SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

SW문제해결응용

<알림>

본 강의는 삼성 청년 SW아카데미의 컨텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Day6. Priority Queue

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Priority Queue

우선순위 큐

- 우선순위에 따라서 데이터를 관리하는 구조이다.

다양한 알고리즘에서 활용된다.

시뮬레이션
다익스트라
MST
등등

우선순위 큐를 사용하면 내부적으로 정렬할 수 있다

- 가장 큰 값이 top에 오게끔 할 수 있다. (default, MAX Heap)
- 내부적으로 “Heap 트리” 자료 구조를 사용하고 있다.
- .top() : 가장 우선 순위 높은 요소 추출

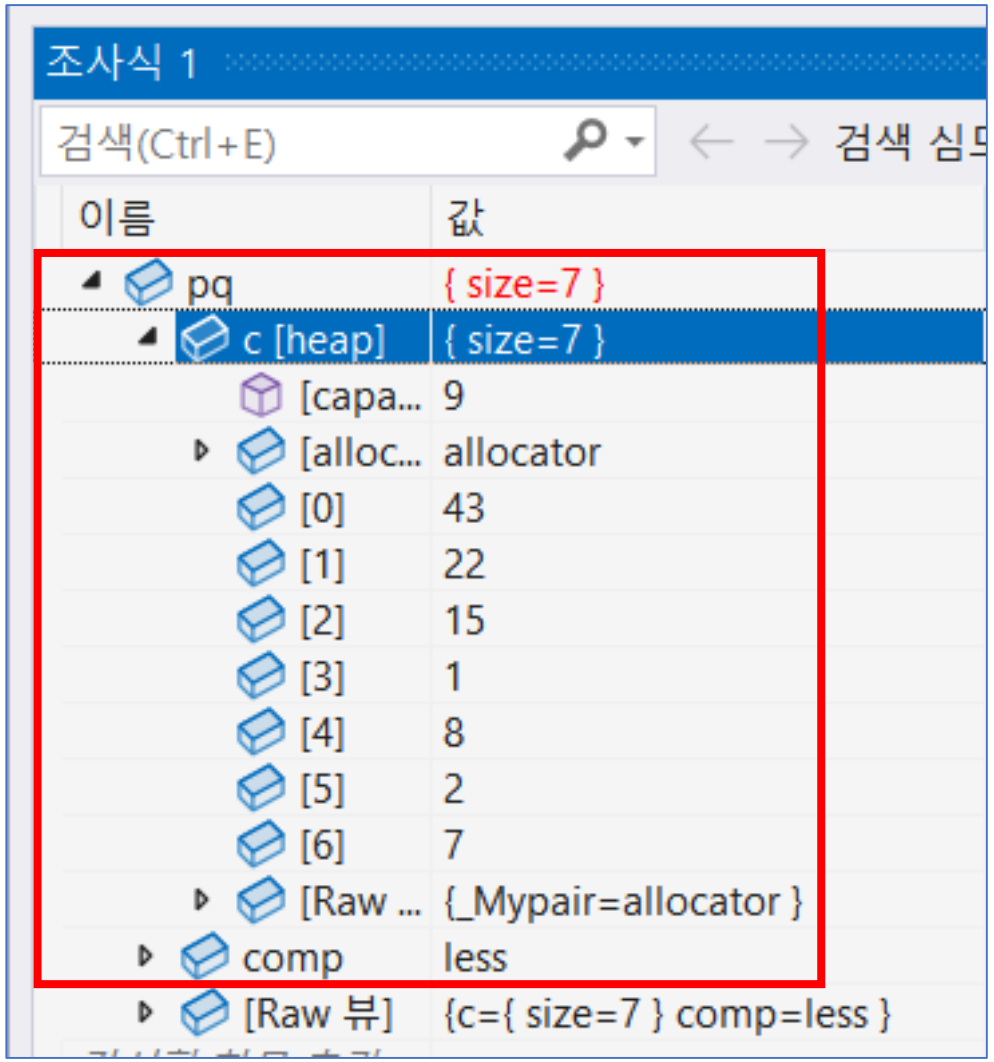
```
priority_queue<int> pq;

int n;
cin >> n;

for (int i = 0; i < n; i++) {
    int num;
    cin >> num;
    pq.push(num);
}

cout << "priority_queue size : " << pq.size() << '₩n';

while (!pq.empty()) {
    //.top() : 가장 우선순위 높은 요소
    cout << pq.top() << '₩n';
    pq.pop();
}
```

https://gist.github.com/hoconoco/de8937aff6915bfa002f3b7676947cfe

디버깅으로 확인한다.

데이터를 입력한 뒤, 조사식으로 살펴본다.

- [heap] 구조 확인 가능
- comp : less

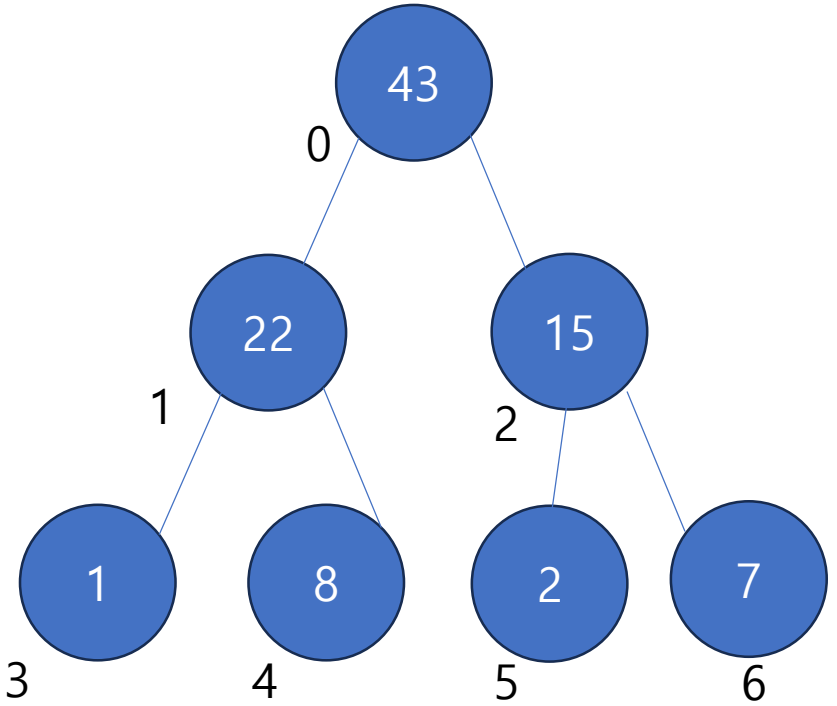


[참고] Heap Tree 자료구조

다음과 같은 규칙을 따른다.

1. completely binary tree
   - 자식 노드는 최대 2개
   - 데이터를 삽입 / 제거 시 왼쪽부터 먼저 채운다.

2. 직접 연결된 노드 간 우선순위가 있다.
   - 최대 힙 : 부모 노드 > 자식 노드
   - 최소 힙 : 부모 노드 < 자식 노드
   - 데이터를 삽입 / 제거 시 Heapify(힙 조정) 가 발생한다
   - Heapify의 시간복잡도 : logN

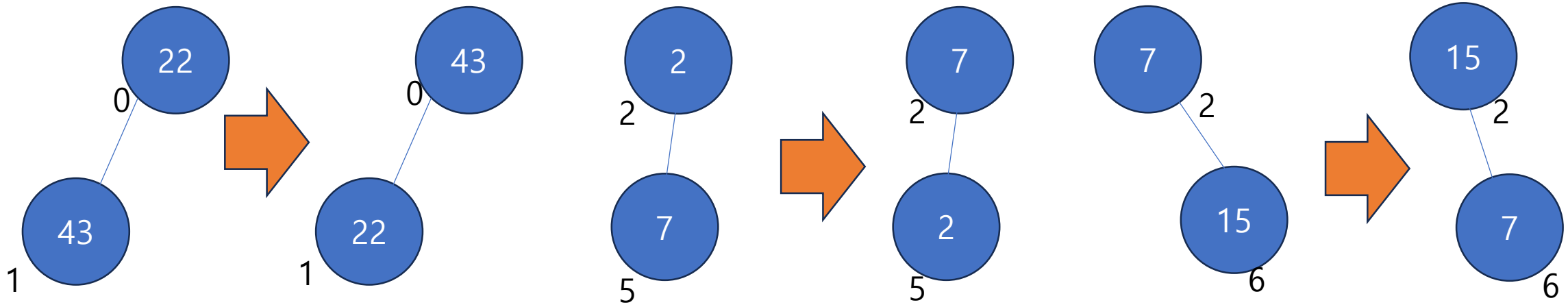


Heap Tree 의 데이터 삽입

데이터가 들어오면, 왼쪽 노드에서부터 채우며,

직접 연결된 부모 노드와의 크기를 비교해서 위치를 바꾼다.

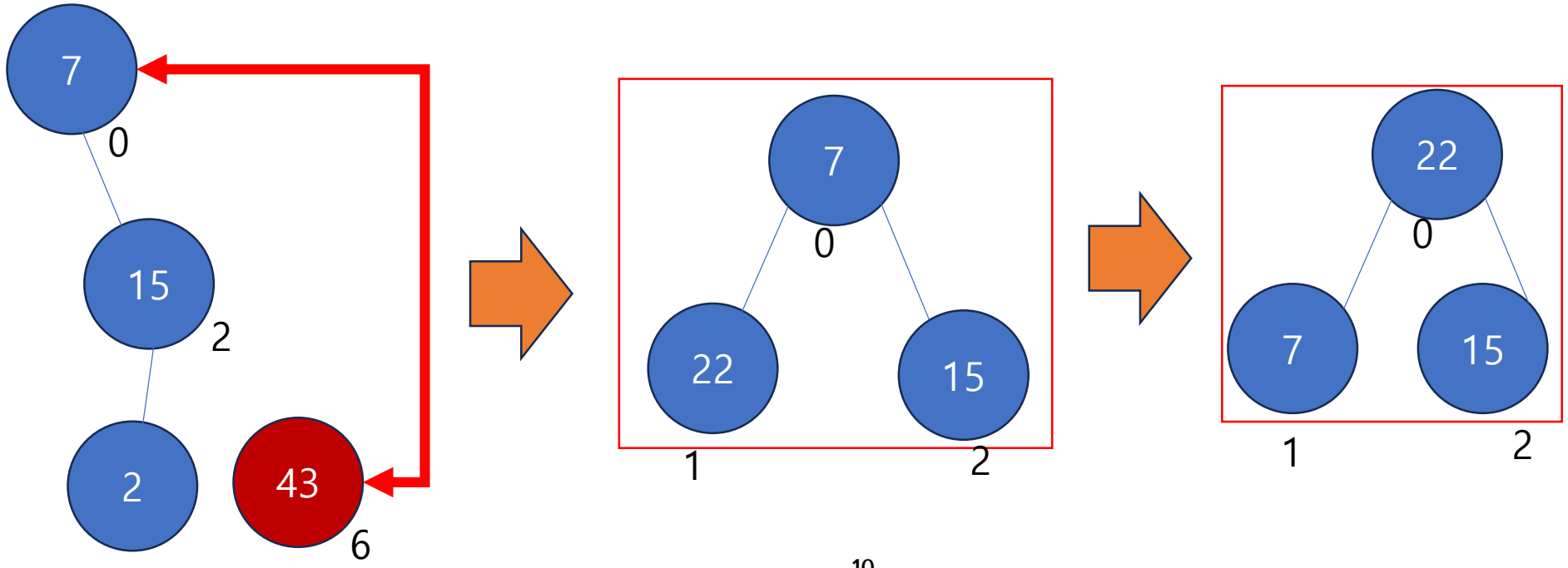
- 들어올 데이터 순서 : 22 → 43 → 2 → 1 → 8 → 7 → 15
- 1. 22 → 43 일 때, 43의 크기가 22보다 크니 위치를 바꾼다.
- 2. 2 → 7 일 때, 7의 크기가 2보다 크니 위치를 바꾼다.
- 3. 7 → 15 일 때, 15의 크기가 7보다 크니 위치를 바꾼다.



Heap Tree 의 데이터 삭제

데이터를 삭제 → 가장 우선순위 높은 노드(root) 부터 제거

- 트리 구조를 지켜가면서 삭제 해야 한다.
- 1. 가장 나중 번호 노드와 root 노드 순서 변경
- 2. 가장 나중 번호 노드 삭제
- 3. Heapify 처리 (반복)



우선순위 큐 정렬하기 (내부적으로 vector를 사용한다.)

- **내부 데이터를 큰 값부터 정렬 (최대힙, MAX Heap)**
  - default
  - less<int>
- **작은 값부터 정렬 (최소힙 MIN Heap)**
  - greater<int>

- **sort() 와 헷갈릴 수 있다.**
- **PQ (logN) > sort (NlogN)**

```
int main() {
    //vector 를 사용해서 정렬한다.
    ////less<int> : default (내림차순)
    //priority_queue<int, vector<int>, less<int> > pq;

    //greater<int> : (오름차순)
    priority_queue<int, vector<int>, greater<int> > pq;
```

https://gist.github.com/hoconoco/1aaf18bc205714fdc9f8c7b8cc1ac983

[참고] Priority Queue 의 Container 는 Vector이다.

Priority Queue 는 container adapter(어댑터) 이다.

- 자체적으로 데이터를 저장하는 container 가 없고, vector 사용한다.
- vector를 우선순위에 맞춰서 관리한다

```
_EXPORT_STD template <class _Ty, class _Container = vector<_Ty>, class _Pr = less<typename _Container::value_type>>
class priority_queue {
public:
    using value_type      = typename _Container::value_type;
    using reference       = typename _Container::reference;
    using const_reference = typename _Container::const_reference;
    using size_type       = typename _Container::size_type;
    using container_type  = _Container;
    using value_compare   = _Pr;
```

Priority Queue 내에 구조체를 넣어서 정렬한다.

- **다양한 자료구조를 적용할 수 있다.**

```
5
faker 28
karina 25
IU 25
yeona 21
ssafy 30
priority_queue size : 5
ssafy 30
faker 28
IU 25
karina 25
yeona 21
```

```cpp
struct NODE {
    string name;
    int age;

    // 연산자 오버로딩 (우선순위 설정)
    bool operator<(const NODE& right) const {
        //1. 나이가 큰 순서대로
        if (age < right.age) return true;
        if (age > right.age) return false;

        //2. 나이가 같으면 이름 길이가 짧은 순서대로
        return name.length() > right.name.length(); // 이름 길이가 짧은 순서대로
    }
};
```

https://gist.github.com/hoconoco/8fd817e152fb74e8831091ca902d81ad

같은 구조체를 사용하지만, 다르게 동작한다.

- 구조체 내 operator< () : 내림차순
- PQ : 오름차순 적용
- sort : 내림차순 적용

```
5
IU 20
karina 12
hifaker 15
hoho 123
ssafy 25
PQ 는 default 가 내림차순이므로, 오름차순 적용
karina 12
hifaker 15
IU 20
ssafy 25
hoho 123
----------------------
구조체 내 연산자 오버로딩으로 내림차순 정렬 했을 때
hoho 123
ssafy 25
IU 20
hifaker 15
karina 12
```

```cpp
struct NODE {
    string name;
    int age;

    bool operator< (const NODE& right)const {
        //나이 많은 순 정렬(단순 sort에선 내림차순)
        //PQ에선 반대로 오름차순이 된다.
        if (age < right.age) return false;
        if (age > right.age) return true;
        return false;
    }
};
```

https://gist.github.com/hoconoco/3bfe8ffabcd86100d8a620a6fcea757d

greater, less 이외에 다른 우선순위를 적용하고 싶다면, 사용자 정의 정렬을 하자

- **구조체로 연산자 오버로딩을 한다.**
- **함수 객체라고 한다.**

```
7
22 43 2 1 8 7 15
priority_queue size : 7
2 8 22 43 15 7 1
```

```cpp
struct cmp {
    bool operator()(int a, int b) {
        // 짝수가 우선순위가 높음
        if ((a % 2 == 0) && (b % 2 != 0)) return false; // a(짝), b(홀) → a 우선
        if ((a % 2 != 0) && (b % 2 == 0)) return true;  // a(홀), b(짝) → b 우선

        // 둘 다 짝수 → 오름차순 정렬
        if (a % 2 == 0 && b % 2 == 0) return a > b;

        // 둘 다 홀수 → 내림차순 정렬
        return a < b;
    }
};

int main() {
    priority_queue<int, vector<int>, cmp> pq;
```

https://gist.github.com/hoconoco/7d1b556a56ca1ac3efbd71ad806bd445

[참고] function object 함수 객체

함수처럼 동작하는 객체

- **더 깊은 내용은 OOP를 학습해야 한다.**

함수 객체를 사용하는 이유

1. **일반 함수보다 빠름**
2. **함수 포인터보다 인라인 최적화 가능**
   - 상태를 가질 수 있음
3. **내부 변수를 유지하며 연산 가능**
   - STL에서 정렬 기준, 우선순위 큐 비교 연산 등에 활용

```
struct cmp {
    bool operator()(int a, int b) {
        // 짝수가 우선순위가 높음
        if ((a % 2 == 0) && (b % 2 != 0)) return false; // a(짝), b(홀) → a 우선
        if ((a % 2 != 0) && (b % 2 == 0)) return true;  // a(홀), b(짝) → b 우선

        // 둘 다 짝수 → 오름차순 정렬
        if (a % 2 == 0 && b % 2 == 0) return a > b;

        // 둘 다 홀수 → 내림차순 정렬
        return a < b;
    }
};

int main() {
    priority_queue<int, vector<int>, cmp> pq;
```

[참고] PQ vs sort

Sort

- intro sort → NlogN 속도 보장
- 장점 : 데이터를 한번 정렬하면 특정 값을 찾기 좋다.
- 단점 : 데이터 변경이 잦은 경우, 매우 비효율적이다.
- 사용 예시 : 데이터를 한번 정렬한 뒤, 데이터 변경이 없을 경우 사용 (greedy)

PQ

- 데이터 삽입/삭제 시 logN 보장
- 장점 : 데이터의 추가/삭제가 빈번한 경우 유리
- 단점 : 전체 데이터 정렬이 안된다. 중간 값 찾기 or 변경이 매우 어렵다.
- 사용 예시 : 실시간으로 데이터의 우선순위가 바뀌는 경우, 최소/최대값을 자주 꺼내야 하는 경우

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

내일 방송에서 만나요!

삼성 청년 SW 아카데미