2025 겨울방학 알고리즘 스터디

집합과 맵

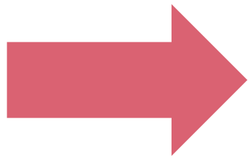
컴퓨터 공학과 20230546 서보경

목차

집합 이란?

집합(Set)이란 **고유한 값만**을 저장하고, 중복을 허용하지 않는 자료구조이다.
내부적으로 이진 탐색 트리 혹은 해시 테이블로 구성이 되어있다.

```
set<int>s;
for (int i = 1; i <= 10; ++i)
{
    s.insert(i);
}
for (int i = 1; i <= 15; ++i)
{
    s.insert(i);
}
cout << format("Set 안에 들어가 있는 요소들₩n");
for (auto& i : s)
{
    cout << i << ' ';
}
```



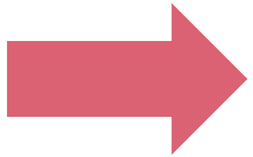
```
Set 안에 들어가 있는 요소들
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

각 Set에 원소를 집어넣었을 때 중복을 제외 하고 모두 들어가 있는 모습이다.

맵 이란?

맵(Map)이란 **키-값 관계**를 저장하고, 각 **키가 고유한** 자료구조이다.
내부적으로 이진 탐색 트리 혹은 해시 테이블로 구성이 되어있다.

```
map<int, int>mm;
for (int i = 1; i <= 10; ++i)
{
    mm.insert({ i,1 });
}
for (int i = 1; i <= 15; ++i)
{
    mm[i]++;
}
cout << format("Map 안에 들어가 있는 요소들₩n");
for (auto& i : mm)
{
    cout << format("키 : {} , 값 : {}₩n", i.first, i.second);
}
```



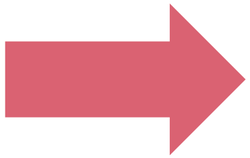
```
Map 안에 들어가 있는 요소들
키 : 1 , 값 : 2
키 : 2 , 값 : 2
키 : 3 , 값 : 2
키 : 4 , 값 : 2
키 : 5 , 값 : 2
키 : 6 , 값 : 2
키 : 7 , 값 : 2
키 : 8 , 값 : 2
키 : 9 , 값 : 2
키 : 10 , 값 : 2
키 : 11 , 값 : 1
키 : 12 , 값 : 1
키 : 13 , 값 : 1
키 : 14 , 값 : 1
키 : 15 , 값 : 1
```

각 Map에 원소를 집어넣었을 때 키를 기준으로 데이터가 삽입되며,
값을 바꾸는 연산자도 사용할 수 있다.

트리 기반의 집합과 맵

트리 기반의 집합은 **레드-블랙 트리** 혹은 **AVL 트리** 기반으로 이루어진 자료구조이다.
C / Swift / Python등에서는 지원하지 않는다. (내부 라이브러리 기준)

```
set<int>s;
for (int i = 10; i >= 1; --i)
{
    s.insert(i);
}
for (int i = 15; i >= 1; --i)
{
    s.insert(i);
}
cout << format("set 안에 들어가 있는 요소들₩n");
for (auto& i : s)
{
    cout << i << ' ';
}
```



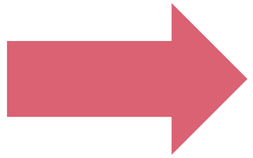
```
set 안에 들어가 있는 요소들
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

데이터 기반으로 정렬된 상태로 유지시키는 자료구조이다.
시간복잡도는 삽입, 삭제, 검색 모두 O(log n)이다.

트리 기반의 집합과 맵

트리 기반의 맵은 **레드-블랙 트리** 혹은 **AVL 트리** 기반으로 이루어진 자료구조 이다.
C / Swift / Python등에서는 지원하지 않는다. (내부 라이브러리 기준)

```
map<int, int>mm;
for (int i = 10; i >= 1; --i)
{
    mm.insert({ i,1 });
}
for (int i = 15; i >= 1; --i)
{
    mm[i]++;
}
cout << format("map 안에 들어가 있는 요소들₩n");
for (auto& i : mm)
{
    cout << format("키 : {} 값 : {}₩n", i.first, i.second);
}
```



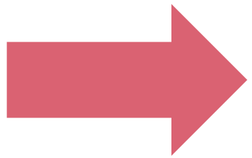
```
map 안에 들어가 있는 요소들
키 : 1 값 : 2
키 : 2 값 : 2
키 : 3 값 : 2
키 : 4 값 : 2
키 : 5 값 : 2
키 : 6 값 : 2
키 : 7 값 : 2
키 : 8 값 : 2
키 : 9 값 : 2
키 : 10 값 : 2
키 : 11 값 : 1
키 : 12 값 : 1
키 : 13 값 : 1
키 : 14 값 : 1
키 : 15 값 : 1
```

키를 기반으로 정렬된 상태를 유지시키는 자료구조이다.
시간복잡도는 삽입, 삭제, 검색 모두 O(log n)이다.

해시 기반의 집합과 맵

해시 기반의 집합은 주로 **해시테이블** 기반으로 이루어진 자료구조 이다.
C 에서는지원하지 않는다. (내부 라이브러리 기준)

```
unordered_set<int>s;
for (int i = 10; i >= 1; --i)
{
    s.insert(i);
}
for (int i = 1; i <= 15; ++i)
{
    s.insert(i);
}
cout << format("Set 안에 들어가 있는 요소들₩n");
for (auto& i : s)
{
    cout << i << ' ';
}
```



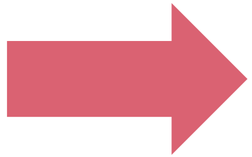
```
Set 안에 들어가 있는 요소들
10 9 8 7 6 5 4 3 2 1 11 12 13 14 15
```

정렬되지 않은 상태로 데이터를 보관하는 자료구조이다.
대신 시간복잡도는 삽입, 삭제, 검색 모두 **O(1)**이다.

해시 기반의 집합과 맵

해시 기반의 맵은 **해시테이블** 기반으로 이루어진 자료구조 이다.
C 에서는지원하지 않는다. (내부 라이브러리 기준)

```
unordered_map<int, int>mm;
for (int i = 10; i >= 1; --i)
{
    mm.insert({ i,1 });
}
for (int i = 1; i <= 15; ++i)
{
    mm[i]++;
}
cout << format("map 안에 들어가 있는 요소들₩n");
for (auto& i : mm)
{
    cout << format("키 : {} 값 : {}₩n", i.first, i.second);
}
```



```
map 안에 들어가 있는 요소들
키 : 10 값 : 2
키 : 9 값 : 2
키 : 8 값 : 2
키 : 7 값 : 2
키 : 6 값 : 2
키 : 5 값 : 2
키 : 4 값 : 2
키 : 3 값 : 2
키 : 2 값 : 2
키 : 1 값 : 2
키 : 11 값 : 1
키 : 12 값 : 1
키 : 13 값 : 1
키 : 14 값 : 1
키 : 15 값 : 1
```

또한 키를 정렬되지 않은 상태로 데이터를 보관하는 자료구조이다.
시간복잡도는 삽입, 삭제, 검색 모두 **O(1)** 이다.

해시는 무적일까?

결론부터 이야기 하면, 안정성이 낮아서 무적이 아니다.

1 - 해시충돌

서로 다른 데이터가 동일한 해시값을 가질 수 있는 가능성은 항상 존재한다.
충돌이 많아지면 해시 테이블의 성능이 O(1)에서 최악 O(n)으로 저하된다.
최근 온라인 대회에서는 출제진이 저격 데이터를 넣어 해시테이블을 터뜨리는 추세이다. (시간초과 유도)

2 – 해시 함수의 품질 의존성

품질이 낮은 해시 함수는 데이터를 고르게 분배하지 못해 성능이 낮아지는 반면,
품질이 좋은 해시 함수는 설계와 계산 비용(CPU)이 추가로 들어간다.

3 – 범위 및 정렬 연산의 부적합성

해시는 정렬된 데이터나 범위 기반의 검색이 필요한 문제에는 적합하지 않는다.
예를 들어, x 이상 y 이하의 값이 몇 개 있는지 찾는 문제는 트리 자료구조가 적합.

https://codeforces.com/blog/entry/62393

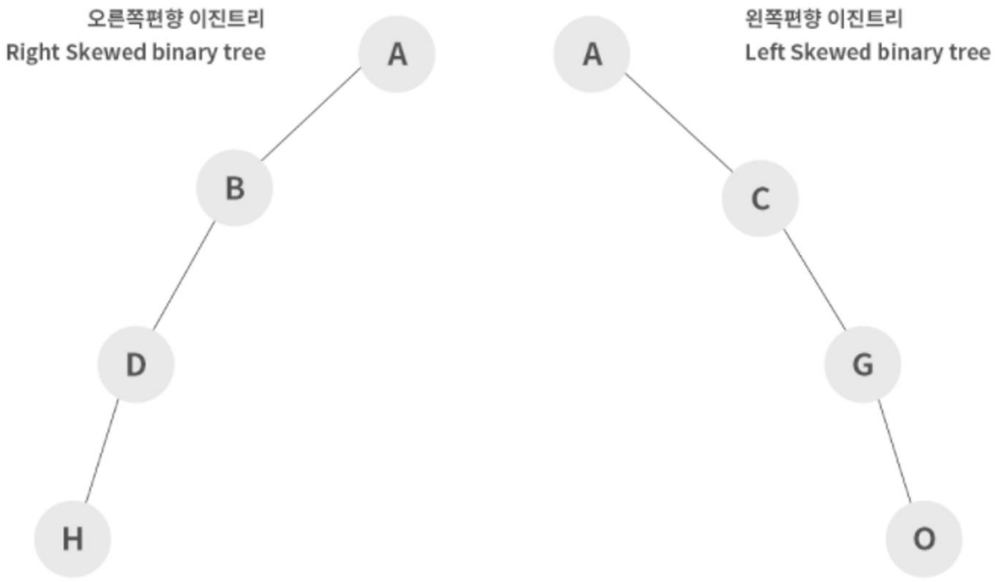
C++ unordered_map 터뜨리기

https://codeforces.com/blog/entry/98994

파이썬 dict 터뜨리기

트리맵이 지원이 안된다면?

코딩테스트에선 해시 충돌을 일으킬 데이터를 넣을 가능성이 낮으니 단순 데이터 보관을 위해서는 해시테이블을 사용하면 됨. 그러나 트리 맵이 필요한 경우에는 대처하기가 매우 어려움.



이진 검색 트리를 직접 구현하게 된다면
어느 정도 트리맵과 비슷하게 만들 수 있다.
그러나 이런 방식에는 치명적인 단점이 있는데,
데이터의 삽입 순서에 따라 트리가 한쪽으로 치우칠 수 있다.
이 치우침은 편향 트리문제를 야기하며 트리의 높이가 증가하게 되고, 성능이 O(log n)에서 **O(n)**으로 저하될 수 있다.
C++과 JAVA에서는 **레드-블랙 트리**를 사용해서 트리의 높이가 한쪽으로 치우치지 않도록 자기 균형을 유지한다. (회전 혹은 색상 변경)
이러한 자기 균형 트리는 구현하기가 매우 힘들다.