

# HW11 실습: Max Heap

(과제 아닙니다, 제출하지 않습니다.)

질문

pemds81718@gmail.com

## 1. 테스트 케이스

// 첫줄에 점수의 개수, 두번째 줄부터는 점수들이다.

<score.txt>

```
6
21
14
20
2
10
15
```

## 2. hw11.cpp

```
#include <fstream>
#include <iostream>

#include "ArrayMaxHeap.h"

int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <input_file>" << std::endl;
        return 1;
    }

    std::ifstream fin(argv[1]);

    if (!fin) {
        std::cerr << "Error opening file: " << argv[1] << std::endl;
        return 1;
    }

    int n;
    fin >> n;

    ArrayMaxHeap<int> heap;
    for (int i = 0; i < n; ++i) {
        int value;
        fin >> value;
        heap.add(value);
    }

    std::cout << heap;

    fin.close();
    return 0;
}
```

### 3. HeapInterface.h

```
/** 추상자료형 Heap 의 인터페이스
 * @file HeapInterface.h
 */

#ifndef __HEAP_INTERFACE_H__
#define __HEAP_INTERFACE_H__


template <typename T>
class HeapInterface {
public:
    /** 힙이 비어있는지 판단
     * @return 비어잇으면 true, 아니면 false
     */
    virtual bool isEmpty() const = 0;

    /** 주어진 데이터를 갖는 새로운 노드를 힙에 추가
     * @param someItem 노드에 추가할 데이터
     * @post 새로운 노드가 힙에 추가도미
     * @return 추가에 성공하면 true, 실패하면 false
     */
    virtual bool add(const T& someItem) = 0;

    /** 힙의 루트 노드(최대/최소)를 제거
     * @return 성공적으로 제거되면 true, 실패하면 false
     */
    virtual bool remove() = 0;

    /** 힙의 모든 노드를 제거 */
    virtual void clear() = 0;

    /** 힙의 루트(탑)에 있는 데이터를 가져옴
     * 최대힙이라면 최댓값이, 최소힙이라면 최솟값이 됨
     * @pre 힙에 원소가 적어도 하나 있어야 함
     * @post 루트의 데이터가 반환되고, 힙은 변경되지 않음
     * @return 루트 노드의 데이터
     */
    virtual T peekTop() const = 0;

    /** 힙의 원소의 개수를 가져옴
     * @return 힙의 원소 개수
     */
    virtual int getNumberOfNodes() const = 0;

    /** 힙의 높이를 가져옴
     * @return 힙의 높이
     */
    virtual int getHeight() const = 0;

    /** 힙을 소멸하고 할당된 메모리를 반환함 */
    virtual ~HeapInterface() {}
}
```

```
}; // end HeapInterface  
#endif // __HEAP_INTERFACE_H__
```

## 4. ArrayMaxHeap.h(참고하여 cpp 구현)

```
/** 추상자료형 Heap 의 배열 기반 구현
 * @file ArrayMaxHeap.h
 */

#ifndef __ARRAY_MAX_HEAP_H__
#define __ARRAY_MAX_HEAP_H__


#include <iostream>
#include <memory>

#include "HeapInterface.h"

template <typename T>
class ArrayMaxHeap : public HeapInterface<T> {
private:
    static const int ROOT_INDEX = 0; // 가독성에 도움
    static const int DEFAULT_CAPACITY =
        21; // 꽉 찬 힙을 테스트해보기 위해 작게 설정
    std::unique_ptr<T[]> heap; // 힙 원소의 배열
    int itemCount; // 힙의 원소 개수
    int maxItems; // 힙의 최대 용량

    // 대부분의 비공개 유ти리티 함수들은 매개변수와 계산에서 배열 인덱스를
    // 사용함. 이 함수들이 비공개이기 때문에 배열이 구현상의 세부사항이라는
    // 점에도 불구하고 이는 안전함.

    // 존재할 경우 해당 배열 인덱스의 왼쪽 자식의 인덱스를 반환
    int getLeftChildIndex(int nodeIndex) const;

    // 존재할 경우 해당 배열 인덱스의 오른쪽 자식의 인덱스를 반환
    int getRightChildIndex(int nodeIndex) const;

    // 해당 노드의 부모 노드의 인덱스를 반환
    int getParentIndex(int nodeIndex) const;

    // 노드가 리프노드인지 판단
    bool isLeaf(int nodeIndex) const;

    // semiheap 을 heap 으로 변환
    void heapRebuild(int subTreeRootIndex);

    // 정렬되지 않은 배열로부터 힙을 생성
    void heapCreate();

public:
    ArrayMaxHeap();
    ArrayMaxHeap(const T someArray[], const int arraySize);
    virtual ~ArrayMaxHeap();

    // HeapInterface 의 public 멤버 함수
    bool isEmpty() const noexcept;
```

```

int getNumberOfNodes() const noexcept;
int getHeight() const noexcept;
T peekTop() const;
bool add(const T& someItem);
bool remove();
void clear() noexcept;

template <typename U>
friend std::ostream& operator<<(std::ostream& os,
                                         const ArrayMaxHeap<U>& heap);
};

// end ArrayMaxHeap

template <typename T>
std::ostream& operator<<(std::ostream& os, const ArrayMaxHeap<T>& heap) {
    if (heap.itemCount == 0) {
        os << "(empty)\n";
        return os;
    }

    // 재귀 출력 함수: prefix 는 현재 줄의 접두사, isLast 는 형제 중 마지막인지 여부
    std::function<void(int, const std::string&, bool)> printNode;
    printNode = [&](int idx, const std::string& prefix, bool isLast) {
        if (idx >= heap.itemCount) return;
        os << prefix;
        os << (isLast ? "└ " : "├ ");
        os << heap.heap[idx] << '\n';

        int left = heap.getLeftChildIndex(idx);
        int right = heap.getRightChildIndex(idx);

        // 자식이 없으면 종료
        if (left >= heap.itemCount && right >= heap.itemCount) return;

        std::string childPrefix = prefix + (isLast ? "    " : "| ");

        // 왼쪽이 있고 오른쪽도 있으면 왼쪽은 isLast=false, 오른쪽은 isLast=true
        if (left < heap.itemCount && right < heap.itemCount) {
            printNode(left, childPrefix, false);
            printNode(right, childPrefix, true);
        } else if (left < heap.itemCount) {
            printNode(left, childPrefix, true);
        } else if (right < heap.itemCount) {
            printNode(right, childPrefix, true);
        }
    };
}

// 루트 출력 후 자식 출력 시작
os << heap.heap[0] << '\n';
int l = heap.getLeftChildIndex(0);
int r = heap.getRightChildIndex(0);
if (l < heap.itemCount && r < heap.itemCount) {
    printNode(l, "", false);
    printNode(r, "", true);
} else if (l < heap.itemCount) {

```

```
    printNode(l, "", true);
}

return os;
}

#include "ArrayMaxHeap.cpp"

#endif // __ARRAY_MAX_HEAP_H__
```

## 5. 결과

실행: ./hw11 score.txt

```
> ./hw11 score.txt
```

```
21
└── 14
    └── 2
        └── 10
    └── 20
        └── 15
```

```
> ./hw11 score.txt
```

```
40
└── 17
    ├── 13
    │   ├── 4
    │   │   ├── 2
    │   │   └── 3
    │   └── 12
    │       └── 5
    └── 16
        ├── 10
        └── 14
└── 21
    ├── 19
    │   ├── 15
    │   └── 18
    └── 20
        └── 11
            └── 1
```

## 6. 더 생각해보기

### 1) 절댓값 최소 힙:

절댓값 최소 힙은 다음과 같은 연산을 지원하는 자료구조이다.

1.INSERT : 배열에 정수  $x$  ( $x \neq 0$ )를 넣는다.

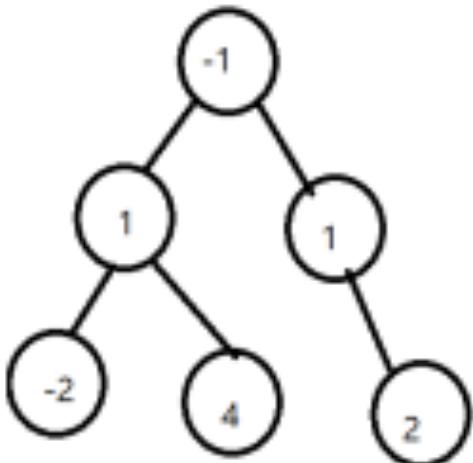
2.POP : 배열에서 절댓값이 가장 작은 값을 출력하고, 그 값을 배열에서 제거한다.

절댓값이 가장 작은 값이 여러개일 때는, 값이 가장 작은 수를 출력하고, 그 값을 배열에서 제거한다.

프로그램은 처음에 비어있는 배열에서 시작하게 된다.

ex)

입력 : -1, 1, 2, -2, 4, 1



pop 반복

위에서 구현한 힙 코드에서 어떤 부분만 바꾸면 될까?

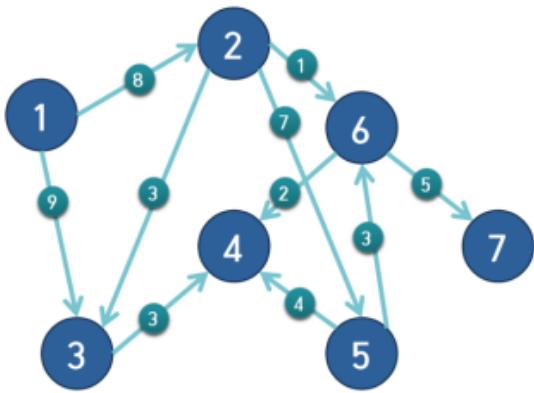
## 2) 이중 우선순위 큐:

이중 우선순위 큐(dual priority queue)는 전형적인 우선순위 큐처럼 데이터를 삽입, 삭제할 수 있는 자료 구조이다. 큐와의 차이점은 데이터를 삭제할 때 연산(operation) 명령에 따라 우선순위가 가장 높은 데이터 또는 가장 낮은 데이터 중 하나를 삭제하는 점이다. 이중 우선순위 큐를 위해선 두 가지 연산이 사용되는데, 하나는 데이터를 삽입하는 연산이고 다른 하나는 데이터를 삭제하는 연산이다. 데이터를 삭제하는 연산은 또 두 가지로 구분되는데 하나는 우선순위가 가장 높은 것을 삭제하기 위한 것이고 다른 하나는 우선순위가 가장 낮은 것을 삭제하기 위한 것이다.

정수만 저장하는 이중 우선순위 큐  $Q$  가 있다고 가정하자.  $Q$  에 저장된 각 정수의 값 자체를 우선순위라고 간주하자.

힙을 이용해 이중 우선순위 큐를 만들 수 있을까?

3) (그래프) 한 노드로부터 다른 노드까지 최단 경로 구하기:



정점	1	2	3	4	5	6	7
거리							