Algorithms
Homework 2 Report

2013-11425 이창영

1. 개요

- Order-Static Tree는 기본적으로 Red-Black Tree의 특성을 가지고 있고, 추가적으로 각 node는 사이즈 정보를 포함하고 있다.
- Order-Static Tree를 직접 구현해보고, 잘 작동하는지 확인하는 Checker Program을 만든다.

2. 사용 언어

python을 사용하여 구현하였다.

3. 컴파일, 실행.

1) ostree

python ostree.py input_filename을 입력하여 실행할 수 있다.

실행 후에 "out.txt" 파일에 input과 결과가 출력된다.

(* input_file의 마지막 줄에 공백이 있으면 안 됨.)

2) checker

python checker.py out.txt를 입력하여 실행할 수 있다.

out.txt를 분석하여 input과 input에 대한 결과가 올바른지 매칭 하는지 확인하여 맞으면 True 틀리면 False를 출력한다

4. 구현방법, 실행 결과(red-black tree 위키 참조하여 구현)

1) ostree

각각의 node를 나타내는 node class와, 트리 전체를 나타내는 ostree class를 만들었다.

(1) node class

color, data, size, left, right, parent 정보를 가지는 변수들,

grandparent, uncle, sibling을 return하는 메소드들을 가지고 있다.

(2) ostree class

head(root)를 나타내는 node를 하나 포함하고 있고,

insert, delete, rank, select를 수행하는 메소드들,

insert, delete할 때 모든 경우에 대해 node의 색을 결정해 주는 메소드들,

그리고 균형을 맞추기 위한 rotate_left, rotate_right 메소드를 가지고 있다.

a. insert

트리를 따라 쭉 쭉 내려가면서 insert할 위치를 찾는다. leafNode를 만나면 insert.
성공하는 insert에 대해 쭉 쭉 내려갈 때 size를 다 1씩 올려준다.
실패하면 다시 root까지 올라가면서 size를 1씩 내려준다.
case 1 : 새로운 노드가 root인 경우
case 2 : 부모 노드가 검은색인 경우
case 3 : 부모 노드와 부모의 형제 노드가 빨간색인 경우
case 4 : 부모 노드는 빨간색, 부모의 형제 노드는 검은색인 경우 (처리 후 case5로 넘어감)
case 5 : rotate를 통해 밸런스를 유지해준다.

b. delete

일단 delete할 노드를 찾은 뒤, 그 노드의 오른쪽 서브 트리에서 가장 작은 값을 찾는다.
가장 작은 값과 찾은 노드의 값을 서로 바꾼다.
바꾼 자리에 있는 노드를 삭제하고 그의 child를 그 자리로 올리면 된다. 이때 여러 case가 있다.
case 1 : root가 되는 경우
case 2 : 형제 노드가 빨간색일 경우
case 3 : 부모, 형제, 자식들이 검은색인 경우
case 4 : 부모는 빨간색, 형제와 형제의 자식들은 검은색인 경우
case 5 : 부모의 왼쪽 자식이면서 형제는 검은색, 형제의 left child는 빨간색, 형제의 right child는 검은색인 경우 또는 이 대칭인 경우
case 6 : 부모의 왼쪽 자식이면서 형제는 검은색, 형제의 right child는 빨간색인 경우 또는 이 대칭인 경우

(3) 실행 결과

input file에 대한 실행 결과를 out.txt 파일에 출력한다.
out.txt에는 input file의 인풋도 포함한다.

2) checker

(1) arr[0~998] 배열을 모두 0으로 초기화 한다.

(2) insert(n) - arr[n-1]이 0이면 1로 바꿔주고 n을 결과에 적는다.
- arr[n-1]이 1이면 0을 결과에 적는다.

(3) delete(n) - arr[n-1]이 1이면 0으로 바꿔주고 n을 결과에 적는다.
- arr[n-1]이 0이면 0을 결과에 적는다.

(4) select(i) - arr을 맨 앞에서부터 읽으면서 1일 경우 count를 센다.
- count가 i가 될 때 그 index+1을 결과에 적는다.
- 끝까지 읽었는데도 count가 i보다 작으면 0을 결과에 적는다.

(5) rank(n) - arr을 맨 앞에서부터 index가 n-1일때까지 1일 경우 count를 센다.
- count를 결과에 적는다.
- arr[n-1]이 0일 경우 0을 결과에 적는다.

(6) 실행 결과

out.txt에 적혀있는 input과 그에 따른 output을 checker의 결과와 비교하여 모두 일치하면 True, 그렇지않으면 False를 stdout에 출력한다.

5. 결론.

- Order-Static Tree를 이용하면 이전에 배웠던 linear time selection 알고리즘보다 빠른 시간(O(logn))에 select를 수행할 수 있고 rank 또한 빠른 시간에 수행할 수 있다.
- 따라서 select나 rank와 같은 일을 수행할 일이 많을 경우 Order-Static Tree 자료구조를 사용하는 것이 좋다.