

# 알튜비튜

## 맵과 셋

오늘은 STL에서 제공하는 associative container인 set과 map에 대해 알아봅니다.  
데이터를 선형으로 저장하는 sequence container (ex. vector)와 달리 연관된 key-value 쌍을 저장합니다.

이런 문제가 있다고 해봅시다.



“배열 [1, 6, 2, 1, 9, 8]에서 중복된 수를 제거한 뒤, 오름차순 정렬한 결과는?”

# 벡터를 사용한다면?



```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main(){
    vector<int> arr = {1, 6, 2, 1, 9, 8};
    vector<int> result;
    for (int i = 0; i < arr.size(); i++) {
        // 존재하지 않는 원소라면 -> result에 넣기(중복 방지)
        if (find(result.begin(), result.end(), arr[i]) == result.end()) {
            result.push_back(arr[i]);
        }
    }
    sort(result.begin(), result.end()); //오름차순 정렬
    return 0;
}
```

시간 복잡도면에서도 효율적이지 않고, 코드도 길다.

## Set - c++

- key 라고 불리는 원소(value)의 집합
- key 값을 정렬된 상태로 저장
- key 값을 중복 없이 저장
- 검색, 삽입, 삭제에서의 시간 복잡도는  $O(\log N)$
- 랜덤한 인덱스의 데이터에 접근 불가

# 셋으로 다시 구현해봅시다!

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

int main() {
    vector<int> arr = {1, 6, 2, 1, 9, 8};
    set<int> result;

    for (int i = 0; i < arr.size(); i++) {
        result.insert(arr[i]);
    }
}
```

← set에서 삽입 = insert()

→ 1 2 6 8 9



# 랜덤한 인덱스에 접근 불가?

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

int main() {
    vector<int> v;
    set<int> s;

    v.push_back(2);
    v.push_back(1);
    s.insert(2);
    s.insert(1);

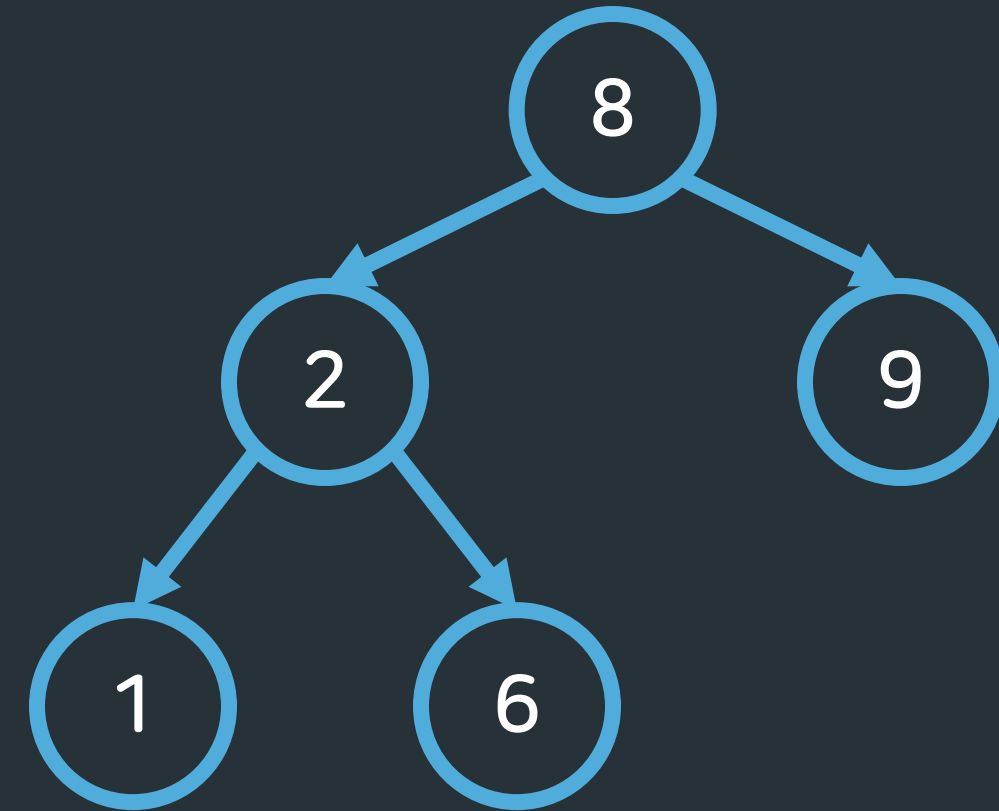
    int a = v[0];
    int b = s[0];
}
```

← 가능  
← 불가능

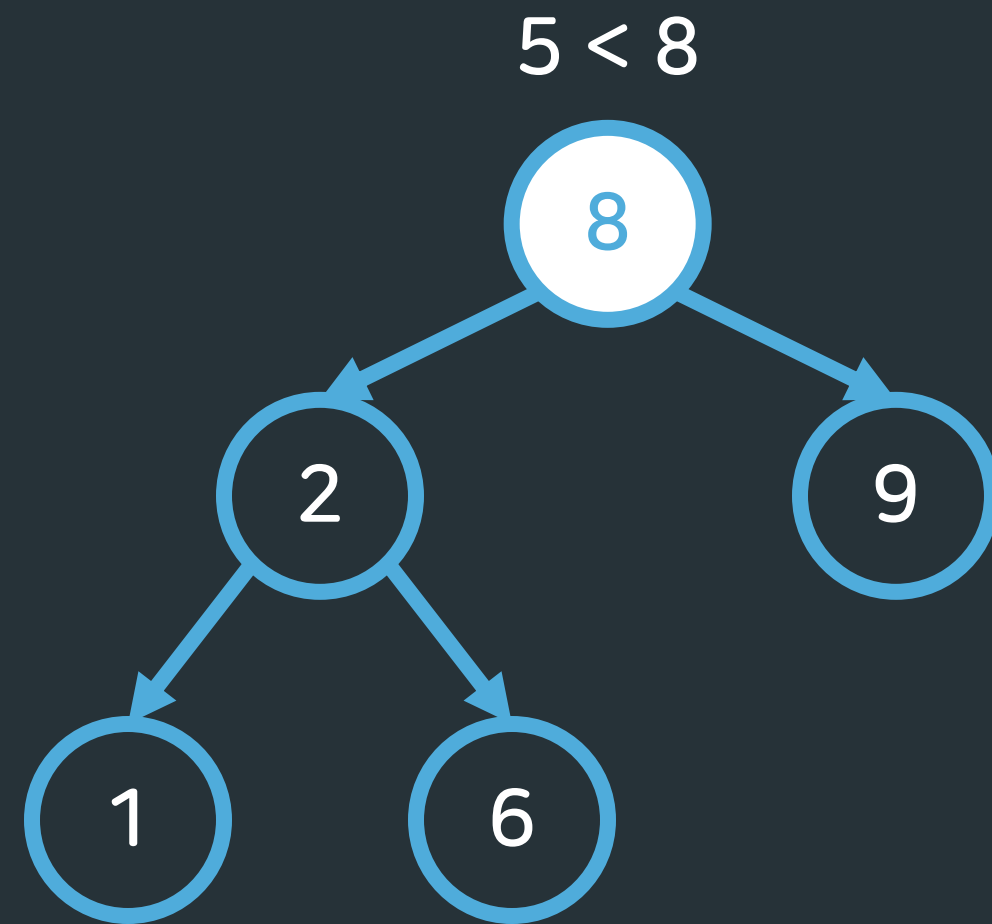
## BST (Binary Search Tree)

- 하나의 parent(root)에 최대 2개의 child가 있음
- 부모의 왼쪽 서브 트리 값들은 모두 부모 노드보다 작음
- 부모의 오른쪽 서브 트리 값들은 모두 부모 노드보다 큼

\* 사실 정확히 말하면 여기서 발전된 형태인 red-black tree(균형 이진 트리)를 사용



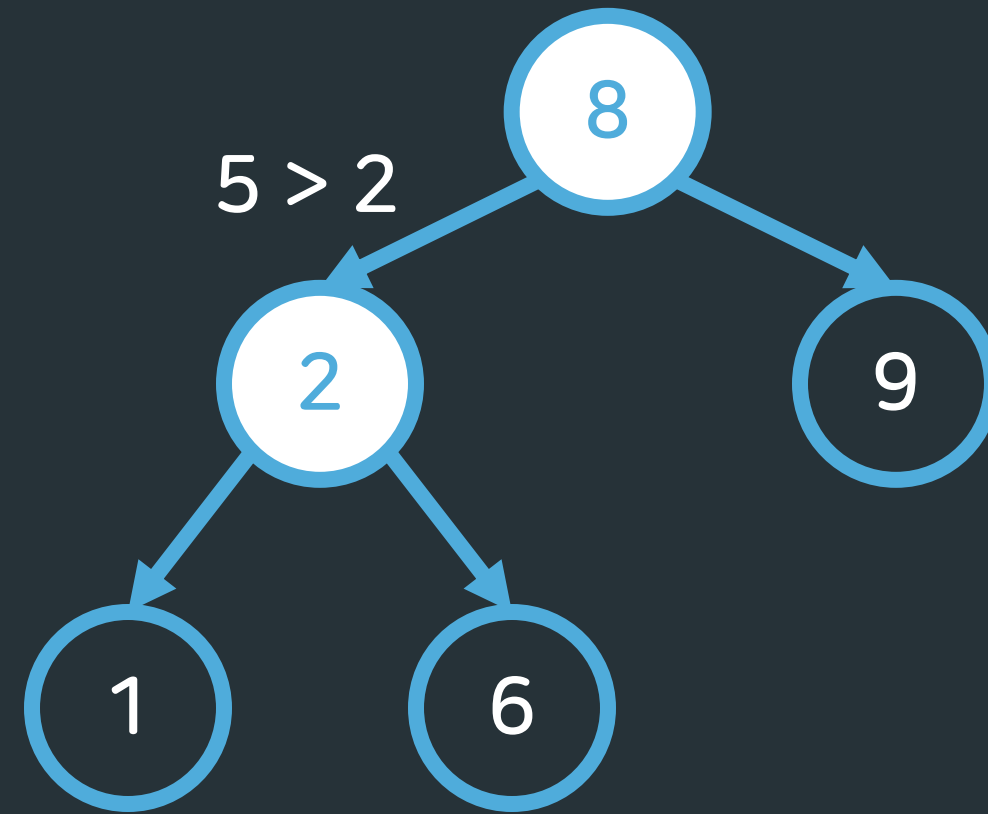
# C++ 셋에 데이터가 들어가는 과정



현재 들어오는 key = 5

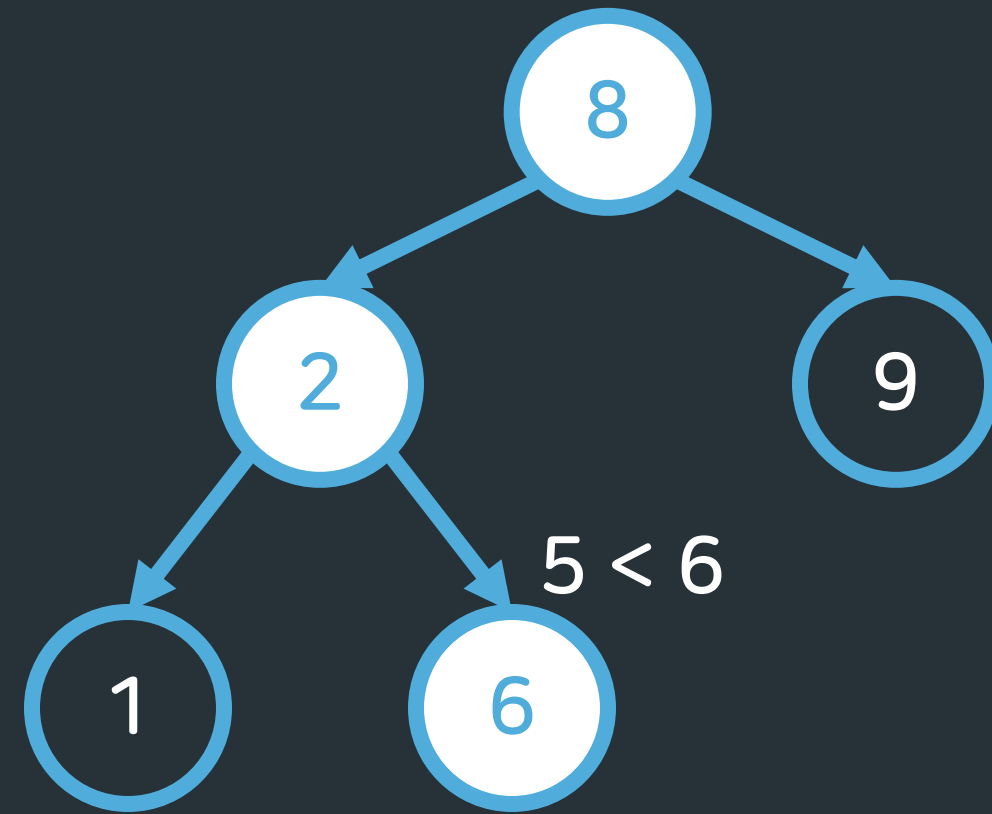


# C++ 셋에 데이터가 들어가는 과정



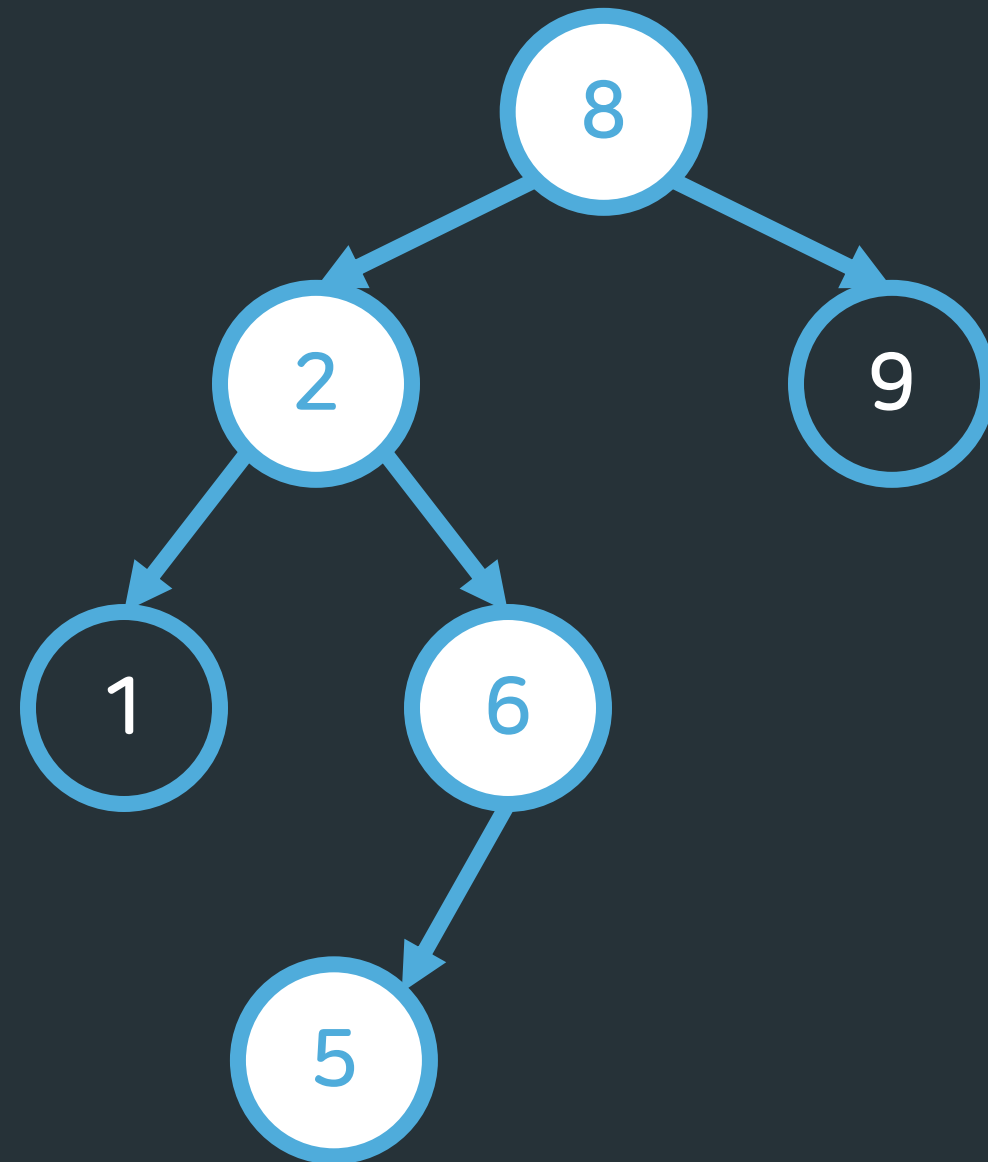
현재 들어오는 key = 5

# C++ 셋에 데이터가 들어가는 과정



현재 들어오는 key = 5

# C++ 셋에 데이터가 들어가는 과정



현재 들어오는 key = 5

## 반복자 (iterator)

- 포인터와 비슷한 개념
- 컨테이너에 보관된 원소에 접근할 때 사용
- "container<자료형>::iterator" 로 사용 가능
- begin(): 순차열의 시작
- end(): 순차열의 끝 (실제 원소를 가르키는 게 아니라 마지막 원소의 다음을 가리킴)
- 임의 접근 반복자(vector, deque)를 제외하고는 사칙연산 불가능

```
#include <iostream>
#include <set>

using namespace std;

int main() {
    set<int> s;

    s.insert(2);
    s.insert(1);

    set<int>::iterator iter; ← 반복자 선언
    for (iter = s.begin(); iter != s.end(); iter++) { ← 순회
        cout << *iter << ' '; ← 포인터로 접근
    }
}
```

# 낯설어 하실 것 같아서 벡터로도 준비했어요

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> v;

    v.push_back(2);
    v.push_back(1);

    vector<int>::iterator iter; ← 반복자 선언
    for (iter = v.begin(); iter != v.end(); iter++) { ← 순회
        cout << *iter << ' '; ← 포인터로 접근
    }
}
```

# C++은 생각보다 똑똑해요

```
#include <iostream>
#include <set>
```

```
using namespace std;
```

```
int main() {
    set<int> s;
```

```
    s.insert(2);
    s.insert(1);
```

```
    for (auto iter = s.begin(); iter != s.end(); iter++) {
        cout << *iter << ' ';
    }
```

← 자동으로 반복자 선언과 동시에 순회

```
    for (auto iter:s) {
        cout << iter << ' ';
    }
```

← 자동으로 반복자 선언과 동시에 순회  
(조금 더 향상된 버전)

```
}
```

## /<> 10867번 : 중복 빼고 정렬하기 - Silver 5

### 문제

- N개의 수를 오름차순 정렬
- 같은 수는 한 번만 출력

### 제한 사항

- N의 범위는  $1 \leq N \leq 100,000$
- 각각의 수 k는  $-1,000 \leq k \leq 1,000$



## /<> 10867번 : 중복 빼고 정렬하기 - Silver 5

### 문제

- N개의 수를 오름차순 정렬
- 같은 수는 한 번만 출력 → 중복 x => set!

### 제한 사항

- N의 범위는  $1 \leq N \leq 100,000$
- 각각의 수 k는  $-1,000 \leq k \leq 1,000$

## 예제 입력1

```
10  
1 4 2 3 1 4 2 3 1 2
```

## 예제 출력1

```
1 2 3 4
```

이런 문제가 있다고 해봅시다.



“학생의 이름과 해당 학생의 수학 성적이 주어진다.  
학생의 이름이 입력되면 해당 학생의 수학 성적을 구하라.”

# 구조체와 벡터를 사용한다면?



```
#include <iostream>
#include <vector>

using namespace std;

struct info {
    string name;
    int math_score;
};

int main() {
    vector<info> student;

    student.push_back({"lee", 42});
    student.push_back({"lim", 100});
    student.push_back({"bae", 50});

    string target = "bae";
    for (int i = 0; i < student.size(); i++) {
        if (student[i].name == target) {
            cout << student[i].math_score;
        }
    }
}
```

cf) vector 컨테이너에 지금 구현한 것처럼  
검색역할을 해주는 함수가 있어요!  
 $O(n)$ 으로 동일한 시간복잡도를 가집니다.

# 구조체와 벡터를 사용한다면?



```
#include <iostream>
#include <vector>

using namespace std;

struct info {
    string name;
    int math_score;
};

int main() {
    vector<info> student;

    student.push_back({"lee", 42});
    student.push_back({"lim", 100});
    student.push_back({"bae", 50});

    string target = "bae";
    for (int i = 0; i < student.size(); i++) {
        if (student[i].name == target) {
            cout << student[i].math_score;
        }
    }
}
```

학생 1명을 찾는데  $O(n)$ 의 시간 복잡도... 만약 찾아야할 학생이 천만명이라면?

## Map

- 다양한 자료형의 데이터를 **key-value** 쌍으로 저장
- key 값을 정렬된 상태로 저장
- key 값을 중복 없이 저장
- 검색, 삽입, 삭제에서의 시간 복잡도는  $O(\log N)$
- 랜덤한 인덱스의 데이터에 접근 불가

# 맵으로 다시 구현해봅시다!



```
#include <iostream>
#include <map>

using namespace std;

int main() {
    map<string, int> student;

    student["lee"] = 42; ← key 값을 인덱스처럼 접근해서 key-value 삽입 가능
    student["lim"] = 100;
    student["bae"] = 50;

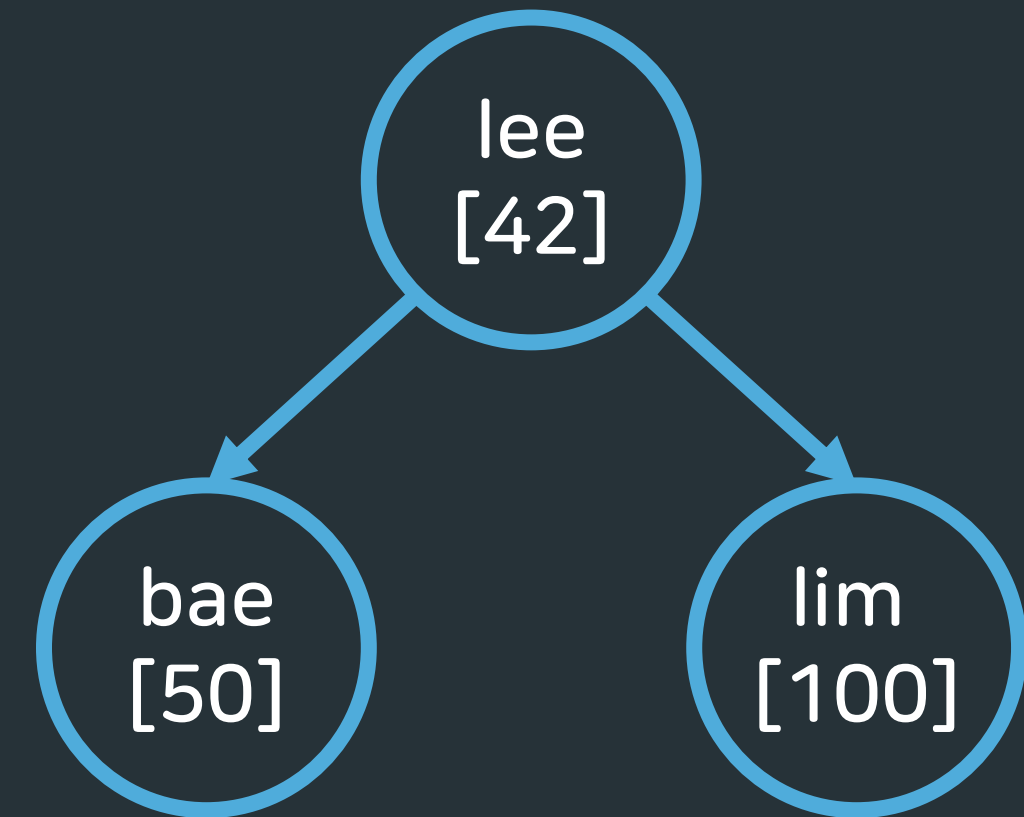
    string target = "bae";
    cout << student[target]; ← key 값을 인덱스처럼 사용해서 value에 접근 가능
}
```



## BST (Binary Search Tree)

- 하나의 parent(root)에 최대 2개의 child가 있음
- 부모의 왼쪽 서브 트리 값들은 모두 부모 노드보다 작음
- 부모의 오른쪽 서브 트리 값들은 모두 부모 노드보다 큼

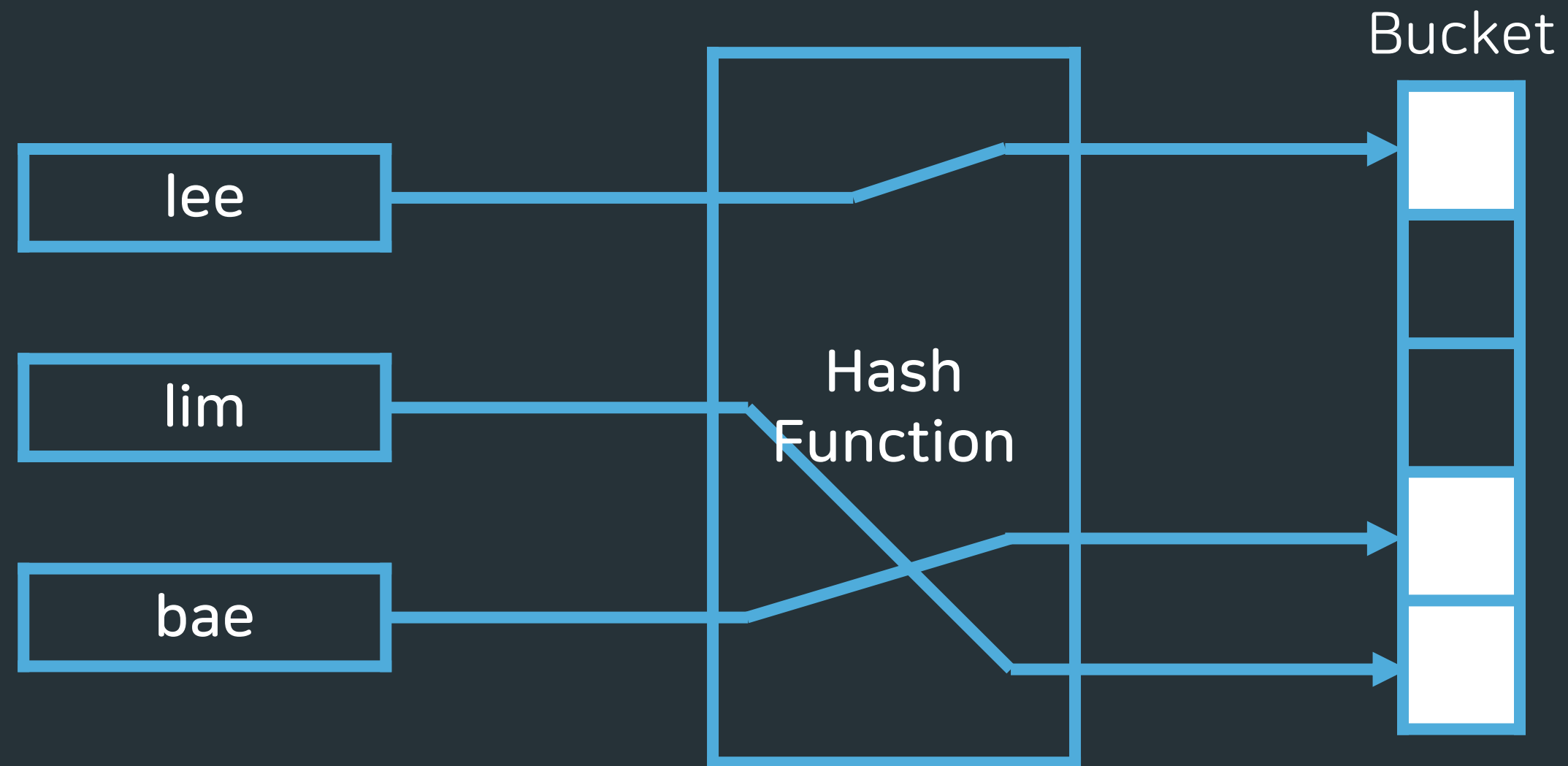
\* 사실 정확히 말하면 여기서 발전된 형태인 red-black tree(균형 이진 트리)를 사용





## 해싱 (Hashing)

- 해시 함수를 이용해 **key**를 특정한 값으로 변환
- 변환된 값은 **주소**로 사용되며 해당 위치에 **value** 저장
- 암호에 많이 사용됨
- 검색, 삽입, 삭제에서의 시간 복잡도는  $O(1)$



“key가 영어 소문자로만 이루어진 문자열 `str`일 때,  
모든 `i` ( $0 \leq i < \text{str.size}()$ )에 대해  $(\text{str}[i] - 'a') * i$ 를 더한 값을  
bucket의 크기로 나눈 나머지”

- \* Bucket은 데이터가 저장되는 공간
- \* Bucket의 크기는 소수 (prime number)로 하는 것을 권장

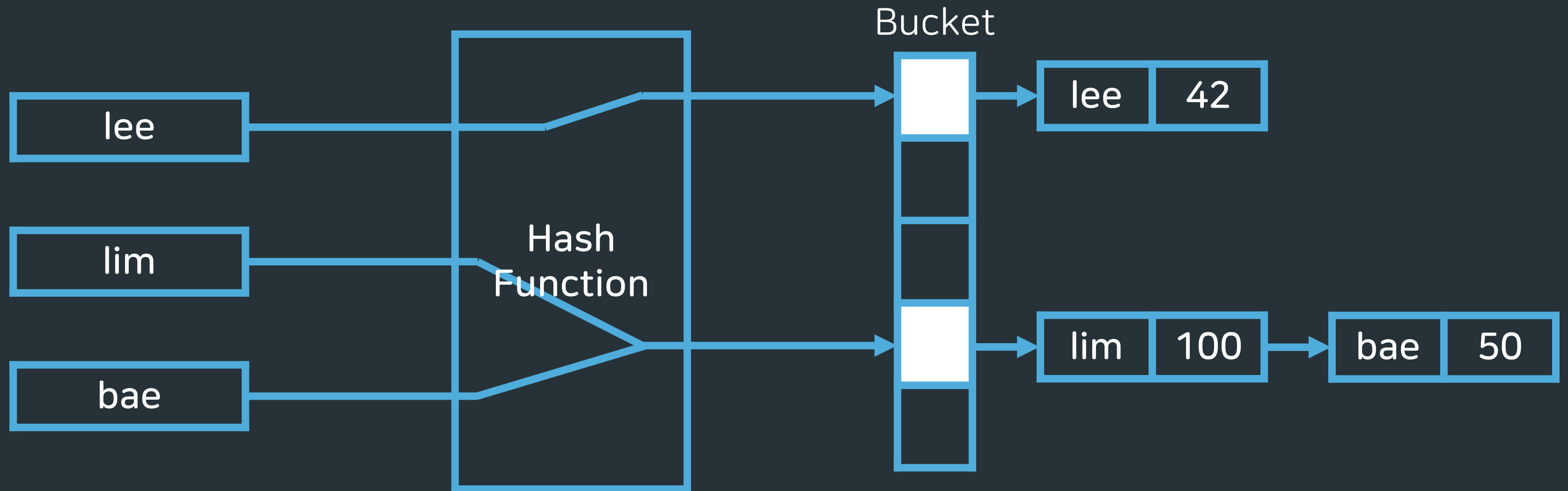
# 서로 다른 key의 해시 함수 결과값이 겹친다면?

## 충돌 (Collision)

- 서로 다른 input에 대해 같은 output이 나오는 현상
- 해결법
  1. 버킷의 크기를 아주 크게 하기 -> 공간 복잡도로 인한 메모리 초과
  2. 이중해싱, 재해싱, 기타등등...
  3. 체이닝

# 체이닝 (Chaining)

- Bucket에서 충돌이 일어나면 기존 값과 새로운 값을 연결리스트로 연결
- 자료들이 많이 연결되면 검색 효율이 낮아짐
- 충돌이 날 때 공간을 생성하므로 메모리 절약



# C++ vs Python3



	C++	Python3
명칭	set, map	set, dictionary
헤더 파일    모듈	# include <set> # include <map>	List처럼 built-in class이므로, 별도의 import 문 필요 없음
선언	set<자료형> s; map<자료형(key), 자료형(value)> m;	s = set() d = dict()
구조	균형 이진 트리	해시
검색	.find() 메소드를 통해 반복자를 확인	in 키워드로 값이 있는지 판단 x in setA
시간복잡도	$O(\log N)$	$O(1)$
자동 정렬	O	X
랜덤 접근	X	X
삽입	s.insert(new_element); d[key] = value	s.add(new_element) d[key] = value

# C++(set, map) vs Python3(set, dictionary)

## 서로 다른 구조

- C++은 set, map의 구조가 Red-Black Tree이므로 검색, 삽입, 삭제에서 시간복잡도가  $O(\log N)$
- Python3는 set, dictionary의 구조가 해싱이므로 검색, 삽입, 삭제에서 시간복잡도가  $O(1)$
- 따라서 Python3의 set과 dictionary는 정렬 X
- C++에서도 해싱 구조인 unordered\_set, unordered\_map STL이 존재 (자동 정렬 x)

## /<> 10867번 : 중복 빼고 정렬하기 - Silver 5

```
import sys
input = sys.stdin.readline

n = int(input())
num_set = set(map(int, input().split()))
num_list = list(num_set)
num_list.sort()

for i in num_list:
    print(i, end=' ')
```

입력을 바로 set에 넣어 중복을 제거  
정렬을 위해 list에 집어 넣기  
오름차순 정렬



## /<> 1620번 : 나는야 포켓몬 마스터 이다솜 - Silver 4

### 문제

- 포켓몬의 이름(string)이 입력되면 해당 포켓몬의 번호를 출력
- 포켓몬의 번호(int)가 입력되면 해당 포켓몬의 이름을 출력

### 제한 사항

- 도감에 수록되어 있는 포켓몬의 수의 범위는  $1 \leq N \leq 100,000$
- 맞춰야 하는 문제의 개수의 범위는  $1 \leq M \leq 100,000$
- 포켓몬의 이름은 첫 글자가 대문자이며 길이가 20이하인 영어 문자열

예제 입력1

26	5	Caterpie	Spearow
Bulbasaur		Metapod	Fearow
Ivysaur		Butterfree	Ekans
Venusaur		Weedle	Arbok
Charmander		Kakuna	Pikachu
		Beedrill	Raichu
Charmeleon		Pidgey	25
Charizard		Pidgeotto	Raichu
Squirtle		Pidgeot	3
Wartortle		Rattata	Pidgey
Blastoise		Raticate	Kakuna

예제 출력1

Pikachu
26
Venusaur
16
14

## /<> 2002번 : 추월 - Silver 1

### 문제

- 차의 목록(string)이 터널에 들어간/나온 순서대로 주어진다.
- 터널 내부에서 반드시 추월했을 차가 몇 대인지 출력

### 제한 사항

- 차의 대수의 범위는  $1 \leq N \leq 1,000$
- 차량 번호는 영어 대문자와 숫자로 이루어진 중복 없는 6 ~ 8글자의 문자열

예제 입력1

4  
ZG431SN  
ZG5080K  
ST123D  
ZG206A  
ZG206A  
ZG431SN  
ZG5080K  
ST123D

예제 출력1

1

예제 입력2

5  
ZG5080K  
PU305A  
RI604B  
ZG206A  
ZG232ZF  
PU305A  
ZG232ZF  
ZG206A  
ZG5080K  
RI604B

예제 출력2

3

예제 입력3

5  
ZG206A  
PU234Q  
OS945CK  
ZG431SN  
ZG5962J  
ZG5962J  
OS945CK  
ZG206A  
PU234Q  
ZG431SN

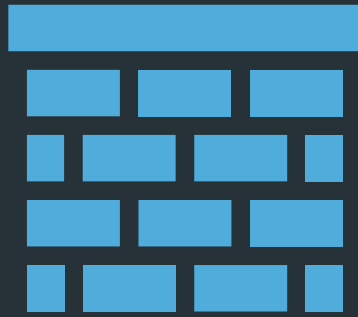
예제 출력3

2

## Hint

1. 각각의 차가 들어간 순서를 숫자로 나타내면 보기 쉽지 않을까요?
2. 'A'차와 'B'차가 있을 때, A가 B를 추월했음을 어떻게 알까요?

# 문제를 간단하게 바꿔 볼게요



ZG431SN



ZG5080K

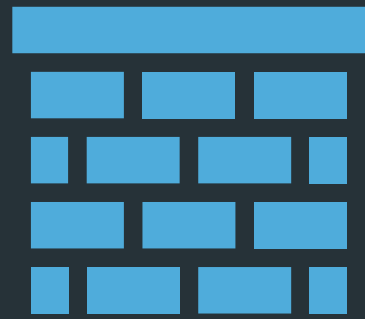


ST123D



ZG206A

# 문제를 간단하게 바꿔 볼게요



1



2

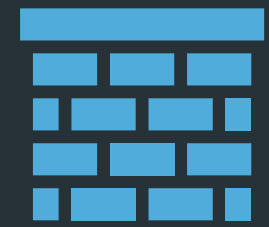


3



4

# 문제를 간단하게 바꿔 볼게요



ZG431SN



ZG5080K



ST123D



ZG206A



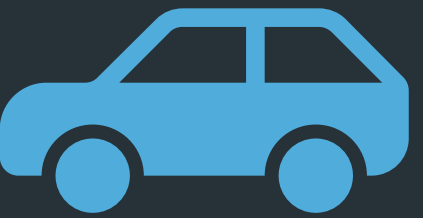
ZG206A



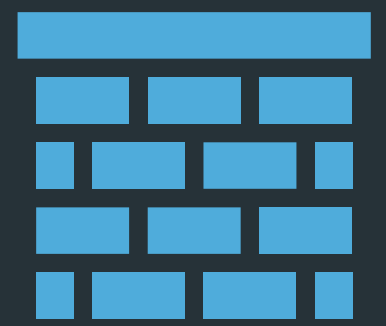
ZG431SN



ZG5080K



ST123D

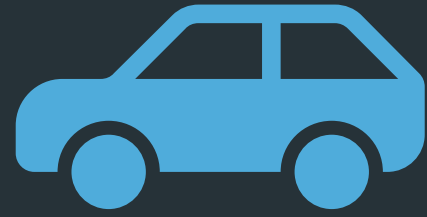




# 문제를 간단하게 바꿔 볼게요



4



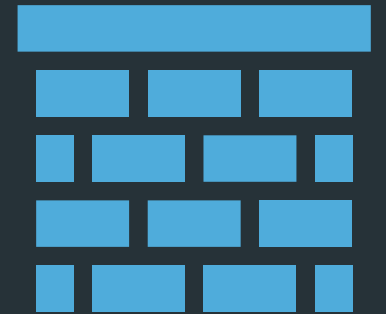
1



2



3



# 아마 터널 안에서는...



1



2

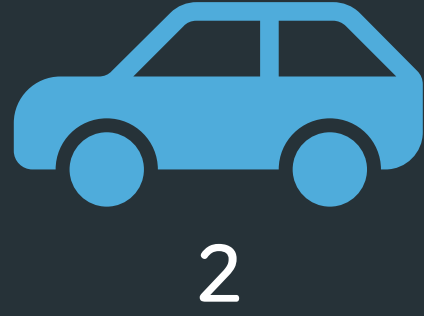
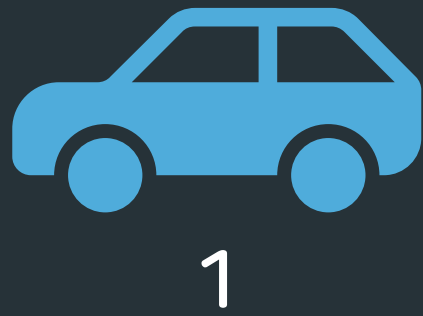


3

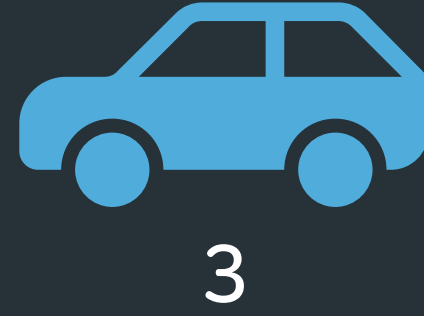
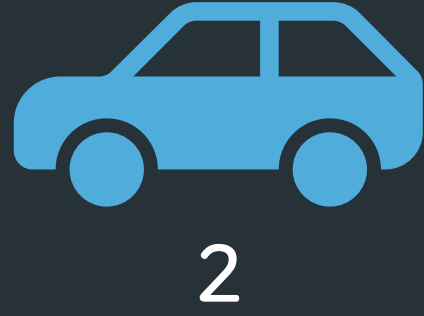
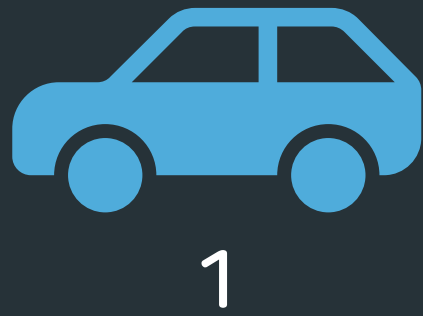


4

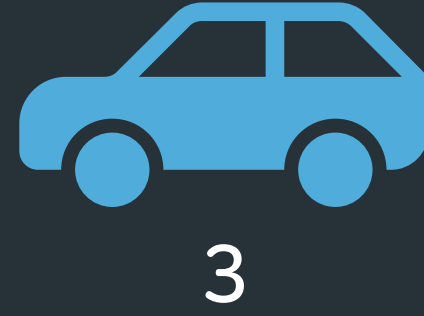
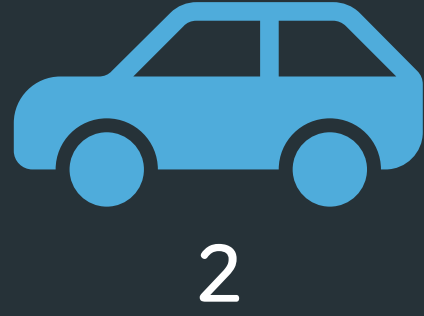
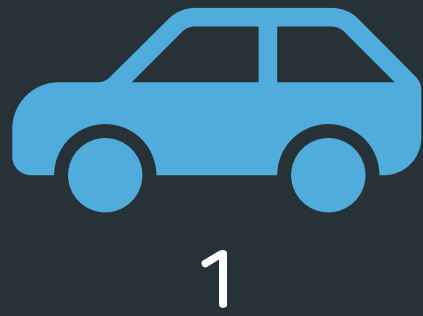
# 아마 터널 안에서는...



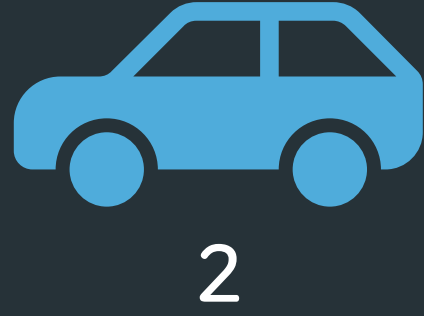
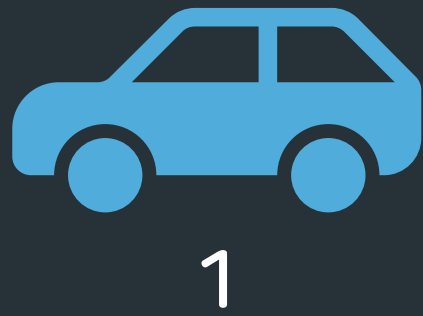
# 아마 터널 안에서는...



# 아마 터널 안에서는...



# 아마 터널 안에서는...



A보다 터널에서 늦게 나온 차 중에서  
A보다 인덱스가 작은 차가 하나라도 있다면  
먼저 들어왔는데 나올때 A보다 뒤에 있다는 것이므로  
A는 터널안에서 추월을 했다!

## 정리

- 연관 컨테이너(Set, Map)은 검색에 최적화된 자료구조
- 내부 구조는 BST에서 발전된 형태인 Red-Black Tree
- 따라서 C++의 Set, Map은 검색, 삽입, 삭제에서 시간복잡도  $O(\log N)$
- C++은 기본적으로 key값을 중복없이 정렬된 상태로 저장하지만, 정렬 없이 중복저장 하는 방법도 있음
- Python3의 set, dictionary는 해시 구조이므로 검색, 삽입, 삭제에서 시간복잡도  $O(1)$
- Set과 Map에 저장된 데이터를 순회하기 위해서는 반복자 (iterator)를 사용해야 함



## 이것도 알아보세요!

- BST와 Red-Black Tree의 **차이**는 뭡까요?
- BST에서 데이터를 **삭제**하기 위해선 어떻게 해야 할까요?
- 해시에서 Bucket의 크기를 **소수**로 하는 이유는 무엇일까요?
- 다음 코드의 **실행 결과**는?

```
#include <iostream>
#include <map>

using namespace std;

int main() {
    map<string, int> m;
    int a = m["no_key"];
    cout << a;
}
```

1. 컴파일 에러
2. 런타임 에러
3. 오류 없음 (그렇다면 출력 결과는?)

```
d = dict()
print(d["no key"])
```

1. 런타임 에러 (그렇다면 어떤 에러?)
2. 오류 없음 (그렇다면 출력 결과는?)