알고리즘2 (2024-2)

9. 그리디 알고리즘과 동적 계획법

국립금오공과대학교 컴퓨터공학과

김 경 수

학습 목표

- 그리디 알고리즘과 동적 계획법을 활용하는 문제의 특성을 이해하고 실제 문제를 해결할 수 있다.
- 실제 문제 해결 과정에서 그리디 알고리즘과 동적 계획법을 적용할 때 유의해야 할 사항이 무엇인지 숙지하고, 이를 실제 문제 풀이 과정에 적용할 수 있다.

문제 1. 구명보트

문제 1. 구명보트

제한시간: 30분

- 무인도에 갇힌 사람들을 구명보트를 이용하여 구출하려고 합니다. 구명보트는 작아서 한 번에 최대 2명씩 밖에 탈 수 없고, 무게 제한도 있습니다.
- 예를 들어, 사람들의 몸무게가 [70kg, 50kg, 80kg, 50kg]이고 구명보트의 무게 제한이 100kg이라면 2번째 사람과 4번째 사람은 같이 탈 수 있지만 1번째 사람과 3번째 사람의 무게의 합은 150kg이므로 구명보트의 무게 제한을 초과하여 같이 탈 수 없습니다.
- 구명보트를 최대한 적게 사용하여 모든 사람을 구출하려고 합니다.
- 사람들의 몸무게를 담은 배열 people과 구명보트의 무게 제한 limit가 매개변수로 주어질 때, 모든 사람을 구출하기 위해 필요한 구명보트 개수의 최솟값을 return 하도록 **solution1** 함수를 작성해주세요.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/42885

문제 1. 구명보트

제한시간: 30분

- **제한사항**
  - 무인도에 갇힌 사람은 1명 이상 50,000명 이하입니다.
  - 각 사람의 몸무게는 40kg 이상 240kg 이하입니다.
  - 구명보트의 무게 제한은 40kg 이상 240kg 이하입니다.
  - 구명보트의 무게 제한은 항상 사람들의 몸무게 중 최댓값보다 크게 주어지므로 사람들을 구출할 수 없는 경우는 없습니다.
- **입출력 예제**

people	limit	return
[70, 50, 80, 50]	100	3
[70, 80, 50]	100	3

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/42885

문제 1. 문제 분석

- **주어진 문제의 목표**
  - ➢ 사람들의 몸무게를 담은 배열 people과 구명보트의 무게 제한 limit가 매개변수로 주어질 때, 모든 사람을 구출하기 위해 필요한 구명보트 개수의 최소값을 계산한다.

- **문제의 특성**
  - ✓ 구명보트의 개수를 최소화하는 것이 중요함.
  - ✓ 구명보트에는 최대 2명만 탑승할 수 있음.
  - ✓ 모든 사람을 구출해야 함.

문제 1. 문제 해결 전략 수립

- **보트의 수를 최소화하고 동시에 모든 사람을 구출해야 한다.**
  - 하나의 보트에 2명씩 태우는 것이 가장 바람직하다.
  - 즉, 한 명을 보트에 태웠을 때 남은 보트의 용량을 최소화하고, 이에 맞는 사람을 태워야 한다.
  - 따라서, 가장 무거운 사람과 가장 가벼운 사람을 짝지어서 보트에 태워야 한다.
- **이를 해결하기 위한 방법**
  - 그리디 알고리즘을 이용하여 가장 무거운 사람과 가장 가벼운 사람을 선택한다.
  - 이를 위해, 사람들의 몸무게를 나타내는 입력 배열 people[]를 오름차순으로 정렬한다.
  - 정렬된 people[] 리스트에서 맨 앞의 사람과 맨 뒤의 사람을 순차적으로 선택하면서 보트에 태울 수 있는 지 확인한다.

문제 1. 문제 해결 방법 기술

- 주어진 입력 리스트 people을 몸무게를 기준으로 오름차순으로 정렬한다.
- 정렬된 리스트 people에서 몸무게가 가장 가벼운 사람을 변수 idx, 가장 무거운 사람을 변수 jdx가 가리키게 한다.
- **while** idx <= jdx **do**
  - 만약 people[idx] + people[jdx] <= limit 이면, 두 사람을 모두 보트에 태우고, idx = idx + 1; jdx = jdx - 1를 수행한다.
  - 만약 두 사람 모두 보트에 태우지 못하면, 무거운 사람만을 보트에 태우고, jdx = jdx - 1를 수행한다.
  - 사람을 태운 보트의 수를 1 증가시키기 위해 count = count + 1을 수행한다.
- 최종 계산된 보트의 개수 count를 반환한다.

문제 1. 구현 및 검증

```
def solution1(people, limit):
    people.sort()
    count = 0
    i = 0  # 가장 가벼운 사람을 가리키는 인덱스
    j = len(people) - 1  # 가장 무거운 사람을 가리키는 인덱스

    while i <= j:
        # 가장 무거운 사람과 가장 가벼운 사람을 같이 태울 수 있으면 모두 태움
        if people[j] + people[i] <= limit:
            i += 1

        # 무거운 사람만 태울 수 있으면 무거운 사람만 태움
        j -= 1
        count += 1

    return count
```

문제 1. 구현 및 검증

테스트 케이스 #1

```
people = [70, 50, 80, 50]
limit = 100
print(solution1(people, limit))
```

테스트 케이스 #2

```
people = [70, 80, 50]
limit = 100
print(solution1(people, limit))
```

문제 2. 기지국 설치

문제 2. 기지국 설치 (1/4)

제한시간: 50분

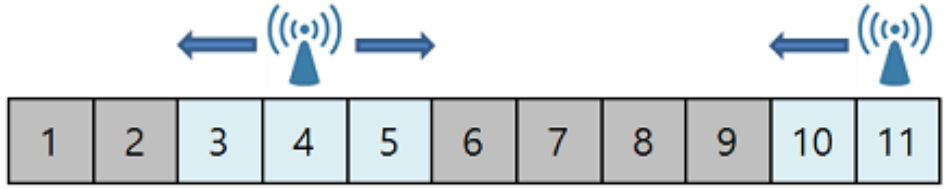
- N개의 아파트가 일렬로 쭉 늘어서 있습니다. 이 중에서 일부 아파트 옥상에는 4g 기지국이 설치되어 있습니다.
- 기술이 발전해 5g 수요가 높아져 4g 기지국을 5g 기지국으로 바꾸려 합니다. 그런데 5g 기지국은 4g 기지국보다 전달 범위가 좁아, 4g 기지국을 5g 기지국으로 바꾸면 어떤 아파트에는 전파가 도달하지 않습니다.
- 예를 들어 11개의 아파트가 쭉 늘어서 있고, [4, 11] 번째 아파트 옥상에는 4g 기지국이 설치되어 있습니다. 만약 이 4g 기지국이 전파 도달 거리가 1인 5g 기지국으로 바뀔 경우 모든 아파트에 전파를 전달할 수 없습니다. (전파의 도달 거리가 W일 땐, 기지국이 설치된 아파트를 기준으로 전파를 양쪽으로 W만큼 전달할 수 있습니다.)

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/12979

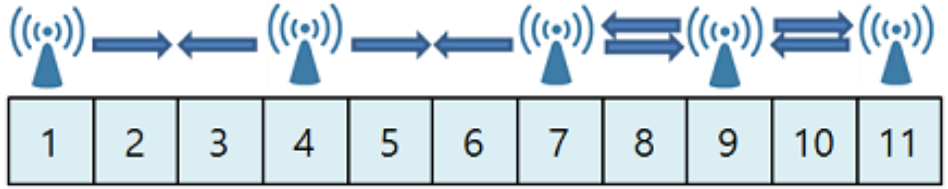
문제 2. 기지국 설치 (2/4)

제한시간: 50분

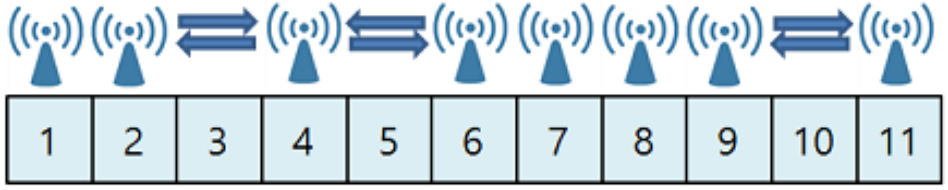
- 초기에, 1, 2, 6, 7, 8, 9번째 아파트에는 전파가 전달되지 않습니다.



- 1, 7, 9번째 아파트 옥상에 기지국을 설치할 경우, 모든 아파트에 전파를 전달할 수 있습니다.



- 1, 7, 9번째 아파트 옥상에 기지국을 설치할 경우, 모든 아파트에 전파를 전달할 수 있습니다.



★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/12979

문제 2. 기지국 설치 (3/4)

제한시간: 50분

- 이때, 우리는 5g 기지국을 최소로 설치하면서 모든 아파트에 전파를 전달하려고 합니다. 위의 예시에선 최소 3개의 아파트 옥상에 기지국을 설치해야 모든 아파트에 전파를 전달할 수 있습니다.
- 아파트의 개수 N, 현재 기지국이 설치된 아파트의 번호가 담긴 1차원 배열 stations, 전파의 도달 거리 W가 매개변수로 주어질 때, 모든 아파트에 전파를 전달하기 위해 증설해야 할 기지국 개수의 최솟값을 리턴하는 **solution2** 함수를 완성해주세요
- **제한사항**
  - N: 200,000,000 이하의 자연수
  - stations의 크기: 10,000 이하의 자연수
  - stations는 오름차순으로 정렬되어 있고, 배열에 담긴 수는 N보다 같거나 작은 자연수입니다.
  - W: 10,000 이하의 자연수

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/12979

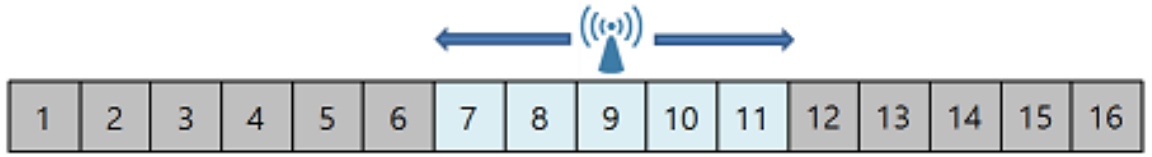
문제 2. 기지국 설치 (4/4)

제한시간: 50분

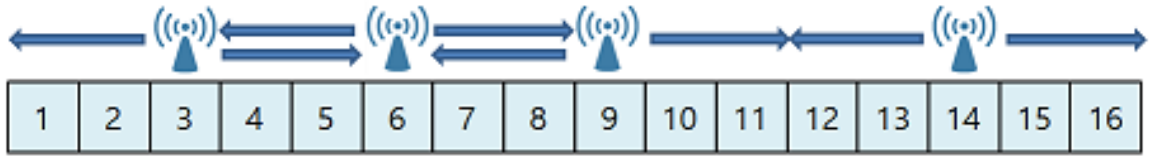
- **입출력 예제**

N	stations	W	answer
11	[4, 11]	1	3
16	[9]	2	3

  - **예제 #1.** 문제의 예시와 같습니다.
  - **예제 #2.**
    - 초기에, 1~6, 12~16번째 아파트에는 전파가 전달되지 않습니다.



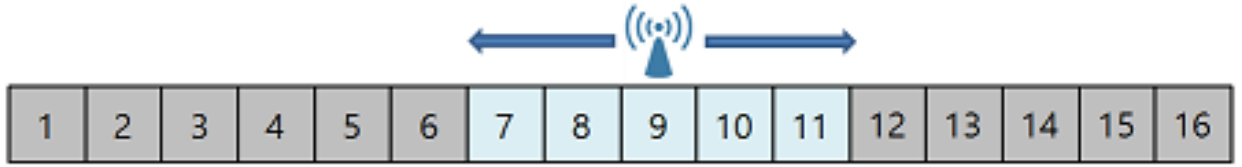
    - 3, 6, 14번째 아파트 옥상에 기지국을 설치할 경우 모든 아파트에 전파를 전달할 수 있습니다.



★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/12979

문제 2. 문제 분석

- **주어진 문제의 목표**
  - 기지국이 설치된 위치와 기지국의 전파 거리가 입력으로 주어졌을 때, 아파트 모든 동에 전파가 도달할 수 있도록 증설해야 하는 기지국의 수를 계산한다.
- **문제의 특성**
  - 기지국의 전파의 도달 거리가 w라면, 기지국의 설치 지점 기준 왼쪽으로 w, 오른쪽으로 w만큼 전파가 도달한다.

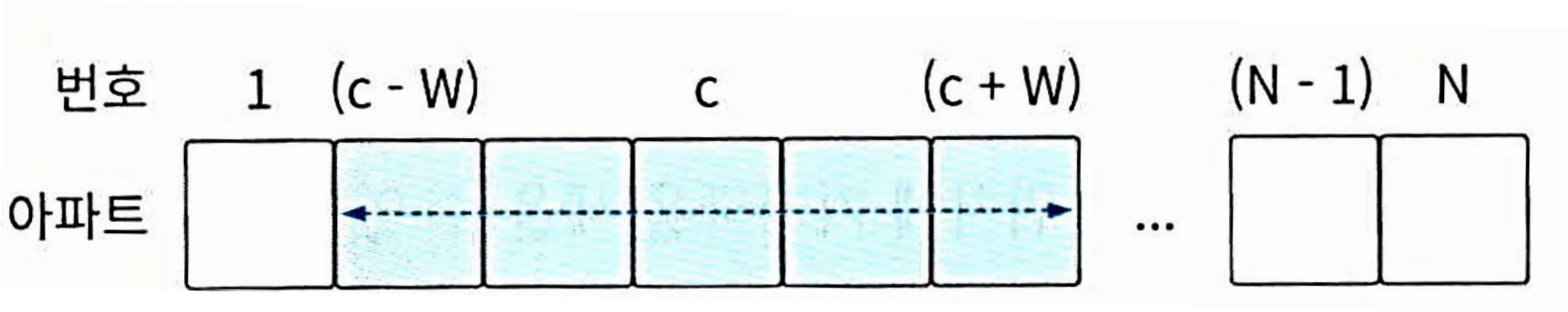


  - 증설하는 기지국의 수를 최소화하기 위해서는 어떻게 해야 하는가?
    - ✓ 전파의 도달 범위가 겹치지 않도록 기지국 설치 위치를 결정해야 한다.

문제 2. 문제 해결 전략 수립

- **효율적으로 기지국을 세우는 방법**
  - 전파 범위가 겹치지 않게 기지국을 설치해야 한다.
  - 이를 위해서는 특정 위치에서 기지국의 전파 도달 범위를 계산해야 한다.
  - 위치 인덱스를 c, 전파 도달 거리를 W라 할 때, 기지국의 전파 도달 범위 k를 계산하는 방법

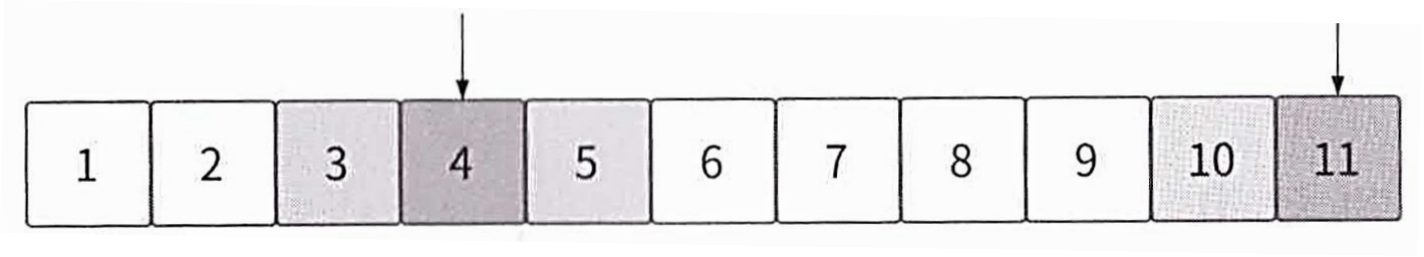
$$\boldsymbol{c - W \leq k \leq c + W}$$



문제 2. 문제 해결 전략 수립

- **전파 범위가 겹치지 않게 기지국 설치 지점을 탐색하는 방법 (1/2)**
  - ➢ 기지국 설치 지점 선택을 위해 아파트 첫 동부터 순차적으로 탐색한다.
  - ➢ 기지국이 설치된 위치의 범위 내에 접근한 경우 도달 범위 바깥으로 바로 이동한다.
    - 즉, 현재 위치가 location이고, location이 기지국이 설치된 위치의 전파 도달 범위 내에 도달했을 때, 해당 기지국의 전파 범위 바깥으로 이동하는 방법은 다음과 같다.

$$\boldsymbol{location = station[i] + W + 1}$$

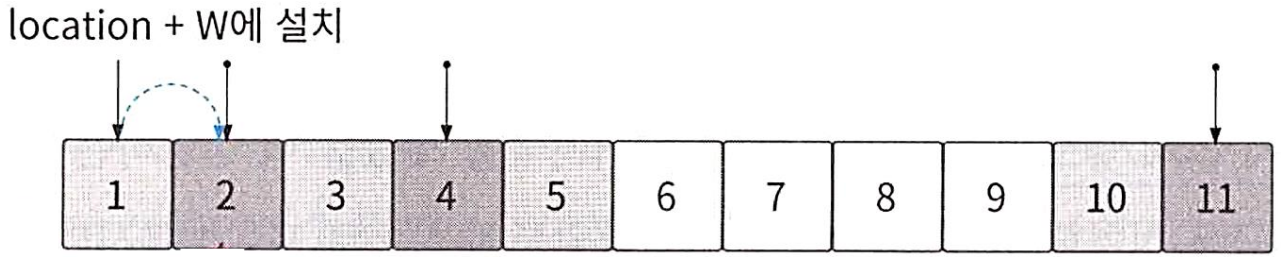


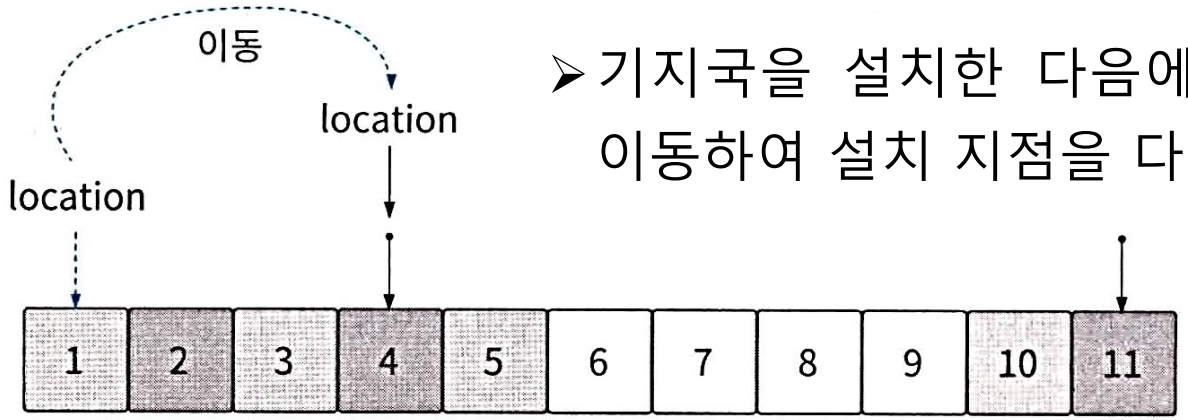
  - ➢ 위 그림의 예제에서 location=3인 경우, 다음 탐색 위치는 아래와 같이 결정한다.

$$\boldsymbol{location = station[1] + W + 1 = 4 + 1 + 1 = 6}$$

문제 2. 문제 해결 전략 수립

- **전파 범위가 겹치지 않게 기지국 설치 지점을 탐색하는 방법 (2/2)**
  - ➢ 현재 위치 $location$이 기지국이 설치된 지점이 아닌 경우에는 해당 위치에 기지국을 설치한 후, 해당 위치에서의 전파 범위 바깥으로 이동한다.
    - 즉, 현재 위치 $location$을 기준으로, 기지국의 최적 설치 지점은 $\boldsymbol{location + W}$이다.





➢ 기지국을 설치한 다음에, $\boldsymbol{location + 2W + 1}$ 만큼 이동하여 설치 지점을 다시 탐색한다.

문제 2. 문제 해결 방법 기술

While location <= N do

- 현재 위치에 기지국이 설치되어 있는 경우
  - 현재 위치에서 전파 도달 범위 바깥으로 탐색 위치를 이동시킨다.
- 현재 위치에 기지국이 설치되지 않은 경우
  - 현재 위치에 기지국을 설치한다.
  - 현재 위치에서 전파 도달 범위 바깥으로 탐색 위치를 이동시킨다.
  - 새로 설치한 기지국의 수를 1 증가시킨다.

새로 설치한 기지국의 수를 반환한다.

문제 2. 구현 및 검증

```
def solution2(N, stations, W):
    answer = 0
    location = 1  # 현재 탐색하는 아파트의 위치
    idx = 0  # 설치된 기지국의 인덱스

    while location <= N:
        # 기지국이 설치된 위치에 도달한 경우
        if idx < len(stations) and location >= stations[idx] - W:
            location = stations[idx] + W + 1
            idx = idx + 1
        # 기지국이 설치되지 않은 위치인 경우
        else:
            location += 2*W + 1  # 기지국을 설치하고 해당 범위를 넘어감
            answer += 1

    return answer
```

문제 2. 구현 및 검증

테스트 케이스 #1

```
N = 11
stations = [4, 11]
W = 1
print( solution2(N, stations, W) )
```

테스트 케이스 #2

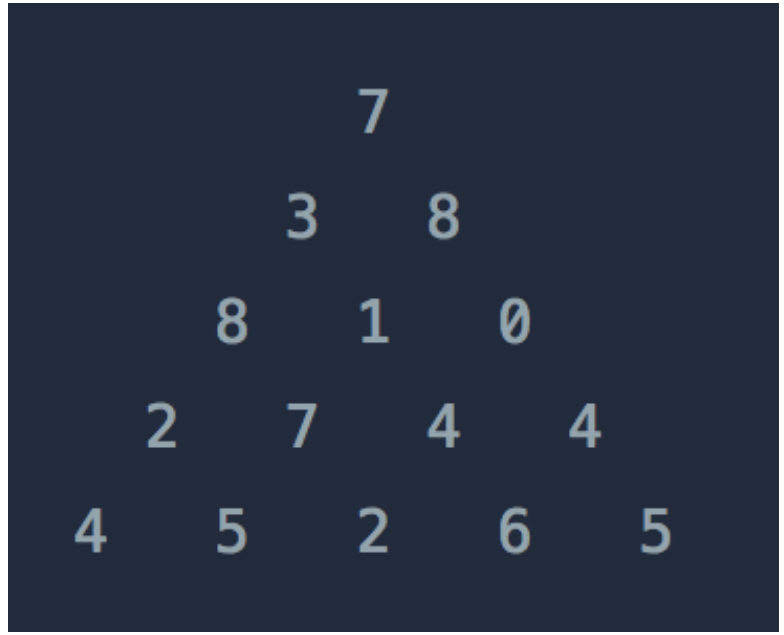
```
N = 16
stations = [9]
W = 2
print( solution2(N, stations, W) )
```

문제 3. 정수 삼각형

문제 3. 정수 삼각형 (1/2)

제한시간: 40분

- 오른쪽 그림과 같은 삼각형의 꼭대기에서 바닥까지 이어지는 경로 중, 거쳐간 숫자의 합이 가장 큰 경우를 찾아보려고 합니다. 아래 칸으로 이동할 때는 대각선 방향으로 한 칸 오른쪽 또는 왼쪽으로만 이동 가능합니다. 예를 들어 3에서는 그 아래칸의 8 또는 1로만 이동이 가능합니다.



- 삼각형의 정보가 담긴 배열 triangle이 매개변수로 주어질 때, 거쳐간 숫자의 최댓값을 return 하도록 **solution3** 함수를 완성하세요.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/43105

문제 3. 정수 삼각형 (2/2)

제한시간: 40분

- **제한사항**
  - 삼각형의 높이는 1 이상 500 이하입니다.
  - 삼각형을 이루고 있는 숫자는 0 이상 9,999 이하의 정수입니다.

- **입출력 예제**
  - 문제의 예시와 같습니다.

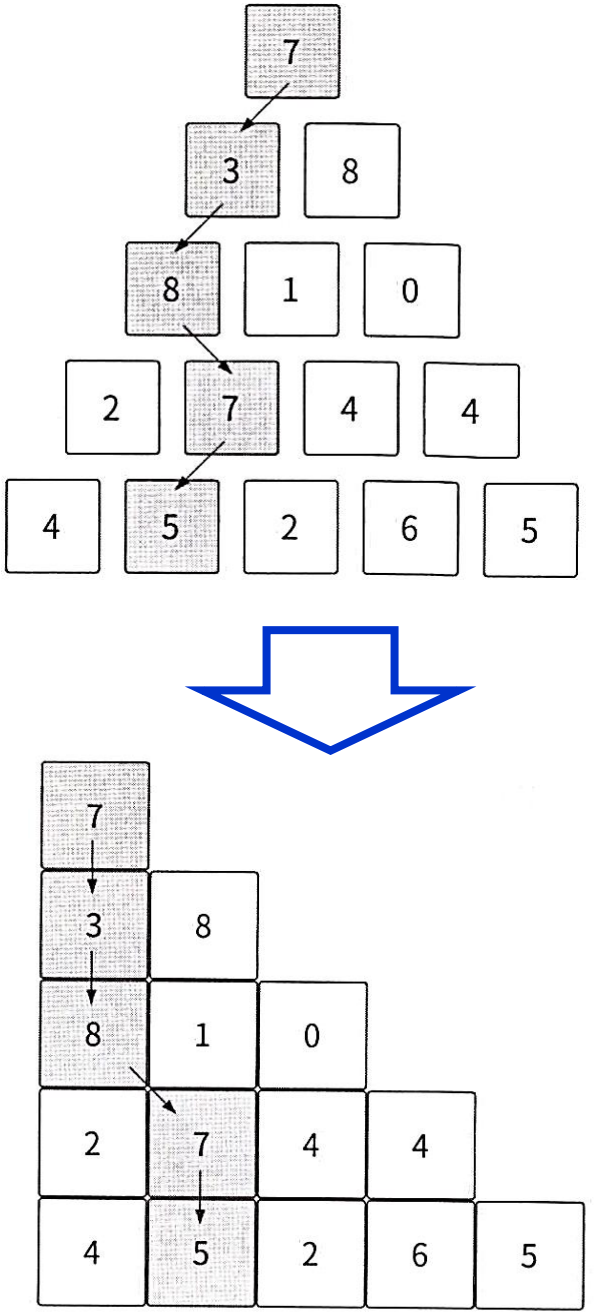
triangle	result
[[7], [3, 8], [8, 1, 0], [2, 7, 4, 4], [4, 5, 2, 6, 5]]	30

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/43105

문제 3. 문제 분석

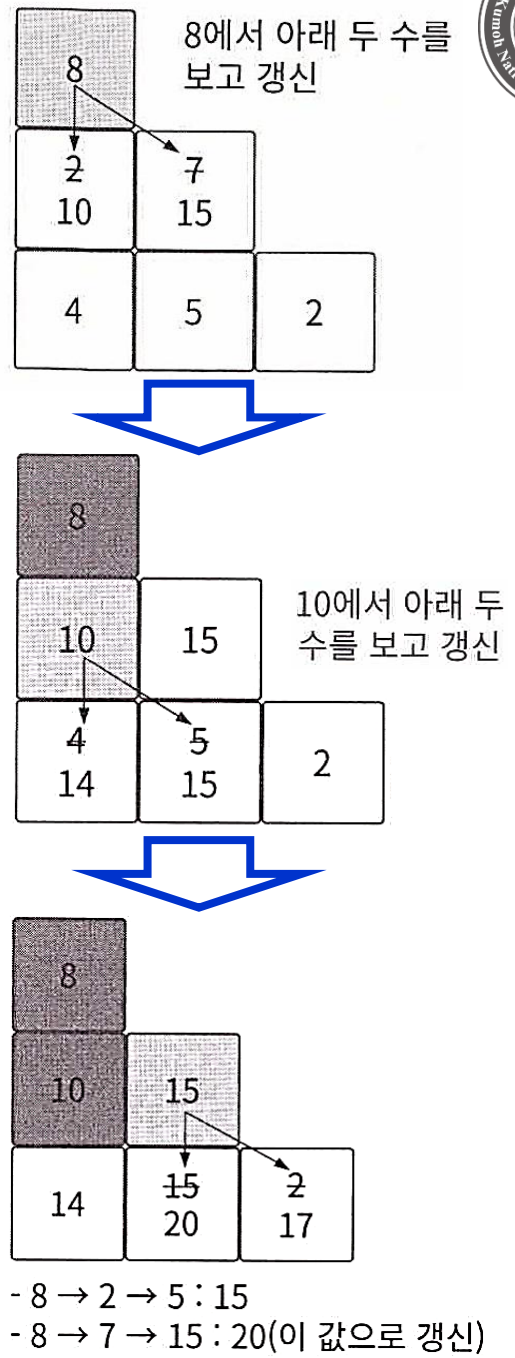
- **주어진 문제의 목표**
  - ➢ 정수 삼각형의 꼭대기에서 바닥까지 어이지는 경로 중 거쳐간 숫자의 합의 최대값을 계산하여 반환한다.

- **문제의 특성**
  - ➢ 문제를 보다 쉽게 분석하기 위해 오른쪽 그림과 같이 주어진 삼각형의 배열을 바꿔서 생각해 보자.
    - ✓ 숫자 이동 규칙에 영향을 주지 않는다면 오른쪽 아래와 같은 그림으로 표현하는 것이 분석하기 더 쉬움.
  - ➢ 삼각형의 꼭대기에서 바닥까지 순차적으로 계산(↓)
    - **Top-down approach**
  - ➢ 삼각형의 바닥에서 꼭대기순으로 올라가면서 계산(↑)
    - **Bottom-up approach**



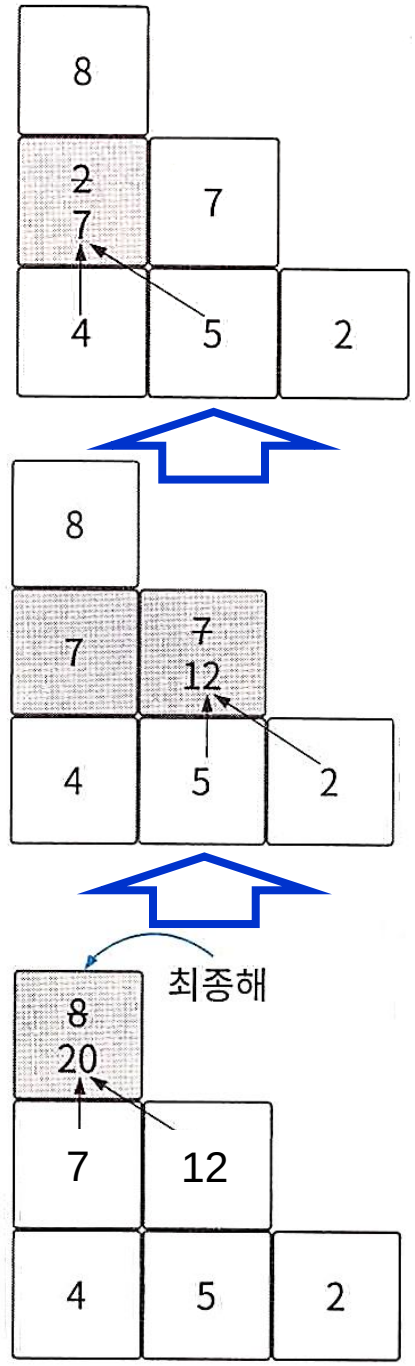
문제 3. 문제 해결 전략 수립

- **Top-down approach**
  - ➢ 맨 위에서부터 밑으로 내려오면서 합을 구하는 과정
    - ① 현재 위치의 값을 기준으로 바로 아래 인접한 두 값에 대한 합을 계산하고, 이 결과로 이들의 값을 갱신한다.
    - ② 아래로 이동한다.
    - ③ 위의 단계 ① - ②를 반복한다.
  - ➢ 장점: 직관적이고 바로 떠올리기 쉬운 방법이다.
  - ➢ 단점: 아래로 내려갈 수록 갈 수 있는 숫자의 범위가 많아진다.
    - 즉, 계산량이 지속적으로 증가한다는 의미이므로 이를 신중히 고려할 필요가 있다.



문제 3. 문제 해결 전략 수립

- **Bottom-up approach**
  - ➢ 맨 아래에서부터 위로 올라오면서 합을 구하는 과정
    - ① 현재 위치의 값을 기준으로 바로 위에 값에 대한 합을 계산한다.
    - ② 이때, 값의 갱신은 두 합의 결과 중에서 가장 큰 값으로 갱신된다.
    - ③ 위의 단계 ① - ②를 반복한다.
  - ➢ 장점: 위로 올라갈 수록 숫자의 범위가 좁아진다.
    - 최종해는 맨 위에 있는 값임.
    - 위로 올라갈수록 고려해야 할 경우의 수가 점점 줄어 효율적임.
  - ➢ 단점: 경우에 따라 top-down approach 보다는 덜 직관적일 수 있다.

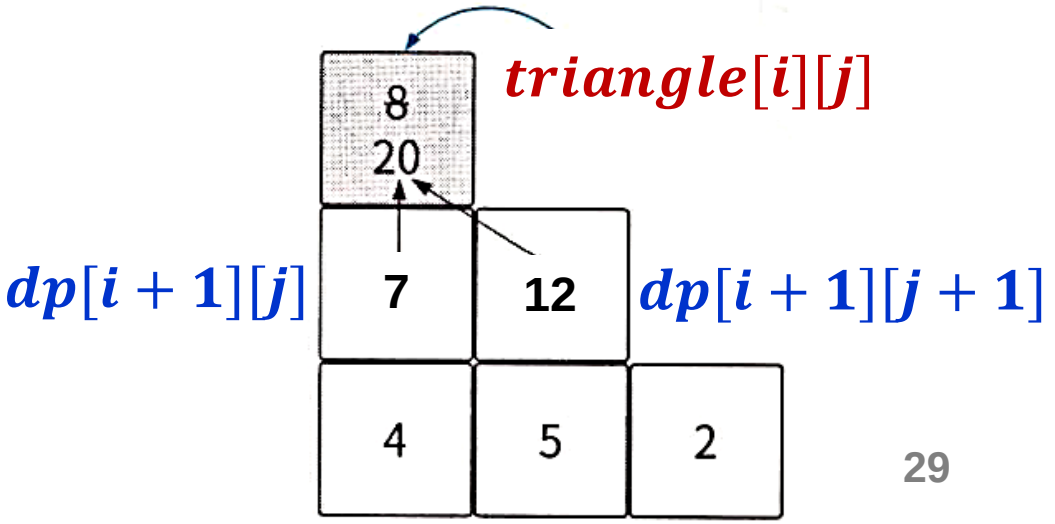


문제 3. 문제 해결 전략 수립

- **Bottom-up approach의 점화식 유도**
  - 삼각형의 맨 아래에서부터 i행 j열 위치까지 거쳐간(=이미 계산된) 숫자의 합 중에서 최대값: $\mathbf{dp}[i][j]$
  - 삼각형 내 i행 j열에 위치하는 원래의 값: $\mathbf{triangle}[i][j]$
  - 점화식

$$\mathbf{dp}[i][j] = \max(\mathbf{dp}[i+1][j], \mathbf{dp}[i+1][j+1]) + \mathbf{triangle}[i][j]$$

  - 최종해는 삼각형의 맨 위의 값인 $\mathbf{dp}[0][0]$이다.



문제 3. 구현 및 검증

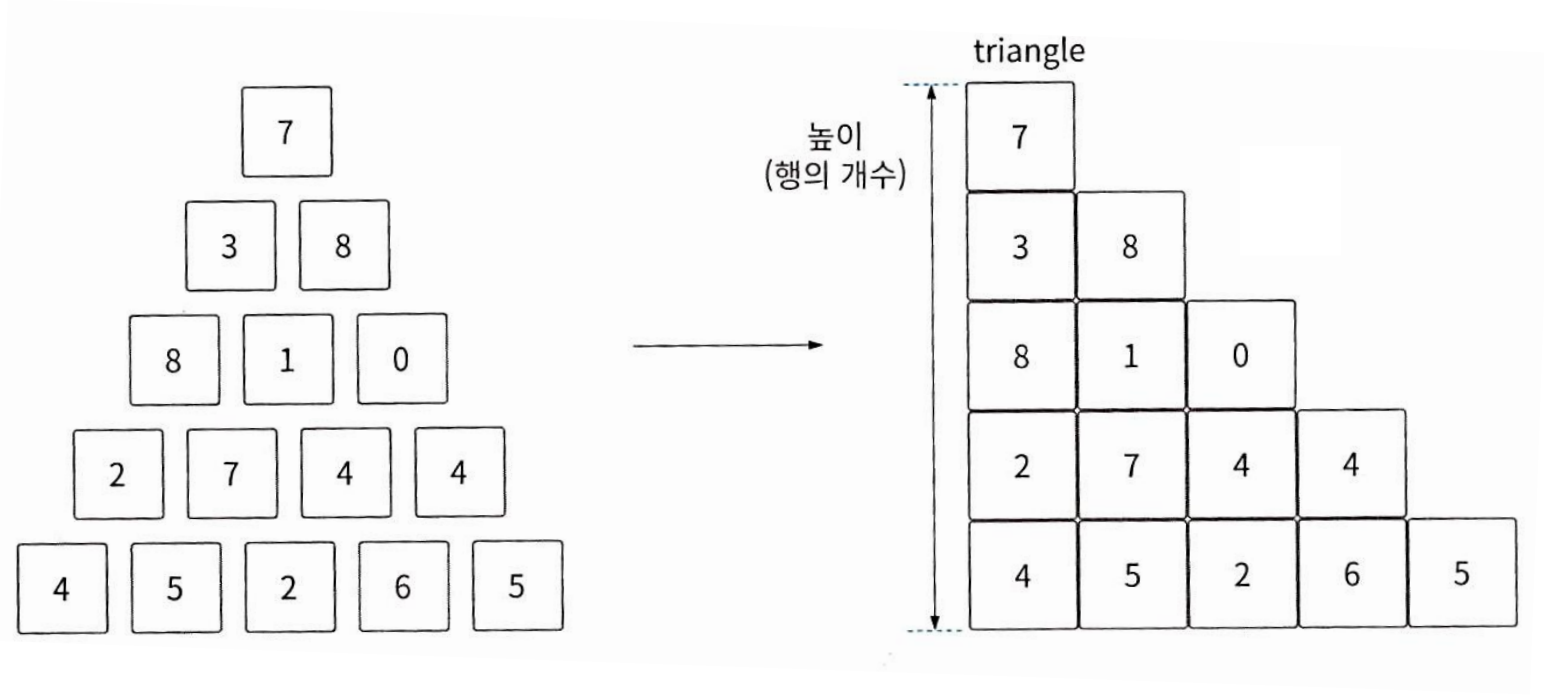
```
def solution3(triangle):
    n = len(triangle)
    dp = [[0] * n for _ in range(n)]  # 1. dp 테이블 초기화

    # 2. dp 테이블의 맨 아래쪽 라인 초기화
    for i in range(n):
        dp[n-1][i] = triangle[n-1][i]

    # 3. 아래쪽 라인부터 올라가면서 dp 테이블 채우기
    for i in range(n-2, -1, -1):
        for j in range(i + 1):
            dp[i][j] = max(dp[i+1][j], dp[i+1][j+1]) + triangle[i][j]

    return dp[0][0]  # 4. 최종 결과 값 반환
```

문제 3. 구현 및 검증 – 상세 설명

```
n = len(triangle)
dp = [[0] * n for _ in range(n)]  # 1. dp 테이블 초기화
```

① 삼각형은 정삼각형이므로 세 변의 길이가 같음.

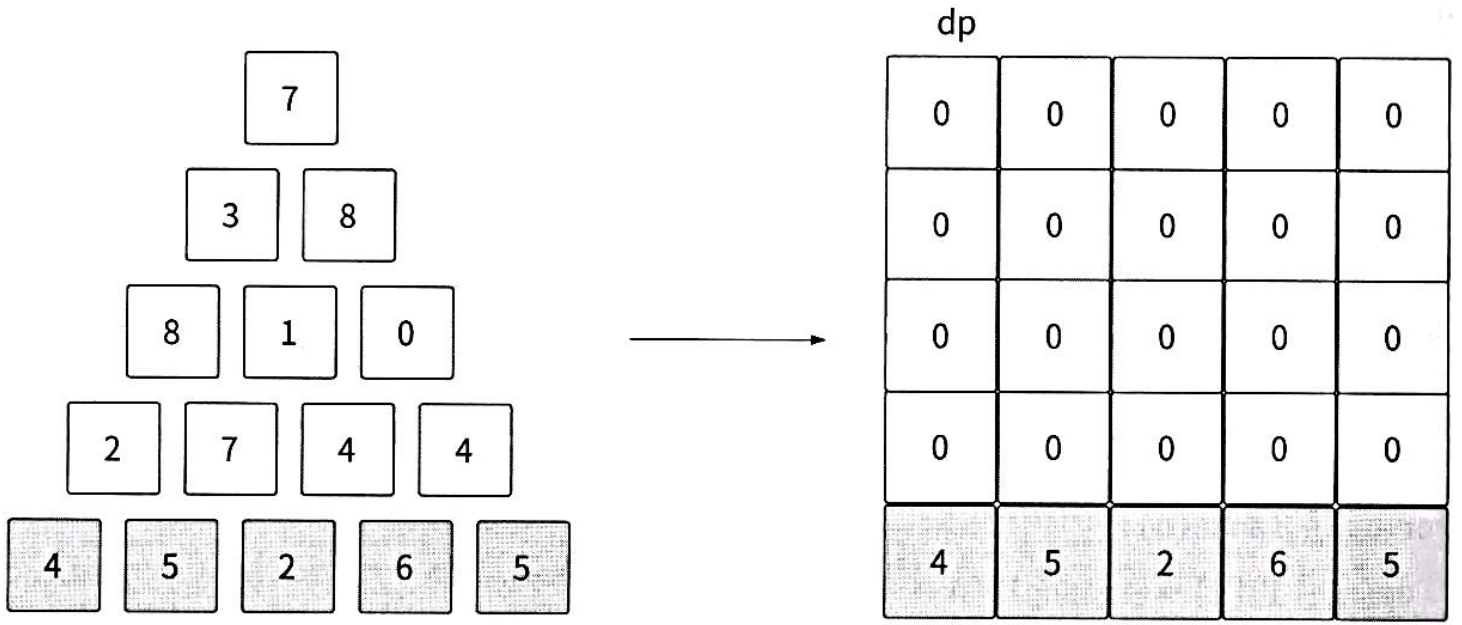
- 따라서, 코드에서 `len(triangle)`은 아래 그림과 같이 **<u>높이</u>**<u>(= 행의 개수)</u>를 나타냄.



문제 3. 구현 및 검증 – 상세 설명

```
# 2. dp 테이블의 맨 아래쪽 라인 초기화
for i in range(n):
    dp[n-1][i] = triangle[n-1][i]
```

② 맨 아래의 숫자는 탐색의 시작 지점이므로 dp 배열에 미리 초기화함.



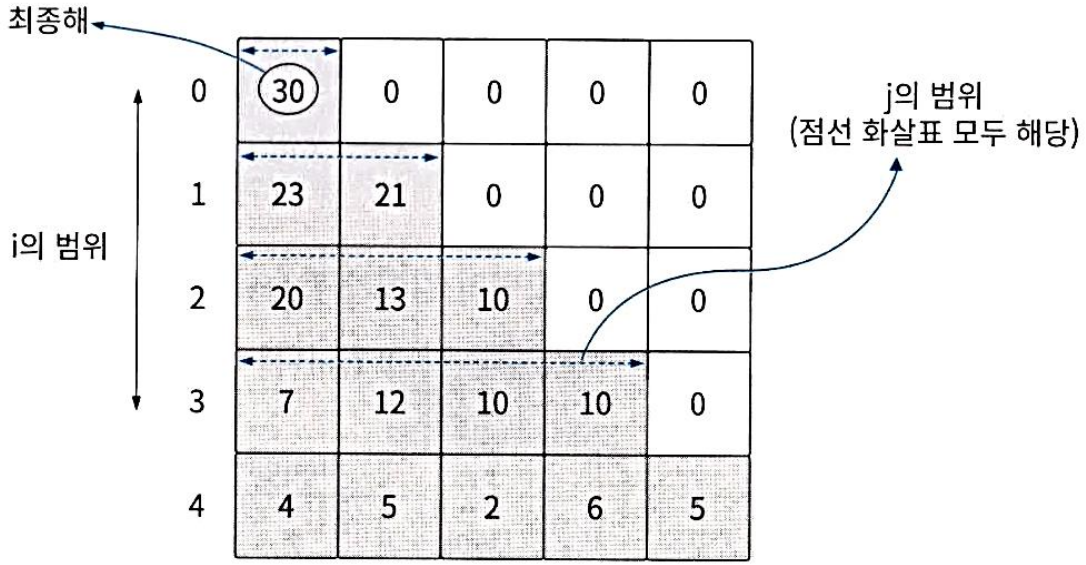
문제 3. 구현 및 검증 – 상세 설명

```
# 3. 아래쪽 라인부터 올라가면서 dp 테이블 채우기
for i in range(n-2, -1, -1):
    for j in range(i + 1):
        dp[i][j] = max(dp[i+1][j], dp[i+1][j+1]) + triangle[i][j]
```

③ 앞서 살펴본 아래 점화식을 그대로 구현한 부분

$$\mathbf{dp}[i][j] = \max(\mathbf{dp}[i+1][j], \mathbf{dp}[i+1][j+1]) + \mathbf{triangle}[i][j]$$

➢ 실제 계산 과정을 그림으로 표현하면 다음과 같음.



문제 3. 구현 및 검증

테스트 케이스

```
triangle = [[7], [3, 8], [8, 1, 0], [2, 7, 4, 4], [4, 5, 2, 6, 5]]
print( solution3(triangle) )
```

Q & A

한 학기동안 수고하셨습니다.