

파이썬으로 배우는 알고리즘 기초

Chap 6. 분기 한정법



6.2

외판원 문제와 분기 한정법

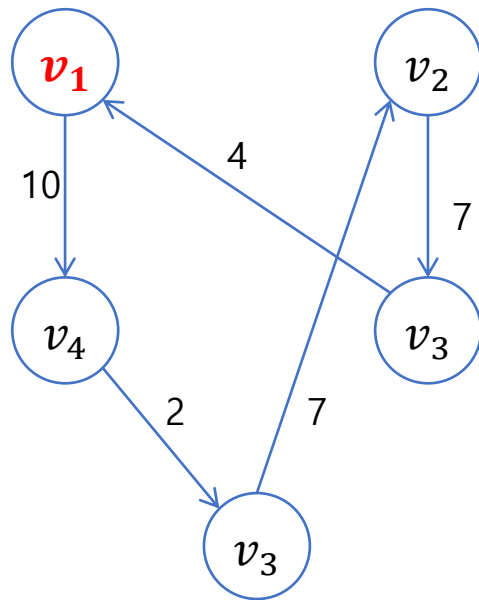




6.2 외판원 문제와 분기 한정법

- **외판원 문제**: Traveling Salesperson Problem
 - 그래프 G 와 출발 정점이 주어졌을 때
 - ▣ 그래프의 모든 노드를 각각 한 번씩만 방문하고 원래의 출발 정점으로 되돌아오는 가장 짧은 경로
 - W : 주어진 그래프 $G = (V, E)$ 의 인접 행렬

| W | 1 | 2 | 3 | 4 | 5 |
|-----|----|----|----|----|----|
| 1 | 0 | 14 | 4 | 10 | 20 |
| 2 | 14 | 0 | 7 | 8 | 7 |
| 3 | 4 | 5 | 0 | 7 | 16 |
| 4 | 11 | 7 | 9 | 0 | 2 |
| 5 | 18 | 7 | 17 | 4 | 0 |





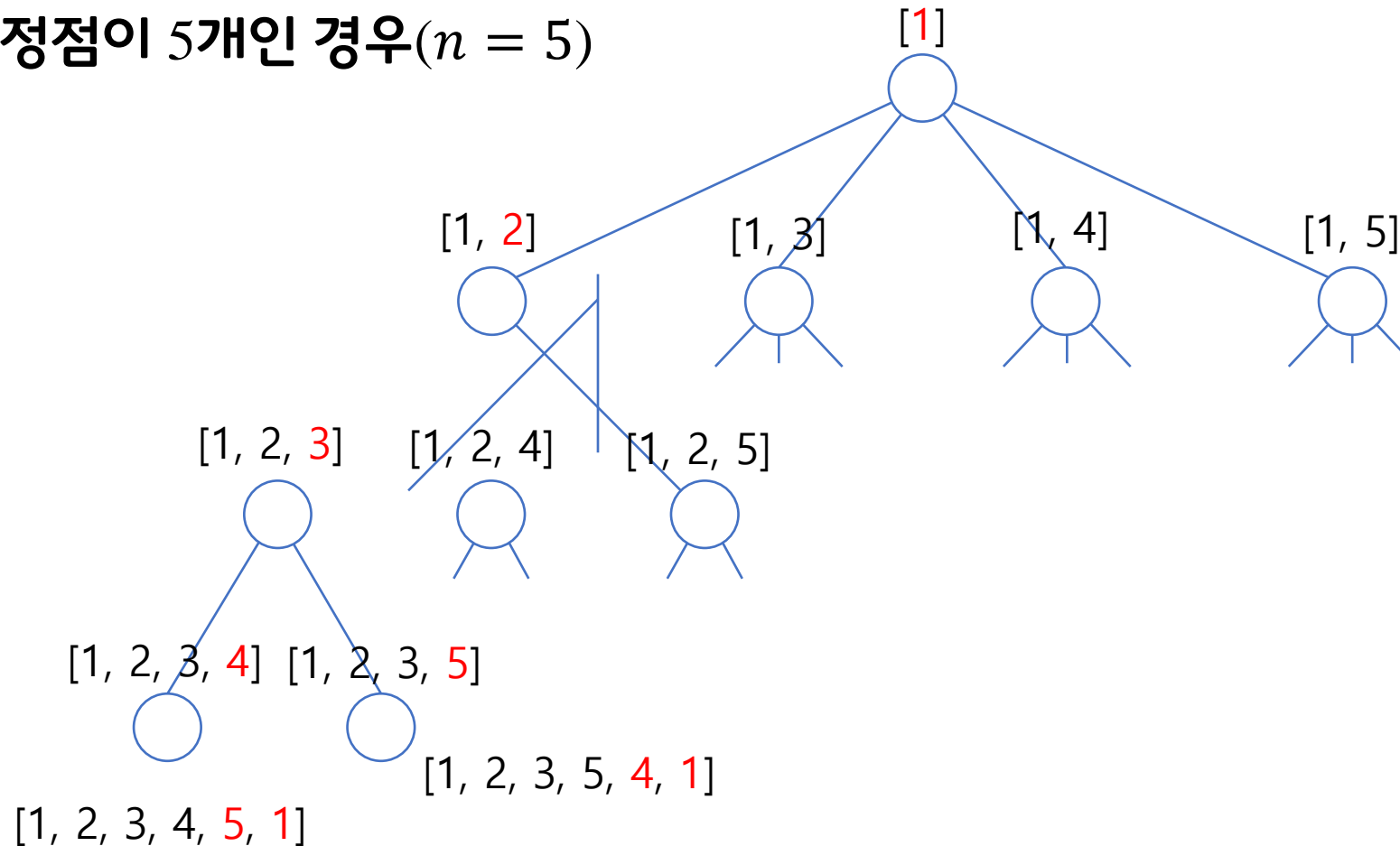
6.2 외판원 문제와 분기 한정법



- 외판원 문제(TSP): 분기 한정으로 풀기
 - 분기 한정 (Branch-and-Bound): 상태공간트리의 **최적우선탐색**
 - 각 노드에서 **한계값(bound)**을 구해야 함
 - 특정 노드에서 얻을 수 있는 경로 길이의 **하한(lower bound)**을 구해야 함
 - **유망 조건**: 경로 길이의 하한이 현재의 최소인 경로 길이보다 작은 경우



- 정점이 5개인 경우($n = 5$)

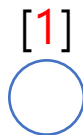




6.2 외판원 문제와 분기 한정법

- 하한값(lower bound) 구하기
 - 각 여행 경로는 정점을 정확히 한 번씩 떠나야 한다.
 - 경로 길이의 하한: 정점을 떠나는 간선 가중치 최소값들의 합

| <i>W</i> | 1 | 2 | 3 | 4 | 5 |
|----------|----|----|----|----|----|
| 1 | X | 14 | 4 | 10 | 20 |
| 2 | 14 | X | 7 | 8 | 7 |
| 3 | 4 | 5 | X | 7 | 16 |
| 4 | 11 | 7 | 9 | X | 2 |
| 5 | 18 | 7 | 17 | 4 | X |



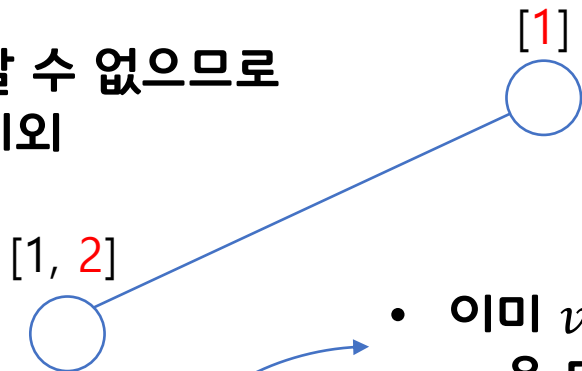
- the lowest bound:
 $4 + 7 + 4 + 2 + 4 = 21$



6.2 외판원 문제와 분기 한정법

- v_2 에서 v_1 으로 바로 되돌아 갈 수 없으므로 v_2 에서 v_1 으로 가는 간선은 제외

| W | 1 | 2 | 3 | 4 | 5 |
|-----|----|----|----|---|----|
| 1 | X | 14 | X | X | X |
| 2 | X | X | 7 | 8 | 7 |
| 3 | 4 | X | X | 7 | 16 |
| 4 | 11 | X | 9 | X | 2 |
| 5 | 18 | X | 17 | 4 | X |



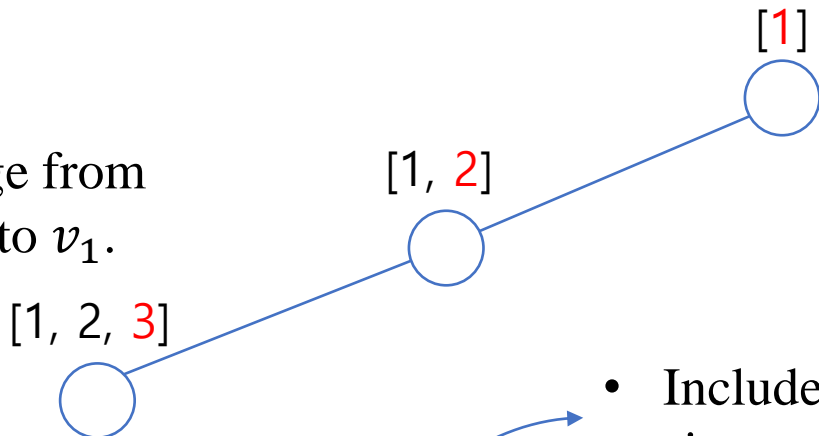
- 이미 v_1 을 떠났으므로 v_1 을 떠나는 비용을 포함시켜야 함
- v_2 는 이미 방문한 노드이므로 v_2 로 가는 간선들을 고려하지 않음

- the lower bounds:
 $14 + 7 + 4 + 2 + 4 = 31$



6.2 외판원 문제와 분기 한정법

- Do not include the edge from v_3 to v_1 , not to return to v_1 .



| W | 1 | 2 | 3 | 4 | 5 |
|----------|----|----|---|---|----|
| 1 | X | 14 | X | X | X |
| 2 | X | X | 7 | X | X |
| 3 | X | X | X | 7 | 16 |
| 4 | 11 | X | X | X | 2 |
| 5 | 18 | X | X | 4 | X |

- Include the cost of leaving v_1, v_2 , since we have already left v_1, v_2 .

- Do not include the edge to v_2, v_3 since we have already been at v_2, v_3 .

- the lower bounds:

$$21 + 7 + 2 + 4 = 34$$




6.2 외판원 문제와 분기 한정법

- 분기한정 가지치기 최고우선탐색:
 - 한계값은 리프 노드가 아닌 노드(내부 노드)에서 계산
 - 여행 경로의 길이: 리프 노드에서 계산

$minlength = \infty$

[1]

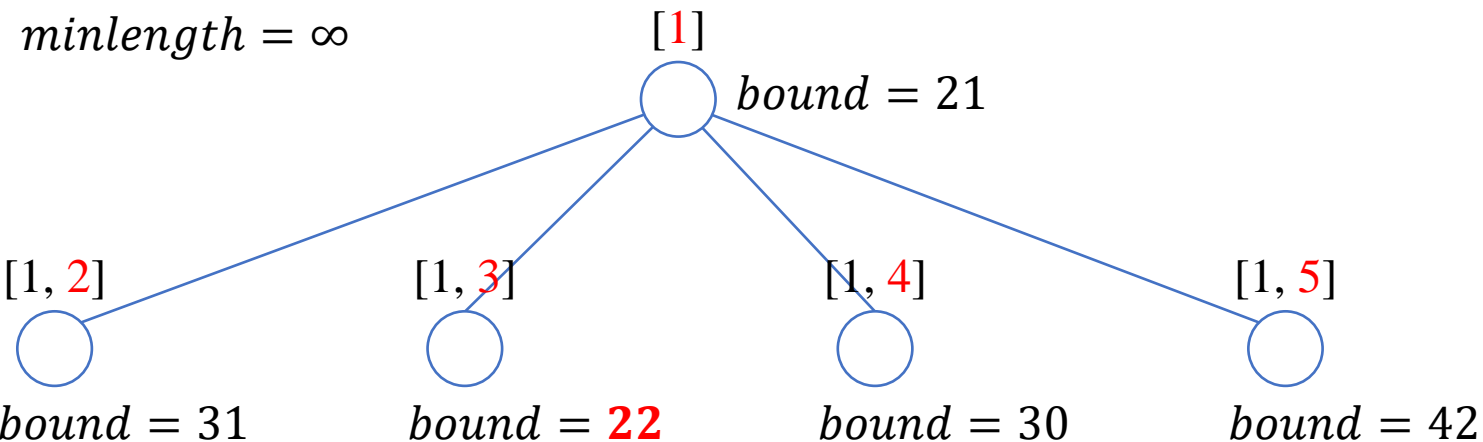


bound = 21

1. Visit node containing [1] (the root)
 - a. Compute bound to be 21
 - b. Set *minlength* to INF.



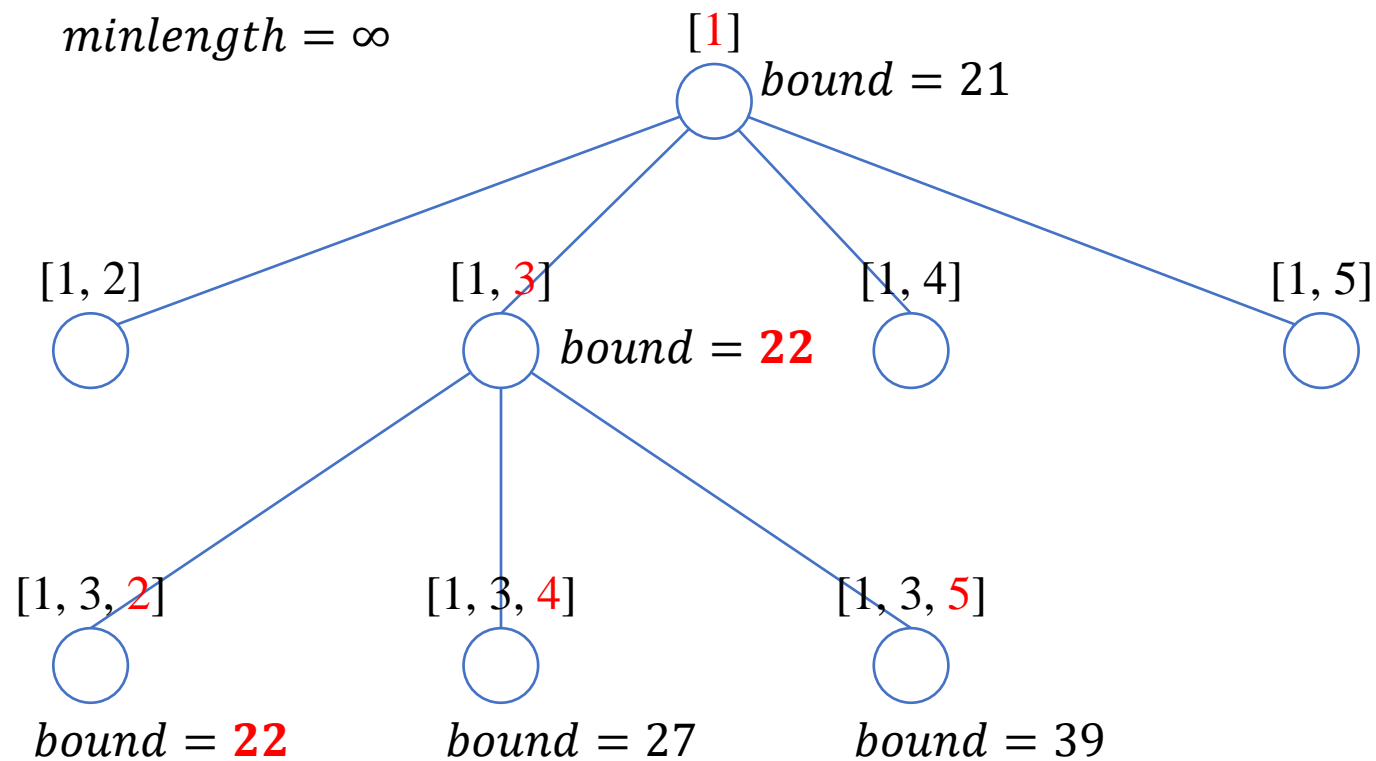
6.2 외판원 문제와 분기 한정법



2. Visit node containing [1, 2]
 - a. Compute bound to be 31.
3. Visit node containing [1, 3]
 - a. Compute bound to be 22.
4. Visit node containing [1, 4]
 - a. Compute bound to be 30.
5. Visit node containing [1, 5]
 - a. Compute bound to be 42.
6. Determine promising, unexpanded node with the smallest bound.
 - a. That is the node [1, 3]
 - b. Visit its children next



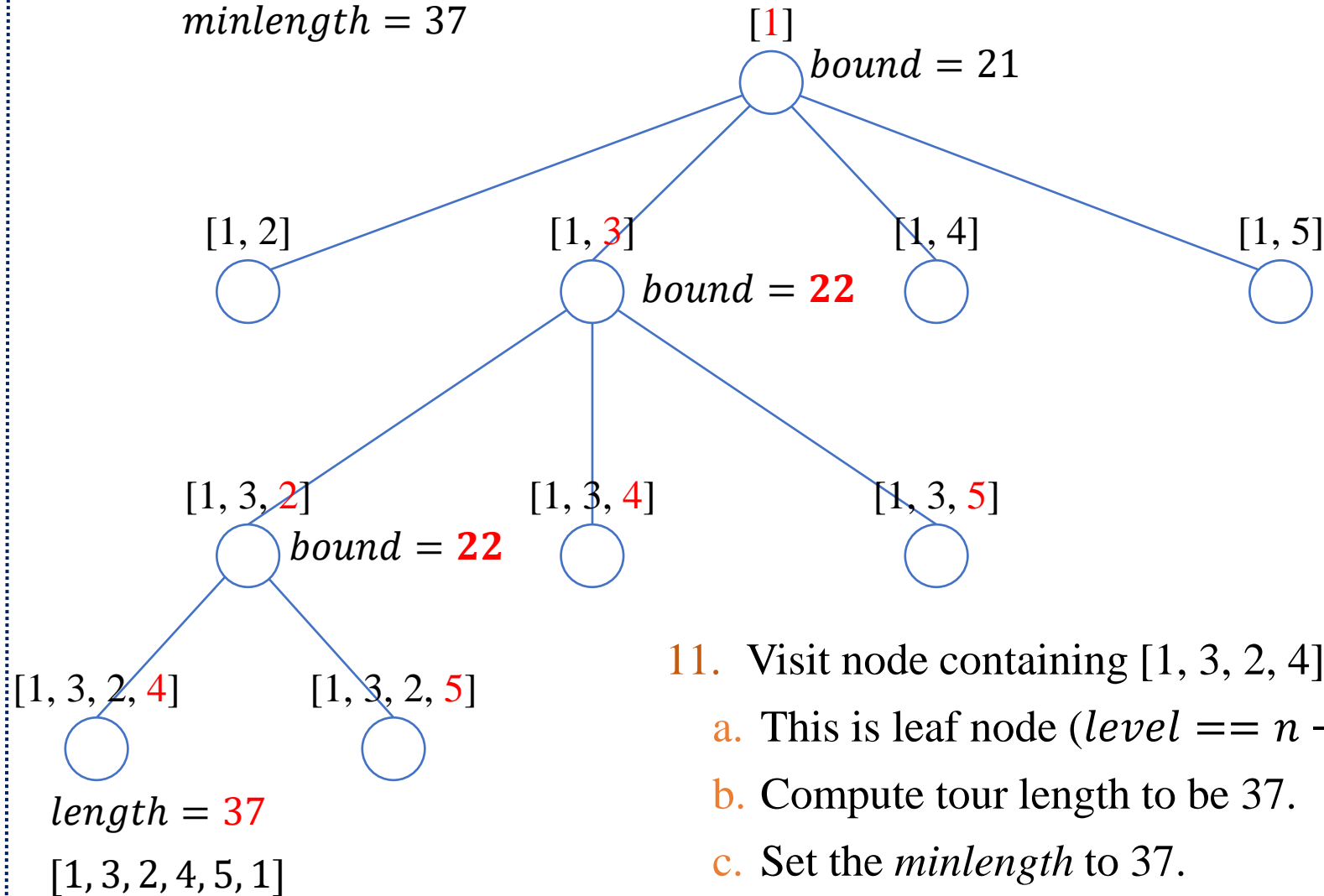
6.2 외판원 문제와 분기 한정법





6.2 외판원 문제와 분기 한정법

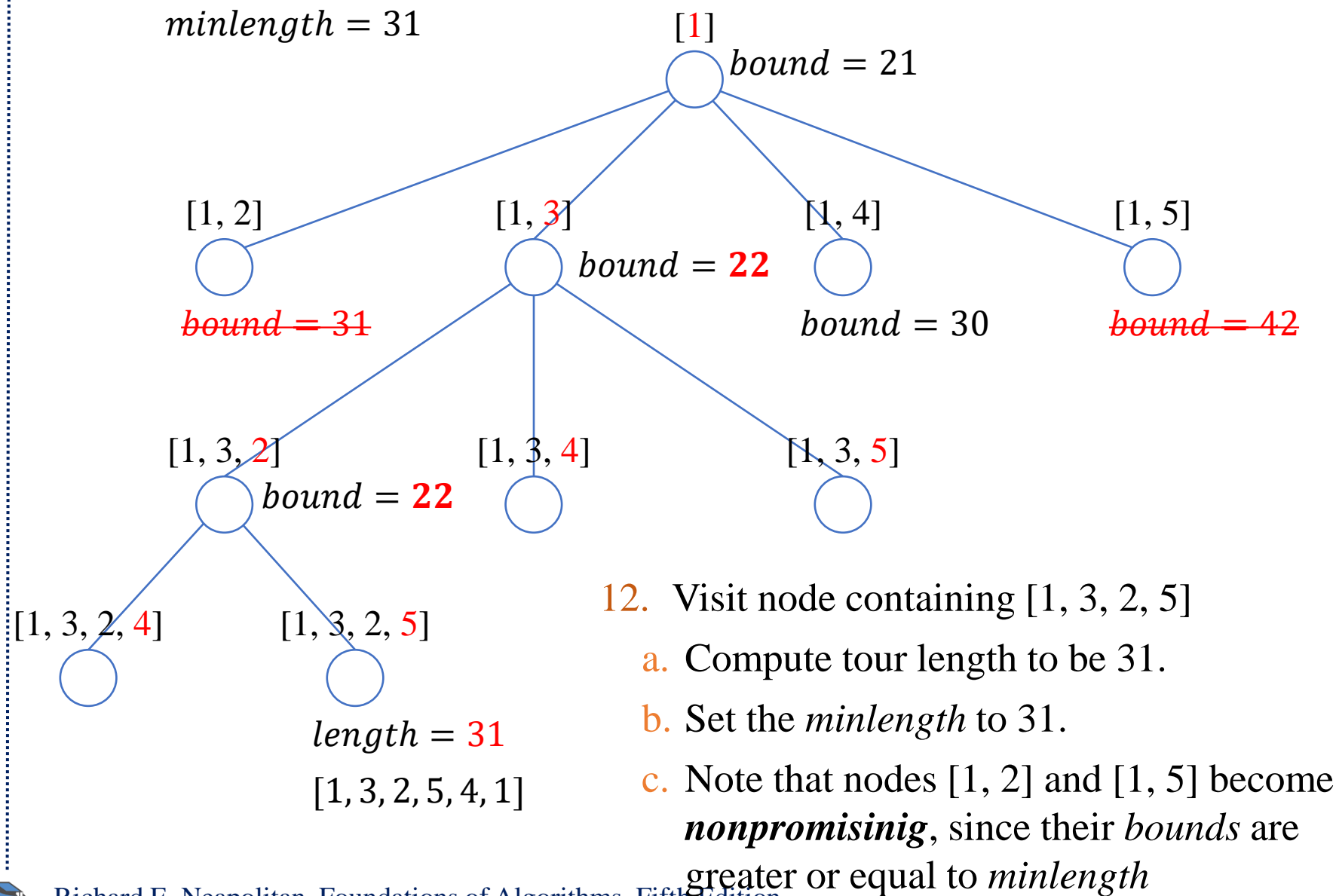
$minlength = 37$



11. Visit node containing $[1, 3, 2, 4]$
 - a. This is leaf node ($level == n - 2$)
 - b. Compute tour length to be 37.
 - c. Set the $minlength$ to 37.



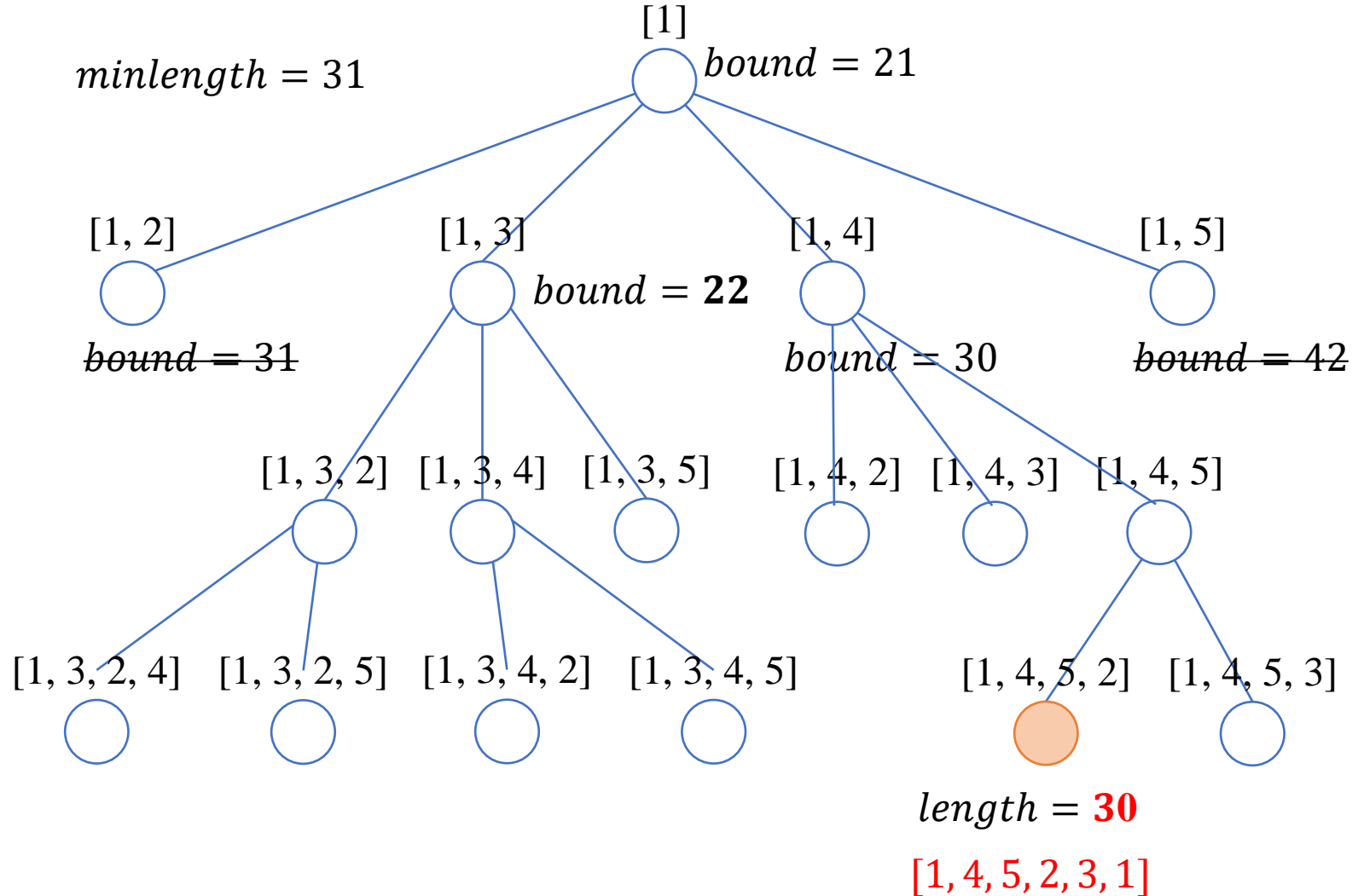
6.2 외판원 문제와 분기 한정법





6.2 외판원 문제와 분기 한정법

■ 최종 상태공간트리





6.2 외판원 문제와 분기 한정법



Algorithm 6.3: Best-First-Search with Branch-and-Bound for the TSP

```
class SSTNode:

    def __init__(self, level):
        self.level = level
        self.path = []
        self.bound = 0

    def contains(self, x):
        for i in range(len(self.path)):
            if (self.path[i] == x):
                return True
        return False
```



6.2 외판원 문제와 분기 한정법



Algorithm 6.3: Best-First-Search with Branch-and-Bound for the TSP

```
def travel2 (W):  
    global minlength, opttour  
    n = len(W) - 1  
    PQ = PriorityQueue()  
    v = SSTNode(0)  
    v.path = [1]  
    v.bound = bound(v, W)  
    minlength = INF  
    PQ.put((v.bound, v))
```



6.2 외판원 문제와 분기 한정법



Algorithm 6.3: Best-First-Search with Branch-and-Bound for the TSP

```
while (not PQ.empty()):
    v = PQ.get()[1]
    if (v.bound < minlength):
        for i in range(2, n + 1):
            if (v.contains(i)):
                continue
            u = SSTNode(v.level + 1)
            u.path = v.path[:]
            u.path.append(i)
```




6.2 외판원 문제와 분기 한정법



Algorithm 6.3: Best-First-Search with Branch-and-Bound for the TSP

```
if (u.level == n - 2):
    for k in range(2, n + 1):
        if (not u.contains(k)):
            u.path.append(k)
    u.path.append(1)
    if (length(u.path, W) < minlength):
        minlength = length(u.path, W)
        opttour = u.path[:]
else:
    u.bound = bound(u, W)
    if (u.bound < minlength):
        PQ.put((u.bound, u))
```



6.2 외판원 문제와 분기 한정법



Algorithm 6.3: Best-First-Search with Branch-and-Bound for the TSP

```
def bound(v, W):
    n = len(W) - 1
    total = length(v.path, W)
    for i in range(1, n + 1):
        if (hasOutgoing(i, v.path)):
            continue
        min = INF
        for j in range(1, n + 1):
            if (i == j): continue
            if (hasIncoming(j, v.path)): continue
            if (j == 1 and i == v.path[len(v.path) - 1]): continue
            if (min > W[i][j]): min = W[i][j]
        total += min
    return total
```



6.2 외판원 문제와 분기 한정법



Algorithm 6.3: Best-First-Search with Branch-and-Bound for the TSP

```
def length(path, W):  
    total = 0  
    prev = 1  
    for i in range(len(path)):  
        if (i != 0):  
            prev = path[i - 1]  
            total += W[prev][path[i]]  
            prev = path[i]  
    return total
```

```
def hasOutgoing(v, path):  
    for i in range(0, len(path) - 1):  
        if (path[i] == v):  
            return True  
    return False
```

```
def hasIncoming(v, path):  
    for i in range(1, len(path)):  
        if (path[i] == v):  
            return True  
    return False
```



6.2 외판원 문제와 분기 한정법



```
INF = 999
W = [
    [-1, -1, -1, -1, -1, -1],
    [-1, 0, 14, 4, 10, 20],
    [-1, 14, 0, 7, 8, 7],
    [-1, 4, 5, 0, 7, 16],
    [-1, 11, 7, 9, 0, 2],
    [-1, 18, 7, 17, 4, 0],
]

minlength = INF
opttour = None
travel2(W)

print("minlength =", minlength)
print("optimal tour =", opttour)
```



주니온TV@Youtube

자세히 보면 유익한 코딩 채널

<https://bit.ly/2JXXGqz>

주니온TV@Youtube

자세히 보면 유익한 코딩 채널

- 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>