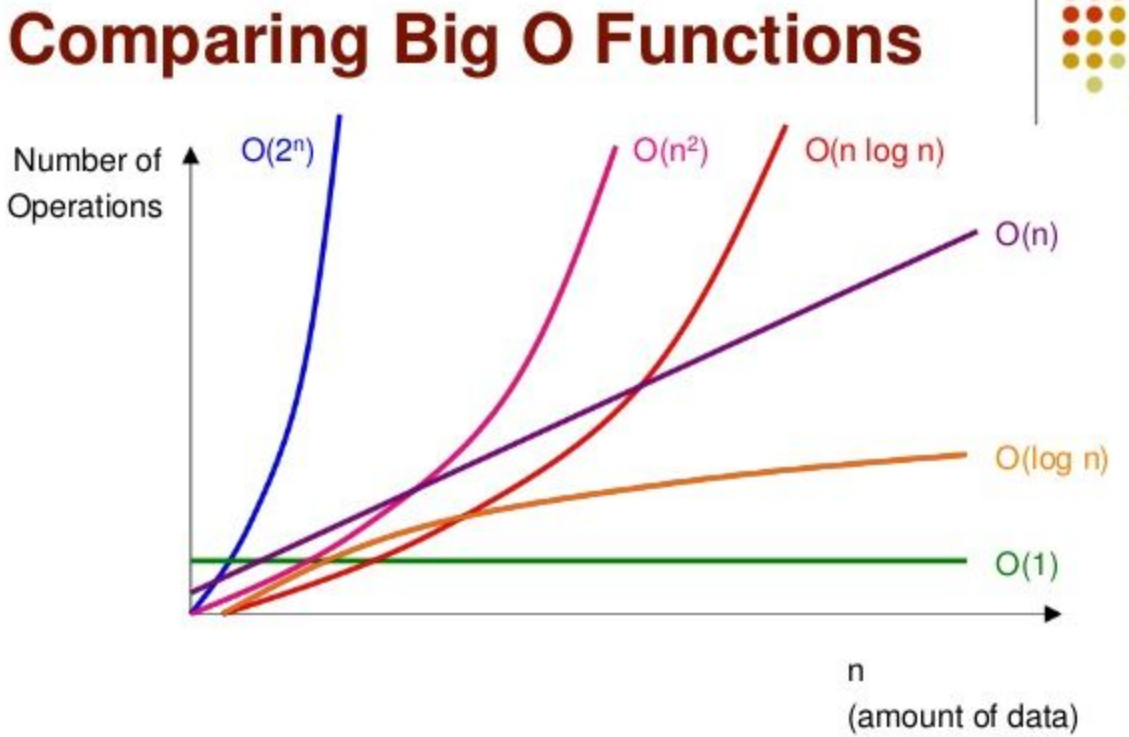
DSC - Algorithm

Algorithm with C++

맘스터치 생각난다

지난 주 이야기 했던 내용


Comparing Big O Functions
Number of Operations
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n)
O(1)
n
(amount of data)

STL이란?

C++의 표준 템플릿 라이브러리(Standard Template Library)의 약자

일반적으로 많이 사용되는 자료구조와 알고리즘의 모음 라이브러리

템플릿(Template)를 이용하여 제작 - Generic Programming

반복자(iterator) 통해 원소에 접근 가능

Ex) vector<int>::iterator it = v.begin();

다룰 내용

Stack

Queue

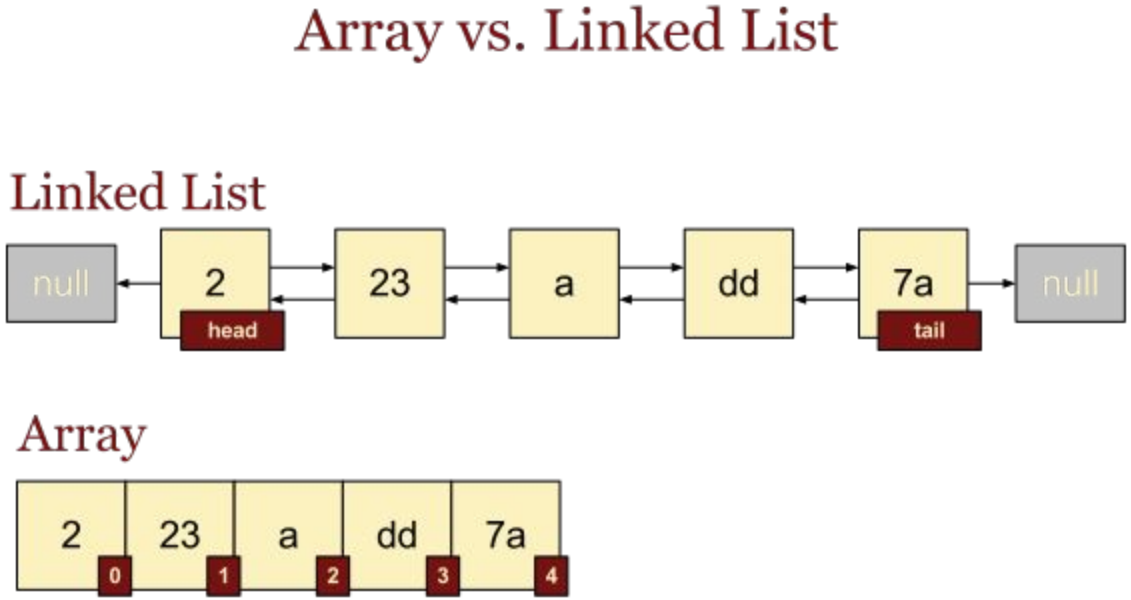
List

Vector

Map

Set

그에 앞서

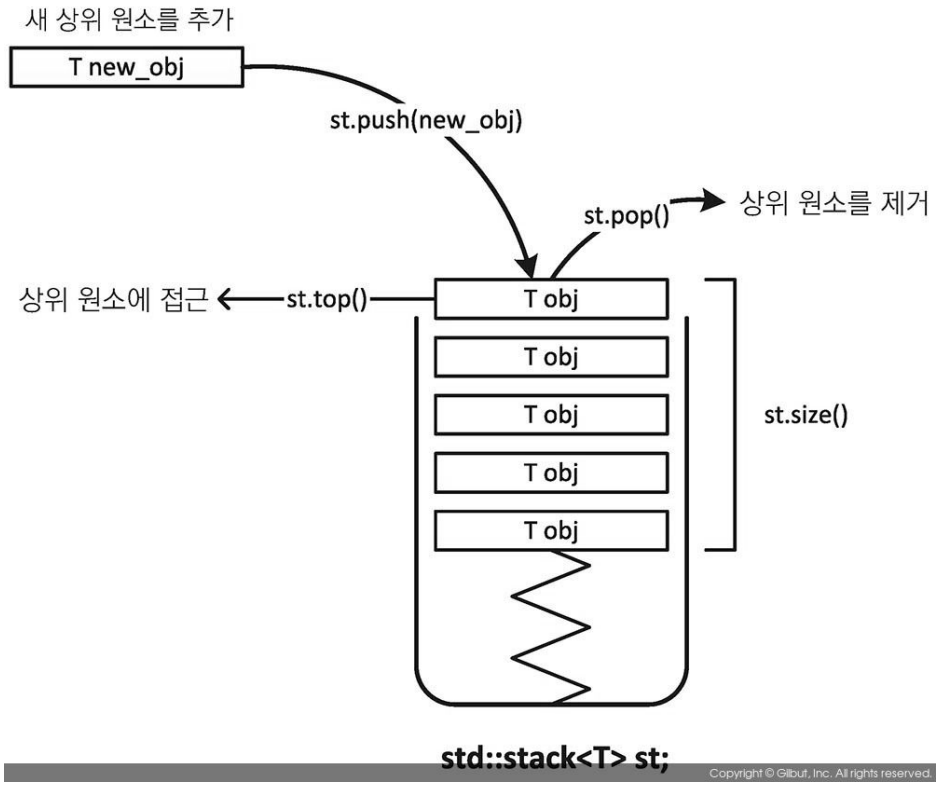

Array vs. Linked List
Linked List
null
2
head
23
a
dd
7a
tail
null
Array
2
0
23
1
a
2
dd
3
7a
4

Stack

한 쪽 끝에서만 자료를 넣고 뺄 수 있는 구조

새로운 자료를 넣으면 맨 위에 올라가고, 뺄 때도 맨 위부터

LIFO(Last In First Out) 구조



stack<type> name : type형 스택 name을 생성

push(element) : top에 원소를 추가

pop() : top에 있는 원소를 삭제

top() : top(스택의 처음이 아닌 가장 끝)에 있는 원소를 반환

empty() : 스택이 비어있으면 true 아니면 false를 반환

Stack - sample code

```
#include <iostream>
#include <stack>
using namespace std;

int main()
{
    stack<int> s;

    s.push(2);
    s.push(1);

    cout << "top : " << s.top() <<endl;

    s.pop(); // 1이 삭제

    cout << "size : " << s.size() <<endl;

    cout << "empty : " << (s.empty() ? "Yes" : "No") <<endl;

    return 0;
}
```

Queue

한 쪽 끝에선 추가만 가능하고, 다른 한 쪽 끝에선 빼는 것만 가능

먼저 들어온 데이터가 먼저 나가는 구조

FIFO(First In First Out)구조

queue<type> name : type형 큐 name을 생성

push(element) : 큐에 원소를 추가(뒤에)

pop() : 큐에 있는 원소를 삭제(앞에)

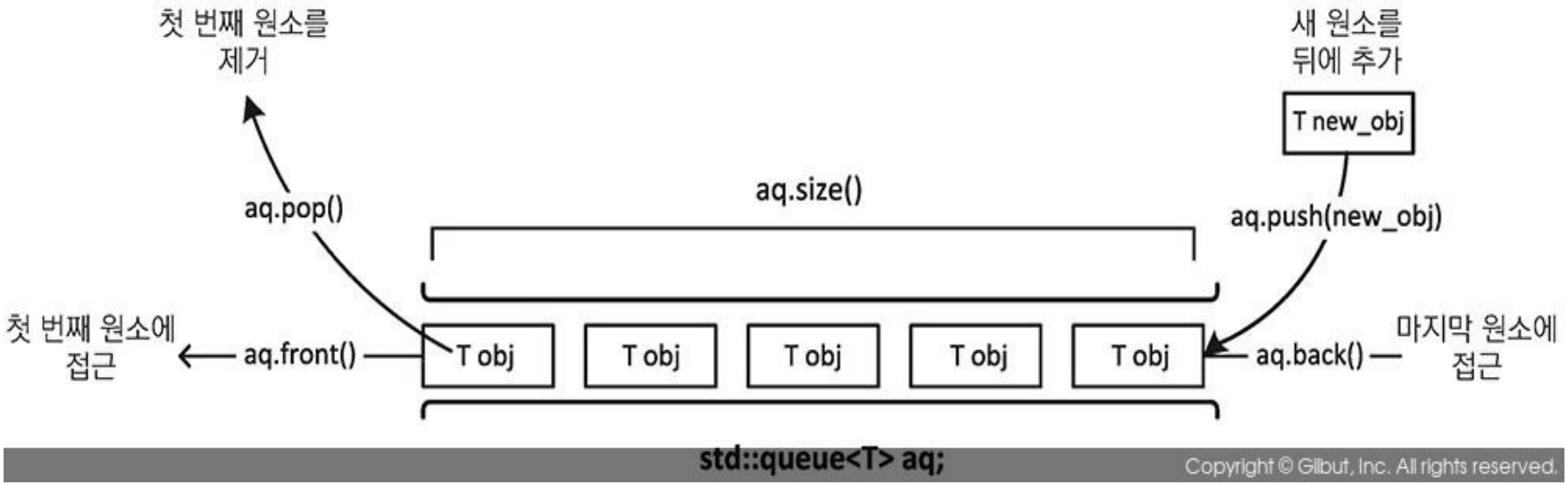
front() : 큐 제일 앞에 있는 원소를 반환

back() : 큐 제일 뒤에 있는 원소를 반환

empty() : 큐가 비어있으면 true 아니면 false를 반환

size() : 큐 사이즈를 반환

Queue



Queue - sample code

```
#include <iostream>
#include <queue>
using namespace std;

int main()
{
    queue<int> q;

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

    q.pop();

    cout << "front : " << q.front() <<endl;

    cout << "back : " << q.back() <<endl;

    cout << "size : " << q.size() <<endl;

    cout << "empty : " << (q.empty() ? "Yes" : "No") <<endl;

    return 0;
}
```

List

Node형 구조

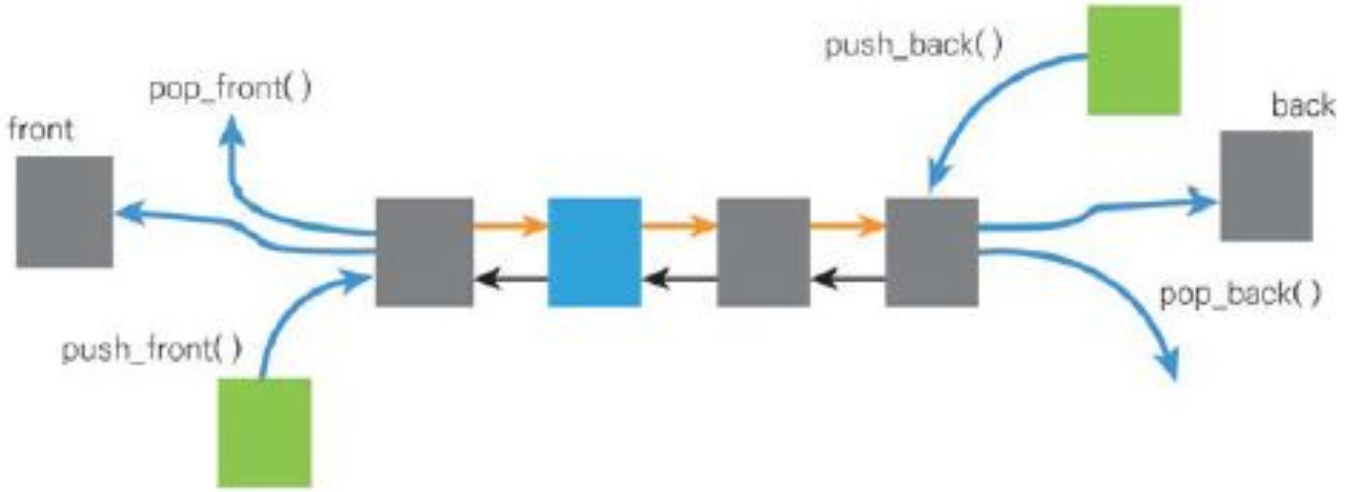
가변적인 구조에 유리(중간 삽입/삭제 용이)

검색이 많은 경우 사용 지양

랜덤 데이터 접근이 효율적이지 못함

list<type> name : type형 리스트 name 생성

List



List - 자주 쓰는 기능

멤버	설명
begin	첫 번째 위치를 가리킨다
end	마지막 위치의 다음을 가리킨다
rbegin	역 방향으로 첫 번째 위치를 가리킨다
rend	역 방향으로 마지막 위치를 가리킨다
push_front	첫 번째 위치에 데이터 추가
pop_front	첫 번째 위치의 데이터 삭제
push_back	마지막 위치에 데이터 추가
pop_back	마지막 위치의 데이터 삭제
front	첫 번째 데이터의 참조 반환
back	마지막 데이터의 참조 반환
clear	저장하고 있는 모든 데이터 삭제
empty	저장 데이터 유/무. 없으면 true 반환
size	저장하고 있는 개수 반환
insert	지정된 위치에 삽입
erase	지정된 범위에 있는 데이터 삭제
remove	지정된 값과 일치하는 모든 데이터 삭제
remove_if	predicate에 만족하는 모든 데이터 삭제
sort	데이터들을 정렬한다.

List - sample code

```
#include <iostream>
#include <list>
using namespace std;

int main(){

    list<int> lt;

    lt.push_back(10);
    lt.push_back(20);

    list<int>::iterator iter;
    for (iter = lt.begin(); iter != lt.end(); ++iter){
        cout << *iter << ' ';
    }
    cout << endl;

    // 리스트에서 원소 10 제거
    lt.remove(10);
    it.pop_back();

    return 0;
}
```

Vector

Array구조

배열 처럼 사용(배열 길이 다 쓰면 알아서 늘려 줌)

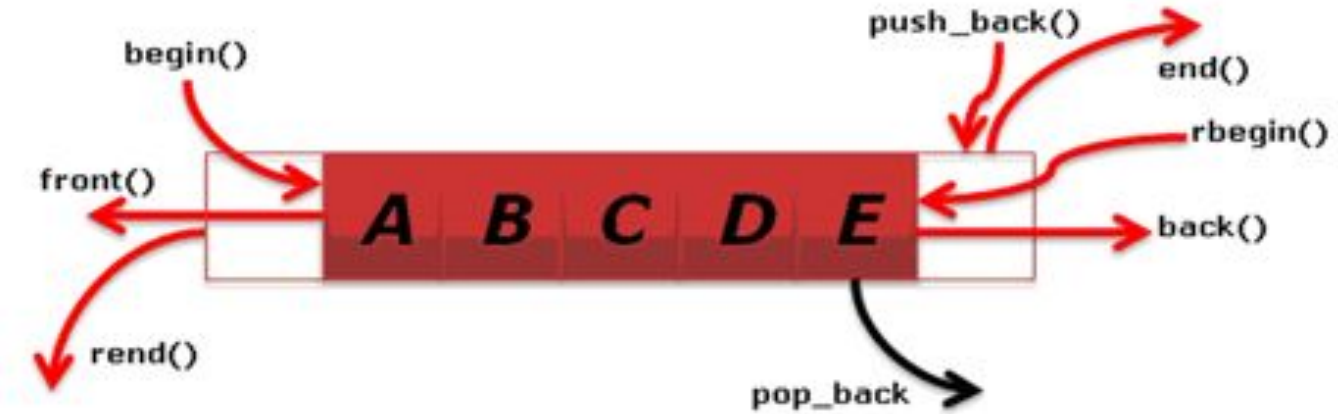
랜덤 접근은 가능하지만, 임의의 위치에 삽입/삭제는 어려움

vector<type> name : type형 vector name을 생성

vector<type> name(num) : type형 vector name을 생성 + num만큼 미리 할당

vector<type> name(num1, num2) type형 vector name을 생성 + num만큼 미리 할당 하고 num2로 초기화

Vector



Vector - 자주쓰는 기능

멤버	설명
at	특정 위치의 원소의 참조를 반환
back	마지막 원소의 참조를 반환
begin	첫 번째 원소의 랜덤 접근 반복자를 반환
clear	모든 원소를 삭제
empty	아무것도 없으면 true 반환
end	마지막 원소 다음의(미 사용 영역) 반복자를 반환
erase	특정 위치의 원소나 지정 범위의 원소를 삭제
front	첫 번째 원소의 참조를 반환
insert	특정 위치에 원소 삽입
pop_back	마지막 원소를 삭제
push_back	마지막에 원소를 추가
reserve	지정된 크기의 저장 공간을 확보
size	원소의 개소를 반환

Vector - sample code

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    vector<int> v(5);
    v.push_back(15);
    for(auto it = v.begin(); it != v.end() ; it++)
    {
        cout<<*it<<endl;
    }

    v.pop_back();

    v[0] = 10;
    for(int i =0; i < v.size(); i++)
    {
        cout<<v[i]<<endl;
    }

    return 0;
}
```

Vector 와 List 비교

	vector	List
구조	Array구조	Node구조
크기 변경 가능	O	O
중간 삽입, 삭제 용이	X	O
순차 접근 가능	O	O
랜덤 접근 가능	O	X

중간 삽입/삭제가 많을 땐 List, 랜덤 접근이 많을 땐 Vector 권장

Map

Key-value 구조로 구성

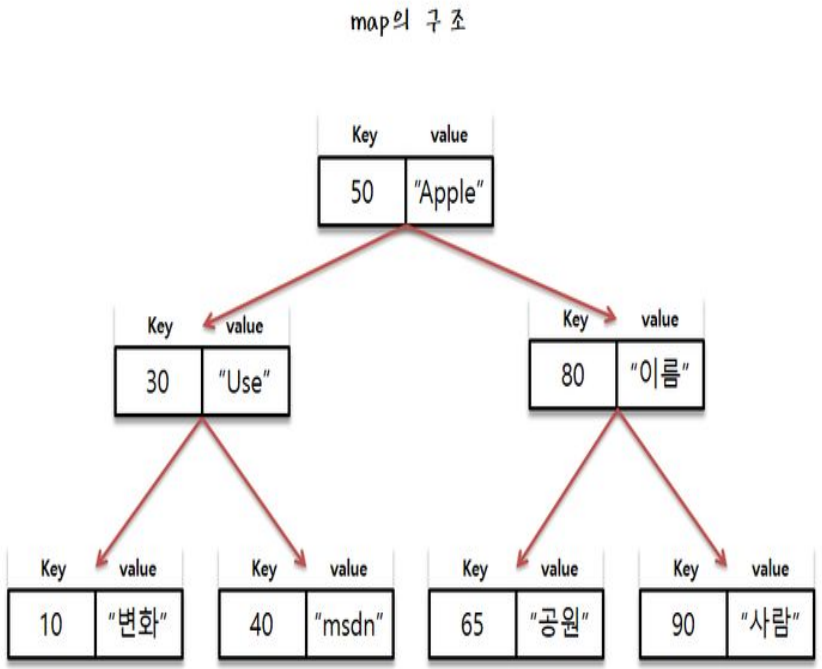
이진탐색 트리(rb-tree)

정렬 해야 할 때

많은 자료를 저장하고, 검색이 빨라야 할 때

중간에 삽입/삭제가 빈번하지 않을 때

map<type1, type2> name : type1을 Key로, type2를 Value로 사용하는 map, name을 생성



Map - 자주 쓰는 기능

멤버	설명
begin	첫 번째 원소의 랜덤 접근 반복자를 반환
clear	저장하고 있는 모든 원소를 삭제
empty	저장 하고 있는 요소가 없으면 true 반환
End	마지막 원소 다음의(미 사용 영역) 반복자를 반환
erase	특정 위치의 원소나 지정 범위의 원소들을 삭제
Find	key와 연관된 원소의 반복자 반환
insert	원소 추가
lower_bound	지정한 key의 요소를 가지고 있다면 해당 위치의 반복자를 반환
operator[]	지정한 key 값으로 원소 추가 및 접근
rbegin	역방향으로 첫 번째 원소의 반복자를 반환
rend	역방향으로 마지막 원소 다음의 반복자를 반환
size	원소의 개수를 반환
upper_bound	지정한 key 요소를 가지고 있다면 해당 위치 다음 위치의 반복자 반환

Map - sample code

```
#include <iostream>
#include <map>
using namespace std;

int main()
{
    map<int, int> m;

    m.insert(pair<int, int>(5, 100));
    m[1] = 10;

    map<int,int>::iterator it;
    for(it = m.begin(); it != m.end(); it++)
    {
        cout << it->first << " " << it->second<<endl;
    }

    it = m.find(5);
    if (it != m.end())
        cout << it->first << " " << it->second<<endl;

    m.erase( m.begin(), m.end() );

    return 0;
}
```

Set

map에서 Value가 없는, Key만 있는 구조

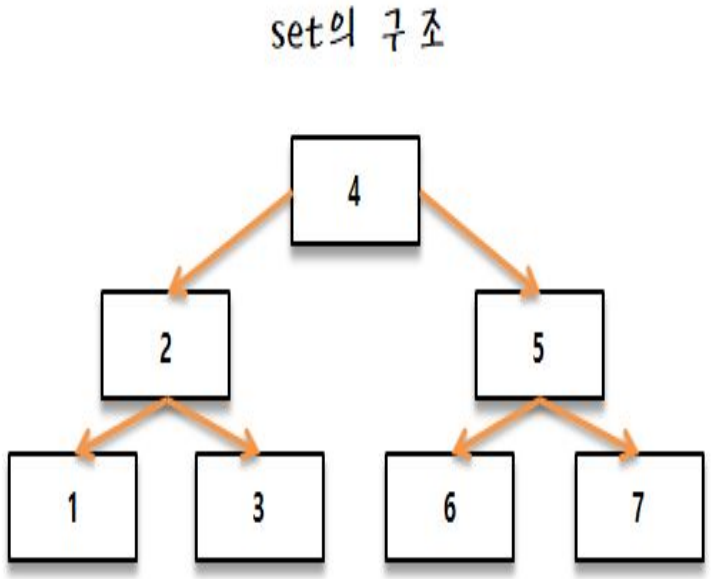
이진탐색 트리 구조

정렬 해야 할 때

Key가 있는지 없는지 알아야할 때

많은 자료를 저장하고, 검색속도가 빨라야할때

set<type> name : type형 set, name 생성



Set - 자주쓰는 기능

멤버	설명
begin	첫 번째 원소의 랜덤 접근 반복자를 반환
clear	저장하고 있는 모든 원소를 삭제
empty	저장 하고 있는 요소가 없으면 true 반환
end	마지막 원소 다음의(미 사용 영역) 반복자를 반환
erase	특정 위치의 원소나 지정 범위의 원소들을 삭제
find	key와 연관된 원소의 반복자 반환
insert	원소 추가
lower_bound	지정한 key의 요소를 가지고 있다면 해당 위치의 반복자를 반환
operator[]	지정한 key 값으로 원소 추가 및 접근
rbegin	역방향으로 첫 번째 원소의 반복자를 반환
rend	역방향으로 마지막 원소 다음의 반복자를 반환
size	원소의 개수를 반환
upper_bound	지정한 key 요소를 가지고 있다면 해당 위치 다음 위치의 반복자 반환

Set - sample code

```
#include <iostream>
#include <set>
using namespace std;

int main()
{
    set<int> s;

    s.insert(2);
    s.insert(3);

    set<int>::iterator it;
    for (it = s.begin(); it != s.end(); it++)
        cout << *it <<endl;

    it = s.find(3);
    if (it != s.end())
        cout << *it << "는 있다" << endl;

    s.erase(3);

    return 0;
}
```

이번주의 숙제

백준

- 10828번 - 스택
- 10845번 - 큐

프로그래머스

- 스킬체크 해보기 (https://programmers.co.kr/skill_checks)

Q&A