Dynamic Programming

- **전체 문제에 대해서 가장 작은 1번 문제에 대한 답을 찾고 이것을 이용하여 2번 문제에 대한 답을 찾고 이것을 이용하여 3번 문제에 대한 답을 찾는다. 이것을 n번까지 반복하여 최종적으로 전체 문제에 대한 답을 구한다.**
- **DP문제는 예시의 첫 번째 원소부터 시작하여 각 원소를 마지막 항일 때로 가정하여 해를 찾는다. 이 과정을 주어진 데이터 전체까지 적용하여 일반화된 식을 찾는다.**
- **Bottom-Up & Top-Down**
  - **네트워크 선 자르기 문제(Bottom-Up 방식)**
    - **선의 길이 만큼에 해당하는 일차원 리스트 선언**
    - 직관적으로 알 수 있는 것은 직접 초기화를 해두고 풀이를 시작한다.
      - ex) 1번 문제에 대한 답, 2번 문제에 대한 답..,
    - ex) 길이가 7인 네트워크 선
      - 길이가 7인 일차원 리스트 dy를 선언
      - dy[1] = 1, dy[2] = 2, ...(직관적으로 알 수 있는 부분은 직접 초기화)
      - dy[3] = ?? -> 길이가 3인 선을 자르는 방법의 개수
        - 선의 가장 오른쪽을 1m로 자른 경우: 남은 부분은 2m -> dy[2] = 2이므로 총 2가지
        - 선의 가장 오른쪽을 2m로 자른 경우: 남은 부분은 1m -> dy[1] = 1이므로 총 1가지
        - 따라서, 총 3가지. dy[3] = 3
  - **네트워크 선 자르기 문제(Top-Down 방식)**
    - **선의 길이 만큼에 해당하는 일차원 or 이차원 리스트 선언**
    - 직관적으로 알 수 있는 것은 직접 초기화를 해두고 풀이를 시작한다.
    - memoization: **재귀 함수 호출**. 구한 값을 초기에 선언한 리스트에 저장하고 다음 값을 구할 때 재귀 함수를 호출하여 리스트에 저장된 값을 이용한다.

- 최대 부분 증가수열(LIS: Longest Increasing Subsequence)

- **ex) 5 3 7 8 6 2 9 4**
- 주어진 숫자들을 저장하고 있는 일차원 리스트 arr를 선언한다.
- 리스트의 첫 번째 원소부터 그 원소가 내가 만들고자 하는 증가수열의 마지막 항이라 가정하고 LIS를 생성한다.
  - 0번 원소(5)가 마지막 항인 경우: LIS – 5. 길이: 1
  - 1번 원소(3)가 마지막 항인 경우: LIS – 3. 길이: 1
  - 2번 원소(7)가 마지막 항인 경우: LIS – ㅁ 7

    ㅁ: 3일 때 – 3이 마지막 항인 경우의 가짓수는 1가지. 따라서, LIS의 길이: 2

    ㅁ: 5일 때 – 5가 마지막 항인 경우의 가짓수는 1가지. 따라서, LIS의 길이: 2
  - 3번 원소(8)가 마지막 항인 경우: LIS - ㅁ 8

    ㅁ: 7일 때 - 2가지. 따라서, LIS의 길이: 3

    ㅁ: 3일 때 - 1가지. 따라서, LIS의 길이: 2

    ㅁ: 5일 때 - 1가지. 따라서, LIS의 길이: 2
- 위 과정을 리스트의 마지막 원소까지 반복한다.
- **DP방식으로 해결하기(위 과정과 연결됨)**
  - 위에서 선언한 리스트 arr의 각 원소를 마지막 항으로 하는 최대 부분 증가수열의 길이를 저장하는 일차원 리스트 dy를 선언한다.
  - 직관적으로 알 수 있는 부분은 직접 초기화하고 시작한다. dy[1] = 1
  - dy[1] = 1, dy[2] = 1, dy[3] = 2, dy[4] = 3, dy[5] = 2 …
  - 위 과정을 주어진 데이터의 개수만큼 반복하여 리스트 dy 전체를 초기화한다.

■ 최대 선 구하기 문제 – 위와 동일ㄴ

■ 가장 높은 탑 쌓기

- 밑면의 넓이가 큰 벽돌이 아래에 위치해야 한다.
- 주어진 데이터를 2차원 리스트 arr에 저장하는데 이때 밑면의 넓이에 대하여 내림차순으로 정렬한다. 벽돌의 무게만 고려하여 문제를 해결해 나가면 된다.

- ◆ 일차원 리스트 dy는 각 인덱스에 해당하는 벽돌이 꼭대기에 위치한 경우 최대 높이를 저장한다.
- ◆ **비교 대상이 여러 개인 경우 비교 대상을 줄이는 방법을 생각해본다. (ex 정렬)**

- 알리바바와 40인의 도둑(Bottom-Up)

- ■ 최단거리 이동(현재 위치에서 오른쪽 or 아래쪽으로만 이동할 수 있음)
- ■ **위에서와 같이 항상 마지막 항의 입장에서 생각하여 해를 구하고 일반화한다.**
- ■ 입력 데이터는 이차원 리스트 형태로 저장
- ■ **2차원 리스트 dy를 선언. 1차원 리스트로 선언하게 되면 사용할 수 있는 정보가 부족함.**

- 알리바바와 40인의 도둑(Top-Down)

- ■ 재귀 함수 DFS를 정의하여 구현
- ■ **memoization을 위해 2차원 리스트 dy를 선언(재귀를 사용하므로 memoization이 필요함)**
- ■ 재귀 함수 DFS를 바로 return하지 말고 dy에 값을 저장한 후에 return한다.
- ■ 최종적으로 구해야 하는 해에 대해서 상태 트리를 그려가면서 문제를 해결한다.