

Algorithm 3월 4주차

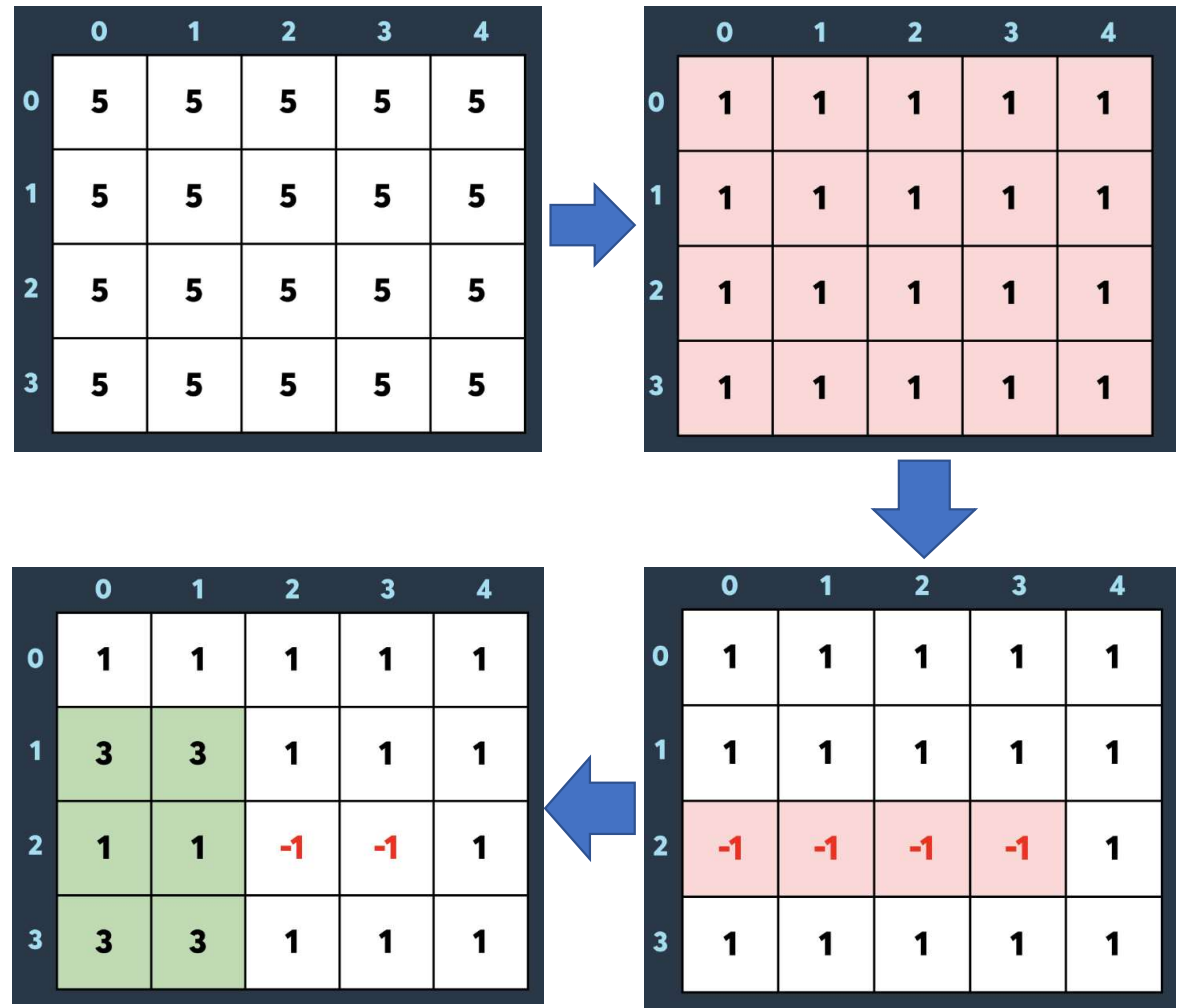
Prefix-Sum

Table of content

- 파괴되지 않은 건물 (Prefix-Sum)

[프로그래머스] 파괴되지 않은 건물

- M x M 크기의 게임 맵
- 각 위치마다 건물 내구도
- 적은 범위 공격, 아군은 범위 회복 스킬
- 입력에는 피아정보, 행 시작점, 열 시작점, 행 끝점, 열 끝점, degree(피해 또는 회복 정도)가 주어짐
- 0이하는 파괴된 건물로 간주됨. 파괴되지 않은 건물 개수를 반환해야 하는 문제



시도 – Brute Force

- 모든 행과 열을 돌아가면서 degree만큼 공격 또는 회복
- 정확성은 통과했으나, 효율성에서 실패함
- Board 크기에서 시간 복잡도 $100만 * skill\ 행의\ 길이\ 25만 = 2500억의\ 시간복잡도$
- 시간복잡도 $O(skill * M * N) = O(n^3)$

- $1 \leq board\ 의\ 행의\ 길이 (= N) \leq 1,000$
- $1 \leq board\ 의\ 열의\ 길이 (= M) \leq 1,000$
- $1 \leq board\ 의\ 원소\ (각\ 건물의\ 내구도) \leq 1,000$
- $1 \leq skill\ 의\ 행의\ 길이 \leq 250,000$

```
def solution(board, skill):
    answer = 0
    for kind, row_s, col_s, row_e, col_e, degree in skill:
        for row in range(row_s, row_e + 1):
            for col in range(col_s, col_e + 1):
                if kind == 1:
                    board[row][col] -= degree
                else:
                    board[row][col] += degree

    for row in range(len(board)):
        for col in range(len(board[0])):
            if board[row][col] > 0:
                answer += 1

    return answer
```

테스트 1 > 통과 (0.01ms, 9.99MB)
테스트 2 > 통과 (0.05ms, 10.3MB)
테스트 3 > 통과 (0.16ms, 10.1MB)
테스트 4 > 통과 (0.56ms, 10.1MB)
테스트 5 > 통과 (1.56ms, 10.2MB)
테스트 6 > 통과 (1.95ms, 10.3MB)
테스트 7 > 통과 (3.03ms, 10.3MB)
테스트 8 > 통과 (4.55ms, 10.2MB)
테스트 9 > 통과 (5.98ms, 10.3MB)
테스트 10 > 통과 (12.39ms, 10.4MB)

테스트 1 > 실패 (시간 초과)
테스트 2 > 실패 (시간 초과)
테스트 3 > 실패 (시간 초과)
테스트 4 > 실패 (시간 초과)
테스트 5 > 실패 (시간 초과)
테스트 6 > 실패 (시간 초과)
테스트 7 > 실패 (시간 초과)

풀이법 - 누적합 (Prefix-sum)

- 시간복잡도를 줄이기 위해서 누적합을 사용해야 한다.
- 누적합이란 특정 배열을 만들어서 첫번째 원소부터 임의의 원소 i 까지의 합 $S[i]$ 를 모든 원소에 대해 구하는 것이다. 그러니까, $S[1], S[2], \dots, S[n]$ 을 구하는 것임

$$\overbrace{a_1 + a_2 + a_3 + \dots + a_{i-1} + a_i + a_{i+1} + \dots + a_j + \dots + a_n}^{S_j} = S_n$$
$$\underbrace{a_1 + a_2 + a_3 + \dots + a_{i-1}}_{S_{i-1}} + \underbrace{a_i + a_{i+1} + \dots + a_j + \dots + a_n}_{S_j - S_{i-1}} = S_n$$

$$S[i] = A[1] + \dots + A[i], S[0] = 0$$

$$S[r] = A[1] + \dots + A[r]$$

$$S[l-1] = A[1] + \dots + A[l-1]$$

$$S[r] - S[l-1] = A[l] + \dots + A[r]$$

- 이 누적합을 이용해서 특정 구간의 합을 구하려면 뺄셈을 한 번만 하면 된다. 특정 구간 $S[r] - S[l-1]$ 을 구하는 데 걸리는 시간복잡도는 $O(1)$

Prefix sum vs Brute force

- 누적합의 시간복잡도는 기존 Brute force에 비해 한 차원 낮다. Brute force를 사용했을 때 시간복잡도가 $O(n^2)$ 라면 누적합을 사용하면 시간복잡도가 $O(n)$ 이 된다.

크기가 N 인 정수 배열 A 가 있고, 여기서 다음과 같은 연산을 최대 M 번 수행해야 하는 문제가 있습니다.

- 구간 l, r ($l \leq r$)이 주어졌을 때, $A[l] + A[l+1] + \dots + A[r-1] + A[r]$ 을 구해서 출력하기

$A[l] + A[l+1] + \dots + A[r-1] + A[r]$ 을 구하기 위해 소스 1과 같이 모두 더하는 방법이 있습니다.

Brute force

```
for j in range(m):
    temp = 0
    for i in range(l, r+1):
        temp += a[i]
    answer.append(temp)
```

이것을 M 번 반복하면 시간복잡도는 $O(M * N) = O(n^2)$

Prefix sum

```
for i in range(1, n + 1):
    s[i] = s[i - 1] + a[i]

for l, r in arr_m:
    answer.append(s[r] - s[l - 1])
```

구간 합을 구하는 데 $O(N)$, 별도의 루프에서 M 번 연산. 시간복잡도는 $O(N + M)$

문제에 누적합 적용하기 - 1차원 배열

- 1차원 배열에 누적합 기록하기 $O(n)$

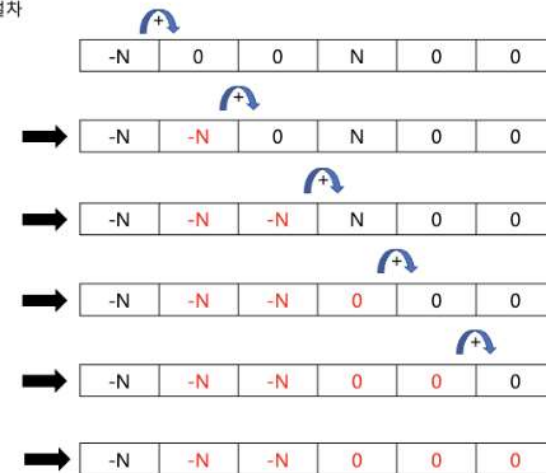
셋팅:

1 [-N, 0, 0, N, 0, 0]

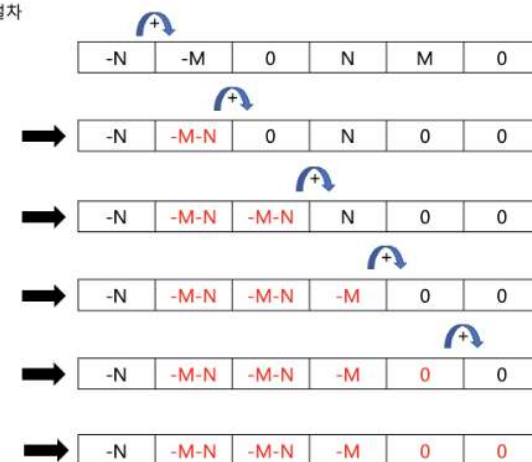
- 1차원 배열 누적합 활용법:
여러 연산을 겹칠 수 있다.

1 [-N, -M-N, -M-N, -M, 0, 0]

누적합 절차



누적합 절차



문제에 누적합 적용하기 - 2차원 배열

- 2차원 배열에 누적합 기록하기

셋팅:

```
1  [[ -N,  0,  0,  N,  0,  0],
2   [  0,  0,  0,  0,  0,  0],
3   [  0,  0,  0,  0,  0,  0],
4   [  N,  0,  0, -N,  0,  0],
5   [  0,  0,  0,  0,  0,  0],
6   [  0,  0,  0,  0,  0,  0]]
```



2. 행 기준 누적합

진행방향 →					
-N	-N	-N	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
N	N	N	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

3. 열 기준 누적합

↓ 진행방향	-N	-N	-N	0	0	0
	-N	-N	-N	0	0	0
	-N	-N	-N	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0

파괴되지 않은 건물 - 누적합 풀이 코드

- 누적합 기록에 $O(\text{skill})$ 소요
- 행 기준 누적합에 $O(MN)$ 소요
- 열 기준 누적합에 $O(MN)$ 소요
- 기존 배열 합하는 데 $O(MN)$ 소요
- 시간복잡도 = $O(\text{skill}) + O(MN) + O(MN) + O(MN) = O(n^2)$

```
def solution(board, skill):
    answer = 0
    prefix = [[0 for j in range(len(board[0]) + 1)] for i in range(len(board) + 1)]

    for kind, row_s, col_s, row_e, col_e, degree in skill:
        prefix[row_s][col_s] += degree if kind == 2 else -degree
        prefix[row_s][col_e + 1] += -degree if kind == 2 else degree
        prefix[row_e + 1][col_s] += -degree if kind == 2 else degree
        prefix[row_e + 1][col_e + 1] += degree if kind == 2 else -degree

    for row in range(len(board)):
        for col in range(len(board[0])):
            prefix[row][col + 1] += prefix[row][col]

    for col in range(len(board[0])):
        for row in range(len(board)):
            prefix[row + 1][col] += prefix[row][col]

    for row in range(len(board)):
        for col in range(len(board[row])):
            board[row][col] += prefix[row][col]
            if board[row][col] > 0:
                answer += 1

    return answer
```