

자료구조 과제 HW#2

Prof.조성현

Asis. 지서연

201711205 황정평

-컴파일 환경

window10, Visualstudio2017, _MSC_VER: 1916, MSYC++14.16

Prob-1.cpp

-알고리즘

1) 힙(heap)은 완전 이진 트리 기반의 특정한 자료구조를 의미한다.

또한 힙은 다음 특성을 만족한다.

- Heap-Order: for every node v other than root,
 $key(v) \geq key(parent(v))$

2) TimeComplexity 만족

문제 "각 정수가 입력될 때마다 가장 작은 N 개의 숫자를 업데이트하는데 걸리는 계산시간은 $O(\log N)$ 이하여야 한다."를 만족하기 위하여, 힙 구조에 N 개의 숫자가 넘어가면 바로 `removeMax`를 하고 힙 구조에 N 개 숫자 이하만 있도록 유지하였다.

-코드 설명

```
#include<iostream>
#include<string>
#include<vector>
using namespace std;
```

자료구조 string과 vector를 받아온다.

```
class MaxHeap {
    vector<int> v;
    vector<string> ans;
```

MinHeap 클래스를 만든다. 멤버 private 변수로 vector v 를 둔다. 그리고 답을 담는 공간인 vector ans 를 선언한다.

```
public:
    MaxHeap() {}
    bool isEmpty() {return v.size() == 0;}
```

vector v가 비었으면 true를 반환하고, 비어있지 않으면 false를 반환하는 bool 함수이다.

```
int getParentIndex(int index) { return (index - 1) / 2; }
int getLeftIndex(int index) { return index * 2 + 1; }
int getRightIndex(int index) { return index * 2 + 2; }
```

특정 index를 넣었을 때, 각각 힙 구조에서 그 index의 부모 index, 왼쪽 자식 index, 오른쪽 자식 index를 반환하는 함수들이다.

```
int getParentKey(int index) { return v[(index - 1) / 2]; }
int getLeftKey(int index) { return v[index * 2 + 1]; }
int getRightKey(int index) { return v[index * 2 + 2]; }
```

특정 index를 넣었을 때, 각각 힙 구조에서 그 index의 부모 key, 왼쪽 자식 key, 오른쪽 자식 key를 반환하는 함수들이다.

```
void insert(int key) {
    v.push_back(key);
    int insertIndex = v.size() - 1;
    while (insertIndex != 0 && key > getParentKey(insertIndex)) {
        int tmp = getParentKey(insertIndex);
        v[getParentIndex(insertIndex)] = key;
        v[insertIndex] = tmp;
        insertIndex = getParentIndex(insertIndex);
    }
}
```

insert 함수는 v벡터에 key값을, 힙 구조를 유지하며 넣어주는 함수이다. 우선 vector v 제일 뒤에 인자인 key값을 넣는다. 그 다음, 어디 index에 삽입할지 insertIndex 변수를 현재 key값이 있는 인덱스로 선언해준다. while문의 조건문이 의미하는 것은, 현재 key값이 있는 인덱스가 루트가 아니고, 부모의 키값보다 크다면, 부모의 키값과 자리를 바꾼다. 그리고, insertIndex를 부모 인덱스로 업데이트 해준다.

```
void removeMax() {
    if (isEmpty()) {
        return;
    }
    int lastKey = v[v.size() - 1];
    int parent = 0; // index 값
    int child = 1; // index 값
    while (child <= v.size() - 1) {
        if (child < v.size() - 2 && getLeftKey(parent) < getRightKey(parent)) {
            child++;
        }
        if (lastKey >= v[child]) {
            break;
        }
        v[parent] = v[child];
        parent = child;
        child = getLeftIndex(child);
    }
}
```

```

        v[parent] = lastKey;
        v.pop_back();
        return;
    }

```

removeMax 는 루트를 제거 해주고, 마지막 key값을 루트로 올리고, 다시 힙 구조를 맞추는 함수이다. 우선, 비어있으면 함수를 종료한다. 비어있지 않으면, lastKey에 마지막 key값을 넣어주고, parent에 0, child에 1을 넣어준다. (각각 루트, 루트의 왼쪽 자식을 가리킨다.) 일종의 시작점을 정해주는 것이다. 그 다음 child 가 vector v의 인덱스를 초과하지 않는 동안 while문을 수행한다. 왼쪽 자식의 key값보다, 오른쪽 자식의 key값이 크다면 child를 오른쪽 자식 인덱스를 가리키도록 1을 더해준다. 마지막 key값이 이러한 child인덱스의 key값보다 크다면, while문을 break하고, parent에 마지막 key값을 넣고, 마지막 인덱스에 있는 마지막 key값은 제거하고 함수를 종료한다. 하지만, 마지막 key값이 이러한 child인덱스의 key값보다 작다면, child인덱스의 key값을 parent인덱스에 넣고, parent에 child 인덱스를 넣고, child에는 child의 왼쪽 자식 인덱스를 넣고 다시 while문을 돌린다.

```

void insertAns(int many) {
    string out = "";
    if (many < v.size()) {
        removeMax();
    }
    for (int i = 0; i < v.size(); i++) {
        out += to_string(v[i]);
        out += " ";
    }
    ans.push_back(out);
}

```

답들을 ans에 담아주는 함수이다. 만약 size가 N보다 크다면 removeMax함수를 실행한다. 가장 작은 N개의 숫자를 업데이트하는 계산시간을 log N 으로 만드는 제일 중요한 알고리즘이다. vector v의 사이즈가 N보다 커지는 즉각적으로 큰 숫자를 제거해주는 것이다. 그 다음 ans 에 있는대로 담는다.

```

void display() {
    for (int i = 0; i < ans.size(); i++) {
        cout << ans[i] << endl;
    }
}

};

```

ans에 담긴 답들을 순서대로 한꺼번에 출력하는 함수이다.

```

int main() {
    doit();
}

```

main함수는 위와 같으므로 doit함수를 살펴보자.

```

void doit() {
    MaxHeap hjp;
    string num_str;
    getline(cin, num_str);
    int many = stoi(num_str);
}

```

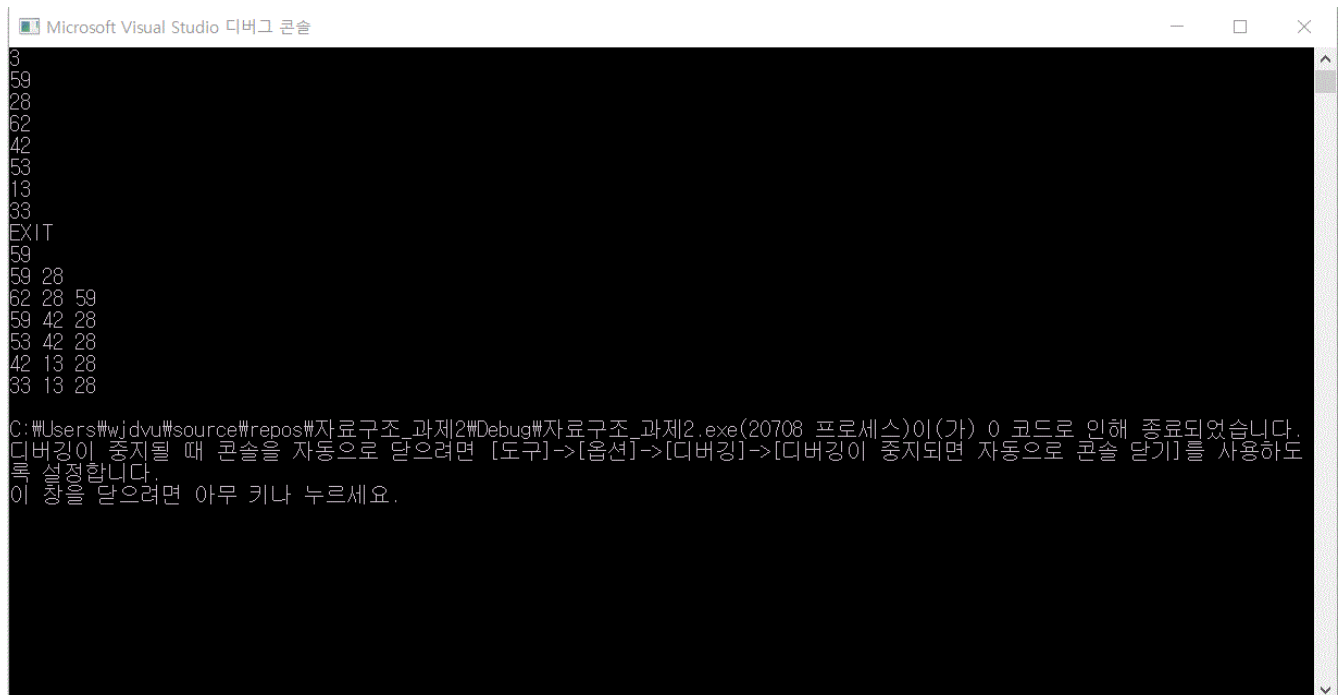
```

while (true) {
    string input;
    getline(cin, input);
    if (input == "EXIT") {
        return;
    }
    int input_int = stoi(input);
    hjp.insert(input_int);
    hjp.display(many);
}
}

```

먼저 MaxHeap class 객체 hjp를 선언한다. N값을 받기 위해, getline을 이용하여 string num_str에 받고, many에 int로 형전환하여 넣어준다. while문 안에서 다시 string input에 입력을 받고, 그 입력이 EXIT 이면 break하고 종료하도록 한다. EXIT가 아니라면 int로 형변환하여 input_int에 넣어주고, hjp에 insert하고, display한다.

-결과창



EXIT를 기준으로 위가 입력값들이고, 아래가 출력값들이다. 주어진 예제 입출력 그대로 잘 나온것을 확인할 수 있다.

Prob-2.cpp

-알고리즘

키와 값을 담은 클래스를 만들고, 그 클래스를 담은 벡터를 만들고, 이 벡터를 담은 벡터가 해시테이블이 되도록 구현하였다. 해시테이블은 해시값으로 그에 해당하는 버킷에 들어가 탐색을 하는 것이다. 모든 배열을 거치지 않고, 해시값 연산으로 특정 버킷에서 시작하는 자료구조이기에 어느 정도 효율성이 있다.

-코드 설명

```
#include<iostream>
#include<string>
#include<vector>
#include<sstream>
using namespace std;
```

자료구조 string, vector를 받아오고 string stream을 쓰기위해 sstream 도 받아온다.

```
class entry {
    string key;
    string value;
public:
    entry(string key, string value) : key(key), value(value) {}
    string getKey() {return key;}
    string getValue() { return value;}
};
```

key값과 value를 저장할 수 있는 class 'entry'를 만들어준다.

```
class hashTable {
    vector<vector<entry>>*> table;
    vector<string> ans;
    int tableSize;
    float loadFactor;
    int entryCount;
```

멤버 변수로는 vector를 담는 vector 'table'을 선언한다.

답을 담는 공간인 vector ans를 선언한다.

테이블 사이즈 값인 tableSize, 적재율 값인 loadFactor, 총 요소 개수 값인 entryCount 를 선언한다.

```
public:
    hashTable(int setSize=10) {
        table = new vector<vector<entry>>*>(setSize);
        tableSize = setSize;
        loadFactor = float(entryCount) / float(tableSize);
        entryCount = 0;
    }
```

인자로써는 setSize를 받아 table 크기를 선언해주고 tableSize에 넣어준다. entryCount 초기값은 0으로 하고, loadFactor 값을 업데이트 해준다.

```
int getTableSize() { return tableSize; }
float getLoadFactor() { return loadFactor; }
int getEntryCount() { return entryCount; }
```

기본적으로 멤버 private변수들을 반환하는 함수를 만든다.

```
int hashFunction(string key) {
    return key.length() % tableSize;
}
```

hashFunction 은 인자로 key를 받으면, 그 key의 길이를 테이블사이즈로 모드 연산한 값을 반환한다.

```
void size_display() {
    string out = "";
    out = to_string(getEntryCount()) + " " + to_string(getTableSize()) + " " +
to_string(getLoadFactor());
    ans.push_back(out);
}
```

size_display함수는 총entry개수, 테이블사이즈, 적재율을 ans에 담는다.

```
void search(string key) {
    int hashValue = hashFunction(key);
    vector<entry>* v = &(table->at(hashValue));
    for (int i = 0; i < v->size(); i++) {
        if (v->at(i).getKey() == key) {
            ans.push_back(v->at(i).getValue());
            return;
        }
    }
    ans.push_back("NOT FOUND");
}
```

search함수는 key값을 인자로 받고, 해쉬값을 구하고, 그에 맞는 해쉬테이블의 위치를 찾고, 그 위치에 해당 key값이 저장되어있는지 for문을 돌려 확인한다. 있다면 value 값을 ans에 담고, 없다면 "NOT FOUND"를 ans에 담는다.

```
void rehashing() {
    hashTable new_hashTable(tableSize * 2);
    for (int i = 0; i < table->size(); i++) {
        for (int j = 0; j < table->at(i).size(); j++) {
            new_hashTable.add(this->table->at(i).at(j));
        }
    }
    this->tableSize *= 2;
    this->loadFactor = float(entryCount) / float(tableSize);
    this->table = new_hashTable.table;
    ans.push_back("REHASHING");
}
```

rehashing함수는 사이즈가 두배인 해쉬테이블을 만들어 그 곳에 다시 모든 entry를 넣어준다. 우선 사이즈가 두배인 새로운 해쉬테이블을 선언한다. 모든 entry를 넣어주기 위해 이중 for문을 돌려 새로운 해쉬테이블에 add한다. 기존 테이블사이즈 변수값을 두배해주고, 적재율도 업데이트해주고, 기존 해쉬테이블을 새로운 해쉬테이블로 교체한다. 그리고 "REHASHING"을 ans에 담는다.

```
void add(entry e) {
    int hashValue = hashFunction(e.getKey());
```

```

        vector<entry>* v = &(table->at(hashValue));
        for (int i = 0; i < v->size(); i++) {
            if (v->at(i).getKey() == e.getKey()) {
                v->at(i) = e;
                return;
            }
        }
        v->push_back(e);
        entryCount++;
        loadFactor = float(entryCount) / float(tableSize);
        if (loadFactor >= 0.5) {
            rehashing();
        }
    }
}

```

add함수는 해쉬테이블에 entry를 넣어준다. key와 value값을 가지고 있는 entry를 인자로 받는다. 그 entry key값에 해당하는 해쉬값을 구하고, 그에 맞는 해쉬테이블의 위치를 찾고 원소를 for문으로 하나하나 우선 비교한다. 만약 key 값이 같은게 있다면 value값을 업데이트한다. 없다면 제일 뒤에 entry를 추가하고 entryCount를 1증가 시키고, 적재율을 업데이트하고, 이 적재율이 0.5이상이라면 rehashing을 한다.

```

void erase(string key) {
    int hashValue = hashFunction(key);
    vector<entry>* v = &(table->at(hashValue));
    for (int i = 0; i < v->size(); i++) {
        if (v->at(i).getKey() == key) {
            v->erase(v->begin() + i);
            entryCount--;
            return;
        }
    }
    ans.push_back("NOT FOUND");
}

```

erase함수는 key값을 인자로 받고, 그에 해당하는 key값을 지운다. key값에 해당하는 해쉬값을 구하고, 그 해쉬값에 해당하는 해쉬테이블의 위치로가서 for문으로 그 위치에 있는 entry를 다 훑는다. 해당하는 key값이 있다면 지우고, entryCount를 1감소 시킨다. 없다면 "NOT FOUND"를 ans에 담는다.

```

void display() {
    for (int i = 0; i < ans.size(); i++) {
        cout << ans[i] << endl;
    }
}
};

```

vector ans에 담긴 답들을 순서대로 출력하는 함수이다.

```

int main() {
    doit();
}

```

main()함수는 위와 같으므로 doit함수를 살펴보자.

```

void doit() {
    hashTable hjp;
    while (true) {
        string input_getline;
        getline(cin, input_getline);
        stringstream input_ss(input_getline);
        string input;
        input_ss >> input;

```

우선 해시테이블 객체 hjp를 선언한다. 그 다음 무한반복문을 열고, string input_getline에 입력을 받고, 이걸 공백기준으로 입력을 넣어주기 위해 stringstream을 이용한다. 이제 input_ss가 하나씩 값을 전달 할 수 있다. (공백기준)

```

    if (input == "EXIT") {
        break;
    }

```

들어온 첫 입력값이 EXIT라면 무한반복문을 break한다.

```

    else if (input == "SIZE") {
        hjp.size_display();
    }

```

들어온 첫 입력값이 SIZE라면 size_display함수를 실행한다.

```

    else if (input == "PUT") {
        string key;
        string value;
        input_ss >> key;
        input_ss >> value;
        entry e(key, value);
        hjp.add(e);
    }

```

들어온 첫 입력값이 PUT이라면 다음 입력값들을 각각 key, value에 받고, entry를 형성하여 객체 hjp에 add함수를 이용하여 담아준다.

```

    else if (input == "FIND") {
        string key;
        input_ss >> key;
        hjp.search(key);
    }

```

들어온 첫 입력값이 FIND라면 다음 입력값을 key에 받고 search함수를 실행한다.


```

        else if (input == "ERASE") {
            string key;
            input_ss >> key;
            hjp.erase(key);
        }
    }
    hjp.display();
}

```

들어온 첫 입력값이 ERASE라면 다음 입력값을 key에 받고 erase함수를 실행한다.

무한반복문을 break하고 나왔을 때, display함수를 실행하여 답을 동시에 보여준다.

-결과창



```

Microsoft Visual Studio 디버깅 콘솔
SIZE
PUT ADELSON-VELSKII 1962
PUT AHO 1990
PUT BARJUKA 1926
PUT BAYER 1972
ERASE BAYER
ERASE AHO
PUT BENTLEY 1983
PUT BOOCH 1994
FIND BENTLEY
PUT BENTLEY 1985
FIND BENTLEY
PUT CORMEN 2001
PUT DIJKSTRA 1959
FIND AHO
PUT LIPPMANN 2000
PUT MCCREIGHT 1976
PUT MEHLHORN 1990
SIZE
EXIT
0 10 0.000000
1983
1985
REHASHING
NOT FOUND
9 20 0.450000

C:\Users#wjdvu\source\repos\자료구조_과제2\Debug\자료구조_과제2.exe(18500 프로세스)이(가) 0 코드로 인해 종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]->[디버깅]->[디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.

```

EXIT를 기준으로 위가 입력값들이고, 아래가 출력값들이다. 주어진 예제 입출력 그대로 잘 나온것을 확인할 수 있다.
(입력을 다 대문자로 하였다.)