Algorithms

HW#2 Report

2013-11706
기계항공공학부 강인재

[레드 블랙 트리]

레드 블랙 트리는 이진 검색 트리의 각 노드가 블랙 또는 레드의 색상 정보를 가지고 있는데, 다음과 같은 특성을 만족한다. 색상 정보는 블랙 또는 레드의 두 가지 경우만 있으므로, 이론상으로는 1bit만 가지고도 나타낼 수 있다.

① 루트 노드는 블랙이다. ② 모든 리프 노드는 블랙이다. ③ 어떤 노드가 레드이면 그 노드의 자식은 블랙이다. ④ 루트 노드에서 임의의 리프 노드에 이르는 경로상의 블랙 노드의 수는 모두 같다. (※ 여기서 리프 노드는 NIL을 의미한다.)

위의 특성을 만족하게 하여, 이진 검색 트리가 연산 순서에 따라 불균형하게 되는 문제를 방지할 수 있다. 키의 개수가 n개인 레드 블랙 트리의 가능한 최대 깊이는 $O(\log n)$이라는 사실을 증명할 수 있다. 그러나 삽입 및 삭제 직후에는 이러한 특성이 만족되지 않을 수 있으므로, 트리의 구조를 수정하여 특성을 만족하도록 하는 추가적인 작업이 필요하다. 이것이 레드 블랙 트리의 연산을 어렵게 하는 원인이다. 삽입 및 삭제 이후에는 문제가 발생한 노드를 중심으로 하여 parent로 올라가면서 문제를 해결하게 된다.

[Order-Statistic 트리]

Order-Statistic 트리는 각 노드마다 크기 정보를 추가적으로 가지고 있다. 각 노드의 크기는 그 노드를 루트로 하는 서브트리 내의 (NIL을 제외한) 모든 노드의 수로 정의된다. 이 크기 정보를 이용하여 전체 트리를 탐색하지 않고도 몇 번째 작은 원소가 무엇인지 또는 주어진 원소가 몇 번째인지 등을 알 수 있다. 이 또한 삽입 및 삭제 과정에서 크기 정보를 업데이트 해 주어야 한다. 이것의 구현에 대해서는 **[알고리즘의 구현]**에서 설명하였다.

[알고리즘의 구현]

Order-Statistic의 특성을 갖는 레드 블랙 트리를 C언어로 구현하였다. 대부분의 코드는 Introduction to Algorithms 교재에 수록된 내용을 기반으로 작성되었다.

트리는 다음과 같이 구성되어 있다. 우선, 트리를 구성하는 각 노드의 구조체는 왼쪽 자식, 오른쪽 자식, 부모의 세 노드형 포인터와 키, 크기, 색상의 세 가지 정수를 가지고 있다. 트리 구조체는 루트 노드와 NIL의 노드형 포인터를 갖는다.

```
typedef struct node
{
	struct node* left;
	struct node* right;
	struct node* p;
	int key;
	int size;
	int color; // BLACK = 0, RED = 1
} Node;

typedef struct tree
{
	Node* root;
	Node* nil;
} Tree;
```

프로그램이 시작되면 트리 구조체가 힙 영역에 동적 할당된다. 그리고 트리 구조체 내의 루트와 NIL 역시 동적 할당된다. NIL의 키와 크기는 0으로, 색상은 블랙으로 초기화된다. 프로그램의 시작 직후에는 트리 구조가 없으므로(또는 루트가 NIL이므로) 루트와 루트의 p는 NIL로 초기화된다.

이후 입력 리스트로부터 연산자와 피연산자를 각각 char형과 int형으로 입력받는다. 이를 콘솔에 출력하고, 연산자의 종류(I, D, S, R 중 하나)에 따라 적절한 함수를 실행한다. main 함수가 직접 호출하는 함수 목록은 다음과 같다. 매개변수 쌍은 전부 `(Tree* T, int i)`의 형태이며, 반환형은 전부 int형이 되도록 하였다.

```
int OS_Insert(Tree* T, int i);
int OS_Delete(Tree* T, int i);
int OS_Select(Tree* T, int i);
int OS_Rank(Tree* T, int i);
```

이들 함수의 실행을 위한 보조 함수들은 다음과 같다. 기본적으로는 Introduction to Algorithms에 수록된 의사코드에 기반한 것이나, 원하는 알고리즘을 구현하기 위해 다음과

같은 몇 가지 작업을 추가하였다.

```
void left_Rotate(Tree* T, Node* x);
void right_Rotate(Tree* T, Node* x);
void RB_Insert_Fixup(Tree* T, Node* z);
void RB_Transplant(Tree* T, Node* u, Node* v);
Node* Tree_Min(Tree* T, Node* x);
void RB_Delete_Fixup(Tree* T, Node* z);
int OS_SelectHelper(Node* x, int i);
```

1. 교재에서 두 번째 매개변수를 노드형 포인터로 받는 코드의 경우에는 int형으로 받는 코드로 고쳐 작성하였다. 함수 내에서 입력값을 키로 갖는 노드를 동적 할당을 통해 생성하고, 왼쪽 자식과 오른쪽 자식을 NIL로, 크기 정보를 1로 초기화하였다.

2. `left_Rotate`와 `right_Rotate`의 마지막 부분에 크기 정보를 업데이트하기 위해 다음 코드를 추가하였다. (이 역시 Introduction to Algorithms에 나와 있다.)

```
y->size = x->size;
x->size = x->left->size + x->right->size + 1;
```

3. `OS_Insert`의 초기에 실패하는 검색을, `OS_Delete`와 `OS_Rank`의 초기에 성공하는 검색을 수행하도록 하였다. 이진 검색 트리의 키 값 비교를 통한 검색 과정이다. 검색의 성공 및 실패 여부가 예상했던 바와 다르면 0을 반환하도록 하였다.

4. 삽입 과정에서 크기 정보를 업데이트하기 위해, 새로 생성된 노드가 트리의 끝에 삽입된 직후 조상을 하나씩 거치면서 루트 노드에 이를 때까지 각 노드의 크기 정보를 1씩 증가시켰다.

5. 삭제 과정에서 크기 정보를 업데이트하기 위해, 삭제하려는 노드가 자식이 없거나 하나만 있을 경우 삽입 과정에서와 마찬가지로 조상을 하나씩 거치면서 루트 노드에 이를 때까지 각 노드의 크기 정보를 1씩 감소시켰다. 왼쪽과 오른쪽 자식을 모두 갖는 노드의 경우에는 동일한 과정을 거친 후, 삭제 노드의 오른쪽 자식이 최소 노드일 경우를 고려하여 최소 노드의 크기 정보에 최소 노드의 오른쪽 자식의 크기 정보를 대입하였다. 이후 오른쪽 서브트리의 최소 노드가 가지게 될 크기 정보를 미리 계산하여(즉, 삭제되는 노드의 왼쪽 및 오른쪽 자식의 크기 정보를 참조하여) 대입하였다. 이 작업들은 노드가 삭제되기 전에 수행되도록 하였다. 코드는 다음과 같다.

```
y = Tree_Min(T, z->right); // z는 삭제 노드

// update
temp = y;
while (temp->p != T->nil)
{
        temp = temp->p;
        (temp->size)--;
}
temp = y;
temp->size = temp->right->size;
temp->size = z->left->size + z->right->size + 1;
```

6. 교재의 OS-Select 함수가 노드형 포인터와 int형 매개변수 쌍을 갖는 재귀함수임을 고려하여 `OS_SelectHelper` 함수를 정의하였다. 이 함수는 교재의 OS-Select를 반환형만 int로 하여 동일하게 구현한 것이다. `OS_Select` 함수는 트리형 포인터를 받아, 트리의 루트 노드를 `OS_SelectHelper`의 인수로 전달한다.

[트러블 슈팅]

1. Introduction to Algorithms 교재의 Insert의 의사코드를 C언어로 구현하던 중, left 또는 right 중 하나를 잘못 옮겨 적었다. 후에 몇몇 입력에 대해 insertion 연산이 작동하지 않는(segmentation fault) 것을 발견하여 삽입 과정을 일일이 그려가며 코드의 잘못된 부분을 찾았다.

2. 교재의 의사코드 중 else가 누락된 부분이 몇몇 있어, else 없이 작성하였다가 프로그램이 오작동하는 것을 발견하였다. 나중에 else를 추가해 줌으로써 오류를 해결하였다.

3. 삭제 과정에서 크기 정보를 업데이트하는 알고리즘의 구현 과정에서, 양쪽의 자식을 갖는 노드의 삭제에 대한 알고리즘을 잘못 구현하여 크기가 제대로 계산되지 않았다. 최소 노드는 왼쪽 자식은 없으나 오른쪽 자식은 있을 수도 있는데, 자식이 없는 경우만 고려하여 잘못된 결과를 얻었다. 이 경우도 삭제 과정을 일일이 그려가며 잘못된 부분을 찾았다.

[checker 프로그램]

checker 프로그램은 배열 arr[1...999]를 이용해 입력 리스트에 대한 결과를 계산하고, 이를 해당되는 'output'의 결과와 비교하여 RB_OSTree 프로그램이 잘 작동하는지를 검사한다. 여기서 output은 RB_OSTree 프로그램의 출력 파일이다. 문제에서 키의 범위는 1부터 999까지이지만 실제 구현에서는 피연산자를 그대로 인덱스로 사용하기 위해 편의상 arr[0...999]로 구현하였다. (arr[0]은 사용하지 않는다.) 또한 배열의 크기를 나타내는 변수를 따로 두어 배열에 저장된 키의 개수를 저장하였다. 구현 방법은 다음과 같다.

먼저, 배열 arr[0...999]를 동적 할당하였다. 배열의 각 원소는 해당되는 인덱스 값의 키를 갖는 노드가 있을 경우 1, 그렇지 않으면 0을 나타낸다. 모두 0으로 초기화하기 위해 동적 할당 시 calloc 함수를 이용하였다. 이후 연산자와 피연산자를 입력 파일로부터, 결과값을 출력 파일로부터 한 줄씩 읽어 각 함수의 경우에 대해 결과를 검사한다.

- Insert

해당 인덱스의 원소 값이 0일 경우, 결과값과 피연산자가 같아야 한다. 이 경우 원소에 1을 저장하고, 크기를 1 증가시킨다. 해당 인덱스의 원소 값이 1일 경우, 결과값은 0이어야 한다.

- Delete

해당 인덱스의 원소 값이 1일 경우, 결과값과 피연산자가 같아야 한다. 이 경우 원소에 0을 저장하고, 크기를 1 감소시킨다. 해당 인덱스의 원소 값이 0일 경우, 결과값은 0이어야 한다.

- Select

피연산자가 배열의 크기보다 크다면 결과값은 0이어야 한다. 그렇지 않을 경우, 배열의 처음부터 원소들을 더하는 과정을 총 합이 피연산자와 같을 때까지 반복한다. 최종적으로 더해진 원소의 인덱스가 결과값과 같아야 한다.

- Rank

해당 인덱스의 원소 값이 1일 경우, 배열의 처음부터 해당 인덱스까지 1인 원소가 몇 개 존재하는지 센다. 그 값이 결과값과 같아야 한다. 해당 인덱스의 원소 값이 0일 경우, 결과값은 0이어야 한다.

조건을 위반하는 경우, 프로그램은 에러 메시지를 출력하고 끝난다. 입력 파일의 끝까지 모든 검사를 전부 통과하는 경우 검증 과정을 통과했다는 메시지를 출력하고 끝난다. 에러 메시지의 포맷은 **[실행의 예]**에서 확인할 수 있다.

[리눅스 환경에서 컴파일 방법]

RB_OSTree(order-statistic tree 연산의 실행 파일)

```
$ gcc -o RB_OSTree main.c fcns.c header.h
```

checker(테스트 프로그램의 실행 파일)

```
$ gcc -o checker checker.c
```

[리눅스 환경에서 실행 방법]

RB_OSTree 실행

```
$ ./RB_OSTree input
```

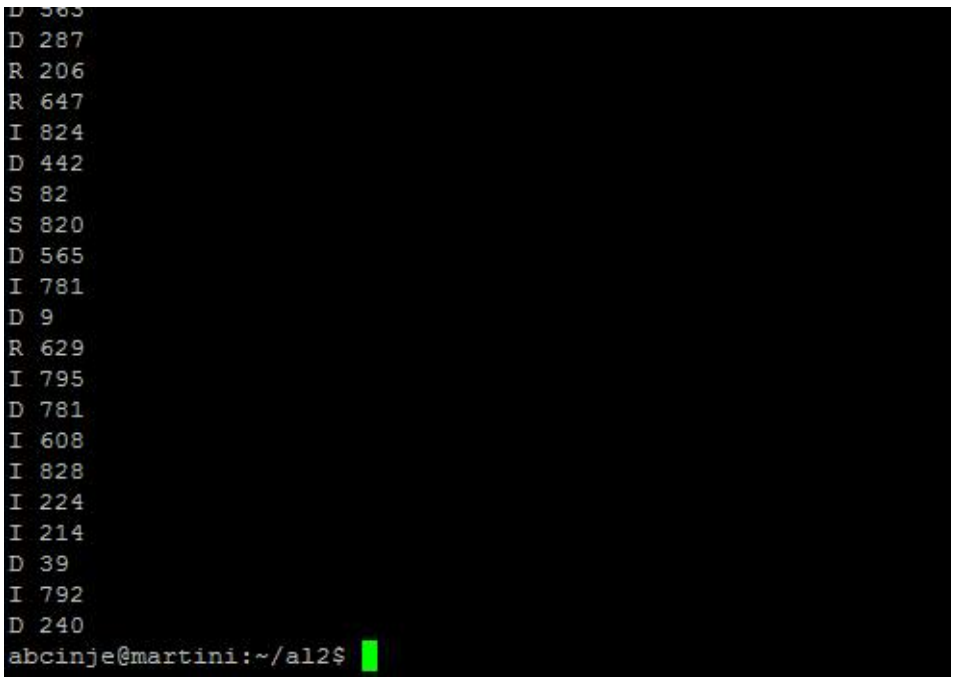
checker 실행

```
$ ./checker input output
```

(input은 주어진 입력이며, output은 RB_OSTree 프로그램의 출력 파일의 이름임.)

[실행의 예]

테스트 케이스로 input1, input2, input3 파일이 있다. 실행 방법을 참고하여 RB_OSTree 프로그램을 실행하면 다음과 같이 입력 리스트를 출력하고 종료됨을 확인할 수 있다. (본 프로그램은 output 파일에 결과를 출력한다.)



RB_OSTree가 잘 구현되었다면 checker 프로그램을 실행하여 다음과 같은 결과를 얻는다.

```
abcinje@martini:~/al2$ ./checker input1 output
Passed correctness verification.
```

틀렸을 경우, 다음과 같은 결과를 얻는다. (이 경우 입력 리스트의 2994번째 줄의 연산 S 152 결과는 379가 되어야 하나, 362가 잘못 출력되었음을 알 수 있다)

```
abcinje@martini:~/al2$ ./checker input1 wrong_output
line 2994 (S 152) : Output should be 379, not 362.
```