8.2 다이나믹 프로그래밍

DP 복습

문제 3, 4, 5

1. 다이나믹 프로그래밍

- 다이나믹 프로그래밍 조건
  1) 큰 문제를 작은 부분 문제로 나눔
  2) 작은 문제의 정답을 큰 문제에 활용

- 구현 방법
  1) 탑다운(메모이제이션, 하향식)
     - 파이썬의 경우 recursion depth limit
  2) 버텀업(상향식)

2. 문제3 – 개미 전사

- 문제 설명

입력 조건
- 첫째 줄에 식량창고의 개수 N이 주어진다. ($3 \le N \le 100$)
- 둘째 줄에 공백으로 구분되어 각 식량창고에 저장된 식량의 개수 K가 주어진다. ($0 \le K \le 1,000$)

출력 조건
- 첫째 줄에 개미 전사가 얻을 수 있는 식량의 최댓값을 출력하시오.

입력 예시

```
4
1 3 1 5
```

출력 예시

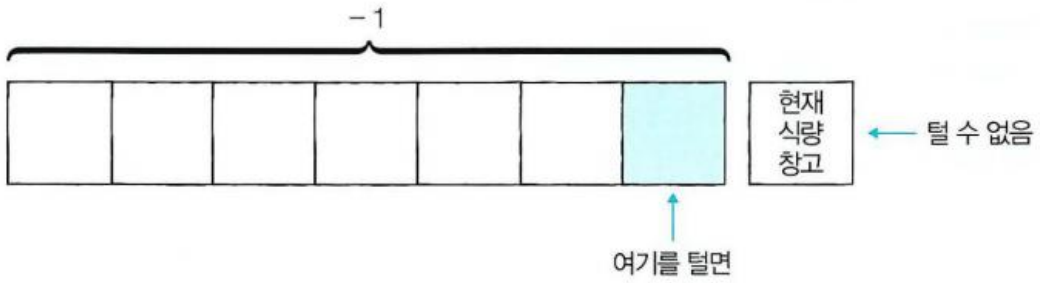
```
8
```

  - 조건: 한 칸 이상 띄어서 약탈 가능

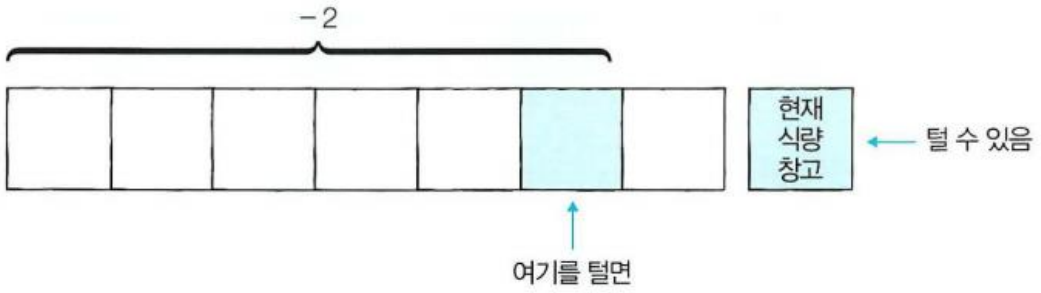
2. 문제3 – 개미 전사

- 풀이1 – 책

ⓐ (i – 1)번째 식량창고를 털기로 결정한 경우 현재의 식량창고를 털 수 없다.



ⓑ (i – 2)번째 식량창고를 털기로 결정한 경우 현재의 식량창고를 털 수 있다.

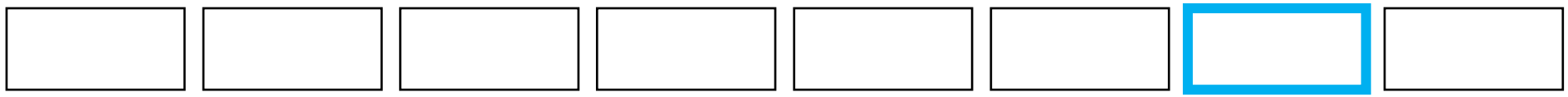


창고 배열: arr
누적해서 저장하는 배열: dp

DP 점화식
dp[i] = max(dp[i-1], dp[i-2]+arr[i])

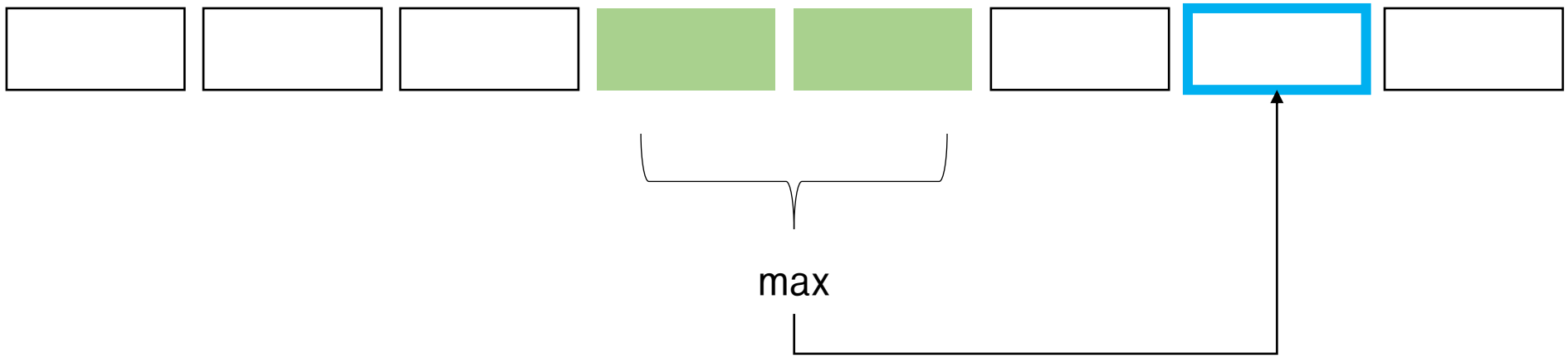
2. 문제3 – 개미 전사

- 번외 풀이



2. 문제3 – 개미 전사

- 번외 풀이



창고 배열: arr
누적해서 저장하는 배열: dp (단, i번째는 i번째 값을 사용하면서 누적 최대값)

DP 점화식
dp[i] = max(dp[i−2], dp[i−3]) + arr[i]

2. 문제3 – 개미 전사

- 풀이 비교

풀이1

```python
def sol2(ls):
    cache = [0] * len(ls)
    cache[0], cache[1] = ls[0], max(ls[0], ls[1])
    for i in range(2, len(cache)):
        cache[i] = max(cache[i-1], cache[i-2] + ls[i])
    return cache[-1]
```

풀이2

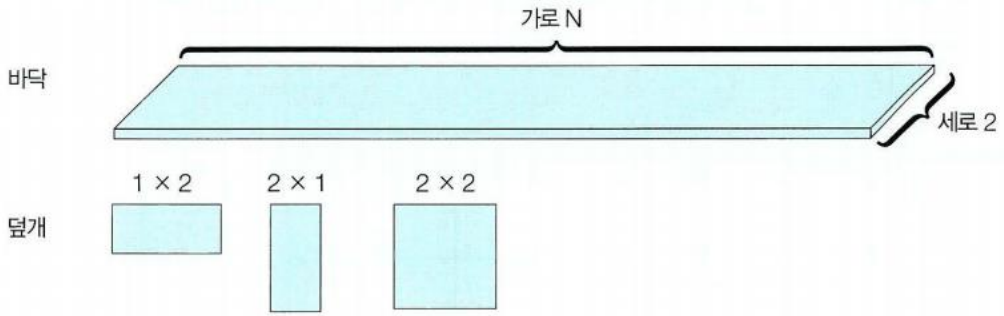
```python
def solution(ls): # ls: 식량 창고 리스트 (길이 >=3)
    cache = [0] * len(ls)
    cache[0], cache[1], cache[2] = ls[0], ls[1], ls[2] + ls[0]
    # ls[i]: i번째를 사용할 때 합의 최대값
    for i in range(3, len(cache)):
        cache[i] = ls[i] + max(cache[i-3], cache[i-2])

    return max(cache[-2:])
```

3. 문제4

- 문제 설명

가로의 길이가 N, 세로의 길이가 2인 직사각형 형태의 얇은 바닥이 있다. 태일이는 이 얇은 바닥을 1 × 2의 덮개, 2 × 1의 덮개, 2 × 2의 덮개를 이용해 채우고자 한다.



이때 바닥을 채우는 모든 경우의 수를 구하는 프로그램을 작성하시오. 예를 들어 2 × 3 크기의 바닥을 채우는 경우의 수는 5가지이다.

입력 조건 • 첫째 줄에 N이 주어진다. (1 ≤ N ≤ 1,000)

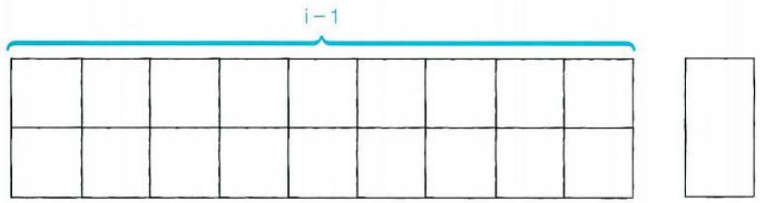
출력 조건 • 첫째 줄에 2 × N 크기의 바닥을 채우는 방법의 수를 796,796으로 나눈 나머지를 출력한다.

입력 예시	출력 예시
3	5

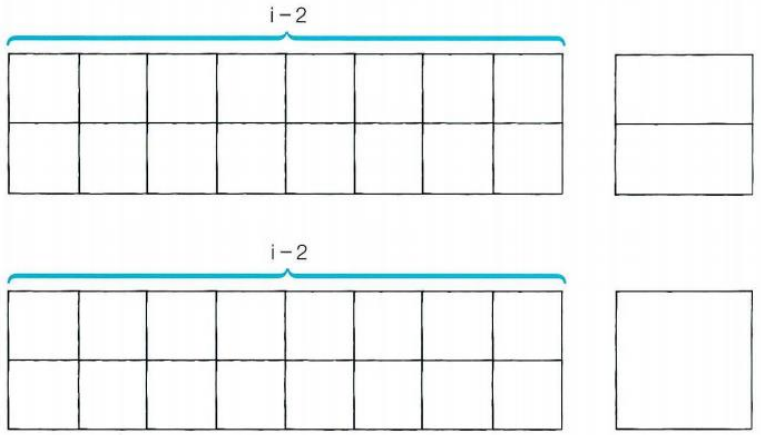
3. 문제4

- 문제 풀이

1 왼쪽부터 i − 1까지 길이가 덮개로 이미 채워져 있으면 2 × 1의 덮개를 채우는 하나의 경우밖에 존재하지 않는다.



2 왼쪽부터 i − 2까지 길이가 덮개로 이미 채워져 있으면 1 × 2 덮개 2개를 넣는 경우, 혹은 2 × 2의 덮개 하나를 넣는 경우로 2가지 경우가 존재한다.



점화식
DP[i] = DP[i−1] + 2 * DP[i−2]

8-7.py 답안 예시

```
# 정수 N을 입력받기
n = int(input())

# 앞서 계산된 결과를 저장하기 위한 DP 테이블 초기화
d = [0] * 1001

# 다이나믹 프로그래밍(Dynamic Programming) 진행(보텀업)
d[1] = 1
d[2] = 3
for i in range(3, n + 1):
    d[i] = (d[i - 1] + 2 * d[i - 2]) % 796796

# 계산된 결과 출력
print(d[n])
```

4. 문제5 – 효율적인 화폐 구성

- 문제 설명

입력 조건
- 첫째 줄에 N, M이 주어진다. (1 ≤ N ≤ 100, 1 ≤ M ≤ 10,000)
- 이후 N개의 줄에는 각 화폐의 가치가 주어진다. 화폐 가치는 10,000보다 작거나 같은 자연수이다.

출력 조건
- 첫째 줄에 M원을 만들기 위한 최소한의 화폐 개수를 출력한다.
- 불가능할 때는 -1을 출력한다.

입력 예시 1

```
2 15
2
3
```

출력 예시 1

```
5
```

입력 예시 2

```
3 4
3
5
7
```

출력 예시 2

```
-1
```

4. 문제5 – 효율적인 화폐 구성

- 문제 풀이
  - 화폐들이 배수가 아니다
    - 예) m=121, 화폐=[1, 11, 13]
    - 13 * 9 + 1 * 5 = 14개
    - 11 * 11 = 11개
  - Greedy X
  - 따라서 가능한 모든 방법 중 최소 방법을 저장해야 함!

4. 문제5 – 효율적인 화폐 구성

- 문제 풀이

8-8.py 답안 예시

```
# 정수 N, M을 입력받기
n, m = map(int, input().split())
# N개의 화폐 단위 정보를 입력받기
array = []
for i in range(n):
    array.append(int(input()))

# 한 번 계산된 결과를 저장하기 위한 DP 테이블 초기화
d = [10001] * (m + 1)

# 다이나믹 프로그래밍(Dynamic Programming) 진행(보텀업)
d[0] = 0
for i in range(n):
    for j in range(array[i], m + 1):
        if d[j - array[i]] != 10001: # (i - k)원을 만드는 방법이 존재하는 경우
            d[j] = min(d[j], d[j - array[i]] + 1)

# 계산된 결과 출력
if d[m] == 10001: # 최종적으로 M원을 만드는 방법이 없는 경우
    print(-1)
else:
    print(d[m])
```

```
def solution(money, total):
    money.sort(reverse=True)
    count = [-1] * (total + 1) # 최소 방법

    # bottom-up으로 채워넣기
    def make_value(num):
        if num > total:
            return

        for m in money:
            tmp = num + m
            if tmp <= total:
                # 최소 경로일 때
                if count[tmp] < 0  or count[num] + 1 < count[tmp]:
                    count[tmp] = count[num] + 1
                    make_value(tmp)

    for m in money:
        if m <= total:
            count[m] = 1
            make_value(m)
    return count[total]
```

감사합니다