알고리즘 설계과제 1 보고서

[게임사 회원 관리 프로그램 설계]

010–8578-7628

12131489 김영훈

12131489@inha.edu

1. 개요

이 프로그램은 게임사 회원 관리를 위한 프로그램이다. 설계의 목적은 많은 수의 회원을 효율적으로 관리하고, 각 기능에 대해 빠른 속도를 보장하는 것이다. 이를 위해서 Tree의 깊이가 균형되도록 유지시키는 Red-Black Tree 자료구조를 이용하였다. 삽입, 탐색 기능을 제공하고, input size가 N일 때 최악의 경우에도 시간복잡도 O(logN)을 제공한다.

프로그램을 설계하기 위해, 프로그램은 운영체제 Windows 10에서 개발도구 Microsoft Visual Studio Community 2017을 이용하여 C++ 언어로 개발하였다.

2. 필요한 자료구조 및 기능

- 필요한 자료구조

1. Member

- 회원에 대한 정보 저장. 회원등급 설정, 금액의 변동 기록 및 출력 등의 기능 제공

2. Node

- RBTree 내 구성되는 노드
- left, right, parent 노드의 포인터와 color 및 Member에 대한 정보를 가짐

3. RBTree

- Red-Black Tree로 Tree의 깊이를 일정하게 유지
- Tree 구조에 Member 정보에 대해 삽입, 탐색, 출력 기능 제공

4. Ground

- ground_idx, ground_price 배열을 통해 땅에 대한 주인에 대한 회원번호와 땅값을 저장
- 땅 구매 기능 제공

5. Rich

- 현재금액이 가장 높은 상위 5명에 대한 정보를 업데이트 및 출력, 관리

6. Priority_queue

- value, key에 대한 pair element
- 변형된 최소값 우선순위 큐로 최소값 출력 등 기능 제공

- 기능

1. Member

- setGrade() : 회원의 현재금액에 맞는 회원등급을 설정
- charge_money(add_money) : add_money만큼 회원의 현재금액에 충전
- spend_money(money) : money만큼 회원의 현재금액에서 사용
- print_change_money_listt(cnt) : 회원의 금액 변동 내역에 대해 cnt 수만큼 출력

2. RBTree

- insert_member(idx, name, phone, grade, money) : 회원번호 idx, 이름 name, 연락처 phone, 회원등급 grade, 현재금액 money 값을 가진 회원을 트리에 삽입
- insert_new_member(idx, name, phone) : 회원번호 idx, 이름 name, 연락처 phone을 가진 회원을 트리에 삽입
- search(idx) : 트리 내에서 회원번호가 idx인 노드를 찾아, 해당 노드의 깊이와 Member instance를 반환
- print_member(depth, member) : member에 대한 회원정보와 깊이 depth를 출력

3. Ground

- buy_ground(buyer, seller, x, y, price) : buyer, seller 회원 간 x, y 좌표의 땅에 대하여 price 가격에 거래를 진행

4. Priority_queue (변형)

- pair<int, int> arr[SIZE] : value, key 값을 저장
- top() : arr에서 가장 value 값이 작은 pair를 반환
- top_key() : arr에서 가장 value 값이 작은 key를 반환
- top_value() : arr에서 가장 value 값이 작은 value를 반환
- getLimit() : arr에서 push 될 index 이전의 index에 대한 pair 반환
- push(value, key) : value, key 값을 arr에 삽입
- update(value, key) : arr에서 key에 대한 value를 업데이트
- print() : arr의 key, value를 차례대로 출력
- sort() : arr를 내림차순으로 정렬
- isFull() : arr가 가득 찼을 경우 1 반환
- isEmpty() : arr가 비었을 경우 1 반환

5. Rich

- Priority_queue arr : 현재금액이 가장 높은 상위 5명에 대한 money, idx 값을 저장
- print_rich() : 현재금액이 가장 높은 상위 5명에 대한 정보를 출력(리스트가 없을 시 "Not found!"를 출력)

- new_rich(money, id) : arr에 속한 money와 회원번호를 money,id와 비교하여, 매개변수 money가 더 크거나, money는 같고 id가 더 작은 수 일때 arr에 삽입

- increse_rich(money, id) : 회원번호 id의 money가 arr에 속한 가장 작은 금액과 비교하여 더 크면 삽입

- decrese_rich(money, id) : 현재금액이 가장 높은 상위 5명 중 현재금액이 감소된 경우, 트리를 순회하여 이 회원보다 현재금액이 높은 회원을 찾아 교체

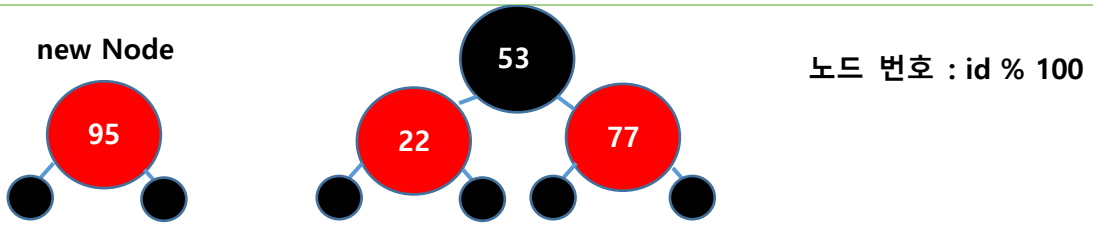
3. 기능별 알고리즘 명세

* 입력 size 는 N 이라 칭한다.

1. Red-Black Tree

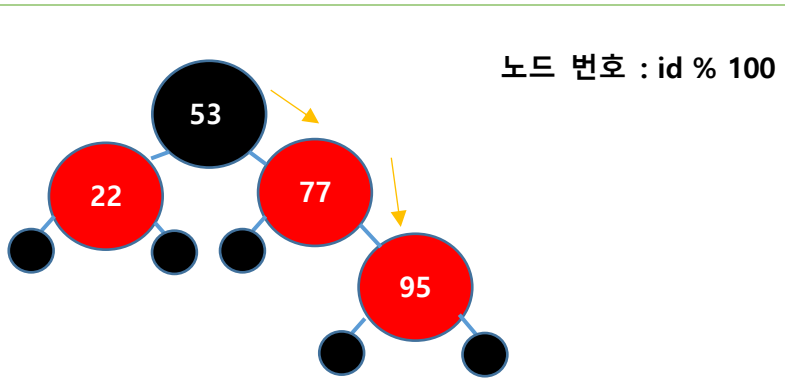
- insert_member

1. member, node 를 동적 할당하여 생성한다. 입력 받은 회원정보를 member 에 넣고 생성한 node 에 넣는다.



2. 이 node 가 red-black tree 내에서 위치할 자리를 찾기 위해 append_node 함수를 이용한다.

append_node(Node* node)

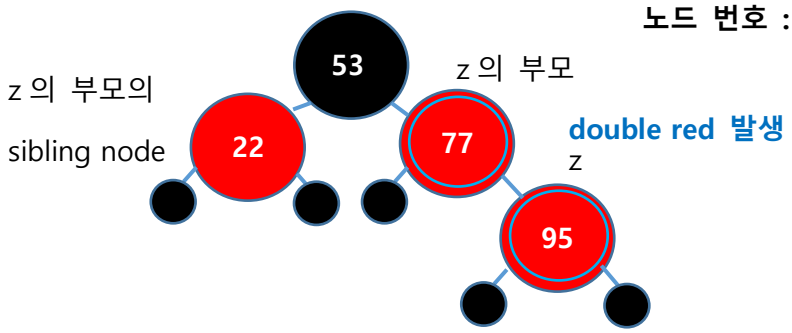


binary search tree의 조건을 만족하며 child로 내려간다. 이는 node가 LEAF(null)가 될 때까지 반복한다. LEAF node가 되었을 때, 삽입할 node를 부모의 자식으로 가리키고, 삽입할 node 또한 부모를 parent로 가리킨다. 삽입한 노드가 존재한 depth를 반환하고, 이미 회원번호 값이 존재할 경우에는 depth를 음수로 반환한다. 트리의 LEAF node까지 이진 탐색하므로, 최악의 경우의 시간 복잡도는 O(logN)이다.

3. node 가 tree 내에 성공적으로 삽입되었을 때, Red-Black Tree property 를 만족하는지 확인해야한다. 삽입한 node 를 z 라 할 때, z 의 부모가 red 인지 doubleRed 함수를 통해 확인한다. z 의 부모가 red 일 경우, doubleRed 가 발생했으므로, z 의 부모의 sibling node 의 color 에 따른 작업을 수행해야한다.

doubleRed()

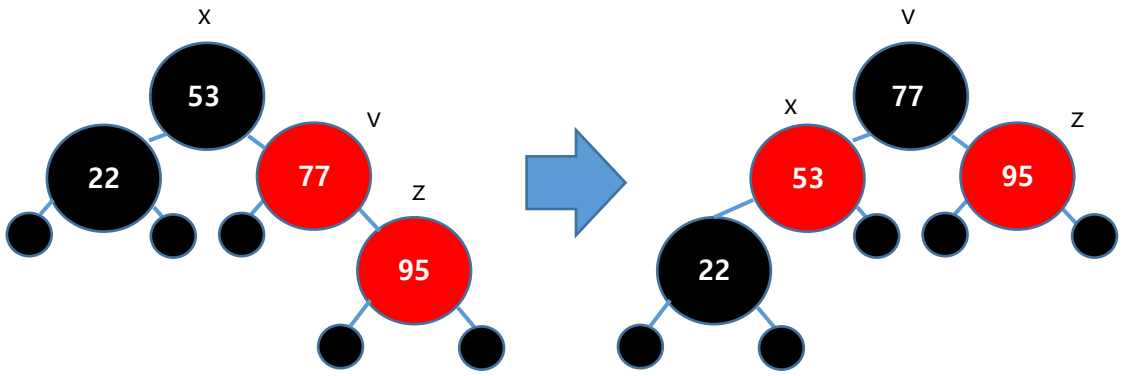
노드 번호 : id % 100



- z의 부모의 sibling node가 black일 경우 : restructure(Node* z) 함수를 실행
- z의 부모의 sibling node가 red일 경우 : recolor(Node* z) 함수를 실행, recolor가 끝나면 z의 부모를 z로 설정하여 doubleRed 확인 반복

restructure(Node* z)

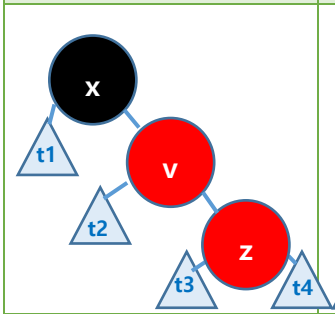
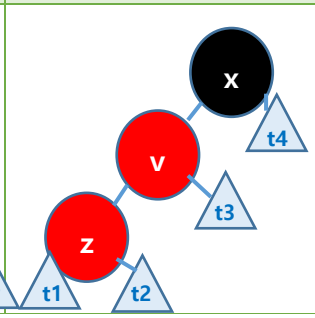
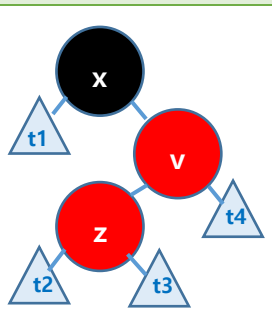
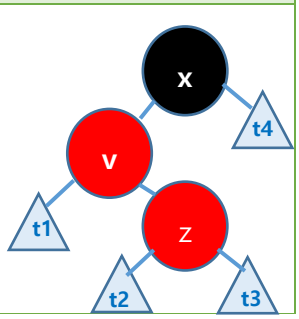
노드 번호 : id % 100



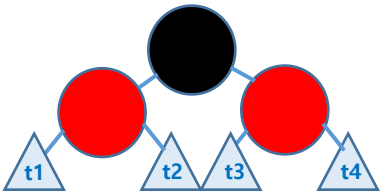
*** restructuring 이 발생하는 경우**

삽입한 node z의 부모를 v, v의 부모를 x라 한다. x, v, z 간에 위치를 재조정 해줘야 한다. x, v, z의 left, right 관계에 따라 조정되는 위치가 다르다.

아래는 각 관계에 따라 어떻게 재조정하고 subtree를 붙여주는지 나타낸 그림이다.

x, v, z 위치에 따른 4 가지 종류			
x->right == v v->right == z	x->left == v v->left == z	x->right == v v->left == z	x->left == v v->right == z
			

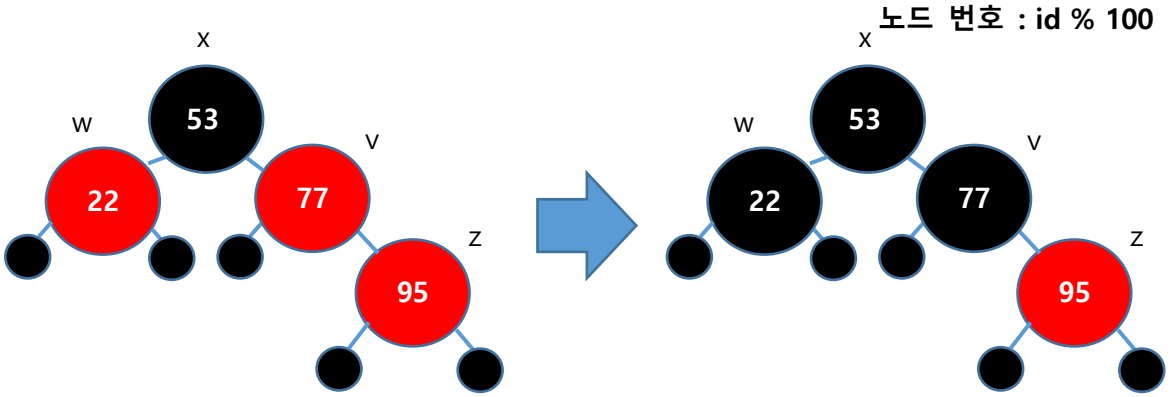
*** 재조정 이후 tree 의 모습**



subtree t1 ~ t4를 왼쪽부터 차례대로 자식-부모 관계를 설정하여 붙여주면 이진 탐색 트리 구조를 만족한다. 그리고 부모가 되는 노드를 Black, 자식은 Red로 설정한다. 이로써 재조정을 통해 Red-Black Tree property를 만족시킬 수 있게 된다. restructure 함수의 최악의 경우의 시간 및 공간 복잡도는 O(1)이다.

recolor(Node* z)

노드 번호 : id % 100



z의 parent를 v, v의 sibling을 w라 하고, v의 부모를 x라 한다. 이 때 v, w의 color를 Red에서 Black으로 바꾼다. x는 root가 아닐 경우 color를 Red로 바꾼다. 이에 대한 시간 및 공간 복잡도는 O(1)이다.

4. 반환값으로 node 가 존재하는 depth 를 반환한다. restructure 함수로 인해 depth 가 변형될 수 있으므로, getDepth 함수를 통해 depth 를 재계산하여 반환한다. getDepth 함수는 node 부터 root 까지 올라가므로 최악의 경우 시간 복잡도는 O(logN)이다.

따라서 insert_member 의 시간복잡도는 O(logN), 공간복잡도는 O(1)이다.

- search(int idx) : 트리 내에서 회원번호 idx 에 대한 depth 와 Member 클래스의 포인터 반환

root 가 NULL 일 경우 비어있는 pair 를 return 한다. 이진 탐색을 하여 idx 와 같은 값을 가진 node->member->idx 를 찾는다. 만약 LEAF(NULL)일 경우 비어있는 pair 를 반환한다. 최악의 경우 트리를 이진 탐색으로 LEAF 까지 내려간다. 따라서 최악의 경우 시간 복잡도는 O(logN)이다. 공간 복잡도는 O(1)이다.

2. Ground

- ground_idx[1000][1000] : x, y 좌표의 땅에 대한 주인을 저장한다.

- ground_price[1000][1000] : x, y 좌표의 땅에 대한 가격을 저장한다.

- buy_ground(Member* buyer, Member* seller, int x, int y, int price)

지역변수 owner_idx, owner_pirce 는 기존 주인의 회원번호와 땅값을 저장한다. 거래가 성사된 후에는 변경된 주인의 회원번호와 땅값을 저장한다. 그리고 buy 가 1 일 경우 구매, 0 일 경우 구매 실패를 나타낸다.

1. buyer 의 idx 가 owner_idx 일 경우 이미 자신이 주인이므로 거래는 실패한다.

2. buyer 의 idx 가 owner_idx 와 다를 경우 이미 주인이 있으므로,

- 돈이 충분하면 구매한다. 이 때 각 Member 의 spend, charge_money 함수를 통해 현재금액을 변경한다. ground_idx[x][y], ground_price[x][y]를 buyer 로 설정한다.

- buyer 의 현재금액이 owner_pirce 보다 적거나 매개변수 price 보다 적은 경우 구매에 실패한다.

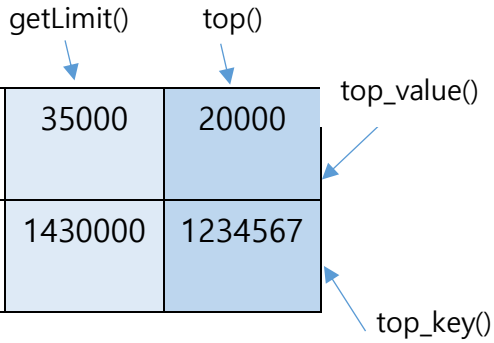
3. 주인은 없지만 price 보다 buyer 의 현재금액이 부족한 경우, 구매에 실패한다.

4. 주인이 없고 돈이 충분한 경우 buyer 는 땅을 구매한다. ground_idx[x][y], ground_price[x][y]를 buyer 로 설정한다.

위 조건에 맞게 실행한 뒤, 구매성공여부 및 buyer 의 money, owner_idx 를 출력하고, 구매성공여부를 반환한다. 이 함수의 최악의 경우 시간 및 공간 복잡도는 O(1)이다.

3. Prioirty_queue

pair<int,int> arr[SIZE] (SIZE = 5)



value (money)	50000	50000	40000	35000	20000
key (id)	1200000	1900000	2200000	1430000	1234567

- push(value, key) : arr가 가득 차 있지 않을 경우 value, key를 삽입하고 sort()
- update(value, key) : arr에서 key에 대한 value를 찾아 업데이트하고 sort() 후 1 반환
  arr에서 key를 찾지 못하면 0을 반환
- sort() : arr 배열을 내림차순으로 정렬

* 모든 시간 및 공간 복잡도 : 최대 25번의 비교가 일어나므로 O(1)

1.push

ex)

value	50000	50000	40000	35000	
key	1200000	1900000	3200000	1430000	

push(45000, 1200003)

value	50000	50000	40000	35000	45000
key	1200000	1900000	3200000	1430000	1200003

arr.top()에 값이 삽입된다.

sort()

value	50000	50000	45000	40000	35000
key	1200000	1900000	1200003	3200000	1430000

sort()를 통해 정렬된다.

push(35000, 1100001)

value	50000	50000	45000	40000	35000
key	1200000	1900000	1200003	3200000	1100001

arr 가 가득차있을 경우 push 를 한 경우에는, 값을 top 에 덮어씌우기 하고, sort()한다.

2.update

ex)

value	50000	50000	45000	40000	35000
key	1200000	1900000	1200003	3200000	1100001

update(70000, 1900000)

value	50000	70000	40000	40000	35000
key	1200000	1900000	1200003	3200000	1100001

key 값이 19000000 인 value 를 70000 으로 업데이트한다.

sort()

value	70000	50000	40000	40000	35000
key	1900000	1200000	1200003	3200000	1100001

배열을 다시 sort()를 통해 정렬한다.

4. Rich

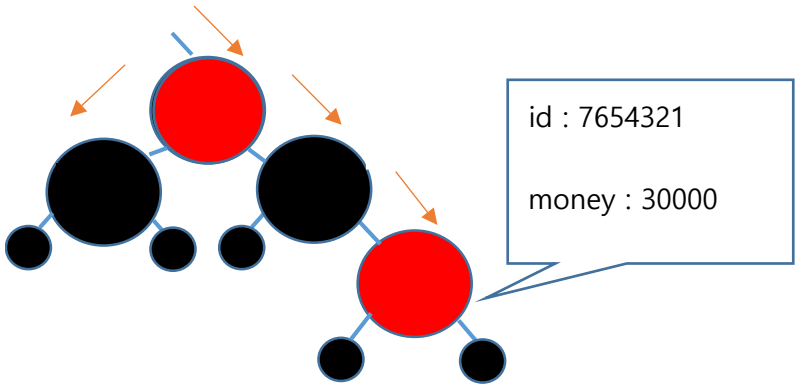
- **Priority_queue arr** : 현재금액이 가장 높은 상위 5명에 대한 money, idx 값을 저장
- **new_rich(money, id) :** Rich에 속한 money와 회원번호가 id의 money를 비교하여, id의 money가 더 클 때 arr에 push 연산을 이용하여 삽입한다. 시간 및 공간 복잡도는 O(1)이다.
- **increse_rich(money, id) :** 매개변수 money가 arr.의 최소값보다 클 때 동작한다. arr의 update 함수를 통해 id가 arr 안에 속해 있으면 update 후 sort 한다. 반면에, arr 안에 id가 속하지 않았을 때 push를 통하여 arr에 삽입 후 정렬된다. 시간 및 공간 복잡도는 O(1)이다.
- **decrese_rich(money, id) :** arr의 update 함수를 통해 id가 arr에 포함된 경우 업데이트 한다. 포함되지 않은 경우에는 함수를 종료한다.

만약 update된 arr내의 money, id의 pair가 최소값일 경우 이 보다 큰 값이 다른 member 내에 존재할 수 있다. 이를 위해 Red-Black Tree를 순회하여 arr의 최솟값 보다 큰 수가 있는지 확인해야 한다. 다음 장은 이를 수행하는 그림과 함수에 대한 묘사이다.

pair<int,int> arr[SIZE] (SIZE = 5)

getLimit() top()

value (money)	50000	50000	40000	35000	20000
key (id)	1200000	1900000	2200000	1430000	1234567



max_val : pair<int,int>(30000, 7654321)

Red-Black Tree를 순회하여 getLimit()보다 작으면서 top()보다 큰 값을 찾는다.

pair<int,int> arr[SIZE] (SIZE = 5)

value (money)	50000	50000	40000	35000	30000
key (id)	1200000	1900000	2200000	1430000	7654321

- arr.getLimit() : arr 내의 최소값(top) 앞 index의 pair를 반환한다.
- search_max(pair<int,int> limit, Node* root) : limit보다 작은 최대값을 찾는다. root부터 tree를 순회하는 travel 함수를 호출하여 찾는다.
- travel(pair<int, int> limit, pair<int, int> max_val, Node* node) : tree를 순회하며 max_val에 최대값을 갱신한다.
- setTop((pair<int,int> p) : arr top의 값을 p로 바꾼다.

decrease_rich는 tree를 순회하기 때문에 최악의 경우 시간 복잡도는 O(N)이다. 공간 복잡도 또한 travel 함수에서 tree를 순회하며 node 공간을 사용하며 재귀적으로 동작하기 때문에 O(N)의 필요하다.

- 각 기능별 최악의 경우 시간 복잡도

1. 신규 회원가입 : insert_new_member 함수에서 insert_member 함수를 호출하기 때문에 O(logN)

2. 회원정보 확인 : search 함수를 호출하기 때문에 O(logN)

3. 충전 : search 함수를 호출하기 때문에 O(logN)

4. 검색 : print_rich() 함수를 호출하기 때문에 O(1)

5. 특정 회원의 최근 금액변동내역 조회 : search 함수를 호출하기 때문에 O(logN)

6. 땅 구매 : 특정 회원의 땅 구매 처리

- buyer 와 seller Member 를 찾기 위해 search 함수를 호출하여 O(logN)

- decrese_rich 함수를 호출하여 tree 를 순회하기 때문에 O(N)

  -> 따라서 총 O(N)의 시간 복잡도가 걸린다.

4. 인터페이스 및 사용법

cmd창에서 실행파일이 존재한 경로로 이동하여 .exe 파일을 실행한다.

- 파일로부터 데이터 불러오기

프로그램을 실행시키면 입력파일을 불러온다.

입력파일의 형식은 아래와 같이 한 줄로 저장되어 있고, 여러 줄을 삽입할 수 있다.

회원번호	회원이름	연락처	x 좌표	y 좌표	회원등급	현재금액

```
C:₩Users₩김영훈₩Desktop₩YoungHoon₩4-1₩알고리즘₩code3₩Release>code3.exe
입력파일의 이름을 입력하세요 : list_50k.txt
```

입력파일의 이름을 입력시킨 뒤 각 명령어를 통해 기능을 수행 할 수 있다.

1. 신규 회원가입

질의 형식 : I (회원번호) (회원이름) (연락처) Ax Ay

출력 형식 : (해당 회원이 저장된 노드의 트리 내 깊이) (승인상태)

출력화면

```
I 1379486 홍길동 01015940904 612 56
14 1
```

2. 회원정보 확인

질의 형식 : P (회원번호)

출력 형식 : (회원이름) (연락처) (회원등급) (현재금액) (해당 회원이 저장된 노드의 트리 내 깊이)

출력화면

3. 충전

질의 형식 : A (회원번호) (충전금액)

출력 형식 : (해당 회원이 저장된 노드의 트리 내 깊이) (충전 후 해당 회원 등급)

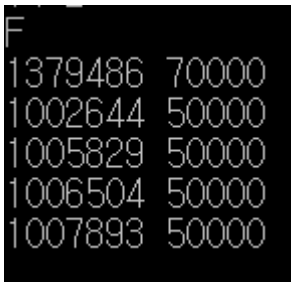
출력화면

4. 검색

질의 형식 : F

출력 형식 : (회원번호) (현재금액)

출력화면



5. 특정 회원의 최근 금액변동내역 조회

질의 형식 : R (회원번호) (조회할 최근 변동내역 수)

출력 형식 : (변동된 금액의 종류) (변동된 금액)

출력화면

6. 땅 구매 : 특정 회원의 땅 구매 처리

질의 형식 : B (회원번호) Ax Ay (거래금액)

출력 형식 : (땅 구매 여부) (구매 요청한 회원의 현재금액) (현재 땅 주인의 회원번호)

출력화면

7. 프로그램 종료

질의 형식 : Q

출력 화면

5. 평가 및 개선 방향

- 구현한 알고리즘의 장점

Red-Black Tree 자료구조를 이용하여 항상 depth가 balance하다. 따라서 입력 size가 N일 때, 삽입 및 탐색에서 최악의 경우 시간 복잡도 O(logN)을 보장한다. 삽입, 탐색이 모두 자주 일어나거나, 최악의 경우에도 O(logN)만큼의 빠른 속도를 보여야 할 때, 이 알고리즘은 유용하게 쓰일 수 있다.

또한 현재금액이 가장 높은 상위 5명을 검색하는데 있어, 최악의 경우에도 O(1)의 시간 복잡도를 가지기 때문에, 이 기능을 자주 사용할 때 효과적이다.

- 구현한 알고리즘의 단점

입력 size가 매우 작을 때, 총 point size가 input size보다 더 커 공간에 대한 overhead가 커질 수 있다.

땅 구매에서 "현재금액이 가장 높은 상위 5명"에 포함된 회원의 현재금액이 감소할 때, 상위 5명에 포함되는 회원을 다시 찾아야 한다. Red-Black Tree가 회원번호로 이진 탐색 할 수 있도록 구조화되어 있다. 따라서 현재금액을 기준으로 찾기 위해서는 트리를 모두 순회해야 하는 단점이 있다. 이는 O(N)의 시간 복잡도를 가진다. 따라서 이러한 특수한 경우가 자주 일어날 경우 오랜 시간이 걸릴 수 있는 단점이 있다.

- 향후 개선 방향

현재금액이 가장 높은 상위 5명을 찾는 것의 Optimality가 O(N)인지 생각해볼 필요가 있다. 더 좋은 알고리즘을 생각해보고 개선할 것이다.