완전탐색

'완전 탐색'은 컴퓨터의 빠른 연산속도를 이용하여 **발생 가능한 모든 경우의 수를 확인하는 방법**이다. 대개 입력으로 주어지는 수가 작은 경우 쉽게 사용할 수 있다. 다만, 완전 탐색은 시간이 오래 걸리는 경우가 많으니 완전 탐색으로 구현하기 전 시간복잡도를 먼저 계산해보는 것이 좋다.

완전 탐색을 하기 위해 사용할 수 있는 방법은 크게 두 가지로 나뉘는데, 첫 번째는 **for 문과 if 문을 사용하는 반복문**이고 두 번째는 **재귀 함수**이다. 반복문은 프로그래밍 언어를 공부할 때 빠질 수 없는 필수 개념이다. 그렇기 때문에 기본적인 프로그래밍 공부를 한 사람이라면 누구나 알고 있으며, 완전 탐색을 구현할 때도 어려움 없이 사용 가능하다. 다만, 재귀 함수에 비해 구현할 수 있는 범위가 매우 제한적인 경우도 있다. 재귀 함수는 함수 안에서 함수 자기 자신을 다시 호출하는 함수를 의미이다. 예를 들어 다음 코드와 같은 모양을 가진다.

```c
#include <stdio.h>

int count_down(int second){
  if(second==0){   //기저사례 (재귀함수의 끝)
    return 0;
  }
  else{
    printf("%d초 ", second);
    count_down(second-1); // 함수 안의 함수 사용: 재귀함수
  }
}

int main(){
  count_down(5);

  return 0;
}
```

```
프로세스가 시작되었습니다.(입력값을 직접 입력해 주세요)
> 5초 4초 3초 2초 1초
프로세스가 종료되었습니다.
```

정지

재귀 함수를 사용할 때는 **기저 사례를 설정**해줘야 하며, **재귀 함수의 깊이가 너무 깊어지지 않도록 주의**해야한다. 반복문도 조건문을 잘못 작성하면 무한루프를 돌 듯, 재귀 함수도 이를

탈출할 수 있는 기저 사례를 설정해주지 않으면 무한루프에 빠지게 된다. 재귀 함수의 깊이는 메모리와 관련이 있는데, 함수가 호출될 때 스택 메모리를 사용하여 함수와 관련된 데이터를 저장해놓기 때문이다. 이때 저장하는 데이터의 양이 너무 많아진다면 스택 오버플로우가 발생할 수 있다. 따라서 너무 많은 함수 호출이 예상되는 경우에는 함수의 호출 횟수를 제한하거나 다른 방식으로 구현하는 등의 접근 방법이 필요하다. 온라인으로 코드를 제출했을 때 코드가 오답으로 처리된 이유가 메모리 초과나 런타임 에러 등과 같은 메시지로 나타난다면 메모리 문제를 한 번쯤은 의심해볼 만하다.

재귀 함수는 추후 다룰 DFS 나 백트래킹 기법 등을 연습할 수 있는 좋은 방법 중 하나이니 용어와 쓰임에 대해 다시 한번 기억해두자!