REPORT

과목명 : 자료구조

담당 : 장형수 교수님

제출마감일 : 2021년 12월 03일

학과 : 생명과학과

학번 : 20171483

이름 : 조주현

SOGANG UNIVERSITY SOGANG UNIVERSITY

1. Problem description

<HW4-1>
Binary tree로 구현된 max heap을 구현하는 문제이다. Tree node 구조체에는 parent, leftChild, rightChild 3개의 포인터와 data_field가 존재한다. 삽입, 삭제를 각각 구현해라.
<HW4-2>
정수열의 크기와 정수열을 입력받고, 이를 각각 inorder, postorder로 출력한다. 이 때 tree로 구현하여야 한다.
<HW4-3>
Max priority heap을 BST(binary search tree)로 구현하라. 가장 큰 값을 출력하는 top, 가장 큰 값을 출력과 동시에 삭제한는 pop, 데이터를 삽입하는 push를 각각 구현하라.

2. Algorithm description

<HW4-1>
Struct queue를 이용해 새로운 leaf를 달아줄 tree node를 찾는다. Struct queue는 tree_node를 가르키는 포인터, 다음 queue_node를 가르키는 포인터로 구성되어 있다. 데이터를 입력받으면 새로운 tree_node에 데이터를 저장한 뒤 해당 node를 새로운 queue_node로 가르킨다. Front는 queue의 첫 node부터 시작하여 front->tree_node의 leftChild, rightChild를 저장 한 뒤에는 다음 queue_node로 이동한다. 새로운 node를 저장하면, max heap 정렬을 통해 값을 정렬해준다. Front에는 마지막으로 값을 추가해준 tree_node의 주소가 담겨 있으므로 이를 이용해 가장 마지막에 저장된 tree_node의 값과 root_tree_node의 값을 바꿔준다. 그 다음 max heap 정렬을 이용해 값을 정렬한다. 중복값은 queue의 head부터 모든 값을 참조하여 비교한뒤 중복값을 걸러낸다. 실제 코드의 경우 첫 root node와 left, righChild의 예외조건은 hard coding하였다.
<HW4-2>
Make_node() 함수의 매개변수로 배열의 시작 인덱스(start), 끝 인덱스(end), 문자열(arr), is_same(중복값 체크)를 사용한다.
정수열의 시작 값을 이용해 새로운 node를 만들어 data로 사용한다. arr에서 arr[start] 보다 큰 값을 찾고, 해당 인덱스를 변수 index에 저장한다. Index는 -1로 초기화 되어있으며, for loop을 이용해 arr[0]보다 큰 값을 찾을 수 있다. 이 뒤로 3가지 경우의 수가 나온다.
1. for loop 이후 index == -1인 경우
이는 문자열의 모든 data가 root node의 data보다 작다는 의미이며 arr[start+1:end]는 모두 앞서 만든 node의 leftChild에 저장됨을 의미한다. 따라서 start = start + 1, end = end, arr = arr 를 이용해 recursive하게 새로운 node를 만들며, 해당 node는 root_node의 leftChild에 저장한다.
2. for loop 이후 index == start + 1인 경우
root data 바로 다음 값이 root data보다 크다는 의미로 start = start + 1, end = end, arr = arr 를 이용해 recursive하게 새로운 node를 만들며, 해당 node는 root_node의 rightChild에 저장된다.
3. for loop 이후 index > start + 1 인 경우
start와 index 사이에 1개 이상의 data가 존재함. 따라서 index 앞까지는 leftChild에, index부터 end까지는 rightChild에 넣어 새로운 node를 저장한다.

for loop 안에서 root data와 arr을 비교하며 중복값을 찾아낸다.

<HW4-3>
기본적인 구조는 HW4-1과 동일하다. Insert와 Push, delete와 pop은 비슷한 구조를 가지지만 각각 삽입할 위치, 삭제할 노드를 고르는 방법이 다르다. 먼저 push의 경우 입력한 data를 크기에 따라 root_node의 data보다 크면 rightChild, 작으면 leftChild로 이동해 다시 비교하는 recursive한 방법을 찾아 삽입한다. Pop의 경우 가장 큰 node는 rightChild를 통해 계속 이동하다가 node->righChild = NULL 일 때가 가장 클 것이므로 해당 node의 data를 출력하고 삭제한다. top의 경우 pop에서 찾는 방법과 동일하게 탐색후 data를 출력한다. 해당 과제의 중복값 체크 함수는 HW4-1과 동일하다.

3. Program output

HW4_1

```
i 4
insert 4
i 4
Exist number
i 5
insert 5
d
Delete 5
d
Delete 4
d
heap is empty
i 3
insert 3
q
Program ended with exit code: 0
```

HW4_3

```
push 3
push 3
push 3
Exist number
top
Top: 3
push 5
push 5
top
Top: 5
pop
POP : 5
push 3
Exist number
pop
POP : 3
pop
The queue is empty
q
Program ended with exit code: 0
```

HW4_2

```
6
30 5 5 40 35 80
cannot construct BST
Program ended with exit code: 0
```

```
6
30 5 2 40 35 80
Inorder: 2 5 30 35 40 80
Postorder: 2 5 35 80 40 30
Program ended with exit code: 0
```