C++: `std::vector`를 포인터로 반환하기

알고리즘및실습

1. 개요

- Howsum 문제와 Bestsum 문제는 전수조사 함수가 원시 타입을 반환하는 것이 아니라 리스트를 반환해야 한다. 이때 해가 없는 경우와 있는 경우를 구분할 수 있어야 한다. 자바나 파이썬은 기본적으로 참조 타입 변수를 사용하며, 동적 생성한 것을 직접 반납할 필요가 없으므로 리스트 반환을 구현하는 특별한 어려움이 없다. 하지만 C++는 직접 반납해야 하므로 간단하지 않다. 크게 다음과 같은 3가지 방법이 있다.
  - 방법 1. 동적 생성한 **`std::vector`**를 반환하고, 적절하게 **`delete`**를 해준다.
  - 방법 2. 스마트 포인터를 사용한다.
  - 방법 3. 동적 생성하지 않고 값으로 반환한다.
- 방법 1은 적절한 위치에서 **`delete`**하는 것이 어려울 수 있다.
- 방법 2는 고급 C++ 개발자가 선택해야 하는 방법이다. 하지만 스마트 포인터에 익숙하지 않으면 사용하기 어려울 수 있다.
- 방법 3은 해가 없는 경우와 있는 경우를 구분해야 한다.
- 각 방법을 다음 절에서 각각 좀 더 구체적으로 살펴보자.

2. 방법 1. 동적 생성

- **`solve`** 메소드의 서명은 다음과 같은 형태를 사용한다.

```
std::vector<int>* solve(int M, const std::vector<int>& nums);
```

- `M < 0`인 경우와 `M == 0`인 경우는 다음과 같이 각각 반환하면 된다.

```
1 if(M < 0) return nullptr;
2 if(M == 0) return new std::vector<int>{};
```

- 여기까지는 직관적이다. 문제는 **`delete`**이다.
- Howsum은 하나의 해만 찾으면 되며, 첫 번째 찾은 해를 반환하기 때문에 여러 개를 동적 생성하지 않는다. 따라서 **`main`**에서 답을 출력한 후에 **`delete`**하면 된다.
- Bestsum은 모든의 해를 만들기 때문에 최종해만 **`delete`**하면 안 된다. **`solve`**에서 조건문으로 반납된 것과 지금까지 후보해를 비교해야 하는데, 이때 진 것은 **`delete`**해 주어야 한다. 참고로 포인터가 **`nullptr`**일 때 **`delete`**를 하여도 문제가 없으므로 **`delete`**하기 전에 포인터가 **`nullptr`**인지 여부를 검사하지 않아도 된다.

3. 방법 2. 스마트 포인터

- 스마트 포인터는 크게 3 종류가 있지만 두 문제 모두 특정 공간을 가리키는 여러 개의 스마트 포인터를 만들 필요가 없으므로 **`std::unique_ptr`**를 사용하면 된다.
- **`solve`** 메소드의 서명은 다음과 같은 형태를 사용한다.

```
std::unique_ptr<std::vector<int>> solve(int M, const std::vector<int>& nums);
```

- `M < 0`인 경우와 `M == 0`인 경우는 다음과 같이 각각 반환하면 된다.

```
if(M < 0) return nullptr;
if(M == 0) return std::make_unique<std::vector<int>>();
```

- 원래 스마트 포인터로 동적 생성한 것을 처리할 때 자동 반납을 위해 유지는 스마트 포인터로 하지만 사용할 때는 다음과 같이 일반 포인터로 사용할 수 있다.

```
class A{
    std::make_unqiue<std::vector<int>> nums;
    //
public:
    //
    const std::vector<int>& getNums() const{
        return *nums;
    }
    //
}
```

- 하지만 이 문제에서는 동적 생성한 데이터를 객체의 멤버 변수로 유지하는 것이 아니므로 다음과 같이 프로그래밍할 수 없다.

```
std::vector<int>* solve(int M, const std::vector<int>& nums){
    if(M < 0) return nullptr;
    if(M == 0) return std::make_unique<std::vector<int>>().get();
    //
}
```

- 스마트 포인터를 사용하였으므로 반납에 대한 걱정 없이 프로그래밍하면 된다. 주의할 것은 값으로 반환해야 하며, 값으로 반환하며 자동으로 이동 생성자를 통해 복사가 아니라 필요한 이동이 이루어진다. 또 반환하는 것을 받을 때 복잡한 타입 대신 다음과 같이 **auto**를 사용하면 간결하게 프로그래밍할 수 있다.

```
auto result{solve(M, nums}};
```

4. 방법 3. 값 반환

- 이 방법은 리스트를 값으로 반환해야 하기 때문에 무거울 것으로 생각할 수 있으나 RVO 최적화 기술 때문에 생각한 것과 달리 성능에 큰 문제는 없다.
- **solve** 메소드의 서명은 다음과 같은 형태를 사용한다.

```
std::vector<int> solve(int M, const std::vector<int>& nums);
```

- 문제는 해가 있는 경우와 없는 경우를 구분할 수 있어야 한다. 이를 위해 절대 해에 포함할 수 없는 값을 해에 포함해 반납할 수 있다.
- `M < 0`인 경우와 `M == 0`인 경우는 다음과 같이 각각 반환하면 된다.

```
if(M < 0) return std::vector<int>{-1};
if(M == 0) return std::vector<int>{};
```

이 경우에도 무언가 최적화를 하겠다고 다음과 같이 반환할 필요가 없다.

```
return std::move(std::vector<int>{-1});
```

std::move를 사용하지 않으면 보통은 RVO 최적화를 할 것이고, 이 최적화를 하지 않아도 자동으로 이동 생성자를 이용하여 임시 객체를 생성한다.

- 반환 받은 **std::vector**가 -1를 가지고 있으면 해가 없는 경우에 해당한다. 이것의 검사는 다음과 같이 할 수 있다.

```
if(curr.empty() || curr.back() != -1)
```

빈 리스트이거나 마지막 요소가 -1이 아니면 해가 반환된 경우이다.