

데이터 구조설계 및 실습

실험제목: Data Structure Lab. Project #1

제출일자: 2020 년 10 월 9 일 (금)

학 과: 컴퓨터정보공학부

담당교수: 이기훈 교수님

학 번: 2019202052

성 명: 김 호 성

1. Introduction

이진 탐색 트리(Binary Search Tree, BST)와 큐(Queue), 힙(heap)을 이용하여 질병 관리 프로그램을 구현한다.

1. Patient_Queue

- data.txt 에 저장된 데이터를 이용하여 구축한다.
- \n 단위로 정보를 입력 받고, spacebar 를 이용해 각각 이름, 체온, 기침 여부, 거주지 등을 구분한다.
- STL 라이브러리를 사용하여 구현한다.
- Location_BST 를 구축한다.
- 우선순위 없이 들어온 순서대로 구축한다.

예외처리

- 중복은 없다고 가정한다.

2. Location_BST

- 초기에 한 번 구축되며, 그 후로 노드 추가 혹은 삭제가 일어나지 않는다.
- 고정적인 순서를 가지고 있다. 중위 순회라고 가정하면, Busan, Daegu, Daejeon, Gwangju, Incheon, Seoul, Ulsan 순으로 출력되어야 한다.
- 멤버 변수로는 지역, Patient_BST 포인터를 가진다. 이 때 Patient_BST 포인터의 경우 이후 Patient_BST 를 만들기 위해 정보들이 어디에 저장되어 있는지 나타내는 역할을 한다.

예외처리

- 지역이름은 첫 글자만 대문자이다. (그렇지 않은 경우의 입력은 없다고 가정한다.)

3. Patient_BST

- Location 값이 같은 것들을 모아(거주지가 같음), 만들어진 BST 다.
- Patient_BST 의 root 노드의 주소 값만 저장하고 있으며, 초기 값은 null 이다.
- Patient_Queue 에서 방출된 데이터(체온, 기침여부, 거주지)를 삽입하여, 음성 양성 여부를 판단한다. (if(Temp>=37.5 && string(Cough) == "Y")
- 최종적으로 Patient_BST 에 저장되는 데이터는 이름, 양성/음성 정보이다.
- Patient_BST 는 연산, 삽입, 삭제, 탐색(중위, 후위, 전위, 레벨)을 지원한다.
- 멤버 변수로 root 정보를 가지고 있고, 멤버 함수로 삽입, 삭제, 탐색 등 요구되는 연산을 위한 함수를 가지고 있다.
- BST 연결 규칙
 1. 부모 노드보다 환자 이름의 사전적 순서가 작은 노드는 왼쪽, 큰 노드는 오른쪽 서브 트리에 반드시 위치한다.
 2. 노드를 제거할 때 양쪽 자식 노드가 모두 존재할 경우에는 왼쪽 자식 노드 중 가장 큰 노드를 제거되는 노드 위치로 이동시킨다.

4. Location_MaxHeap

- HEAP 은 환자 수를 기준으로 정렬된다.
- BST 에서 방출된 데이터를 삽입해, max 를 결정하는 기준은 지역에 속한 환자의 수이다. 다시 말해, BST 에서 가장 많이 방출된 지역이 root 에 오게 된다.
- BST 에서 방출 연산이 일어날 때마다 Heap 을 재정렬한다.
- Heap 을 재정렬할 때, 비교하는 노드의 환자 수가 추가된 노드보다 크거나 같은 경우에 정렬을 완료한다.
- Location_MaxHeap 에서는 삽입 연산만 이루어진다.
- Location_MaxHeap 에 저장되는 데이터는 LocationHeapNode class 로 선언되어 있으며 멤버 변수로는 지역에 속한 환자 수, 지역 이름을 가지고 있다.

예외처리

- BST 와 연산이 상호작용 되어야 한다.

Manager.cpp Introduction

run

Command.txt 에서 함수를 읽어온 후 매치된 명령을 실행한다. 그 명령의 반환 값이 참이면 성공을, 거짓이면 실패를 log 에 저장한다.

LOAD

- data.txt 에서 정보를 불러온다.
- Patient_Queue 에 읽은 텍스트 파일을 한 줄 단위로 저장하며, spacebar 에 따라 정보(이름, 체온, 기침, 거주지역)를 구분한다.
- 만약 텍스트 파일이 존재하지 않거나 자료구조에 이미 데이터가 들어가 있으면 알맞은 에러 코드를 출력한다. (error 100)
- Location_BST 에 데이터를 삽입한다.

ADD

- command.txt 에서 [ADD (정보들)] 구문을 감지했다면, spacebar 를 기준으로 정보를 구분하여, Patient_Queue 에 삽입한다.
- 여러 데이터 조건에 부합하는 경우에만 return 1, 그렇지 않으면 삽입하지 않고, return 0 이라고 한다.

QPOP

- Patient_Queue 의 환자 데이터를 BST 로 옮긴다.
- Queue 의 front 부터 첫 번째 인자의 숫자만큼 데이터가 이동되며 Queue 에 저장된 데이터보다 높은 숫자가 입력될 경우 에러코드를 출력하며, 명령을 실행하지 않는다.

SEARCH

- BST 에 저장된 정보를 찾아 출력하는 명령어로, 첫 번째 인자로 반드시 환자 이름을 입력 받는다. (Patient_BST 에 저장되어야 함.)
- 인자를 입력하지 않은 경우, 해당 이름이 BST 에 존재하지 않는 경우 모두 에러 코드를 출력한다.

PRINT

- 자료구조에 저장된 데이터들을 출력하는 명령어로, 첫 번째로 인자로 B 를 입력할 경우 Patient_BST 에 있는 환자 정보들을 전부 출력한다.
- 탐색 방법은 BST 의 경우 총 4 가지가 있으며, 각각 PRE, IN, POST, LEVEL 로 구분된다. (만약 command 에서 어떤 걸로 해라 안되어 있으면, 어떤 걸로 해도 무관하다.)
- H 의 경우 무조건 level-order 순서대로 출력하며, 출력할 때는 지역에 속한 환자 수/ 지역 이름의 형태로 출력한다.
- 만약 데이터가 존재하지 않을 경우 error 코드를 출력한다.

BPOP

- BST 에 저장된 환자 정보를 지우고, Location_MaxHeap 의 지역별 환자수를 업데이트한다.
- Heap 에 저장된 데이터의 부모 자식간 대소관계가 변경되었다면, Heap 을 재정렬한다.
- 첫 번째 인자는 반드시 이름 정보이며, 인자가 입력되지 않은 경우 에러 코드를 출력하고, 해당 이름 정보가 없을 시 에러 코드를 출력한다.

EXIT

프로그램 상의 메모리를 해제하며, 프로그램을 종료한다.

LocationNode.cpp introduction

GetLoc

거주지의 값을 반환한다.

GetBST

BST 포인터의 값을 반환한다.

GetLeft

왼쪽 노드의 값을 반환한다.

GetRight

오른쪽 노드의 값을 반환한다.

SetLoc

Location 값을 location 에 복사한다.

SetLeft

왼쪽 노드를 지정해준다.

SetRight

오른쪽 노드를 지정해준다.

LocationBST.cpp introduction

GetRoot

Root 를 반환한다.

Insert_Location

LocationBST 를 만든다.

Insert_Patient

PatientBST 를 만든다.

SEARCH

해당 이름을 가진 data 가 있는지 검색한다.

DELETE

해당 이름을 가진 data 를 삭제한다.

Print_PRE

PRE 방법으로 tree 를 순회하여 출력한다.

Print_IN

Inorder 방법으로 tree 를 순회하여 출력한다.

Print_POST

Postorder 방법으로 tree 를 순회하여 출력한다.

Print_LEVEL

LEVEL 방법으로 tree 를 순회하여 출력한다.

PatientBSTNode introduction

GetName

이름을 반환한다.

GetDisease

Disease 를 반환한다.

GetLeft

왼쪽 노드를 반환한다.

GetRight

오른쪽 노드를 반환한다.

SetName

이름 값을 복사한다. (저장이 가능하다.)

SetDisease

열이 37.5 이상이고, 기침을 하면 +를 이외의 아니면 - 값을 저장한다.

SetLeft

왼쪽 노드를 지정해준다.

SetRight

오른쪽 노드를 지정해준다.

PatientBST introduction

GetRoot

Root 를 반환한다.

Insert

PatientBST 를 삽입한다.

Search

PatientBST 를 탐색하여 특정 값을 찾는다.(name)
QPOP 에서 사용한다.

Delete

patentBST 의 노드 하나를 삭제한다.
BPOP 에서 사용한다.

Print_PRE

Preorder 방식으로 순회하여 출력한다.

Print_IN

Inoreder 방식으로 순회하여 출력한다.

Print_POST

Postorder 방식으로 순회하여 출력한다.

Print_LEVEL

LEVEL 방식으로 순회하여 출력한다.

LocationHeapNode introduction

GetCount

Node 의 개수가 몇 개인지 반환한다.

GetLoc

location 값을 반환한다.

SetCount

Node 의 개수가 몇 개인지 계산한다.

SetLoc

Location 값을 복사하여 location 에 넣어준다.

LocationHeap introduction

INSERT

Location 에 맞게 입력한다.

Print

Levelorder 를 기준으로 순회하여 출력한다.

2. Flowchart(의사코드)

main

1. manager.run("command.txt")

run (manager.cpp 내에서)

0. 생성자를 통해 Location_BST 생성

1. log.txt 파일이 열렸는지 확인

2. 열렸다면 command 를 한 줄 단위로 입력 받음

3. 입력 받은 command 를 공백으로 다시 구분해줌.

4. 명령이 존재한다면 명령에 따른 함수 실행(Load, ADD..)

4-1. 명령에 인자 값으로 토큰화하고 남은 command 도 같이 넣어줌

(후에 남은 명령들을 구분하면서 명령실행 ex) PRINT B PRE

LOAD

0. data.txt 를 불러옴 만약 실패하면 return 0(false)

1. data.txt 에서 한 줄 단위로 data 를 가져옴.

2. 가져온 data 를 공백에 따라 전부 구분함. (token)

3. 각각에 나눠진 정보를 queue 에 맞게 형 변환해준 후 하나의 노드로 넣어줌.

ADD

0. command 뒤에 남은 data 를 불러옴.

1. 만약 data 가 모자라면 return 0;

2. 토큰화한 정보를 queue 에 맞게 형변환해준 후 하나의 노드로 삽입.

QPOP

0. command 에서 들어온 값을 int 로 변환.

1. 명령한 값보다 queue 의 사이즈가 작다면 return 0
2. 아니라면 patient node 에 queue 의 가장 앞부분의 노드에 넣어준 후 qpop 으로 queue 노드 하나 삭제
3. 기침여부와 체온을 기준으로 양성, 음성을 정한 후 patientBSTnode 에 삽입(단, location 에 맞게)
4. 명령한 값만큼 반복

SEARCH

1. location 순서대로 BST 를 탐색(location 하위 노드인 patient_bst 에선 Preorder 순으로 탐색)

PRINT

0. 들어온 PRINT 이후의 값이 null 이면 return 0
1. 아니라면 토큰화 한 후 B 나 H 가 아니면 return 0
2. 만약 B 라면 다음 토큰이 PRE, IN, POST, LEVEL 이면 각각의 순회 방법에 맞게 출력
 - 2-1. 만약 null 이면 Preorder 로 출력
 - 2-2. 그 이외의 경우 return 0;
3. H 의 경우 Levelorder 로 출력

BPOP

구현하지 못함.

EXIT

프로그램 종료

3. Algorithm

Queue

기본적으로 First In, First Out 의 구조를 저장하는 형식이다.

Queue 의 연산

- Push(element): 큐에 원소를 추가한다. (뒤에)
- Pop(): 큐에 있는 원소를 삭제한다. (앞에)
- Front(): 큐 제일 앞에 있는 원소를 반환
- Back(): 큐 제일 뒤에 있는 원소를 반환
- Empty(): 큐가 비어있으면 true 아니면 false 를 반환한다.
- Size(): 큐 사이즈를 반환한다.

Push, Pop 은 복잡도: $O(\log n)$ $n = \text{size}$

나머지는 $O(1)$

Binary Search Tree

하나 이상의 노드로 이루어진 유한한 집합이며, 자식 노드로는 최대 두 개까지만 가지고 있는 tree 이다.

Node i 의 부모 node: $i/2$ 의 몫, 만약 몫이 0 이면 Root 다.

Node i 의 leftchild: $2i > n$, 만약 n 보다 크면 leftchild 가 없다.

Node i 의 rightchild: $2i+1 > n$, 만약 n 보다 크면 rightchild 가 없다.

Node 의 최대 값: $(2^{\text{높이}} - 1)$

Node 의 최소 값: 높이

순회 방법

Preorder

- Visit(t)
- PreOrder(t -> leftChild)
- PreOrder(t -> rightChild)

Inorder

- InOrder(t -> leftChild);

- Visit(t);
- InOrder(t -> rightChild);

Postorder

- PostOrder(t -> leftChild);
- PostOrder(t -> rightChild);
- Visit(t);

Levelorder

```
While(currentNode){  
- Visit(currentNode);  
- If(currentNode -> leftChild) q.push(currentNode -> leftChild);  
- If(currentNode -> rightChild) q.push(currentNode -> rightChild);  
- If(q.isEmpty()) return ;  
- currentNode = q.Front();  
- q.pop();  
}
```

MaxHeap

우선 최대 트리(Max Tree)는 각 노드의 키(key)값이 (자식 노드가 있다면) 그 자식의 키(Key)값 보다 크지 않은 트리이다.

최대 힙(Max Heap)은 최대 트리(Max Tree)이면서 완전 이진 트리(Complete Binary Tree)이다.

Heap 의 기본연산

- Complete Binary Tree 이므로 array 연산을 해도 괜찮다.
- 부모 노드 구하기: $i/2$ 의 몫
- Left child 구하기 $2*i > n$ (n = 노드의 수이며 $2i$ 가 n 보다 크면 자식이 없다.
- Right child 구하기 $2*i + 1 > n$ (n = 노드의 수이며 $2i + 1$ 이 n 보다 크면 자식이 없다.)
- Heap 의 높이: $\log_2 n + 1$ 의 올림

MaxHeap 에서의 삽입

최대 힙은 완전이진트리를 따르기 때문에 여기서 하나의 원소를 추가시킨다면 올바르지 않은 노드에 삽입되게 된다. 거기서 루트 쪽으로 올라가는 방법을 사용하며, 재 삽입된 위치도 올바르지 않다면 다시 루트 쪽으로 올라가는 방법을 반복한다.

결론적으로 부모 노드보다 값이 클 경우 서로 교환하면서 위로 올라가는 과정을 반복한다.

복잡도: $O(\log n)$ n 은 힙의 크기

MaxHeap 에서의 삭제

1. Root 를 delete 한다.
2. 저장되어 있는 가장 마지막 원소를 root 자리에 둔다.
3. 새로운 루트하고 자식 노드를 비교해서 순서가 맞도록 변경한다.
4. 새로 가져온 요소하고 큰 자식 노드를 바꾼다

삭제의 복잡도 $O(\log n)$

4. Result Screen

1 회 실행했을 시의 log

===== LOAD =====

Success

=====

===== ERROR =====

100

=====

===== ADD =====

Success

=====

===== QPOP =====

Success

=====

===== ERROR =====

300

=====

===== PRINT =====

Success

=====

===== ERROR =====

600

=====

===== ERROR =====

400

=====

===== ERROR =====

400

=====

===== ERROR =====

600

=====

===== ERROR =====

600

=====

===== ERROR =====

500

=====

===== EXIT =====

Success

=====

Command.txt

파일(F) 편집(E) 서식(O) 보기(V) 도움말

LOAD

LOAD

ADD hoseong 36.5 N Seoul

QPOP 11

QPOP 1234567

PRINT B PRE

BPOP tom

SEARCH erin

SEARCH mia


BPOP erin

BPOP elsa

PRINT H

EXIT

Data.txt

 data - Windows 메모장

파일(F) 편집(E) 서식(O)

```
tom 37.2 Y Seoul  
emily 36.5 N Incheor  
erin 38.1 Y Daegu  
elsa 38.4 Y Seoul  
mia 36.4 Y Ulsan  
minji 36.4 N Gwangju  
subin 36.1 N Daejeon  
greg 36.4 N Busan  
june 35.4 N Daegu  
moly 37.2 Y Seoul
```

5. Consideration

1. 명령어 성공 여부:

PRINT H 의 경우 Heap 영역이 만들어져 있지 않기 때문에 구동하지
않음.

BPOP 의 경우 PRINT H 와 마찬가지로 Heap 을 만들지 않았기 때문에
구동하지 않음.

2. Serach 를 만들었긴 했으나, QPOP 에서 오류가 났기 때문에 정상적으로 돌지 않음

3. Patient_BST, Location_BST 에서 삽입이 정확히 구동하는 것은 디버깅을 통해 확인했으나, pdf 파일에 나온 명령의 경우에서 실행하지 못하는 경우가 발생

4. Linux 환경에서 돌릴 수 없음

여담: 4 일동안 10 시간 자면서 만들었는데,,, 많이 성공하지 못했습니다,,,
열심히 노력했다는 점만 알려주세요,,,