레드-블랙트리를 활용한

알고리즘 앱스토어 관리 프로그램 설계

과목 : 알고리즘

12170595 조영현 ([cyh197@naver.com](mailto:cyh197@naver.com))

1. **개요**

레드블랙트리를 이용하여 앱스토어에 등록된 애플리케이션들을 ID를 기준으로 관리하는 프로그램을 설계 하였습니다.. ID, 이름 ,용량 가격을 표준입력으로 받고 등록, 검색, 업데이트, 할인 등의 기능을 수행하고 그 결과를 표준출력으로 출력하는 프로그램입니다. 해당 프로그램은 C++로 작성 되었습니다..

1. **필요한 자료구조 및 기능**

* 필요한 자료구조

이진탐색트리, 레드블랙트리

* 기능

레드블랙트리는 이진탐색트리의 종류중 하나로서, 입력시 balance를 유지할 수 있도록 해주는 tree입니다. 해당 프로그램을 구현함에 있어 레드블랙트리를 이용하였습니다. 가능한 최소한의 자료구조를 이용하여 구현하고자 노력하였고, 그 외에 사용한 STL은 없습니다.

애플리케이션의 정보를 Node에 함께 담아 구현하였으며, Class를 활용하여 트리를 생성하였습니다.

1. 기능별 알고리즘 명세

* 기능을 위한 알고리즘 설명(복잡도 분석)

1. **leftRotate, rightRotate** : 입력노드를 기준으로 좌(우)회전 하는 기능

Restructuring 수행시 노드를 회전시켜 주기 위해 사용되는 함수입니다.

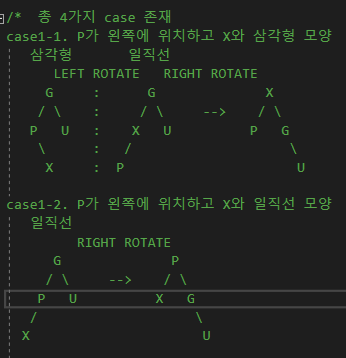
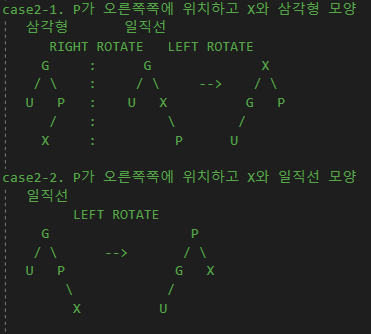
단순히 노드를 이진탐색트리의 성질을 유지하며 시계, 반시계 방향으로 회전시켜주기 때문에 **O(1) time**에 수행됩니다.

1. **resturcure** : double red발생시 retrucuring을 수행

restructure는 더블레드 발생시 삼촌노드가 black인 경우에 실행됩니다.

해당 경우는 크게 4가지로 나뉠 수 있습니다.

<1> 부모가 좌측 위치 (삼각형,직선) <2> 부모가 우측 위치 (삼각형, 직선)

이해를 돕기위해 그림으로 나타내면 위와 같습니다.

* 1. P를 기준으로 left rotate후 G를 기준으로 right rotate를 필요로합니다.
  2. G를 기준으로 right rotate를 필요로 합니다.
  3. P를 기준으로 right rotate후 G를 기준으로 left rotate를 필요로합니다.
  4. G를 기준으로 left rotate를 필요로 합니다.

위 연산은 Rotate를 최대 2번 사용하기 때문에 **O(1) time**에 수행가능합니다.

1. **findDepth** : 해당 ID를 가진 노드의 깊이를 탐색하고 없으면 -1반환

기존 이진탐색트리의 탐색함수와 완전히 동일합니다.

루트에서 시작해 key값보다 작으면 좌측 크면 우측으로 이동하며 노드의 깊이를 탐색합니다.

트리의 높이 h만큼 소요되며 총 n개의 노드가 존재시 **O**(**lg(n)) time**에 수행가능합니다.

해당 함수는 retructuring 이후 트리의 depth를 구하기 위해 사용됩니다.

1. **insert** : 애플리케이션 정보를 가진 노드를 생성하여 트리에 삽입

I를 입력후, 순서대로 ID, Name, Size, Price를 입력받은 뒤 노드를 생성하고

해당 노드가 이미 존재한다면 등록은 거절되며 깊이를 출력합니다. 존재하지 않는다면

해당 노드를 적절한 위치를 탐색후 등록합니다. 이때 탐색과정에서 **O**(**lg(n)) time** 이

소요되고 노드의 부모,자식 관계를 설정하는데 **O(1) time**이, 이때 restrucutre

와 recoloring 을 사용합니다. restructure시 **O(1)time**, recoloring은 부모노드와

삼촌노드를 black으로 조부모 노드를 red로 색상을 변경합니다.

단순히 노드의 색상을 변경하기 때문에 한 번 수행시 **O(1) time**에 수행 가능하나,r

최악의 경우 root까지 전파되어 **o(lg n) time**이 소요됩니다. 마지막으로 균형을

맞춘뒤 depth를 탐색하는 과정을 거치며 **O(lg n)time**이 소요되고 총 **O(3 lg n) time**

이 소요됩니다.

1. search : 해당 ID를 가진 노드 탐색후 애플리케이션 정보 출력

노드를 찾으면 애플리케이션의 Depth, Name, Size, Price를 찾지못하면 NULL 을 출력

합니다. 이진탐색트리와 같이 최악의 경우 트리의 높이 h만큼 소요되며

즉**, O(lg n) time**이 소요됩니다.

1. **update**: 해당 ID를 가진 노드의 name, size, price 변경

ID를 통해 탐색하는 과정에서 **O(lg n) time**, 입력받은 값으로 정보를

변경하는데 **O(1) time**이 소요됩니다. 총 **O(lg n) time**이 소요됩니다.

1. **discount** : 범위에 해당하는 ID를 가진 노드들의 price를 할인

preorder를 통해 모든 노드를 탐색하고,

시작범위와 종료범위 사이에 해당하는 노드들은 입력받은 할인값 p를

적용하여 Price를 수정합니다.

따라서 총 **O(n) time**이 소요됩니다.

8) preorder: 각 노드를 전위순회하며 노드의 정보와 색상을 출력합니다.

1. 인터페이스 및 사용법

* 간단한 사용법 설명

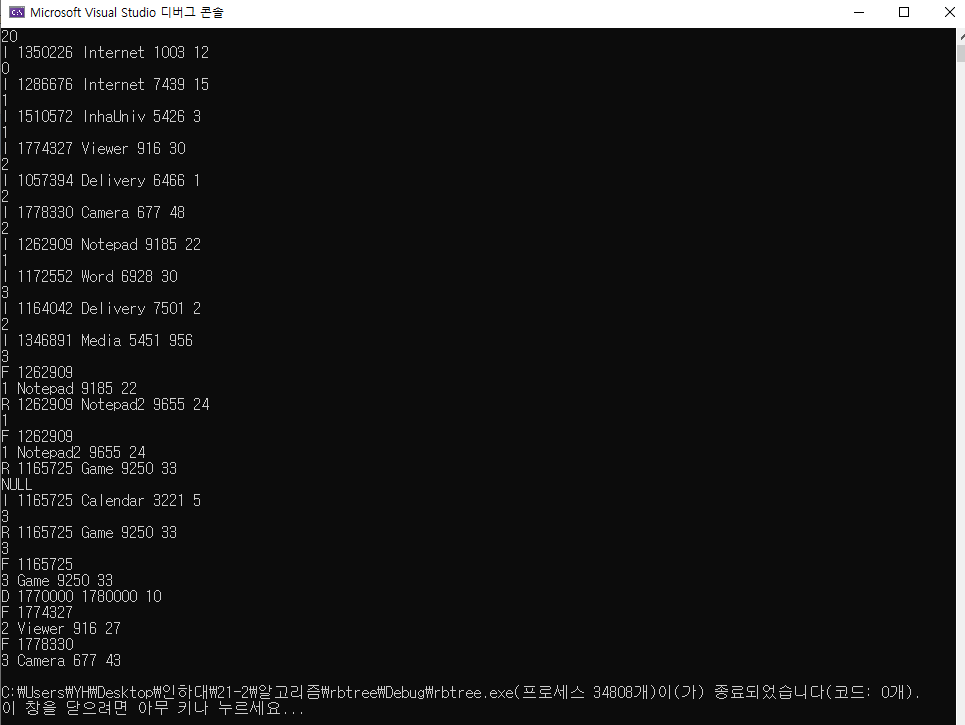
첫번째 줄에 질의의 수를 입력합니다. 이후 아래와 같은 방법으로 사용가능합니다.

등록 : ID Name Size Price => 출력 Depth

탐색 : F ID => 출력 Depth Name Size Price

업데이트 : F ID Name => 출력 Depth or NULL

할인 : D x y p => 출력 없음

* 실행 화면 캡쳐 포함
* 

1. 평가 및 개선 방향

* 구현한 알고리즘의 장점

balanced tree이기 때문에 worst-case에서 일정한 실행 시간 O(lg n)을 보장합니다.

* 구현한 알고리즘의 단점 및 향후 개선 방향

restructure 및 recolor 이후 depth를 출력하도록 구현하였기 때문에 삽입과정에 search가

총 2번 이루어집니다. restructure 및 recoloring을 진행하는 과정에서 depth를 갱신하도록

구현한다면 총 1번의 search로 지금보다 짧은 시간에 수행할 수 있을 것입니다.