파이썬으로 배우는 알고리즘 기초

Chap 5. 되추적(백트래킹)

5.7

0-1 배낭 문제와 백트래킹

5.7 0-1 배낭 문제와 백트래킹

- 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: $\sum_{item_i \in A} p_i$

5.7 0-1 배낭 문제와 백트래킹

- 0-1 배낭 문제: 백트래킹(Backtracking)
  - 최적화 문제 (optimization problem): 전체 탐색이 필요함
  - 상태공간트리의 구성: 부분집합의 합 문제와 동일함
    - 최적해를 찾는 것이 목표: 공집합을 최적해 집합으로 초기화
    - 어떤 아이템을 최적해 집합에 포함시켜서 전체 이익을 계산
    - 이 때의 전체 이익이 최적해보다 이익이 많으면 이 집합이 최적해 집합

5.7 0-1 배낭 문제와 백트래킹

- 0-1 배낭 문제에 대한 백트래킹 알고리즘

```
void checknode (node v)
{
  node u;
  if (value(v) is better than best)
     best = value(v);
  if (promising(v))
     for (each child u of v)
        checknode(u);
}
```

5.7 0-1 배낭 문제와 백트래킹

- 가지치기와 유망 함수
  - 배낭에 아이템을 넣을 공간이 남아 있지 않으면 유망하지 않음
    - $weight >= W$.

5.7 0-1 배낭 문제와 백트래킹

- 가지치기와 유망 함수
  - 현재까지 찾은 최적해의 이익이 현재 노드에서 앞으로 얻을 수 있는 최대 이익보다 더 크면 유망하지 않음
    - $bound <= maxprofit$.
    - $maxprofit$: 현재까지 찾은 최적해의 이익 값
    - $bound$: 현재 노드에서 앞으로 얻을 수 있는 최대 이익

$$\boldsymbol{bound} = \left(profit + \sum_{j=j+1}^{k-1} p_j \right) + (W - totweight) \times \frac{p_k}{w_k}$$

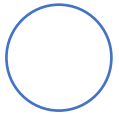
$$\boldsymbol{totweight} = weight + \sum_{j=i+1}^{k-1} w_j$$

$\boldsymbol{profit}$: the *sum* of the *profits* of the items included

5.7 0-1 배낭 문제와 백트래킹

- $n = 4, W = 16$
- $p_i = [40, 30, 50, 10]$
- $w_i = [\ 2,\ \ 5, 10,\ \ 5]$
- $\frac{p_i}{w_i} = [20,\ \ 6,\ \ 5,\ \ 2]$

(0, 0) (*level, position*)



(0, 0, 115)

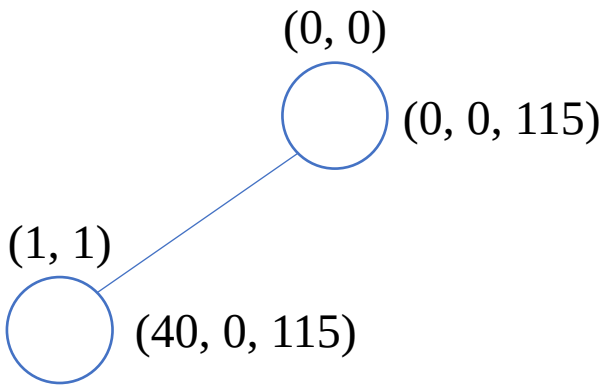
(*profit, weight, bound*)

1. Set $maxprofit$ to 0.

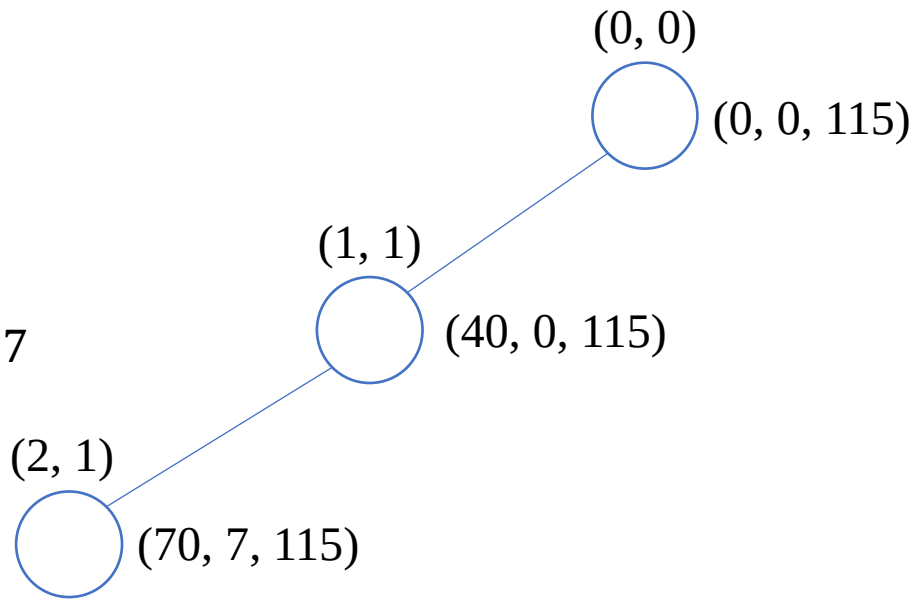
2. Visit node (0, 0) (the root)
   - Compute its profit and weight.
     - $profit = 0, weight = 0$
   - Compute its bound.
     - $totweight = 0 + 2 + 5 = 7$
     - $bound = 0 + 40 + 30 + (16 - 7)) \times \frac{50}{10} = 115$
   - Promising? yes
     - $weight < W\ (0 <\ 16)$ &&
     - $bound > maxprofit\ (115 >\ 0)$

5.7 0-1 배낭 문제와 백트래킹

3. Visit node (1, 1).
   - Compute its profit and weight.
     - $profit = 0 + 40 = 40$
     - $weight = 0 + 2 = 2$
   - Set maxprofit to 40.
     - $weight > W\ (2 > 16)$
     - $profit > maxprofit\ (40 > 0)$
   - Compute its bound.
     - $totweight = 2 + 5 = 7$
     - $bound = 40 + 30 + (16 - 7)) \times \frac{50}{10} = 115$
   - Promising? yes
     - $weight < W\ (2 < 16)$ &&
     - $bound > maxprofit\ (115 > 40)$



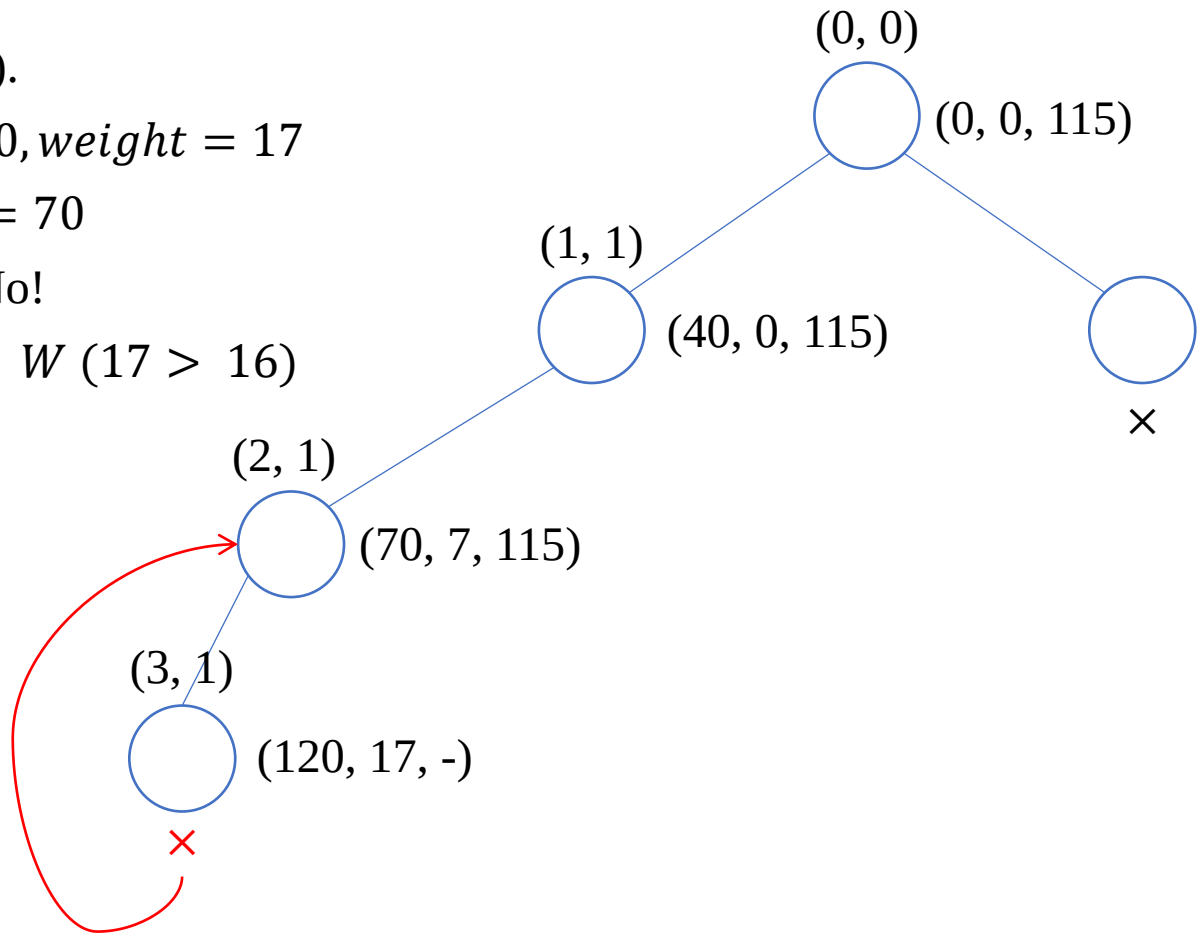
5.7 0-1 배낭 문제와 백트래킹



4. Visit node (2, 1).
   - $profit = 70, weight = 7$
   - $maxprofit = 70$
   - $totweight = 7$
   - $bound = 115$
   - Promising? yes
     - $weight < W\ (7 < 16)$ &&
     - $bound > maxprofit\ (115 > 70)$

5.7 0-1 배낭 문제와 백트래킹

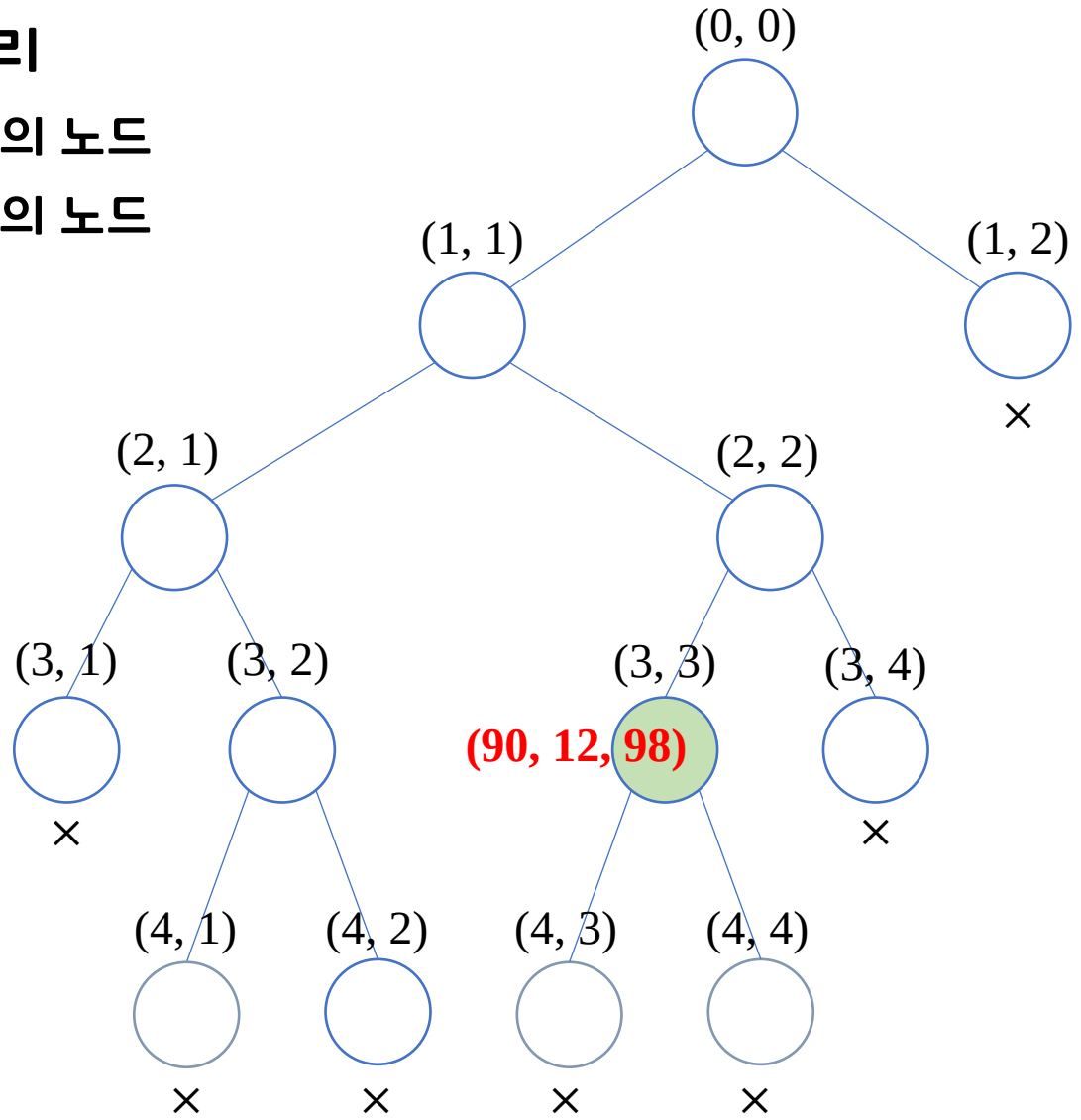
5. Visit node (3, 1).
   - $profit = 120, weight = 17$
   - $maxprofit = 70$
   - Promising? No!
     - $weight > W$ $(17 > 16)$



5. Backtrack to node (2, 1)

5.7 0-1 배낭 문제와 백트래킹

- 최종 상태공간트리
  - 가지치기 전: 31개의 노드
  - 가지치기 후: 13개의 노드



Algorithm 5.7: Backtracking Algorithm for the 0-1 Knapsack Problem

```
def knapsack3 (i, profit, weight):
    global maxprofit, numbest, bestset
    if (weight <= W and profit > maxprofit):
        maxprofit = profit
        numbest = i
        bestset = include[:]
    if (promising(i, profit, weight)):
        include[i + 1] = True
        knapsack3(i + 1, profit + p[i+1], weight + w[i+1])
        include[i + 1] = False
        knapsack3(i + 1, profit, weight)
```

Algorithm 5.7: Backtracking Algorithm for the 0-1 Knapsack Problem

```
def promising (i, profit, weight):
    if (weight > W):
        return False
    else:
        j = i + 1
        bound = profit
        totweight = weight
        while (j <= n and totweight + w[j] <= W):
            totweight += w[j]
            bound += p[j]
            j += 1
        k = j
        if (k <= n):
            bound += (W - totweight) * p[k] / w[k]
        return bound > maxprofit
```

$$\boldsymbol{bound} = \left(profit + \sum_{j=j+1}^{k-1} p_j \right) + (W - totweight) \times \frac{p_k}{w_k}$$

$$\boldsymbol{totweight} = weight + \sum_{j=i+1}^{k-1} w_j$$

5.7 0-1 배낭 문제와 백트래킹


```
n = 4
W = 16
w = [0, 2, 5, 10, 5]
p = [0, 40, 30, 50, 10]
maxprofit = 0
numbest = 0
bestset = []
include = [False] * (n + 1)

knapsack3(0, 0, 0)
print(bestset[1:len(bestset)])
```

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