학과 : 고려대학교 영어영문학과
학번 : 2014130522
이름 : 오현근
제출 일자 : 2016년 6월 25일

1. 개요

수업 시간에 배운 자료구조를 이용하여 주어진 기능을 수행하는 프로그램을 구현한다. 구현한 프로그램은 TwittyTidbit이라 명했다.

본 프로젝트의 개발 환경은 다음과 같다.

운영체제	Windows 7
사용 언어	Python 3.5.1

2. 사용한 데이터 구조

본 실습에서는 search, insert, delete가 모두 O(logn)의 시간으로 빠르게 수행 되는 red-black tree 데이터 구조를 채택했다. red-black tree는 class(rbNode, rbTree)로 구현했으며, 본 실습에서는 red black tree 데이터 구조를 이용하여 다섯 개의 dynamic set을 만들어 작동하도록 프로그램을 설계했다. 또한 rbNode() 클래스의 buf 및 buf2 attribute은 Python에서 제공하는 dict 자료형이다. dict 자료형은 hash 구조를 이용하여 element 개수가 적을 때 평균적으로 search, insert, delete operation을 O(1)에 가까운 시간으로 빠르게 수행하기에 이를 채택했다.

다음의 표는 다섯 개의 dynamic set의 attribute들이 각각 무엇을 의미하는 지 보여준다. 하나의 클래스로 서로 다른 이름의 attribute을 지정하는 법을 체득하지 못하여 이 방법을 택했다.

user_rb	① key : 하나의 user id ② buf : key - 해당 user id의 친구 id / value - 의미 없음 ③ buf2 : key - 해당 user가 mention한 단어 / value - 그 횟수 ④ count : user의 총 mention 개수 (==buflen)
word_rb	① key : 하나의 word ② buf : key - 해당 word를 멘션한 user id / value - 그 횟수 ③ count : word의 총 멘션된 횟수 (==buflen)

friend_num_rb	① key : 한 유저의 총 friend 개수 ② string : 그 유저의 id
tweets_per_rb	① key : 한 유저의 총 tweet 횟수 ② string : 그 유저의 id
word_freq_rb	① key : 한 word의 총 멘션된 횟수 ② string : 해당 word
p, left, right, color은 모두 red-black tree 구현을 위한 attribute이다.	

3. 수행시간 예측

n은 tree set에서 노드의 개수이다.

0. Read data files : O(nlogn)
data files을 읽어 들여서 다섯 개의 dynamic set을 build하는 부분이다. readUserTxt, readWordTxt, readFriendTxt는 모두 파일을 읽어 들이면서 line별로 loop을 돌아 동작하고 loop 안에 O(logn) operation이 있기에 O(nlogn)의 시간이 걸린다. buildFriendNumRBtree, buildTweetsPerUserRBtree, buildWordFreqRBtree는 모두 각각 user_rb, word_rb를 inorder traverse하면서 friend_num_rb, tweets_per_rb 및 word_freq_rb를 build하는 함수들이다. tree를 traverse하면서 각 O(logn)의 operation을 수행하기에 총 O(nlogn)의 시간이 걸린다. 각각의 set을 build하는 데 O(nlogn)의 시간이 걸리기 때문에 (6 * O(nlogn)) 총 opertion time complexity는 O(nlogn)로 예측한다.
1. Display statistics : O(1)
buildFriendNumRBtree, buildTweetsPerUserRBtree, buildWordFreqRBtree 함수를 수행하면서 값을 이미 얻어냈기에 O(1)의 시간이 걸린다.
2. Get more detailed statistics : O(logn)
avg값은 O(1)시간이 걸린다. min, max값은 minimum, maximum 메소드를 수행하기에 각각 O(logn)의 시간이 걸린다. (O(1) + O(logn)) 총 O(logn)의 시간이 걸린다.
3. Top 5 most tweeted words : O(logn)
word_freq_rb에서 maximum 값을 찾고(O(logn)), 거기서 predecessor를 다른 값이 4번 나올 때까지 찾도록 한다. (k * O(logn) k는 평균적으로 상수임) 따라서 총 O(logn)의 시간이 걸린다.

4. Top 5 users who tweeted the most : O(logn)
대상 set이 tweets_per_rb인 것을 제외하곤 option 3번째와 동일하게 진행된다.
5. Find users who tweeted a word : O(logn)
입력받은 word를 key로 하는 노드를 word_rb에서 찾고(O(logn)) 그 노드의 buf의 모든 key값을 보여준다. (O(k) - k는 평균적으로 상수) 총 O(logn)이 걸린다.
6. Find all people who are friends of the above users : O(logn)
입력받은 id를 key로 하는 노드를 user_rb에서 찾는다. O(logn) 그 노드의 buf의 모든 key값을 보여준다. (O(k) - k는 평균적으로 상수) 총 O(logn)이 걸린다.
7. Delete all mentions of a word : O(nlogn)
입력받은 word를 key로 하는 노드를 word_rb에서 찾은 뒤 (O(logn)), 그 word를 지운다. (O(logn)) 그리고 user_rb에서 buf2와 count를 조정한다. (O(logn)) 그런 다음 buildFriendNumRBtree, buildTweetsPerUserRBtree, buildWordFreqRBtree를 수행하여 friend_num_rb, tweets_per_rb, word_freq_rb를 다시 build한다. (3 * O(nlogn))
8. Delete all users who mentioned a word : O(nlogn)
입력받은 word를 key로 하는 노드를 word_rb에서 찾은 뒤 (O(logn)), 해당 노드의 buf를 참조하여 해당 word를 사용한 user들을 user_rb에서 모두 찾아가 지운다. 이에 알맞게 word_rb와 user_rb를 조정하면 총 O(n) 시간이 걸린다. 그런 다음 buildFriendNumRBtree, buildTweetsPerUserRBtree, buildWordFreqRBtree를 수행하여 friend_num_rb, tweets_per_rb, word_freq_rb를 다시 build한다. (3 * O(nlogn))

4. 개선 방안

옵션 7번째나 8번째를 선택하여 delete를 한 뒤 word_rb와 user_rb를 조정하고 friend_num_rb, tweets_per_rb, word_freq_rb를 다시 build할 때 O(nlogn)이 걸린다. 이는 delete를 할 때마다 매번 수행되야 하기에 조금 버거운 복잡도이다. word_rb와 user_rb를 조정하는 것처럼 friend_num_rb, tweets_per_rb, word_freq_rb도 O(n) 정도의 시간으로 update 할 수 있으면 성능이 더 개선될 것이다.