알고리즘 설계과제 1(Red-Black Tree)

알고리즘 4분반

컴퓨터공학과

12171679

이준수

이메일 : [ijh1205@naver.com](mailto:ijh1205@naver.com)

1. **개요**

앱스토어 관리 프로그램을 Red-Black Tree를 이용하여 구현하기 위해 본 알고리즘을 설계하였다. Red-Black Tree를 이용하여 주어진 기능에 대한 정보를 표준입력으로 입력 받고, 수행한 결과를 표준출력으로 출력하는 프로그램을 개발하는 것이 요구된다. 또한 질의는 최대 100,000개가 입력되며, 전체 질의에 대해 2초의 제한 시간 이내에 수행되어야 한다. 본 알고리즘은 Visual Studio 2019 환경에서 C++ 언어로 개발되었다.

1. **필요한 자료구조 및 기능**

본 알고리즘에는 vector와 string STL이 사용되었으며, Binary Search Tree를 기반으로 개발되었다. 이 프로그램에서 요구하는 기능은 아래와 같다.

* Red-Black Tree에 새로운 애플리케이션(노드)를 등록(삽입)한 후, 그 노드의 깊이를 출력하는 기능 (이 때 이미 등록된 애플리케이션이라면 등록을 거절하고, 해당 ID를 가지는 노드의 깊이를 출력한다.)
* 특정 ID를 가지는 애플리케이션을 검색하여 이름, 용량, 가격을 출력하는 기능 (해당 노드가 존재하지 않으면 “NULL”을 출력한다.)
* 특정 ID를 가지는 애플리케이션을 찾은 후, 새로운 이름, 용량, 가격을 업데이트하는 기능 (해당 노드가 존재하지 않으면 “NULL”을 출력한다.)
* ID가 x 이상, y 이하인 애플리케이션을 모두 찾은 후, 각 애플리케이션의 가격에 P%의 할인율을 적용하는 기능(소수 부분은 버린다.)

1. **기능별 알고리즘 명세**

* **애플리케이션 등록**

노드의 key인 ID가 들어갈 적절한 위치를 찾는다. root에서부터 시작하여 현재 탐색중인 노드의 ID와 비교하여 노드의 left 또는 right로 내려가거나, 현재 탐색 중인 노드의 ID와 등록할 노드의 ID가 같다면 이미 등록된 애플리케이션이므로 등록을 거절하고 반복문을 빠져나와서 깊이를 출력하고 종료된다. 이 과정의 수행 시간은 O(log n)이다. 노드가 들어갈 적절한 위치를 찾았다면 노드와 부모의 관계를 적절히 설정해주고 recoloring이나 restructuring이 필요한지 확인한다. 이때 restructuring이 발생한다면 해당하는 경우에 1번만 수행하면 되므로 수행 시간은 O(1)이다. Recoloring이 발생한다면 수행 시간은 역시 O(1)이지만, 이때 다시 double red가 발생할 수 있으므로 계속해서 recoloring이 발생한다면 수행 시간은 O(log n)이다. Rebuilding 과정까지 끝나면 등록된 노드의 깊이를 계산하기 위해 해당 노드에서 root까지 올라간다. 이때 수행 시간은 O(log n)이다. 따라서 애플리케이션 등록 기능의 시간 복잡도는 O(log n)이다. 공간 복잡도는 n개의 노드를 사용하고 있으므로 O(n)이다.

* **애플리케이션 검색**

root에서부터 탐색을 시작하여 현재 탐색 중인 노드의 ID와 검색할 노드의 ID를 비교하여 노드의 left 또는 right로 내려간다. 이때 노드의 깊이를 출력하기 위해 깊이를 세는 변수의 값도 1씩 증가시킨다. 그러다가 두 ID가 같다면 검색을 종료하고 빠져나온다. 해당 노드가 존재한다면 깊이, 이름, 용량, 가격을 출력한다. 반대로 트리의 끝까지 탐색했는데도(leaf에 도달했는데도) 노드를 발견하지 못했다면 존재하지 않는 것이므로 “NULL”을 출력한다. 이 알고리즘에서 노드의 left 또는 right로 내려가는 과정이 반복되므로 수행 시간은 O(log n)이다. 따라서 애플리케이션 검색 기능의 시간 복잡도는 O(log n)이다. 공간 복잡도는 n개의 노드를 사용하고 있으므로 O(n)이다.

* **애플리케이션 업데이트**

업데이트 기능은 검색 기능과 거의 유사하다. 차이점은 노드를 찾은 후, 새로운 이름, 용량, 가격으로 정보를 업데이트한 후에 깊이만 출력하는 것이다. 검색 기능과 마찬가지로 노드가 존재하지 않으면 “NULL”을 출력한다. 핵심 알고리즘은 검색 기능과 동일하게 노드의 left 또는 right로 내려가면서 탐색하는 것이므로 애플리케이션 업데이트 기능의 시간 복잡도는 O(log n)이다. 공간 복잡도는 n개의 노드를 사용하고 있으므로 O(n)이다.

* **애플리케이션 할인**

트리의 전위 순회를 구현하여 할인 기능을 구현했다. 모든 노드를 탐색하면서 ID가 x이상 y 이하인 노드를 찾아서 해당 노드의 가격에 P%의 할인율을 적용한다. 전위 순회의 수행 시간은 O(n)이므로 애플리케이션 할인 기능의 시간 복잡도는 O(n)이다. 공간 복잡도는 n개의 노드를 사용하고 있으므로 O(n)이다.

1. **인터페이스 및 사용법**

텍스트이(가) 표시된 사진

자동 생성된 설명

프로그램을 실행하면 질의의 수를 입력받는다. 이후 각 줄에는 질의 정보가 주어진다.

* **I**

애플리케이션을 등록하는 명령어이다. 등록할 애플리케이션의 ID, 이름, 용량, 가격을 입력하여 애플리케이션을 등록한다.

* **F**

애플리케이션을 검색하는 명령어이다. 검색할 애플리케이션의 ID를 입력하여 애플리케이션의 정보를 검색한다.

* **R**

애플리케이션을 업데이트하는 명령어이다. 애플리케이션의 ID, 업데이트할 이름, 업데이트할 용량, 업데이트할 가격을 입력하여 애플리케이션의 정보를 업데이트한다.

* **D**

애플리케이션을 할인하는 명령어이다. 범위 x와 y, 할인율을 입력하여 애플리케이션의 가격에 할인율을 적용한다.

1. **평가 및 개선 방향**

일반적인 Binary Search Tree는 한 쪽으로 치우치면 최대 O(n)의 시간 복잡도를 가질 수 있다. 그러나 본 알고리즘은 Red-Black Tree, 즉 Balanced Binary Search Tree이기 때문에 이러한 단점을 해결할 수 있고, 따라서 노드를 삽입하거나 검색할 때 최악의 경우에도 O(log n)이 보장된다는 것이 장점이다. 그러나 할인 기능을 구현할 때 전위 순회를 사용하여 모든 노드를 탐색하기 때문에 불필요한 노드까지 모두 탐색한다는 단점이 있다. 따라서 향후 본 알고리즘은 특정 노드를 root로 가지는 subtree에서 left subtree는 모두 root보다 key 값이 작고, right subtree는 모두 root보다 key 값이 크다는 Binary Search Tree의 특성을 이용하여 모든 노드를 탐색하지 않고 범위 내의 노드를 찾을 수 있도록 개선될 수 있다.