Report & self evaluation
최종 제출 코드 파일명은 Main-RevisedMainProject.py 입니다.

2015150018 통계학과 김정욱

1. What data structure you chose and why?

총 두가지 버전을 만들었습니다.
천번째 버전은 Tweet과 User을 모두 Class를 이용해 객체화 시키고 대부분의연결관계를 포인터로 연결하는 방법을 사용했습니다.
첫번째 버젼인 MainProject.py에서는 data structure로 linked list와 binary searched tree를 사용하였습니다. 각 메뉴의 연산 중에는 User을 찾아야 하는 연산이 꽤 많은 비중을 차지하고 있고 이를 그냥 list에 넣을시 User 검색 복잡도는 O(n)이므로 다른 연산과 겹쳐질 때 많이 느려질 수 있기 때문에 이를 효율화 하기 위해 binary search tree를 사용하여 user을 정리하였습니다.

python의 Array(list)는 다른 언어의 Array와는 다르게 크기를 먼저할당해 주지 않아도 새로운 요소를 지속적으로 추가가 가능하기 때문에 사실상 Array의 매서드인 append를 활용하면 더 간단히 tweet를 관리할 수 있습니다. 그러나 이렇게 과제를 해결할 경우 자료구조 수업에서 과제를 내주신 목적에 부합하지 않다고 생각했기때문에 따로 Tweet을 관리하는 linked list class를 만들어 자료를 저장했습니다. User과는 다르게 tweet들은 이를 인식할 수 있는 수치정보가 없었으므로 간단히 linked list를 활용하여 저장하였습니다. 이부분에서 Hashmap을 활용하여 Tweet을 저장할 수도 있었겠지만 미리 Array크기를 지정하고 이에 index를 활용할 수 있는 부분이 python의 Array와는 맞지 않는 부분이 있기 때문에(while로 미리 갯수를 지정해서 None등을 할당해 놓아야함) Hashmap을 사용하지는 않았습니다.

두번째 버전은 Tweet과 User 그리고 Friend 정보를 Array에 담아 관리하는 방법을 사용했습니다.
두번째 버전인 RevisedMainProject.py에서는 data structure로 Array와 binary search, 그리고 heap을 주로 사용하였습니다. python에서는 array 원소의 추가와 삭제가 간편한것이 장점이므로 이를 살려서 보다 효율적으로 자료를 관리하였습니다.
heapsort를 통해서 문자열도 정렬함으로써 top5Tweet 같은 연산도 O(n)복잡도 안에 가능하게 만든것이 특징입니다.

2. What is your expected performance

첫번째 버전 즉 user와 tweet을 binary search tree와 linked list로 관리한 버전은 user객체의 pointer을 찾는데에 O(logn) 복잡도를 기대했지만 주어진 User 데이터가 모두 id순으로 정렬되어있었던 탓에 사실상 O(n)의 성능과 같았습니다.

0 번째 메뉴인 데이터 읽기연산은 user, friend, tweet 데이터를 읽는데 O(n)의 복잡도를, 이를 모두 삽입하는데 각각 O(nlogn), O(nlogn) 그리고 O(n)의 시간 복잡도를 가질 것이라고 기대했습니다. 그러나 실제로 앞의 두 연산은 사실상 O(n^2)의 복잡도를 가지는것으로 보입니다.

두번째 버전은 편중되는 데이터에 상관없이 heapsort를 이용하여 데이터를 정렬한후 binary search를 활용하여 데이터를 검색하는 방법을 사용했습니다. 이는 모두 O(logn)에 데이터를 찾을수 있는 방법이라 기대합니다.
또 이 특징을 살려서 sort만 시켜준다면 unique value를 찾는 일이나 해당 value의 반복수를 세는 일을 동시에 O(nlogn) <= (정렬O(nlogn) + uniquevalue와 반복수 카운팅 O(n))만에 끝낼수 있으리라 기대합니다. 이를 통하면 0~3까지의 메뉴를 모두 O(nlogn)시간 안에 해결할 수 있습니다. 4~7까지의 메뉴는 이런 기본적 연산을 n보다 확실히 적은 양의 횟수로 반복하는 것이기 때문에 최악의 경우에는 O((n^2)logn)까지 시간이 걸릴 수 있지만 현실적으로는 O(nlogn)증가속도의 성능을 보일것입니다.
문제는 graph 알고리즘인데, 노드와 노드간에 서로 연결정보를 모두 등록하려면 최악의 경우 O(n^2)의 시간 복잡도가 된다. 그런데 이번 과제에서 제공된 데이터를 그대로 사용할 경우 데이터 정렬 과정을 거친다고 해도 사실상 O((n^2)logn)의 시간 복잡도가 나옵니다. 이 경우 처음 데이터를 읽을때부터 node를 만들어준다면 조금의 나아짐이 있을 수는 있을 것 같습니다.

3. How would you improve the system in the future

이번 과제에 한정해서는 해당 데이터를 Array에 두는것이 효율적이었지만, 데이터가 많아질수록 그리고 user들의 숫자가 많아질수록 일반데이터 중심의 관리보다 사용자 중심의 데이터를 객체화, 캡슐화 하는것이 유지보수, 보안 측면에서 더 좋을 것이라고 생각합니다. 따라서 User와 tweet정보를 모두 객체로 만들고, User은 red-black tree를 활용하여 정렬된 순서로 계정이 생성되더라도 효율성을 유지할 수 있게, 그리고 tweet은 HashMap으로 관리하여 tweet의 검색과 삭제를 효율적으로 만들고싶습니다. 또한 graph관련 알고리즘의 시간복잡도 개선을 위해 서도 객체와 링크로만 구성된 dataset을 구성해 보고싶습니다.

4. Self evaluation

submit a github account (10/10)
commit source code displaying menu (10/10)
commit the first draft of manual (10/10)
read data files(20/20)
statistics(20/20)
Top 5 most tweeted words (10/10)
Top 5 most tweeted users(5/5)
Find all users who mentioned a word (10/10)
Find all users who are friend of the above user(5/5)
Top 5 strongly connected components(6/10)

Find shortest path from a user (9/10)