동적 프로그래밍

23.11.27 송이

동적 프로그래밍이란?

기본 접근

- 순차 탐색이 막히면-> 데이터를 정렬해 이진 탐색을 진행
- 배열로 처리하기 어려우면 -> 해시테이블 사용
- 위의 방법들로 시간 복잡도를 줄였다.
- 만약 반드시 사용해야만 하는 개념이 있다면 어떻게 해야 할까?
- 구현해야 할 기능이 특정 범위를 반복적으로 호출하는 구조, 겹쳐지는 구조를 갖는다면
- 해당 범위의 결과를 따로 저장하며 나중에 다시 계산하지 않도록 최적화

바꾸기 어려운 논리를 극한의 최적화를 통해 연산을 줄이는 방법을 동적 프로그래밍이라고 한다.

동적 프로그래밍이란?

완전 탐색의 문제점

- 특정 범위를 반복적으로 호출하거나 조금씩 겹치는 구조라면 가장 먼저 완전 탐색을 생각할 수 있지만 경우의 수가 많아지게 되면 시간이 오래 걸려 사용할 수 없다.
- 아무리 컴퓨터가 사람보다 속도가 빨라도 하나하나 다 따지는 방식은 비효율적이다.

시간 복잡도가 크다

동적 프로그래밍이란

동적 프로그래밍의 조건

조건 최적 부분 구조 중복되는 부분 문제

최적 부분 구조

작은 문제에서 최선의 선택을 했을 때 최종 결과 역시 최선의 선택이 되어야 하는 구조

모든 작은 문제는 독립성을 지녀야 조건을 만족한다.

전체 문제의 답을 부분 문제로 쪼개서 풀 수 있어야 한다.

중복되는 부분 문제

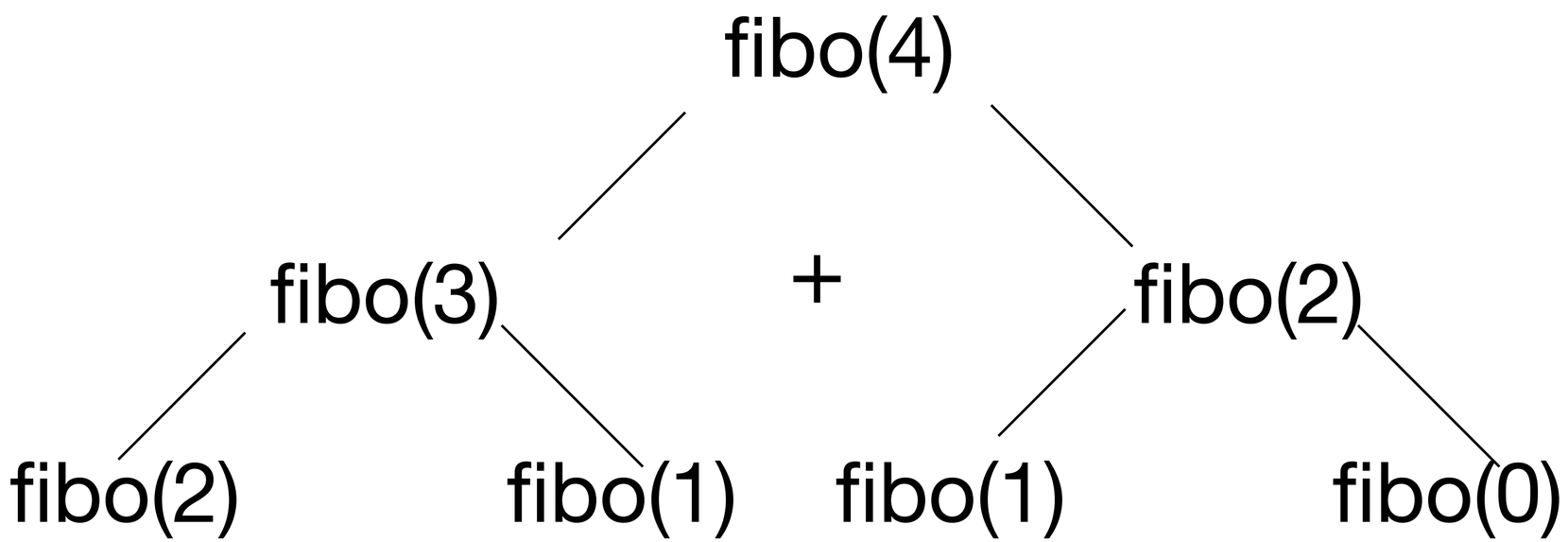
하나의 큰 문제를 여러 개의 작은 문제로 쪼갤 때 하나의 작은 문제가 반복적으로 등장하는 구조,
즉 작은 문제가 계속해서 사용되는 구조

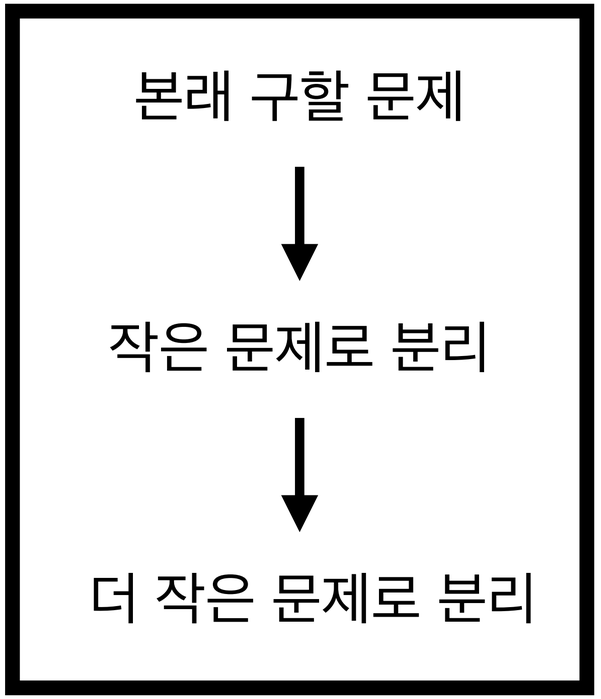
모든 문제가 하나의 문제에 중복적인 구조를 가질 필요는 없지만,
조금이라도 중복되는 부분이 있어야 조건을 만족한다.

이 문제들 중에서 중복되는 부분이 있다.

동적 프로그래밍이란

탑 다운 방식 메모이제이션





중복된 작은 문제가 존재한다면 이 문제의 결과를 기록하여 문제를 다시 푸는 일을 방지함.

가장 큰 장점은 완전 탐색 풀이와 동일한 구성을 갖추고 있기 때문에 완전탐색 + 중복 제거 방식으로 동적 프로그래밍을 구현할 수 있음.

동적 프로그래밍이란

바텀 업 방식 테뷸레이션

이미 알고 있는 정답 ⟶ 새 정답 도출

반복문을 기반으로 하기 때문에 정확히 몇번 반복할지 정하는 지표 필요

어떤 계산을 하는지 명확한 기준이 필요.

fibo(0) + fibo(1) ⟶ fibo(2)

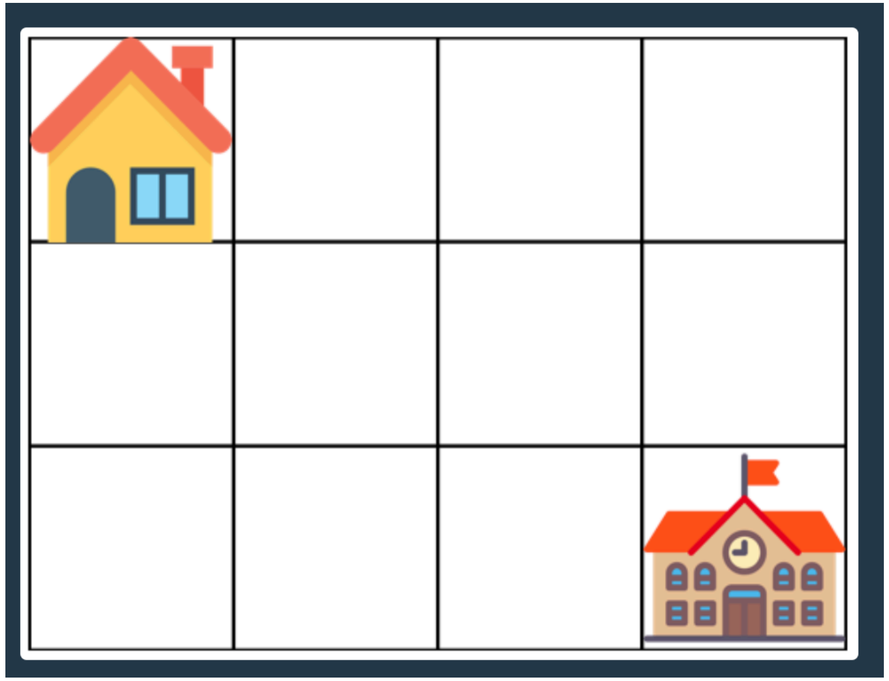
fibo(1) + fibo(2) ⟶ fibo(3)

fibo(2) + fibo(3) ⟶ fibo(4)

문제 41

문제 41

문제 이해하기

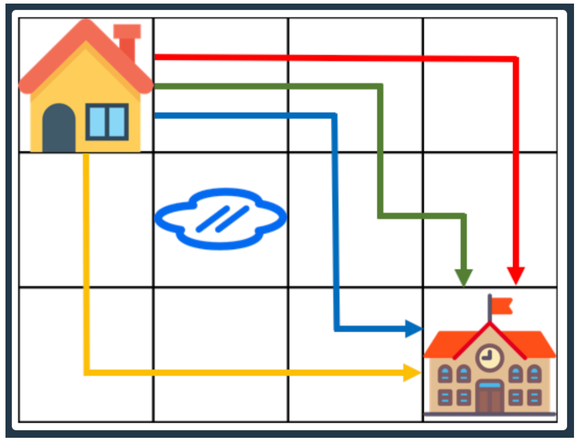


계속되는 폭우로 일부 지역이 물에 잠겼습니다. 물에 잠기지 않은 지역을 통해 학교를 가려고 합니다. 집에서 학교까지 가는 길은 m x n 크기의 격자모양으로 나타낼 수 있습니다.

아래 그림은 m = 4, n = 3 인 경우입니다.

가장 왼쪽 위, 즉 집이 있는 곳의 좌표는 (1, 1)로 나타내고 가장 오른쪽 아래, 즉 학교가 있는 곳의 좌표는 (m, n)으로 나타냅니다.

격자의 크기 m, n과 물이 잠긴 지역의 좌표를 담은 2차원 배열 puddles이 매개변수로 주어집니다. **오른쪽과 아래쪽으로만 움직여** 집에서 학교까지 갈 수 있는 최단경로의 개수를 1,000,000,007로 나눈 나머지를 return 하도록 solution 함수를 작성해주세요.



입출력 예

m	n	puddles	return
4	3	[[2, 2]]	4

41
접근방식

수학 공식을 적용해 풀자 **합의 법칙**

1	1	1	1
1	0	1	2
1	1	2	4

시작 지점부터 다음 지점까지 1로 표시하고

다음칸으로 넘어가면서 위와 왼쪽 숫자를 모두 더해간다.

웅덩이가 있을때는 무조건 0

41
접근방식

수학 공식을 적용해 풀자

합의 법칙

1	1	1	1
1	0	1	2
1	1	2	4

1

```
def solution(m,n,puddles):
    dp = [[0] * m for _ in range(n)]
    dp[0][0] = 1
```

초기 세팅으로 모든 값을 0으로 세팅

2

```
for y in range(n):
    for x in range(m):
        if ([x +1, y+1] in puddles) or ((y,x) == (0,0)) : continue
        dp[y][x] = (dp[y-1][x] + dp[y][x-1]) % 100000007
```

0,0 시작점이거나 물 웅덩이면 그냥 continue

새로운 값 윗쪽 값 왼쪽 값

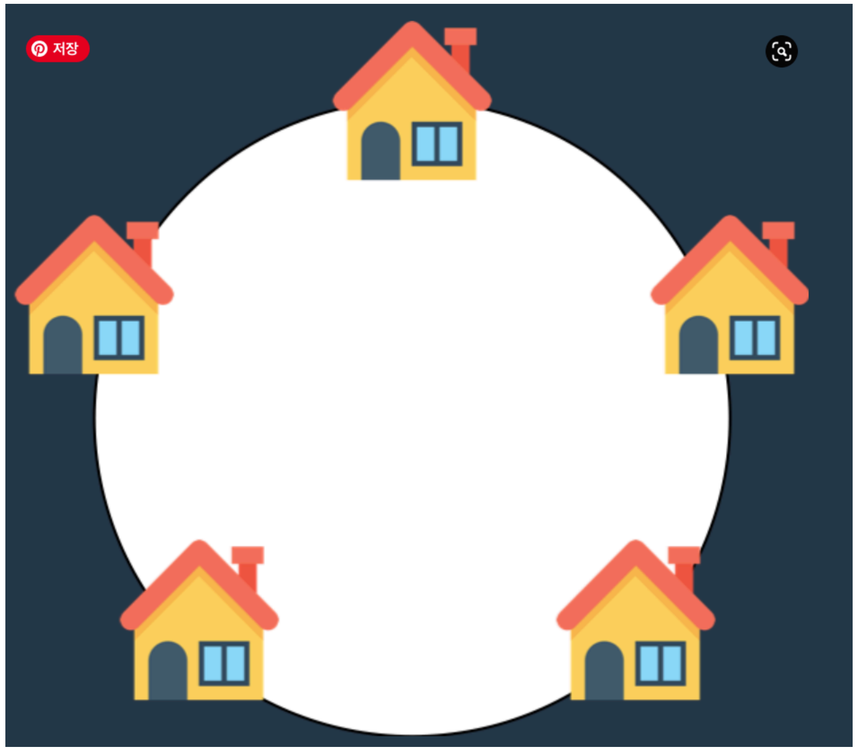
3

```
return dp[-1][-1]
```

문제 42

문제 42

문제 이해하기



각 집들은 서로 인접한 집들과 방범장치가 연결되어 있기 때문에 인접한 두 집을 털면 경보가 울립니다.

각 집에 있는 돈이 담긴 배열 money가 주어질 때, 도둑이 훔칠 수 있는 돈의 최댓값을 return 하도록 solution 함수를 작성하세요.

제한사항

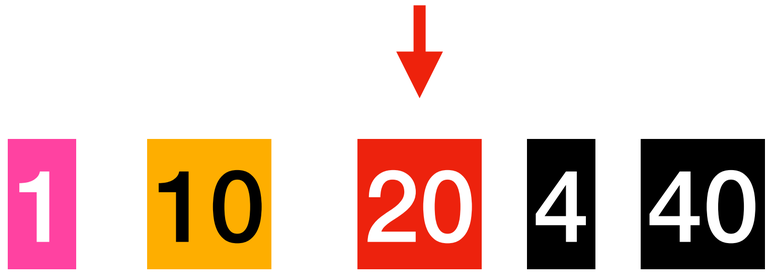
- 이 마을에 있는 집은 3개 이상 1,000,000개 이하입니다.
- money 배열의 각 원소는 0 이상 1,000 이하인 정수입니다.

입출력 예

money	return
[1, 2, 3, 1]	4

문제 42

접근방식



1 + 20　　10

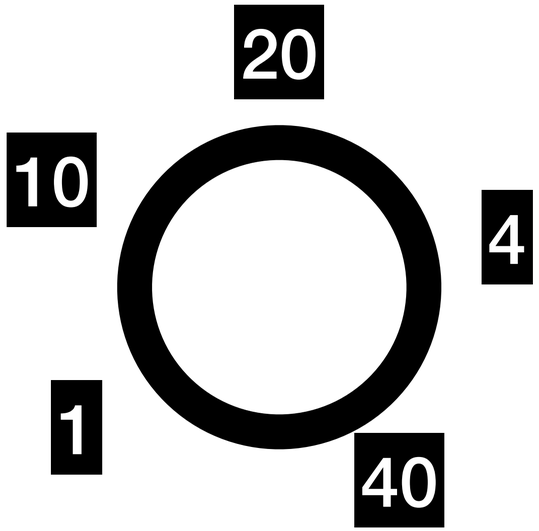
현재 숫자에서 다다음 숫자를 더하는 것이 클까?
아니면 현재 숫자를 고르지 않고 다음 숫자를 고르는 것이 더 클까?

max 지금 털 집의 금액 + 두 집 전의 집을 턴 금액　　한 집 전의 집을 턴 금액

c[i] + + dp[i-2]　　dp[i-1]

문제 42
접근방식

[1,10,20,4,40]



집은 원형으로 되어 있음.

첫번째 집부터 털게 되면 가장 가치 있는 마지막 집을 털지 못하게 됨

max 첫번째 집부터 터는 경우 두번째 집부터 터는 경우

max [1,20.....] max [10.....40]

문제 42

접근방식

첫번째 집부터 터는 경우

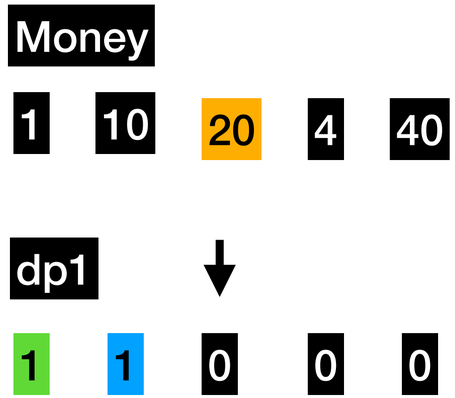
dp1 = [0] * len(money)

각 위치별로 가장 많이 가질 수 있는 금액을 기록할 배열

dp1[0] = dp1[1] = money[0]

첫번째 집부터 터는 경우 가장 많이 가질 수 있는 금액을 저장한 배열 -> dp1

첫번째 집부터 터는 거니까 Money[0] 값을 넣어준다.



첫번째 집부터 털 경우 마지막 집은 털 수 없으므로 -1을 해준다.

위의 점화식을 그대로 적용

```
for i in range(2,len(money)-1):
    dp1[i] = max((money[i] + dp1[i-2]),dp1[i-1])
```

문제 42

접근방식

두번째 집부터 터는 경우

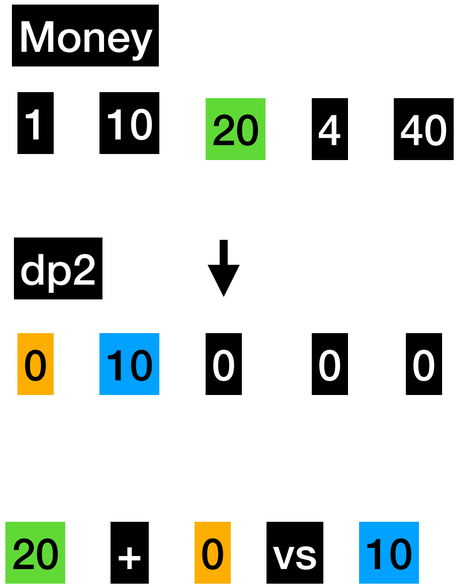
dp2 = [0] * len(money)

dp2[0] = 0 두번째 집부터 털 경우 첫번째 집은 안털었으므로 0을 넣어준다.

dp2[1] = money[1] 두번째 집부터 터니까 money[1] 값을 넣어준다.

for i in range(2,len(money)): 두번째 집부터 털 경우 마지막 집까지 모두 털 수 있다.

    dp2[i] = max((money[i] + dp2[i-2]),dp2[i-1])



첫번째 경우와 두번째 경우의 max 값을 비교하여 더 큰 값을 리턴

return max(max(dp1),max(dp2))