

DS_project1 보고서

2018707010

전자통신공학과

김재혁

● Introduction

연결 리스트와 큐, 이진 탐색트리, 힙을 사용하여 계정 관리 프로그램을 구현한다.

data.txt로부터 사용자의 이름, 나이, ID를 읽어들이고 Queue를 구현하여 이 정보들을 Queue에 저장한다.

Queue에 저장된 데이터들을 Pop하여 이진탐색트리로 구성된 Account_BST와 Linked List으로 구성된 User_List에 각각 저장한다.

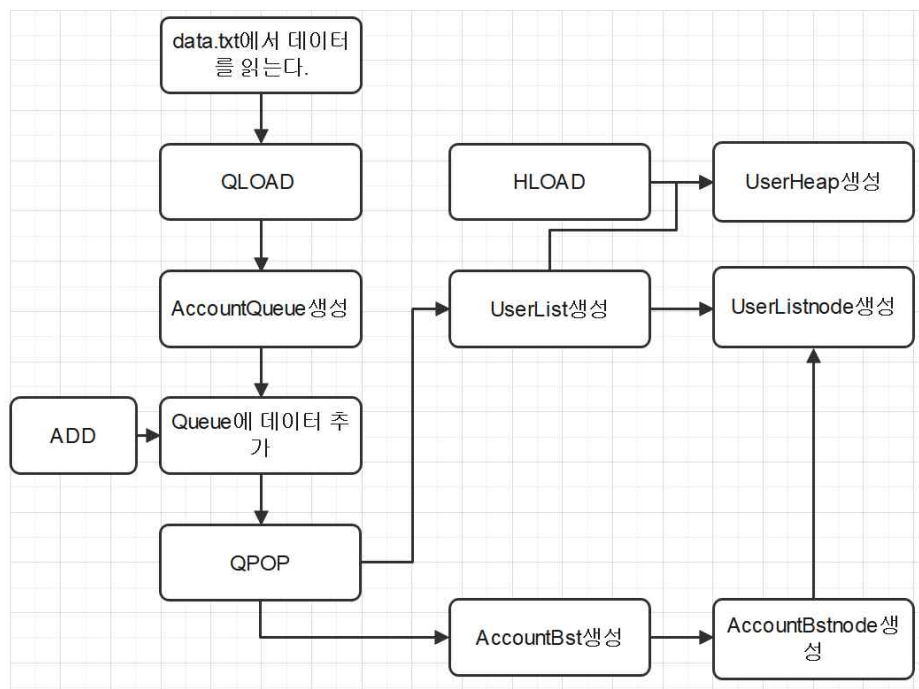
Account_BST는 완전이진탐색트리로 ID순으로 삽입한다. 사전적 순서가 작으면 왼쪽 크면 오른쪽으로 가고 숫자는 영문자보다 왼쪽으로 삽입을 실시한다.

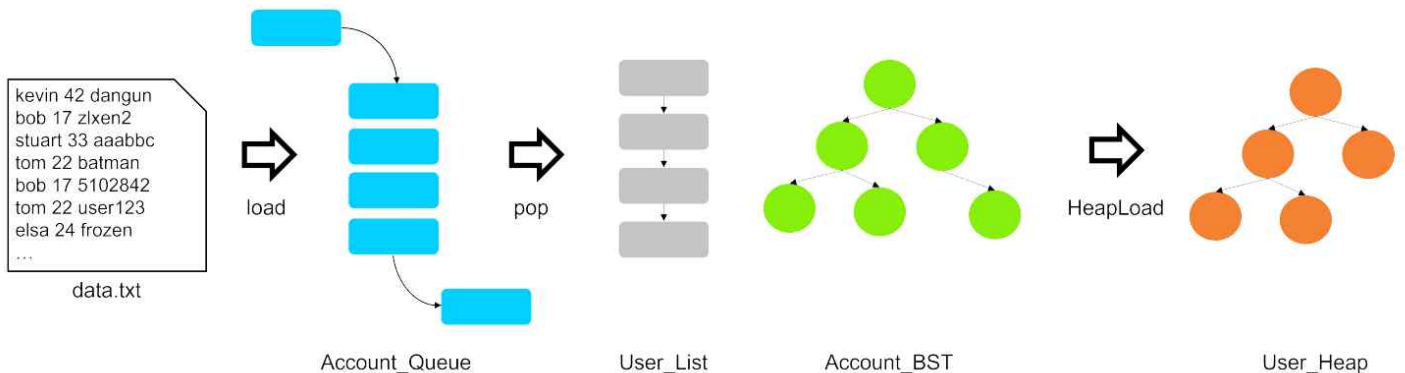
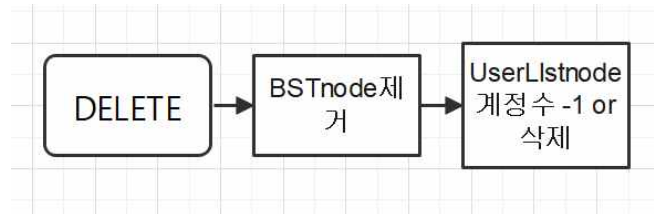
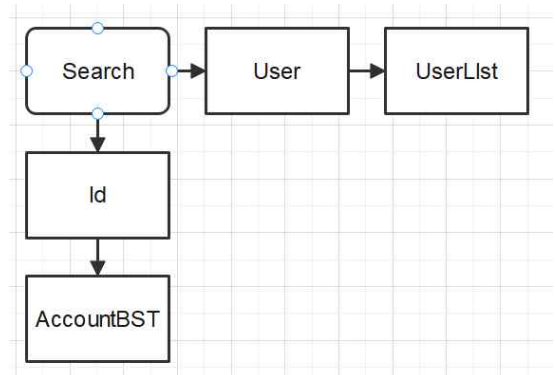
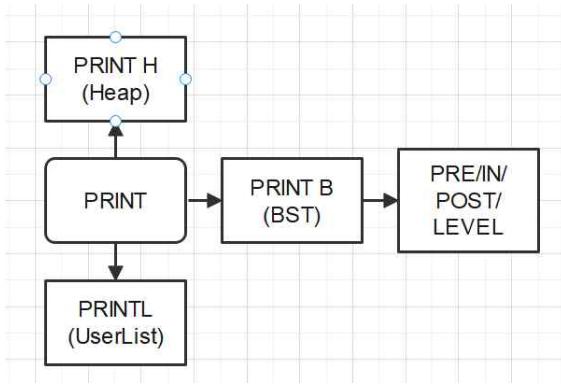
한 사람당 ID는 최대 3개를 보유 할 수 있고 같은 사용자끼리는 연결된다.

User_List는 사용자의 이름 나이 계정수의 데이터를 가지고 있으며 Account_BST에서 같은 사용자 끼리 연결된다.

HeapLoad를 통해 Heap구조를 가진 User_Heap을 생성하는데 User_List에 있는 데이터들을 가져와 연령대별 사용자 수로 노드를 나눠 자료구조를 생성한다.

● Flowchart





● Algorithm

-Queue

Queue에는 맨 앞을 가리켜줄 Front 맨 뒤를 가리켜줄 Rear 포인터변수를 사용했다.

Push함수 사용시 현재 맨 마지막노드 다음에 연결해주고 Rear를 통해 가리킨다.

Pop함수 사용시 Front를 이용해 맨 앞 노드를 반환시켜주고 그 다음 연결된 노드를 가리키게 한다.

Front와 Rear를 이용해 Queue를 관리한다.

-BST

완전이진트리로 구성되어 있으며 node는 rightnode와 leftnode를 가리킬 수 있다.

Insert시 Root가 비어있으면 Root에 Insert Root에 값이 있을 시 들어온 node의 Id값을 사전순으로 비교해 작으면 왼쪽 그렇지 않으면 오른쪽으로 탐색하고 만약 노드가 비어있다면 그곳에 노드를 배치시킨다.

string을 이용해 id값을 단순대소비교로 진행하였다.

Search또한 Id값을 사전순으로 비교해 왼쪽/오른쪽 방향을 정하고 맞는 Id가 있을때까지 진행한다. 빈 노드가 나왔다면 일치하는 값이 없다는 뜻이다.

Delete시 일단 일치하는 ID가 있는지 Search후 일치하는 값이 있으면

3가지로 경우를 구분해 진행한다.

- 1) 자식노드가 없는 경우
- 2) 자식노드가 하나 일 경우
- 3) 자식노드가 둘 다 있을 경우

1) 자식노드가 없을 경우

- 부모노드에서 연결을 끊어준다.

2) 자식노드가 하나 일 경우

- 자식노드와 부모노드를 연결시켜준다.

3) 자식노드가 둘 다 있을 경우

- 대체노드를 찾는다. 대체노드는 삭제노드의 오른쪽 자식들 중에 Id값이 제일 작은 값으로 한다.

대체노드의 부모노드와 자식노드를 연결시켜주고 삭제노드랑 연결되어있는 노드들을 대체노드에 연결 시켜준다. 기존에 삭제노드가 있던 곳을 대체노드로 바꿔주는 방식이다.

Root노드가 삭제될시 Root노드를 다시 설정 해준다.

BST에서 Print는 4가지 방식이 있다.

- 1) 전위순회
- 2) 중위순회
- 3) 후위순회
- 4) 레벨순회

1), 2), 3)의 경우 재귀적으로 함수를 만들어 순회하도록한다.

전위순회는 VLR 중위순회는 LVR 후위순회는 LRV순으로 재귀하도록 만들었다.

레벨순회는 BST의 위부터 아래순 왼쪽에서 오른쪽 순으로 방문하여 순회한다.

STL의 Queue를 사용했다. 선입선출의 개념을 이용해 레벨순회를 구현했다.

-UserList

UserList는 기본적으로 Linked List를 사용해 만든다.

Queue에서 Pop된 데이터들이 BST에 추가되면 UserList에도 추가한다.

추가시 이미 같은 사용자의 노드가 존재하면 계정수를 1증가시키고 같은 사용자가 없으면 노드를 만들어 추가한다.

UserList노드는 Bst노드와 연결되어야한다. 따라서 Inset시 Bst노드를 인자로 받고 UserList에 생성되는 노드와 연결시켜준다. 이미 사용자가 있다면 사용자의 마지막 BST노드 다음으로 연결시켜준다. 계정이 3개라면 Insert를 하지 않는다.

Search는 Root노드부터 방문하여 차례차례 값을 비교하여 찾는다.

Delete시 경우를 나눠보면

- 1) Root노드이고 현재 계정수가 1인 경우
- 2) Root노드가 아니고 현재 계정수가 1인 경우
- 3) 현재 계정수가 2이상인 경우

총 3가지 경우가 있다. 자세히 살펴보면

1) Root노드이고 현재 계정수가 1인 경우

Root노드의 자식노드로 Root노드를 변경하고 연결된 노드를 끊어준다.

2) Root노드가 아니고 현재 계정수가 1인 경우

삭제노드의 부모노드와 자식노드를 연결해준다.

3) 현재 계정수가 2이상인 경우

계정수를 1감소시킨다.

Print시에는 Root노드부터 차례대로 출력한다.

-UserHeap

HLOAD 명령시 UserList에서 값들을 받아와 Heap데이터구조를 생성한다.

Insert시 받은 데이터의 나이를 이용해 연령대를 정하고 같은 연령대가 있다면 그 노드의 NumUser를 1만큼 증가시키고 없다면 노드를 하나 만든다.

Heap구조를 따른다. STL queue를 이용하여 데이터구조를 만드는데

Heap에서는 0번의 index를 사용하지 않는다. Heap은 완전 이진트리의 구조를 따르므로 부모노드와 자식노드의 index의 관계는 부모=[자식/2] 가 성립한다.

따라서 새로운 노드가 들어오면 일단 Queue맨뒤에 배치한다. 그리고

부모=[자식/2]를 이용해 부모 자식간 NumUser를 비교하고 더 큰 노드가 위로 올라가는 방식을 구현한다. 이렇게 구현하면 MaxHeap구조가 만들어진다.

Print시에는 Queue구조에는 index순으로 되어있으므로 level순회의 방식으로 출력이 된다.

-manager

프로그램을 운영하는 handler이다. data.txt에서 읽어들인 data들과 command.txt에서 읽어들인 command들로 프로그램을 진행한다.

파일스트림을 이용해 파일을 열고 log.txt에 출력한다.

파일을 읽을때는 getline함수를 이용해 정해진 크기만큼 읽고

strtok함수를 이용해 띄어쓰기 기준으로 문자열을 분리해 각각의 데이터들과

명령들을 구분한다. 이 때 명령들의 오류들을 검출해낼 수 있다.(지정된 명령외의 명령들과 공백)

● Result Screen

예시중 데이터 파일의 명시가 없으면 데이터들은 예시에서의 data들을 사용하였다.

kevin 42 dangun

bob 17 zlxe2

stuart 33 aaabbc

tom 22 batman

bob 17 5102842

tom 22 user123

elsa 24 frozen

1. QLOAD

QLOAd(성공)

*command.txt - 메모장
파일(F) 편집(E) 포맷(O) 보기
QLOAD

log.txt - 메모장
파일(F) 편집(E) 포맷(O) 보기(V) 도움말(H)
=====QLOAD=====
kevin/42/dangun
bob/17/zlxen2
stuart/33/aaabbc
tom/22/batman
bob/17/5102842
tom/22/user123
elsa/24/frozen
=====

이미 자료구조에 데이터가 존재하는 경우(오류)

command.txt - 메모장
파일(F) 편집(E) 포맷(O)
QLOAD
QLOAD

log.txt - 메모장
파일(F) 편집(E) 포맷(O) 보기(V) 도움말(H)
=====QLOAD=====
kevin/42/dangun
bob/17/zlxen2
stuart/33/aaabbc
tom/22/batman
bob/17/5102842
tom/22/user123
elsa/24/frozen
=====

===== ERROR =====
100
=====

data.txt가 존재하지 않는 경우(오류)

*command.txt - 메모장
파일(F) 편집(E) 포맷(O) 보기
QLOAD


log.txt - 메모장
파일(F) 편집(E) 포맷(O) 보기(V) 도움말(H)
===== ERROR =====
100
=====

데이터에 인자가 부족한 경우

kevin 42 dangun
bob 17 zlxen2
stuart 33
tom batman
bob
tom 22 user123
elsa 24 frozen


log.txt - 메모장
파일(F) 편집(E) 포맷(O) 보기(V) 도움말(H)
=====QLOAD=====
kevin/42/dangun
bob/17/zlxen2
100
100
100
tom/22/user123
elsa/24/frozen
=====

2. ADD

 *command.txt - 메모장

파일(F) 편집(E) 포맷(O)

QLOAD
ADD tom 22 user222
ADD err idid
ADD er

 log.txt - 메모장

파일(F) 편집(E) 포맷(O) 보기(V) 도움말(H)

=====QLOAD=====

kevin/42/dangun
bob/17/zlxen2
stuart/33/aaabbc
tom/22/batman
bob/17/5102842
tom/22/user123
elsa/24/frozen

=====

=====ADD=====

tom/22/user222

=====

===== ERROR =====

200

=====

===== ERROR =====

200


=====

3.QPOP

 command.txt - 메모장

파일(F) 편집(E) 포맷(O)

QLOAD
QPOP 4
QPOP 5
QPOP 3

 log.txt - 메모장

파일(F) 편집(E) 포맷(O) 보기(V) 도움말(H)

=====QLOAD=====

kevin/42/dangun
bob/17/zlxen2
stuart/33/aaabbc
tom/22/batman
bob/17/5102842
tom/22/user123
elsa/24/frozen

=====

===== QPOP =====

Success

=====

===== ERROR =====

300

=====

===== QPOP =====

Success

=====

대소문자 구분여부와 Id중복시 예러

```
data.txt - 메모장
파일(F) 편집(E) 포맷(C)
stuart 33 aaabbc
tom 22 AAabbc
bob 17 aaabbc
tom 22 user123
```

```
log.txt - 메모장
파일(F) 편집(E) 포맷(O) 보기(V) 도움말(H)
=====QLOAD=====
stuart/33/aaabbc
tom/22/aaabbc
bob/17/aaabbc
tom/22/user123
=====

===== QPOP =====
301
301
Success
=====
```

4.SEARCH

```
SEARCH user tom
SEARCH id frozen
SEARCH id err
SEARCH id
SEARCH
```

```
===== SEARCH =====
User
tom/22
batman
user123
=====
```

```
===== SEARCH =====
ID
frozen/elsa
=====
```

```
===== ERROR =====
400
=====
```

```
===== ERROR =====
400
=====
```


5.PRINT

QPOP 7
HLOAD
PRINT L
PRINT H
PRINT B PRE
PRINT B IN
PRINT B POST
PRINT B LEVEL

===== PRINT =====
LIST
kevin/42/1
bob/17/2
stuart/33/1
tom/22/2
elsa/24/1
=====

===== PRINT =====
BST PRE
dangun/kevin
aaabbc/stuart
5102842/bob
batman/tom
zlxen2/bob
user123/tom
frozen/elsa
=====

===== PRINT =====
BST POST
aaabbc/stuart
5102842/bob
batman/tom
zlxen2/bob
user123/tom
frozen/elsa
dangun/kevin
=====

===== PRINT =====
Heap
2/20
1/40
1/30
1/10
=====

===== PRINT =====
BST IN
aaabbc/stuart
5102842/bob
batman/tom
dangun/kevin
zlxen2/bob
user123/tom
=====

===== PRINT =====
BST LEVEL
dangun/kevin
aaabbc/stuart
zlxen2/bob
5102842/bob
batman/tom
user123/tom
frozen/elsa

UserList와 UserHeap에 데이터가 없을시(NO QPOP,NO HLOAD)

command.txt -
파일(E) 편집(E) 3

QLOAD
PRINT L
PRINT H
PRINT B PRE
PRINT B IN
PRINT B POST
PRINT B LEVEL

===== ERROR =====
500
=====

===== ERROR =====
500
=====

===== ERROR =====
500
=====

===== ERROR =====
500
=====

===== ERROR =====
500
=====

===== ERROR =====
500
=====

6. DELETE

command.txt - 메모장
파일(E) 편집(E) 포맷(O)

QLOAD
QPOP 7
DELETE batman
DELETE aaabbc
DELETE error
DELETE

===== DELETE =====
Success
=====

===== DELETE =====
Success
=====

===== ERROR =====
600
=====

===== ERROR =====
600
=====

DLELTE 시 LIST와 BST에서 삭제과정이 잘 되는지 확인해보았다.

```
command.txt - 메모장
파일(F) 편집(E) 포맷(O)
QLOAD
QPOP 7
PRINT B PRE
PRINT L
DELETE batman
DELETE aaabbc
PRINT B PRE
PRINT L
```

```
===== PRINT =====
BST PRE
dangun/kevin
aaabbc/stuart
5102842/bob
batman/tom
zl Xen2/bob
user123/tom
frozen/elsa
=====
===== PRINT =====
LIST
kevin/42/1
bob/17/2
stuart/33/1
tom/22/2
elsa/24/1
=====
```

```
===== DELETE =====
Success
=====
===== DELETE =====
Success
=====
===== PRINT =====
BST PRE
dangun/kevin
aaabbc/stuart
batman/tom
=====
===== PRINT =====
LIST
kevin/42/1
bob/17/2
tom/22/1
elsa/24/1
=====
```

7.HLOAD

```
command.txt -
파일(F) 편집(E)
QLOAD
QPOP 7
PRINT H
HLOAD
PRINT H
```

```
===== ERROR =====
500
=====
===== HLOAD =====
Success
=====
===== PRINT =====
Heap
2/20
1/40
1/30
1/10
=====
```

8.EXIT

```
command.t
파일(F) 편집(I)
QLOAD
EXIT
```

```
===== EXIT =====
Success
=====
```

9. 잘못된 명령어

```
QLOAD
LOAD
PIRNT
```

```
===== ERROR =====
800
=====
===== ERROR =====
800
=====
```

- Consideration

C++을 처음사용해보았다. C++문법이 생소하다보니 코드를 작성 할 때 자료구조적 오류들 보다 문법적인 부분들을 많이 해메었다.

강의에서 배운 QUEUE, LINKED LIST, BINARY SEARCH TREE, HEAP 등의 개념을 이용하여 프로그램을 구현해봤는데 진행하면서 이런 구조들이 어떻게 돌아가는지 원리등을 자세히 이해할 수 있었다 특히 NEXT, RIGHT, LEFT등 다음 객체를 가리키는 포인터객체들의 편리성을 많이 깨달았다. 한 줄 씩 작성하며 실력이 많이 늘은 것 같다. 이보다 복잡한 프로그램도 구현 할 자신감이 생겼다.