6.자료구조

23안해성

배우는 내용

- 연결 리스트
- 스택
- 큐
- 우선순위 큐

자료구조란?

- 어떤 데이터를 처리하기 편하도록 “특정한 방법 " 저장하는 것.
- 인덱스 소트에서 어떤 숫자를 추가한다고 했을 때,
  배열에서 숫자와 같은 인덱스에 +1을 하는 것을 생각하면 된다.

  이렇게 되면, 정렬 혹은 무슨 숫자가 몇 개 있는 지를
  처리하기 쉽게 된다.

연결리스트

Linked List

연결리스트

- 데이터를 저장하는 자료구조이다.

- 데이터를 “노드”라는 데이터로 분리하고 노드끼리 서로 연결하면서 저장하기 때문에, 삭제, 데이터 삽입이 쉽다.

- 연결리스트 그림 표현 :
  1 5 7 3 2 를 넣는다고 해보자

연결리스트 Vs 배열

- 데이터 접근 :
  배열 : 메모리에 연속적으로 저장되기 때문에, arr[2]처럼 한 번에 접근이 가능하다. O(1)
  연결리스트 : 앞에서 부터 차례대로 넘어가야지 원하는 데이터를 볼 수 있다. O(n)

- 데이터 삽입/삭제 :
  배열 : 중간에서 데이터 삽입/삭제 시 뒤에 있는 데이터를 뒤/앞으로 미뤄야 돼서 느리다. O(n)
  연결 리스트 : 노드끼리 연결을 추가하거나, 끊기만 하면 되기에 빠르게 가능하다. O(1)

연결리스트 Vs 배열

- 배열은 크기가 정적이고, 연결리스트는 동적이라는 등의 차이가 더 있다.

- 배열은 데이터 접근에 빠르지만 삽입/삭제가 느리고,
  연결리스트는 삽입/삭제가 빠르고 데이터 접근이 느리다는 것처럼
  자료구조에는 각각의 장단점이 있다.

- 즉, 문제보고 알맞는 자료구조를 선택하는 능력이 필요하다!

연결 리스트 사용하기

- 가장 먼저 〈list〉 헤더파일 추가하야 한다.

- std::list〈{저장할 타입}〉 {이름};
  의 형태로, 연결 리스트를 만들 수 있다.

- ! 아까 봤던 리스트의 그림과 달리, 화살표가 양쪽으로 있는
  양방향 리스트이다.

list 멤버 함수

Element access

front	access the first element (public member function)
back	access the last element (public member function)

Iterators

begin cbegin (C++11)	returns an iterator to the beginning (public member function)
end cend (C++11)	returns an iterator to the end (public member function)
rbegin crbegin (C++11)	returns a reverse iterator to the beginning (public member function)
rend crend (C++11)	returns a reverse iterator to the end (public member function)

Operations

merge	merges two sorted lists (public member function)
splice	moves elements from another list (public member function)
remove remove_if	removes elements satisfying specific criteria (public member function)
reverse	reverses the order of the elements (public member function)
unique	removes consecutive duplicate elements (public member function)
sort	sorts the elements (public member function)

Capacity

empty	checks whether the container is empty (public member function)
size	returns the number of elements (public member function)
max_size	returns the maximum possible number of elements (public member function)

Modifiers

clear	clears the contents (public member function)
insert	inserts elements (public member function)
insert_range (C++23)	inserts a range of elements (public member function)
emplace (C++11)	constructs element in-place (public member function)
erase	erases elements (public member function)
push_back	adds an element to the end (public member function)
emplace_back (C++11)	constructs an element in-place at the end (public member function)
append_range (C++23)	adds a range of elements to the end (public member function)
pop_back	removes the last element (public member function)
push_front	inserts an element to the beginning (public member function)
emplace_front (C++11)	constructs an element in-place at the beginning (public member function)
prepend_range (C++23)	adds a range of elements to the beginning (public member function)
pop_front	removes the first element (public member function)
resize	changes the number of elements stored (public member function)
swap	swaps the contents (public member function)

cppreference.com

list 함수 테스트

```
list<int> _list({1,5,7,3,2});

cout << "리스트 초기₩n";
print_list(_list);

cout << "list.back() / list.front()₩n";
cout << _list.back() << " / " << _list.front() << "₩n₩n";

cout << "list.empty() / list.size()₩n";
cout << _list.empty() << " / " << _list.size() << "₩n₩n";

cout << "list.push_back(3) / list.push_front(5) 이후 리스트 ₩n";
_list.push_back(3);
_list.push_front(5);
print_list(_list);

cout << "list.insert(list.begin(), 9) / list.erase(++list.begin()) 이후 리스트₩n";
_list.insert(_list.begin(), 9);
_list.erase(++_list.begin());
print_list(_list);

cout << "list.clear() 이후₩n";
_list.clear();
print_list(_list);
cout << "list.size() : " << _list.size();
cout << "₩nlist.empty() : " << _list.empty();
```

```
리스트 초기
1 5 7 3 2

list.back() / list.front()
2 / 1

list.empty() / list.size()
0 / 5

list.push_back(3) / list.push_front(5) 이후 리스트
5 1 5 7 3 2 3

list.insert(list.begin(), 9) / list.erase(++list.begin()) 이후 리스트
9 1 5 7 3 2 3

list.clear() 이후


list.size() : 0
list.empty() : 1
```

스택

stack

스택이란?

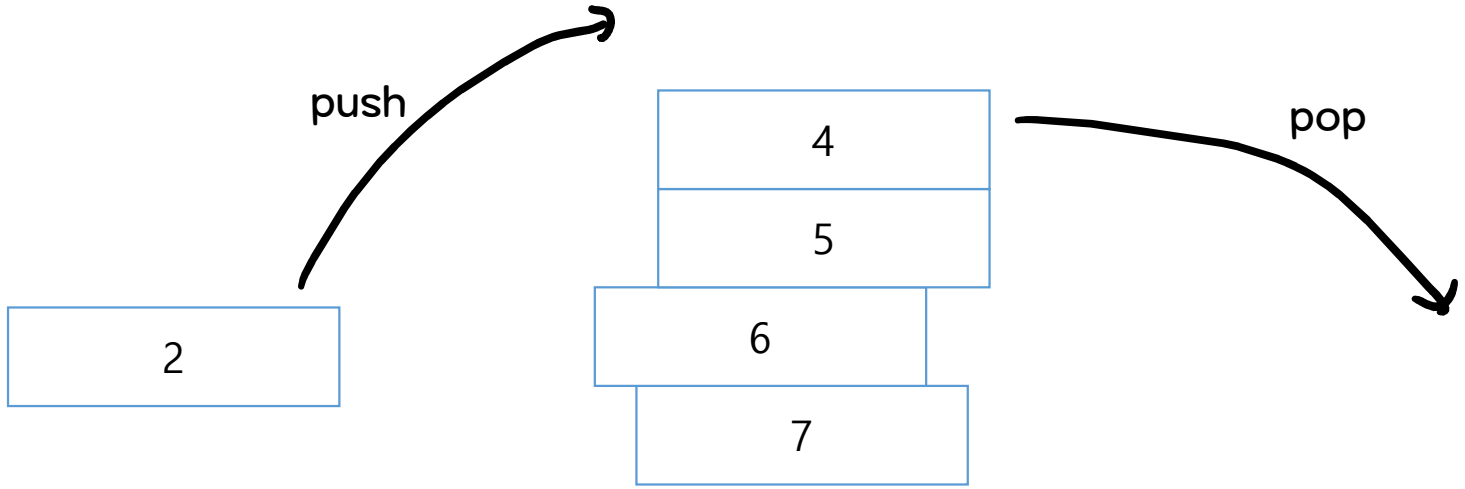
- 한 책 위에 책 100권이 쌓여 있다,
  여기서 맨 아래의 책을 꺼내려면 어떻게 해야 할까?

- 당연히,
  맨 위의 책부터 차례대로 꺼내야 마지막의 책을 꺼낼 수 있다.

스택

- 방금 예시처럼 스택은,
  데이터를 추가할 때는 이미 쌓여 있는 데이터 위로 쌓고,
  뺄 때는, 맨 위의 데이터를 가장 먼저 빼는 식으로 데이터를 저장한다.

  즉 먼저 들어간 데이터가 가장 나중에 나오는 구조이다.



스택 사용하기

- 가장 먼저 〈stack〉 헤더파일 추가.

- stack〈{저장할 타입}〉 {이름};
  으로 스택을 만들 수 있다.

스택 멤버 함수

Element access	
top	accesses the top element (public member function)
Capacity	
empty	checks whether the underlying container is empty (public member function)
size	returns the number of elements (public member function)
Modifiers	
push	inserts element at the top (public member function)
emplace (C++11)	constructs element in-place at the top (public member function)
push_range (C++23)	inserts a range of elements at the top (public member function)
pop	removes the top element (public member function)
swap (C++11)	swaps the contents (public member function)

cppreference.com

스택 테스트

```cpp
int main() {
    stack<int> _stack;

    cout << "stack.size() / stack.empty()₩n";
    cout << _stack.size() << " / " << _stack.empty() << "₩n₩n";

    _stack.push(1);
    _stack.push(2);
    _stack.push(3);

    cout << "stack.push(1) / stack.push(2) / stack.push(3) 이후₩n";
    cout << "stack.top() / stack.size() / stack.empty()₩n";
    cout << _stack.top() << " / " << _stack.size() << " / " << _stack.empty() <<"₩n₩n";

    _stack.push(5);
    cout << "stack.push(5) 이후₩n";
    cout << "stack.top() / stack.size() / stack.empty()₩n";
    cout << _stack.top() << " / " << _stack.size() << " / " << _stack.empty() << "₩n₩n";

    _stack.pop();
    cout << "stack.pop() 이후₩n";
    cout << "stack.top() / stack.size() / stack.empty()₩n";
    cout << _stack.top() << " / " << _stack.size() << " / " << _stack.empty() << "₩n₩n";
```

```
stack.size() / stack.empty()
0 / 1

stack.push(1) / stack.push(2) / stack.push(3) 이후
stack.top() / stack.size() / stack.empty()
3 / 3 / 0

stack.push(5) 이후
stack.top() / stack.size() / stack.empty()
5 / 4 / 0

stack.pop() 이후
stack.top() / stack.size() / stack.empty()
3 / 3 / 0
```

9012.괄호

- 문제 요약 :
  괄호로만 이루어진 문자열이 있을 때, 모든 열린 괄호(“(”)가 한 쌍이 되는
  닫힌 괄호(“)”)가 있는지 판별하는 문제.

- 문제 풀이 :
  닫힌 괄호는 가장 최근에 나온 열린 괄호와 짝이 된다.

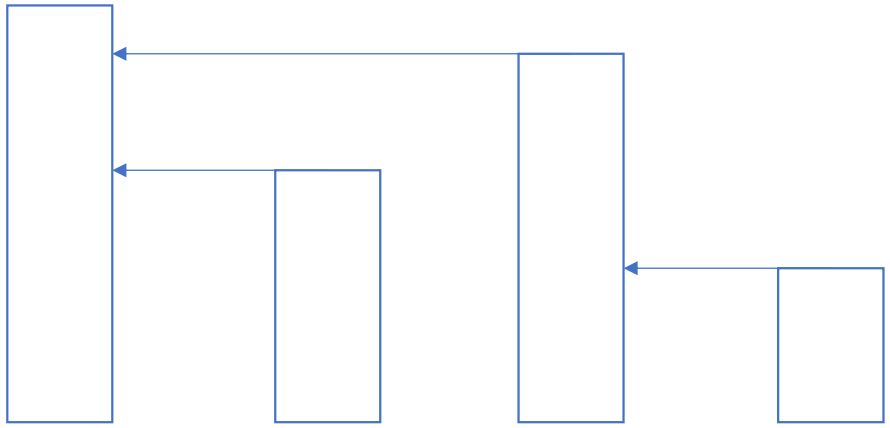
  즉 문자열을 읽어가며 열린 괄호를 스택에 넣고 닫힌 괄호가 나오면 열린 괄호를
  빼면 된다.

- 더 나은 방법이 있을까?

2493.탑

- 문제 요약:
  높이가 다른 막대 그래프의 높이가 왼쪽부터 오른쪽까지 여러 개 나온다.

  이때, 한 막대의 왼쪽 막대 중 자신보다 크면서 가장 가까운 막대를 찾는 문제이다.



2493.탑 풀이

- 가장 오른쪽 값부터 차례대로 스택에 넣는다.
  단, 지금 넣게 되는 값이 스택의 맨 위에 있는 값보다 작아야 한다.

  즉 지금 넣는 수가 스택의 맨 위 값보다 크면 스택의 위에 있는 값을 계속 빼면 된다.

- 이렇게 했을 때, 어떤 스택에서 어떤 값을 빼게 하는 그 값이

  “한 막대의 왼쪽 막대 중 자신보다 크면서 가장 가까운 막대”가 된다.

참고

- 2493번 같이 스택의 내용물을 오름차순, 내림차순으로 유지하면서 값을 넣고 빼고 하는 것을 monotone stack 기법이라고 한다.

더 어려운 문제는?

- 1725.히스토그램 (분할 정복 때 문제와 같음!)

Queue

큐란?

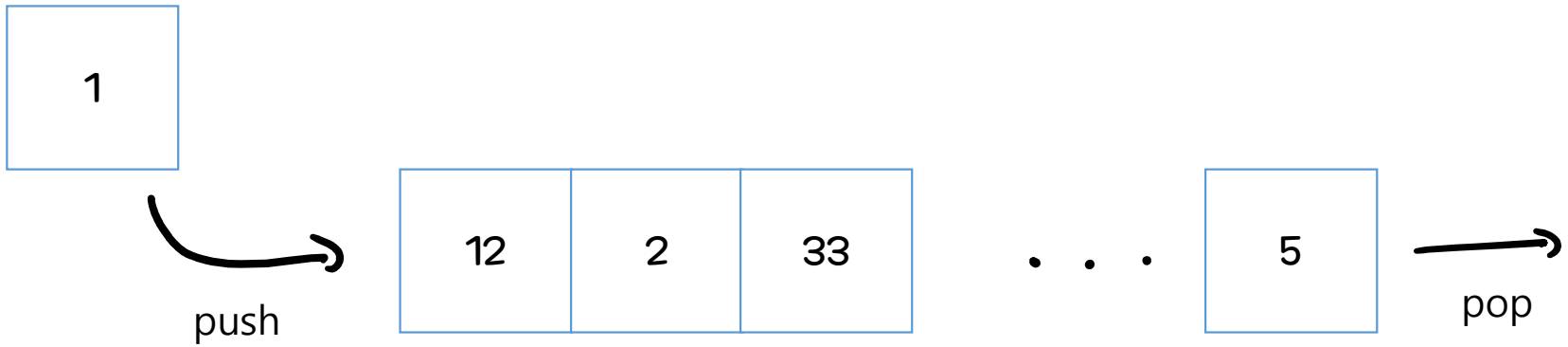
- 유명한 가게에 들어가기 위해 줄을 서는 것을 생각해보자,
  이미 줄이 있을 때, 새로 온 사람은 어디로 서야 하고, 누가 먼저 들어갈까?

- 당연하게 새로 온 사람은 이미 있는 줄 뒤에 서고,
  맨 앞에, 즉 가장 먼저 온 사람이 가장 먼저 가게로 들어가게 된다.

큐

- 큐 또한 데이터를 넣을 때는 이미 있던 데이터의 맨 뒤에 넣고, 뺄 때는 맨 앞의 데이터를 빼게 된다.

  즉 가장 먼저 들어간 데이터가 가장 먼저 나오는 구조이다.



큐 사용하기

- 〈queue〉 헤더파일 추가하기

- queue〈{저장할 타입}〉 {이름};
  으로 큐를 만들 수 있다.

큐의 멤버 함수

Element access	
front	access the first element (public member function)
back	access the last element (public member function)
Capacity	
empty	checks whether the underlying container is empty (public member function)
size	returns the number of elements (public member function)
Modifiers	
push	inserts element at the end (public member function)
emplace (C++11)	constructs element in-place at the end (public member function)
push_range (C++23)	inserts a range of elements at the end (public member function)
pop	removes the first element (public member function)
swap (C++11)	swaps the contents (public member function)

cppreference.com

큐 테스트

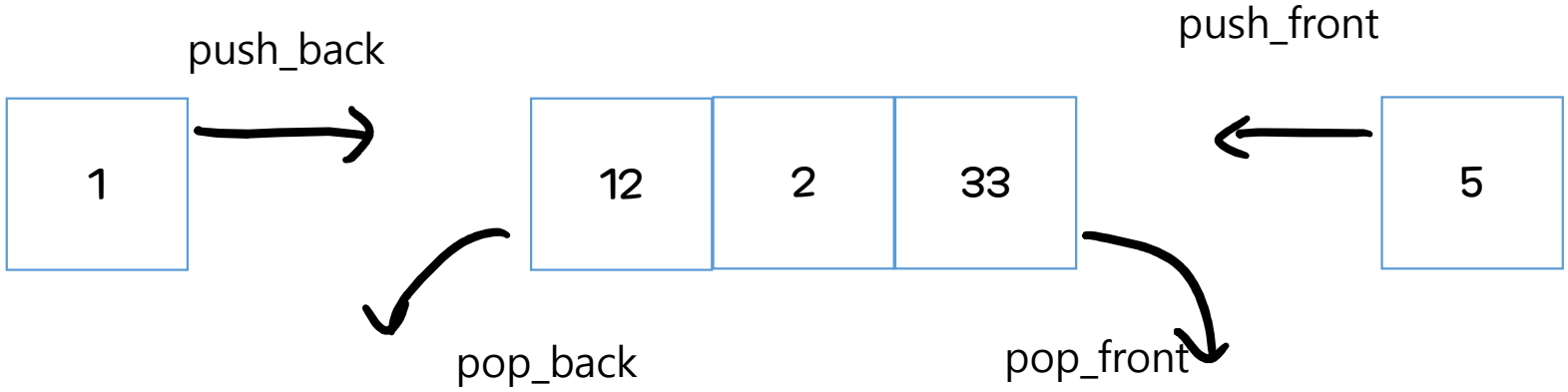
```
queue.size() / queue.empty()
0 / 1

queue.push(1) / queue.push(2) / queue.push(3) 이후
queue.front() / queue.back() /queue.size() / queue.empty()
1 / 3 / 3 / 0

queue.push(5) 이후
queue.front() / queue.back() /queue.size() / queue.empty()
1 / 5 / 4 / 0

queue.pop() 이후
queue.front() / queue.back() /queue.size() / queue.empty()
2 / 5 / 3 / 0
```

```
queue<int> _queue;

cout << "queue.size() / queue.empty()₩n";
cout << _queue.size() << " / " << _queue.empty() << "₩n₩n";

_queue.push(1);
_queue.push(2);
_queue.push(3);

cout << "queue.push(1) / queue.push(2) / queue.push(3) 이후₩n";
cout << "queue.front() / queue.back() /queue.size() / queue.empty()₩n";
cout << _queue.front() << " / " << _queue.back() << " / " << _queue.size() << " / " << _queue.empty() << "₩n₩n";

_queue.push(5);
cout << "queue.push(5) 이후₩n";
cout << "queue.front() / queue.back() /queue.size() / queue.empty()₩n";
cout << _queue.front() << " / " << _queue.back() << " / " << _queue.size() << " / " << _queue.empty() << "₩n₩n";

_queue.pop();
cout << "queue.pop() 이후₩n";
cout << "queue.front() / queue.back() /queue.size() / queue.empty()₩n";
cout << _queue.front() << " / " << _queue.back() << " / " << _queue.size() << " / " << _queue.empty() << "₩n₩n";
```

큐 + 스택

- 덱(deque)은 즉 데이터를 앞에서도 뺄 수 있고, 뒤에서도 뺄 수 있는 구조이다.
  즉 큐처럼 쓸 수도 있고, 스택처럼 쓸 수도 있다.

- 또한 데이터를 뒤에서 넣을 뿐만 아니라, 앞으로도 넣을 수 있다.



2164.카드2

- 문제 요약:
  카드 더미가 주어지고
  맨 위의 카드를 버리고, 다음에 나온 카드를 카드 더미 맨 밑에 넣는다.

  이를 마지막 1장 남았을 때까지 반복했을 때, 마지막 1장은 무엇인가.
  (카드는 1~N까지 숫자가 적혀져 있다.)

2164 풀이

- 큐에 1~N까지 차례대로 넣고,
  마지막 1장이 나올 때까지 빼고, 넣고를 반복하면 된다.

- 이렇게 해도 O(N)이기 때문에 충분하다.

더 어려운 문제는?

- 11003.최솟값 찾기

우선 순위 큐

Priority Queue

우선 순위 큐?

- 큐는 “가장 먼저 들어온” 데이터가 가장 먼저 나가는 큐이다.

  그럼 우선 순위 큐는 어떤 데이터가 가장 먼저 나갈까?

- 말 그대로 미리 정한 우선 순위가 가장 높은 데이터가 가장 먼저 나가게 된다.

우선 순위 큐

- 수가 가장 작을 수록 우선 순위가 높다고 해보자.

- 3, 5, 2, 7, 2, 0 이 우선 순위 큐에 데이터로 들어왔다고 해보자,

  어떤 값이 가장 먼저 나갈까?

- 3이 추가로 우선 순위 큐에 들어왔다면,

  어떤 값이 가장 먼저 나갈까?

우선 순위 큐 사용하기

- 가장 먼저 〈queue〉 헤더파일 추가하기.
- priority_queue〈{저장할 타입}, {저장할 공간 자료형}, {비교 객체}〉 {이름};
  으로 우선 순위 큐를 만들 수 있습니다.

- {저장할 공간 자료형}, {비교 객체}을 안 넣는다면, 기본적으로

  값이 가장 큰 값이 우선순위가 높은 우선순위 큐가 만들어 집니다.

{비교 객체}

- 우선 순위 큐에서는 어떤 값이 더 우선순위가 높은지를 검사할 수 있는 데이터(객체)를 넣어 주어야 합니다.

- func(int a, int b);를 비교 객체에 넣었다고 했을 때,

func함수가 참이면, 왼쪽에 있는 매개변수가 더 우선 순위가 높다고 판단하게 됩니다.

{비교 객체}

```
bool comp(int a, int b) {
        return a > b;
}
```

- 위의 함수를 {비교 객체}로 넣게 된다면, a가 b보다 클 때 참이 되므로 값이 큰 데이터가 우선순위가 높게 됩니다.

우선 순위 큐 만들기

- 단순히 작은 수가 우선 순위가 높을 때,
  priority_queue<int> pq;

- 단순히 큰 수가 우선 순위가 높을 때,
  priority_queue<int, vector<int>, greater<int>> pq;

- comp라는 비교 함수를 직접 만들었을 때,
  priority_queue<int, vector<int>, bool(*)(int,int)> pq(comp);

우선 순위 큐의 맴버 함수

Element access	
`top`	accesses the top element (public member function)
Capacity	
`empty`	checks whether the underlying container is empty (public member function)
`size`	returns the number of elements (public member function)
Modifiers	
`push`	inserts element and sorts the underlying container (public member function)
`emplace` (C++11)	constructs element in-place and sorts the underlying container (public member function)
`push_range` (C++23)	inserts a range of elements and sorts the underlying container (public member function)
`pop`	removes the top element (public member function)
`swap` (C++11)	swaps the contents (public member function)

우선 순위 큐 테스트

```cpp
int main() {
    priority_queue<int> pq;

    cout << "pq.size() / pq.empty()₩n";
    cout << pq.size() << " / " << pq.empty() << "₩n₩n";

    pq.push(3);
    pq.push(5);
    pq.push(2);
    pq.push(7);
    pq.push(2);
    pq.push(0);
    cout << "우선 순위 큐에 3, 5, 2, 7, 2, 0을 넣은 후₩n";
    cout << "pq.top() / pq.size() / pq.empty()₩n";
    cout << pq.top() << " / " << pq.size() << " / " << pq.empty() << "₩n₩n";

    pq.push(3);
    cout << "우선 순위 큐에 3을 넣은 후₩n";
    cout << "pq.top() / pq.size() / pq.empty()₩n";
    cout << pq.top() << " / "  << pq.size() << " / " << pq.empty() << "₩n₩n";

    pq.pop();
    cout << "pq.pop() 이후₩n";
    cout << "pq.top() / pq.size() / pq.empty()₩n";
    cout << pq.top() << " / " << pq.size() << " / " << pq.empty() << "₩n₩n";
```

```
pq.size() / pq.empty()
0 / 1

우선 순위 큐에 3, 5, 2, 7, 2, 0을 넣은 후
pq.top() / pq.size() / pq.empty()
7 / 6 / 0

우선 순위 큐에 3을 넣은 후
pq.top() / pq.size() / pq.empty()
7 / 7 / 0

pq.pop() 이후
pq.top() / pq.size() / pq.empty()
5 / 6 / 0
```

1374. 강의실

- 문제 요약:
  강의실을 이용하는데 시작 시간과 끝나는 시간이 주어진다.
  이때, 강의가 겹치지 않도록 강의실을 배정해 줘야 한다.

  이때, 사용할 강의실 개수는 최소 몇 개여야 할까?

1374 풀이

- 작은 값이 우선순위가 높은 우선순위 큐를 만들고,
  강의 시간들을 시작 시간에 맞춰 오름차순 정렬을 한다.

  지금 넣을 강의 시간의 시작시간이,

  큐의 맨 앞의 값보다 크면, 큐에서 값을 빼고 강의의 끝나는 시간을 넣고
  아니면 그냥 끝나는 시간을 넣는다.

더 어려운 문제는?

- 13334.철로