알고리즘 HW2

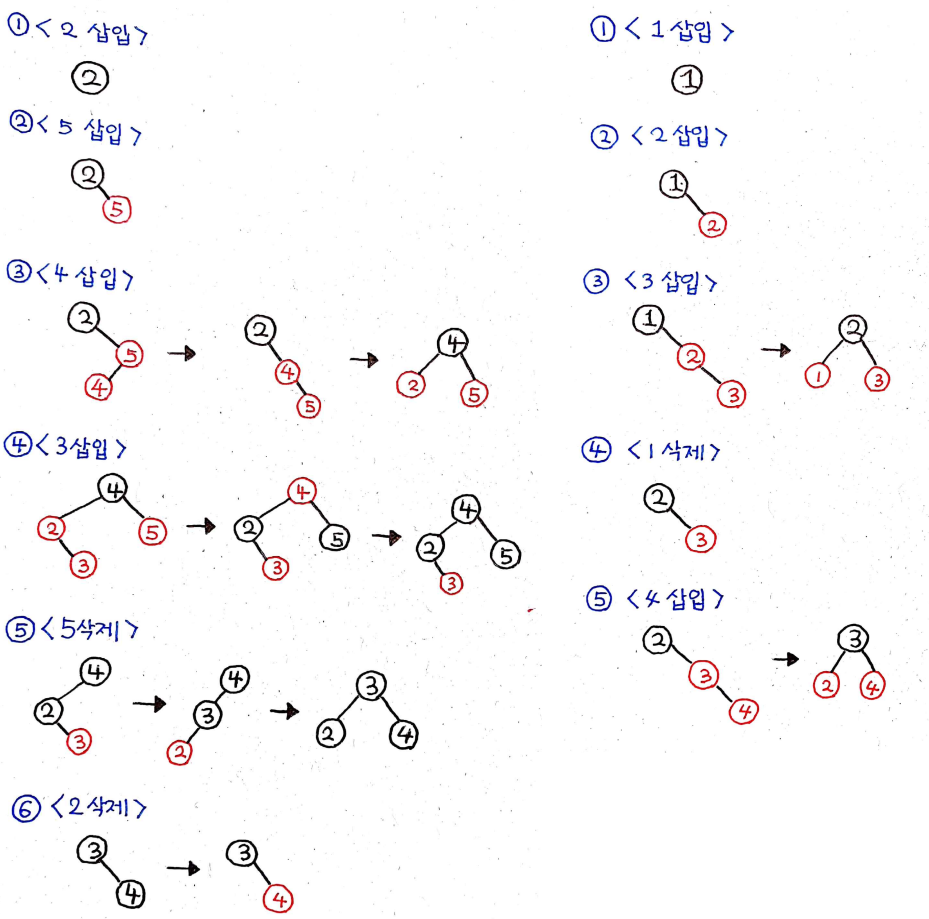
컴퓨터공학부 최덕경
2016-10399

일반적인 Binary Search Tree는 최악의 경우 탐색, 삽입, 삭제에 O(n)의 시간이 걸리는데, 이를 보완하기 위하여 고안된 것이 Red Black Tree(이하 RB 트리)이다. RB 트리는 매순간 이진 트리의 균형이 무너지지 않도록 하여 항상 탐색, 삽입, 삭제에 O(logn)의 시간만 걸리도록 보장해준다. 이때 트리의 각 노드에 size 정보가 저장된다면 Order-Statistic Tree(이하 OS 트리)라고 부른다. 특정 노드의 size 값은 그 노드가 루트 노드인 서브 트리에 존재하는 모든 노드의 개수를 뜻한다.

이번 과제에서는 OS 트리에 대한 삽입 및 삭제 연산을 구현해보면서 RB 트리의 삽입 및 삭제가 어떻게 이뤄지는지 이해하게 될 것이며, 그 과정에서 각 노드의 size 정보는 어떻게 갱신되어야 하는지도 이해하게 될 것이다. 또한 그렇게 각 노드에 저장된 size 정보를 바탕으로 i번째로 작은 값을 찾아내는 Select 연산과 특정 값이 몇 번째로 작은 값인지 알아내는 Rank 연산도 구현해봄으로써 OS 트리의 장점도 이해하게 될 것이다.

- 사용 언어 : Python 3.6.7
- 실험 환경 : Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-45-generic x86_64)
- 실험 방법 : python3 main.py

- **How does OS tree change for my two test cases?**



<좌측 사진이 Test Case 1, 우측 사진이 Test Case 2>

- **Experiment for my two test cases** : 직접 만든 두 가지 테스트 케이스에 대해 checker program을 돌려봄으로써 함수들을 올바르게 구현했는지 확인해봤다.

```
<Test Case 1 : Insert 2, Insert 5, Insert 4, Insert 3, Delete 5, Delete 2>
opt_seq : [0, 0, 0, 0, 1, 1]
val_seq : [2, 5, 4, 3, 5, 2]
out_seq : [2, 5, 4, 3, 5, 2]
Check Result : True

<Test Case 2 : Insert 1, Insert 2, Insert 3, Delete 1, Insert 4>
opt_seq : [0, 0, 0, 1, 0]
val_seq : [1, 2, 3, 1, 4]
out_seq : [1, 2, 3, 1, 4]
Check Result : True
```

- **Implementation of checker program**

  우리가 구현한 OS 트리 관련 연산들의 리턴값은 다음과 같다.

  os_insert(**x**) : x가 없으면 x를 삽입한 뒤 x를 반환. x가 있으면 **0**을 반환.

  os_delete(**x**) : x가 있으면 x를 삭제한 뒤 x를 반환. x가 없으면 **0**을 반환.

  os_select(**i**) : i번째 작은 값을 반환. "i > 트리 크기"면 **0**을 반환.

  os_rank(**x**) : x의 순위(= 몇 번째로 작은지)를 반환. x가 없으면 **0**을 반환.

  즉 리턴값들은 현재 OS 트리의 구조가 어떤지에 의존하는 것이 아니라, 단순히 어떤 자료를 얼마나 가지고 있는지에만 의존한다는 것을 알 수 있다. 따라서 동일한 입력값에 대한 동일한 연산을 '배열'에 대해 수행하면 우리가 구현한 OS 트리 관련 연산들의 리턴값이 올바른지 쉽고 명확하게 확인 가능하다. 그래서 check 함수에서는 빈 리스트 [] 하나를 선언한 뒤, 이 리스트에 대하여 일련의 연산들(insert, delete, select, rank) 각각을 다음과 같이 '똑같이' 수행하면서 매번 우리가 구현한 함수의 리턴값이 올바른지 체크한다.

```
for each operation in sequence
  1) insert x : x가 없으면 x를 리스트에 넣고 check_val ← x, 있으면 check_val ← 0
  2) delete x : x가 있으면 x를 리스트에서 삭제하고 check_val ← x, 없으면 check_val ← 0
  3) select i : check_val ← 리스트에서 i번째 작은 값. "i > 트리 크기"면 check_val ← 0
  4) rank x : check_val ← x의 순위, x가 없으면 check_val ← 0
  Check if the output of our corresponding function equals to check_val
```