

과제2 - "가장 긴 증가하는 부분 수열"

문제 풀이 방법 (알고리즘)

입력된 정수 수열 S 가 있고, 이 수열의 i 번째 원소 $s[i]$ 로 끝나는 가장 증가 부분 수열의 길이를 찾아 h 배열에 저장한다. 따라서, $h[i]$ 는 $s[i]$ 로 끝나는 가장 증가 부분 수열의 길이가 된다. 이 중 가장 증가 부분 수열의 길이를 찾기 위해서는, 수열 S 의 첫번째 원소부터 시작하여 마지막 원소까지 h 값을 찾은 후 가장 큰 값을 골라주면 된다. 다만 알고리즘의 일반화를 위하여 수열의 0번째 index에 전체 수열의 원소들보다 충분히 작은 값을 추가해주고, $s[0]$ 의 h 값은 0으로 초기화하였다. 또한, 배열을 하나 더 생성하여 해당 부분 수열의 원소 값 직전에 등장하는 원소의 index를 저장해두면 가장 증가 부분 수열을 이루는 원소들을 차례로 출력할 수 있다. 이는 직전에 계산된 h 값을 이용하는 재귀적인 방식을 취하고 있으므로, 동적 계획법을 이용한 문제 풀이 방법이라고 할 수 있다.

소스코드 설명

```
#입력 : 수열이 크기, 정수수열 s
#출력 : 가장 증가 부분 수열 lis, 그 길이 maxValue

#i=0 인 경우도 저장할 수 있어야 하므로, 입력으로 받은 수열의 크기보다 1 큰 값을 MAX
변수에 저장한다.
MAX = int(input())+1
#입력받은 수열 s 를 정수 리스트 형태로 저장한다.
s = [int(k) for k in (input()).split()]
s.insert(0,-10)    #0 번째 수열에는 충분히 작은 -10 을 저장

h=[0 for row in range(MAX)] #h 는 가장 증가 부분 수열의 길이를 저장하는 배열
p=[0 for row in range(MAX)] #p 는 부분수열에서 직전 값의 인덱스를 저장하는 배열

#해당 원소 전까지 가장 증가 부분 수열의 길이를 구하는 알고리즘
for i in range(1, MAX):
    for j in range(0, i):    #i 인덱스 전에 있는 원소들만 비교
        # s[i]보다 작은 값을 갖는 원소 중에서 가장 큰 h 값을 갖는 원소의 값을 찾아서
        if (s[i] > s[j] and h[i] < h[j] + 1):
            h[i] = h[j] + 1    #그보다 1 큰 값으로 h 값을 업데이트 한다.
            p[i] = j    #부분수열의 직전에 나오는 값의 index 를 저장한다.

#부분수열의 가장 긴 길이 값을 찾는 알고리즘
maxValue = 0    #가장 증가 부분 수열의 길이를 나타내는 변수, 0 으로 초기화
for i in range(1, MAX):
    if (maxValue < h[i]):
        maxValue = h[i]
    #index 변수는 가장 증가 부분 수열의 마지막 원소의 인덱스로 설정해 놓는다.
```

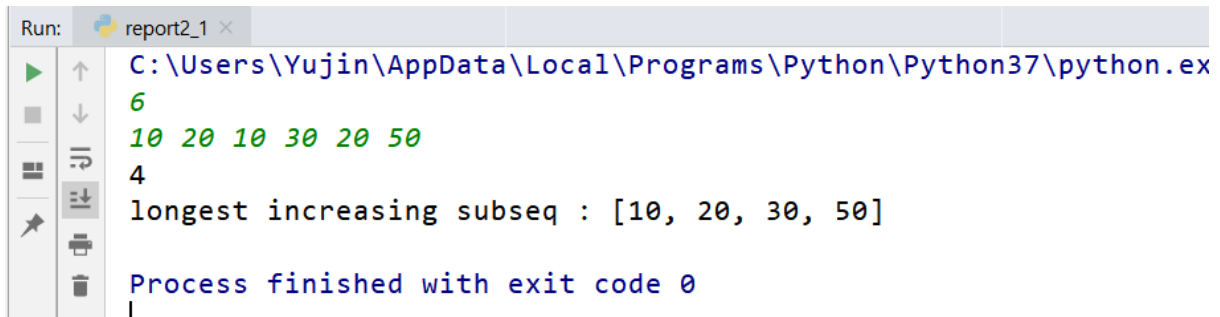
```

        index = i
    print(maxValue) #최장 증가 부분 수열의 길이값 출력

#최장 증가 부분 수열을 찾아 출력하는 알고리즘
lis = [] #최장 증가 부분 수열을 담을 변수 생성
while (index != 0) : #0 번째 인덱스에 도달할 때 까지 반복
    lis.append(s[index]) #최장 증가 부분 수열의 원소를 찾아 추가
    # p 배열을 이용하여, 최장 증가 부분 수열 원소의 직전 원소들의 index 값을 찾아 변수를
    업데이트 시켜준다.
    index = p[index]
lis.reverse() #가장 큰 수부터 거꾸로 담기게 되므로 역순으로 뒤집어준다.
print("longest increasing subseq :",lis) #최장 증가 부분 수열 출력

```

실행결과(예시)



```

Run: report2_1 x
C:\Users\Yujin\AppData\Local\Programs\Python\Python37\python.exe
6
10 20 10 30 20 50
4
longest increasing subseq : [10, 20, 30, 50]
Process finished with exit code 0

```

위와 같이 수열의 크기(6)와 수열 값(10, 20, 10, 30, 20, 50)이 입력으로 주어졌을 때,

최장 증가 수열의 길이는 4 이고, 수열은 [10, 20, 30, 50]임을 결과로 출력하게 된다.

과제2 - "정수삼각형"

문제 풀이 방법 (알고리즘)

주어진 정수삼각형의 배열을 tri 라고 하고, tri[r][c] 를 r번째 열의 c번째 행에 존재하는 원소라고 가정한다. 그렇다면 tri[r][c]에는 그보다 더 상위 레벨에 있는 tri[r-1][c-1]번째 원소 혹은 tri[r-1][c]를 통해서만 도착할 수 있다. 그렇다면 이 두 값 중에서 최적값이 더 큰 값을 갖는 경로를 택해서 다음 원소가 갖는 최적값을 업데이트 해 나간다면, 마지막 레벨에 도달했을 때 가장 큰 최적값을 갖는 원소가 전체 삼각형의 최적 경로를 형성하게 된다.

이러한 논리를 구현하려면, 직전 레벨의 원소의 최적값이 모두 계산되어 있다는 가정이 필요하다.

따라서 레벨 1에 있는 원소부터 시작하여, 레벨 r에 있는 c번째 원소에서 끝나는 최적의 경로의 값을 구하는 알고리즘을 설계하고, 재귀적인 방법을 활용하여 동적 계획법으로 문제를 해결하였다. 또한 직전 경로를 추적할 수 있도록 'R'과 'L' 문자를 별도의 배열에 저장하여, 이를 활용하여 전체 숫자 삼각형의 최적 경로를 출력할 수 있도록 하였다.

소스코드 설명

```
#입력 : n*n 2 차원 배열 tri, 최대 레벨 n
#출력 : 최적의 경로의 값 maxResult, 경로에 있는 수 bestPath

n = int(input())    #삼각형의 크기 n 을 입력받는다.

tri= []             # 빈 tri 배열 생성
# 정수 삼각형을 입력받아서 tri 배열 안에 정수 리스트 형태로 저장한다.
for i in range(n) :
    tri.append([int(k) for k in input().split()])

# 각 원소까지 최적 경로의 값을 저장할 best 배열을 생성, 모든 원소 -1 로 초기화
best = [[-1 for col in range(n)] for row in range(n)]
# path 배열을 생성, 모든 원소는 빈 str 값으로 초기화
path = ["" for col in range(n)] for row in range(n)]

#best 의 첫번째 값에는 삼각형의 원소 값이 그대로 들어간다.
best[0][0] = tri[0][0]

maxResult = 0       # 전체 숫자삼각형의 최적 경로 값, 0 으로 초기화

# 각 원소까지 최적 경로를 택하는 알고리즘
for i in range(1,n) :
    for j in range (0,i+1) :
        best[i][j] = tri[i][j]+ max(best[i-1][j-1],best[i-1][j])

# 마지막 레벨의 best 배열 안에 담긴 값 중 가장 큰 값을 maxResult 로 저장
# 여기서 i 가 0 부터 시작하기 때문에 n-1 값이 마지막 레벨이 된다.
for i in range(0,n) :
    if (maxResult<best[n-1][i]) :
        maxResult = best[n-1][i]
        # 최적 경로의 도착지점에 해당하는 행의 index 를 pathIndex 변수에 저장
        pathIndex = i

#전체 숫자삼각형의 최적 경로 값 출력
print(maxResult)

#best 배열의 값을 이용하여 각 원소까지 최적알고리즘이 지나온 경로('R', 'L')를 저장하는
#알고리즘
for i in range(n) :
    for j in range (n) :
        # 제일 처음 원소는 출발점이므로 그대로 비워둔다.
        if (i == 0 and j == 0):
            pass
        # i=j 일 경우, 무조건 왼쪽에서 내려오는 값을 택하게 된다.
```

```

elif (j == i):
    path[i][j] = "L"
# j=0 일 경우, 무조건 오른쪽에서 내려오는 값을 택하게 된다.
elif (j == 0):
    path[i][j] = "R"
elif (j <= i):
    # 바로 위 level 의 왼쪽 원소값이 더 클 경우 왼쪽의 값을 택한다.
    if (best[i-1][j-1] > best[i-1][j]):
        path[i][j] = "L"
    # 그렇지 않을 경우 오른쪽의 값을 택한다.
    else:
        path[i][j] = "R"

#path 배열에 저장된 값을 통해 최적의 경로를 찾아, bestPath 변수에 저장하는 알고리즘
#리스트는 인덱스가 0 번부터 시작하므로, 최대 레벨 n 에서 1 번째 값을 level 변수에
저장한다.
level = n-1
bestPath = "" #최적의 경로에 있는 수를 출력할 str type 의 변수를 생성.
while(level != 0) : #최상위 레벨에 도달할 때까지 반복한다.
    bestPath += str(tri[level][pathIndex]) + "->" #경로의 원소 값을 추가한다.
    # 현재 지점의 path 에 저장된 값이 L 이면 (상위 level 의 왼쪽에서 온 경로라면)
    if path[level][pathIndex] == "L":
        pathIndex -= 1 #pathIndex 변수를 1 감소시킨다.
        # 그렇지 않으면 pathIndex 값을 그대로 유지한다.
    level -= 1 #한 단계 위의 level 로 옮겨간다.

#최적의 경로에 있는 수를 (제일 하위의 level 부터) 출력한다.
#어떤 경로든 출발 지점은 일정하므로, 알고리즘을 이용해 찾은 경로값에 tri[0][0] 값을
추가하여 출력해주면 된다.
print("print optimal path from the bottom :",bestPath + str(tri[0][0]) +
"(starting point)")

```

실행결과 (예시)

```

Run: report2_2 x
C:\Users\Yujin\AppData\Local\Programs\Python\Python37\python.exe "C:/Users/
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
30
print optimal path from the bottom : 5->7->8->3->7(starting point)

Process finished with exit code 0

```

위와 같이 최대 레벨 $n = 5$ 의 크기를 갖는 정수삼각형이 입력되었을 때 (2~6 번째 줄),

최적 경로의 값은 30 이고, 그 경로에 있는 수는 아래서부터 순서대로 5->7->8->3->7 임을
결과로 출력한다.

소스코드 파일 (QR코드 첨부)




< 과제2


Q ...

Name ↑

☰

 report2_1.py
⚙ Modified Apr 30, 2020

...

 report2_2.py
⚙ Modified Apr 30, 2020

...