객체지향프로그래밍과 자료구조 (실습)

Lab.8. (보충설명) Event Processing with Priority Queue

Prof. Young-Tak Kim

Advanced Networking Technology Lab. (YU-ANTL)

Dept. of Information & Comm. Eng, College of Engineering,

Yeungnam University, KOREA

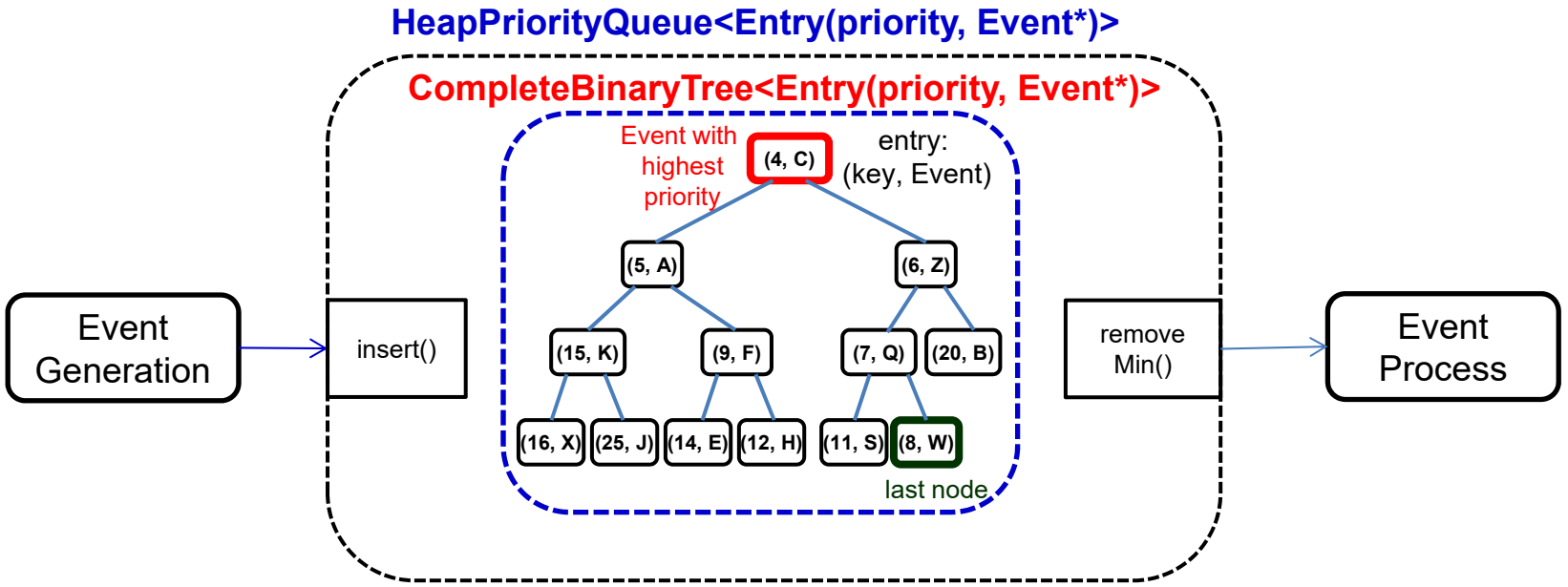
(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- **Heap Priority Queue**
- **Complete Binary Tree**
  - C++ implementation of a Complete Binary Tree with template array
- **Heap**
  - height of heap
  - insertion into a heap with up-heap bubbling
  - removal from a heap with down-heap bubbling
- **Priority-based Event Handling with Heap Priority Queue**

Priority-based Event Handling

◆ PriorityQueue based on Generic Complete BinaryTree



Event

class Event

```
/* Event.h (1) */

#ifndef EVENT_H
#define EVENT_H
#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>
using namespace std;

enum EventStatus { GENERATED, ENQUEUED, PROCESSED, UNDEFINED };
#define MAX_EVENT_PRIORITY 100
#define NUM_EVENT_GENERATORS 10

class Event
{
    friend ostream& operator<<(ostream& fout, const Event& e);
public:
    Event() { } // default constructor
    Event(int event_id, int event_pri, int srcAddr); //constructor
    void printEvent(ostream& fout);
    void setEventHandlerAddr(int evtHndlerAddr) { event_handler_addr = evtHndlerAddr; }
    void setEventGenAddr(int genAddr) { event_gen_addr = genAddr; }
```

```
/* Event.h (2) */

    void setEventNo(int evtNo) { event_no = evtNo; }
    void setEventPri(int pri) { event_pri = pri; }
    void setEventStatus(EventStatus evtStatus) { eventStatus = evtStatus; }
    int getEventPri() { return event_pri; }
    int getEventNo() { return event_no; }
    bool operator>(Event& e) { return (event_pri > e.event_pri); }
    bool operator<(Event& e) { return (event_pri < e.event_pri); }
private:
    int event_no;
    int event_gen_addr;
    int event_handler_addr;
    int event_pri;  // event_priority
    EventStatus eventStatus;
};

Event* genRandEvent(int evt_no);

#endif
```

```
/* Event.cpp (1) */
#include "Event.h"

Event::Event(int evt_no, int evt_pri, int evtGenAddr)
{
    event_no = evt_no;
    event_gen_addr = evtGenAddr;
    event_handler_addr = -1; // event handler is not defined at this moment
    event_pri = evt_pri;  // event_priority
    eventStatus = GENERATED;
}

Event* genRandEvent(int evt_no)
{
    Event *pEv;
    int evt_prio;
    int evt_generator_id;

    evt_prio = rand() % MAX_EVENT_PRIORITY;
    evt_generator_id = rand() % NUM_EVENT_GENERATORS;

    pEv = (Event *) new Event(evt_no, evt_prio, evt_generator_id);

    return pEv;
}
```

```
/* Event.cpp (2) */
#include "Event.h"

void Event::printEvent(ostream& fout)
{
    fout << "Event(pri:" << setw(3) << event_pri << ", gen:" << setw(3) << event_gen_addr;
    fout << ", no:" << setw(3) << event_no << ")";
}

ostream& operator<<(ostream& fout, const Event& evt)
{
    fout << "Event(pri:" << setw(3) << evt.event_pri << ", gen:" << setw(3) << evt.event_gen_addr;
    fout << ", no:" << setw(3) << evt.event_no << ")";

    return fout;
}
```

TA_Entry<K, V>

탐색 키를 포함하는 class T_Entry<K, V>

```
/* T_Entry.h (1) */
#ifndef T_ENTRY_H
#define T_ENTRY_H
#include <fstream>

template<typename K, typename V>
class T_Entry
{
    friend ostream& operator<<(ostream& fout, T_Entry<K, V>& entry)
    {
        fout << "[" << setw(2) << entry.getKey() << ", " << *(entry.getValue()) << "]";
        return fout;
    }
private:
    K _key;
    V _value;

public:
    T_Entry(K key, V value) { _key = key; _value = value; }
    T_Entry() {} // default constructor
    ~T_Entry() {}
    void setKey(const K& key) { _key = key; }
    void setValue(const V& value) { _value = value; }
    K getKey() const { return _key; }
    V getValue() const { return _value; }
```

```
/* T_Entry.h (2) */

    bool operator>(const T_Entry& right) { return (_key > right.getKey()); }
    bool operator>=(const T_Entry& right) { return (_key >= right.getKey()); }
    bool operator<(const T_Entry& right) { return (_key < right.getKey()); }
    bool operator<=(const T_Entry& right) { return (_key <= right.getKey()); }
    bool operator==(const T_Entry& right)
        { return ((_key == right.getKey()) && (_value == right.getValue())); }
    T_Entry& operator=(T_Entry& right);
    void fprint(ostream& fout);
}; // end of class T_Entry<K, V>

template<typename K, typename V>
T_Entry<K, V>& T_Entry<K, V>::operator=(T_Entry<K, V>& right)
{
    _key = right.getKey();
    _value = right.getValue();

    return *this;
}

template<typename K, typename V>
void T_Entry<K, V>::fprint(ostream& fout)
{
    fout << "[Key:" << setw(2) << this->getKey() << ", " << this->getValue() << "]";
}
#endif
```

T_Entry의 일반화 배열
class TA_Entry<K, V>

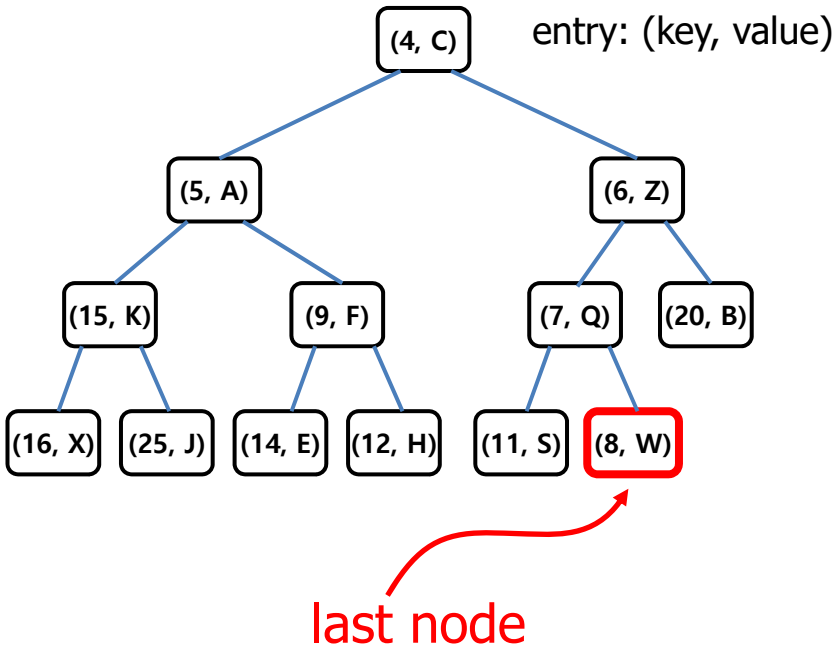
```
template<typename K, typename V>
class TA_Entry
{
public:
    TA_Entry(int n, string nm); // constructor
    ~TA_Entry(); // destructor
    int size() { return num_elements; }
    bool empty() { return num_elements == 0; }
    string getName() { return name; }
    void reserve(int new_capacity);
    void insert(int i, T_Entry<K, V> element);
    void remove(int i);
    T_Entry<K, V>& at(int i);
    void set(int i, T_Entry<K, V>& element);
    void fprint(ofstream &fout, int elements_per_line);
    void fprintSample(ofstream &fout, int elements_per_line, int num_sample_lines);
    bool isValidIndex(int i);
    T_Entry<K, V>& operator[](int index) { return t_array[index]; }
protected:
    T_Entry<K, V> *t_array;
    int num_elements;
    int capacity;
    string name;
};
```

Priority Queue

Heaps

- A heap is a binary tree storing keys at its nodes and satisfying the following properties:
- Heap-Order: for every internal node v other than the root, $key(v) \geq key(parent(v))$
- Complete Binary Tree: let h be the height of the heap
  - for $i = 0, \ldots, h-1$, there are 2^i nodes of depth i
  - at depth $h-1$, the internal nodes are to the left of the external nodes

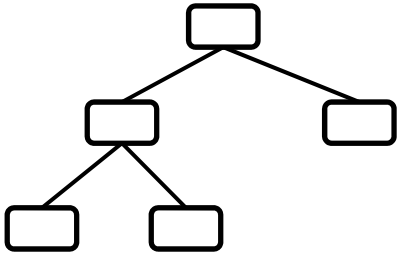
- The last node of a heap is the rightmost node of maximum depth



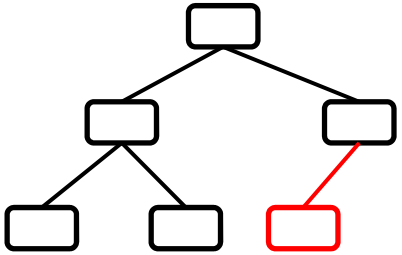
Complete Binary Tree

◆ Complete Binary Tree Property

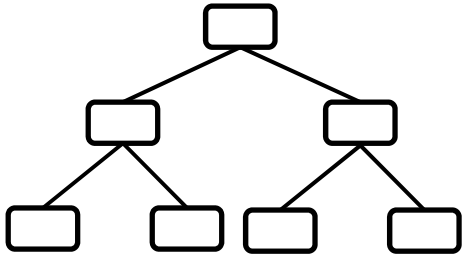
- a heap T with height h is a complete binary tree, that is, levels 0, 1, 2, …, h-1 of T have the maximum number of nodes possible (namely, level i has 2^i nodes, for $0 \le i \le h-1$) and nodes at level h fill this level from left to right



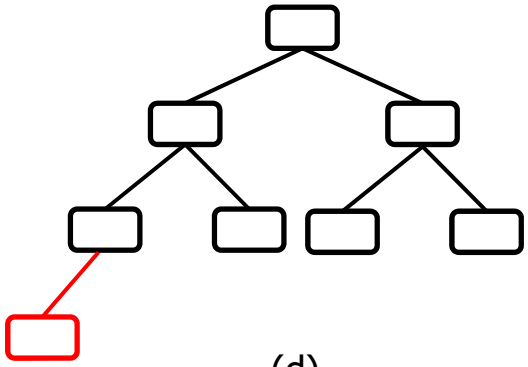
(a)



(b)



(c)



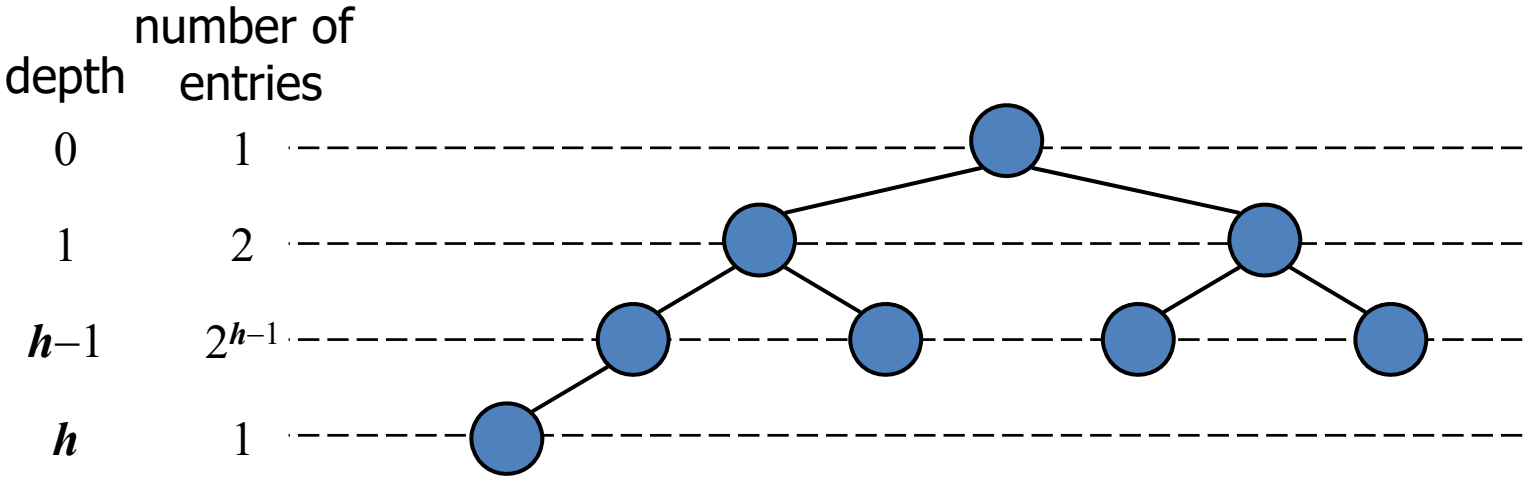
(d)

Height of a Heap

- **Theorem: A heap storing n keys has height $O(\log n)$**

  Proof: (we apply the complete binary tree property)

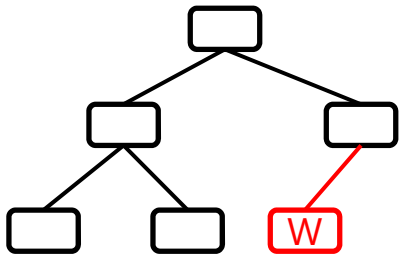
  - Let h be the height of a heap storing n keys
  - Since there are 2^i keys at depth $i = 0, \ldots, h-1$ and at least one key at depth h, we have $n \geq 1 + 2 + 4 + \ldots + 2^{h-1} + 1$
  - Thus, $n \geq 2^h$, i.e., $h \leq \log n$



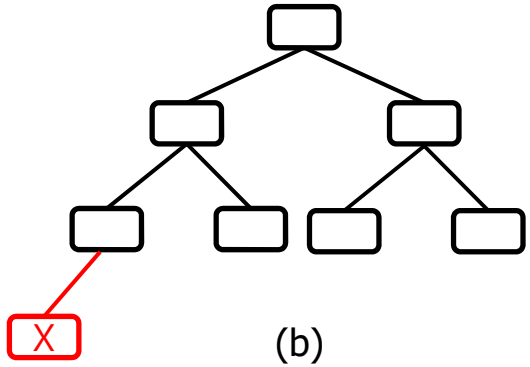
Array Representation of a Complete Binary Tree

◆ Array representation of a complete binary tree

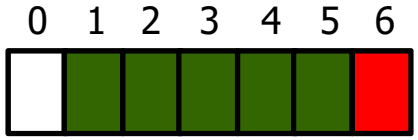
- if v is the root of CBT, then pos(v) = 1
- if lc is the left child of node u, then pos(lc) = 2 × pos(u)
- if rc is the right child of node u, then pos(rc) = 2 × pos(u) + 1



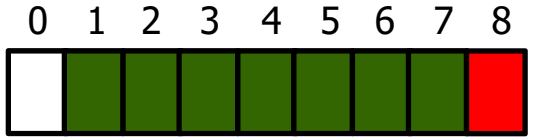
(a)



(b)



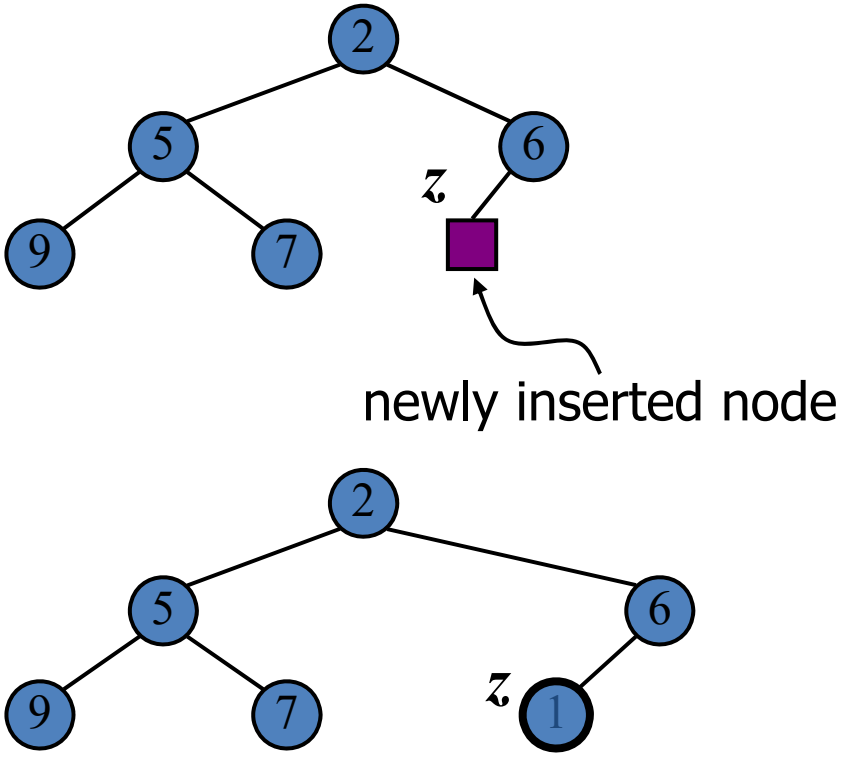
(c)



(d)

Insertion into a Heap

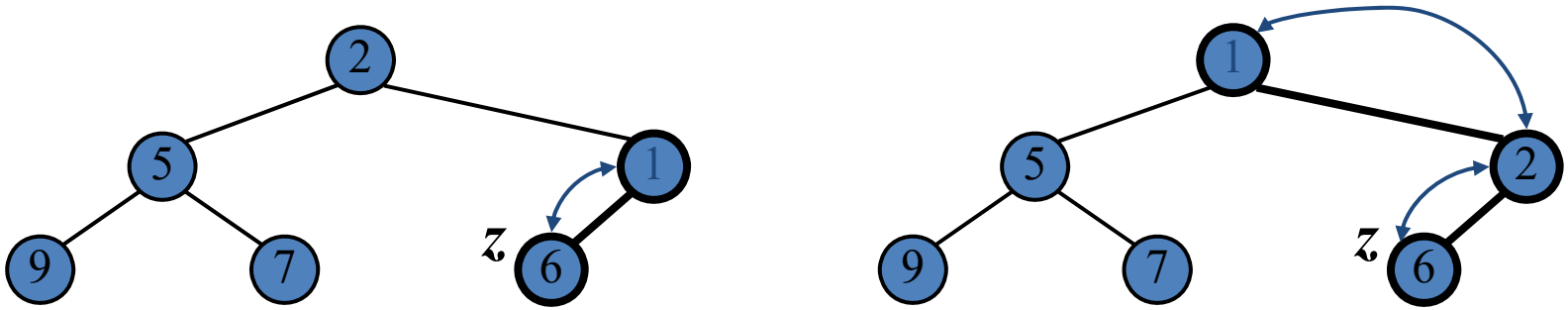
- **Method insertItem of the priority queue ADT corresponds to the insertion of a key k to the heap**
- **The insertion algorithm consists of three steps**
  - Find the insertion node z (the new last node)
  - Store k at z
  - Restore the heap-order property (discussed next)



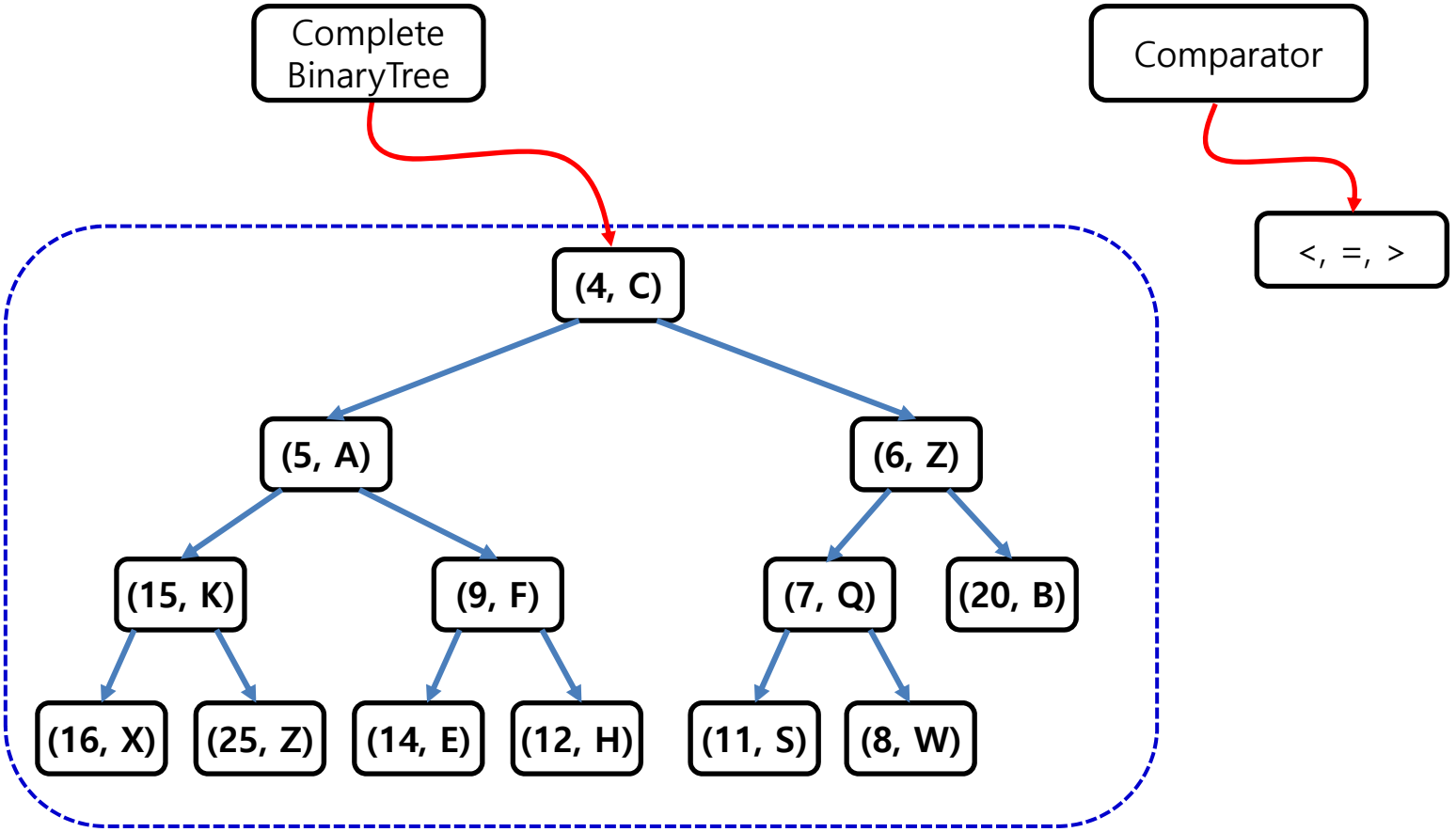
Upheap

◆ Upheap

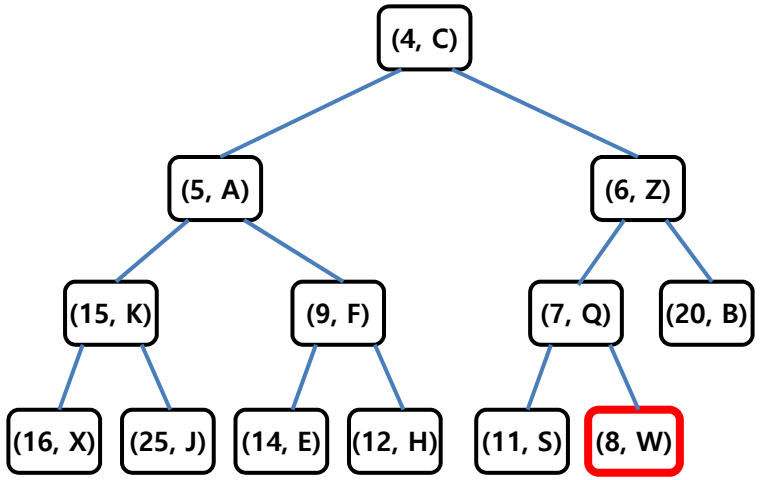
- After the insertion of a new key k, the heap-order property may be violated
- Algorithm upheap restores the heap-order property by swapping k along an upward path from the insertion node
- Upheap terminates when the key k reaches the root or a node whose parent has a key smaller than or equal to k
- Since a heap has height $O(\log n)$, upheap runs in $O(\log n)$ time



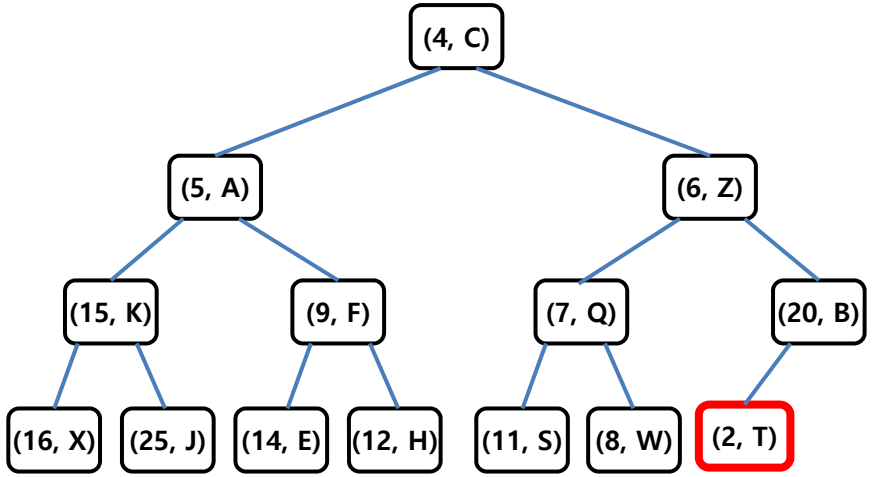
◆ Insertion with up-heap bubbling (1)



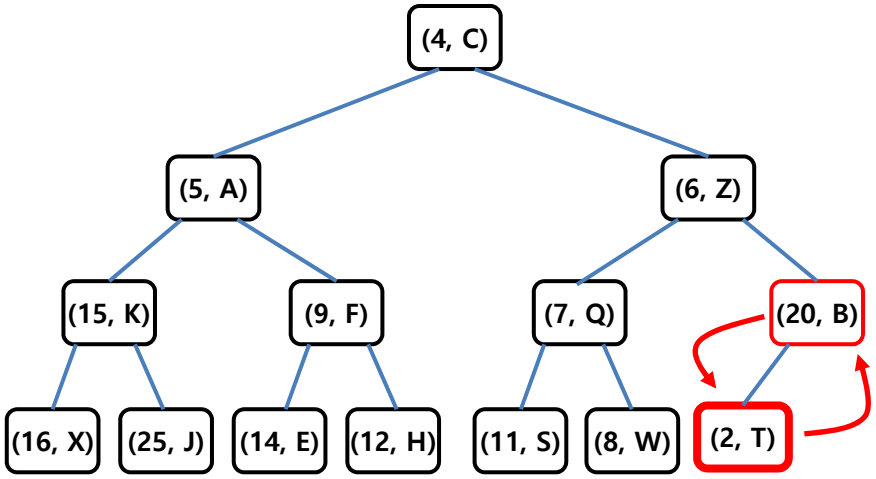
◆ Insertion with up-heap bubbling (2)



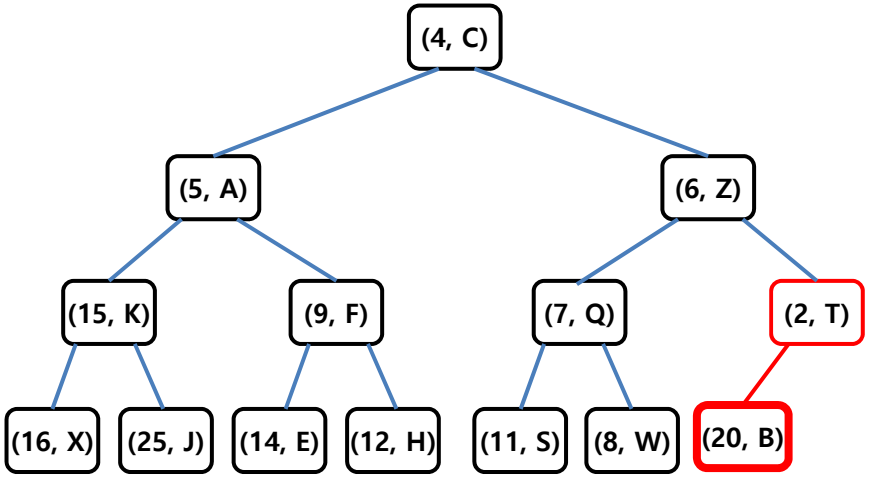
(a) last status



(b) insert a new node

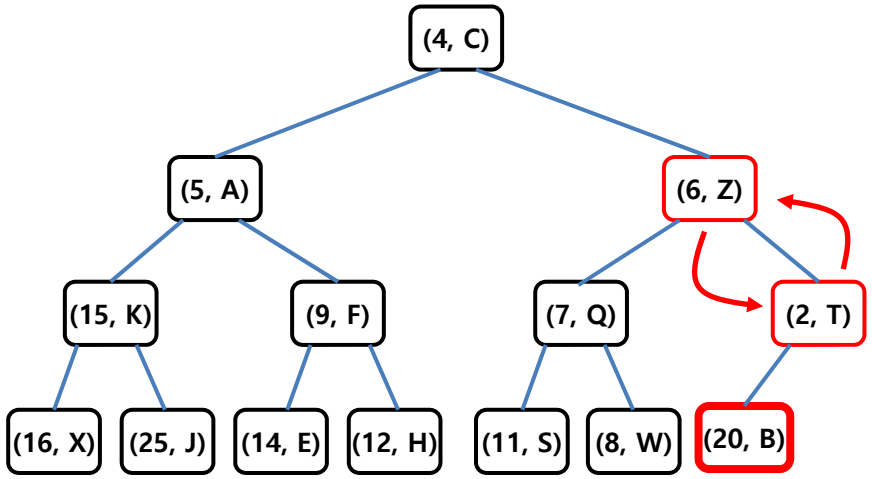


(c) swapping

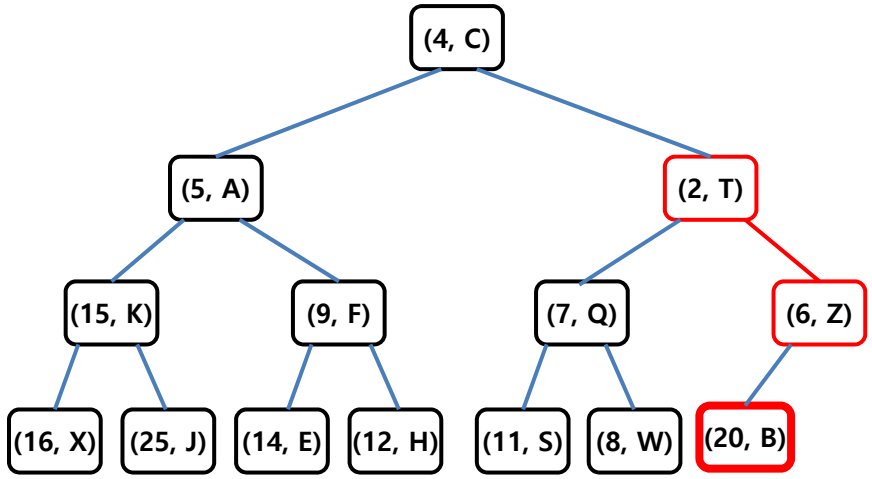


(d) after swapping

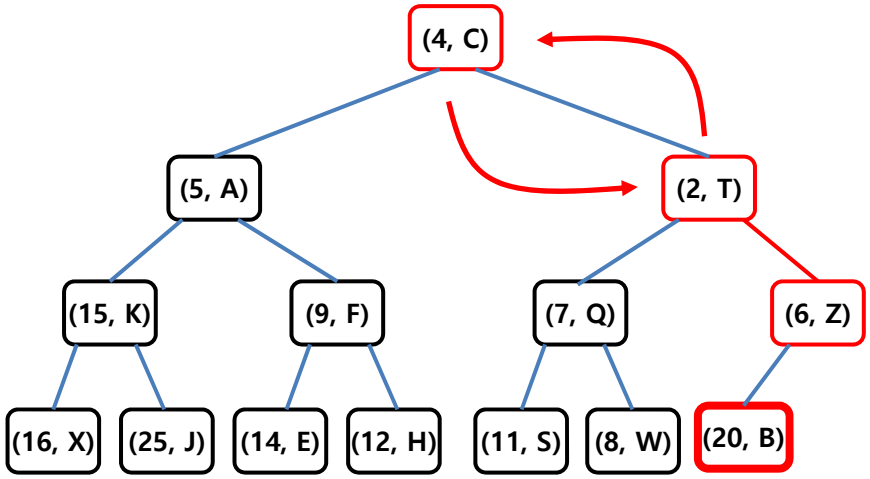
◆ Insertion with up-heap bubbling (3)



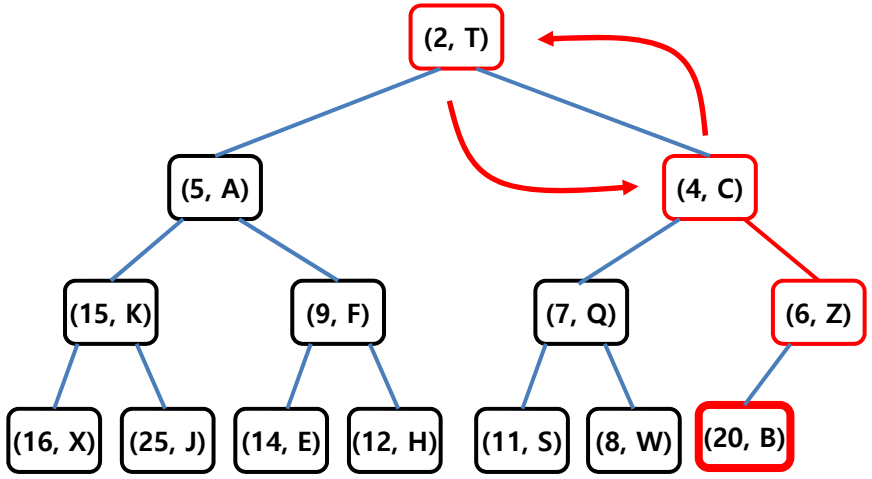
(e) swapping



(e) after swapping



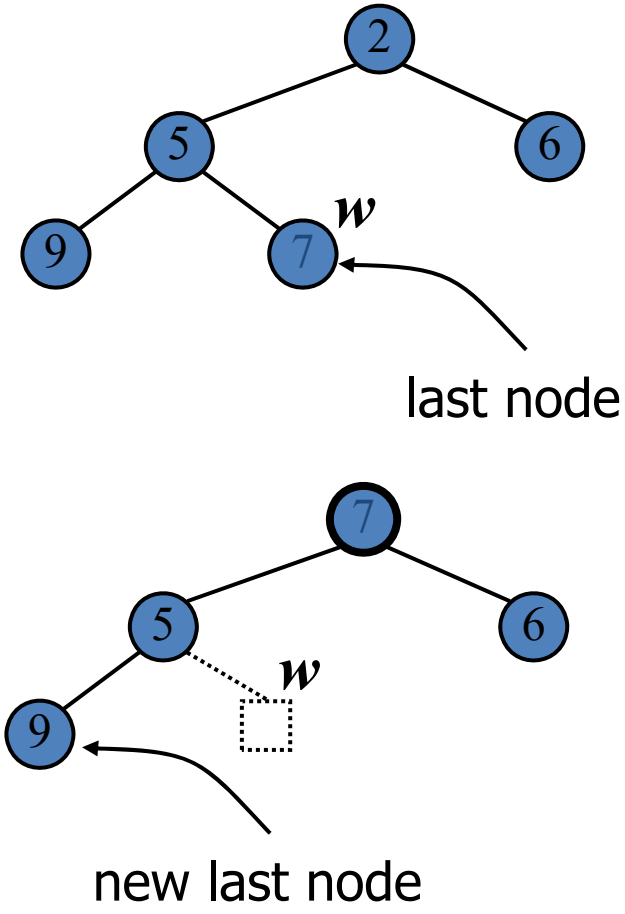
(e) swapping



(e) after swapping

Removal from a Heap

- **Method removeMin of the priority queue ADT corresponds to the removal of the root key from the heap**
- **The removal algorithm consists of three steps**
  - Replace the root key with the key of the last node *w*
  - Remove *w*
  - Restore the heap-order property (discussed next)

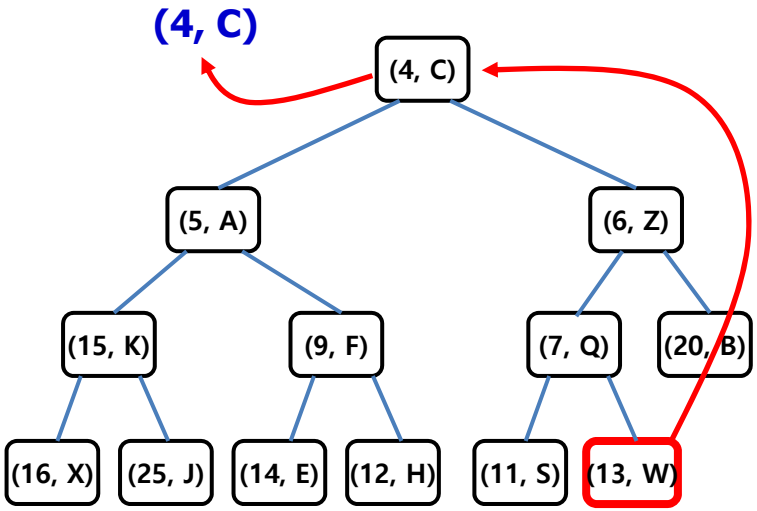


Downheap

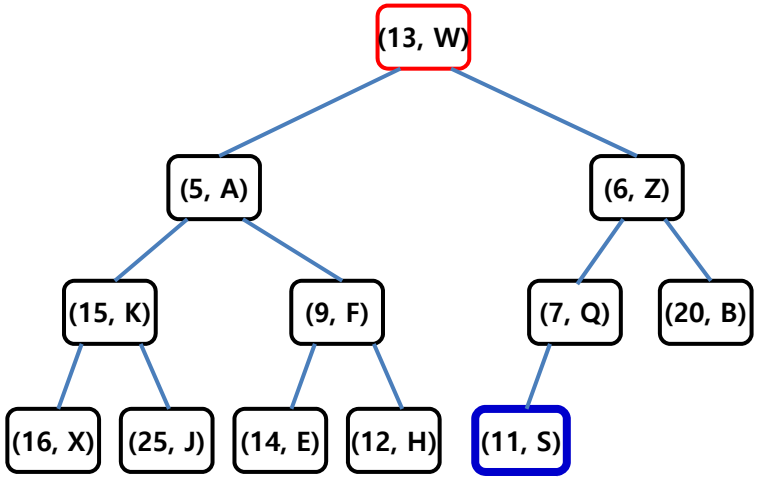
◆ Downheap

- After replacing the root key with the key k of the last node, the heap-order property may be violated
- Algorithm downheap restores the heap-order property by swapping key k along a downward path from the root
- Downheap terminates when key k reaches a leaf or a node whose children have keys greater than or equal to k
- Since a heap has height $O(\log n)$, downheap runs in $O(\log n)$ time

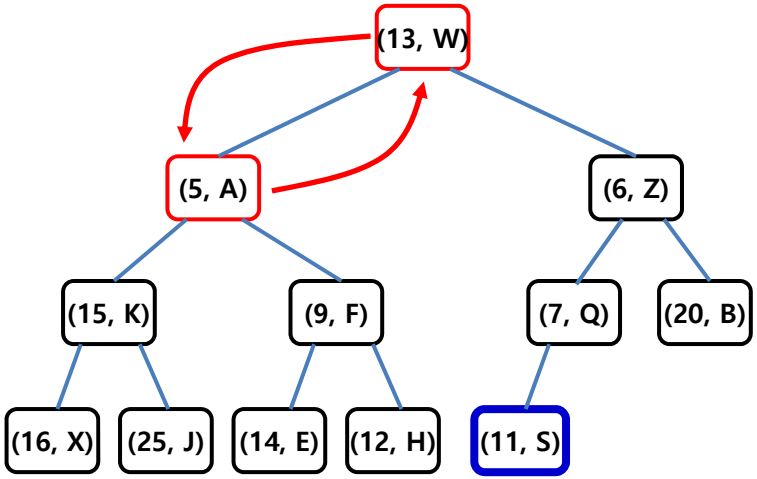
◆ Down-heap Bubbling after a Removal (1)



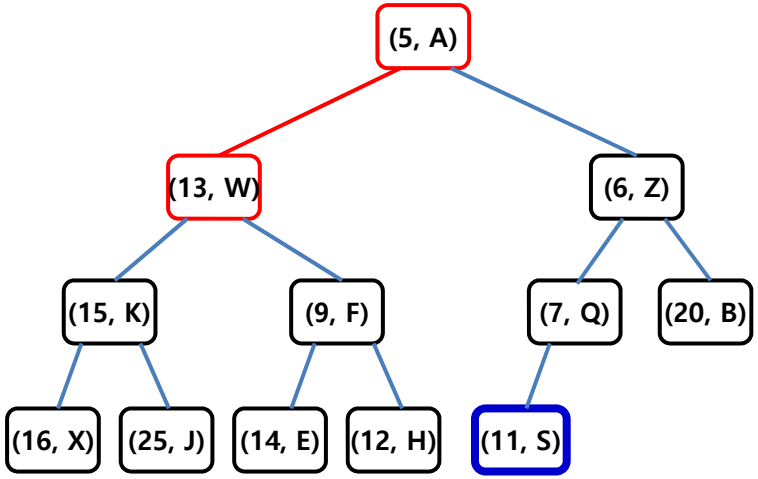
(a) last status



(b) after changing root by the last node

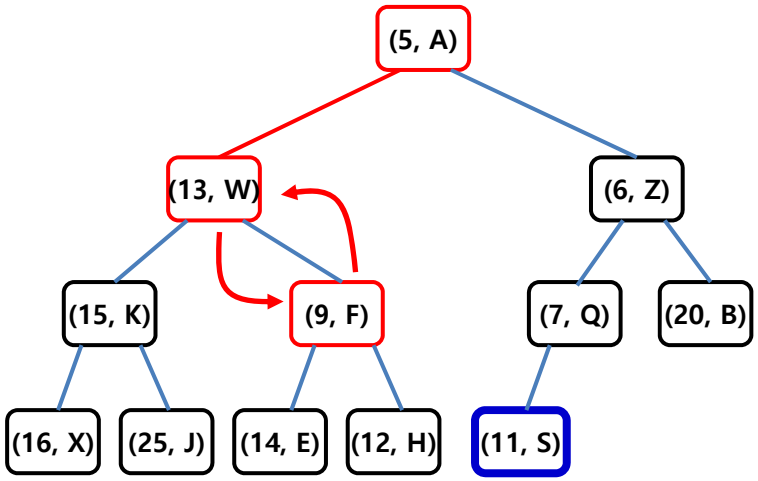


(c) compare and swap

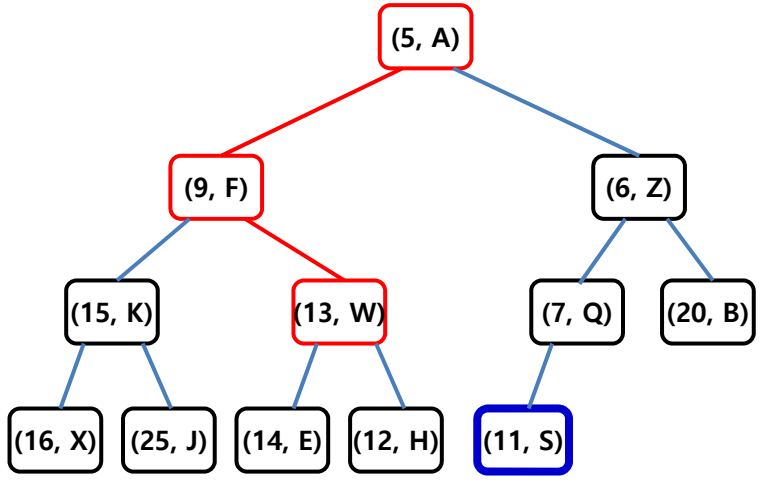


(d) after swapping

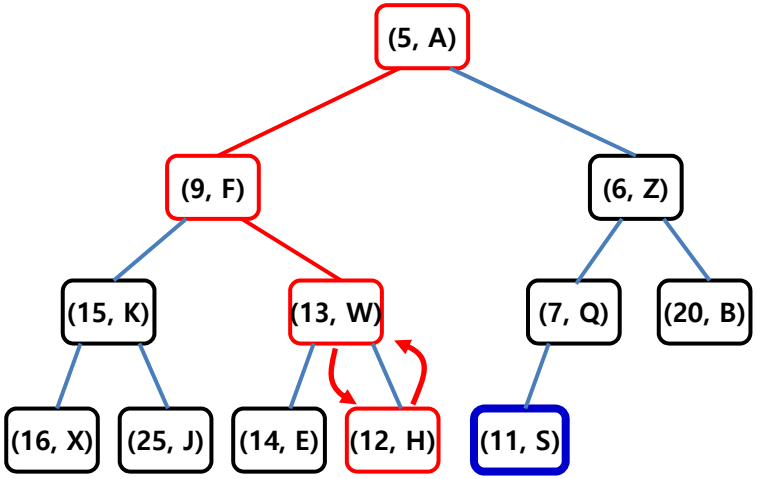
◆ Down-heap Bubbling after a Removal (2)



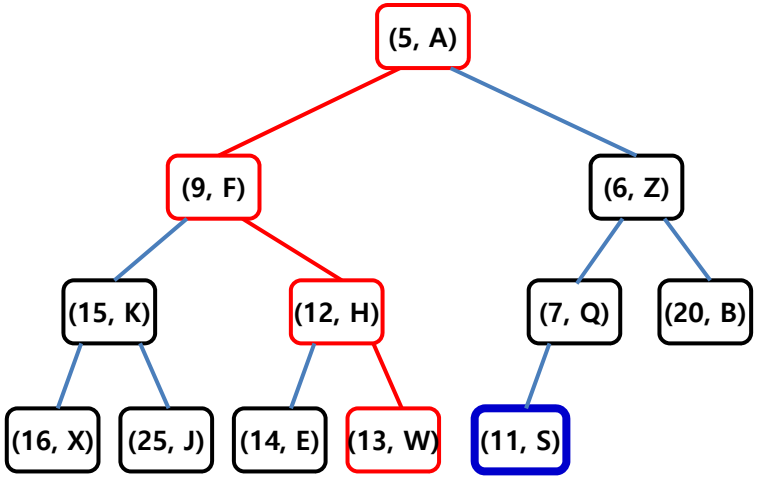
(e) compare and swap



(f) after swapping



(g) compare and swap

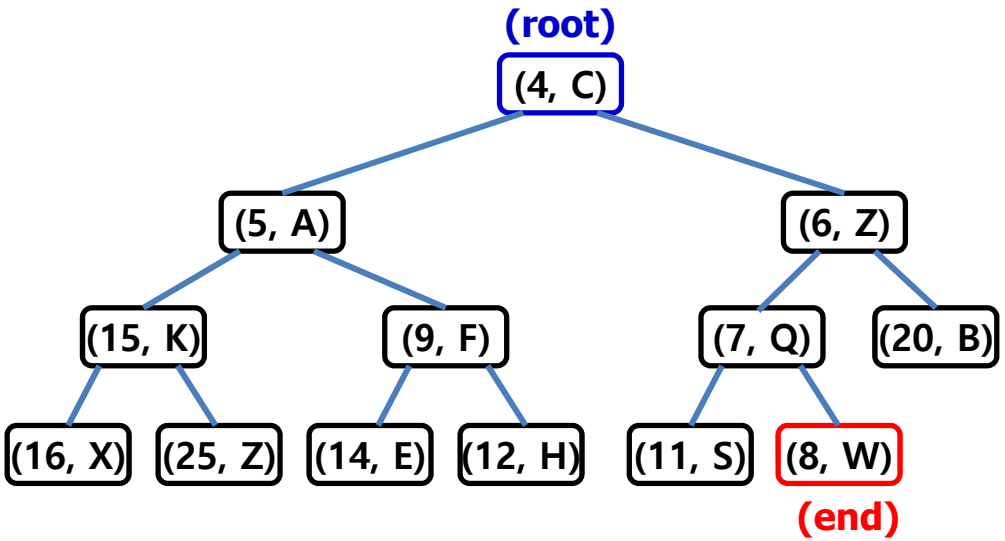


(h) after swapping

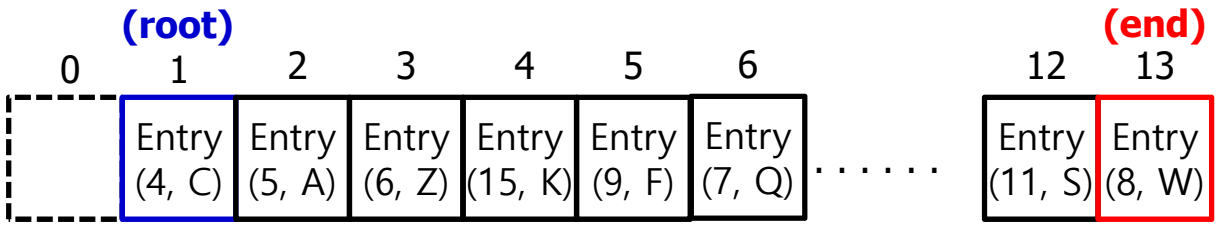
Priority-based Event Handling

Complete Binary Tree

◆ Template Complete Binary Tree of T_Entry<K, V>



(a) Example Complete Binary Tree



(b) Implementation of a Complete Binary Tree with TA_Entry

class CompleteBinaryTree

```
/* CompleteBinaryTree.h (1) */
#ifndef COMPLETE_BINARY_TREE_H
#define COMPLETE_BINARY_TREE_H
#include "TA_Entry.h"
#include "T_Entry.h"
#define CBT_ROOT 1

template<typename K, typename V>
class CompleteBinaryTree : public TA_Entry<K, V>
{
public:
    CompleteBinaryTree(int capa, string nm);
    int add_at_end(T_Entry<K, V>& elem);
    T_Entry<K, V>& getEndElement() { return t_array[end]; }
    T_Entry<K, V>& getRootElement() { return t_array[CBT_ROOT]; }
    int getEndIndex() { return end; }
    void removeCBTEnd();
    void fprintCBT(ofstream &fout);
    void fprintCBT_byLevel(ofstream &fout);
protected:
    void _fprintCBT_byLevel(ofstream &fout, int p, int level);
    int parentIndex(int index) { return index / 2; }
    int leftChildIndex(int index) { return index * 2; }
    int rightChildIndex(int index) { return (index * 2 + 1); }
    bool hasLeftChild(int index) { return ((index * 2) <= end); }
    bool hasRightChild(int index) { return ((index * 2 + 1) <= end); }
    int end;
};
```

```
/* CompleteBinaryTree.h (2) */

template<typename K, typename V>
CompleteBinaryTree<K, V>::CompleteBinaryTree(int capa, string nm)
   :TA_Entry<K, V>(capa, nm)
{
    end = 0; // reset to empty
}

template<typename K, typename V>
void CompleteBinaryTree<K, V>::fprintCBT(ofstream &fout)
{
    if (end <= 0)
    {
       fout << this->getName() << " is empty now !!" << endl;
       return;
    }
    int count = 0;
    for (int i = 1; i <= end; i++)
    {
        fout << setw(3) << t_array[i] << endl;
        //if ((((count + 1) % 10) == 0) && (i != end))
        //fout << endl;
        count++;
    }
}
```

```
/* CompleteBinaryTree.h (3) */

template<typename K, typename V>
void CompleteBinaryTree<K, V>::_fprintCBT_byLevel(ofstream &fout, int index, int level)
{
    int index_child;
    if (hasRightChild(index))
    {
        index_child = rightChildIndex(index);
        _printCBT_byLevel(fout, index_child, level + 1);
    }

    for (int i = 0; i<level; i++)
       fout << "     ";
    t_array[index].fprint(fout);
    fout << endl;

    if (hasLeftChild(index))
    {
        index_child = leftChildIndex(index);
        _printCBT_byLevel(fout, index_child, level + 1);
    }
}

template<typename K, typename V>
void CompleteBinaryTree<K, V>::fprintCBT_byLevel(ofstream &fout)
{
    if (end <= 0)
    {
        fout << "CBT is EMPTY now !!" << endl;
        return;
    }
    _printCBT_byLevel(fout, CBT_ROOT, 0);
}
```

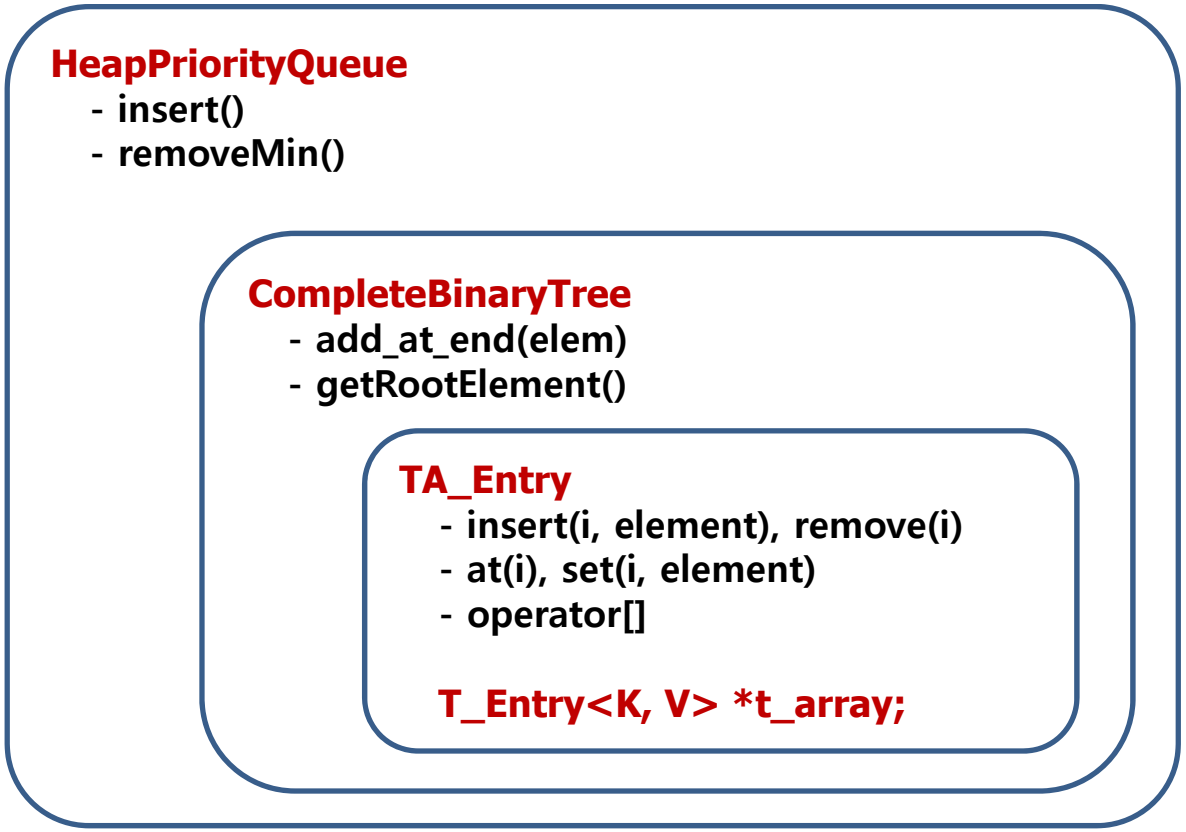
```
Final status of insertions :
                3
          2
                10
     1
                9
          4
                13
0
                7
          6
                12
     5
                11
          8
                14
```

```
/* CompleteBinaryTree.h (4) */

template<typename K, typename V>
int CompleteBinaryTree<K, V>::add_at_end(T_Entry<K, V>& elem)
{
    if (end >= capacity)
    {
       cout << this->getName() << " is FULL now !!" << endl;
       return end;
    }
    end++;
    t_array[end] = elem;

    return end;
}

template<typename K, typename V>
void CompleteBinaryTree<K, V>::removeCBTEnd()
{
    end--;
    num_elements--;
}
#endif
```

CompleteBinaryTree 기반 Heap Priority Queue 구현

◆ Heap Priority Queue 구조

- class HeapPrioQueue는 class CompleteBinaryTree를 상속
- class CompleteBinaryTree는 class TA_Entry<K, V>를 상속



class HeapPrioQ

```
/* HeapPrioQ.h (1) */

#ifndef HEAP_PRIO_QUEUE_H
#define HEAP_PRIO_QUEUE_H
#include "CompleteBinaryTree.h"

template<typename K, typename V>
class HeapPrioQueue : public CompleteBinaryTree<K, V>
{
public:
      HeapPrioQueue(int capa, string nm);
      ~HeapPrioQueue();
      bool isEmpty() { return size() == 0; }
      bool isFull() { return size() == capacity; }
      int insert(T_Entry<K, V>& elem);
      T_Entry<K, V>* removeHeapMin();
      T_Entry<K, V>* getHeapMin();
      void fprint(ofstream &fout);
      int size() {return end; }
private:
};
```

```
/* HeapPrioQ.h (2) */

template<typename K, typename V>
HeapPrioQueue<K, V>::HeapPrioQueue(int capa, string nm)
:CompleteBinaryTree(capa, nm)
{ }

template<typename K, typename V>
HeapPrioQueue<K, V>::~HeapPrioQueue()
{ }

template<typename K, typename V>
void HeapPrioQueue<K, V>::fprint(ofstream &fout)
{
    if (size() <= 0)
    {
       fout << "HeapPriorityQueue is Empty !!" << endl;
       return;
    }
    else
       CompleteBinaryTree::printCBT(fout);
}
```

```
/* HeapPrioQ.h (3) */

template<typename K, typename V>
int HeapPrioQueue<K, V>::insert(T_Entry<K, V>& elem)
{
    int index, parent_index;
    T_Entry<K, V> temp;
    if (isFull())
    {
        cout << this->getName() << " is Full !!" << endl;
        return size();
    }
    index = add_at_end(elem);

    /* up-heap bubbling */
    while (index != CBT_ROOT)
    {
        parent_index = parentIndex(index);
        if (t_array[index].getKey() > t_array[parent_index].getKey())
            break;
        else
        {
            temp = t_array[index];
            t_array[index] = t_array[parent_index];
            t_array[parent_index] = temp;
            index = parent_index;
        }
    }
    return size();
}
```

```
/* HeapPrioQ.h (4) */

template<typename K, typename V>
T_Entry<K, V>* HeapPrioQueue<K, V>::getHeapMin()
{
    T_Entry<K, V>* pMinElem;
    if (size() <= 0)
    {
       return NULL;
    }
    pMinElem = (T_Entry<K, V>*) new T_Entry<K, V>;
    *pMinElem = getRootElement();
    return pMinElem;

}
```

```
/* HeapPrioQ.h (4) */

template<typename K, typename V>
T_Entry<K, V>* HeapPrioQueue<K, V>::removeHeapMin()
{
    int index_p, index_c, index_rc;
    T_Entry<K, V> *pMinElem;
    T_Entry<K, V> temp, t_p, t_c;
    int HPQ_size = size();

    if (HPQ_size <= 0)
    {
       return NULL;
    }
    pMinElem = (T_Entry<K, V>*) new T_Entry<K, V>;
    *pMinElem = getRootElement();
    if (HPQ_size == 1)
    {
       removeCBTEnd();
    }
    else
    {
        index_p = CBT_ROOT;
        t_array[CBT_ROOT] = t_array[end];
        end--;
```

```
/* HeapPrioQ.h (5) */

        /* down-heap bubbling */
        while (hasLeftChild(index_p))
        {
            index_c = leftChildIndex(index_p);
            index_rc = rightChildIndex(index_p);
            if (hasRightChild(index_p) && (t_array[index_c] > t_array[index_rc]))
                index_c = index_rc;
            t_p = t_array[index_p];
            t_c = t_array[index_c];
            if (t_p > t_c)
            {
                //swap(index_u, index_c);
                temp = t_array[index_p];
                t_array[index_p] = t_array[index_c];
                t_array[index_c] = temp;
                index_p = index_c;
            }
            else
                 break;

        } // end while
    }
    return pMinElem;
}
#endif
```

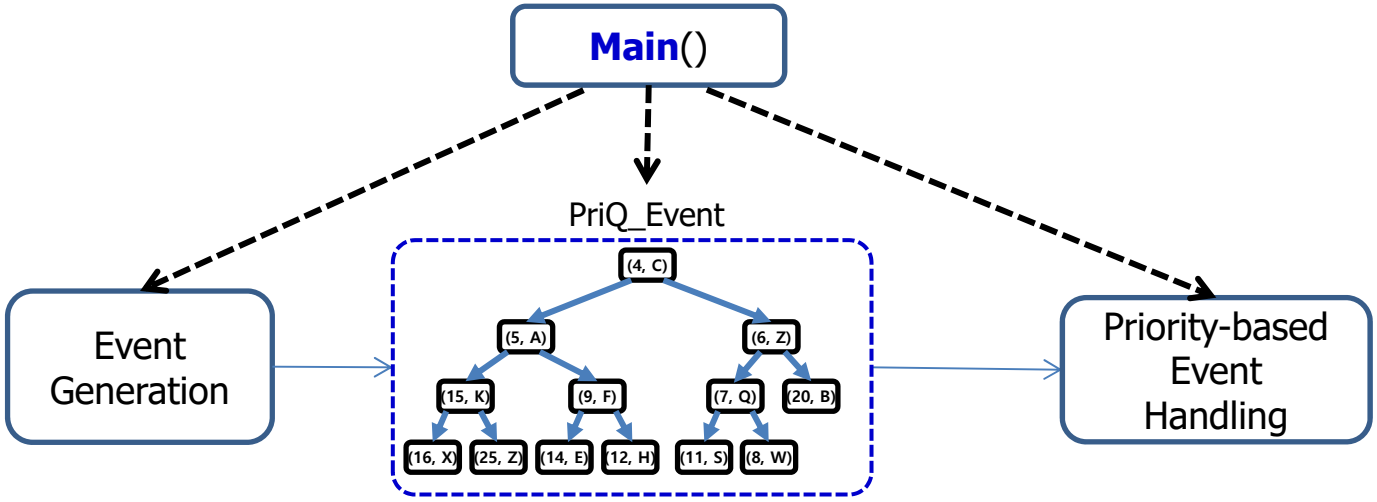
Priority Queue의 응용 예제

◆ Simple Simulation of Priority-based Event Handling

- Event Generation
- Event Handling

● Shared Priority Queue

- PriQ_Event



```
/* main() for Heap Priority Queue based on Complete Binary Tree (1) */
#include <iostream>
#include <fstream>
#include "Event.h"
#include "HeapPrioQ.h"
#include <string>
#include <stdlib.h>
using namespace std;

#define INITIAL_CBT_CAPA 100
#define TEST_HEAP_PRIO_Q_EVENT
#define NUM_EVENTS 15

void main()
{
	ofstream fout;
	string tName = "";
	char tmp[10];
	int priority = -1;
	int current_top_priority;
	int duration = 0;
	int size;
	int *pE;

	fout.open("output.txt");
	if (fout.fail())
	{
	   cout << "Fail to open output.txt file for results !!" << endl;
	   exit;
	}
```

```
/* main() for Heap Priority Queue based on Complete Binary Tree (2) */

    Event events[NUM_EVENTS] =
    {
       //Event(int evt_no, int evt_pri, string title, int gen_addr)
       Event(0, 14, "evt_00", 0), Event(1, 13, "evt_01", 1), Event(2, 12, "evt_02", 2),
       Event(3, 11, "evt_03", 3), Event(4, 10, "evt_04", 4), Event(5, 9, "evt_05", 5),
       Event(6, 8, "evt_06", 6), Event(7, 7, "evt_07", 7), Event(8, 6, "evt_08", 8),
       Event(9, 5, "evt_09", 9), Event(10, 4, "evt_10", 10), Event(11, 3, "evt_11", 11),
       Event(12, 2, "evt_12", 12), Event(13, 1, "evt_13", 13), Event(14, 0, "evt_14", 14)
    };

    HeapPrioQueue<int, Event*> HeapPriQ_Event(INITIAL_CBT_CAPA,
    string("Event_Heap_Priority_Queue"));
    Event *pEv;
    T_Entry<int, Event*> entry_event, *pEntry_Event;

    for (int i = 0; i<NUM_EVENTS; i++)
    {
        entry_event.setKey(events[i].getEventPri());
        entry_event.setValue(&events[i]);
        HeapPriQ_Event.insert(entry_event);
        fout << "Insert " << events[i];
        fout << " ==> Size of Heap Priority Queue : " << setw(3) << HeapPriQ_Event.size() << endl;
    }

    fout << "Final status of insertions : " << endl;
    HeapPriQ_Event.fprintCBT_byLevel(fout);
```

```
/* main() for Heap Priority Queue based on Complete Binary Tree (2) */

    for (int i = 0; i<NUM_EVENTS; i++)
    {
        fout << "₩nCurrent top priority in Heap Priority Queue : ";
        pEntry_Event = HeapPriQ_Event.getHeapMin();
        fout << *pEntry_Event << endl;
        pEntry_Event = HeapPriQ_Event.removeHeapMin();
        fout << "Remove " << *pEntry_Event;
        fout << " ==> " << HeapPriQ_Event.size() << " elements remains." << endl;
        HeapPriQ_Event.fprintCBT_byLevel(fout);
        fout << endl;
    }

    fout.close();
} // end main();
```

◆ 실행 결과 (1)

```
Insert Event(no:  0; pri: 14; gen:  0, title: evt_00) ==> Size of Heap Priority Queue :  1
Insert Event(no:  1; pri: 13; gen:  1, title: evt_01) ==> Size of Heap Priority Queue :  2
Insert Event(no:  2; pri: 12; gen:  2, title: evt_02) ==> Size of Heap Priority Queue :  3
Insert Event(no:  3; pri: 11; gen:  3, title: evt_03) ==> Size of Heap Priority Queue :  4
Insert Event(no:  4; pri: 10; gen:  4, title: evt_04) ==> Size of Heap Priority Queue :  5
Insert Event(no:  5; pri:  9; gen:  5, title: evt_05) ==> Size of Heap Priority Queue :  6
Insert Event(no:  6; pri:  8; gen:  6, title: evt_06) ==> Size of Heap Priority Queue :  7
Insert Event(no:  7; pri:  7; gen:  7, title: evt_07) ==> Size of Heap Priority Queue :  8
Insert Event(no:  8; pri:  6; gen:  8, title: evt_08) ==> Size of Heap Priority Queue :  9
Insert Event(no:  9; pri:  5; gen:  9, title: evt_09) ==> Size of Heap Priority Queue : 10
Insert Event(no: 10; pri:  4; gen: 10, title: evt_10) ==> Size of Heap Priority Queue : 11
Insert Event(no: 11; pri:  3; gen: 11, title: evt_11) ==> Size of Heap Priority Queue : 12
Insert Event(no: 12; pri:  2; gen: 12, title: evt_12) ==> Size of Heap Priority Queue : 13
Insert Event(no: 13; pri:  1; gen: 13, title: evt_13) ==> Size of Heap Priority Queue : 14
Insert Event(no: 14; pri:  0; gen: 14, title: evt_14) ==> Size of Heap Priority Queue : 15
Final status of insertions :
                [Key: 3, Event(no: 11; pri:  3; gen: 11, title: evt_11)]
           [Key: 2, Event(no: 12; pri:  2; gen: 12, title: evt_12)]
                [Key:10, Event(no:  4; pri: 10; gen:  4, title: evt_04)]
      [Key: 1, Event(no: 13; pri:  1; gen: 13, title: evt_13)]
                [Key: 9, Event(no:  5; pri:  9; gen:  5, title: evt_05)]
           [Key: 4, Event(no: 10; pri:  4; gen: 10, title: evt_10)]
                [Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
[Key: 0, Event(no: 14; pri:  0; gen: 14, title: evt_14)]
                [Key: 7, Event(no:  7; pri:  7; gen:  7, title: evt_07)]
           [Key: 6, Event(no:  8; pri:  6; gen:  8, title: evt_08)]
                [Key:12, Event(no:  2; pri: 12; gen:  2, title: evt_02)]
      [Key: 5, Event(no:  9; pri:  5; gen:  9, title: evt_09)]
                [Key:11, Event(no:  3; pri: 11; gen:  3, title: evt_03)]
           [Key: 8, Event(no:  6; pri:  8; gen:  6, title: evt_06)]
                [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]
```

◆ 실행 결과 (2)

```
Current top priority in Heap Priority Queue : [ 0, Event(no: 14; pri:  0; gen: 14, title: evt_14)]
Remove [ 0, Event(no: 14; pri:  0; gen: 14, title: evt_14)] ==> 14 elements remains.
          [Key: 3, Event(no: 11; pri:  3; gen: 11, title: evt_11)]
               [Key:10, Event(no:  4; pri: 10; gen:  4, title: evt_04)]
     [Key: 2, Event(no: 12; pri:  2; gen: 12, title: evt_12)]
               [Key: 9, Event(no:  5; pri:  9; gen:  5, title: evt_05)]
          [Key: 4, Event(no: 10; pri:  4; gen: 10, title: evt_10)]
               [Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
[Key: 1, Event(no: 13; pri:  1; gen: 13, title: evt_13)]
               [Key: 7, Event(no:  7; pri:  7; gen:  7, title: evt_07)]
          [Key: 6, Event(no:  8; pri:  6; gen:  8, title: evt_08)]
               [Key:12, Event(no:  2; pri: 12; gen:  2, title: evt_02)]
     [Key: 5, Event(no:  9; pri:  5; gen:  9, title: evt_09)]
               [Key:11, Event(no:  3; pri: 11; gen:  3, title: evt_03)]
          [Key: 8, Event(no:  6; pri:  8; gen:  6, title: evt_06)]
               [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]


Current top priority in Heap Priority Queue : [ 1, Event(no: 13; pri:  1; gen: 13, title: evt_13)]
Remove [ 1, Event(no: 13; pri:  1; gen: 13, title: evt_13)] ==> 13 elements remains.
          [Key:10, Event(no:  4; pri: 10; gen:  4, title: evt_04)]
     [Key: 3, Event(no: 11; pri:  3; gen: 11, title: evt_11)]
               [Key: 9, Event(no:  5; pri:  9; gen:  5, title: evt_05)]
          [Key: 4, Event(no: 10; pri:  4; gen: 10, title: evt_10)]
               [Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
[Key: 2, Event(no: 12; pri:  2; gen: 12, title: evt_12)]
               [Key: 7, Event(no:  7; pri:  7; gen:  7, title: evt_07)]
          [Key: 6, Event(no:  8; pri:  6; gen:  8, title: evt_08)]
               [Key:12, Event(no:  2; pri: 12; gen:  2, title: evt_02)]
     [Key: 5, Event(no:  9; pri:  5; gen:  9, title: evt_09)]
               [Key:11, Event(no:  3; pri: 11; gen:  3, title: evt_03)]
          [Key: 8, Event(no:  6; pri:  8; gen:  6, title: evt_06)]
               [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]
```

◆ 실행 결과 (3)

```
Current top priority in Heap Priority Queue : [10, Event(no:  4; pri: 10; gen:  4, title: evt_04)]
Remove [10, Event(no:  4; pri: 10; gen:  4, title: evt_04)] ==> 4 elements remains.
     [Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
[Key:11, Event(no:  3; pri: 11; gen:  3, title: evt_03)]
     [Key:12, Event(no:  2; pri: 12; gen:  2, title: evt_02)]
          [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]


Current top priority in Heap Priority Queue : [11, Event(no:  3; pri: 11; gen:  3, title: evt_03)]
Remove [11, Event(no:  3; pri: 11; gen:  3, title: evt_03)] ==> 3 elements remains.
     [Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
[Key:12, Event(no:  2; pri: 12; gen:  2, title: evt_02)]
     [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]


Current top priority in Heap Priority Queue : [12, Event(no:  2; pri: 12; gen:  2, title: evt_02)]
Remove [12, Event(no:  2; pri: 12; gen:  2, title: evt_02)] ==> 2 elements remains.
[Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
     [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]


Current top priority in Heap Priority Queue : [13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
Remove [13, Event(no:  1; pri: 13; gen:  1, title: evt_01)] ==> 1 elements remains.
[Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]


Current top priority in Heap Priority Queue : [14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]
Remove [14, Event(no:  0; pri: 14; gen:  0, title: evt_00)] ==> 0 elements remains.
CBT is EMPTY now !!
```

Oral Test 8

(1) 완전 이진 트리 (complete binary tree)와 이진 탐색 트리의 차이점에 대하여 세부 항목별 대조표를 만들어 설명하라.

(2) 힙 우선 순위 큐 (heap priority queue)에 새로운 항목이 추가하기 위한 insert() 함수의 세부 동작을 pseudo code으로 표현하고, 상세하게 설명하라.

(3) 힙 우선 순위 큐 (heap priority queue)의 포함된 항목 중 가장 우선 순위가 높은 항목을 추출하는 removeMin() 함수의 세부 동작을 pseudo code으로 표현하고, 상세하게 설명하라.

(4) STL (Standard Template Library)에서 제공되는 Iterator는 무엇이며, Circular Queue를 위한 Iterator인 class CirQ_Iterator는 어떻게 구현할 수 있는가에 대하여 pseudo code를 사용하여 설명하라