[Algorithm] 29강 : 다이나믹 프로그래밍의 기초 문제 풀이 — 나무늘보의 개발 블로그

노트북: 첫 번째 노트북
만든 날짜: 2020-11-17 오후 5:25
URL: https://continuous-development.tistory.com/193?category=736684

Algorithm

[Algorithm] 29강 : 다이나믹 프로그래밍의 기초 문제 풀이

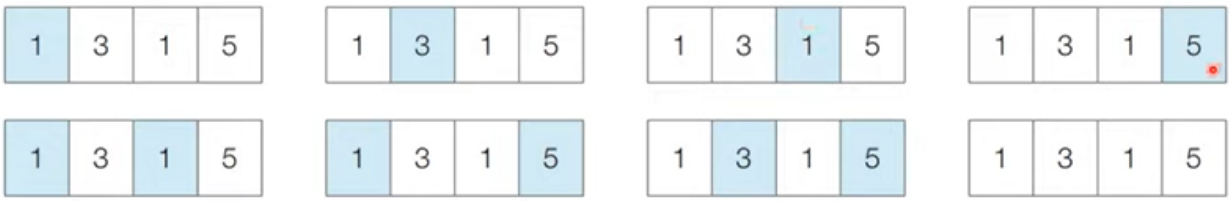
2020. 11. 17. 17:23 수정 삭제 공개

<문제> 개미 전사 : 문제설명

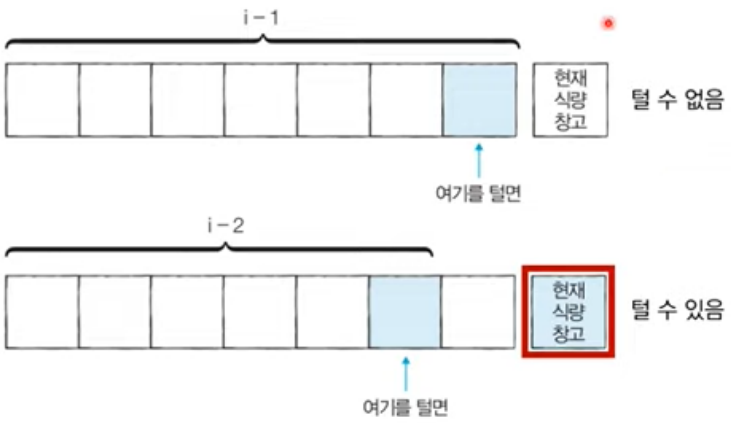
- 개미 전사는 부족한 식량을 충당하고자 메뚜기 마을의 식량창고를 몰래 공격하려고 합니다. 메뚜기 마을에는 여러 개의 식량창고가 있는데 식량창고는 일직선으로 이어져 있습니다.
- 각 식량창고에는 정해진 수의 식량을 저장하고 있으며 개미 전사는 식량창고를 선택적으로 약탈하여 식량을 빼앗을 예정입니다. 이때 메뚜기 정찰병들은 일직선상에 존재하는 식량창고 중에서 서로 인접한 식량창고가 공격받으면 바로 알아챌 수 있습니다.
- 따라서 개미 전사가 정찰병에게 들키지 않고 식량창고를 약탈하기 위해서는 최소한 한 칸 이상 떨어진 식량창고를 약탈해야 합니다.

문제 해결 아이디어

- 예시를 확인해 봅시다. **N = 4**일 때, 다음과 같은 경우들이 존재할 수 있습니다.
  - 식량을 선택할 수 있는 경우의 수는 다음과 같이 8가지입니다.
  - 7번째 경우에서 8만큼의 식량을 얻을 수 있으므로 **최적의 해는 8**입니다.



- 왼쪽부터 차례대로 식량창고를 턴다고 했을 때, 특정한 i번째 식량창고에 대해서 털지 안 털지의 여부를 결정하면, 아래 **2가지** 경우 중에서 더 많은 식량을 털 수 있는 경우를 선택하면 됩니다.



큰 문제를 위해 작은 문제를 통해 해결

즉 이 문제의 점화식은

$$a_i = max(a_{i-1}, a_{i-2} + k_i)$$

이렇게 된다.

문제 풀이

```
# 정수 N을 입력 받기
n = int(input())
# 모든 식량 정보 입력 받기
array = list(map(int,input().split()))

# 앞서 계산된 결과를 저장하기 위한 DP 테이블 초기화
d = [0] * 100

# 다이나믹 프로그래밍(Dynamic Programming) 진행(보텀업)
```

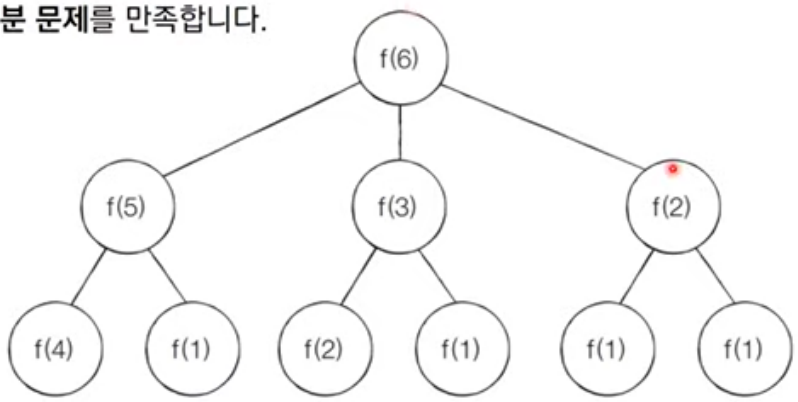
```
d[0] = array[0]
d[1] = max(array[0], array[1])
for i in range(2,n):
    d[i] = max(d[i-1],d[i-2] + array[i])

계산된 결과 출력
print(d[n-1])
```

<문제> 1로 만들기

- 정수 X가 주어졌을 때, 정수 X에 사용할 수 있는 연산은 다음과 같이 **4가지**입니다.
  1. X가 5로 나누어 떨어지면, 5로 나눕니다.
  2. X가 3으로 나누어 떨어지면, 3으로 나눕니다.
  3. X가 2로 나누어 떨어지면, 2로 나눕니다.
  4. X에서 1을 뺍니다.
- 정수 X가 주어졌을 때, 연산 4개를 적절히 사용해서 값을 1로 만들고자 합니다. 연산을 사용하는 횟수의 최솟값을 출력하세요. 예를 들어 정수가 26이면 다음과 같이 계산해서 3번의 연산이 최솟값입니다.
  - 26 → 25 → 5 → 1

문제 해결 아이디어

- 피보나치 수열 문제를 도식화한 것처럼 함수가 호출되는 과정을 그림으로 그려보면 다음과 같습니다.
  - **최적 부분 구조**와 **중복되는 부분 문제**를 만족합니다.



이렇게 각각의 경우의 수에서 가장 적은 값을 찾으면 된다.

이 해당하는 문제에 대한 점화식은

$$a_i = min(a_{i-1}, a_{i/2}, a_{i/3}, a_{i/5}) + 1$$

이 값이 된다.

문제 풀이

```
#정수 x를 입력 받기
x = int(input())

# 앞서 계산된 결과를 저장하기 위한 DP 테이블 초기화
d = [0] * 30001

# 다이나믹 프로그래밍(Dynamic Programming) 진행
for i in range(2,x+1):
    # 현재의 수에서 1을 빼는 경우
    d[i] = d[i-1]+1
    #현재의 수가 2로 나누어 떨어지는 경우
    if i % 2 == 0:
        d[i] == min(d[i],d[i//2] +1)
    #현재의 수가 3으로 나누어 떨어지는 경우
    if i % 3 ==0:
        d[i] == min(d[i],d[i//3]+1)
    #현재의 수가 5로 나누어 떨어지는 경우
    if i % 5 ==0:
        d[i] == min(d[i],d[i//5]+1)

print(d[x]
```

<문제> 효율적인 화폐구성

- N가지 종류의 화폐가 있습니다. 이 화폐들의 개수를 최소한으로 이용해서 그 가치의 합이 M원이 되도록 하려고 합니다. 이때 각 종류의 화폐는 몇 개라도 사용할 수 있습니다.
- 예를 들어 2원, 3원 단위의 화폐가 있을 때는 15원을 만들기 위해 3원을 5개 사용하는 것이 가장 최소한의 화폐 개수입니다.
- M원을 만들기 위한 최소한의 화폐 개수를 출력하는 프로그램을 작성하세요.

문제 해결 아이디어

- a_i = 금액 i를 만들 수 있는 최소한의 화폐 개수
- k = 각 화폐의 단위
- **점화식**: 각 화폐 단위인 k를 **하나씩 확인하며**
  - a_{i-k}를 만드는 방법이 존재하는 경우, $a_i = \min(a_i, a_{i-k} + 1)$
  - a_{i-k}를 만드는 방법이 존재하지 않는 경우, $a_i = INF$

해당 점화식을 가진다. 이 경우에는 각 화폐의 단위를 확인한다.
2, 3, 5 를 모두 돌아가면서 해당하는 부분을 확인한다.

- $N = 3$, $M = 7$이고, 각 화폐의 단위가 2, 3, 5인 경우 확인해 봅시다.
- Step 1
  - 첫 번째 화폐 단위인 2를 확인합니다.
  - 점화식에 따라서 다음과 같이 리스트가 갱신됩니다.

인덱스	0	1	2	3	4	5	6	7
값	0	10,001	1	10,001	2	10,001	3	10,001

4원을 만들기 위한 개수는 2개: (2원 + 2원)

7원을 만드는 방법이 없음

이런 식으로 모든 단위에다가 진행한다.

- $N = 3$, $M = 7$이고, 각 화폐의 단위가 2, 3, 5인 경우 확인해 봅시다.
- Step 3
  - 세 번째 화폐 단위인 5를 확인합니다.
  - 점화식에 따라서 다음과 같이 최종적으로 리스트가 갱신됩니다.

인덱스	0	1	2	3	4	5	6	7
값	0	10,001	1	1	2	1	2	2

7원을 만들기 위한 개수는 2개: (2원 + 5원)

문제 풀이

```
# 정수 N,M을 입력 받기
n,m = map(int,input().split())

#N개의 화폐 단위 정보를 입력 받기

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])
```

<문제> 금광 문제 설명

- n × m 크기의 금광이 있습니다. 금광은 1 × 1 크기의 칸으로 나누어져 있으며, 각 칸은 특정한 크기의 금이 들어 있습니다.
- 채굴자는 첫 번째 열부터 출발하여 금을 캐기 시작합니다. 맨 처음에는 첫 번째 열의 어느 행에서든 출발할 수 있습니다. 이후에 m - 1번에 걸쳐서 매번 오른쪽 위, 오른쪽, 오른쪽 아래 3가지 중 하나의 위치로 이동해야 합니다. 결과적으로 **채굴자가 얻을 수 있는 금의 최대 크기**를 출력하는 프로그램을 작성하세요.

1	3	3	2
2	1	4	1
0	6	4	7

얻을 수 있는 금의 최대 크기: 19

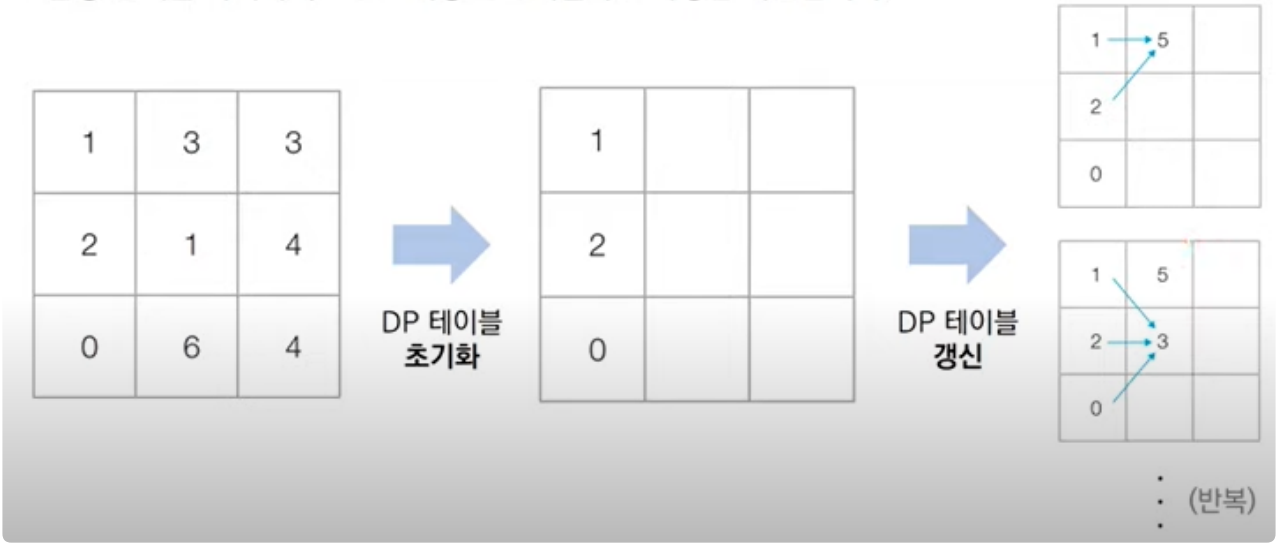
문제 해결 아이디어

금광의 모든 위치에 대하여 다음의 세 가지만 고려하면 된다.
1. 왼쪽 위에서 오는 경우
2. 왼쪽 아래에서 오는 경우
3. 왼쪽에서 오는 경우
세 가지 경우중에서 가장 많은 금을 가지고 있는 경우를 테이블에 갱신해주어 문제를 해결
해당 문제의 점화식은

$$dp[i][j] = array[i][j] + max(dp[i - 1][j - 1], dp[i][j - 1], dp[i + 1][j - 1])$$

이렇게 구성된다.

- 금광 문제를 다이나믹 프로그래밍으로 해결하는 과정을 확인합시다.



이 과정을 계속 해서 진행한다.

문제 풀이

```
# 테스트 케이스(Test case) 입력
for tc in range(int(input())):
    #금광 정보입력
    n,m = map(int(input().split())
    array = list(map(int,input().split()))
    # 다이나믹 프로그래밍을 위한 2차원 DP 테이블 초기화
    dp = []
    index = 0
    for i in range(n):
        dp.append(array[index:index +m])
        index += m
    # 다이나믹 프로그래밍 진행
    for j in range(1,m):
        for i in range(n):
            # 왼쪽 위에서 오는 경우
            if i ==0:
                left_up =0
            else:
                left_up = dp[i-1][j-1]
                #왼쪽 아래에서 오는 경우
            if i == n-1 :
                left_down = 0
            else:
                left_down = dp[i+1][j-1]
                # 왼쪽에서 오는 경우
            left = dp[i][j-1]
            dp[i][j] = dp[i][j] + max(left_up, left_down, left)
        result = 0
        for i in ragne(n):
            result = manx(result, dp[i][m-1])
        print(result)
```

<문제> 병사 배치하기

- N명의 병사가 무작위로 나열되어 있습니다. 각 병사는 특정한 값의 전투력을 보유하고 있습니다.
- 병사를 배치할 때는 **전투력이 높은 병사가 앞쪽에 오도록 내림차순으로 배치**를 하고자 합니다. 다시 말해 앞쪽에 있는 병사의 전투력이 항상 뒤쪽에 있는 병사보다 높아야 합니다.
- 또한 배치 과정에서는 특정한 위치에 있는 병사를 열외시키는 방법을 이용합니다. 그러면서도 남아 있는 병사의 수가 최대가 되도록 하고 싶습니다.

문제 해결 아이디어

이 문제의 기본 아이디어는 가장 긴 증가하는 부분 수열(Longest Increasing Subsequence, LIS)로 알려진 전형적인 다이나믹 프로그래밍 문제이다.
해당 식의 점화식은

$$\text{모든 } 0 \le j < i \text{ 에 대하여, } D[i] = max(D[i],\ D[j] + 1) \text{ if } array[j] < array[i]$$

이렇게 구성 된다.

```
n = int(input())
array = list(map(int, input().split()))

#순서를 뒤집어 최장 증가 부분 수열 문제로 변환
array.reverse()

# 다이나믹 프로그래밍을 위한 1차원 DP 테이블 초기화
dp = [1] * n

# 가장 긴 증가하는 부분 수열 알고리즘 수행
for i in range(1,n):
    for j in ragne(0,i):
        if array[j] < array[i]:
            dp[i] = max(dp[i],dp[j]+1)

# 열외해야 하는 병사의 최소 수를 출력
print(n - max(dp))
```

이 자료는 동빈 나 님의 이코 테 유튜브 영상을 보고 정리한 자료입니다.
www.youtube.com/watch?v=m-9pAwq1o3w&list=PLRx0vPvlEmdAghTr5mXQxGpHjWqSz0dgC

'**Algorithm**' 카테고리의 다른 글

[Algorithm] 29강 : 다이나믹 프로그래밍의 기초 문제 풀이

[Algorithm] 28강 : 다이나믹 프로그래밍의 정의와 구현

[Algorithm] 27강 : 이진 탐색 기초 문제 풀이

[Algorithm] 26강 : 이진 탐색 알고리즘 정의와 구현

[Algorithm] 25강 : 정렬 알고리즘 복잡도 비교 및 기본 문제

[Algorithm] 24강 : 계수 정렬의 정의와 구현코드

다이나믹 문제풀이 다이나믹 프로그래밍 다이나믹 프로그래밍 문제풀이

나아무늘보

혼자 끄적끄적하는 블로그 입니다.