2022-1 Algorithm Homework2 보고서

서어서문학과 2017-16849 이윤경

1. 실행 환경 및 파일 IO

vs code를 통해 코드를 구현하였고, Python 3.9.6 버전을 이용하였다. 윈도우 환경에서 테스트를 진행하였으며, hw2.py 파일 자체를 실행하거나 터미널에서 hw2.py가 존재하는 디렉토리로 이동 후, python hw2.py을 입력하여 실행 가능하다. input.txt는 hw2.py가 존재하는 디렉토리의 하위 디렉토리인 input1에 존재하여야 한다. output.txt와 checker.txt도 모두 하위 디렉토리인 input1에 생성된다.

입력은 각 명령어인 I, D, S, R과 명령을 수행할 값인 1부터 9999이하의 정수를 하나의 string으로 입력한다. 파일은 input.txt를 한 줄씩 읽으며 입력이 공백이나 개행문자로 이루어졌을 경우 이는 무시한다. 그 외에는 올바른 입력만을 가정하며 공백을 기준으로 입력 line을 split하여 명령어와 숫자 값으로 구분한다. output.txt에 input sequences를 출력하고 checker program 실행을 위해 각 line을 inputList 리스트에 담는다. 이후 각 명령어에 따라 함수를 실행하고, 결과값을 resultList 리스트에 담는다. 이후 Checker program을 실행할 때, resultList의 각 원소를 output.txt에 출력하고, Checker program을 통해 이 값이 옳은지 판단한다.

Checker program은 아래에서 설명할 알고리즘에 따라 inputList의 input sequences를 읽어 그에 따라 얻은 checkresult와 output sequences가 있는 resultList의 각 원소를 비교하여 두 값이 다를 경우 checker.txt에 두 결과를 출력한다. 모든 결과가 일치한다면 checker.txt는 비어있다.

2. 알고리즘 구현

(1) 전반적 알고리즘 설명

Introduction to Algorithms 교재의 Red-Black tree와 Order-Statistic tree에 대한 의사 코드 및 알고리즘 설명에 기반하여 Python으로 코드를 작성하였다.

Node class는 Tree의 Node를 의미하는 부분으로 data, parent, right, left, color, size를 변수로 가진다. 초기의 Node는 parent, right, left를 None으로 가지고 있으며 Red-Black tree에서 Node를 삽입할 때 우선 색깔을 Red로 지정하기 때문에 color는 1로 지정하였다. 색깔은 1이 Red, 0이 Black을 의미한다. 또한 NIL을 제외한 모든 Node는 초기에 1의 size를 갖기 때문에 size를 1로 지정하였다.

Tree Class의 os_insert와 os_delete 함수는 교재의 Red-Black tree 챕터에 제시된 의사코드에 기반하여 코드를 작성하였고, Tree Class 자체가 Tree를 의미하기 때문에 클래스의 self 개념을 통해 Tree를 함수의 매개변수로 사용할 수 있게 하였다. os_rank와 os_select 함수는 교재의 Order-Statistic tree 챕터에 제시된 의사코드에 기반하여 코드를 작성하였다.

초기 설정(__init__)과 관련하여 os_insert와 os_delete 함수에서 Red-Black tree의 속성을 어길 경우 이를 수정하는 fixup 함수에서 x.parent.parent 및 x.parent.right, x.parent.right 등의 변수를 이용하게 된다. 이때 nil Node의 각 child와 parent를 None으로 설정할 경우 None은 node의 속성을 가지지 않아 Exception이 발생하기 때문에 조작을 수월하게 하기 위해 nil node의 left, right, parent를 모두 self.nil로 설정하였다. nil node는 0의 크기를 가지며 색깔은 Black으로

지정하였다. 또한 초기의 Tree는 root를 None으로 가지지만, 앞서 말한 조작을 수월하게 하게 위하여 root도 nil node로 지정하였다.

Order-Statistic Tree의 핵심 요소인 size를 추가하기 위해 update_size 함수를 추가하였다. 각 Node의 사이즈는 그 Node의 left child, right child의 size의 합에 1(자기 자신)을 더한 것이기 때문에 Node가 nil이 아닌 경우에 한하여 size를 return하도록 코드를 작성하였다.

(2) Tree의 각 함수 설명

각 함수별로 책의 코드에서 수정된 부분을 분석하자면, left_rotate와 right_rotate는 교재의 Order-Statistic Tree에 제시된 size update를 참고하여 코드의 끝에 size를 update하는 과정을 추가하였다.

os_insert의 경우, 삽입 노드인 z가 이미 tree에 존재하는지를 확인하기 위해 while문을 통해 node의 data값이 z와 같은 경우 0을 return하고, 노드가 nil이 될 경우 while문을 빠져나가도록 구현하였다. 또한 Tree의 root가 nil일 경우 삽입하는 노드가 root가 되기 때문에 삽입 노드를 root로 지정하고, 색깔을 Black(0)으로 변경한 후 return 하도록 구현하였다. 이후 책의 의사코드에 따라 insert 작업이 끝난 이후, size를 update하기 위해 삽입 노드에서 시작하여 root에 다다를 때까지 size를 update하도록 구현하였다.

os_delete의 경우, 작업을 도와줄 transplant, tree_minimum, delete_fixup의 함수를 구현하였다. os_delete 함수는 먼저 z가 tree에 있는지를 while문을 통해 확인하며 z가 있을 경우 루프를 break한다. while문을 다 돌고, 확인 노드인 x가 nil일 경우 삭제할 노드가 tree에 존재하지 않는 것이기 때문에 0을 return한다. 또한 size에 관련하여, 삭제할 노드가 하나의 child를 가지거나 child를 가지지 않는 경우, 그 노드의 parent node부터 root에 이를 때까지 update_size 함수를 통해 size를 변경하였다. 2개의 child를 가질 경우, 삭제 노드의 right child 중 가장 작은 노드를 찾고, 그 노드를 삭제 노드의 위치로 변경한다. 이 과정에서 삭제 노드의 원래 위치에서의 parent 노드부터 root에 이를 때까지 size가 변경되기 때문에, y.right를 temp로 지정하고 이 temp의 parent부터 root까지 update_size 함수를 수행하도록 하였다.

os_select는 교재와 동일하게 구현하였으나, i가 root의 size, 즉 tree의 size보다 클 경우 0을 return하도록 하였다. os_rank도 교재와 동일하지만, 먼저 x가 tree에 존재하는지 확인한 후, 존재하지 않을 경우 0을 return하도록 구현하였다.

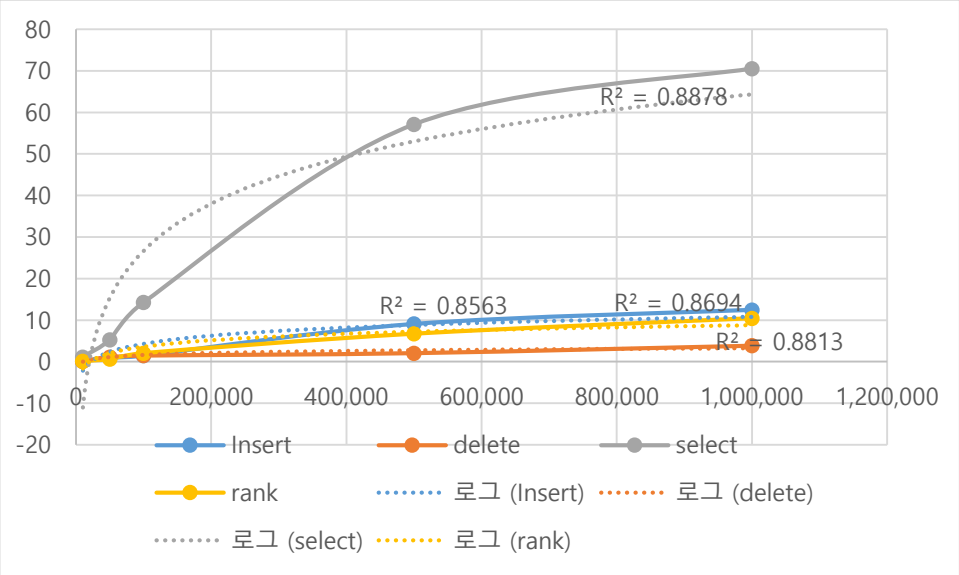
(3) 시간 복잡도

RB-tree의 insert와 delete의 시간 복잡도는 O(log n)이다. Order-Statistic Tree의 insert와 delete는 RB-tree의 함수에서 size 변경만 추가되었다. insert와 delete 모두 삽입 또는 삭제 노드(삭제할 노드의 기존 위치에서 부모 노드)부터 root 노드까지 size update가 이루어지므로 이 과정 또한 O(log n)이다. 그러므로 os_insert와 os_delete의 시간 복잡도는 O(log n)이다.

select의 경우 i와 root.left.size+1을 비교하여 재귀적으로 i와 root.left.size+1이 같을 때까지 left와 right를 분리하며 검색하는 과정을 반복한다. 이 과정은 O(log n)이 소요된다. rank도 마찬가지로 먼저 x가 tree에 있는지를 확인하는 과정이 O(log n), 해당 노드가 부모의 right child인 경우 root에 다다를 때까지 rank를 업데이트하는 과정이 반복되므로 O(log n)이 소요된다.

각 order의 시간 복잡도를 측정하기 위해 먼저 insert를 수행하여 시간을 측정하였다.

delete는 insert를 100만번 실행한 후 delete를 수행하여 delete가 수행된 시간만을 측정하여 input의 크기 별로 graph로 나타내었다. rank와 select는 delete없이 insert를 100만번 실행한 후 각 input의 크기에 따라 시간을 측정하여 나타내었다. 이 과정을 그래프로 나타내면 다음과 같다.



3. Checker program

(1) 구현 방식

Checker program은 Checker Class를 통해 [0...9999]의 배열(파이썬의 경우 리스트)과 self.size를 이용하였다. 배열의 index는 0으로 초기화하였고, Tree Class와 동일하게 각 명령에 따라 os_insert, os_delete, os_select, os_rank의 함수를 구현하였다. 배열의 각 index값은 tree 노드의 각 data값을 의미한다. os_insert의 경우, 삽입하려는 값을 배열의 index로 가지는 원소가 0일 경우, 그 값이 tree에 없다는 것을 의미하므로 삽입이 이루어진다. 그러므로 이 원소를 1로 변경하고, self.size에 1을 더한 후 그 index값을 return한다. 반면 그 원소가 1일 경우, 이미 tree에 삽입하려는 값이 존재한다는 것을 의미하므로 삽입이 실패하여 0을 return한다.

os_delete의 경우 insert와 유사한 구조로 삭제하려는 값을 배열의 index로 가지는 원소가 1인 경우, 삭제가 성공적으로 이루어지므로 이 원소를 0으로 변경하고, 그 index값을 return한다. 원소가 0인 경우 그 값이 tree에 존재하지 않는다는 것을 의미하므로 0을 return한다.

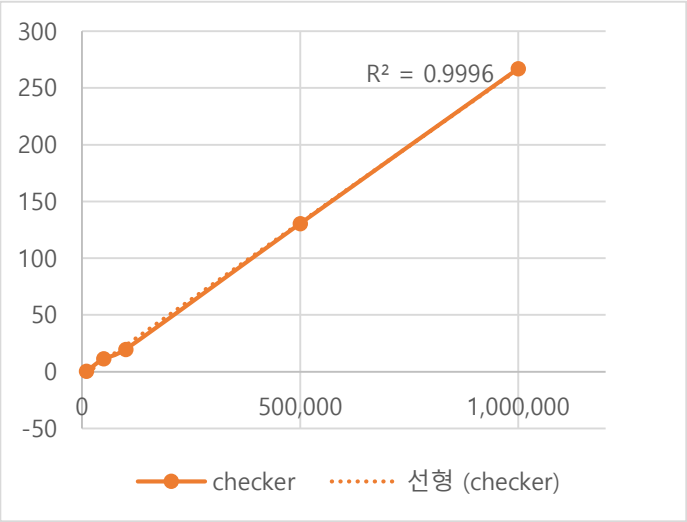
os_select의 경우 먼저 tree의 size보다 i가 클 경우 0을 return한다. 또한 i번째로 큰 노드의 값을 찾기 위해 self.checker 배열에서 원소 값이 1일 경우 cnt를 1씩 증가시키며 이 cnt값이 i와 같을 경우 그 원소의 index값을 return하도록 하였다.

os_rank의 경우 먼저 self.checker[x]값이 0일 경우 그 값이 tree에 존재하지 않는다는 것을 의미하므로 0을 return한다. x의 rank를 찾기 위해 self.checker의 index를 1부터 x까지 증가시키며 그 값이 1일 경우 cnt를 증가시키고, for loop가 끝난 후 cnt값을 return하도록 하였다.

이러한 결과를 위에서 언급한 바와 같이 각 input에 따른 output과 checker의 결과인 check 변수와 비교하여 checker program을 수행한다.

(2) 시간 복잡도

n	10,000	50000	100,000	500,000	1,000,000
checker	0.38	11.512	19.736	130.505	267.126



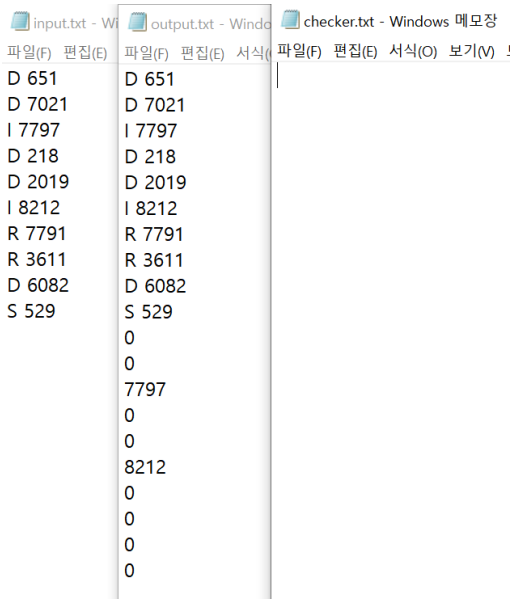
checker program의 시간 복잡도는 O(n)이다. checker program이 한 번 실행될 때마다 insert와 delete의 경우 x에 해당하는 index를 찾는 과정이므로 O(1)이다. select의 경우 self.checker의 index를 루프를 통해 돌면서 i값과 cnt값이 일치할 때 함수가 return된다. 이는 최악의 경우 self.checker의 길이인 10,000만큼이 소요되고 평균적으로는 그보다 작은 상수의 시간이 소요된다. rank의 경우 1부터 x까지의 루프를 돌며 self.checker 리스트의 원소 값이 1인지를 탐색한다. 이 경우 O(x)의 시간 복잡도를 지니지만, 입력이 1보다 크고 10000보다 작은 정수라는 점에서 이 또한 상수 시간이 소요된다. 즉 checker program의 각 함수는 상수 시간이 소요되고, 명령의 개수, 즉 n만큼 반복되므로 시간 복잡도는 O(n)이다. 그래프를 통해서도 시간복잡도가 O(n)임을 확인할 수 있다.

4. 실행 예 / example running

```
import random
f = open("./input1/input.txt", 'w')
order=["I", "D", "S", "R"]
n=1000000
for i in range(n):
    r_order=random.choice(order)
    # r_order="I"
    f.write(str(r_order)+ " "+ str(random.randrange(1,10000))+ "₩n")
```

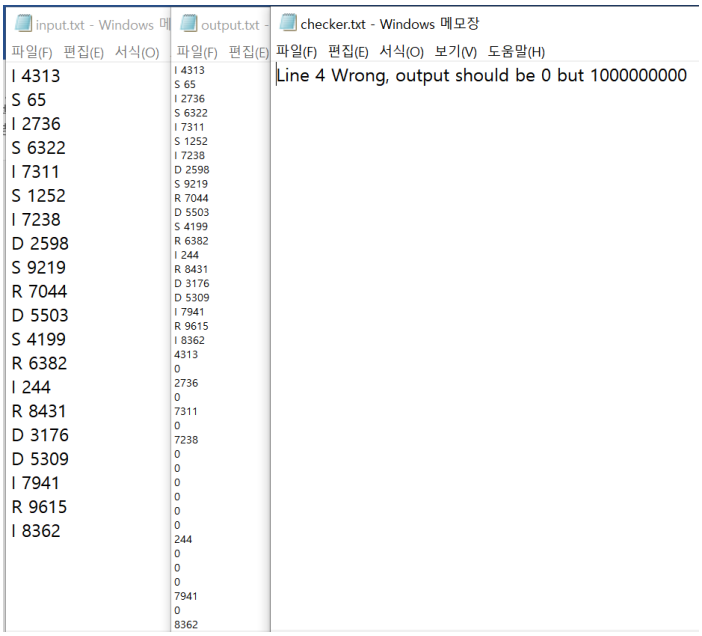
위와 같은 방식으로 n의 크기에 따라 test case를 만들어 프로그램을 실행하였다. order는 I, D, S, R 중 랜덤으로 선택하고, 입력 data는 1부터 9999까지의 정수 중에서 랜덤으로 선택하였다. input 크기가 큰 경우는 제출 파일의 input1(n=1000000), input2(n=500000)로 첨부하였고, 보고서에는 input 크기가 작은 경우를 예시로 첨부한다.

i) n=10



n이 10인 경우의 예시이다.

결과가 모두 checker program의 결과와 일치하여 checker.txt는 비어있다. output.txt는 먼저 input sequences를 모두 출력한 후 프로그램의 실행결과를 한 줄씩 출력하였다.



ii) n=20

임의로 결과값을 수정하여 틀린 값이 존재할 때 checker.txt가 어떻게 변화하는지를 보여주었다. i)과 다른 예시는 동일하다.