Algorithm 3월 3주차

박시준

Table of content

거스름돈

[프로그래머스] 거스름돈

- 거스름돈 줘야 하는 금액, 가지고 있는 화폐 단위가 주어짐.
- 가지고 있는 화폐로 거스름 돈을 주는 방법의 수를 return하는 문제

Finn은 편의점에서 야간 아르바이트를 하고 있습니다. 야간에 손님이 너무 없어 심심한 Finn은 손님들께 거스름돈을 n 원을 줄 때 방법의 경우의 수를 구하기로 하였습니다.

예를 들어서 손님께 5원을 거슬러 줘야 하고 1원, 2원, 5원이 있다면 다음과 같이 4가지 방법으로 5원을 거슬러 줄 수 있습니다.

- 1원을 5개 사용해서 거슬러 준다.
- 1원을 3개 사용하고, 2원을 1개 사용해서 거슬러 준다.
- 1원을 1개 사용하고, 2원을 2개 사용해서 거슬러 준다.
- 5원을 1개 사용해서 거슬러 준다.

거슬러 줘야 하는 금액 n과 Finn이 현재 보유하고 있는 돈의 종류 money가 매개변수로 주어질 때, Finn이 n 원을 거슬러 줄 방법의 수를 return 하도록 solution 함수를 완성해 주세요.

n	money	result
5	[1,2,5]	4

[프로그래머스] Brute force로 시도 - 실패

- Brute force 방식으로 풀려고 하다가 실패

```python
def solution(n, money):
    answer = 0
    money.sort(reverse=True)
    for m in money:
        start = m
        total = 0
        while total < n:
            total += m

    return answer % 1000000007
```

[프로그래머스] 푸는 방법 - DP

- Dynamic programming으로 푸는 문제였다.
- 코드는 매우 간단하나 그 코드를 발상하는 게 매우 어려웠던 문제

```python
def solution(n, money):
    answer = 0
    record = [1] + [0] * n

    for coin in money:
        for i in range(coin, n + 1):
            if i >= coin:
                record[i] += record[i - coin]

    answer = record[n]

    return answer % 1000000007
```

[프로그래머스] 발상법 - 점화식 생성

- 경우의 수 하나씩 따져봐서 점화식을 만들 수 있는지 확인해본다.
- 거스름돈 1원을 돌려주려면 1원 화폐 하나를 사용하는 경우의 수 하나밖에 없음
- 거스름돈 2원을 돌려주려면 1원 화폐 둘을 사용하는 경우 하나, 2원 화폐 하나만 사용하는 경우 하나 => 총 2가지 경우가 있음
- 거스름돈 3원을 돌려주려면 1원 화폐 세 개 사용하거나, 1원 화폐 한 개 + 2원 화폐 하나 => 총 2가지 경우가 있음
- 거스름돈 4원을 돌려주려면 1원 네 개 사용 또는 1원 두 개 + 2원 한 개 + 2원 두 개 => 총 3가지 경우가 있음

거스름돈	1원	2원	3원	4원
경우의 수	1	2	2	3

[프로그래머스] 점화식 생성

- 이를 좀 더 세부적으로 정리하면 아래와 같다.

거스름돈 금액	1원	2원	3원	4원	5원	6원	7원	8원	9원
1원만 사용하는 경우의 수	1	1	1	1	1	1	1	1	1
2원을 사용하는 경우의 수	0	1	1	2	2	3	3	4	4
5원을 사용하는 경우의 수	0	0	0	0	1	1	2	2	3
합산	**1**	**2**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

[프로그래머스] 점화식 생성

거스름돈 금액	1원	2원	3원	4원	5원	6원	7원	8원	9원
1원만	1	1	1	1	1	1	1	1	1
2원	0	1	1	2	2	3	3	4	4
5원	0	0	0	0	1	1	2	2	3
합산	**1**	**2**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

- 표를 잘 살펴보면 위와 같은 규칙을 발견할 수 있다. 더 낮은 금액의 경우의 수 합산을 이용해 현재 거스름돈 금액의 경우의 수에 더할 수 있음.
- 물론 7원 때부터는 이것은 안 통한다. 합산 표에 5원을 사용하는 경우의 수도 포함되어 있기 때문
- 5원을 사용하는 경우의 수를 제외하면 규칙이 통할 것

[프로그래머스] 1원 2원만 고려했을 때

거스름돈 금액	1원	2원	3원	4원	5원	6원	7원	8원	9원
1원만	1	1	1	1	1	1	1	1	1
2원	0	1	1	2	2	3	3	4	4
합산	**1**	**2**	**2**	**3**	**3**	**4**	**4**	**5**	**5**

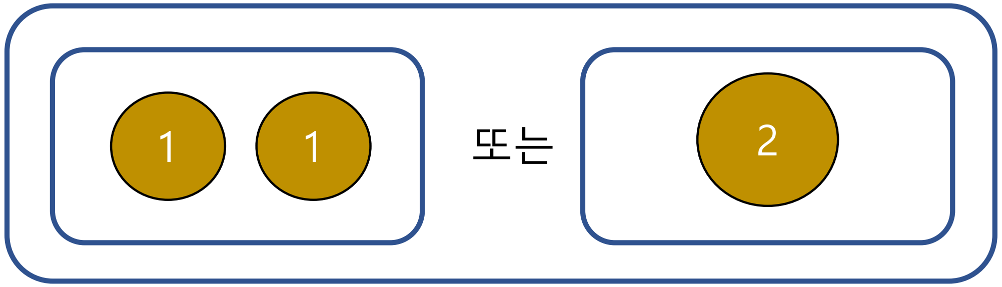
- 5원을 사용하는 경우의 수를 제외하니 규칙이 보인다.
- 이전에 구한 더 낮은 금액의 경우의 수 합산 기록을 현재 거스름돈 금액의 경우의 수에 사용할 수 있음
- 이 규칙을 일반화하여 사용할 수 있을까? 이 논리가 타당하다는 것을 다음 장에서 증명함.

[프로그래머스] 일반화가 가능한 이유

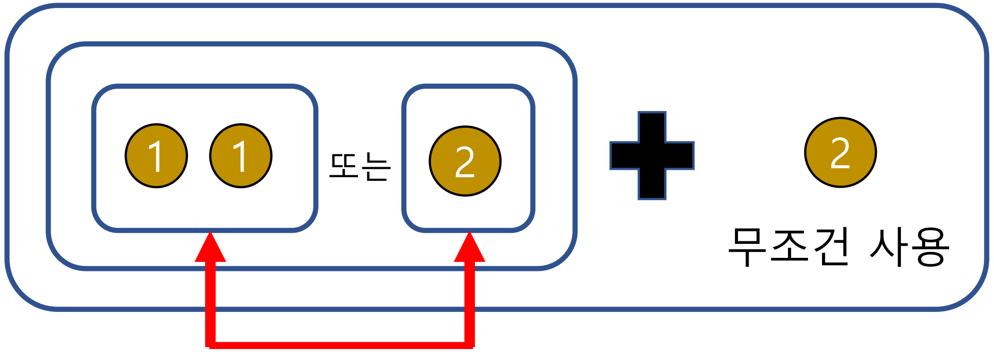
- 거스름돈 4원을 줄 때 2원을 사용하는 경우의 수에서는 2원은 무조건 한 번은 사용해야 한다. 100% 사용하므로 경우의 수에서 2원만큼 빼고 고려해도 된다.
- (2원을 만드는 경우의 수 + 1원만 사용하는 경우의 수 = 4원을 사용하는 경우의 수) 가 나온다.

거스름돈 금액	1원	2원	3원	4원
1원만	1	1	1	1
2원 사용	0	1	1	2
합산	1	2	2	3

2원을 만드는 경우의 수



4원을 만드는 경우의 수



2원을 만드는 경우의 수 2가지를 그대로 계승함

[프로그래머스] 일반화가 가능한 이유

- 그런 논리로 점화식을 세워서 풀면 이와 같은 코드가 나온다.
- 발상이 매우 어려웠던 문제
- 100% 사용한다면 경우의 수에서 그 만큼의 금액을 빼고 고려해도 된다 <- 이것을 떠올리는 것이 중요한 듯 함.
- Record[0]=1도 반드시 빼먹지 말고 해줘야 한다. 거스름돈 0원의 경우의 수는 실질적인 내용은 없지만, 점화식을 돌리기 위해 반드시 필요한 존재.

```python
def solution(n, money):
    answer = 0
    record = [1] + [0] * n

    for coin in money:
        for i in range(coin, n + 1):
            if i >= coin:
                record[i] += record[i - coin]

    answer = record[n]

    return answer % 1000000007
```

연속 부분 수열의 합

[프로그래머스] 연속 부분 수열의 합

- 연속 펄스 수열 다음과 같이 주어짐

[1, -1, 1, -1, 1, -1, ...]

[-1, 1, -1, 1, -1, 1, ...]

- 문제에 주어진 수열에 연속 펄스 수열을 곱해서 최댓값이 나오도록 하고, 그 최댓값을 반환하면 됨
- 예시에서는 [3, -6, 1]에 [1, -1, 1]을 곱해서 나오는 3+6+1 = 10이 최댓값

어떤 수열의 연속 부분 수열에 같은 길이의 펄스 수열을 각 원소끼리 곱하여 연속 펄스 부분 수열을 만들려 합니다. 펄스 수열이란 [1, -1, 1, -1 …] 또는 [-1, 1, -1, 1 …] 과 같이 1 또는 -1로 시작하면서 1과 -1이 번갈아 나오는 수열입니다.

예를 들어 수열 [2, 3, -6, 1, 3, -1, 2, 4]의 연속 부분 수열 [3, -6, 1]에 펄스 수열 [1, -1, 1]을 곱하면 연속 펄스 부분수열은 [3, 6, 1]이 됩니다. 또 다른 예시로 연속 부분 수열 [3, -1, 2, 4]에 펄스 수열 [-1, 1, -1, 1]을 곱하면 연속 펄스 부분수열은 [-3, -1, -2, 4]이 됩니다.

정수 수열 `sequence` 가 매개변수로 주어질 때, 연속 펄스 부분 수열의 합 중 가장 큰 것을 return 하도록 solution 함수를 완성해주세요.

sequence	result
[2, 3, -6, 1, 3, -1, 2, 4]	10

Brute force로 시도 - 시간초과

sequence	result
[2, 3, -6, 1, 3, -1, 2, 4]	10

원소 3개만 사용하는 경우

- [2, 3, -6] * [1, -1, 1] = -7
- [3, -6, 1] * [1, -1, 1] = 10
- [1, 3, -1] * [1, -1, 1] = -3

원소 4개만 사용하는 경우

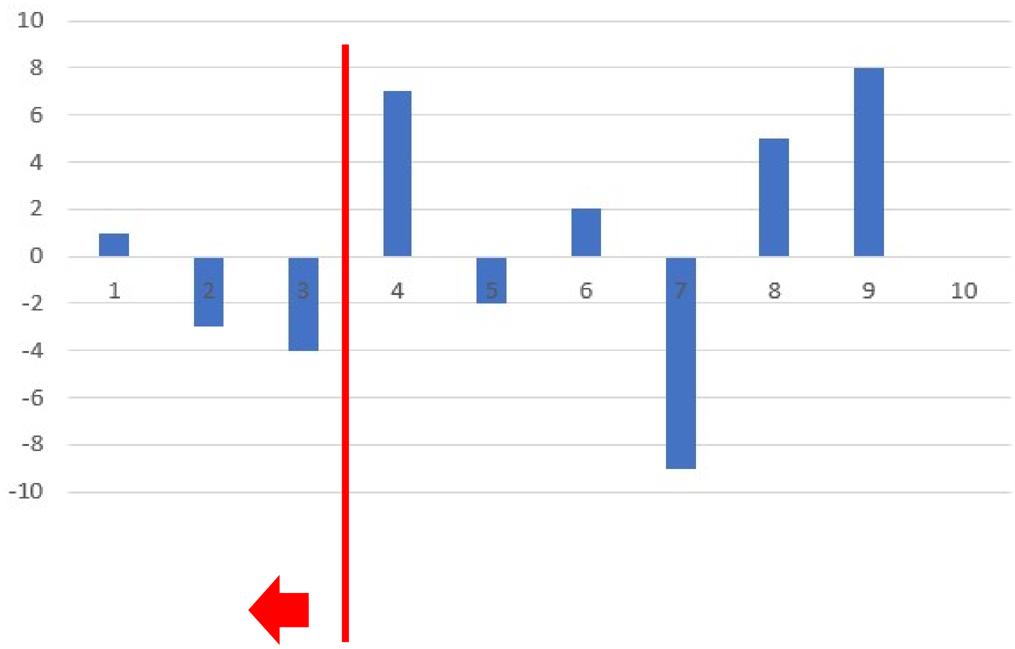
- [2, 3, -6, 1] * [1, -1, 1, -1] = -8
- [3, -6, 1, 3] * [1, -1, 1, -1] = 7

…

- Brute force로 모든 경우의 수를 구할 수도 있겠지만, 이렇게 구할 경우 O(n^2) 시간이 걸린다. 너무 많은 경우를 모두 연산해야 함.

첫번째 방법 - Dynamic programming

- 일일이 모든 경우의 수를 구하지 말고 합산했을 때 값이 커지는 경우만 기록하면 된다.
- 기록된 dp가 양수면 더해지는 값이 기록될 것이고, dp가 음수면 현재의 값만 기록이 될 것이다.
- 기록된 dp가 작은 값이라도 양수면 계속 더하면 된다. 더하는 횟수에는 제한 없기 때문.



이전까지 쌓아온 값이 음수면 그냥 버린다.

첫번째 방법 - Dynamic programming

- 우선 첫번째로 주어진 sequence 배열에 pulse 배열 [1, -1, 1, ...]을 곱한다.
- 반복문 돌려서 dp에 max값을 기록한다. 전체 dp에서 max값을 구한다
- 다시 두번째로 sequence 에 pulse [-1, 1, -1, ...] 곱한다.
- 반복문 돌려서 dp에 max값 기록한다. 전체 dp에서 max값을 구한다.
- 이렇게 하면 공간을 좀 사용하지만, 시간 복잡도는 O(n)으로 풀 수 있다.

```python
def solution(sequence):
    n = len(sequence)
    dp = [0] * n

    # first pulse
    for i in range(1, n, 2):
        sequence[i] *= -1
    dp[0] = sequence[0]
    for i in range(1, n):
        dp[i] = max(sequence[i], sequence[i] + dp[i - 1])
    first = max(dp)

    # second pulse
    for i in range(n):
        sequence[i] *= -1
    dp[0] = sequence[0]
    for i in range(1, n):
        dp[i] = max(sequence[i], sequence[i] + dp[i - 1])
    second = max(dp)

    return max(first, second)
```

두번째 방법 – 수학을 사용하여

- DP를 사용하지 않고 푸는 방법이 있음.
- 메모이제이션을 하지 않으므로 공간을 더 절약할 수 있다.
- 시간도 DP보다 더 빠르다.

```python
def solution(sequence):
    answer = -inf
    n = len(sequence)
    sum1 = sum2 = 0
    min_sum1 = min_sum2 = 0
    pulse = 1

    for i in range(n):
        sum1 += sequence[i] * pulse
        sum2 += sequence[i] * (-pulse)

        answer = max(answer, sum1-min_sum1, sum2-min_sum2)

        min_sum1 = min(sum1, min_sum1)
        min_sum2 = min(sum2, min_sum2)
        pulse *= -1

    return answer
```

```
테스트 11 〉 통과 (10.39ms, 10.4MB)
테스트 12 〉 통과 (85.79ms, 13.9MB)
테스트 13 〉 통과 (68.52ms, 13.8MB)
테스트 14 〉 통과 (98.63ms, 14.2MB)
테스트 15 〉 통과 (89.73ms, 14.2MB)
테스트 16 〉 통과 (68.74ms, 14.1MB)
테스트 17 〉 통과 (324.39ms, 31.3MB)
테스트 18 〉 통과 (319.25ms, 31.1MB)
테스트 19 〉 통과 (322.67ms, 31.8MB)
테스트 20 〉 통과 (345.06ms, 31.6MB)
```

연속 펄스 부분 수열의 최대합 =

E까지의 누적합 – E 이전까지의 누적합의 최소값