Data structure Report

2015410066 김준하

1. What data structure you chose and why

Id 입력 시 바로 id node를 찾고, word 입력 시 바로 word를 찾을 수 있다는 점에 끌려 hash로 짜기 시작했습니다. 이러면 Dynamic set의 가장 기본 연산인 Test 작업이 매우 빨라집니다. Insert, delete도 입력한 id 혹은 word에 대해 test 후 있으면 그 node에 정보를 추가, 없으면 새로운 node를 만들어 hash table에 linked list로 연결하는 작업을 하는데, 이 때 test 과정이 매우 빠를 것입니다. 단점이 있다면 top 5 작업(2,3번 기능) 에 대해서는 입력 받으면서 계속 top 5를 갱신하여 저장하지 않는 이상 빠르게 찾을 방법이 없을 것 같습니다. 이는 insert와 delete를 할 경우 더 복잡해 진다고 판단하여, top 5를 찾는 함수 내에서 BST를 만들어 top 5 word를 찾았습니다. Red black tree는 구현하지 못해 sample2에서 2번 메뉴의 경우 잘 돌아가지 않았습니다. 그래도 dynamic set에서 가장 많이 이루어지는 Test 작업에 있어서의 이점 때문에 hash table을 이용하게 되었습니다.

2. What is your expected performance

Hash table을 mod 100000으로 계산했습니다. 같은 index라면 linked list를 따라 연결 했습니다. 그렇다면 n개의 input이 들어온다면, 거의 O(1)이라고 생각했는데, n이 커지면 linked list의 길이도 커질 것이기 때문에, O(n/100000)이 될 것 같습니다. 2,3번의 경우는 red black tree는 아니지만 Binary Search Tree로 구현해서, 입력에 따라 다르지만, balanced인 경우 nlogn에 찾을 수 있을 것 같습니다. (balanced라는 보장이 없어서 그런지 2번 sample의 top 5 word는 잘 작동하지 않네요)

그 외의 4번의 어떤 단어를 tweet한 user를 찾는 데, hash table에서 단어 node의 pointer 찾는 데 O(n/100000), 그 단어 node를 이용한 사람 찾아 출력하는 데 O(1)만큼 걸려, O(n/100000)으로 수행이 될 것 입니다.

5번의 경우는 입력된 단어를 hash table에서 찾는 데 O(n/1000), 그 단어를 이용한 user를 찾는데 O(1), 그 찾은 user의 친구의 node를 hash table에서 찾는데 O(n/100000), 그 친구를 각각 출력하는데 O(1)이므로, O(n/100000) + O(1)\*O(n/100000)\*O(1) = 약 2\*O(n/100000) 정도에 수행이 될 것입니다.

6번 delete word의 경우는, 입력된 단어를 찾는데 O(n/100000), 단어 삭제하는데 O(1), 단어를 tweet한 user의 tweet한 단어 목록에서 이 단어를 지우는데 있어서, 단어를 tweet한 user id 찾을 때 O(1), user node찾을 때 O(n/100000), user가 tweet한 단어 중 이 단어를 찾는데 O(1), 삭제 시 O(1), 따라서 O(n/100000)+O(1)+O(1)\*O(n/100000)\*O(1)\*O(1)= 마찬가지로 2\*O(n/100000) 정도에 수행될 것 입니다.

7번의 경우도 6번과 비슷하지만, 단어를 찾고 (O(n/100000)), 단어를 언급한 user id를 찾고O(1), 그 user id로 user node를 찾고(O(n/100000)), user node에서 prev로 이 user를 친구로 하는 fr\_user id를 찾고(O(1)), id로 node를 찾고 (O(n/100000)), fr\_user의 친구 목록에서 user를 지우고(O(1)), user node를 지우는 작업까지(O(1)) 복잡한 작업을 걸친다. 하지만 while문 상으로 봤을 때, O(n/100000)+O(1)\*{O(n/100000)+O(1)\*O(n/100000)}+O(1) 정도 걸려 이도 약 2\*O(n/100000)이 걸릴 것입니다.

O(n/100000)는 결국 O(n)이지만, data 크기가 무한대까지 발산하지 않으면 즉, 우리의 sample 정도, 10만이하의 n, 작으면 작을수록 큰 감소가 이루어 진다고 생각합니다. 다시 생각해보면 red black tree보다 뒤쳐지는 면이 있기는 하나, hash라는 것만의 장점을 알고 이해하는 것도 의미있다고 생각합니다.

3.How would you improve the system in the future

아주 기본적인 개념의 hash로 구현했지만, 정말 N이 몇이든 Constant complexity로 구현할 수 있는 hash에 대해서도 알아보고 싶고(사실 구현을 하다 보니, 제가 배운 hash는 O(1)라고 말하기 힘들 것 같다는 생각이 들었습니다.), 이번엔 못했지만, 2,3번에 대해서 linked list로 sort할 수 있는, merge sort 혹은 다른 알고리즘으로 구현하여 tree 없이도 더 좋은 performance로 작동하게 만들 수 있을 것입니다. 또한 비슷한 구조의 함수들을 많이 썼는데, 아직 Class를 몰라 그랬지만, 다형성을 이용하면 훨씬 더 간단한 코드를 작성할 수 있을 것 입니다.

궁금한 점

그리고 교수님께서 어떤 hash를 쓰든 red black tree를 쓰든 둘 다 장단점이 있다고 하셨는데,

(hash는 test에서, red black tree는 sort에서 좋음), evaluation criteria에는 O(nlogn) 에도 점수 부여가 있더라고요! 모든 부분에서 O(nlogn)만에 수행은 안되더라도, 빠르게 처리되는 부분도 있을 것이라 생각해서 처음부터 hash로 짜기 시작해서요. Hash를 이용하면 이 부분에서는 점수를 아예 못 받는 건가요?