파이썬으로 배우는 알고리즘 기초

Chap 5. 되추적(백트래킹)

4.5

0-1 배낭 문제와 동적 계획법

4.5 0-1 배낭 문제와 동적 계획법

- 0-1 배낭 문제 (0-1 Knapsack Problem)
  - n개의 아이템 집합: $S = \{item_1, item_2, \dots, item_n\}$
  - 아이템의 무게: $w = [w_1, w_2, \dots, w_n]$
  - 아이템의 가치: $p = [p_1, p_2, \dots, p_n]$
  - 배낭의 용량: W
  - 배낭의 용량을 넘지 않으면서 가치가 최대가 되는 S의 부분집합 A 찾기.
    - 배낭의 용량 조건: $\sum_{item_i \in A} w_i \leq W$
    - 가치 평가 함수: $\sum_{item_i \in A} p_i$

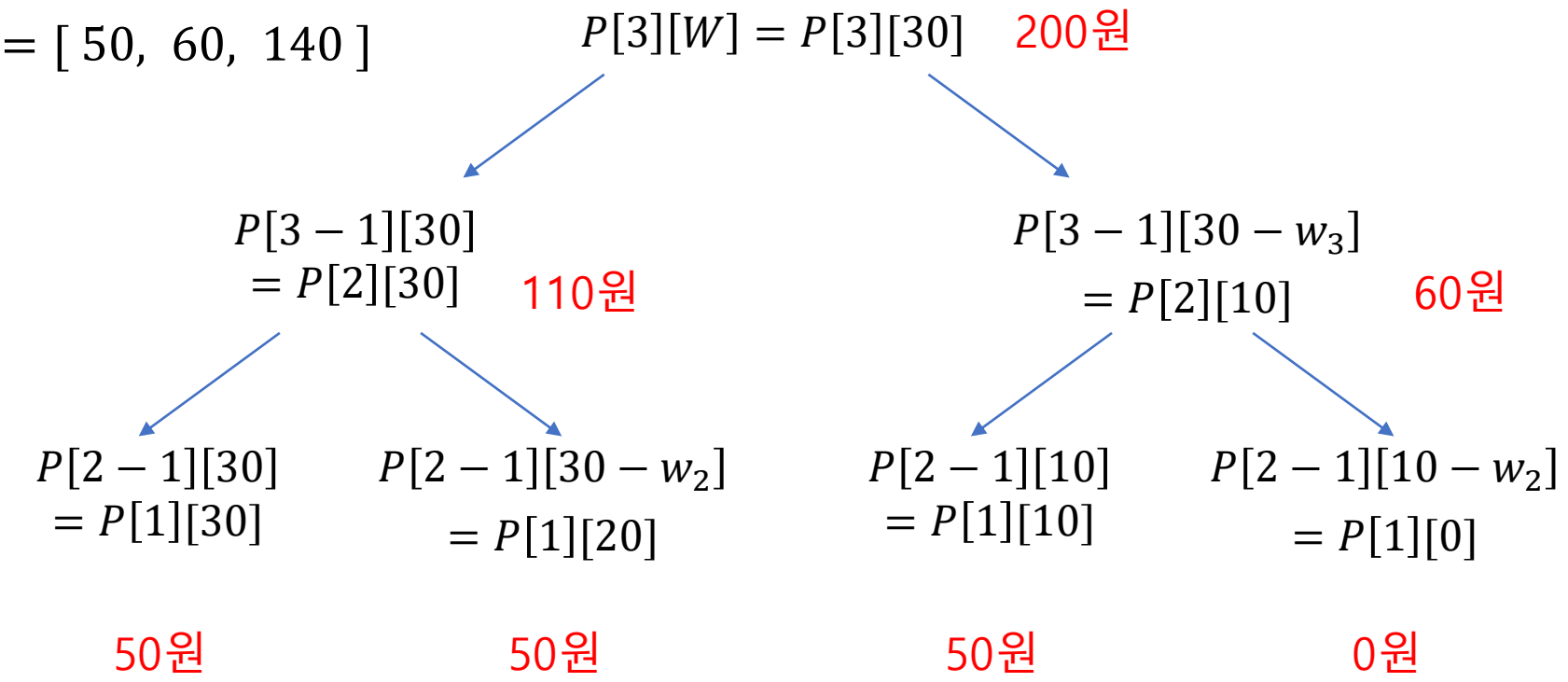
4.5 0-1 배낭 문제와 동적 계획법

- 0-1 배낭 문제: 동적 계획법(Dynamic Programming)
  - $P[i][w]$: 총 무게가 w를 초과할 수 없다는 제약조건 하에서 처음 i개 아이템에서만 선택할 때 얻는 최적의 이익
    - $P[n][w]$: n개의 아이템으로 얻을 수 있는 최대 이익
  - 재귀 관계식 구하기
    - $$P[i][w] = \begin{cases} \max(P[i-1][w],\ p_i + P[i-1][w-w_i]), & \text{if } w_i \le w \\ P[i-1][w], & \text{if } w_i > w \end{cases}$$

- 0-1 배낭 문제와 동적 계획법
  - P 배열의 크기는 nW: 시간 복잡도가 $\Theta(nW)$
    - 만약 W가 매우 큰 값이면? (예: $W = n!$)
    - 단순무식법으로 푸는 방법은 $\Theta(2^n)$
  - 효율적으로 동적 계획을 사용하려면?
    - $P[n][W]$를 계산하려면 $P[n-1][W], P[n-1][W-w_n]$만 필요
    - $P[i][w]$를 계산하려면 $P[i-1][w], P[i-1][w-w_i]$만 필요

4.5 0-1 배낭 문제와 동적 계획법

- $n = 3,\ W = 30$
- $w = [\ 5,\ 10,\ 20\ ]$
- $p = [\ 50,\ 60,\ 140\ ]$



$$P[1][w] = \begin{cases} \max(P[0][w], & 50 + P[0][w-5]), \quad \text{if } 5 \le w \\ & P[0][w], \quad \text{if } 5 > w \end{cases}$$

Algorithm 4.7: Dynamic Programming for the 0-1 Knapsack Problem

```
def knapsack2(i, W, w, p):
    if (i <= 0 or W <= 0):
        return 0
    if (w[i] > W):
        return knapsack2(i - 1, W, w, p)
    else: # w[i] <= W
        left = knapsack2(i - 1, W, w, p)
        right = knapsack2(i - 1, W - w[i], w, p)
        return max(left, p[i] + right)
```

4.5 0-1 배낭 문제와 동적 계획법


```
W = 30
w = [0, 5, 10, 20]
p = [0, 50, 60, 140]
profit = knapsack2(len(w)-1, W, w, p)
print(profit)
```

주니온TV@Youtube

자세히 보면 유익한 코딩 채널

https://bit.ly/2JXXGqz

- 여러분의 **구독**과 **좋아요**는 강의제작에 큰 힘이 됩니다.
- 강의자료 및 소스코드: **구글 드라이브**에서 다운로드
  (다운로드 주소는 영상 하단 설명란 참고)

  https://bit.ly/3fN0q8t