Greedy & DP

BABO 4기 이론반

23기 분석 박지원

CONTENTS

1. 3주차 과제 코드리뷰

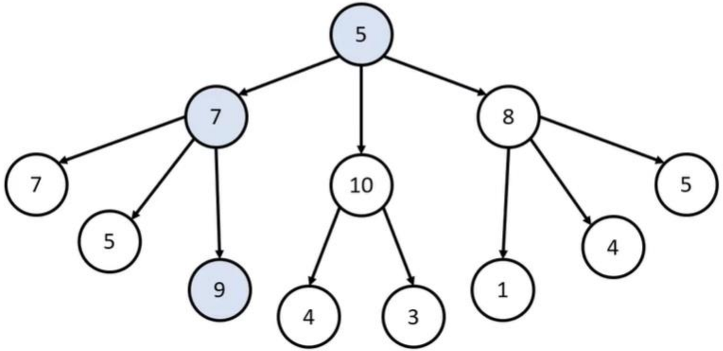
2. Greedy

그리디 (Greedy)

- **현재 상황에서 지금 당장 좋은 것만 고르는 방법**
- 단순히 가장 좋아 보이는 것을 반복적으로 선택해도 최적의 해를 구할 수 있는지 검토 필수
  - 현재의 최적해 != 전체의 최적해

Q. 루트 노드부터 시작해서 거쳐가는
노드 값의 합이 최대인 경로

A. 전체의 최적해 [5 → 7 → 9]

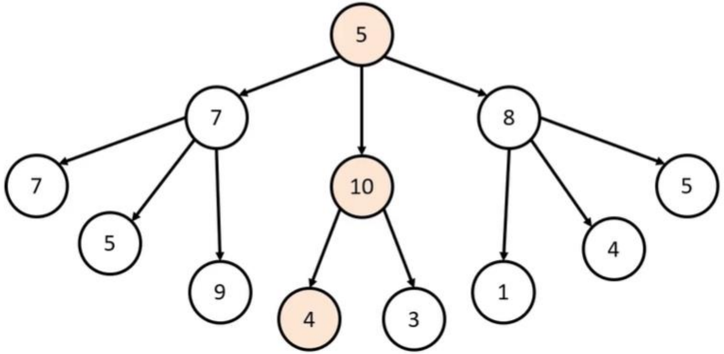


그리디 (Greedy)

- **현재 상황에서 지금 당장 좋은 것만 고르는 방법**
- 단순히 가장 좋아 보이는 것을 반복적으로 선택해도 최적의 해를 구할 수 있는지 검토 필수
    - 현재의 최적해 != 전체의 최적해

Q. 루트 노드부터 시작해서 거쳐가는 노드 값의 합이 최대인 경로

A. 그리디의 최적해 [5 → 10 → 4]



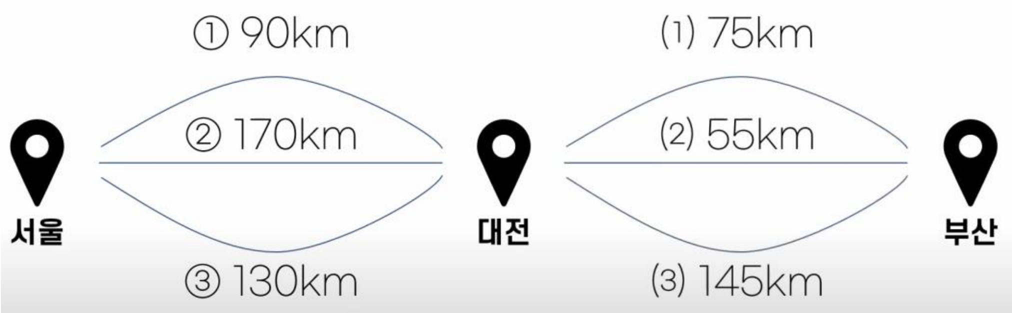
그리디 (Greedy)의 정당성

- **탐욕적 선택 속성 (greedy choice property)**
  - 이전의 선택이 이후 선택에 영향을 주지 않음

- **최적 부분 구조 (optimal substructure)**
  - 부분 문제들에 대한 최적해를 통해 전체 문제의 최적해(global optimum)을 구성할 수 있음

그리디 (Greedy)의 정당성

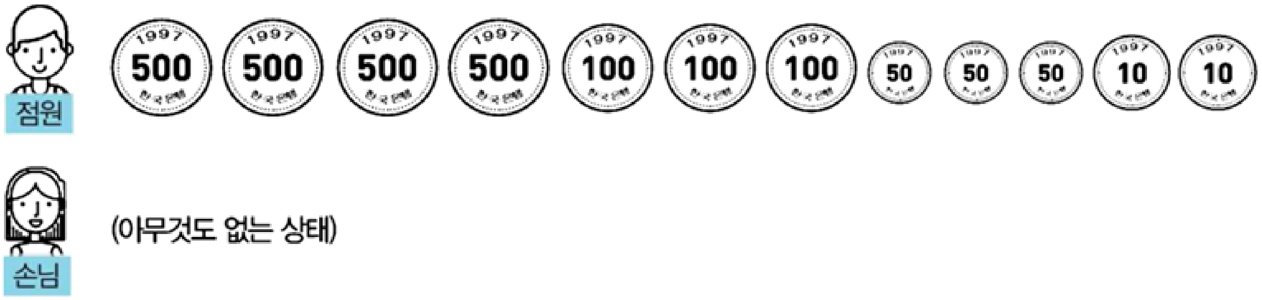
- 서울에서 부산까지 가는 최단 루트



  - (서울 – 대전) 경로가 (대전 – 부산) 경로에 영향을 미치지 않음 → 탐욕적 선택 속성
  - min(서울 – 부산) = min(서울 – 대전) + min(대전 – 부산) → 최적 부분 구조

거스름돈

- 500원, 100원, 50원, 10원짜리 동전이 무한히 존재
- 손님에게 거슬러 줘야 할 돈이 N원일 때 거슬러줘야 할 동전의 최소 개수 (N은 항상 10의 배수)
- N = 1260



거스름돈

- 아이디어 –가장 큰 화폐 단위부터

화폐 단위	500	100	50	10
손님이 받은 개수	2	2	1	1

```
N = 1260
count = 0
# 큰 단위의 화폐부터 차례대로 확인
coin_types = [500, 100, 50, 10]
for coin in coin_types:
    count += N // coin
# 해당 화폐로 거슬러 줄 수 있는 동전의 개수
    N %= coin
print(count)
```

거스름돈

- 만약, 400원짜리 동전이 추가 된다면? [500, 400, 100, 50, 10]
  - N = 830
  - 그리디 : 500 * 1 + 100 * 3 + 10 * 3 → 7
  - 최적해 : 400 * 2 + 10 * 3 → 5

- 모든 동전의 종류가 각 동전의 배수일 때 → 그리디
  - 큰 단위가 항상 작은 단위의 배수이므로 작은 단위의 동전들을 종합해 다른 해가 나올 수 없음
  - Ex. N 〉 500보다 큰 경우에는 무조건 500원짜리 동전을 사용하는 것이 좋다
  - 화폐가 서로 배수 형태가 아닌 경우는 다른 알고리즘 사용

3. Dynamic Programming

다이나믹 프로그래밍 (Dynamic Programming)

- **복잡한 문제를 작은 문제로 나누어 해결하는 방법**
- 메모이제이션(Memoization)
    - 1번 구현한 결과를 메모리 공간에 저장해두고, 다시 호출하면 그 결과를 그대로 가져오는 것

- **조건**
    - **중복되는 부분 문제 (Overlapping Subproblem) :** 부분 문제가 여러 번 반복되어 등장 → 시간 절약 가능
    - **최적 부분 구조 (optimal substructure) :** 전체 문제에 대한 최적해(global optimum)가 부분 문제들의 최적해로부터 구성됨

다이나믹 프로그래밍 구현 방법

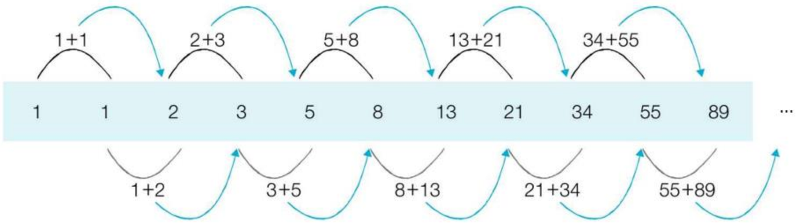
- Top-Down 방식
  - 전체 문제를 작은 부분 문제로 나누고, 이를 해결하기 위해 재귀적으로 동작
  - 중복 부분 문제 해결 위해 메모이제이션 사용
  - 재귀 함수 이용해 구현

    → 가독성은 높아지지만 많은 stack 메모리 사용
- Bottom-Up 방식
  - 작은 부분 문제들을 먼저 해결하고, 이를 이용하여 전체 최적해 구함
  - 반복문을 이용해 구현

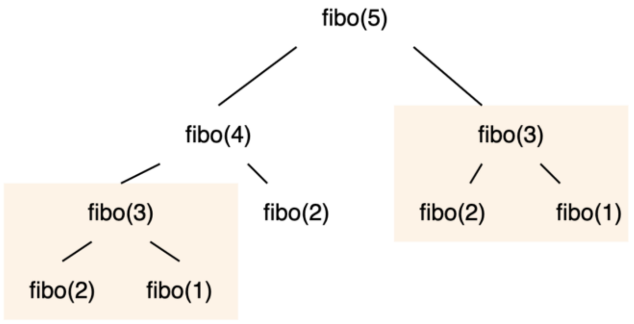
    → 가독성은 낮아질 수 있지만 stack을 사용하지 않아 메모리 절약 효과

피보나치 수열

- 이전 두 항의 합을 다음 항으로 정의하는 수열

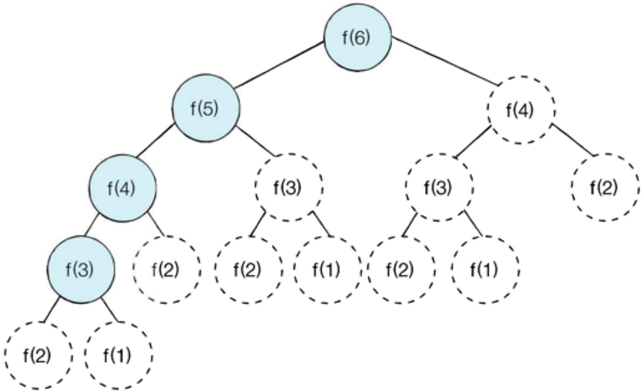



```
fib(0) = 0
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2), (n >= 2)
```



피보나치 수열 _ Top-down

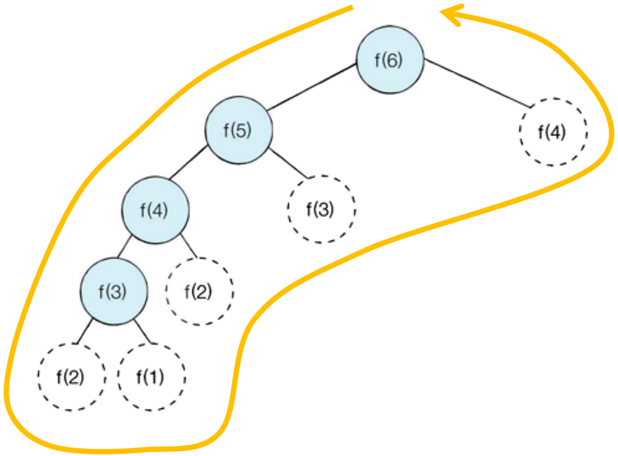
- f(6)을 해결하기위해 f(5), f(4)… 호출




```
d = [0] * 100 # 리스트 초기화
def fibo_topdown(x):
    if x == 1 or x == 2:
        return 1
    #이미 계산한 적 있는 문제라면 그대로 반환
    if d[x] != 0:
        return d[x]
    d[x] = fibo_topdown(x - 1) + fibo_topdown(x - 2)
    return d[x]
print(fibo_topdown(6))
```

피보나치 수열 _ Top-down

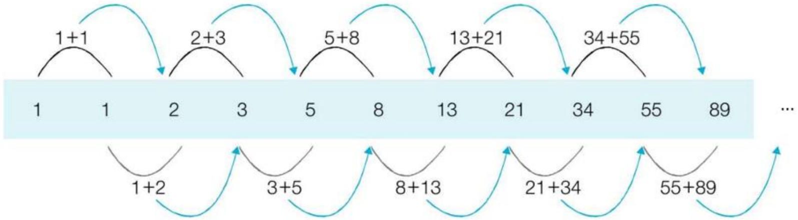
- f(6), f(5), f(4) , f(3) , f(2) , f(1) , f(2) , f(3) , f(4)




```
d = [0] * 100 # 리스트 초기화
def fibo_topdown(x):
    if x == 1 or x == 2:
        return 1
    #이미 계산한 적 있는 문제라면 그대로 반환
    if d[x] != 0:
        return d[x]
    d[x] = fibo_topdown(x - 1) + fibo_topdown(x - 2)
    return d[x]
print(fibo_topdown(6))
```

피보나치 수열 _ Bottom-Up

- f(6)을 해결하기위해 f(1), f(2) , f(3) ⋯ 계산




```python
d = [0] * 100 # 리스트 초기화

d[1] =1
d[2] = 1
n = 6

for i in range(3, n + 1):
    d[i] = d[i - 1] + d[i - 2]
    print(d[n])
```

과제

– 백준 1931 – 회의실 배정 (S1)

– 백준 11501 – 주식 (S2)

– 백준 1912 – 연속합 (S2)

번외

– 백준 11000 – 강의실 배정(G5)

– 프로그래머스 42884– 단속카메라 (lv3)