

# 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++) { //result에 arr[i]가 없다면 삽입
        if (find(result.begin(), result.end(), arr[i]) == result.end())
            result.push_back(arr[i]);
    }
    sort(result.begin(), result.end()); //정렬
}
```

시간 복잡도면에서도 효율적이지 않고, 코드도 길다.

## Set

- 다양한 자료형의 데이터 저장 (key)
- key 값을 중복 없이 저장
- key 값을 정렬된 상태로 저장
- 검색, 삽입, 삭제에서의 시간 복잡도는  $O(\log n)$
- 랜덤한 인덱스의 데이터에 접근 불가

# 셋으로 다시 구현해봅시다!

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

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]);
}
```



# 랜덤한 인덱스에 접근 불가?



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

using namespace std;

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

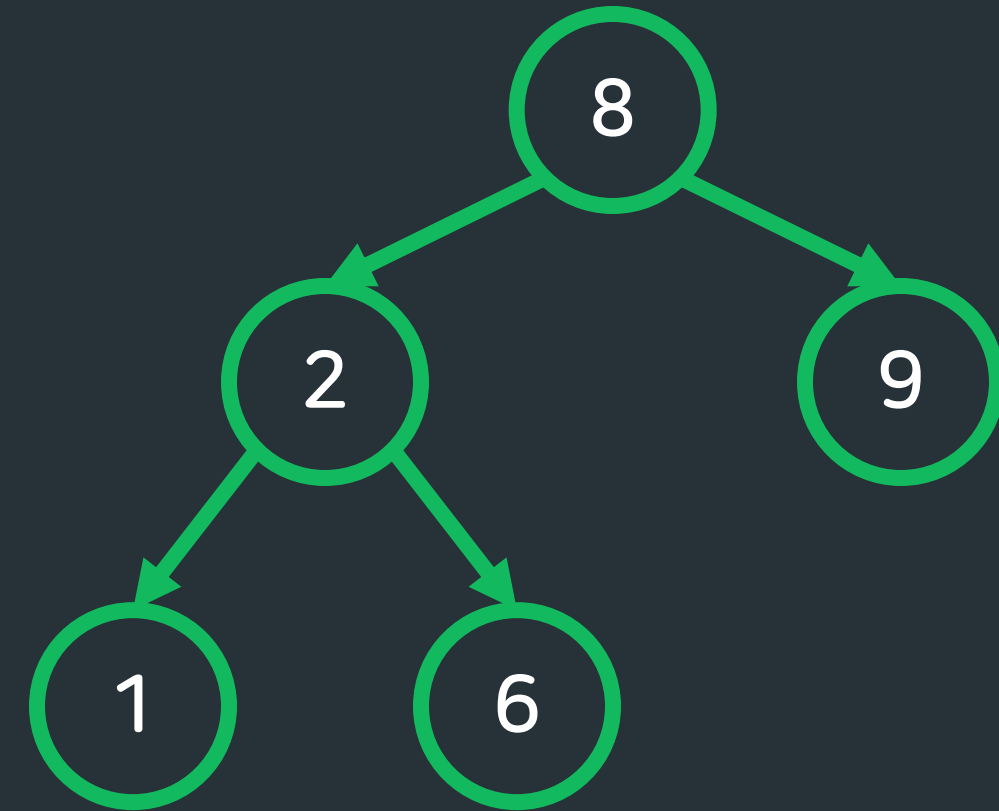
    vec.push_back(2);
    vec.push_back(1);
    s.insert(2);
    s.insert(1);

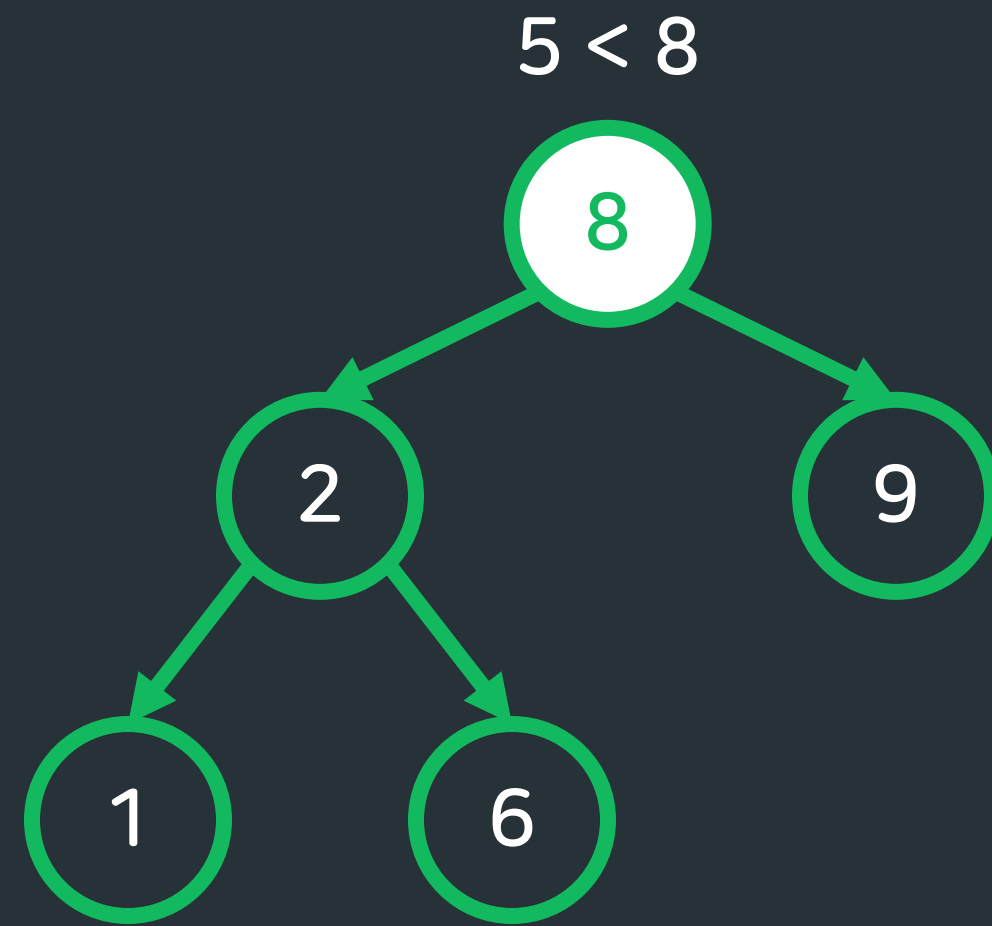
    int a = vec[0]; //가능
    int b = s[0];  //불가능
}
```

## 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

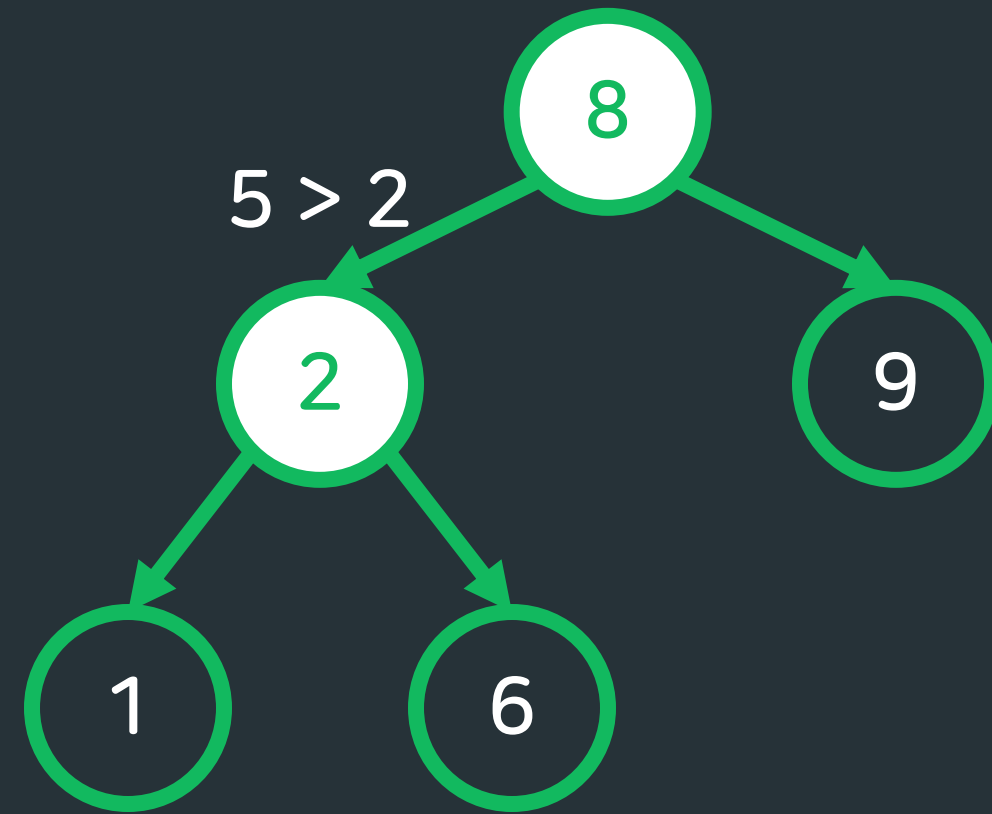
\* 사실 정확히 말하면 여기서 발전된 형태인 red-black tree를 사용



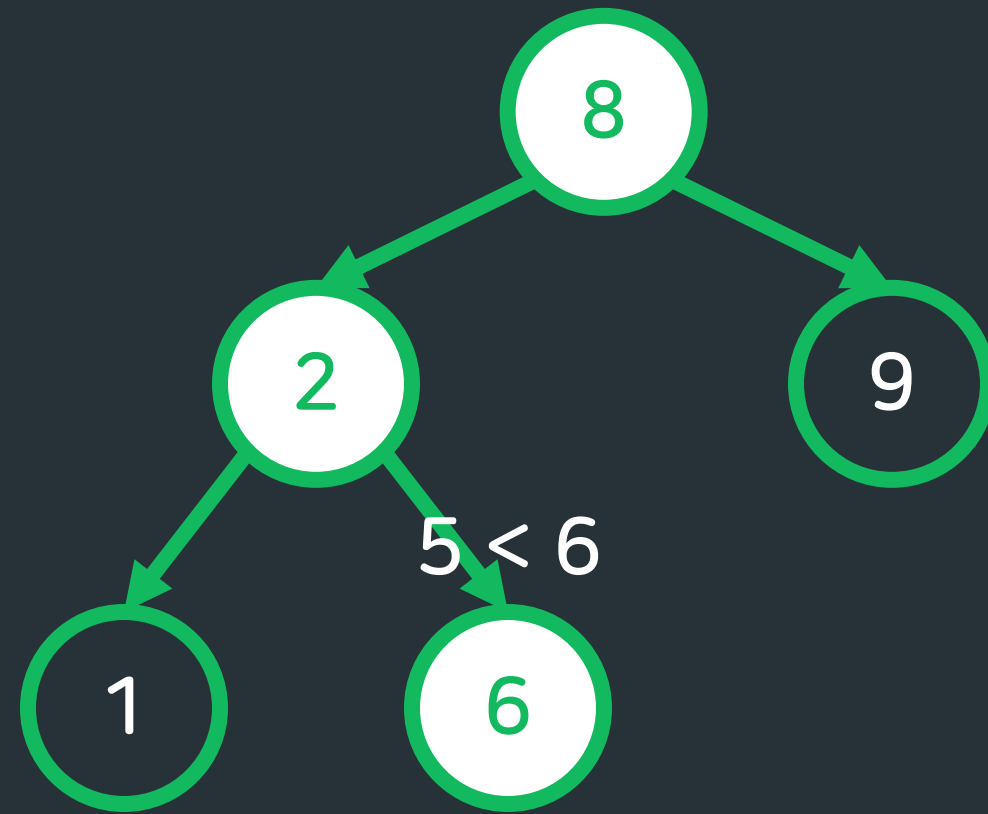


key = 5

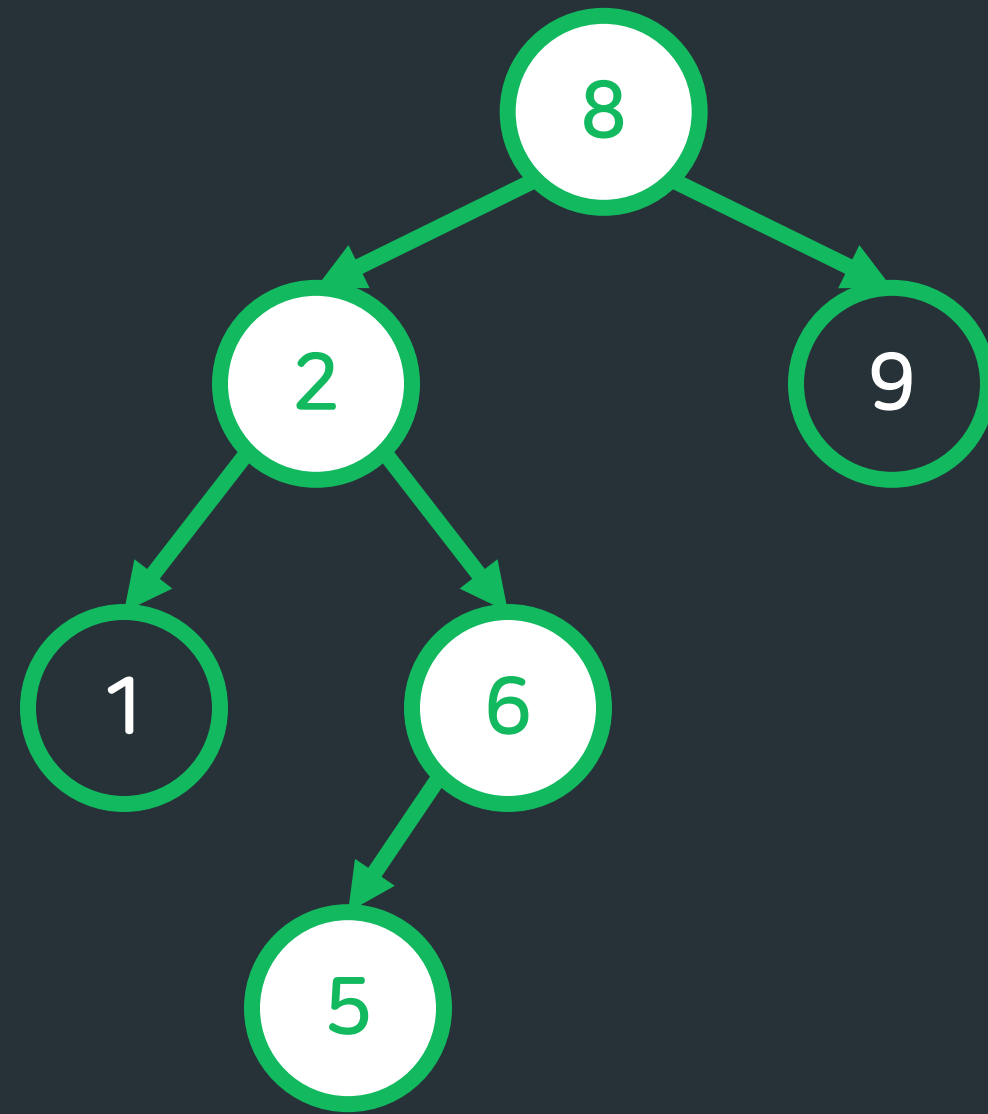




key = 5



key = 5



key = 5

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

using namespace std;

int main() {
    set<int> s;
    s.insert(2);
    s.insert(1);

    set<int>::iterator iter; //포인터와 비슷한 개념
    for (iter = s.begin(); iter != s.end(); iter++)
        cout << *iter << ' ';
}
```

# 낯설어하실 것 같아서 벡터로도 준비했어요



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

using namespace std;

int main() {
    vector<int> vec;
    vec.push_back(2);
    vec.push_back(1);

    vector<int>::iterator iter; //포인터와 비슷한 개념
    for (iter = vec.begin(); iter != vec.end(); iter++)
        cout << *iter << ' ';
}
```

# 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 << ' ';

    for (auto iter:s) //향상된 for문
        cout << iter << ' ';
}
```

## /<> 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  $-1,000 \leq k \leq 1,000$

### 예제 입력

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

### 예제 출력

```
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({"kim", 100});
    student.push_back({"lim", 75});

    string target = "lim";
    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 **중복 없이** 저장
- key 값을 **정렬된 상태**로 저장
- 검색, 삽입, 삭제에서의 시간 복잡도는  $O(\log n)$
- **랜덤한 인덱스**의 데이터에 **접근 불가**

# 맵으로 다시 구현해봅시다!



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

using namespace std;

int main() {
    map<string, int> student;
    student["lee"] = 42;
    student["kim"] = 100;
    student["lim"] = 75;

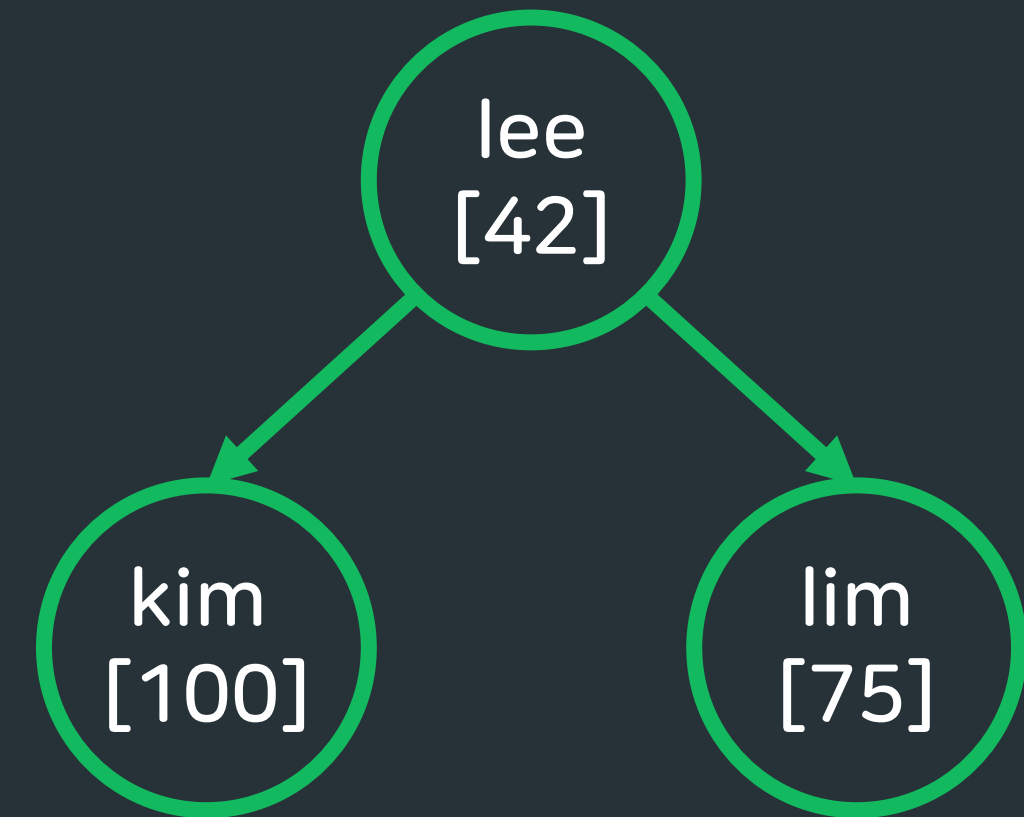
    string target = "lim";
    cout << student[target];
}
```



## BST (Binary Search Tree)

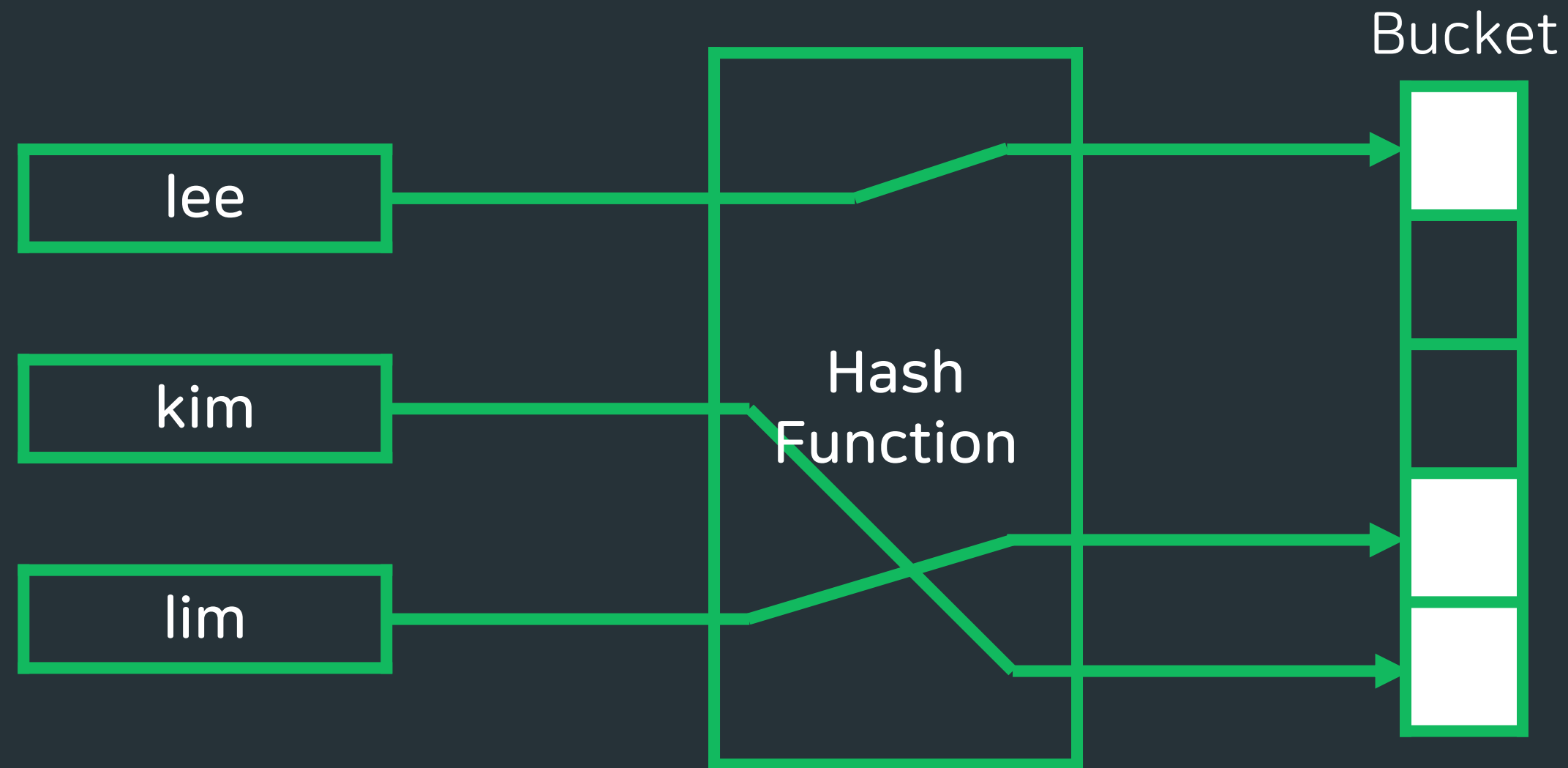
- 하나의 parent(root)에 최대 2개의 child가 있음
- 부모의 왼쪽 서브 트리 값들은 모두 부모 노드보다 작음
- 부모의 오른쪽 서브 트리 값들은 모두 부모 노드보다 큼

\* 사실 정확히 말하면 여기서 발전된 형태인 red-black tree를 사용



## 해싱 (Hashing)

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



\* Bucket은 배열과 유사한 개념

“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”

\* Bucket의 크기는 소수 (prime number)로 하는 것을 권장

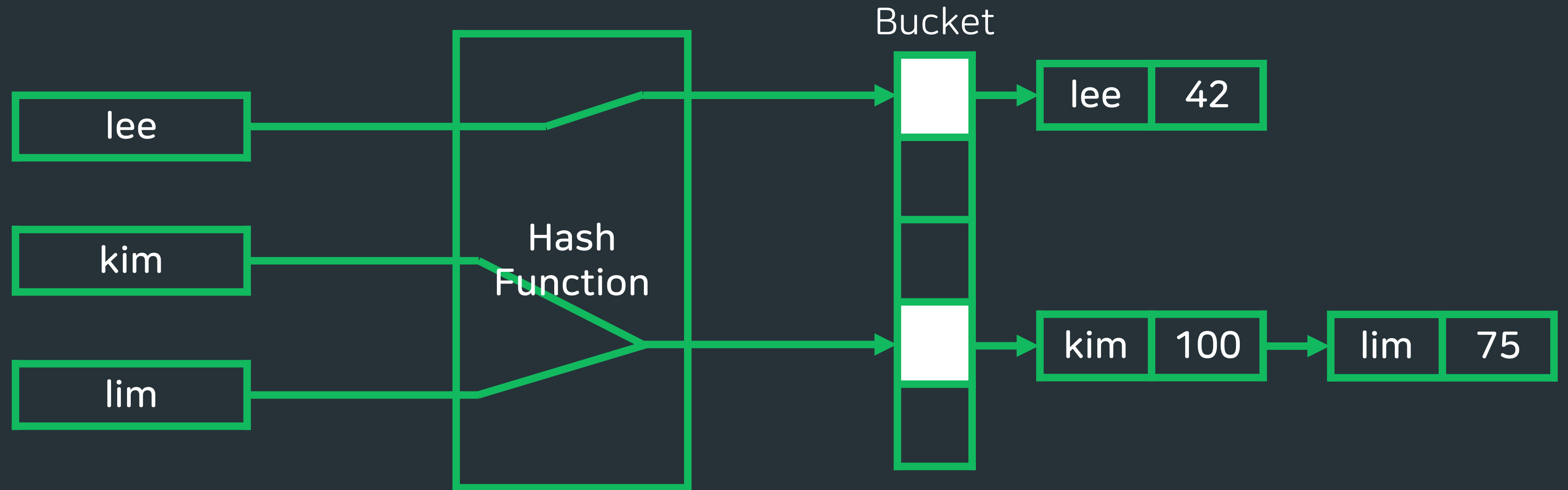
# 서로 다른 key의 해시 함수 결과값이 겹친다면?

## 충돌 (Collision)

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



# 체이닝 (Chaining)



체이닝도 결국 하나하나 찾게 되는건데?

중앙 도서관에서 '별들 사이'라는 책 찾아와 어디 있는지는 모르겠어  
-> 중앙 도서관에서 '별들 사이'라는 책 찾아와 SF 분류 책장에 있어



## /<> 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 입력

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

## 예제 출력

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 입력

4  
ZG431SN ZG5080K ST123D ZG206A  
ZG206A ZG431SN ZG5080K ST123D

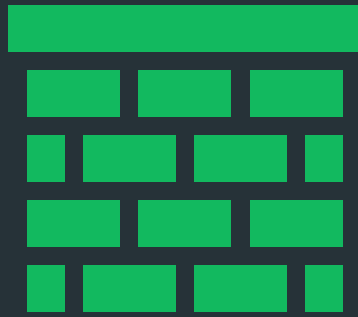
## 예제 출력

1

## 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



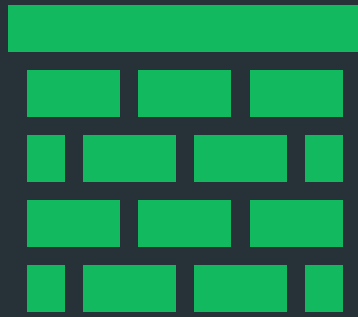
ST123D



ZG206A



# 문제를 간단하게 바꿔 볼게요



1



2



3



4

# 문제를 간단하게 바꿔 볼게요



ZG206A



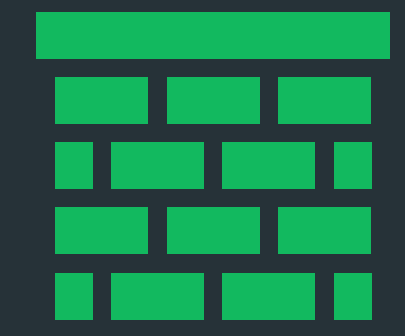
ZG431SN



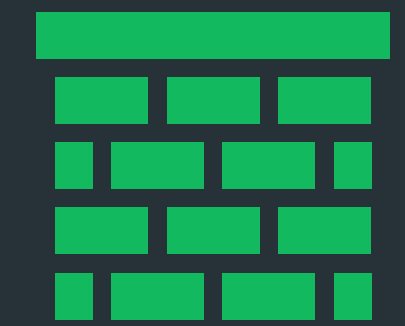
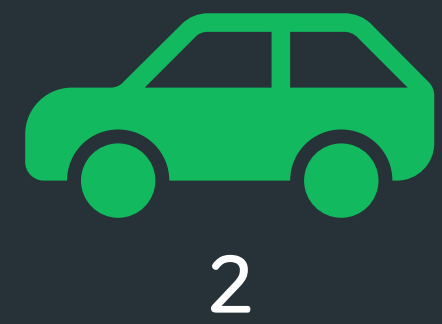
ZG5080K



ST123D



# 문제를 간단하게 바꿔 볼게요



# 아마 터널 안에서는...



1



2



3



4

# 아마 터널 안에서는...



# 아마 터널 안에서는...



# 아마 터널 안에서는...



1



2



3



4

# 아마 터널 안에서는...



1



2



3



4



A보다 터널에서 늦게 나온 차 중에서  
A보다 인덱스가 작은 차가 하나라도 있다면  
A는 터널안에서 추월을 했다!

## 정리

- 연관 컨테이너(Set, Map)은 검색에 최적화된 자료구조
- 내부 구조는 BST에서 발전된 형태인 Red-Black Tree
- 기본적으로 key값을 중복없이 정렬된 상태로 저장하지만, 정렬 없이 중복저장 하는 방법도 있음
- Set과 Map에 저장된 데이터를 순회하기 위해서는 반복자 (iterator)를 사용해야 함

## 이것도 알아보세요!

- BST와 Red-Black Tree의 차이는 뭡까요?
- BST에서 데이터를 삭제하기 위해선 어떻게 해야 할까요?
- 해시에서 Bucket의 크기를 소수로 하는 이유는 무엇일까요?
- 오른쪽 코드의 실행 결과는?
  1. 컴파일 에러
  2. 런타임 에러
  3. 오류 없음 (그렇다면 출력 결과는?)

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

using namespace std;

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

## 필수

- /<> 19636번 : 요요 시뮬레이션 - Silver 5
- /<> 10757번 : 큰 수 A+B - Bronze 5

## 3문제 이상 선택

- /<> 18870번 : 좌표 압축 - Silver 2
- /<> 1764번 : 듣보잡 - Silver 4
- /<> 9375번 : 패션왕 신해빈 - Silver 3
- /<> 4358번 : 생태학 - Gold 5
- /<> 2015번 : 수들의 합 4 - Gold 5
- /<> 14425번 : 문자열 집합 - Silver 3