**알고리즘 설계보고서**

**학번: 12151362**

**이름: 김성원**

**과목명: 알고리즘\_1분반**

**이메일: sungone95@naver.com**

**1. 개요**

**설계목적**

레드블랙트리(red-black tree)를 이용하여 환자들의 정보를 효율적으로 관리할 수 있는 프로그램을 설계한다.

**요구사항**

이 프로그램을 통해 환자 관리를 위한 몇 가지 기능들을 수행하고 기능에 대한 정보를 표준입력으로 입력받은 뒤, 수행한 결과를 표준출력으로 출력하는 프로그램을 구현한다.

**개발 환경**

visual studio에서 c++언어 사용

**2. 필요한 자료구조 및 기능**

**필요한 자료구조 및 기능**

**red-black-tree** - 환자의 정보를 전체적으로 관리하는 큰 틀로서의 역할을 하는 자료구조

**node** - 환자 개개인의 정보를 저장하고 이를 red-black-tree에 저장하기 위한 자료구조

**vector** - 모든 node들을 저장하는데 쓰고, 유행병을 조사할때 쓰기위한 자료구조

**pair** - 환자의 (병명,money)의 짝으로 자료를 벡터에 저장하기 위해 짝지어주는 자료구조

**3. 기능별 알고리즘 명세**

**1) node class의 객체 생성**

각 노드에서 생성해야 하는 정보가 상수개이고 모든 과정이 상수시간안에 수행가능하므로 상수시간에 1개의 node를 생성하는 과정이 실행가능하다. 따라서 node 1개의 생성과정은 **O(1)**에 bound 된다고 할 수 있다. **->O(1) time bound**

**2) RED-BLACK-TREE - insert함수**

**RED-BLACK-TREE**에 노드를 입력하는 함수이다. 우선 입력받은 정보를 통해 새로운 노드를 생성한다 **-> O(1) time bound**

그 후 **RED-BLACK-TREE** 해당 노드의 위치를 찾는다. 이 과정에서 **RED-BLACK-TREE는** 자식노드가 2개인 구조이므로 위치탐색의 단계가 진행될때마다 탐색해야할 노드의 갯수가 절반씩 줄어든다는 의미이다. 따라서 RED-BLACK-TREE에 있는 노드의 갯수가 N이라고 할때 탐색과정은 log(N)(밑이2인 로그)에 bound 됨을 알 수있다. **-> O(logN) time bound (밑이 2)**

위치를 찾은후, 삽입해야할 노드와 그 부모와의 관계를 설정해줘야 한다. 이는 노드 2개의 포인터만 조정하면 되므로 상수시간에 bound됨을 알 수 있다. **-> O(1) time bound**

**3)** **RED-BLACK-TREE - find함수**

임의의 node형 포인터변수를 선언하고 입력받은 환자번호에 해당하는 노드를 찾는 과정이다.

insert에서 해당 노드의 위치를 찾는 과정과 탐색과정은 구조적으로 동일하므로 RED-BLACK-TREE에 노드의 갯수가 N개 있다면 log(N)(밑이2인 로그)에 bound 됨을 알 수있다.

**-> O(logN) time bound (밑이 2)**

**4) RED-BLACK-TREE - check함수**

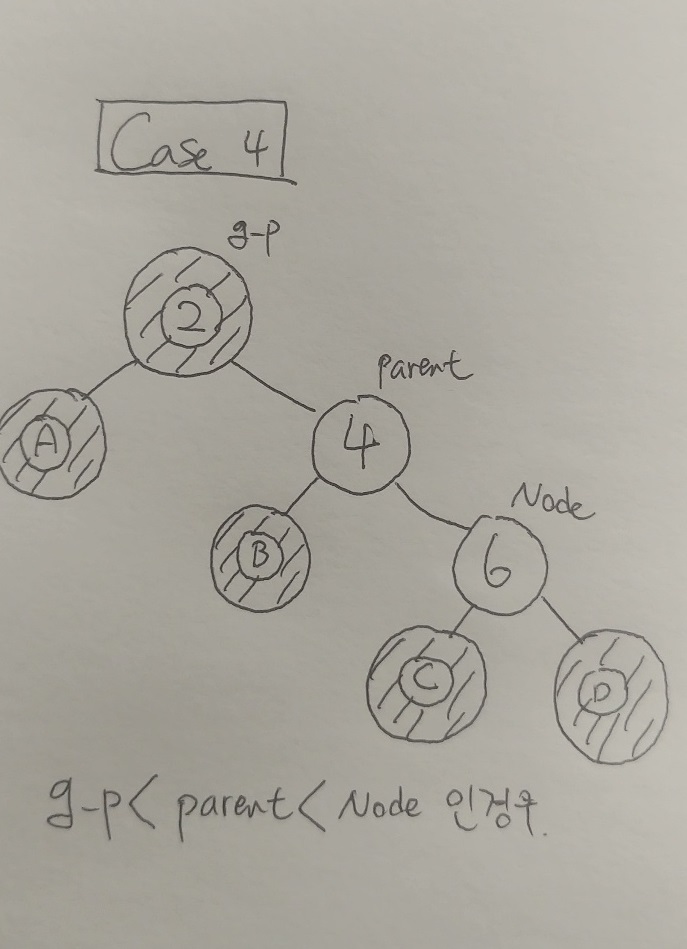
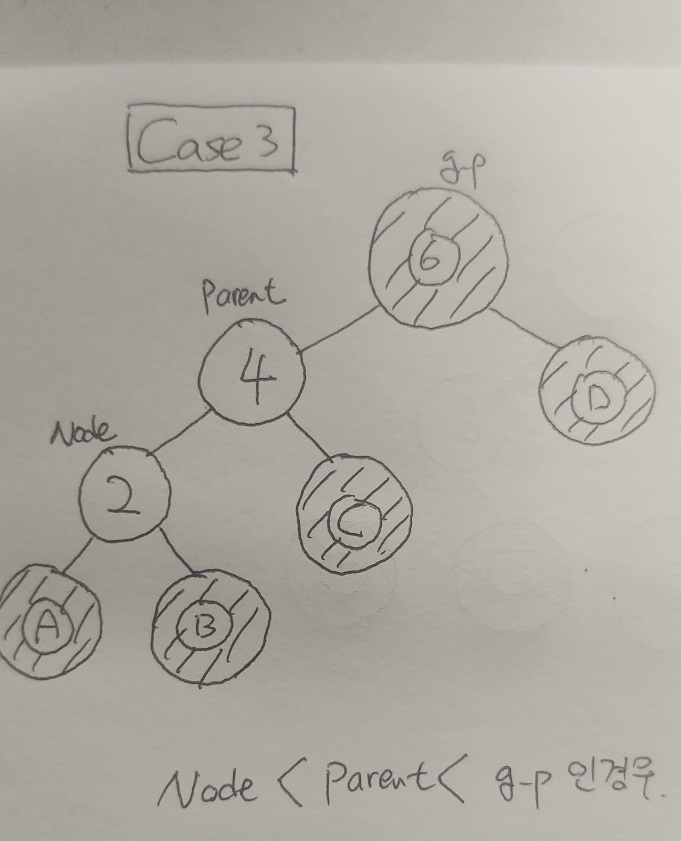
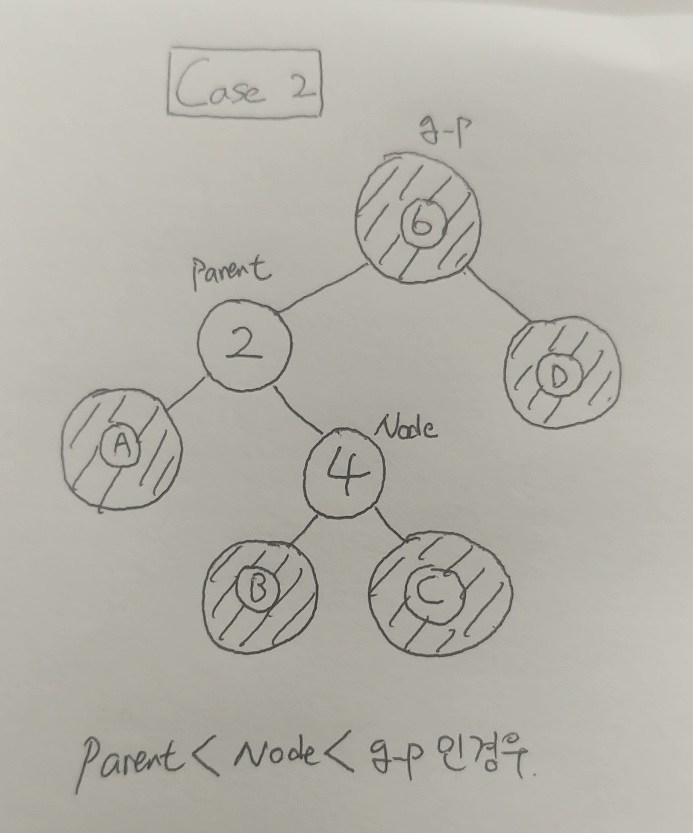
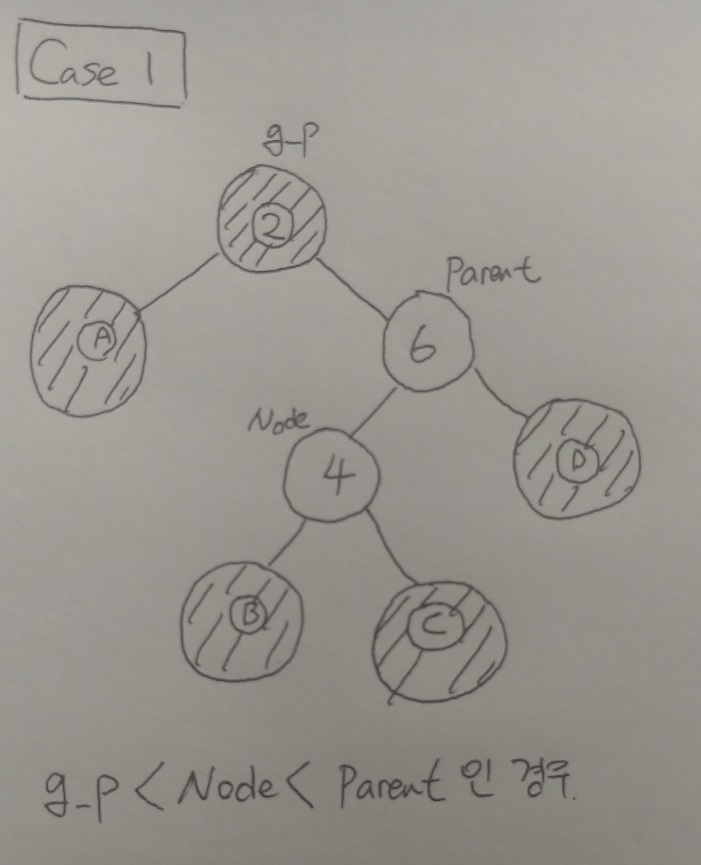
입력받은 유행병의 환자수를 조사할 때 쓰는 함수이고 이는 node의 입력을 받을 때 list벡터에 저장된 노드들을 순차 탐색하는 알고리즘이다. 따라서 벡터에 노드의 갯수 N개가 있다고 할때 이를 순차탐색 하게되면 O(N)에 bound 됨을 알 수 있다. **->O(N) time bound**

**5) RED-BLACK-TREE - d\_reset함수**

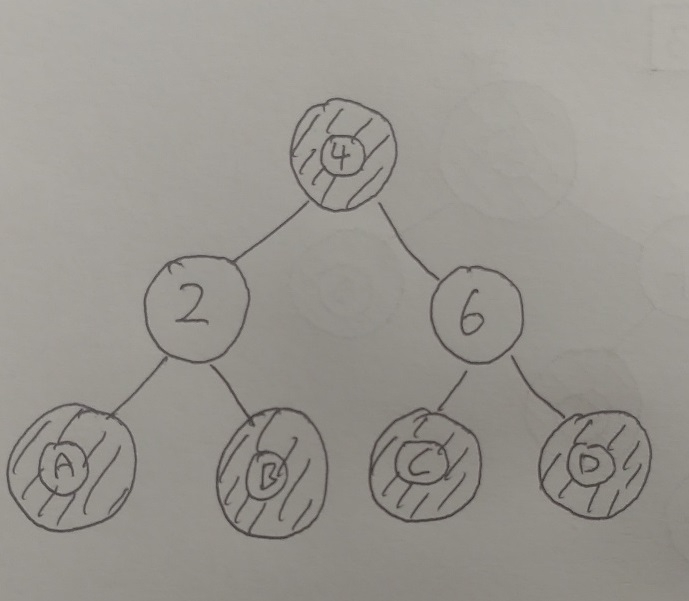
RED-BLACK-TREE 위치를 조정한 후에 해당 subtree 노드의 깊이를 확인후 재조정 하는 함수이다. 이 함수의 경우 n개의 input이 들어올 수 있고 각 경우마다 case5의 경우 최악으로 Tree의 root까지 올라가 d\_reset을 호출 할수 있으므로 O(n)에 가까운 연산을 수행할 수 있다. 따라서 경우에 따라 O(n2)에 가까운 수의 연산을 할 수 있다.

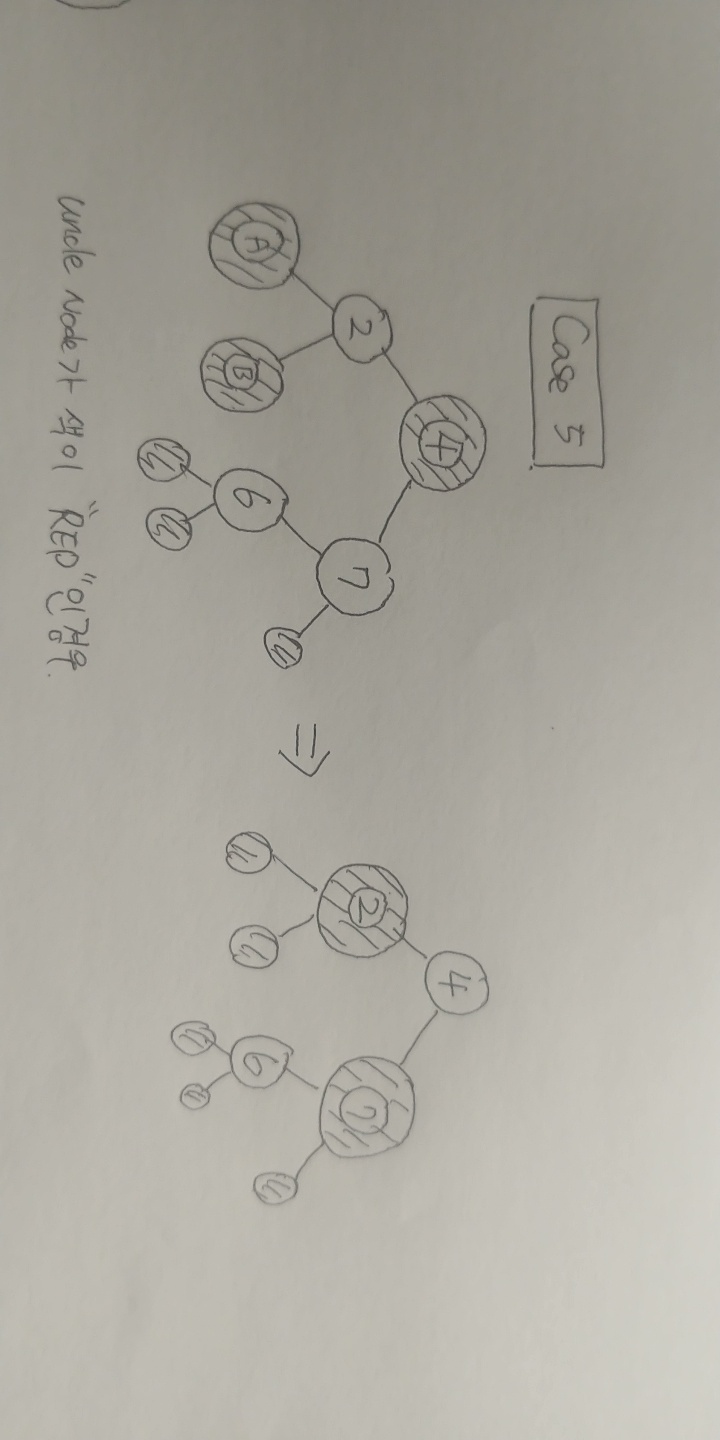
**6) RED-BLACK-TREE - st\_change함수**

RED-BLACK-TREE에서 부모노드와 새로 insert된 노드가 모두 색깔이 RED일때 경우를 나눠 구조를 재조정해주는 함수이다. case1~case2일 경우에는 상수시간에 수행가능한 subtree만 구조가 변하므로 O(1)에 bound되지만 case4인 경우 subtree의 root노드가 반복적으로 연속된 RED노드의 색이 나타날수 있으므로 재귀적으로 함수를 호출해야하는 경우가 있을 수 있다. 이 경우에 Tree의 root노드까지 재귀호출을 할 수 있으므로 이런경우 O(logn)(밑이2)에 bound됨을 알 수 있다.

****

**st\_change함수를 통해 구조를 바꿔야 하는 case1~ case4까지 예시 이며 이 경우들은 모두 Node의 uncle node의 색이 black이다.**

** 구조를 변경한 상태는 모두 이와 같다.**

****

**case5는 Node의 uncle node의 색이 red인 경우이며 위 그림과 같이 구조는 변경하지 않고**

**uncle node, parent node, g\_p node의 색만 각가 변경해주고 g\_p의 노드를 재귀적으로**

**st\_change함수를 통해 검사해준다.**

**4. 인터페이스 및 사용법**

**1) 간단한 사용법 설명**

환자의 정보를 관리하는 시스템으로 처음에 확인할 정보의 갯수N를 입력한후, N만큼의 질문을 한다. 질문의 종류는 **I(신규가입), F(환자검색), A(추가진료), E(유행병 조사)**가 있다.

**I 입력시** “I K(환자번호) N(이름) H(연락처) Ax(주소좌표) Ay(주소좌표) DI(병명) C(진료비)”와 같이 환자와 관련된 추가정보를 입력해야 하며 이에 따른 출력은 트리에서 “D(트리에서 노드의 깊이) S(승인가능 여부)를 출력한다.”

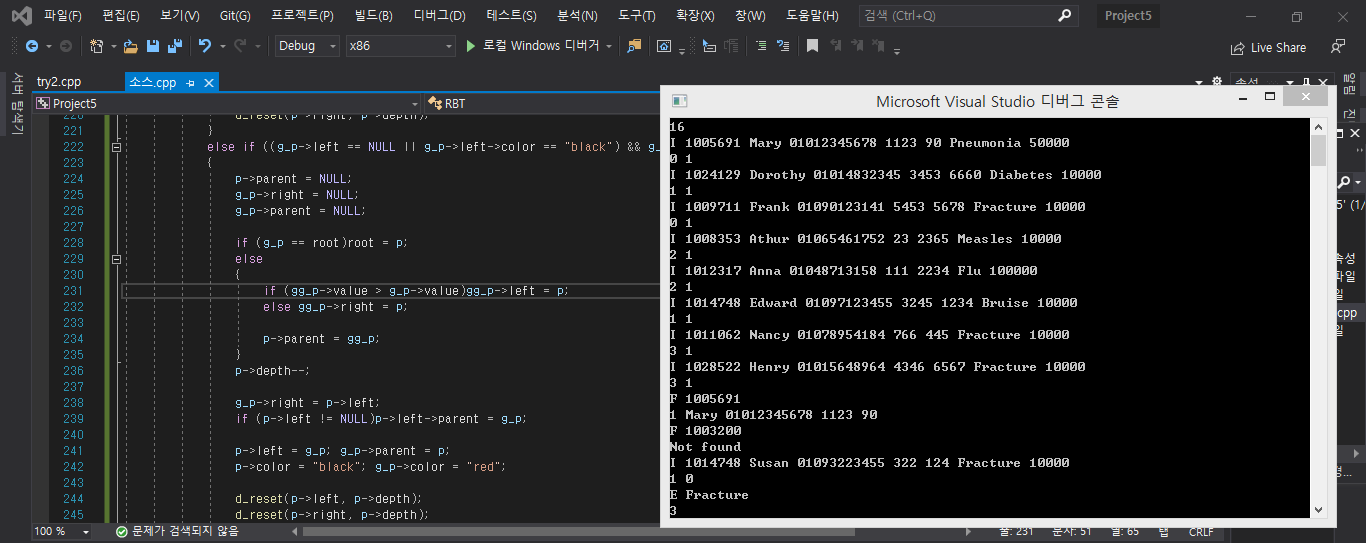
**F 입력시** “F K(환자번호)”와 같이 입력후 출력은 찾았을 경우, “D N H Ax Ay”와 같이 나타내고

못 찾았을 경우 “Not found”를 출력한다.

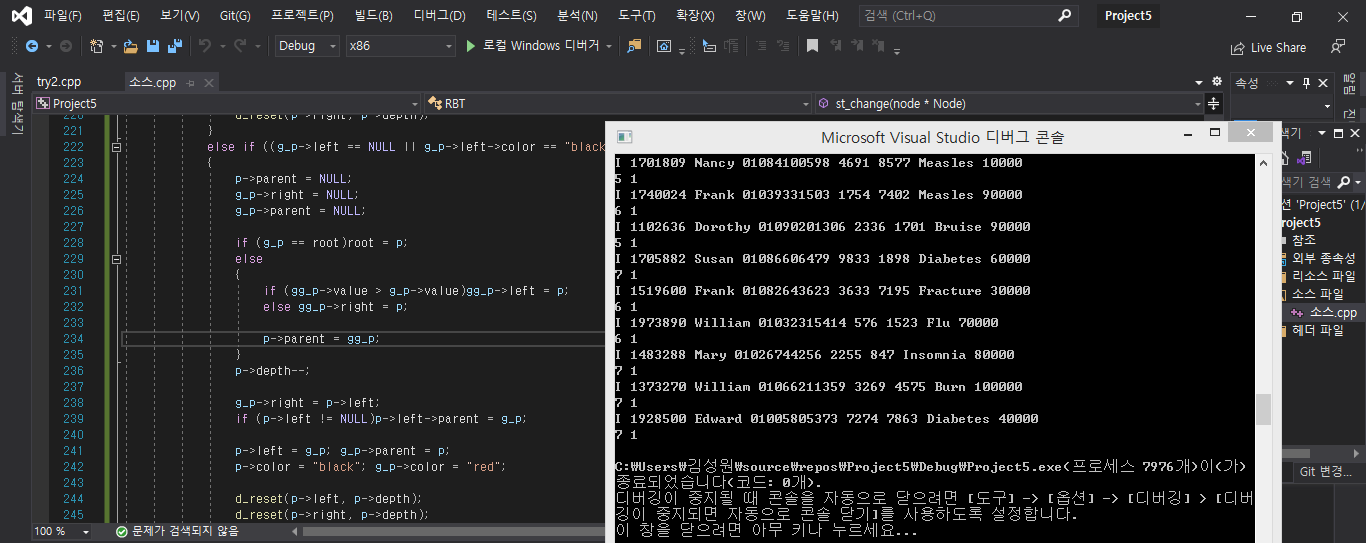
**A 입력시** “A K DI(병명) C(진료비)”의 형태로 추가정보를 입력하면 입력받은 진료내용을 진료기록에 추가후에 “D(해당 노드의 깊이)” 또는 “Not found”를 출력한다.

**E 입력시** “E DI”의 형태로 입력 후 지금까지 입력된 환자들의 마지막 병명을 기준으로 집계를 하고 그 수를 출력한다.

**2)실행 화면 캡쳐 포함 (예시1 input 16개)**



**(예시2 input 100개)**

****

**5. 평가 및 개선 방향**

**- 구현한 알고리즘의 장점**

레드 블랙 트리란 이진트리의 구조를 기본으로 하며 각 노드별로 색(BLACK or RED)의 속성을 추가함으로서 전체적으로 균형있게 만들어진 트리이다. 노드별 색을 통해 트리의 균형(balance)를 조정하기 때문에 input의 개수 n이 아무리 많아도 입력, 탐색, 삭제의 과정이 logn(밑이 2)의 시간으로 수행이 가능하다. 따라서 input의 갯수가 많은 실생활의 적용에 매우 효율적이다.

**- 구현한 알고리즘의 단점**

구조를 엄격하게 관리하여 삽입,삭제,탐색 연산이 모두 logn(밑이2)에 bound되어 정보의 input수가 많을 수록 효율적이지만 input의 수가 적을때 굳이 자료의 구조를 엄격하게 다룬다면 오히려 더 비효율적일 수 있다.

**- 향후 개선 방향**

출력에서 node의 depth를 구하는 경우가 많이 있는데 현재 알고리즘상으로 구현을 한다면 구조를 바꿀때마다 d\_reset(구조변화에 의해 깊이를 재지정 해주는 함수)함수를 호출하게 된다. 이러한 함수는 최악의 경우 가장많은 연산시간을 소요할 수 있으므로 이와 관련된 함수의 효율성 개선이 필요해 보인다.

**-구현 및 개발에 있어 특이 사항**

다른 부분에서는 쉽게 구현할 수 있었지만 새로운 노드를 insert하는 과정에서 구조를 조정하는 부분의 구현이 복잡하고 까다로웠다.