**Red-Black Tree**

알고리즘

학 번 12191659

이 름 이화연

이 메 일 lhylhy0416@naver.com

소속학과 컴퓨터공학과

교 수 님 김영호 교수님

제출일자 2021.11.14

**1. 개요**

1) 설계 목적

애플리케이션을 효과적으로 관리하기 위함이다. balanced binary search tree인 red-black tree를 사용함으로써 애플리케이션을 ID별로 차례로 정리한다. 또한 일렬로 정렬하는 것이 아닌 tree 형태로 정렬하기 때문에 node의 수, 즉 애플리케이션의 수가 n이라고 한다면, 기능들이 최대한 O(lg n) 이내로 실행되도록 한다.

2) 요구사항

(1) 애플리케이션 등록

입력된 애플리케이션 정보를 새로운 node에 저장하고 depth를 출력한다. 만약 이미 있는 ID라면 새로운 node를 만들지 않고 그 ID를 가진 node의 depth를 출력한다.

(2) 애플리케이션 검색

입력된 ID를 가진 node가 있다면 그 node의 depth, 이름, 용량, 가격을 출력한다. 만약 해당 node가 존재하지 않는다면 NULL을 출력한다.

(3) 애플리케이션 업데이트

입력된 ID를 가진 node가 있다면 그 node의 이름, 용량, 가격을 입력된 내용을 업데이트하고 해당 node의 depth를 출력한다. 만약 해당 node가 존재하지 않는다면 NULL을 출력한다.

(4) 애플리케이션 할인

입력된 범위 내의 ID를 가진 node들의 애플리케이션 가격에 할인을 적용한다. 할인율이 P라고 한다면, 해당되는 각 애플리케이션의 가격에 P%의 할인율을 적용한다.

3) 개발 환경 등

개발환경: window10 64x

수행환경: Ubuntu 18.04 (64-bit)

개발도구: visual studio 2019

**2. 필요한 자료구조 및 기능**

필요한 자료구조는 red-black tree이다. red-black tree의 특징은 다음 다섯 가지가 있다.

(1) 모든 node는 red 또는 black의 색을 갖는다.

(2) root node는 항상 black이다.

(3) 새로 삽입되는 node의 색은 red이다. (root node 제외)

(4) red node의 자식은 모두 black이다. (즉, double red가 불가능하다.)

(5) 모든 leaf node에서의 black depth는 같다.

*\* black depth란 depth를 구할 때 black node만을 고려한 것*

binary search tree에서 ‘자신보다 작은 데이터는 왼쪽 자식에, 자신보다 큰 데이터는 오른쪽 자식에 있다.’는 특징이 있다. 이에 따른 기능으로 애플리케이션이 ID의 크기에 따라 차례로 저장될 수 있다.

red-black tree는 위의 특징에 의해 balanced binary search tree이고, 이에 따라 tree의 height가 최소화된다. 이에 따른 기능으로 알고리즘의 수행시간이 줄어들게 된다.

**3. 기능별 알고리즘 명세**

1) 애플리케이션 등록

애플리케이션 등록을 위해서는 새로운 node를 삽입해야 한다.

tree가 비어 있는 경우에는 root node에 새로운 node를 삽입하고 색을 black으로 바꿔주면 되기 때문에 상수시간에 수행된다. 즉, 시간 복잡도가 O(1)이다.

tree가 비어 있지 않은 경우에는 새로 삽입되는 node의 자리를 찾기 위해 root node에서 leaf node까지 순회하기 때문에 O(lg n)의 시간 복잡도를 가지게 된다. 새로운 node를 삽입한 후 double red라면 node를 recoloring하거나 restructuring을 수행해야 한다.

recoloring의 경우, recoloring을 수행한 후에도 계속해서 double red가 발생할 수 있기 때문에 최악의 경우 O(lg n)의 시간 복잡도를 가지게 된다.

restructuring의 경우, 한 번의 restructuring으로 double red가 사라지게 되므로 상수시간에 수행된다. 즉, 시간 복잡도는 O(1)이다.

node 삽입 과정이 모두 끝난 후, depth 출력하기 위해 depth를 구해야 한다. 이 때, 해당 node에서 root node까지 순회하게 된다. 해당 node가 leaf node일 때 최악의 경우이고, 시간 복잡도는 O(lg n)이다.

**따라서, 애플리케이션 등록의 최악 시간 복잡도는 O(lg n)이다.**

2) 애플리케이션 검색

애플리케이션 검색을 위해서는 입력된 ID를 가진 node를 찾아야 한다. 이 경우 binary search tree의 ‘자신보다 작은 데이터는 왼쪽 자식에, 자신보다 큰 데이터는 오른쪽 자식에 있다.’와 같은 특징으로 인해 모든 node를 순회할 필요가 없다. 최악의 경우는 root node부터 leaf node까지 순회하는 것이다. 즉, 해당 node를 찾는데 있어서 O(lg n)의 시간 복잡도를 가진다.

해당 node를 찾은 후 depth, 이름, 용량, 가격을 출력해야 한다. depth는 위에서 언급한 것처럼 O(lg n)의 시간 복잡도를 가지고 나머지는 전부 상수시간에 실행된다.

**따라서, 애플리케이션 검색의 최악 시간 복잡도는 O(lg n)이다.**

3) 애플리케이션 업데이트

애플리케이션 업데이트를 위해서는 입력된 ID를 가진 node를 찾아야한다. 이는 위에서 설명한 것과 같이 O(lg n)의 시간 복잡도를 가진다.

해당 ID를 가진 node의 이름, 용량, 가격을 업데이트 하는 것은 모두 상수시간으로 수행된다.그리고 해당 node의 depth를 출력해야 한다. 이 때 depth는 O(lg n)의 시간 복잡도를 가진다.

**따라서, 애플리케이션 업데이트의 최악 시간 복잡도는 O(lg n)이다.**

4) 애플리케이션 할인

애플리케이션 할인을 위해서는 inorder traversal을 수행해야 한다. inorder traversal을 수행하며 방문한 node의 ID가 입력한 범위 이내라면, 가격을 할인해준다.

inorder traversal의 경우 모든 node를 순회해야 하므로 시간 복잡도는 O(n)이다.

**따라서, 애플리케이션 할인의 최악 시간 복잡도는 O(n)이다.**

**4. 인터페이스 및 사용법**

기능을 이용하기 전, 질의의 수를 입력한다.

(1) 애플리케이션 등록

명령어로 ‘I’가 입력된다면 실행된다.

input: ID, 이름, 용량, 가격

output: 해당 애플리케이션 정보가 저장된 node의 depth

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

(2) 애플리케이션 검색

명령어로 ‘F’가 입력된다면 실행된다.

input: ID

output: 해당 애플리케이션 정보가 저장된 node의 depth, 이름, 용량, 가격

또는 NULL

 

(3) 애플리케이션 업데이트

명령어로 ‘R’이 입력된다면 실행된다.

input: ID, 이름, 용량, 가격

output: 해당 애플리케이션 정보가 저장된 node의 depth

또는 NULL

텍스트이(가) 표시된 사진

자동 생성된 설명

(4) 애플리케이션 할인

명령어로 ‘D’가 입력된다면 실행된다.

input: 범위의 시작 값, 범위의 끝 값, 할인율

output: 없음

텍스트이(가) 표시된 사진

자동 생성된 설명

**5. 평가 및 개선 방향**

1) 구현한 알고리즘의 장점

(1) 알고리즘이 native하게 구현되었다. 따라서 각각의 함수들이 무슨 기능을 하는 지 쉽게 알 수 있다.

(2) 각각의 node는 서로 다른 객체에 저장되어 있으므로 node들을 관리하기 쉽다.

(3) 애플리케이션 할인을 제외한 다른 기능들은 O(lg n)의 시간 복잡도를 가지므로 빠른 시간에 기능을 수행할 수 있다.

2) 구현한 알고리즘의 단점

(1) node 클래스 내에 grandparent와 uncle 관계를 가지는 node가 없어 사용하려면 여러 번 접근해야 한다.

(2) 애플리케이션 할인 기능이 O(n)의 시간 복잡도를 가져 느리게 수행된다.

3) 향후 개선 방향

(1) node 클래스 내의 grandparent와 uncle 관계를 가지는 node를 추가한다. restructuring을 수행할 때, grandparent와 uncle를 포함한 node들 간의 연결에 대해 더 자세히 배우고 개선하도록 한다.

(2) 애플리케이션 할인 기능을 수행할 때 모든 node들을 순회하지 않으면서 범위에 해당하는 node만 접근할 수 있는 방법이 있을 것 같다. 아직은 자세히 모르겠지만 이에 대해 찾아보고 개선하도록 한다.

**6. 기타**

알고리즘 구현을 하며 수행 시간이 오래 걸려 time limit이 계속해서 나타났다. 이를 해결하기 위해 반복문은 최소한으로 활용하였다. 공통된 코드들은 중복으로 실행하지 않고 함수를 만들거나 변수를 만들어 코드가 한 번만 실행되도록 하였다. 그리고 depth를 구하거나 입력된 ID의 node를 찾는 함수는 원래 모든 node를 순회하며 기능을 수행하도록 구현되어 있었다. 하지만 tree 형태라는 것을 다시 한 번 상기하고, tree의 height만큼만 순회할 수 있도록 코드를 수정하였다. 그리고 모든 node들을 저장하는 vector가 존재하였는데 함수들의 시간 복잡도를 낮추기 위해 코드를 수정하다 보니 모든 node들을 저장하는 vector를 쓰지 않고 구현할 수 있다는 것을 깨닫게 되었다. 해당 vector를 지워 공간 복잡도도 줄일 수 있었다.