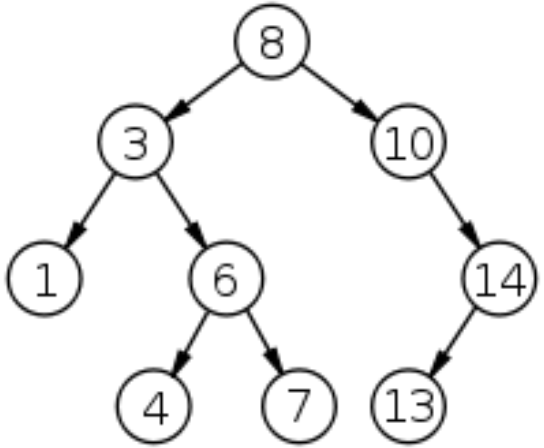
delete_node(): L or R

- Left Node만 존재
  - Left Node의 parent Node를 연결
  - parent Node의 Left/Right Node를 연결
  - Size 줄이기
- Right Node만 존재
  - Left와 방향만 다르고 같음



delete_node(): L and R

- Left, Right Node가 모두 존재
  - Left Tree의 Max Node / Right Tree의 Min Node 찾기
  - 찾은 Node의 값을 지우려는 위치에 할당
  - 찾은 Node 삭제
    - Max/Min Node는 자식이 0~1개 (why?)
  - Size 줄이기



search_by_key()

- search_by_key(key)
  - key를 가지고 탐색
  - Node 반환
  - 없으면 None 반환

print_bst()

- print_bst()
  - inorder로 출력
    - key 오름차순 출력
    - print key, value
  - Recursive {사용/미사용} 둘 다 가능

결과: 입력 리스트

- Key - Value 리스트
  - 리스트를 순회하며 노드 추가

```python
import copy

from binary_search_tree import BinarySearchTree, is_bst


def main():
    target_list = [
        (8, '여덟'),
        (2, '둘'),
        (9, '아홉'),
        (3, '셋'),
        (6, '여섯'),
        (5, '다섯'),
        (4, '넷'),
        (1, '하나'),
        (7, '일곱')
    ]
    print(target_list)
```

결과: BST 생성

- BST에 insert_node()

```
20
21
22
23
```

```
bst = BinarySearchTree()
for key, value in target_list:
    bst.insert_node(key, value)
bst.print_bst()
```

```
1 하나
2 둘
3 셋
4 넷
5 다섯
6 여섯
7 일곱
8 여덟
9 아홉
```

결과: BST 노드 삭제

- 삭제 후 결과

```
25      for key, value in target_list:
26          copied_bst = copy.deepcopy(bst)
27          print('Delete {0}'.format(key))
28          copied_bst.delete_node(key)
29          copied_bst.print_bst()
30          print('Is Binary Search Tree? {0}'.format(is_bst(copied_bst.get_root())))
```

```
Delete 8
1 하나
2 둘
3 셋
4 넷
5 다섯
6 여섯
7 일곱
9 아홉
Is Binary Search Tree? True
```

```
Delete 7
1 하나
2 둘
3 셋
4 넷
5 다섯
6 여섯
8 여덟
9 아홉
Is Binary Search Tree? True
```

테스트 코드: is_bst?

- BST 추가, 삭제 확인

```
6   def is_bst(root):
7       isit = True
8       queue = list()
9       queue.append(root)
10
11      while len(queue) != 0:
12          current = queue.pop(0)
13          if current.get_left() is not None:
14              queue.append(current.get_left())
15              if current.get_key() < current.get_left().get_key():
16                  isit = False
17                  break
18          if current.get_right() is not None:
19              queue.append(current.get_right())
20              if current.get_key() > current.get_right().get_key():
21                  isit = False
22                  break
23
24      return isit
```

기타 유용한 정보

과제) Binary Search Tree 구현

1. Binary Search Tree 구현
    - insert_node(key, value): 난이도 중
    - delete_node(key): 난이도 상
    - search_by_key(key): 난이도 하
    - get_size(): 구현 제공
    - get_root(): 구현 제공
    - print_bst(): 구현 제공

실습 숙제 제출

- 숙제 제출 기한: 2018. 11. 14. 23:59:59
  - 실습 전 날
- 파일 제목: AL_학번_이름_09.zip
  - 파일 제목 다를 시 채점 안 합니다.
  - .egg 안 됨!

실습 숙제 제출할 것

- 2가지 파일을 제출
- AL_학번_이름_숙제번호.zip
  - 소스코드
    - Pycharm을 사용했을 경우 Project 디렉터리에 .idea, venv 같은 디렉터리는 제외
    - Jupyter + IPython을 사용했을 경우 ‘File - Download as’ 에서 .py 다운로드 가능
  - AL_학번_이름_숙제번호.pdf
    - 보고서는 무조건 .pdf
    - .hwp, .doc 등 채점 안 함

실습 보고서에 들어가야할 것

- 목표(할 일)
- 과제를 해결하는 방법
  - 알아야 할 것
- 과제를 해결한 방법
  - 주요 소스코드: 굳이 소스코드 전체를 붙일 필요는 없음
- 결과화면
  - 결과화면 설명(해석), 테스트코드 통과
- 보고서는 기본적으로 '내가 숙제를 했음'을 보이는 것
  - 지나치게 대충 작성하면 의심하게 됨

출석부 및 실습 점수가 궁금하다면?

- 출석부 및 실습 채점표
  - 수업 시작 후 30분까지 지각, 이후 결석
  - 실습 딜레이 1일당: -2점
    - 딜레이 2일까지: -2
    - 이후 -1씩 추가
- 튜터의 테스트 결과

질문이 생기면?

- 이름: 문현수
- 전공: 통신및보안
- 과정: 석박사통합과정 8학기
- 연구실: 데이터네트워크연구실(공5633)
- 메일: munhyunsu@cs-cnu.org
- 알고리즘은 함께 해결해가는 과목이므로 과감하게 연락
- 이메일로 처리가 안 되는 급한일: 문자/전화 등