데이터 구조

1차 프로젝트

학　　과: 컴퓨터정보공학부

담당교수: 이기훈교수님

학　　번: 2017202087

성　　명: 홍 세 정

1. Introduction

이진 탐색 트리(Binary Search Tree), 큐(Queue:STL), 힙(Heap) 이용하여 질병관리 프로그램을 구현한다. 이름, 체온, 기침 여부, 지역이 저장되어있는 data.txt를 읽어와 Patient_Queue를 구축하고 queue는 first in first out으로 먼저 들어온 data가 먼저 나가는 구조이다. Patient_Queue에서 pop하여 Location_BST를 구축하고, Location_BST node마다 Patient_BST 연결되어 있고, BST에서 pop하여 Location_MaxHeap을 구축한다.

또한 각 명령어에 대해 다음과 같은 수행과 출력을 할 수 있다.

- LOAD: 텍스트 파일을 불러 데이터를 한줄씩 읽어와 Patient_Queue를 구축한다.
- ADD: Patient_Queue에 직접 데이터를 추가한다.
- QPOP: Patient_Queue의 데이터를 BST에 추가하는 명령어이다. 입력받은 수 보다 데이터가 적으면 에러코드를 출력한다.
- SEARCH: 이름을 입력받아 BST에 환자가 존재하면 양성/음성의 정보를 출력한다.
- PRINT: BST에 저장된 데이터들을 출력한다. preorder, inorder, postorder, leverlorder로 출력가능하고 B를 입력받으면 location_BST 출력, H를 입력받으면 Heap_BST를 출력하게 된다.
- BPOP: 입력받은 이름의 정보를 삭제하고 Heap_BST에 지역별 환자수를 업데이트하는 명령어이다.
- EXIT: 프로그램을 종료하는 명령어이다.

-

다음과 같은 클래스로 구현되어 있고 각 클래스에는 다음과 같은 함수로 구현되어 있다.

- main: main 함수
  - manager를 실행하는 함수이다. Command.txt파일을 읽어온다.
- manager: 수행해야할 명령어를 입력 받으면 명령어에 맞게 프로그램을 조정하는 역할이다.
  - Void run(const char* command) : 명령어에 맞는 함수로 이동시켜주는 함수이다.

- Bool LOAD(queue<PatientNode*>* Patient_Queue): LOAD 명령어 수행
- Bool ADD(char* name, float temperature, char* cough, char* location, queue<PatientNode*>* ds_queue) : ADD 명령어 수행
- bool QPOP(LocationBST* DS_BST, int size): QPOP 명령어 수행
- bool SEARCH(LocationBST* DS_BST, char* name): SEARCH 명령어 수행
- bool PRINT(LocationBST* DS_BST, char* BH, char* traverse): PRINT 명령어 수행
- bool BPOP(LocationBST* DS_BST, char* name): BPOP 명령어 수행
- void PrintErrorCode(int num): 에러 메시지 log.txt파일에 출력, 각 명령어에 대한 오류 num은 LOAD(100), ADD(200), QPOP(300), SEARCH(400), PRINT(500), BPOP(600) 이다.
- void PrintSuccess(const char* act): 명령어를 맞게 수행하면 success를 출력한다.
- void AddSuccess(const char* act, char* name, float temperature, char* cough, char* location): ADD명령어를 에러 없이 수행한다면 출력하는 함수이다.

- PatientNode: 환자 정보를 node이다. 이름, 체온, 기침여부, 지역 정보를 포함하고 있다.
  - 변수: Name, temperature, Cough, Location, pLeft, pRight
  - Get: 변수를 반환해주는 함수
  - Set: 변수를 입력 받는 함수

- PatientBST: 환자 정보 BST이다. 저장되는 데이터는PatientBSTNode class로 선언되어 있다.
  - 변수: Root
  - bool Insert(PatientBSTNode* node): 환자의 정보를 추가하는 함수
  - int Search(char* name): 환자를 검색하는 함수

- bool Delete(char* name): 환자의 정보를 삭제하는 함수
- void Print_PRE, Print_IN, Print_POST, Print_LEVEL(): 환자의 정보를 전위순회 중위순회, 후위순회, 레벨순회로 출력하는 함수

- PatientBSTNode: 환자 정보 BST node이다. 이름, 양성/음성 판단 정보를 포함하고 있다.
  - 변수: Name, disease의 정보를 가지고 있다.
  - Get: 변수를 반환해주는 함수
  - Set: 변수를 입력 받는 함수

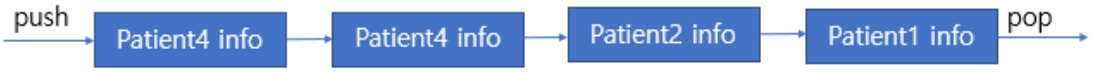
- LocationNode: 지역 Node이다. 지역정보를 포함하고 있다.
  - 변수: Location, BST, pLeft, pRight
  - Get: 변수를 반환해주는 함수
  - Set: 변수를 입력 받는 함수

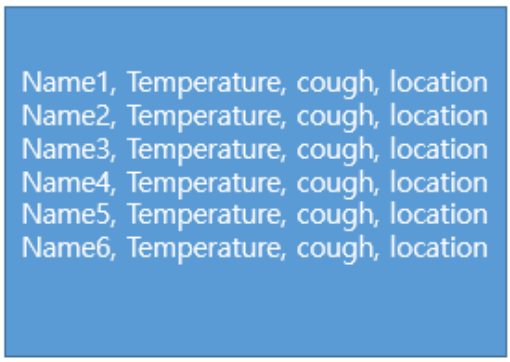
- LocationBST: 지역 BST이다.
  - void Insert_Location(LocationNode* node): 지역을 추가한다. 지역은 Gwangju, Daegu, Seoul, Busan, Daejeon, Incheon, Ulsan 순으로 추가한다.
  - bool Insert_Patient(PatientNode* node): 환자정보를 추가한다. 환자 정보 중 지역을 보고 해당 지역으로 노드를 추가한다.
  - int Search(char* name): 환자의 이름만으로 환자가 어느 지역에 있는지 판단한다. 같은 이름을 가진 환자는 없다고 가정한다.
  - char* Delete(char* name): 환자를 찾는 함수이다. 환자가 존재하지 않는다면 error이다.
  - void Print_PRE, Print_IN, Print_POST, Print_LEVER(): Location_BST를 순회하여 출력

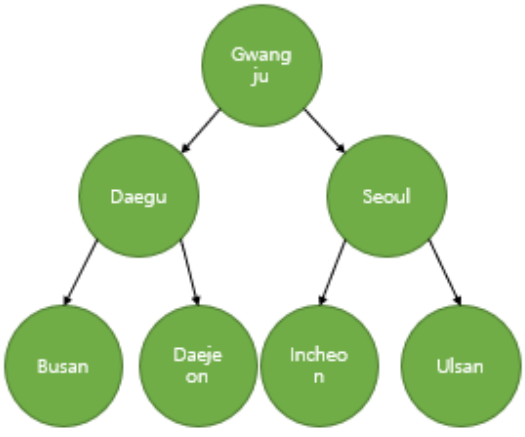
한다.

- LocationHeap: 지역 max heap이다.
  - **heap을 변수로 가지고 잇다. Heap은 배열 형태로 저장되므로 이중 포인터로 선언하였다.
  - bool Insert(char* location): 지역을 추가한다. 지역이 이미 존재한다면 환자의 수를 증가시켜준다. 지역은 총 7개 이므로 max_heap의 최대는 7이다.
  - void Print(): max heap을 출력한다. 환자 수가 많은 순서대로 출력.
- LocationHeapNode: 지역 max heap node이다. 지역, 환자 수를 포함하고 있다.
  - 변수: Count(삭제한 환자 수), location(해당지역)
  - Get: 변수를 반환해주는 함수
  - Set: 변수를 입력 받는 함수

2. Flowchart



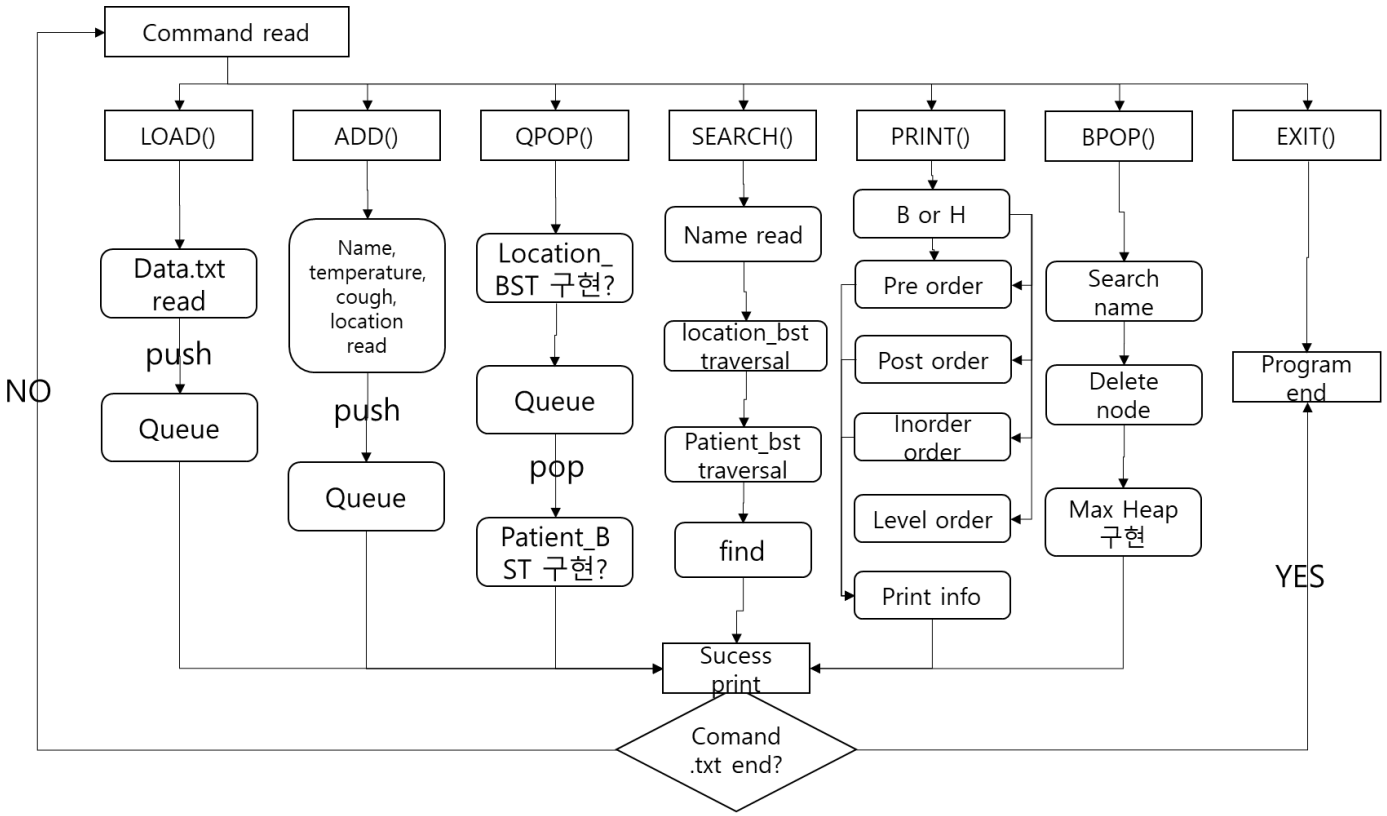




Data.txt에 name, temperature, cough, location이 저장되어 있다. PatientNode를 생성하여 queue에 저장할 수 있다.

Location Node는 추가되거나 삭제되지 않고, 순서대로 삽입되므로 다음과 같은 BST로 구

축되는 것을 확인할 수 있다.

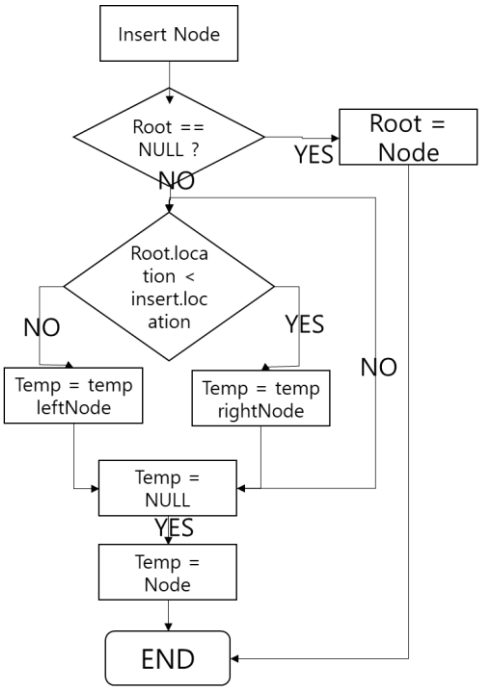


전체적인 flowchart를 나타낸 것이다. 다음은 false인 경우까지 추가하면 너무 복잡해져 true인 경우만 나타내었다. QPOP에서 queu가 없으면 false, search, print, bpop 명령어를 실행할 때 location_BST와 Patient_BST가 없다면 false가 나와 error를 출력할 것이다.

Command.txt 명령어를 한 줄씩 읽어와 알맞은 함수를 실행할 수 있다. Txt를 모두 read하거나 EXIT 명령어를 입력 받으면 프로그램을 종료한다.

3. Algorithm

- BST insert



- Root가 비어 있다면 삽입되는 노드는 루트이다.

- Temp는 삽입될 위치의 노드이다.

- temp노드의 지역과 삽입되는 노드의 지역을 알파벳순으로 비교하여 작으면 leftnode, 크면 rightNode로 이동한다. 이동하다 비어있는 node를 만나면 그곳에 삽입한다.

- PatientBST도 다음과 같은 알고리즘으로 삽입된다.

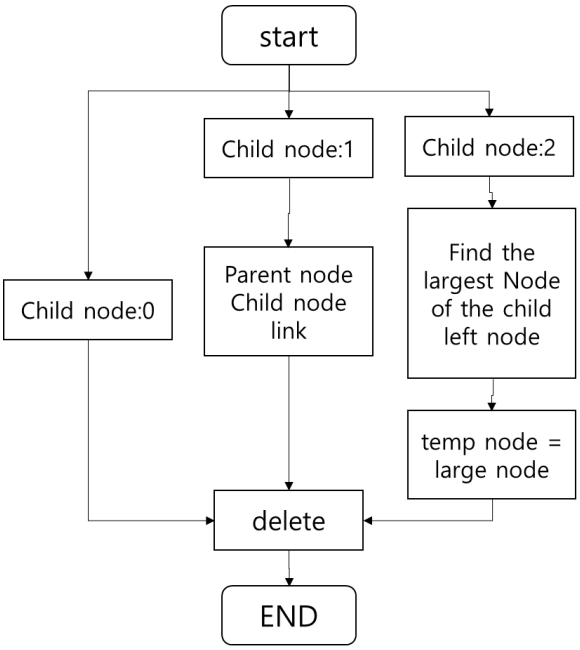
- BST delete

  - 삭제될 위치의 노드가 자식 노드가 두개, 자식 노드가 left만, 자식 노드가 right, 자식 노드가 없는 것으로 구분할 수 있다.

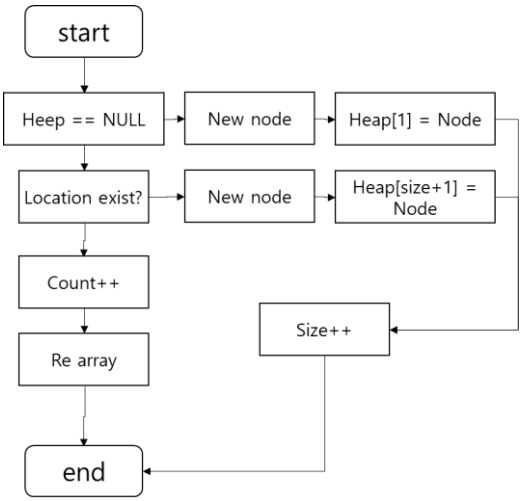
  - 자식 노드가 없으면 삭제될 노드를 삭제한다.

  - 자식 노드가 하나 있다면 자식 노드를 부모 노드와 연결한다.

  - 자식 노드가 두개라면 왼쪽 자식 노드 중에서 가장 큰 노드를 골라서 삭제될 위치 노드에 저장한다.

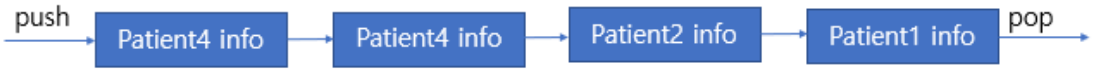


- HEAP



- Heap 삽입은 BST삽입과 다르다.
- 처음 heap이 존재하지 않는다면 heap배열을 선언해주어야 한다.
- 추가할 지역이 이미 존재한다면 count를 증가시키고 재배열한다.
- Heap[0]은 NULL이다.

- Queue

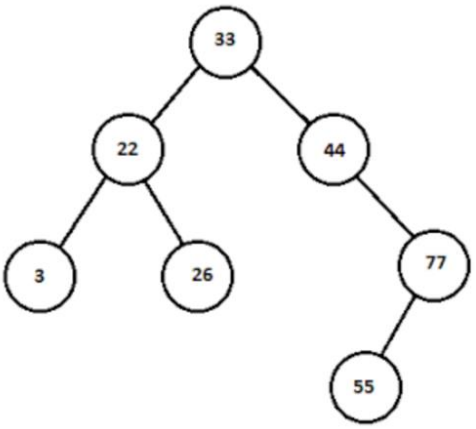


- queue는 first in first out으로 먼저 들어온 data가 먼저 나가는 구조이다.
- STL을 이용하여 쉽게 구현 가능하다.

- Preorder, inorder, postorder, level

  이진 탐색 트리(binary search tree)를 구현한다. Queue를 사용하여 preorder, inorder, postorder 순회를 출력할 수 있다.

  - BST tree



이진 탐색 트리(BST)는 다음과 같은 속성을 가지는 노드 기반이 이진 트리 데이터구조이다.

노드의 왼쪽 하위 트리에는 키가 노드 키보다 작은 노드만 포함되고, 노드의 오른쪽 하위 트리에는 키가 노드 키보다 큰 노드만 포함된다.

이렇게 왼쪽, 오른쪽 하위 트리는 모두 이진 검색 트리이어야 한다.

출처 - https://joosjuliet.github.io/check_if_a_binary_tree_is_BST_or_not/

여기서 제일 처음 들어온 값이 33, root라고 할 수 있고 루트 밑은 자식 노드라고 불린다. 들어온 순서대로 다음과 같이 트리를 만들 수 있다.

이진 탐색 트리를 순회할 때는 전위 순회(preorder), 중위 순회(inorder), 후위 순회(postorder) 순회를 할 수 있다. 목적에 맞게 순회를 사용할 수 있다.

전위 순회는 루트를 먼저 방문하여 출력한 후 왼쪽 서브 트리를 방문하고 오른쪽 서브 트리를 마지막으로 방문하여 출력하게 된다. 위의 예시를 보면 전위 순회는 33 – 22 – 3 – 26 – 44 -77 – 55 순으로 방문하게 된다.

중위 순회는 왼쪽 서브 트리, 루트, 오른쪽 서브 트리 순으로 방문하여 출력하게 된다. 따라서 위의 예시는 3 – 22 -26 – 33 – 44 – 77 -55으로 방문하게 된다.

후위 순회는 왼쪽 서브 트리, 오른쪽 서브 트리, 루트 순으로 방문하여 출력한다. 예시로는 3 – 26 – 22 – 55 – 77 – 44 – 33 순으로 방문하게 된다.

순회는 공통적으로 트리의 모든 노드를 방문하게 된다. 다음과 같은 순회 구별은 노드를 방문하는 순서에 의해 결정되게 된다. 따라서 이번 과제를 구현할 때도 방문하여 출력하는 순서만 바꿔준다면 알맞게 구현이 가능하다.

4. Result screen

command.txt

```
LOAD
LOAD
```

LOAD 명령어를 두 번 입력하였다.

log.txt

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

========== ERROR ==========
100
============================
```

처음 명령어는 성공하였지만 두 번째 LOAD 명령어는 이미 데이터가 들어가 있으므로 error가 출력된다.

Command.txt

```
ADD risa 38.1 Y Incheon
```

ADD명령어로 환자의 정보를 queue에 추가하였다.

Log.txt

```
========== ADD ==========
risa/38.1/Y/Incheon
============================
```

queue에 데이터가 추가되었으므로 출력되었다.

Command.txt

```
QPOP 4
QPOP 5
```

Queue에 있는 환자 정보를 BST로 옮기는 명령어이다.

Log.txt

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

========== ERROR ==========
300
============================
```

Data에 저장되어 있는 환자 정보는 5명이다. QPOP 4은 데이터가 충분하므로 성공하였다. QPOP 5는 데이터를 옮기지 못하므로 error가 출력된다.

command.txt

```
PRINT B PRE
PRINT B IN
PRINT B POST
PRINT B LEVEL
```

다음과 같이 입력을 주었다.

log.txt

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

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

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

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

preorder

inorder

postorder

level

각 순회에 맞게 출력되는 것을 확인할 수 있다.

Command.txt

```
SEARCH erin
SEARCH mia
```

Log.txt

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

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

mia 이름을 가진 데이터는 없으므로 error가 출력된다.

Command.txt

```
BPOP tom
BPOP erin
BPOP elsa
```

BPOP 명령어로 삭제를 수행하고 Heap을 추가하였다.

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

Command.txt

```
PRINT H
```

heap을 출력하면

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

다음과 같이 크기 순서대로 출력되는 것을 확인할 수 있다.

```
EXIT
```

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

5. Consideration

이번 데이터구조의 1차 프로젝트는 질병관리 프로그램을 구현하는 것이었다. 데이터 구조에 첫번째 프로젝트라서 어렵지 않을 것이라고 생각했는데 많은 어려움이 있었다.

처음 구현할 때 queue, ds_bst, ds_heap 모두 동적할당을 해주지 않아서 주소 값을 가지지 않았다. 무조건 동적할당을 해주어야 한다.

Delete node를 구현할 때 생각할 부분이 많았었다. 수업시간에 배운 알고리즘을 이용하여 구현가능 하였다. 자식 노드가 없거나 한 개라면 간단하게 구현가능 하였는데 자식 노드가 두개일 때가 문제였다. 삭제할 노드의 왼쪽 자식 노드 중에서 가장 큰 노드를 선택하여 삭제될 위치에 삽입하면 되었다. 그 후 큰 노드의 자식 노드의 연결을 어느 노드와 연결할지 직접 그려보면서 어떤 경우가 있는지 판단하여 삭제하고 알맞은 곳에 연결해줄 수 있었다.

Heap에 대한 이해가 부족 했었다. 처음에는 location BST와 같이 연결하면 될 것이라고 생각했지만 Left Node와 Right Node가 없어서 변수를 생성하여 연결해주었다. 하지만 Heap과 location BST는 다른 구조를 가졌다. 다시 Heap의 구조를 이해하고 Max-Heap을 구현할 수 있었다. **이중 포인터를 가진 node라서 당황했지만 일반 배열과 같이 생각해주니 쉽게 구현할 수 있었다.