2020-2

Data Structure Project #1

학번: 2019202045

이름: 박수연

담당교수: 이형근 교수님

제출일자: 2020-10-09

Data Structure Lab. Project #1 Report

A. Introduction

1. 제목

이진 탐색 트리(BST), 큐(Queue), 힙(Heap)을 이용한 질병 관리 프로그램

2. 프로그램 설명

본 프로젝트에서는 BST, Queue, Heap을 이용해 질병 관리 프로그램을 구현한다. 본 프로그램은 Patient_Queue, Location_BST, Patient_BST, Location_MaxHeap을 가진다.

질병 의심 대상 환자 데이터로부터 이름, 체온, 기침 여부, 지역을 입력 받아 이를 Patient_Queue에 구축하며, Patient_Queue로부터 pop된 환자 데이터는 지역 정보를 담고 있는 Location_BST와 연결된 Patient_BST에 삽입된다. 이 때, Patient_BSTNode에서는 환자 이름과 질병의 양음성 여부를 저장한다. BST에서 pop 명령어를 실행하면 Location_MaxHeap구조로 데이터가 pop되며, 힙의 각 노드는 지역에 해당하는 환자 수를 기준으로 정렬된다.

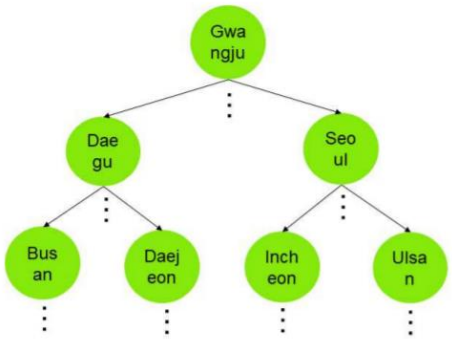
1) Patient_Queue

data.txt로부터 데이터를 불러와 환자 이름, 체온, 기침여부, 지역이 저장된 PatientNode로 이루어진 Queue를 구축한다. Queue의 구현에는 <queue> 라이브러리의 STL을 이용한다.

2) Location_BST

Location_BST는 초기에 한 번 구축되어 노드 변경이 일어나지 않는다. BST는 Gwangju, Daegu, Seoul, Busan, Daejeon, Incheon, Ulsan 순으로 추가되어 구축한다.

Patient_Queue에서 방출된 노드의 지역 정보를 확인하여 해당 LocationNode에 연결된 Patient_BST에 삽입한다.



3) Patient_BST

각 Location_BST는 Patient_BST를 갖는다. Location_BST는 Patient_BST의 root 노드의 주소 값을 저장하는 방식으로 연결된다.

Patient_Queue에서 방출된 데이터는 LocatioBSTNode의 타입으로 Patient_BST를 사전순으로 구축하며 저장되며, 각 노드는 이름과 양음성 여부를 저장한다. 체온 37.5도 이상과 기침 여부 Y가 만족할 때를 양성으로 판단하여 +기호로 표기한다.

Patient_BST에서는 Insert, Delete, Print(PRE, IN, POST, LEVEL Order)의 연산을 수행할 수 있다. 노드를 제거할 때, 양쪽 자식 노드가 모두 존재하는 경우 왼쪽 자식 노드 중 가장 큰 노드를 제거되는 노드 위치로 변경하는 원칙을 따른다.

4) Location_MaxHeap

BST에서 pop된 데이터가 삽입되는 Heap으로, 각 지역의 노드 환자 수를 count하여 환자 수 기준으로 정렬된다. BST에서 방출 연산이 일어나면 Heap을 재정렬하며, 삽입 연산과 Print 기능만 수행한다.

5) Function

LOAD: txt 파일로부터 정보를 읽어 Patient_Queue에 저장한다.

ADD: Patient_Queue에 직접 노드를 추가하는 기능을 한다. 4개의 인자를 모두 입력해야만 작동한다.

QPOP: Patient_Queue의 환자 데이터를 BST 노드로 옮기는 명령어로, 첫 번째 인자의 숫자 만큼 Queue의 front부터 데이터가 pop된다.

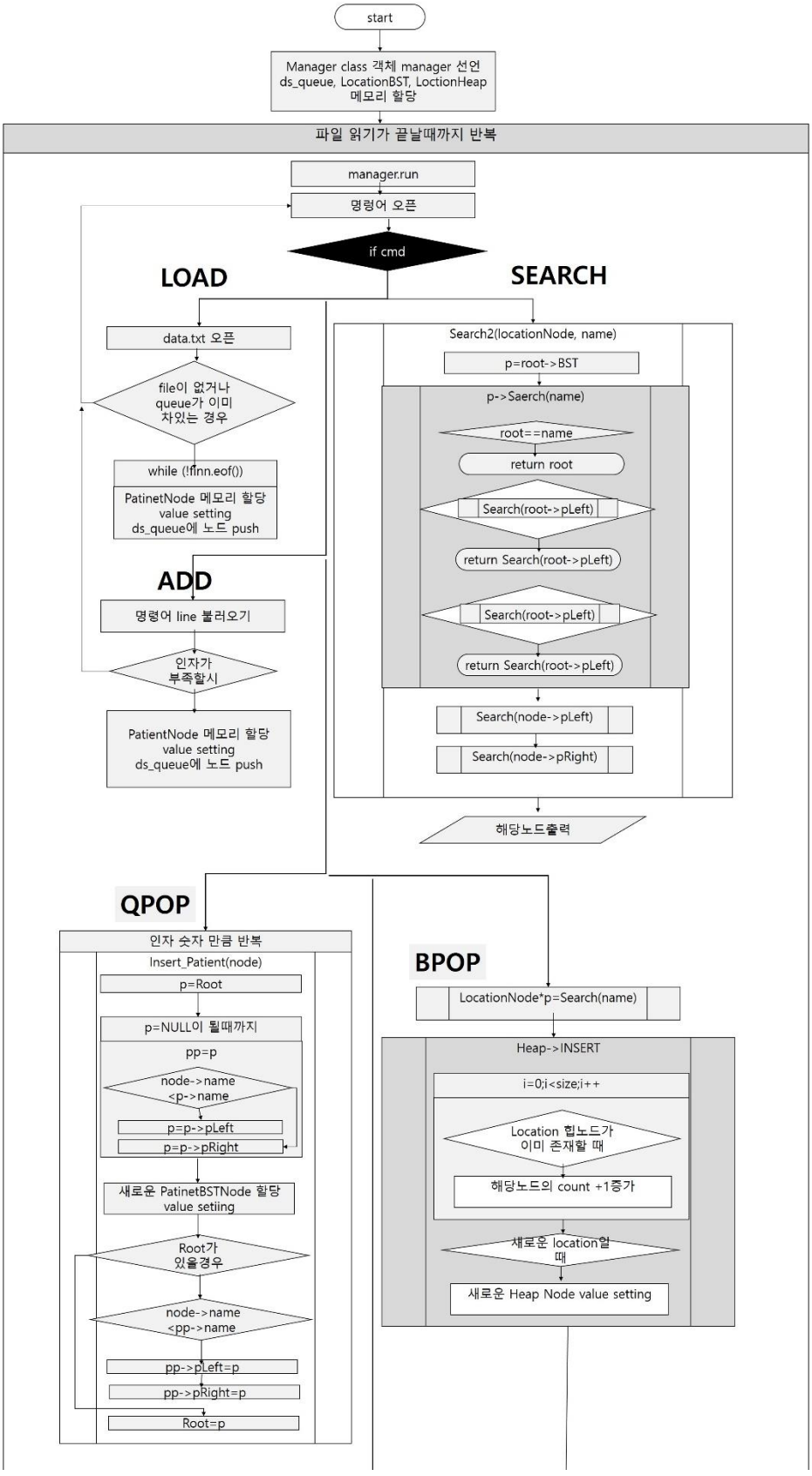
SEARCH: 환자의 이름을 인자로 입력받아 BST에 저장된 환자 정보를 찾아 이름/양성여부를 출력한다.

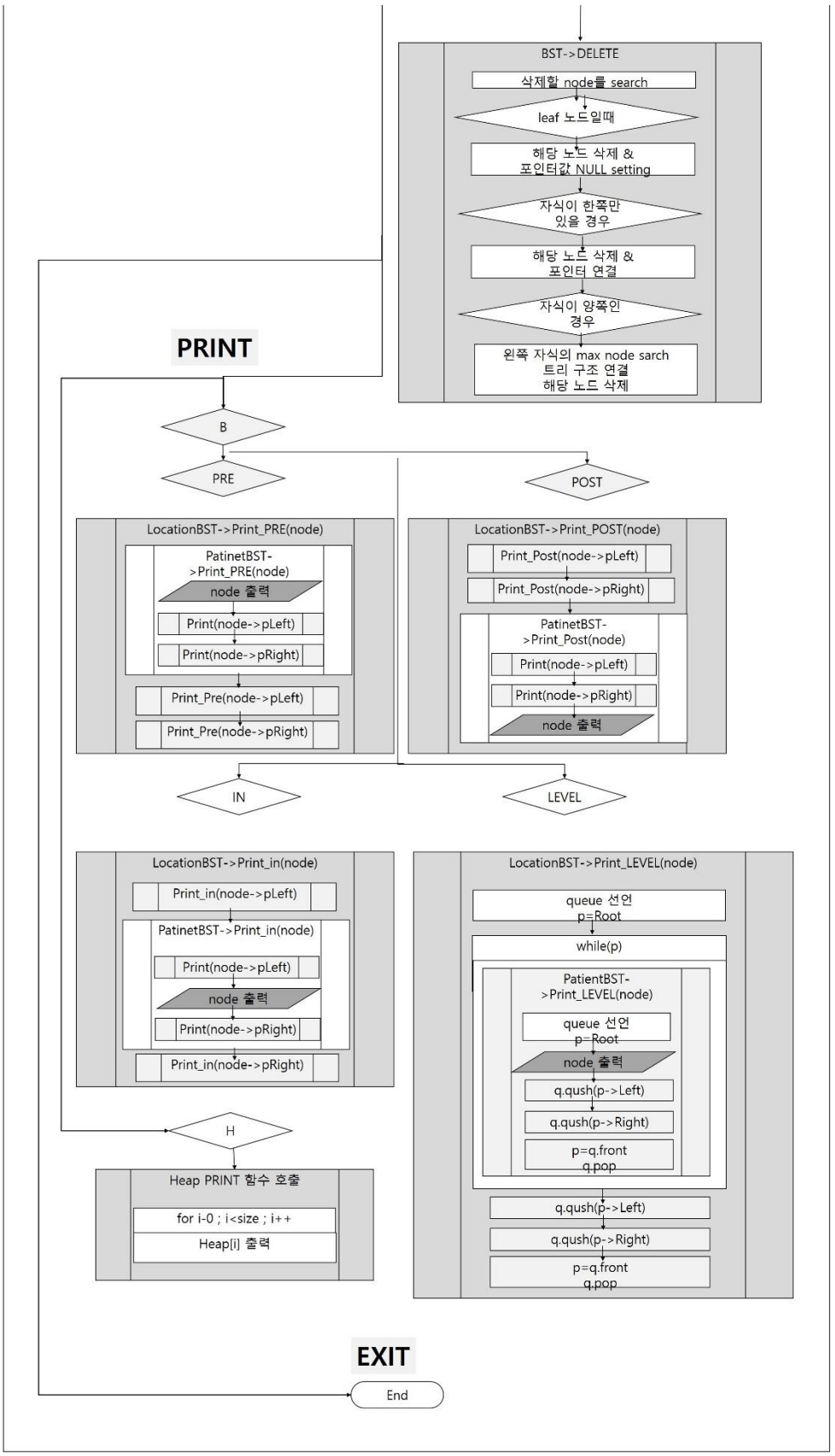
PRINT: 자료구조 속 데이터를 출력하는 명령어로, 첫 번째 인자로 B를 입력 시, Patient_BST에 있는 모든 환자 정보를 출력한다. Pre, in, post, level-order의 네가지 트리 순회 방법을 따른다. 첫 번째 인자로 H를 입력시, Location_MaxHeap에 저장된 지역 노드를 level-order로 출력한다.

BPOP: BST에 저장된 데이터를 Location_MaxHeap으로 pop하는 명령어로, BST에 저장된 노드는 삭제되며, 인자로는 환자 이름을 입력받는다.

EXIT: 프로그램 상의 메모리를 해제, 프로그램을 종료한다.

B. Flow chart


start
Manager class 객체 manager 선언
ds_queue, LocationBST, LoctionHeap
메모리 할당
파일 읽기가 끝날때까지 반복
manager.run
명령어 오픈
if cmd
LOAD
SEARCH
data.txt 오픈
file이 없거나
queue가 이미
차있는 경우
while (!finn.eof())
PatinetNode 메모리 할당
value setting
ds_queue에 노드 push
ADD
명령어 line 불러오기
인자가
부족할시
PatientNode 메모리 할당
value setting
ds_queue에 노드 push
Search2(locationNode, name)
p=root->BST
p->Saerch(name)
root==name
return root
Search(root->pLeft)
return Search(root->pLeft)
Search(root->pLeft)
return Search(root->pLeft)
Search(node->pLeft)
Search(node->pRight)
해당노드출력
QPOP
인자 숫자 만큼 반복
Insert_Patient(node)
p=Root
p=NULL이 될때까지
pp=p
node->name
<p->name
p=p->pLeft
p=p->pRight
새로운 PatinetBSTNode 할당
value setiing
Root가
있을경우
node->name
<pp->name
pp->pLeft=p
pp->pRight=p
Root=p
BPOP
LocationNode*p=Search(name)
Heap->INSERT
i=0;i<size;i++
Location 힙노드가
이미 존재할 때
해당노드의 count +1증가
새로운 location일
때
새로운 Heap Node value setting


BST->DELETE
삭제할 node를 search
leaf 노드일때
해당 노드 삭제 &
포인터값 NULL setting
자식이 한쪽만
있을 경우
해당 노드 삭제 &
포인터 연결
자식이 양쪽인
경우
왼쪽 자식의 max node sarch
트리 구조 연결
해당 노드 삭제
PRINT
B
PRE
POST
LocationBST->Print_PRE(node)
PatinetBST-
>Print_PRE(node)
node 출력
Print(node->pLeft)
Print(node->pRight)
Print_Pre(node->pLeft)
Print_Pre(node->pRight)
LocationBST->Print_POST(node)
Print_Post(node->pLeft)
Print_Post(node->pRight)
PatinetBST-
>Print_Post(node)
Print(node->pLeft)
Print(node->pRight)
node 출력
IN
LEVEL
LocationBST->Print_in(node)
Print_in(node->pLeft)
PatinetBST->Print_in(node)
Print(node->pLeft)
node 출력
Print(node->pRight)
Print_in(node->pRight)
LocationBST->Print_LEVEL(node)
queue 선언
p=Root
while(p)
PatientBST-
>Print_LEVEL(node)
queue 선언
p=Root
node 출력
q.qush(p->Left)
q.qush(p->Right)
p=q.front
q.pop
q.qush(p->Left)
q.qush(p->Right)
p=q.front
q.pop
H
Heap PRINT 함수 호출
for i-0 ; i<size ; i++
Heap[i] 출력
EXIT
End

C. 알고리즘

Manager

command.txt로부터 cmd를 불러온다. 파일 오픈 실패 시 에러 문구 출력한다.
fin에 cmd를 저장해, 값을 LOAD ADD QPOP PRINT SEARCH BPOP EXIT의 문자열과 비교해 strcmp==0이 될 때 해당 함수로 진입하며, 각 함수는 return값이 true일 때 성공 문구를 그렇지 않을 때 에러 문구를 출력한다. 이 과정을 파일이 끝날 때까지 반복 수행한다.

LOAD

data.txt로부터 데이터를 불러오는데 파일 오픈 실패 시, 혹은 ds_queue가 이미 차있을 때 false를 반환한다.
data.txt 파일의 끝까지 데이터를 불러들어오는 과정을 반복하여, 새로운 PatientNode의 할당 및 값 설정을 완료한다. 각 노드는 생성 후 ds_queue에 push한다.

ADD

각 인자를 위한 변수를 선언하고, line 단위로 데이터를 읽어온다. 인자 데이터를 차례대로 변수에 저장하는데, 이 때 하나의 변수라도 존재하지 않을 시 false를 반환한다. 새로운 PatinetNode의 할당 및 값 설정 후 ds_queue에 push한다.

QPOP

queue에서 pop할 노드의 개수를 입력받고, 이 때 n이 ds_queue의 size보다 클 시 false를 반환한다. while 문을 통하여 n개의 노드를 front로부터 차례대로 BST에 insert, queue에서 pop한다.

- **LocationBST::Insert_Patient => PatientBST::Insert**

  전달 받은 노드의 location의 값을 확인해 해당 LocationNode의 BST에 노드를 삽입힌다. PatientBST에서는, P_Root부터 시작해 해당 노드의 이름과 사전순의 비교를 반복함으로 노드가 삽입될 위치를 정한다. 만약 Root가 존재하지 않을 시에는 해당 노드가 루트가 된다.

PRINT

첫째 인자가 B일경우 BST를 출력한다. 이 경우 다음 인자 값을 확인해 각각 Print_PRE, Print_IN, Print_POST, Print_LEVEL 함수를 호출한다.

- **Print_PRE**

  가장 먼저 루트에 접근 후, 해당 함수 연산을 left subtree, Rights ubtree 순으로 재귀적으로 반복 진행한다. 이는 LocationBST와 PatientBST 모두 동일하며 PRINT 명령어에서는 PatientBSTNode의 값을 출력해야 하므로 LocationBST를 순회하면서 각노드에서 PatientBST의 PRINT함수를 호출하는 방식으로 접근했다.

- **Print_IN / Print_POST**

  각각 left subtree, 루트, right subtree순으로 접근 / left subtree, right subtree, root 순으로 접근한다.

- **Print LEVEL**

  LEVEL order의 출력에는 queue STL을 이용했다. root부터 시작하여 root 출력, left subtree가 있을시 왼쪽 자식을 push, right subtree가 있을시 오른쪽 자식을 push 후 front에 접근한 후 pop 연산을 수행하는 과정을 큐가 빌 때까지 반복한다.

- **LocationHeap::Print**

  인자가 H일 경우 Heap을 출력한다. HeapNode는 배열 내 LEVEL 순서대로 저장되어 있기 때문에 배열요소를 반복문으로 출력한다.

SEARCH

인자값을 name 문자열에 저장해 해당 name을 갖는 PatientBSTNode를 Search2 함수를 통해 반환받는다. 만약 노드가 없을 시 false를 반환하고, 노드가 있을 시 그 이름과 양성여부를 출력한다.

- **PatientBST::Search**

  환자 이름으로부터 PatientBSTNode를 찾아 반환한다. 전달 받은 이름 값과 PatientBSTNode의 name 변수를 비교하여 그 값이 같을 경우 해당 노드를 리턴, 그렇지 않을 경우 left, right subtree를 인자로 해당 Search 함수를 재귀적으로 반복한다.

• **LocationBST::Search2**

환자 이름으로부터 PatientBSTNode를 반환하는 함수로, PRINT 함수와 같은 알고리즘으로 각 노드에 접근하여 PatientBST의 Search 함수 연산을 반복하여 그 반환 값이 있을 경우 이를 반환한다.

BPOP

인자 값을 name 문자열에 저장하여 해당 name의 환자가 있는 LocationNode를 찾아 반환받는다. LocationNode의 반환 값이 없는 경우 false를 반환한다.
해당 노드의 Location을 ds_heap에 insert, 해당 환자 이름의 BSTNode를 BST에서 Delete하는 함수를 수행한다.

• **LocationHeap::Insert**

전달 받은 location의 LocationHeapNode가 이미 존재하는지 확인하는 과정을 갖는다. 만약 이미 해당 Location이 존재한다면, 해당 노드의 count를 1 증가시키고, heap 내부를 재정렬한다. 재정렬을 위해서는 child node(n)와 parent node((n-1)/2)의 값을 비교해 만약 child가 parent 보다 클 시 노드를 swap한다.

해당 Location이 존재하지 않았다면, 마지막 배열요소에 새로운 location 노드를 할당하고 count를 1로 지정한다.

• **PatientBST::Delete**

환자 이름의 BSTNode를 삭제하는 연산을 수행한다. 루트에서 시작하여 전달받은 name 값과 노드의 name 값을 사전순으로 비교하는 과정을 반복하여 삭제하고자 하는 노드의 위치를 찾는다.

만약 노드가 leaf node 라면, 해당 node를 삭제하고 이를 가리키던 포인터 변수를 null로 초기화시킨다.

만약 한쪽의 subtree만을 갖는 노드라면, 좌우의 크기관계를 확인하여 포인터를 연결시킨 후 노드를 삭제한다.

만약 노드가 양쪽의 subtree를 갖고 있다면, 왼쪽 subtree에서 가장 큰 값을 갖고 있는 노드를 찾은 후 이를 삭제할 노드의 위치로 대체시키고 노드를 삭제한다.

D. 실행결과

1) Assignment pdf에서 주어진 예시

Data.txt

```
tom 37.2 Y Seoul
emily 36.5 N Incheon
erin 38.1 Y Daegu
elsa 38.4 Y Seoul
mia 36.4 Y Ulsan
```

Command.txt

```
LOAD
LOAD
QPOP 4
QPOP 5
PRINT B PRE
BPOP tom
SEARCH erin
SEARCH mia
BPOP erin
BPOP elsa
PRINT H
EXIT
```

Log.txt

```
========== LOAD ==========
Success
==========================

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

========== QPOP ==========
Success
==========================

========== ERROR ==========
300
==========================

========== PRINT ==========
BST
erin/+
tom/-
elsa/+
emily/-
==========================


========== BPOP ==========
Success
==========================

========== SEARCH ==========
erin/+
==========================

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

========== BPOP ==========
Success
==========================

========== BPOP ==========
Success
==========================

========== PRINT ==========
Heap
2/Seoul
1/Daegu
==========================


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

예시와 같은 결과를 내고 있음을 확인.

2)

Data.txt

tom 37.2 Y Seoul
emily 36.5 N Incheon
erin 38.1 Y Daegu
elsa 38.4 Y Seoul
mia 36.4 Y Ulsan
sooy 38.9 Y Seoul
pew 35.5 N Incheon
mike 36.6 Y Daegu
jin 37.7 Y Seoul

Command.txt

LOAD
QPOP 10
ADD soo 45.4
ADD happy 35.5 Y Incheon
QPOP 10
PRINT B IN
PRINT B POST
BPOP tom
BPOP happy
SEARCH tom
BPOP sss
BPOP mike
BPOP jin
PRINT H
BPOP pew
BPOP elsa
BPOP erin
BPOP mia
PRINT H
EXIT

Log.txt

```
========== LOAD ==========
Success
==========================

========== ERROR ==========
300
==========================

========== ERROR ==========
200
==========================

========== ADD ==========
happy/35.5/Y/Incheon
==========================
```

[QPOP 10]: 입력 인자가 큐의 노드 수보다 많아 에러코드 300이 출력

[ADD so 45.4]: 입력 인자가 부족하여 에러코드 200이 출력

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

========== PRINT ==========
BST
erin/+
mike/-
emily/-
happy/-
pew/-
elsa/+
jin/+
sooy/+
tom/-
mia/-
==========================


========== PRINT ==========
BST
mike/-
erin/+
happy/-
pew/-
emily/-
mia/-
jin/+
sooy/+
elsa/+
tom/-
==========================
```

ADD로 추가된 sooy까지 포함하여 여러가지 출력이 원활히 작동

Data.txt

```
tom 37.2 Y Seoul
emily 36.5 N Incheon
erin 38.1 Y Daegu
elsa 38.4 Y Seoul
mia 36.4 Y Ulsan
sooy 38.9 Y Seoul
pew 35.5 N Incheon
mike 36.6 Y Daegu
jin 37.7 Y Seoul
```

Command.txt

```
LOAD
QPOP 10
ADD soo 45.4
ADD happy 35.5 Y Incheon
QPOP 10
PRINT B IN
PRINT B POST
BPOP tom
BPOP happy
SEARCH tom
BPOP sss
BPOP mike
BPOP jin
PRINT H
BPOP pew
BPOP elsa
BPOP erin
BPOP mia
PRINT H
EXIT
```

Log.txt

```
========== BPOP ==========
Success
==========================

========== BPOP ==========
Success
==========================

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

========== ERROR ==========
600
==========================
```

[SEARCH tom]: tom은 이미 BPOP되었기 때문에 BST에 존재하지 않아, 에러코드 500 출력

[BPOP sss]: 입력인자가 존재하지 않는 환자 이름이므로 에러코드 600 출력

```
========== BPOP ==========
Success
==========================

========== BPOP ==========
Success
==========================

========== PRINT ==========
Heap
2/Seoul
1/Incheon
1/Daegu
==========================


========== BPOP ==========
Success
==========================

========== BPOP ==========
Success
==========================

========== BPOP ==========
Success
==========================

========== BPOP ==========
Success
==========================

========== PRINT ==========
Heap
3/Seoul
2/Incheon
2/Daegu
1/Ulsan
==========================


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

Heap 연산이 올바르게 이뤄지고 있음을 확인

E. 고찰

자료를 저장하는 다양한 자료구조를 활용할 수 있는 프로젝트였다. 자료 구조를 다뤄본 경험이 별로 없어 코드가 생소하고 코드를 설계하는 과정이 쉽지 않았지만, 이번 과제를 수행하면서 Queue와 BST, Heap에 대한 이해가 대폭 증가한 것 같다. 1학기 때 Linked list를 구현하는 과제를 수행한 경험이 있어, 노드를 연결하는 과정은 비교적 수월했지만, 주로 메모리 할당과 접근에서 많은 오류가 발생하였다. 각각의 구조체를 어떻게 동적으로 메모리를 할당하고 초기화 시킬 것인가에 대한 개념을 확실히 해놔야 겠다. 메모리가 제대로 할당되지 않아 nullptr을 반환할 수 없음, 혹은 읽기 권한이 없음 등의 오류와 여러 번 마주했는데, 해당 문제들을 해결하면서 메모리와 자료 구조 포인터의 연결에 대해 익힐 수 있는 유익한 과정이었다.

또한 해당 과제에서는 Ubuntu를 활용하여 코드를 작성 및 컴파일하는 것이 요구되었는데, 대부분의 경우 호환에 문제가 없었지만 일부 코드가 원활히 작동하지 않는 경우가 있었다. 특히 BPOP명령어의 인자가 없는 경우 예외처리가 가능하도록 코드를 구현하고자 getline으로 줄 단위 문자열을 입력받아 strtok으로 공백 단위로 나눠 그 값이 NULL이면 인자 입력에 오류가 있다고 인식하는 코드를 작성하였는데, 해당 코드가 윈도우 환경의 비쥬얼 스튜디오에서는 요구대로 작동하였으나 우분투 환경에서는 옳지 않은 결과를 냄을 확인할 수 있었다. 이와 같은 호환성 문제를 염두하여 프로젝트를 진행해야 겠다고 생각했다.