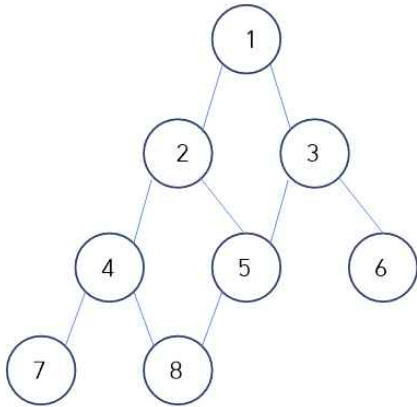


● Windows 10, Dev-C++ 5.11, TDM-GCC 4.9.2 64-bit Release 환경에서 작성하였다.

문제 1 : heap 기반 Priority Queue 구현

1. CompleteBinaryTree/heap의 구현 방법

- heap property 중 Complete binary tree여야 한다는 조건이 있으므로, Complete binary tree를 구현하도록 한다.



원 안의 숫자는 벡터에서의 index이다.

- 벡터의 index를 iterator position으로 다시 정의한다. 벡터의 첫 칸의 index 인 0은 빈칸으로 남겨 두고, tree의 root 노드는 index 1로 설정한다.

- 어떤 internal node의 index가 i 라고 하면, 그 internal node의 left child의 index는 $2i$, right child의 index는 $2i+1$ 이다.

- heap에 새로운 key를 추가할 때는 벡터의 마지막에 원소를 추가한 후, Up-heap bubbling으로 heap property를 맞춘다.

- heap에서 최소 key를 삭제할 때는 최소 key와 마지막 key의 위치를 바꾸고, 마지막 위치의 key를 삭제하면 최소키가 삭제된다.

최소키를 삭제한 뒤에는 현재 root에 있는 key를 Down-heap bubbling을 이용하여 heap property를 만족하는 위치에 가도록 한다.

* Up-heap bubbling과 Down-heap bubbling은 isless 함수와 swap 함수를 이용한다. isLess 함수를 이용하여 key간의 대소 관계를 파악하고, 순서 관계를 만족하지 않으면 swap을 실행하여 순서 경향성을 맞춘다.

* vector의 마지막에 원소를 넣은 후에, 아래에서부터 Up-heap bubbling을 이용해서 heap property(여기서는 작은 수가 우선순위가 더 높다)를 맞추므로, heap의 내용을 업데이트 하는 데 걸리는 시간은 $O(\log n)$ 이 걸린다.

2. PriorityQueue에 저장된 값 중 n개의 최솟값을 순서대로 출력하기

- heap(priority Queue)에 값을 입력한다.
- 현재 heap에 들어 있는 내용을 새 heap에 그대로 복사한다. (원래 heap을 보존하기 위함. 원래 heap에서 removeMin을 실행하면 다음 단계에서 heap의 최솟값이 바뀜)
- 복사한 heap에서 n개 최솟값을 차례대로 뽑는다. n개 최솟값의 출력 순서는 중요하지 않음
- 알아낸 최솟값을 output string에 저장한다.
- 새로운 값이 입력되면 위의 과정을 반복한다.
- 값의 입력을 중단하면 output string을 출력한다.

3. VectorCompleteTree 클래스의 구성 요소

3-1 private 멤버

V : complete tree가 될 벡터

Vcopy : V의 내용을 복사해 둔 벡터

minValues : 벡터 V(Vcopy)의 최솟값을 저장하는 벡터

3-2 protected 멤버

Position pos(int l) : 벡터의 인덱스를 반환하는 함수

int idx(const Position& p) : 현재 위치의 인덱스를 반환하는 함수

3-3 public 멤버

VectorCompleteTree() : 생성자

int size() : tree의 노드 수 반환

Position left(const Position &p), right(const Position &p) : 현재 노드의 left, right child의 인덱스 반환

Position parent(const Position &p) : 현재 노드의 parent의 인덱스 반환

bool hasLeft(const Position &p), hasRight(const Position &p) : 현재 노드가 left, right child를 가지는지 판별

bool isRoot(const Position &p) : 현재 노드가 root인지 판별

Position Root(), last() : root, 마지막 노드의 인덱스 반환

void addLast(const int &e), removeLast : 현재 tree의 마지막에 노드 추가/삭제

void swap(const Position &p, const Position &q) : up-heap bubbling을 구현.

* heap property를 유지시키는 데 중요한 역할을 하는 swap 함수는 자세히 소개한다. 부모 노드의 우선순위가 자식 노드의 우선순위보다 높아야 한다는 조건이 있으므로, isLess 함수로 노드의 우선순위를 파악한 다음, 규칙에 맞지 않으면 swap 함수로 규칙을 만족하도록 한다.

<pre>void swap(const Position& p, const Position& q){ int e = *q; *q = *p; *p = e; }</pre>	<pre>int HeapPriorityQueue::isLess(int a, int b){ if (a<b) return true; else return false; }</pre>
--	---

void vectorcopy(), vectorNstore(), vectorNMin(), printNmin() : 벡터 V의 내용을 Vcopy에 복사하고, Vcopy에 있는 n개의 최솟값을 minValues에 저장한 다음, minValues를 출력.

string intToString(int n) : int 타입을 string 타입으로 변환

4. HeapPriorityQueue 클래스의 구성 요소

int size() : PQ의 크기 반환

bool empty() : PQ가 비었는지 판단

void insert() : PQ에 원소를 삽입

int &min() : PQ의 최소 원소 반환, 즉 heap의 root원소 반환

void removeMin(), void removeLast() : heap의 root 노드, 마지막 노드 삭제

int isLess(int a, int b) : a가 b보다 작은지 판별

void heapcopy(), heapNstore(int num), heapNmin(), printheapNmin() : 현재 PQ의 내용을 복사해서(원래 heap을 보존) n개의 최솟값을 저장한 후, n개의 최솟값을 출력

5. 작성한 코드

```
#include <cstdio>
#include <string.h>
#include <cstring>
#include <iostream>
#include <vector>
#include <algorithm>
#include <sstream>

using namespace std;
class VectorCompleteTree{
private:
    vector<int> V; //heap이 될 completebiarytree
    vector<int> Vcopy; //heap의 내용을 복사할 벡터
    vector<int> minValues; //heap의 n개의 최솟값을 저장할 벡터

public:
    typedef typename std::vector<int>::iterator Position; //tree의 위치를 position이라
정의
    string output; //결과를 한꺼번에 출력할 string

protected:
    Position pos(int i){
        return V.begin() + i;
    }
    int idx(const Position& p) const{
        return p-V.begin();
    }

public:
    int min;
    VectorCompleteTree() : V(1) { }
    int size() const {
        return V.size()-1;
    }
    Position left(const Position& p){
        return pos(2*idx(p));
    }
    Position right(const Position& p){
        return pos(2*idx(p)+1);
    }
    Position parent(const Position& p){
        return pos(idx(p)/2);
    }
    bool hasLeft(const Position& p) const {
        return 2*idx(p) <= size();
    }
    bool hasRight(const Position& p) const{
        return 2*idx(p) + 1 <= size();
    }
    bool isRoot(const Position& p) const{
        return idx(p)==1;
    }
}
```

```

        Position root(){
            return pos(1);
        }
        Position last(){
            return pos(size());
        }
        void addLast(const int& e){
            V.push_back(e);
        }
        void removeLast(){
            V.pop_back();
        }
        void swap(const Position& p, const Position& q){
            int e = *q;
            *q = *p;
            *p = e;
        }
        void vectorcopy(){
            Vcopy.clear();
            Vcopy.assign(V.begin()+1,V.end());
        }
        void vectorNstore(int num){
            minValues.clear();
            if (Vcopy.size()<num){
                minValues.assign(Vcopy.begin(), Vcopy.end());
            }
            else{
                for(int i=0; i<num; i++){
                    min = *min_element(Vcopy.begin(), Vcopy.end());
                    minValues.push_back(min);
                    Vcopy.erase(min_element(Vcopy.begin(), Vcopy.end()));
                }
            }
        }
        void vectorNmin(){
            for(int i=0; i<minValues.size(); i++){
                output += intToString(minValues[i]);
                output += " ";
            }
            output += "\n";
        }
        void printNmin(){
            cout << output << endl;
        }
        string intToString(int n){
            stringstream s;
            s << n;
            return s.str();
        } // 정수를 string타입으로 변환
};

class HeapPriorityQueue{

```

```

public:
    int size() const;
    bool empty() const;
    void insert(const int& e);
    const int& min();
    void removeMin();
    void removeLast();
    VectorCompleteTree T;
    int isLess(int a, int b);
    void heapcopy();
    void heapNstore(int num);
    void heapNmin();
    void printheapNmin();
    typedef typename VectorCompleteTree::Position Position;
};

int HeapPriorityQueue::size() const{
    return T.size();
}

bool HeapPriorityQueue::empty() const{
    return T.size() == 0;
}

const int& HeapPriorityQueue::min(){
    return *(T.root());
}

void HeapPriorityQueue::insert(const int& e){
    T.addLast(e);
    Position v = T.last();
    while(!T.isRoot(v)){
        Position u = T.parent(v);
        if(!isLess(*v, *u))
            break;
        T.swap(v,u);
        v = u;
    }
}

void HeapPriorityQueue::removeMin(){
    if(size() == 1)
        T.removeLast();
    else{
        Position u = T.root();
        T.swap(u,T.last());
        T.removeLast();
        while(T.hasLeft(u)){
            Position v = T.left(u);
            if(T.hasRight(u) && isLess(*(T.right(u)),*v)){
                v = T.right(u);
            }
            if(isLess(*v, *u)){
                T.swap(u,v);
                u = v;
            }
        }
        else
    }
}

```

```

                                break:
                                }
                                }
}
void HeapPriorityQueue::heapcopy() {
    T.vectorcopy();
}
void HeapPriorityQueue::heapNstore(int num){
    T.vectorNstore(num);
}
void HeapPriorityQueue::heapNmin(){
    T.vectorNmin();
}
void HeapPriorityQueue::printheapNmin(){
    T.printNmin();
}
int HeapPriorityQueue::isLess(int a, int b){
    if (a<b)
        return true;
    else
        return false;
}
int main(void){
    int num;
    int element;
    string output;
    cin >> num;
    HeapPriorityQueue Q;
    while(1){
        cin >> element;
        if (cin.good()){
            Q.insert(element); // 값을 하나씩 넣음
            Q.heapcopy(); // heap의 내용을 복사
            Q.heapNstore(num); // heap의 num개의 최솟값 저장
            Q.heapNmin(); // 단계별 num개의 최솟값을 output에 저장
        }
        else if (cin.fail())
            break:
    }
    Q.printheapNmin(); // output 출력
}

```

6. 실행 결과 - lms의 test code/과제 안내 사항의 test code

test code를 입력해 본 결과, 과제 수행자의 컴퓨터에서 오류는 발생하지 않았다.

```
C:\Users\이성욱\Desktop\과제\자료구조\자료구조 과제2 PQ.HashMap\최종\Prob-1.exe
3
5
5
78
53
13
25
1
11
49
34
35
49
11
3
299
EXIT
5
5
5
78
78 53
13 25 78
13 25 63 78
1 5 13 25 53
1 5 11 13 25
1 5 11 13 25
1 5 11 13 25
1 5 11 13 25
1 5 11 13 25
1 5 8 11 13
1 5 8 11 13

C:\Users\이성욱\Desktop\과제\자료구조\자료구조 과제2 PQ.HashMap\최종\Prob-1.exe
3
59
28
62
42
53
13
33
EXIT
59
28 59
28 59 62
28 42 59
28 42 53
13 28 42
13 28 33

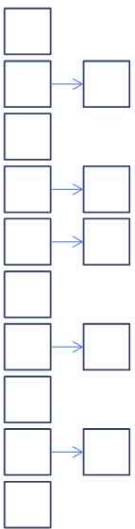
-----
Process exited after 1.211 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .

-----
Process exited after 2.739 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

문제 2 : Hash Table 기반 Map 구현

- Windows 10, Dev-C++ 5.11, TDM-GCC 4.9.2 64-bit Release 환경에서 작성하였다.
- 변수 초기화 부분에서 경고가 발생했으나, 과제 수행자의 컴퓨터에서 코드는 작동하였다.

1. HashMap 구현 방법

- Node* table entry
- 
- Collision Handling 방법으로 Separate Chaining 방법을 사용하므로, LinkedList로 구현하였다.
 - 초기에 Node* table[table_size] 라는 Node 타입의 포인터 배열을 선언한다. 각각의 노드에는 0~table_size-1 까지의 index가 부여된다. 초기 Hash table의 크기가 10이므로 table_size=10 이다. Rehashing이 필요할 때, table_size=20으로 변경하고 새 포인터 배열을 만든다.
 - 새로운 Entry(node)가 입력되면, HashFunction을 이용하여 string type의 key를 숫자로 변환한다. HashFunction은 key의 길이를 table_size로 나눈 값이다.
 - key가 변환되면, entry node를 table의 해당하는 index에 연결한다. 아래 코드에서는 table node가 새롭게 만들어진 entry node를 가리키는 형식으로 노드를 연결하였다.
 - 이미 저장되어 있는 key를 삽입 시도하면, 기존에 저장되어 있던 entry를 삭제하고 새로운 값을 저장한다.
- load factor가 0.5 이상이 되면, 크기를 2배로 늘린 table을 선언 후, 기존 table에 연결되어 있던 node를 새 table에 연결한다.

2. transform/HashFunction 함수 설명

inline int transform(string key) : string type key를 정수형으로 변환한다.

inline int hashFunction(string key) : string type key를 정수형으로 변환한 후, 정수를 table_size로 나누어서 할당할 index를 결정한다.

3. Node 클래스의 구성 요소

string key : Entry가 저장하는 key

string value : Entry가 저장하는 value

Node* link : 다음 Node를 가리키는 포인터 변수

Node(string k = "", string v = "", Node* link=NULL) : 생성자

void set(string k, string v) : key를 k로, value를 v로 설정

void reset() : key, value를 기본값으로 초기화 //test용 코드

bool isEmpty() : 현재 노드가 비었는지 확인

bool equal(string k) : 입력받은 key와 노드의 key가 동일한지 확인

string display() : 현재 노드가 저장하는 value 반환

Node *getLink() : 다음 노드의 주소 반환

void setLink(Node* next) : 현재 노드의 다음 노드를 next로 지정

string getKey() : 현재 노드의 key 반환

4. HashChainMap 클래스의 구성 요소

Node* table[table_size] : Hash Table의 기본이 되는 포인터 배열. 나중에 load factor가 0.5 이상이 되면, 새로 만든 table에 entry들을 넘겨주고 삭제된다.

double entrynum : entry의 개수 정보를 저장

HashChainMap() : table 포인터가 모두 NULL을 가리키도록 함. HashTable의 생성자

addEntry(string key, string value) : key, value를 가지는 Entry를 Hash Table에 추가.

새 노드는 table node와 기존에 table에 연결되어 있던 노드 사이에 삽입한다. 그리고 table node 가 새로 삽입한 만든 entry 노드를 가리키도록 한다. 노드를 추가한 뒤에는 entrynum 을 1 증가시킨다.

* 만약 같은 key를 추가한다면, 기존 노드를 초기화 하여 내용을 지운 후 entrynum을 1 감소시키고, 새로운 노드를 추가한 뒤 entrynum을 1 증가시킨다.

* Rehashing 구현 부분은 코드를 아래에 첨부하였다.

```
if((entrynum)/table_size >= 0.5){
    table_size = 20;
    Node *newtable = new Node[table_size];
    for(int i=0; i<table_size; i++){
        (newtable+i)->link = (table+i) -> link;
    }
    Node* old = table;
    table = newtable;
    delete [] old;
    return "REHASHING\n";
}
```

string searchEntry(string key) : key를 가지는 Entry가 가지는 value를 반환

void eraseEntry(string key) : key를 가지는 node의 내용을 삭제한다. (내용이 빈 node로 만든다.) node를 삭제한 후, entrynum을 1 감소시킨다.

5. 작성한 코드

```
#include <stdio>
#include <cstring>
#include <string.h>
#include <iostream>
#include <sstream>
using namespace std;
int table_size = 10;
int new_table_size = 20; //사용하지 않는 변수

inline int transform(string key){
    int number = 0;
    number = key.size();
    return number;
} // 문자열로 된 키를 숫자로 변환

inline int hashFunction(string key){
    return transform(key) % table_size;
} // 변환한 수의 나머지를 구하는 함수
```

```

class Node{
    public:
        string key;
        string value;
        Node* link: //다음 노드를 가리킴
        Node(string k = "", string v = "", Node* link = NULL){
            set(k,v);
            link = NULL;
        }
        void set(string k, string v){
            key = k;
            value = v;
        }
        void reset(){
            set(" ",0);
        }
        bool isEmpty(){
            return key == "";
        }
        bool equal(string k){
            return !k.compare(key);
        }
        string display(){
            return value;
        }
        Node* getLink(){
            return link;
        }
        void setLink(Node* next){
            link = next;
        }
        string getKey(){
            return key;
        }
};

class HashChainMap{
    Node *table = new Node[table_size];
    //Node *newtable = new Node[new_table_size]; 사용하지 않는 부

    public:
        double entrynum = 0;
        HashChainMap(){
            for(int i=0; i<table_size; i++){
                (table+i)->link = NULL;
            }
        }

        string addEntry(string key, string value){

```

```

        int hashValue = hashFunction (key);
        for(Node *p = table+hashValue ; p != NULL; p=p->getLink()){
            if(p->equal(key)){
                p->set(" ", " ");
                entrynum --;
            }
        }
        Node* n = new Node;
        n->set(key,value);
        Node* p = (table+hashValue) -> link;
        (table+hashValue) -> link = n;
        n->link = p;
        entrynum = entrynum + 1;

        if((entrynum)/table_size >= 0.5){
            table_size = 20;
            Node *newtable = new Node[table_size];
            for(int i=0; i<table_size; i++){
                (newtable+i)->link = (table+i) -> link;
            }
            Node* old = table;
            table = newtable;
            delete [] old;

            return "REHASHING\n";
        }
        else
            return "";

    }

    string searchEntry(string key){
        int hashValue = hashFunction(key);
        for(Node *p= table+hashValue; p != NULL; p=p->getLink()){
            if(p->equal(key)){
                //printf("탐색 성공\n"); //test용
                return p->display();
            }
        }
        return "NOT FOUND";
    }

    void eraseEntry(string key){
        int hashValue = hashFunction(key);
        for(Node *p=(table+hashValue); p != NULL; p=p->getLink()){
            if(p->equal(key)){
                p->set(" ", " ");
                entrynum--;
            }
        }
    }

};

string doubleToString(double n){

```

```

        stringstream s;
        s << n;
        return s.str();
    }
}

int main(void){
    HashChainMap hash;
    string operation;
    string key;
    string value;
    string output;
    string find = "FIND" ; string put = "PUT"; string erase = "ERASE";
    string size = "SIZE" ; string exit = "EXIT";
    while(1){
        cin >> operation;
        if (operation.compare(find)==0) //find
        {
            cin >> key;
            output += hash.searchEntry(key);
            output += "\n";
        }
        else if (operation.compare(put)==0) //entry 추
        {
            cin >> key;
            cin >> value;
            output += hash.addEntry(key,value);
        }
        else if (operation.compare(erase)==0)
        {
            cin >> key;
            hash.eraseEntry(key);
        }
        else if (operation.compare(size)==0)
        {
            output += doubleToString(hash.entrynum);
            output += " ";
            output += doubleToString(table_size);
            output += " ";
            output += doubleToString((hash.entrynum)/(table_size));
            output += "\n";
        }
        else if (operation.compare(exit)==0)
        {
            break;
        }
        else
            cout << "wrong instruction" << endl;
    }
    cout << endl;
    cout << output << endl;          //명령 실행 결과 출력
}

```

6. 실행 결과 - lms의 test code/과제 안내 사항의 test code

test code를 입력해 본 결과, 과제 수행자의 컴퓨터에서 오류는 발생하지 않았다.

C:\Users\이상목\Desktop\과제\자료구조\자료구조 과제2 PQ,HashMap\최종	C:\Users\이상목\Desktop\과제\자료구조\자료구조 과제2 PQ,HashMap\최종
SIZE PUT APPLE 2 PUT PEACH 50 PUT GRAPE 31 PUT BANANA 99 FIND AVOCADO PUT LEMON 11 SIZE FIND BANANA PUT MANGO 444 ERASE MANGO PUT ORANGE 70 FIND ORANGE PUT ORANGE 77 FIND ORANGE SIZE EXIT 0 10 0 NOT FOUND REHASHING 5 20 0.25 99 70 77 6 20 0.3 ----- Process exited after 2.082 seconds with return value 0	PUT Adelson-Velskii 1962 PUT Aho 1990 PUT Baruvka 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 1983 1985 REHASHING NOT FOUND 9 20 0.45 ----- Process exited after 1.048 seconds with return value 0 계속하려면 아무 키나 누르십시오 . . .