chapter9. 동적 프로그래밍

- Hello Coding 그림으로 개념을 이해하는 알고리즘 -

TABLE OF CONTENTS

1. 동적 프로그래밍

dp, dynamic programming이라고 불리는 동적 프로그래밍은 메모리를 적절히 사용하여 수행시간 효율성을 비약적으로 향상시키는 방법입니다.

큰 문제를 작은 문제로 나눠서 푸는 알고리즘입니다.

메모이제이션 기법을 이용해 이미 계산된 결과(작은 문제)를 별도의 메모리에 저장해 다시 계산하지 않고 필요한 경우에 사용하는 방식입니다.

<dp 조건 2가지>

1. **최적 부분 구조**
   - 큰 문제를 작은 문제로 나눌 수 있으며 작은 문제의 답을 모아서 큰 문제를 해결할 수 있다.
2. **중복되는 부분 문제**
   - 동일하게 반복되는 작은 문제로 해결해야한다.

2. 분할 정복, 탐욕법과의 차이

대표적인 분할정복 방법으로 퀵 정렬과 병합 정렬을 떠올려보세요.

각각 분할했던 원소들이 정렬과정에서 다른 원소에 영향을 끼치지 않습니다.

하지만 피보나치 수열만 보더라도 소문제가 상위문제에 사용됨으로 영향을 끼쳐서 독립적이지 않으며 동적 프로그래밍이 적용가능하게 됩니다.

탐욕 알고리즘과의 차이는 탐욕 알고리즘은 각 단계별로 현재 상태에서 가장 최적의 경우를 판단해서 결정하기 때문에 최적해를 구할 수 없을 수도 있지만 **동적 프로그래밍은** 모든 가능성을 고려하므로 **항상 최적의 결과가 도출됩니다.**

3. 중복 연산 방지법 2가지

1. **메모이제이션(하향식)**
   하향식(Top-Down)은 하위 문제에 대한 정답을 계산했는지 확인해가며 문제를 자연스러운 방식으로 풀어나가는 방법입니다.

2. **타뷸레이션(상향식)**
   상향식(Bottom-Up)은 더 작은 하위문제부터 살펴 본 다음 작은 문제의 정답을 이용해서 큰 문제의 정답을 풀어나가는 방법입니다.

4. 최적성의 원리

동적 프로그래밍은 어떤 문제에 적용해야 할까요?

어떤 문제를 보고 우선 이 문제가 **최적성의 원리를 만족하는지 살펴봐야 합니다.**

여기서 **최적성의 원리란 주어진 문제의 부분해가 전체 문제의 해를 구하는데 사용되는지를 의미**합니다.

만약 최적성의 원리가 적용됨을 확인했다면 주어진 문제를 소문제로 분해하여 최적해를 제공하는 점화식을 도출해야 합니다.

이 점화식을 코드로 옮겨 작성하면 됩니다.

5. 동적 프로그래밍 예제

Q. 계단을 오를 때 한 칸 or 두 칸 씩 올라갈 수 있다고 가정할 때 꼭대기까지 올라가는 방법의 수를 구하시오.

1 계단: 1

2 계단: 1+1, 2

3 계단: 1+1+1, 1+2, 2+1

4 계단: 1+1+1+1, 2+2, 2+1+1, 1+1+2, 1+2+1

이전에 구했던 부분해를 전체 문제의 해를 구하는 데 사용함으로 최적성의 원리를 만족합니다.

이를 점화식으로 나타내면, n > 2 일 때, C(n) > C(n-1) + c(n-2)로 나타낼 수 있습니다.

처음 두 항이 숫자가 달라서 그렇지, 피보나치 수하고 점화식이 똑같습니다.

```python
def climbstairs(n):
    c = [0] * (n + 1)
    c[0] = 1
    c[1] = 2
    if n < 3:
        return c[n-1]
    for i in range(2,n):
        c[i] = c[i-1] + c[i-2]
    return c[n-1]

print(climbstairs(4))
```

c는 dp테이블이며 상향식(타뷸레이션)을 사용하였습니다.