Algorithm Study

03.10.2024 김홍주

#2342. Dance Dance Revolution /Gold3

Q. #2342. Dance Dance Revolution (Gold 3)

Dance Dance Revolution 성공

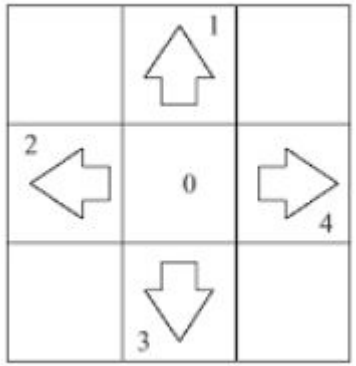
☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	17420	6858	5028	38.408%

문제

승환이는 요즘 "Dance Dance Revolution"이라는 게임에 빠져 살고 있다. 하지만 그의 춤 솜씨를 보면 알 수 있듯이, 그는 DDR을 잘 하지 못한다. 그럼에도 불구하고 그는 살을 뺄 수 있다는 일념으로 DDR을 즐긴다.

DDR은 아래의 그림과 같은 모양의 발판이 있고, 주어진 스텝에 맞춰 나가는 게임이다. 발판은 하나의 중점을 기준으로 위, 아래, 왼쪽, 오른쪽으로 연결되어 있다. 편의상 중점을 0, 위를 1, 왼쪽을 2, 아래를 3, 오른쪽을 4라고 정하자.



<문제요약 >

1. 처음에 게이머는 두 발을 중앙에 모으고 있다.(그림에서 0의 위치)

2. 게임이 시작하면, 지시에 따라 왼쪽 또는 오른쪽 발을 움직인다. 하지만 그의 두 발이 동시에 움직이지는 않는다.

3. 두 발이 같은 지점에 있는 것이 허락되지 않는 것이다. (물론 게임 시작시에는 예외이다)

   만약, 한 발이 1의 위치에 있고, 다른 한 발이 3의 위치에 있을 때, 3을 연속으로 눌러야 한다면, 3의 위치에 있는 발로 반복해야 눌러야 한다는 것이다.

4.발이 움직이는 위치에 따라서 드는 힘이 다르다

   (1) 만약 같은 지점을 한번 더 누른다면
   (2) 중앙에 있던 발이 다른 지점으로 움직일 때
   (3) 다른 지점에서 인접한 지점으로 움직일 때 (예를 들면 왼쪽에서 위나 아래로 이동할 때의 이야기이다.)
   (4) 반대편(맞은편)으로 움직일 때 (위쪽에서 아래쪽으로, 또는 오른쪽에서 왼쪽으로)

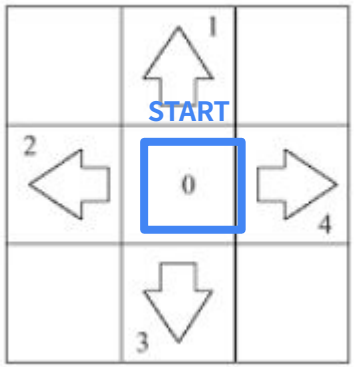
Q. #2342. Dance Dance Revolution (Gold 3)

예제 입력 1 복사

```
1 2 2 4 0
```

예제 출력 1 복사

```
8
```



만약 1 → 2 → 2 → 4를 눌러야 한다고 가정해 보자.

당신의 두 발은 처음에 (point 0, point 0)에 위치하여 있을 것이다.

그리고 (0, 0) → (0, 1) → (2, 1) → (2, 1) → (2, 4)로 이동하면, 당신은 8의 힘을 사용하게 된다.

다른 방법으로 발을 움직이려고 해도, 당신은 8의 힘보다 더 적게 힘을 사용해서 1 → 2 → 2 → 4를 누를 수는 없을 것이다.

<입력 조건 >

- 입력은 지시 사항으로 각각의 지시 사항은 하나의 수열로 이루어진다
- 각각의 수열은 1, 2, 3, 4의 숫자들로 이루어지고, 이 숫자들은 각각의 방향을 나타낸다.
- 입력 파일의 마지막에는 0이 입력된다.
- 입력 수열의 길이 < 100,000

<출력 조건>

- 한 줄에 모든 지시 사항을 만족하는 데 사용되는 최소의 힘을 출력한다..

Hint

1. 문제 유형 : DP

- 부분 문제 O
- Greedy X
  - 이전 위치에 따라 현 위치 이동 까지 드는 power양이 다름
  - 그리디는 각 단계 기준 최선의 선택, 즉 현재 상태가 미래에 영향을 주지 않음
- DP
  - 모든 단계를 고려함

  => DP에 단계(level)별로 왼발 & 오른발 위치(1~4) 정보 함께 저장

Hint

1. 문제 유형 : DP
2. 3차원 DP 테이블 **dp[level][R최종 위치][L 최종 위치] = 누적 power**

[Level=0]

R/L	0	1	2	3	4
0	0	INF	INF	INF	INF
1	INF	INF	INF	INF	INF
2	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF

[Level=1] 0 -> 1

R/L	0	1	2	3	4
0	INF	0-> 2	INF	INF	INF
1	0-> 2	INF	INF	INF	INF
2	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF

[Level=2] 1->2

R/L	0	1	2	3	4
0	INF	2->5	2->4	INF	INF
1	2->5	INF	INF	INF	INF
2	INF	2->4	INF	INF	INF
3	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF

My Solution

1. **3차원 DP 테이블 정의 & 초기화**
   dp[level][R최종 위치][L 최종 위치] = 누적 power

2. **반복문으로 dp 점화식(memorization) 구현**

점화식 :
(1) 누적power 업데이트
(2) 각 단계별 최종 (R,L)에 도달하기 위한 최소 누적 power 업데이트
- 오른발만 이동
- 왼발만 이동

- **getpower():**
  dp[level][R최종 위치][L 최종 위치]
  = dp[level-1][이전 R][이전 L] + 현 상황에서 드는 power

(2) 2가지 경우(오른 발만 이동할 때 / 왼쪽 발만 이동할 때),
누적 power 값 업데이트

- 5x5 반복 : 오른쪽, 왼쪽 모든 위치 기록

```
for r in range(5) :
  for l in range(5) :
        # 이전 업데이트 된 r, l 의 경우의 수에 한정(최적화)
    if dp[level-1][r][l] !=INF :
      cur_p = dp[level-1][r][l]
      # (1) 오른 발만 이동하는 경우
      dp[level][target][l]= min(dp[level][target][l],getPower(r,target,cur_p))
      #(2) 왼 발만 이동하는 경우
```

```
# 이전 지점에서 현재 지점으로 이동할때 드는 power 값 반환
def getPower(before_foot , current_foot , preview_power) :
  add_power = 0
  if before_foot == current_foot :
    add_power= 1
  elif  before_foot == 0 and current_foot !=0 :
    add_power= 2
  elif abs(before_foot-current_foot)==2 : # 인접 지점 누를 경우
    add_power=4
  else : # 반대편 지점 누를 경우
    add_power= 3
  return preview_power + add_power
```

📌 **같은 지점에 두 발이 있지 않는 경우**

- 있을 수 있지만, 해당 경우
  power(다른 위치→ 중복 위치로 이동≥2) >> power(같은 위치 머무르기 =1)
  최소가 될 수 없기 때문에 고려하지 않아도 됨

My Solution

1. **3차원 DP 테이블 정의 & 초기화**
   dp[level][R최종 위치][L 최종 위치] = 누적 power

2. **반복문으로 dp 점화식(memorization) 구현**

- **getpower():**
  dp[level][R최종 위치][L 최종 위치]
  = dp[level-1][이전 R][이전 L] + 현 상황에서 드는 power

```
# 이전 지점에서 현재 지점으로 이동할때 드는 power 값 반환
def getPower(before_foot , current_foot , preview_power) :
  add_power = 0
  if before_foot == current_foot :
    add_power= 1
  elif  before_foot == 0 and current_foot !=0 :
    add_power= 2
  elif abs(before_foot-current_foot)==2 : # 인접 지점 누를 경우
    add_power=4
  else : # 반대편 지점 누를 경우
    add_power= 3
  return preview_power + add_power
```

- Bottom up 구현

(1) 각 단계(level)별에서 밟아야 하는 목표(target) 칸 확인

(2) 바로 이전 단계에서 값이 업데이트 된 [r, l] 의 경우에 한정 탐색(선택)

- 시간 최적화 옵션 (if dp[level-1][r][l] !=INF)

(3) 2가지 경우(오른 발만 이동할 때 / 왼쪽 발만 이동할 때),
누적 power 값 업데이트

- 5x5 반복 : 오른쪽, 왼쪽 모든 위치 기록

```
for r in range(5) :
  for l in range(5) :
        # 이전 업데이트 된 r, l 의 경우의 수에 한정(최적화)
    if dp[level-1][r][l] !=INF :
      cur_p = dp[level-1][r][l]
      # (1) 오른 발만 이동하는 경우
      dp[level][target][l]= min(dp[level][target][l],getPower(r,target,cur_p))
      #(2) 왼 발만 이동하는 경우
      dp[level][r][target]= min(dp[level][r][target],getPower(l,target,cur_p))
```

📌 **같은 지점에 두 발이 있지 않는 경우**

- 있을 수 있지만, 해당 경우
  power(다른 위치→ 중복 위치로 이동≥2) >> power(같은 위치 머무르기 =1)
  최소가 될 수 없기 때문에 고려하지 않아도 됨

My Solution

```python
import sys
input = sys.stdin.readline
INF = int(1e9)
commands = list(map(int,input().split()))[:-1]

# 이전 지점에서 현재 지점으로 이동할때 드는 power 값 반환
def getPower(before_foot , current_foot , preview_power) :
  add_power = 0
  if before_foot == current_foot :
    add_power= 1
  elif  before_foot == 0 and current_foot !=0 :
    add_power= 2
  elif abs(before_foot-current_foot)==2 : # 인접 지점 누를 경우
    add_power=4
  else : # 반대편 지점 누를 경우
    add_power= 3
  return preview_power + add_power


#1.dp 초기화 (현재 L , R 위치 , 누적 power)
# 3차원 : dp[level][r][l] = 누적 power
dp = [[[INF for k in range(5)] for i in range(5)] for _ in range(len(commands)+1)]
dp[0][0][0] = 0

# # #2. 반복문으로 dp  점화식(memorization) 구현
cur_r ,cur_l = 0 ,0
# 각 게임 단계별로 업데이트
for level in range(1,len(commands)+1):
  target = commands[level-1] # 현 단계에서 이동할 자리

  for r in range(5) :
    for l in range(5) :
            # 이전 업데이트 된 r, l 의 경우의 수에 한정(최적화)
      if dp[level-1][r][l] !=INF :
        cur_p = dp[level-1][r][l]
        # (1) 오른 발만 이동하는 경우
        dp[level][target][l]= min(dp[level][target][l],getPower(r,target,cur_p))
        #(2) 왼 발만 이동하는 경우
        dp[level][r][target]= min(dp[level][r][target],getPower(l,target,cur_p))

# 3. 최종 단계에서 최소 힘 출력
result = INF
for i in range(5):
  for k in range(5):
    result = min(result,  dp[-1][i][k])

print(result)
```

1. **3차원 DP 테이블 정의 & 초기화**
   dp[level][R최종 위치][L 최종 위치] = 누적 power

2. **반복문으로 dp 점화식(memorization) 구현**

- **getpower():**
  dp[level][R최종 위치][L 최종 위치]
  = dp[level-1][이전 R][이전 L] + 현 상황에서 드는 power

- Bottom up 구현

(1) 각 단계(level)별에서 밟아야 하는 목표(target) 칸 확인

(2) 바로 이전 단계에서 값이 업데이트 된 [r, l] 의 경우에 한정 탐색(선택)

- 시간 최적화 옵션 (if dp[level-1][r][l] !=INF)

(3) 2가지 경우(오른 발만 이동할 때 / 왼쪽 발만 이동할 때),
누적 power 값 업데이트

- 5x5 반복 : 오른쪽, 왼쪽 모든 위치 기록

📌 **같은 지점에 두 발이 있지 않는 경우**

- 있을 수 있지만, 해당 경우
  power(다른 위치→ 중복 위치로 이동≥2) >> power(같은 위치 머무르기 =1)
  최소가 될 수 없기 때문에 고려하지 않아도 됨

다른 Solution

top-down (재귀) 구현

```
import sys
sys.setrecursionlimit(10**6)

def move(a, b):
    if a == b:
        return 1
    elif a == 0:
        return 2
    elif abs(b-a)%2 == 0:
        return 4
    else:
        return 3

def solve(n, l, r):
    global dp
    if n >= len(arr)-1:
        return 0

    if dp[n][l][r] != -1:
        return dp[n][l][r]

    dp[n][l][r] = min(solve(n+1, arr[n],r) + move(l, arr[n]), solve(n+1, l, arr[n]) + move(r, arr[n]))
    return dp[n][l][r]

arr = list(map(int, sys.stdin.readline().split()))
dp = [[[-1]*5 for _ in range(5)] for _ in range(100000)]

print(solve(0, 0, 0))
```

Assignment

[백준#2281 데스노트] Silver2

- 문제 : https://www.acmicpc.net/problem/2281

#DP