파이썬으로 배우는 알고리즘 기초

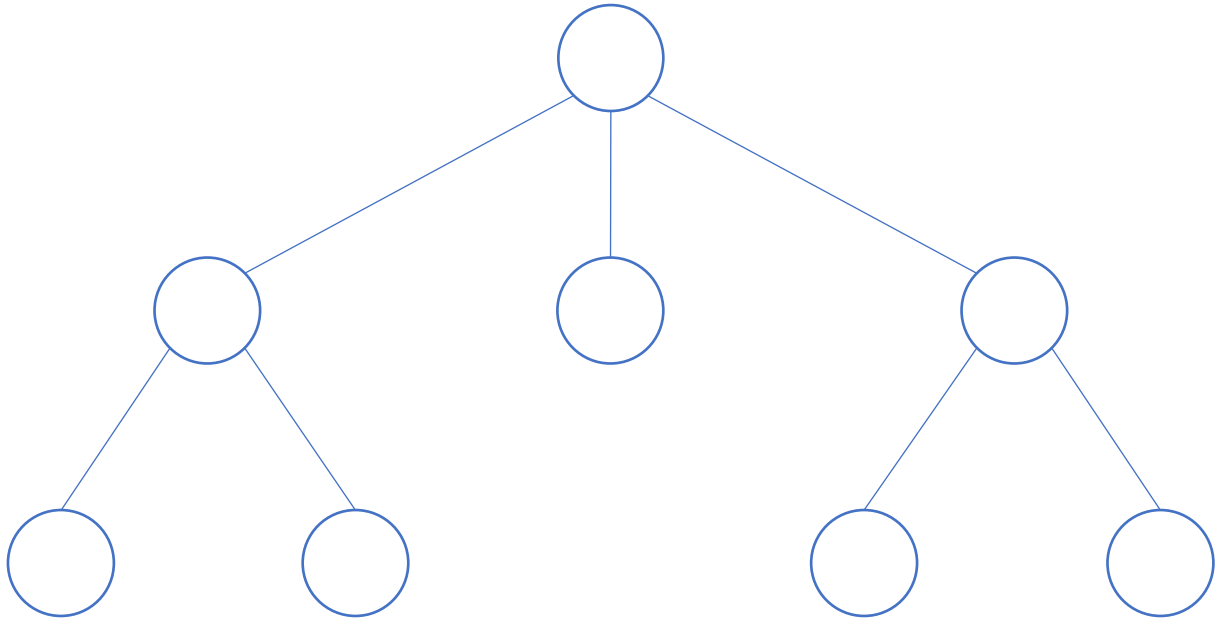
Chap 6. 분기 한정법

6.1

분기 한정법과 0-1 배낭 문제

6.1 분기 한정법과 0-1 배낭 문제

- DFS와 BFS
  - **깊이우선탐색**: DFS (Depth-First-Search)
  - **너비우선탐색**: BFS (Breadth-First-Search)



6.1 분기 한정법과 0-1 배낭 문제

- **분기 한정** (The Branch-and-Bound)
  - 백트래킹과 동일하게 상태공간트리로 문제를 해결
  - DFS를 기반으로 하는 백트래킹과 달리, BFS를 기반으로 함
  - 최적화 문제에서 유리한 점이 있음: 항상 한계값(bound)를 고려
  - 분기한정 가지치기 최적우선탐색
    - Best-First-Search with Branch-and-Bound Pruning

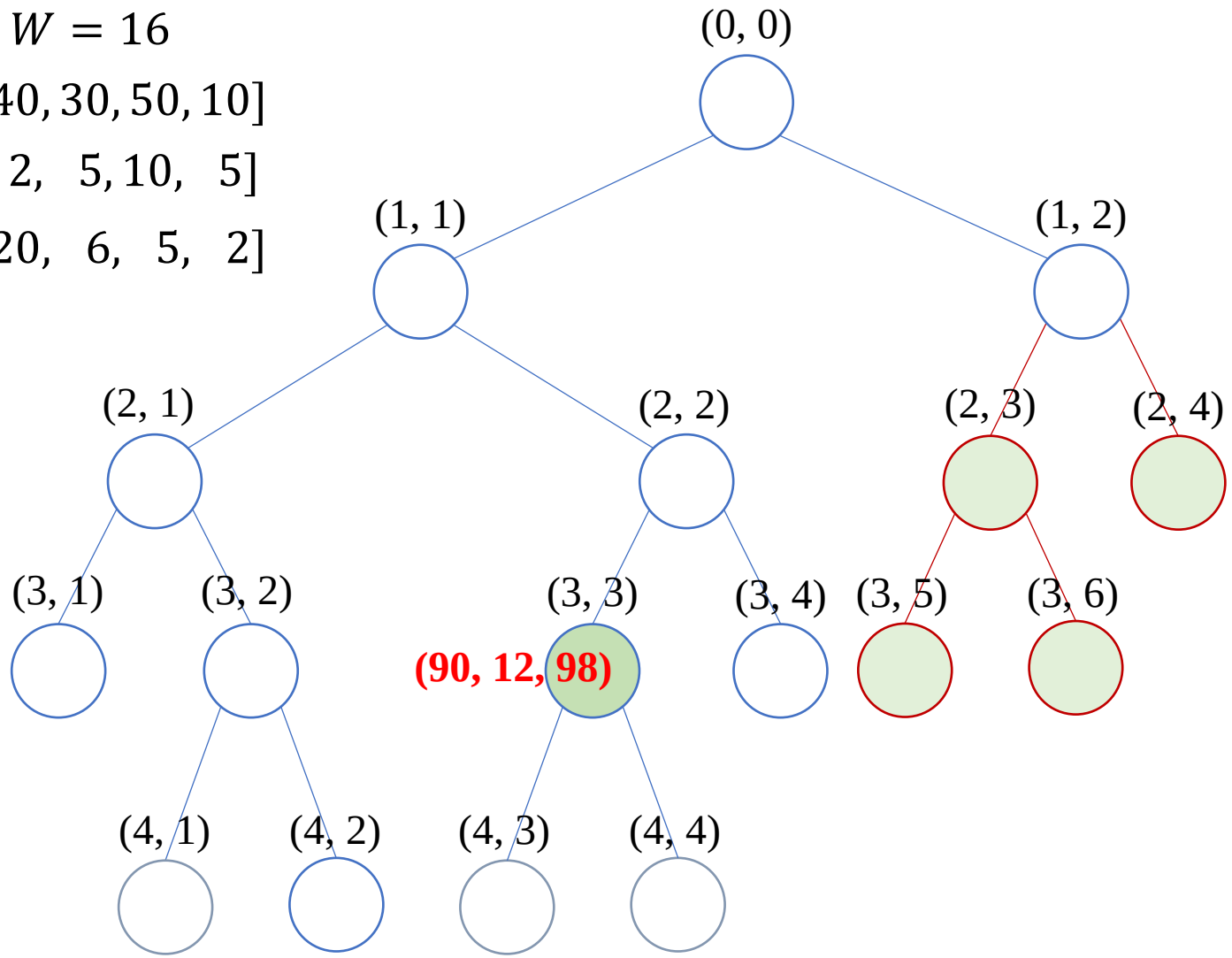
6.1 분기 한정법과 0-1 배낭 문제

- 0-1 배낭 문제: 너비우선탐색으로 풀기

```
def breadth_first_search (tree T):
  queue_of_node Q;
  node u, v;
  initialize(Q);
  v = root of T;
  visit v;
  enqueue(Q, v);
  while (! empty(Q)):
    dequeue(Q, v);
    for (each child u of v):
      visit u;
      enqueue(Q, u);
```

6.1 분기 한정법과 0-1 배낭 문제

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

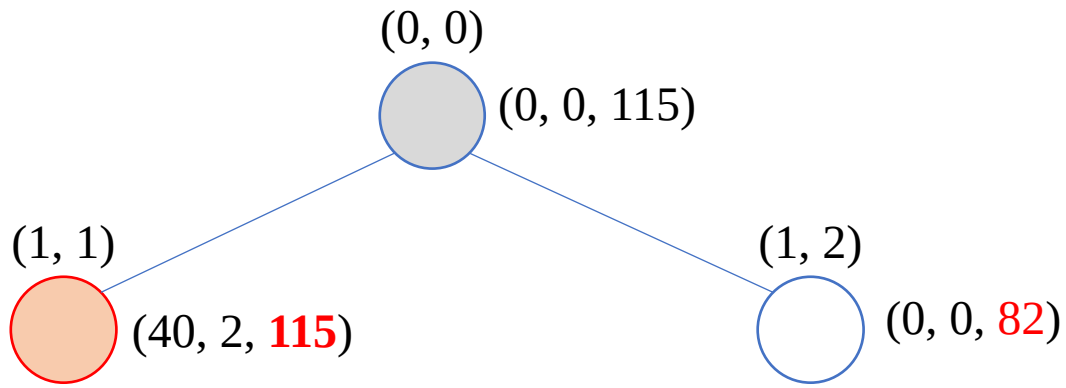


6.1 분기 한정법과 0-1 배낭 문제

- 0-1 배낭 문제: **분기 한정법**으로 풀기
  - 일반적인 방법으로는 BFS가 DFS보다 좋은 점이 없음
  - 유망 함수 외에 추가적인 용도로 **한계값(bound)을 사용**해야 함
  - 주어진 어떤 노드의 모든 자식 노드를 방문한 후
    - 유망하면서 확장하지 않은 노드들을 모두 살펴보고
    - 그 중에서 한계값이 가장 좋은 노들을 우선적으로 확장한다.
  - 지금까지 찾은 최적의 해답보다 그 한계값이 더 좋다면 그 노드는 유망함

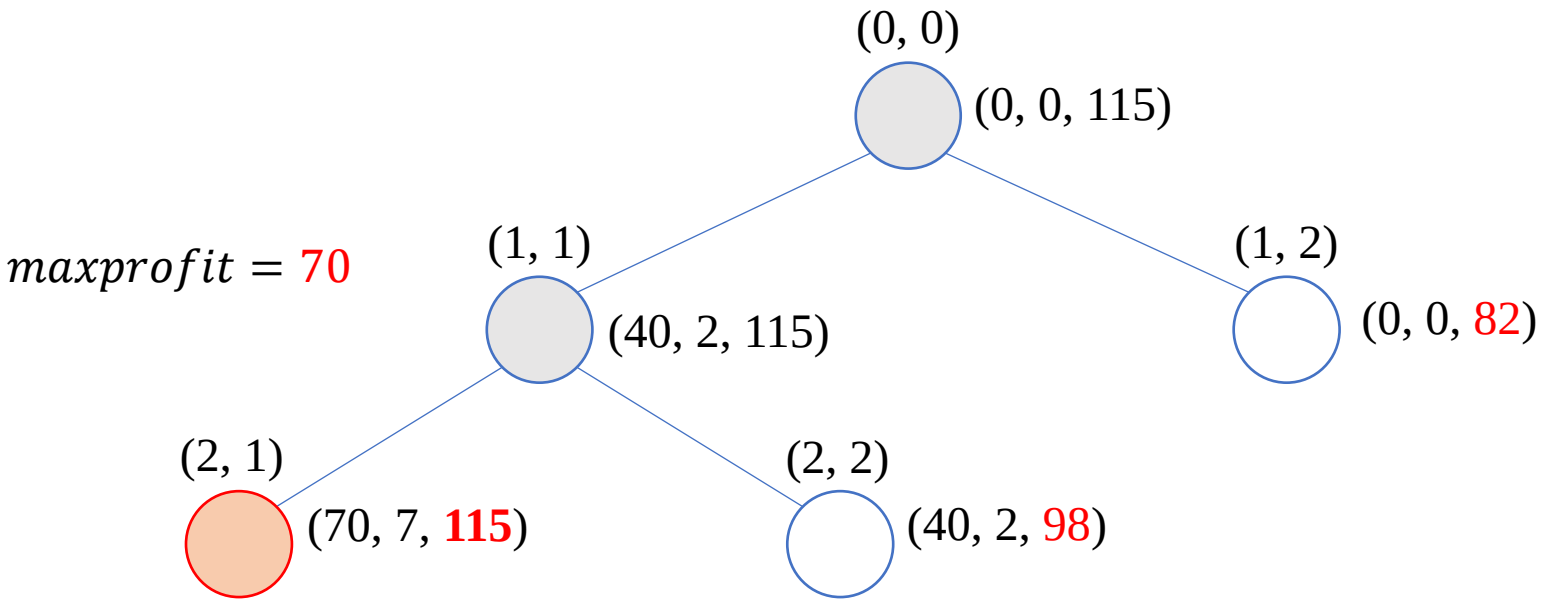
6.1 분기 한정법과 0-1 배낭 문제

$maxprofit = 40$



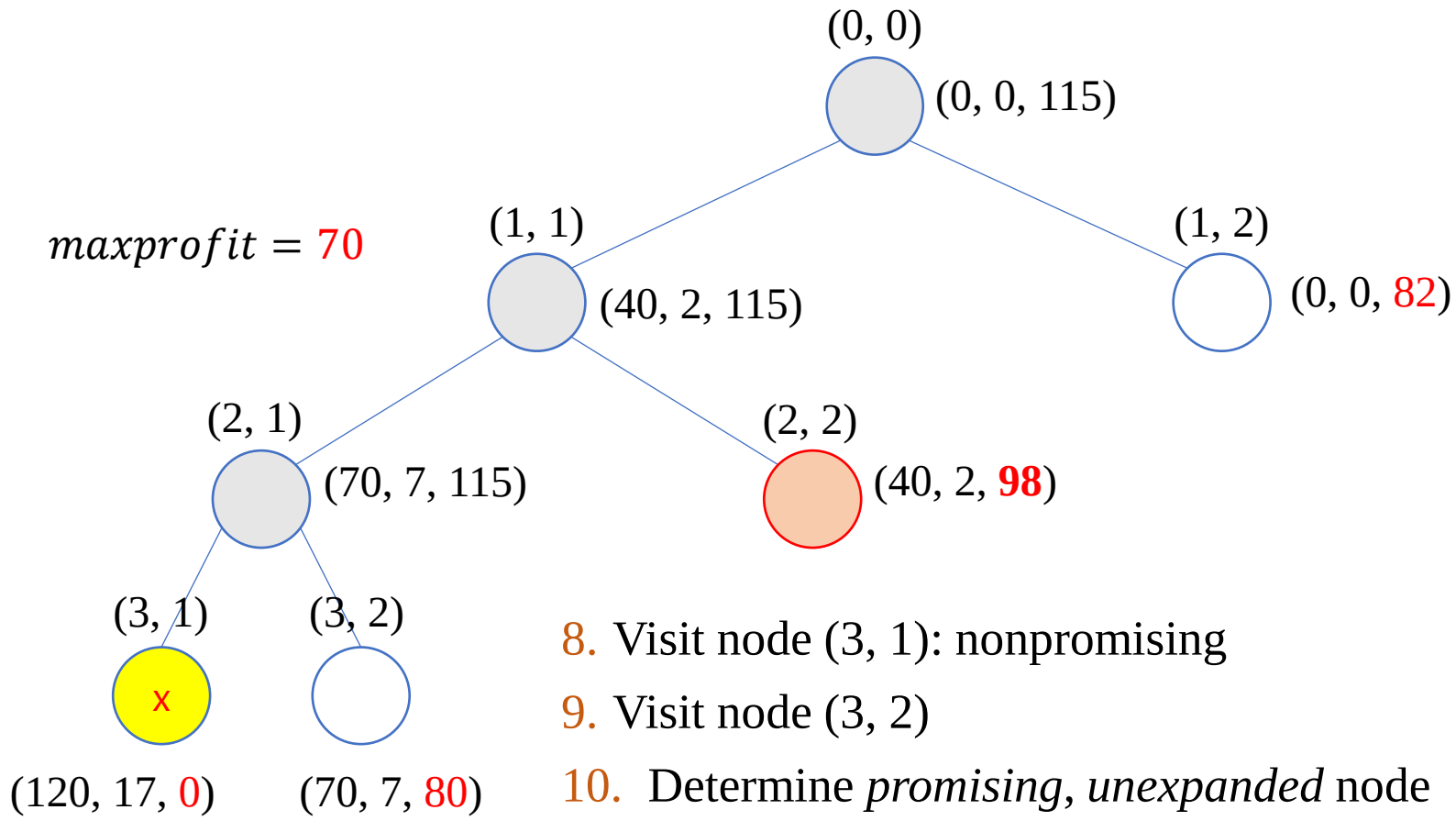
1. Visit node (0, 0) (the root)
   a. (profit, weight, bound) = (0, 0, 115), maxprofit = 115
2. Visit node (1, 1)
   a. (profit, weight, bound) = (40, 2, 115), maxprofit = 40
3. Visit node (1, 2)
   a. (profit, weight, bound) = (0, 0, 82), maxprofit = 40
4. Determine *promising*, *unexpanded* node with the *greatest bound*
   a. We visit the children of (1, 1), the *best promising* node

6.1 분기 한정법과 0-1 배낭 문제



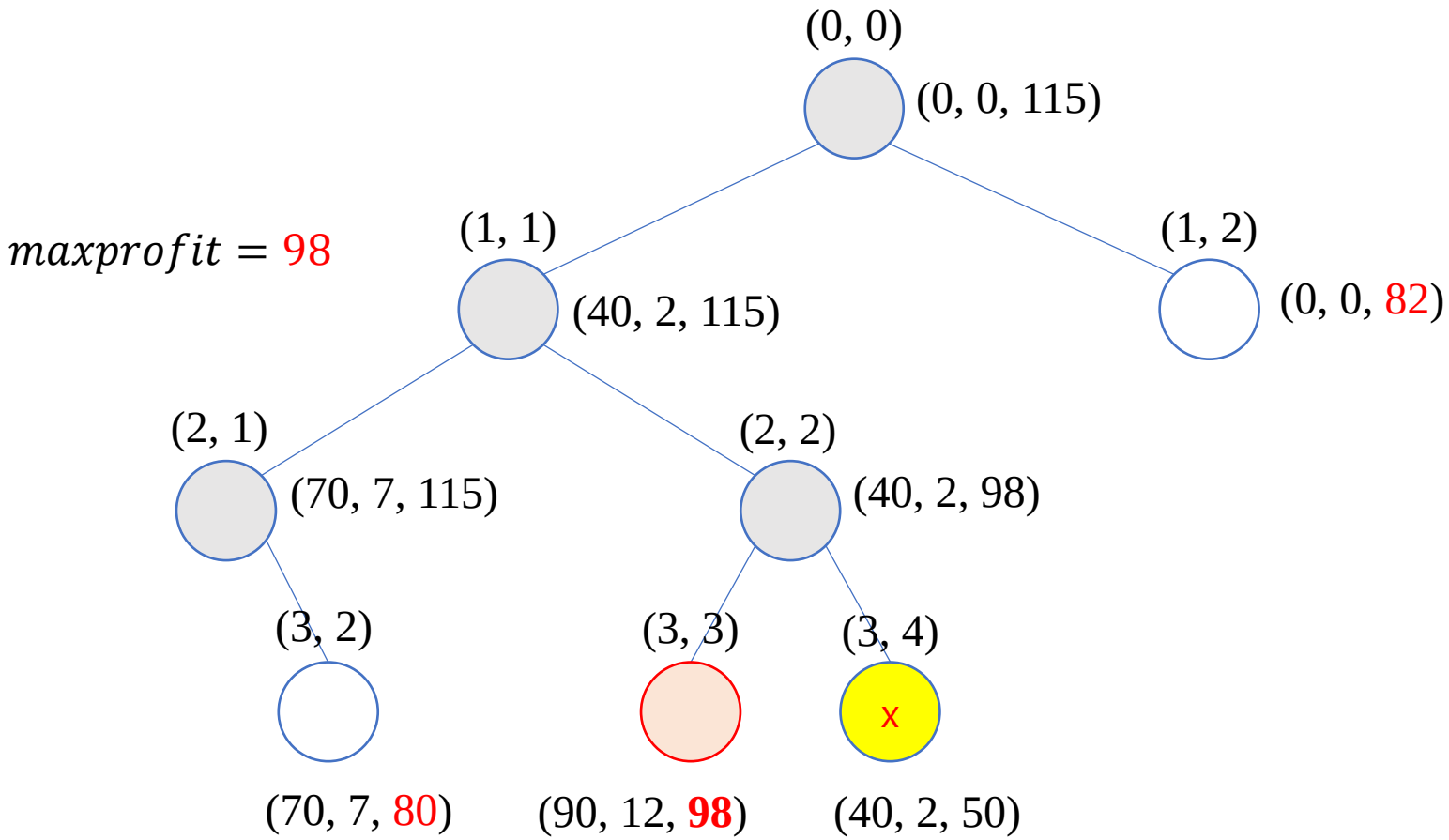
5. Visit node (2, 1): maxprofit = 70
6. Visit node (2, 2)
7. Determine *promising*, *unexpanded* node with the *greatest bound*
   a. We visit the children of (2, 1), the *best promising* node

6.1 분기 한정법과 0-1 배낭 문제

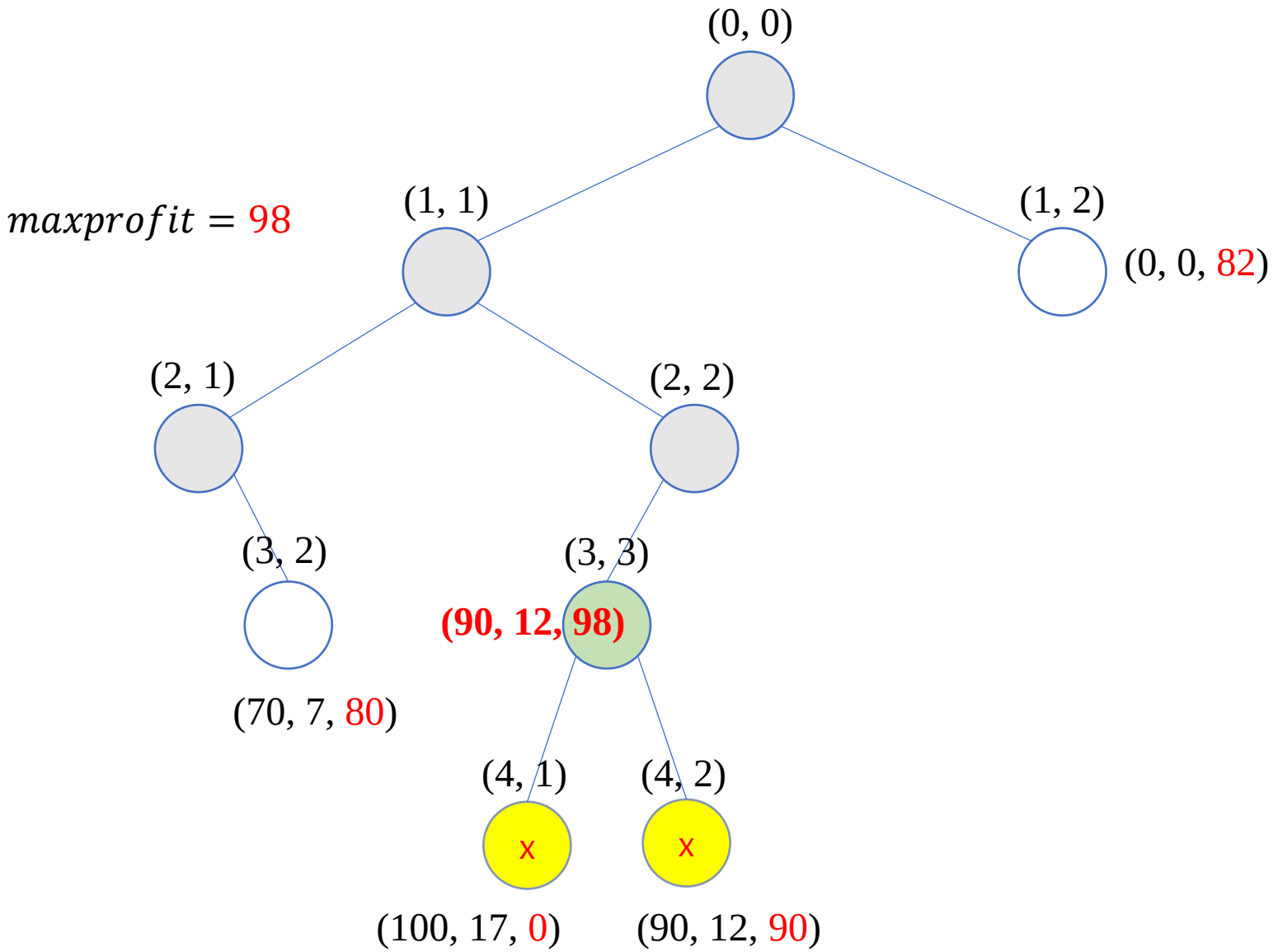


8. Visit node (3, 1): nonpromising
9. Visit node (3, 2)
10. Determine *promising*, *unexpanded* node with the *greatest bound*
    a. We visit the children of (2, 2), the *best promising* node

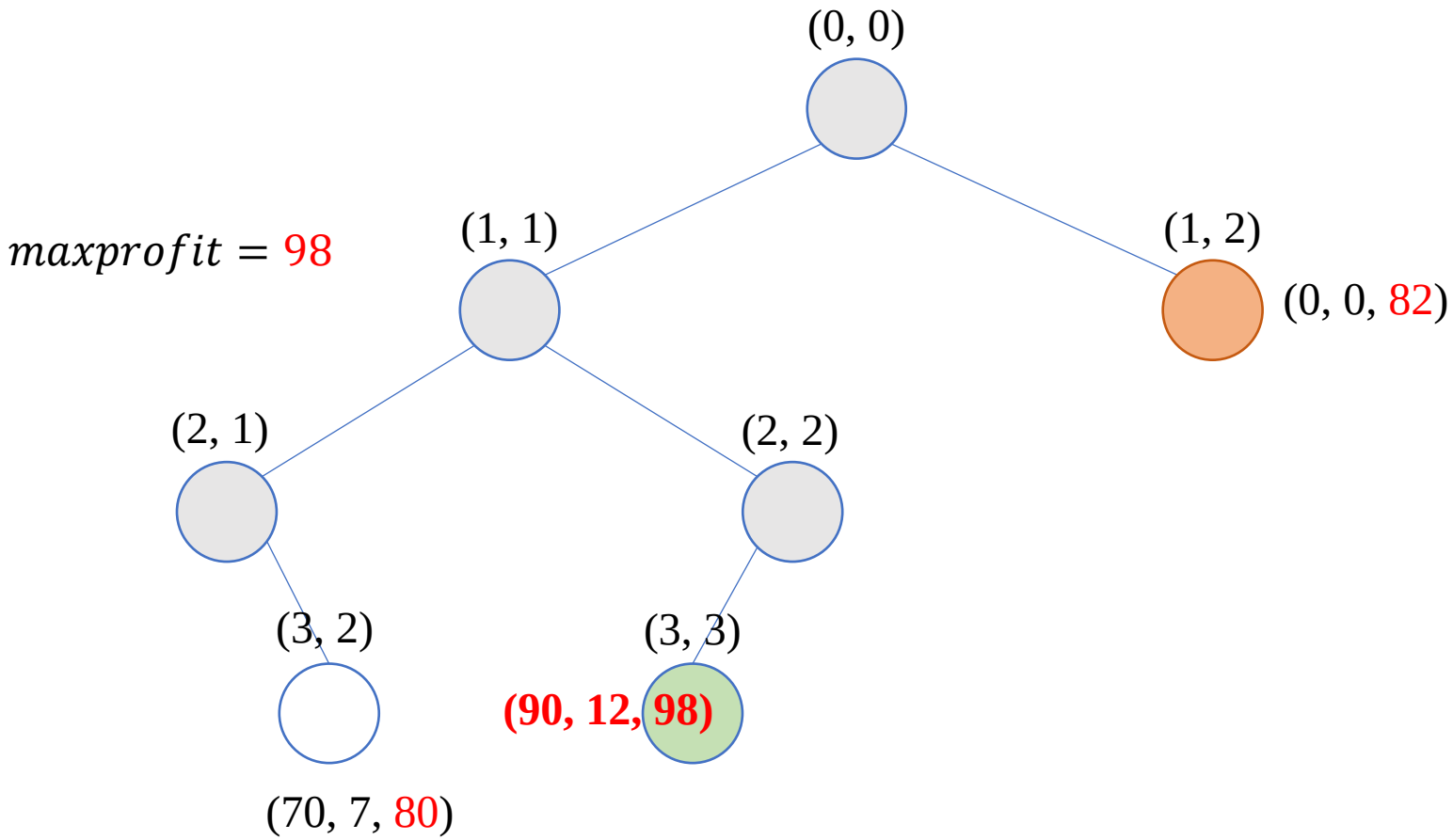
6.1 분기 한정법과 0-1 배낭 문제



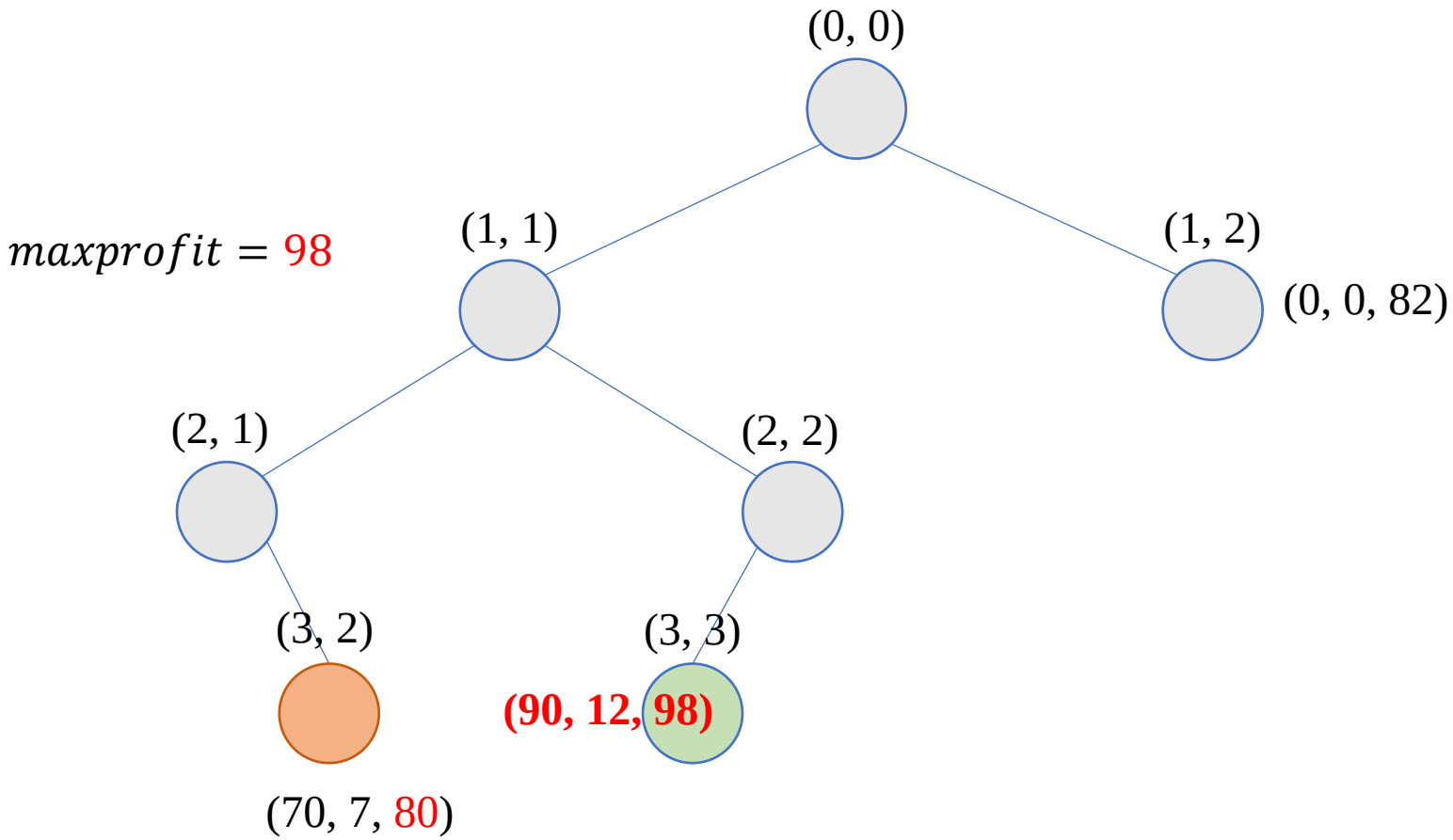
6.1 분기 한정법과 0-1 배낭 문제



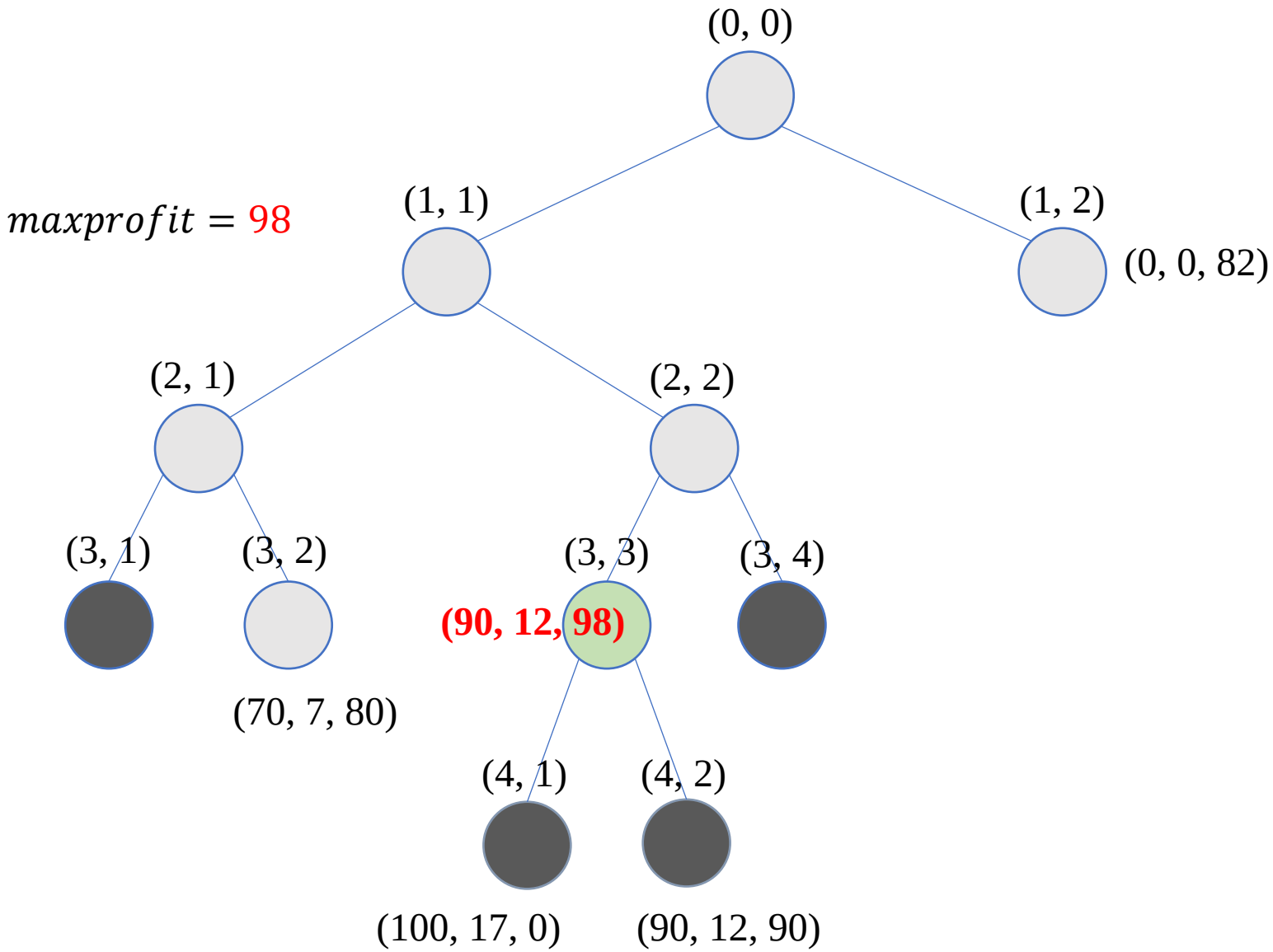
6.1 분기 한정법과 0-1 배낭 문제



6.1 분기 한정법과 0-1 배낭 문제



6.1 분기 한정법과 0-1 배낭 문제



6.1 분기 한정법과 0-1 배낭 문제


```
from queue import PriorityQueue

PQ = PriorityQueue()
data = [4, 1, 3, 2]
for i in range(len(data)):
    PQ.put(data[i])
while (not PQ.empty()):
    print(PQ.get())
```

6.1 분기 한정법과 0-1 배낭 문제


```
PQ = PriorityQueue()
data = ["Apple", "Banana", "Pear"]
for i in range(len(data)):
    PQ.put((len(data[i]), data[i]))
while (not PQ.empty()):
    print(PQ.get())

PQ = PriorityQueue()
data = ["Apple", "Banana", "Pear"]
for i in range(len(data)):
    PQ.put((-len(data[i]), data[i]))
while (not PQ.empty()):
    print(PQ.get()[1])
```

6.1 분기 한정법과 0-1 배낭 문제

Algorithm 6.2: Branch-and-Bound for the 0-1 Knapsack Problem

```
from queue import PriorityQueue

class SSTNode:

    def __init__ (self, level, profit, weight):
        self.level = level
        self.profit = profit
        self.weight = weight
        self.bound = 0

    def print(self):
        print(self.level, self.profit, self.weight, self.bound);
```

Algorithm 6.2: Branch-and-Bound for the 0-1 Knapsack Problem

```
def knapsack4 (p, w, W):
    PQ = PriorityQueue()
    v = SSTNode(0, 0, 0)
    maxprofit = 0
    v.bound = bound(v, p, w)
    PQ.put((-v.bound, v))
    while (not PQ.empty()):
        v = PQ.get()[1]
```

Algorithm 6.2: Branch-and-Bound for the 0-1 Knapsack Problem

```
        if (v.bound > maxprofit):
            level = v.level + 1
            weight = v.weight + w[level]
            profit = v.profit + p[level]
            u = SSTNode(level, profit, weight)
            if (u.weight <= W and u.profit > maxprofit):
                maxprofit = u.profit
            u.bound = bound(u, p, w)
            if (u.bound > maxprofit):
                PQ.put((-u.bound, u))
            u = SSTNode(level, v.profit, v.weight)
            u.bound = bound(u, p, w)
            if (u.bound > maxprofit):
                PQ.put((-u.bound, u))
    return maxprofit
```

6.1 분기 한정법과 0-1 배낭 문제

Algorithm 6.2: Branch-and-Bound for the 0-1 Knapsack Problem

```
def bound(u, p, w):
    n = len(p) - 1
    if (u.weight >= W):
        return 0
    else:
        result = u.profit
        j = u.level + 1
        totweight = u.weight
        while (j <= n and totweight + w[j] <= W):
            totweight += w[j]
            result += p[j]
            j += 1
        k = j
        if (k <= n):
            result += (W - totweight) * p[k] / w[k]
        return result
```

6.1 분기 한정법과 0-1 배낭 문제


```
profit = [0, 40, 30, 50, 10]
weight = [0, 2, 5, 10, 5]
W = 16

maxprofit = knapsack4(profit, weight, W)
print('maxprofit =', maxprofit)
```

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