KOREATECH
한국기술교육대학교

14장. 상태 공간 트리의 탐색

Youn-Hee Han
LINK@KOREATECH
http://link.koreatech.ac.kr

인공지능은 미국인들 특유의 천진난만함의 표현이다.
– 에드거 다익스트라

에츠허르 데이크스트라

네덜란드 컴퓨터 과학자

에츠허르 비버 데이크스트라는 네덜란드의 컴퓨터 과학자이다. 1972년에 프로그래밍 언어 분야에 대한 지대한 공헌을 인정받아 튜링상을 수상했다. 위키백과

출생: 1930년 5월 11일, 네덜란드 로테르담

사망 정보: 2002년 8월 6일, 네덜란드 뉘넌

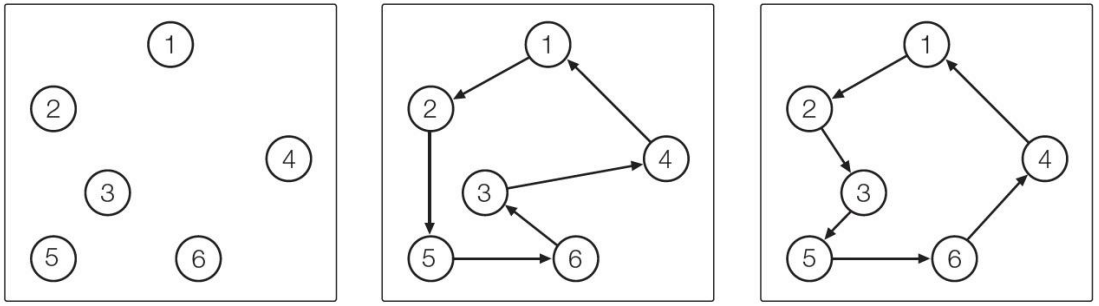
학습 목표

- ◈ 상태 공간 트리의 탐색을 이해한다.
- ◈ 상태 공간 트리가 무엇인지 이해한다.
- ◈ 백트래킹 기법의 작동 원리를 이해한다.
- ◈ 한정 분기의 작동 원리를 이해하고, 백트래킹에 비해 장점이 무엇인지 이해하도록 한다.
- ◈ A* 알고리즘의 작동 원리를 이해하고, 어떤 문제들이 A* 알고리즘의 적용 대상인지 감지하도록 한다. (SKIP)

01. 상태 공간 트리

TSP 문제와 상태 공간 트리

◈ TSP (Traveling Salesman Problem)



(a) TSP 예 (b) 해의 한 예 (c) 최적해

그림 14-1 TSP 문제의 예와 해들

- 해밀토니안 사이클
  - 주어진 그래프에서 모든 정점을 한번씩만 순회하고 돌아오는 사이클

- TSP 문제
  - 주어진 그래프에 존재하는 여러 개의 해밀토니안 사이클 중 가장 짧은 것을 찾는 문제
  - 두 가지 TSP 세부 문제
    - 대칭형 (Symmetric) TSP
      - » 임의의 두 정점 u, v를 잇는 두 간선 $\langle u, v\rangle$와 $\langle v, u\rangle$의 길이가 동일
    - 비대칭형 (Asymmetric) TSP
      - » 임의의 두 정점 u, v를 잇는 두 간선 $\langle u, v\rangle$와 $\langle v, u\rangle$의 길이가 다름

TSP 문제와 상태 공간 트리

◈ 상태 공간 트리

- 상태 공간 트리 (State Space Tree)
  - 문제 해결 과정의 중간 상태를 각각 한 노드로 나타낸 트리
- 상태 공간 트리의 각 노드
  - 문제 풀이 과정 중의 진행 상황 또는 상태
  - 리프 노드는 하나의 Candidate Solution (후보 해답)
    - ➢ TSP 문제에서는 리프 노드에 도달하면 하나의 해밀토니안 사이클 산출
- 모든 경우의 수 탐색(사전식 탐색)에 용이하게 사용 가능

◈ 비대칭 TSP 문제 예

	1	2	3	4	5
1	0	10	10	30	25
2	10	0	14	21	10
3	10	18	0	7	9
4	8	11	7	0	3
5	14	10	10	3	0

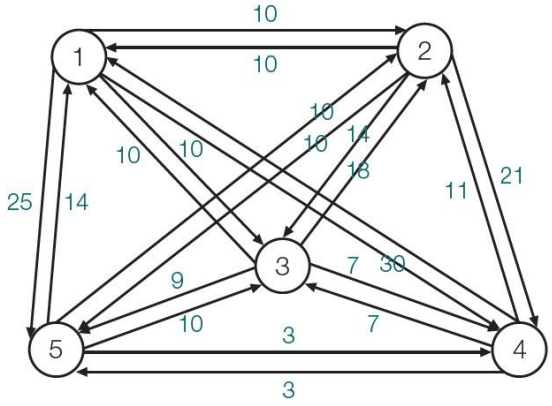


그림 14-2 비대칭 TSP 문제의 인접 행렬과 그래프

TSP 문제와 상태 공간 트리

	1	2	3	4	5
1	0	10	10	30	25
2	10	0	14	21	10
3	10	18	0	7	9
4	8	11	7	0	3
5	14	10	10	3	0

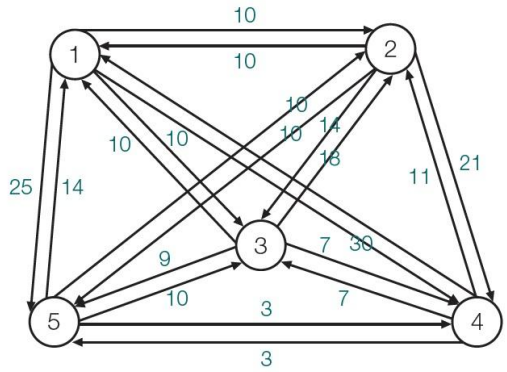


그림 14-2 비대칭 TSP 문제의 인접 행렬과 그래프

◈ 비대칭 TSP 문제의 상태 공간 트리

- 리프 노드의 네모에 적힌 숫자는 각 헤밀토니안 사이클 길이
- 모든 헤밀토니안 사이클의 개수
  - 4! = 24
- 총 노드의 개수
  - 41개

1+4+12+24=41

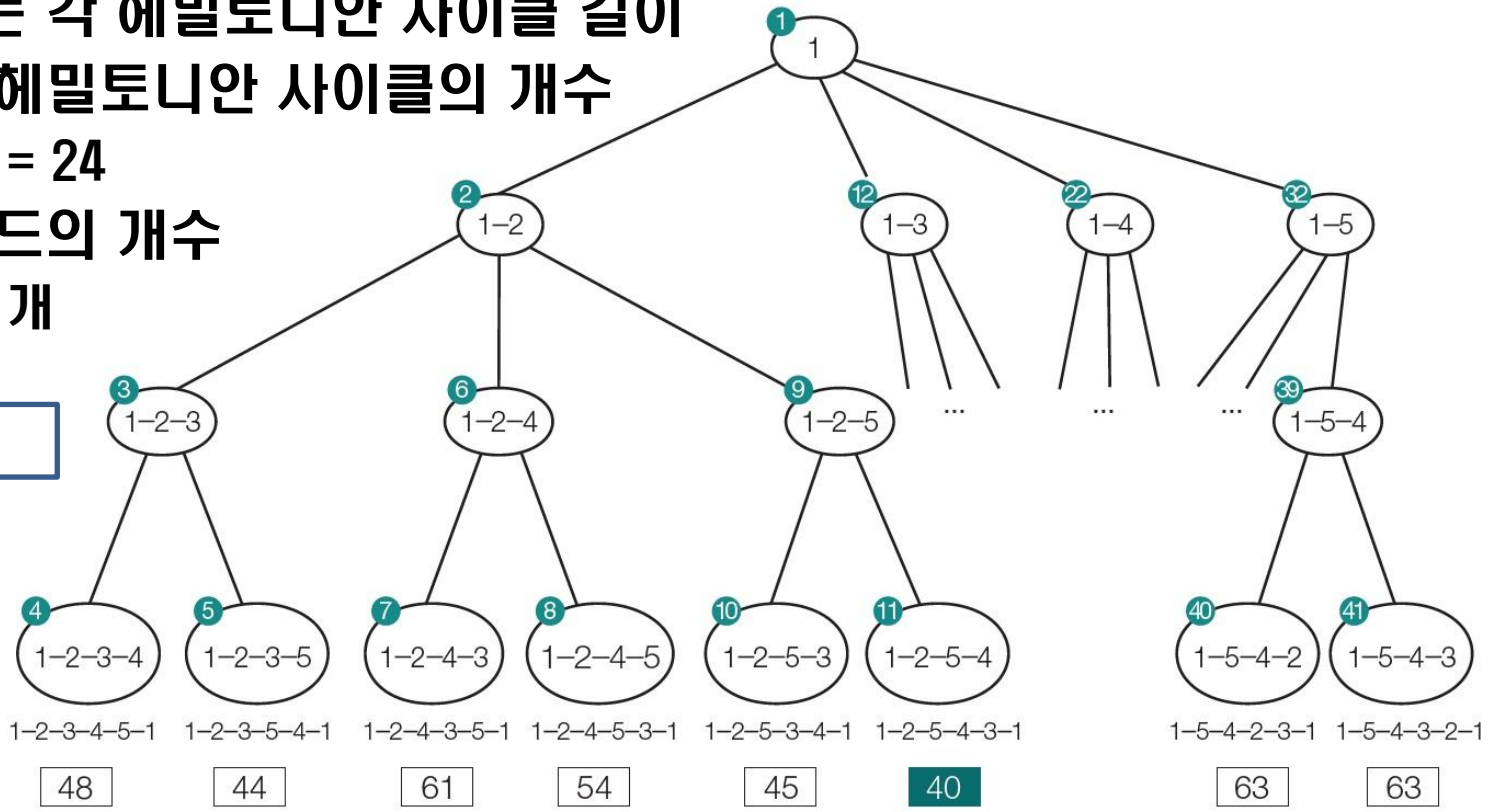


그림 14-3 [그림 14-2]의 TSP 예를 대상으로 한 사전식 탐색의 예

상태 공간 트리와 알고리즘

◈ 상태 공간 트리와 관련된 알고리즘 전략

- 백트래킹 (Backtracking)

- 분기한정 (Branch-and-Bound)
  - 최고 우선 검색 분기 한정 가지치기
    (Best-first search with branch-and-bound pruning)

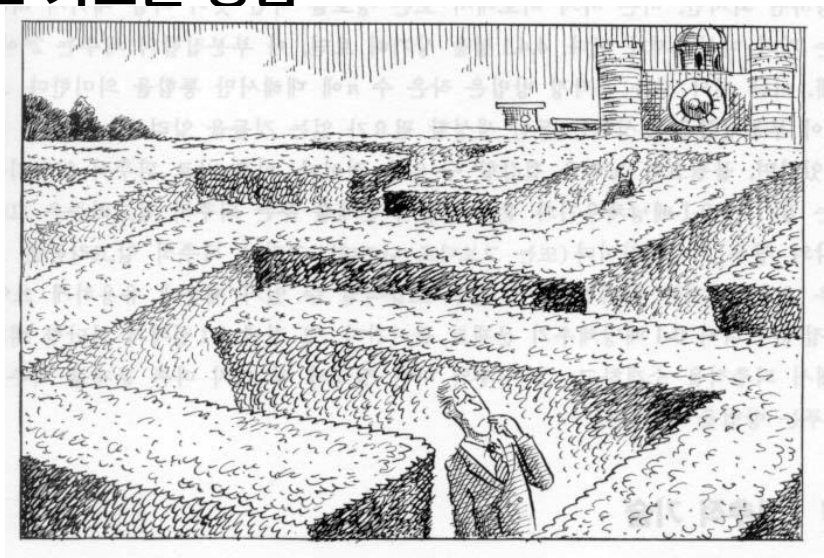
- A* 알고리즘 (A* Algorithm)

02. 백트래킹

백트래킹

◈ 백트래킹 (Backtracking)

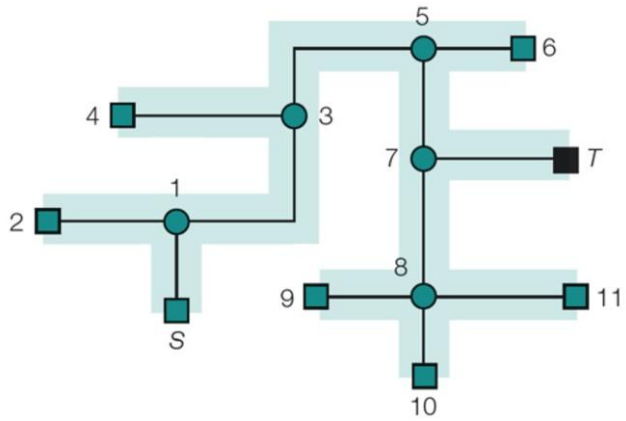
- 상태 공간 트리에서 새로운 탐색이 무의미하다고 판단되면, 다른 새로운 탐색이 가능한 선택 포인트 (choice point) 로 backtrack 하여 새로운 탐색을 시도
- 더 이상의 선택 포인트가 존재하지 않으면, 탐색은 실패로 끝난다
  - 되추적은 갈림길에 표시를 해 두고 막다른 골목에 다다르면 갈림길까지 되돌아가서 다른 골목으로 가보는 방법
- 깊이 우선 탐색과 관련

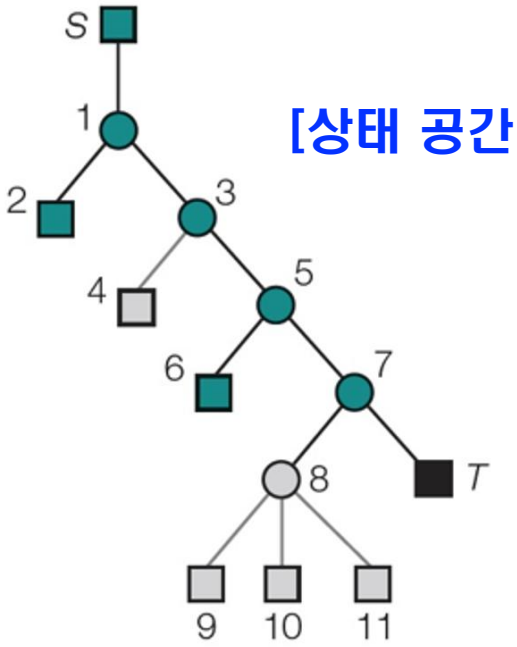


미로 찾기 문제

◈ 미로 찾기 문제 정의

- S: 시작 지점, T: 목표 지점
- 각 끝점과 분기점을 노드(정점)으로 하여 상태 공간 트리 구성





[상태 공간 트리]

- 운이 좋으면 시행착오를 덜 거치면서 목적지에 도착
- 최악의 경우에는 모든 경우를 모두 거쳐서 목적지에 도착

미로 찾기 문제

◈ 미로 찾기 문제 의사 코드

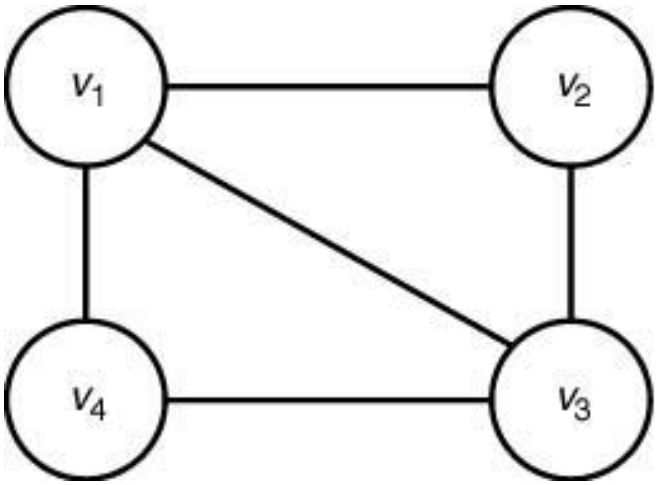
알고리즘 14-1 미로 찾기 문제를 위한 백트래킹 알고리즘

```
maze(v)
{
    visited[v]=YES;
    if (v=T) then {return "성공";}        ▷ 끝내기
    for each x ∈ L(v)                     ▷ L(v) : 정점 v와 인접한 정점 집합
        if (visited[x]=NO) then {
            prev[x] ← v;
            maze(x);
        }
}
```

색칠 문제

◈ 색칠 문제 (Coloring Problem)

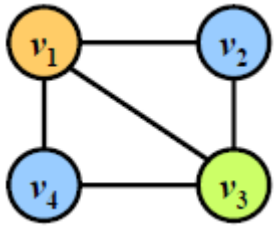
– 주어진 그래프에서 인접한 정점은 같은 색을 칠할 수 없는 조건 하에서 k개의 색상을 사용하여 전체 그래프를 칠할 수 있는가?



- 응용 분야: 지도 색칠하기

- 이 그래프에서 두 가지 색으로 문제를 풀기는 불가능하다.
- 세 가지 색을 사용하면 총 여섯 개의 해답을 얻을 수 있다.
  - 여러 답 중 하나

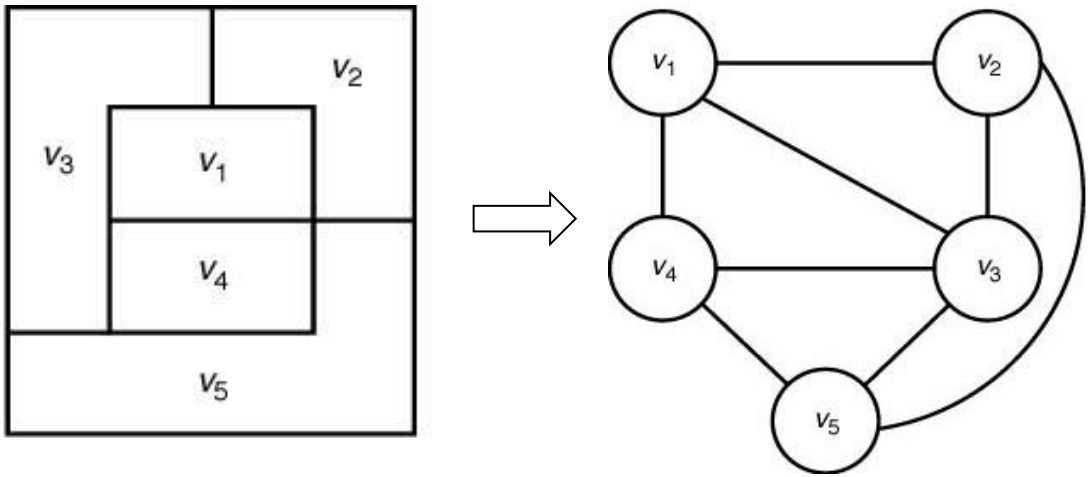
v_1 : 색1
v_2 : 색2
v_3 : 색3
v_4 : 색2



색칠 문제

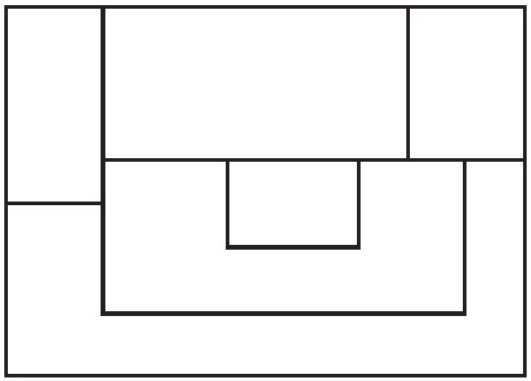
◈ 일반 지도를 그래프로 변환

- 임의의 지도에서 각 지역을 그래프의 노드로 하고, 한 지역이 어떤 다른 지역과 인접해 있으면 그 지역들을 나타내는 노드들 사이에 이음선을 연결한다.
- 그러면, 모든 지도(map)는 그에 상응하는 그래프(graph)로 변형하여 표현할 수 있다.

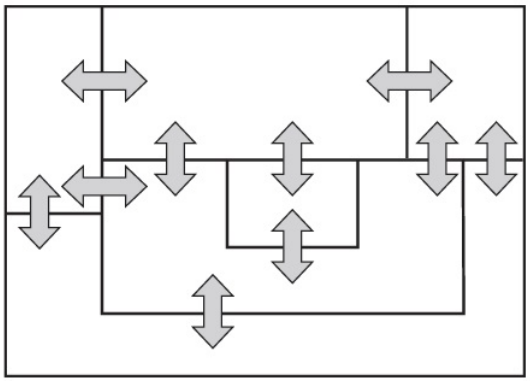


색칠 문제

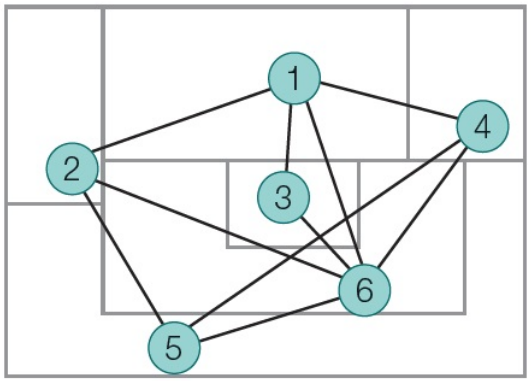
◈ 일반 지도를 그래프로 변환



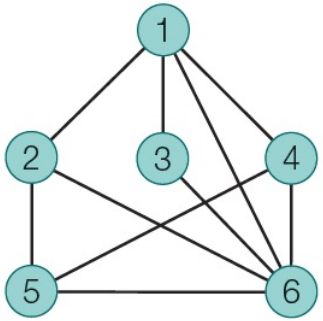
(a) 지도



(b) 구역 간의 인접 관계



(c) 연결 관계를 정점과 간선으로 나타낸 것



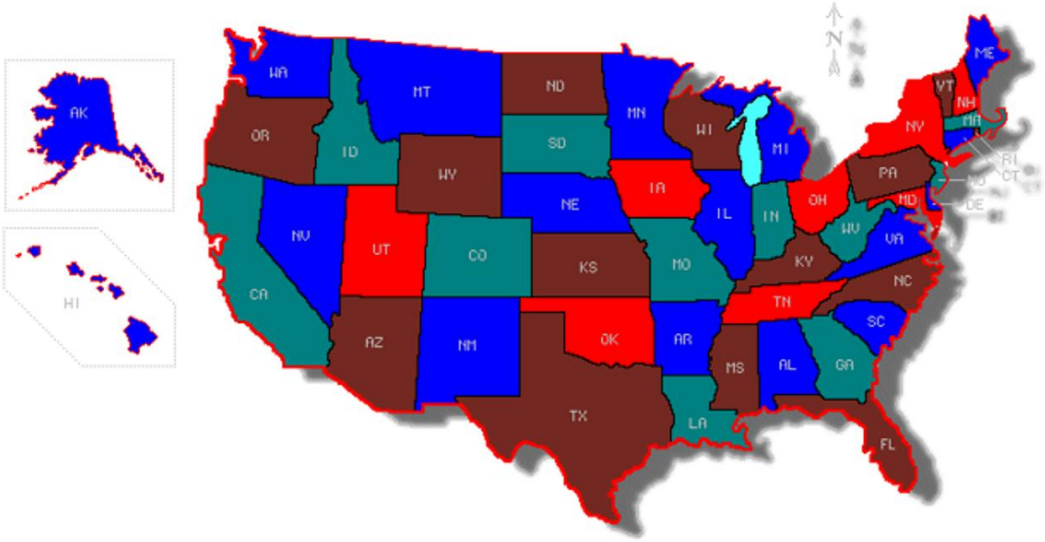
(d) (c)와 동일한 그래프

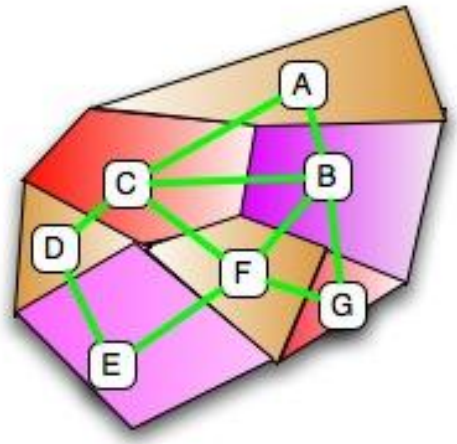
그림 14–5 지도 색칠 문제의 예

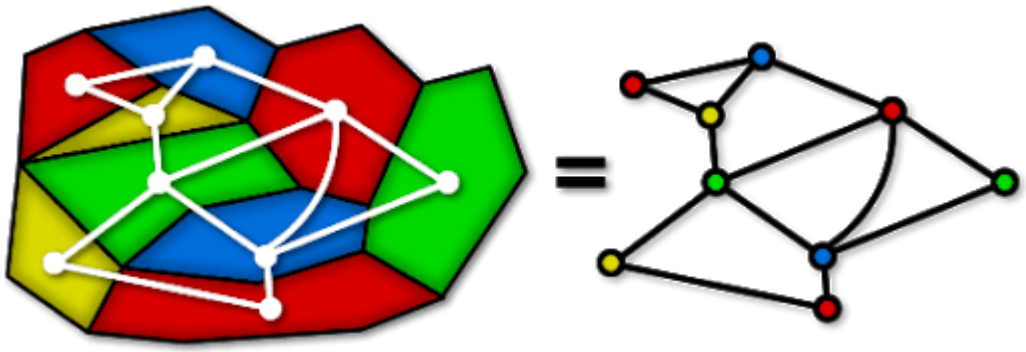
색칠 문제

◈ 일반 지도를 그래프로 변환

[4개 색상으로 미국 지도 색칠]



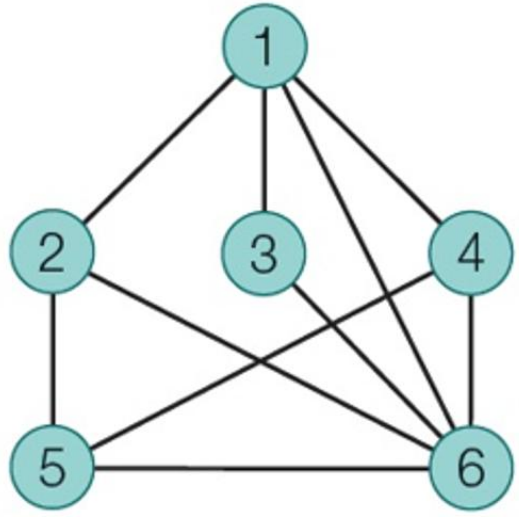




색칠 문제

◈ 색칠 문제를 위한 상태 공간 트리 구성

- 노드 정의
  - (i, c)
    - ➢ 정정 i에 색상 c를 칠함
- valid 개념 도입



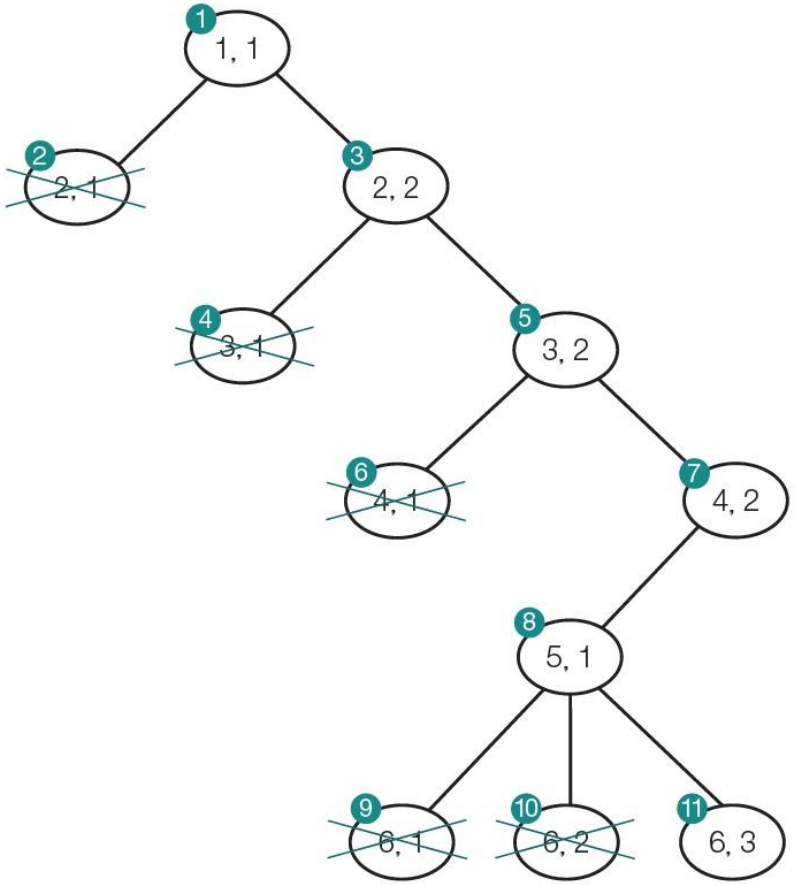


그림 14-6 [알고리즘 14-2]의 작동 과정에 대응되는 상태 공간 트리

색칠 문제

◈ 색칠 문제 의사 코드

- K개의 색으로 주어진 그래프를 색칠할 수 있는가?
- 깊이 우선 탐색 진행
- valid(i, c)
  - 노드 (i, c)가 유효한지 검증

알고리즘 14-2 색칠 문제를 위한 백트래킹 알고리즘

메인에서는 kColoring(1, 1) 호출

```
kColoring(i, c)
▷ i : 정점, c : color
▷ 질문 : 정점 i-1까지는 제대로 칠이 된 상태에서 정점 i를 색상 c로 칠하려면
         K개의 색으로 충분한가?
{
    if (valid(i, c)) then {
        color[i] ← c;
        if (i = n) then { return TRUE; }
        else {
            result ← FALSE;
            d ← 1;                                  ▷ d : color
            while (result = FALSE and d ≤ k) {
                result ← kColoring(i+1, d);         ▷ i+1 : 다음 정점
                d++;
            }
        }
        return result;
    } else { return FALSE; }
}
```

색칠 문제

◈ 색칠 문제에서 valid(i, c) 의사 코드

```
valid(i, c)
▷ i : 정점, c : color
▷ 질문 : 정점 i-1까지는 제대로 칠이 된 상태에서 정점 i를 색상 c로 칠하면
         이들과 색이 겹치지 않는가?
{
    for j ← 1 to i-1 {
        ▷ 정점 i와 j 사이에 간선이 있고, 두 정점이 같은 색상이면 안 된다.
        if ((i, j) ∈ E and color[j] = c) then return FALSE;
    }
    return TRUE;
}
```

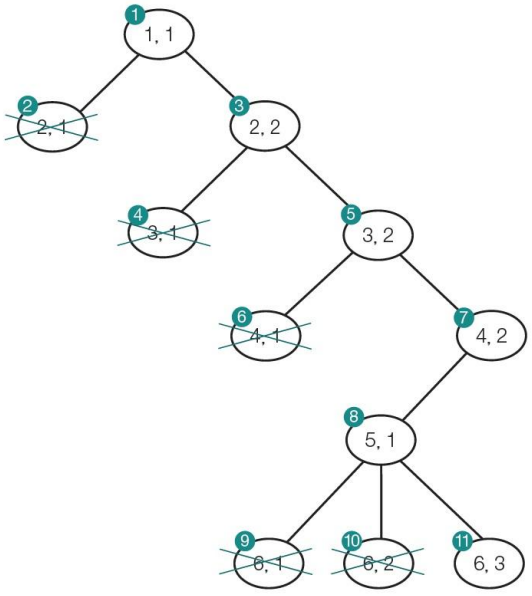


그림 14-6 [알고리즘 14-2]의 작동 과정에 대응되는 상태 공간 트리

- 가지치기 (pruning)
  - 백트래킹 과정에서 유망하지 않은 노드들은 가지쳐서(pruning) 더 이상 탐색을 하지 않는다

03. 한정 분기 (Branch and Bound)

한정 분기 전략 동기

◈ 되추적의 비효율

- 되추적의 탐색 과정에서 Valid가 적용되더라도 여전히 상당한 비율의 정점을 방문하지 않아도 됨을 발견
- 상태 공간 트리의 순회(Traverse)를 항상 깊이 우선으로 수행함

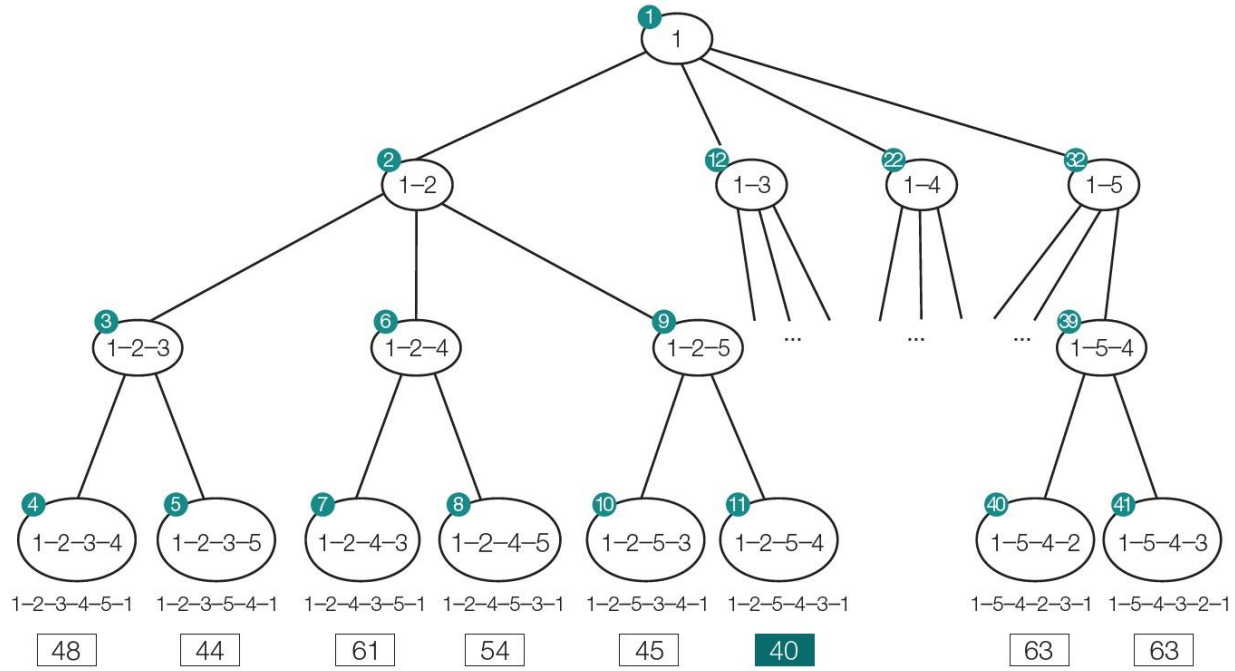
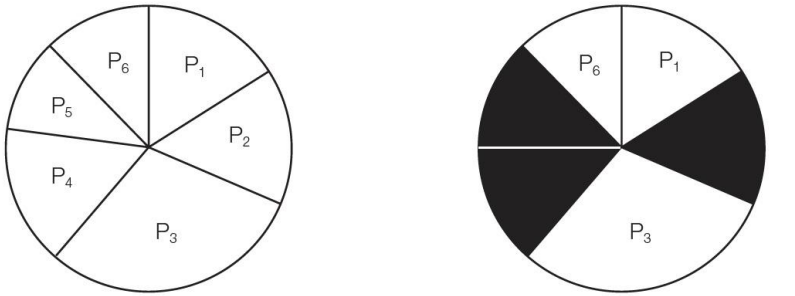


그림 14–3 [그림 14–2]의 TSP 예를 대상으로 한 사전식 탐색의 예

한정 분기 전략 동기

◈ 되추적의 비효율 해결책

- 각 노드를 방문할 때 마다 **한계치(bound)**를 계산한다.
  - 한계치(Bound): 해당 노드로부터 가지를 뻗어나가서(branch) 얻을 수 있는 해답치의 한계 값
- 각 노드의 **유망함(Promising)** 조건
  - 해당 노드에서의 한계치가 이전까지 찾은 최고 해답 값보다 더 좋음
  - 해당 노드가 유망하지 않다면 그 노드에서 **가지치기(Pruning)** 수행



(a) 어느 시점에 가능한 선택들 (b) 최적해를 포함하지 않아 제외된 선택

그림 14-7 가지치기의 원리를 설명하는 그림

- 탐색의 방법 및 순서를 변경
  - 가장 유망해 보이는 노드부터 검사 (**최고 우선 검색**)

0/1 Knapsack Problem

◈ 문제의 정형적 정의

- **입력**:
  - **보석** $(item)$**의 개수**: n, $S = \{item_1, item_2, \ldots, item_n\}$
  - $w_i = item_i$**의 무게**, $p_i = item_i$**의 가치**
  - $M =$ **배낭에 넣을 수 있는 총 무게**
- **문제**:
  - $\sum_{item_i \in A} w_i \leq M$**를 만족하면서** $\sum_{item_i \in A} p_i$ **가 최대가 되는** $A(\subset S)$**를 결정하는 문제**

0/1 Knapsack Problem + 한정 분기

◈ 깊이 우선 검색을 사용한 **한정 분기**(Branch and Bound) 전략

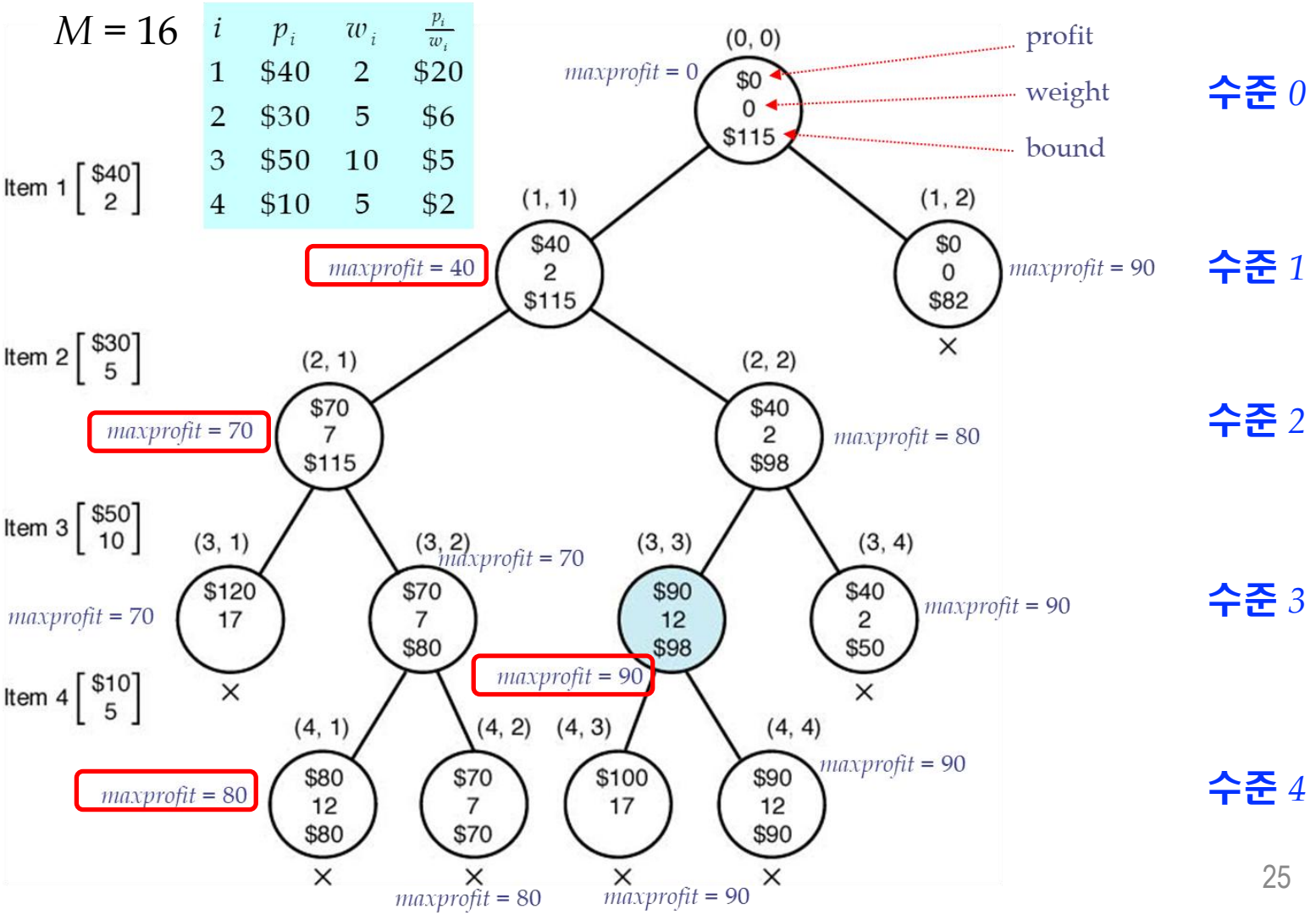
- w_i와 p_i를 각각 i번째 아이템의 무게와 값어치라고 하면, p_i/w_i 의 값이 큰 것부터 내림차순으로 아이템을 정렬
- $n = 4, M = 16$이고, 오른쪽 아이템 내역 가정

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2

- 상태 공간 트리 구축
  - 루트 노드 (트리 수준 0)에서
    왼쪽으로 가면 **첫번째 아이템을 배낭에 넣음**,
    오른쪽으로 가면 **첫번째 아이템을 배낭에 넣지 않음**
  - 동일한 방법으로 다음 노드 (트리 수준 1)에서
    왼쪽으로 가면 **두 번째 아이템을 배낭에 넣음**,
    오른쪽으로 가면 **두 번째 아이템을 배낭에 넣지 않음**
- 이런 식으로 계속하여 상태 공간 트리를 구축하면, 루트 노드로부터 리프 노드까지의 각각의 경로는 **해답 후보**가 된다.

0/1 Knapsack Problem + 한정 분기

◈ 깊이 우선 검색을 사용한 한정 분기(Branch and Bound) 전략



0/1 Knapsack Problem + 한정 분기

◈ 깊이 우선 검색을 사용한 **한정 분기**(Branch and Bound) 전략

- 다음 값들을 각 노드에 대해서 계산한다.
  - *profit*: 그 노드에 오기까지 넣었던 아이템 값어치의 합
  - *weight*: 그 노드에 오기까지 넣었던 아이템 무게의 합
  - *bound*(**향후 이익 한계치**): → 다음 페이지!
  - *maxprofit* : 지금까지 찾은 최고 값어치 해답 ← 전역 변수

- 이 문제는 **최적의 해를 찾는 문제**(optimization problem)이므로 검색이 완전히 끝나기 전에는 해답을 알 수가 없다.
- 따라서 검색을 하는 과정 동안 항상 그 때까지 찾은 최적의 해(maxprofit)를 기억해 두어야(메모리에 저장해 두어야) 한다.

0/1 Knapsack Problem + 한정 분기

◈ 깊이 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

- *bound*(향후 이익 한계치)
  - 임의의 노드가 수준 i에 있다고 하고, 자손 노드 중 수준 k $(k > i)$에 있는 노드에서 총 무게가 M을 넘는다고 하자. 그러면, 다음과 같이 *bound*를 구할 수 있다.

i번째 아이템까지 고려했을 때의 무게와
i+1번째 부터 k-1 번째 아이템 까지의 무게의 합

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

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

빈틈없이 배낭 채우기
문제에서의 개념 사용

k 번째 아이템 을 부분적으로나마
넣을 수 있다고 가정할 때의 이익

k-1 번째 아이템 까지의 이익

[주의] Root 노드: 수준 0

0/1 Knapsack Problem + 한정 분기

◈ 깊이 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

- 루트 노드 설정: *maxprofit* := $0; *profit* := $0; *weight* := 0
- 깊이 우선 순위로 각 노드를 방문하여 다음을 수행한다:
  - 그 노드의 *profit*과 *weight*를 계산한다.
  - 그 노드의 *bound*를 계산한다.
  - 현재 노드의 *profit*이 *maxprofit*보다 크면 *maxprofit* 값을 갱신한다.
  - (*weight* < *M*) && (*bound* > *maxprofit*)이면, 유망하다고 보고 가지를 뻗어(Branch)나간다.
    - ➢ 즉, 무게가 초과하지 않았고, 향후 가치가 최대 가치보다 큼
  - 그렇지 않으면 유망하지 않다고 보고, 되추적(Backtraking)한다.

- 상기 과정을 깊이 우선 순회 방법으로 수행한다.
  - 임의의 노드에서 가지치기가 되므로, 모든 노드를 방문하지는 않음

Bound 값 구하기 예

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

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

◈ 깊이 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

- (0,0) 에서 bound 값 구하기

$M = 16$

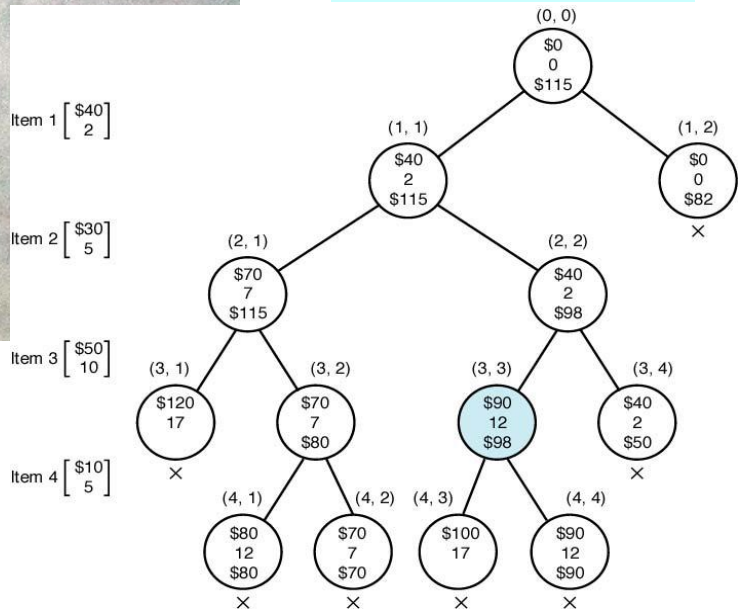
i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2

그 한계를 계산한다. $2 + 5 + 10 = 17$이고, $17 > 16$(W의 값)이므로, 3번째 아이템을 취하면 무게의 합이 W를 넘는다. 따라서 $k = 3$이고, *totweight*와 *bound*는 다음과 같이 계산한다.

$$totweight = weight + \sum_{j=0+1}^{3-1} w_j = 0 + 2 + 5 = 7$$

$$bound = profit + \sum_{j=0+1}^{3-1} p_j + (W - totweight) \times \frac{p_3}{w_3}$$

$$= \$0 + \$40 + \$30 + (16 - 7) \times \frac{\$50}{10} = \$115$$



Bound 값 구하기 예

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

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

◈ 깊이 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

– (1,1) 에서 bound 값 구하기

$M = 16$

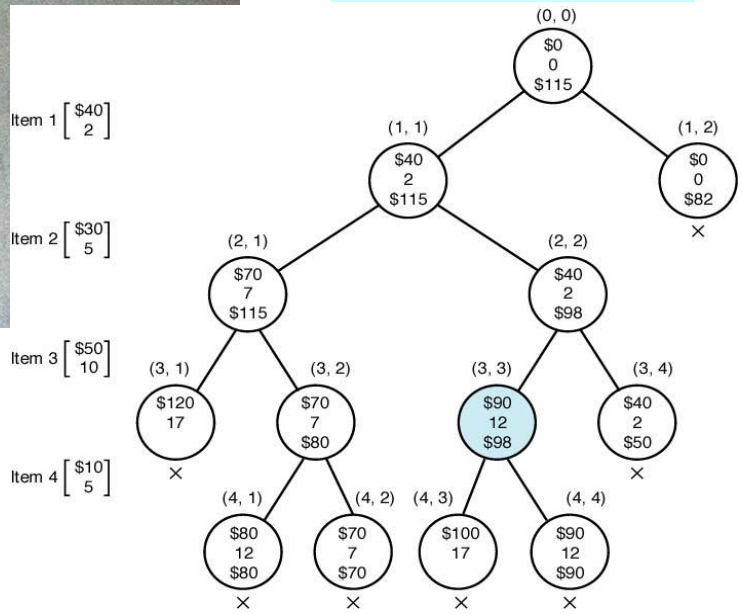
i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2

그 한계를 계산한다. $2 + 5 + 10 = 17$ 이고, $17 > 16$(W의 값)이므로, 3번째 아이템을 취하면 무게의 합이 W를 넘는다. 따라서 $k = 3$ 이고, *totweight* 와 *bound* 는 다음과 같이 계산한다.

$$totweight = weight + \sum_{j=1+1}^{3-1} w_j = 2 + 5 = 7$$

$$bound = profit + \sum_{j=1+1}^{3-1} p_j + (W - totweight) \times \frac{p_3}{w_3}$$

$$= \$40 + \$30 + (16 - 7) \times \frac{\$50}{10} = \$115$$



Bound 값 구하기 예

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

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

◈ 깊이 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

– (2,1) 에서 bound 값 구하기

- $k = 3, i = 2$

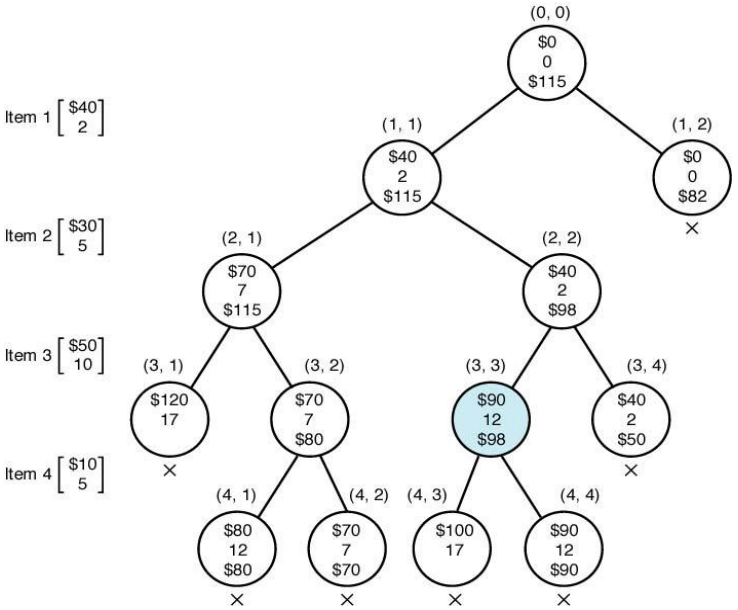
그 한계를 계산한다.

$$totweight = weight + \sum_{j=2+1}^{3-1} w_j = 7$$

$$bound = \$70 + (16 - 7) \times \frac{\$50}{10} = \$115$$

$M = 16$

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2



Bound 값 구하기 예

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

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

◈ 깊이 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

- (2,1) 에서 bound 값 구하기
  - $k = 3, i = 2$

그 한계를 계산한다.

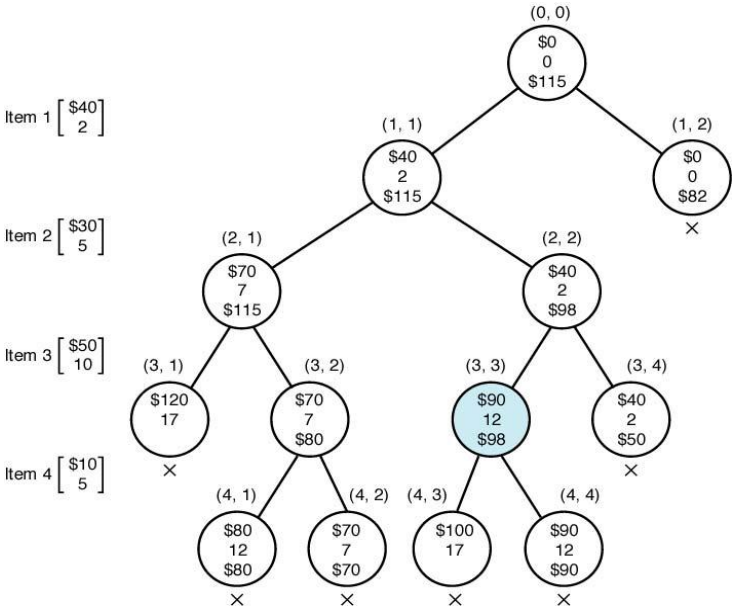
$$totweight = weight + \sum_{j=2+1}^{3-1} w_j = 7$$

$$bound = \$70 + (16 - 7) \times \frac{\$50}{10} = \$115$$

- (3,1) 에서는 bound를 구할 필요가 없다.
  - Why? weight=17이 W=16을 초과하였으므로

$M = 16$

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2



Bound 값 구하기 예

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

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

◈ 깊이 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

- (3,2) 에서 bound 값 구하기

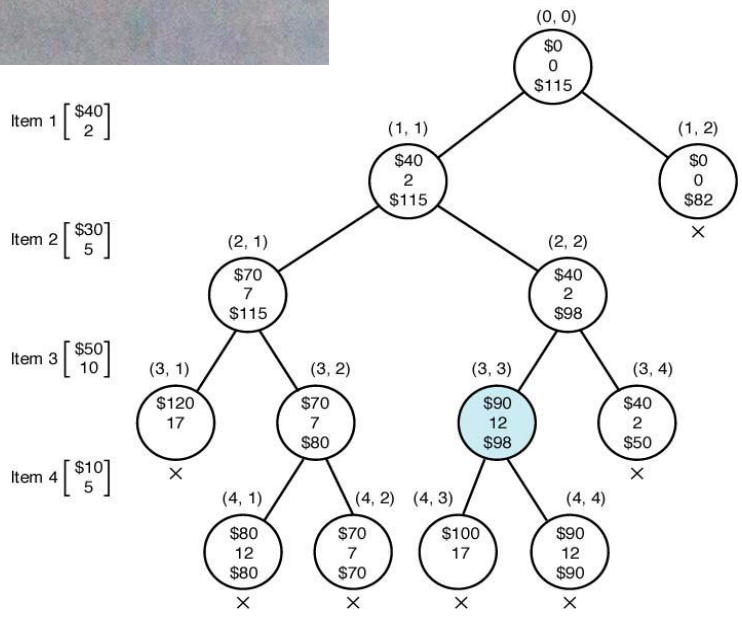
$M = 16$

(c) 그 한계를 계산한다. 4번째 아이템의 무게를 더해도 합이 W를 초과하지 않고, 아이템이 4개만 있다. 따라서 $k = 5$이고, $bound$는 다음과 같이 계산한다.

$$bound = profit + \sum_{j=3+1}^{5-1} p_j = \$70 + \$10 = \$80$$

(d) 무게 7은 W의 값인 16보다 작고, 한계 \$80은 $maxprofit$의 값인 \$70보다 크므로, 이 마디는 유망하다고 결정한다.

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2



Bound 값 구하기 예

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

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

◈ 깊이 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

- …
- (1,2) 에서 bound 값 구하기
  - $k = 4, i = 1$

$$totweight = weight + \sum_{j=2}^{4-1} w_j = 0 + w_2 + w_3 = 15$$

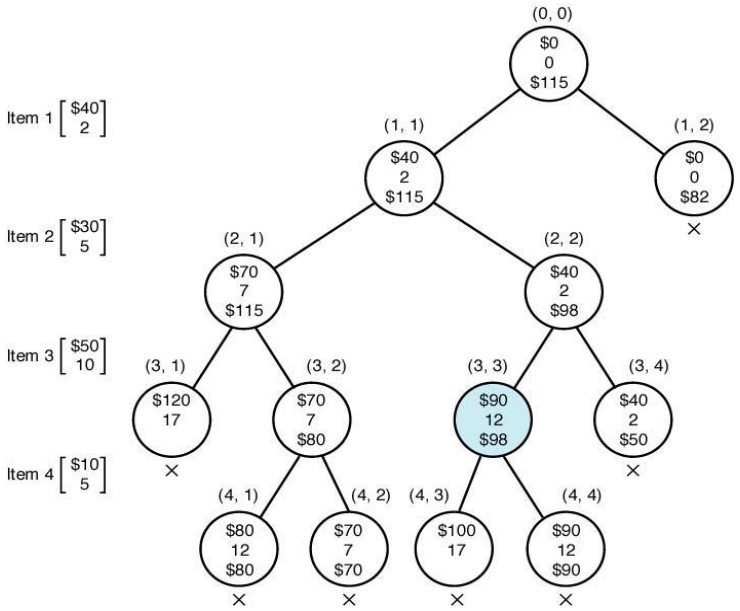
$$bound = \left(profit + \sum_{j=2}^{4-1} p_j \right) + (M - totweight) \times \frac{p_4}{w_4}$$

$$= (0 + p_2 + p_3) + (M - totweight) \times \$2$$

$$= \$80 + (16 - 15) \times \$2 = \$82$$

$M = 16$

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2



본 예제의 경우: 점검한 노드의 총 개수는 13개

0/1 Knapsack Problem + 한정 분기

◈ 너비 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

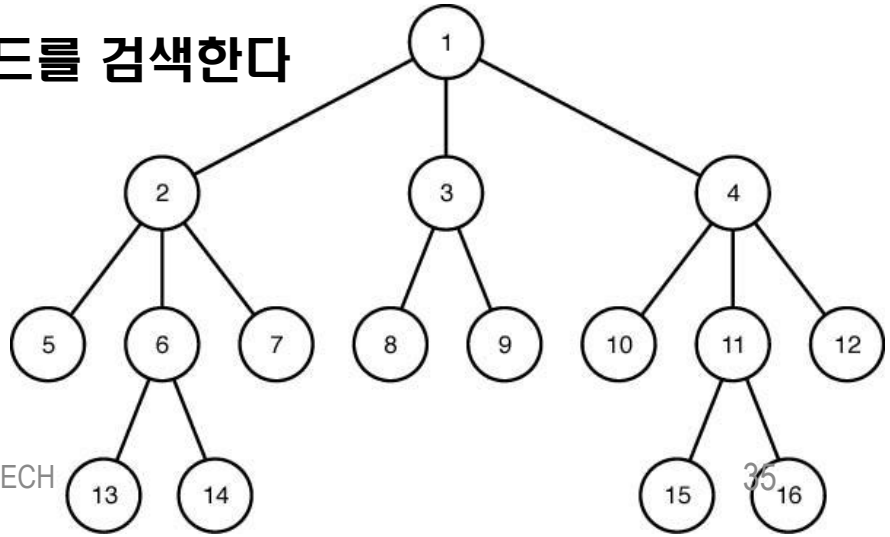
- 기존 깊이 우선 검색(DFS) 기반 알고리즘은 분기한정의 특성을 제대로 활용하고 있지 못하다.
- 그렇다면…너비 우선 검색(BFS)을 수정하여 구현해본다.
- 너비우선검색(Breadth-First Search)순서

  (1) 루트 노드를 먼저 검색한다.

  (2) 다음에 수준 1에 있는 모든 노드를 검색한다.
  (왼쪽에서 오른쪽으로)

  (3) 다음에 수준 2에 있는 모든 노드를 검색한다
  (왼쪽에서 오른쪽으로)

  (4) …

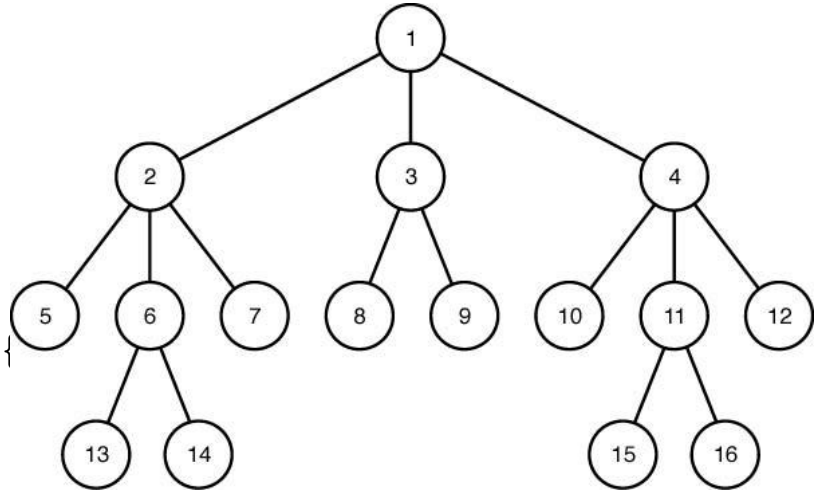


0/1 Knapsack Problem + 한정 분기

◈ 너비 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

- 큐(queue)를 사용한 너비 우선 검색

```
void 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);
      }
   }
}
```



0/1 Knapsack Problem + 한정 분기

◈ 너비 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

– 0/1 Knapsack Problem 의사 코드

```
void breadth_first_branch_and_bound2 (state_space_tree T) {
   queue_of_node Q;
   node u, v;
   number maxprofit;
   initialize(Q);                  // Q는 빈 대기열로 초기화
   v = root of T;                  // 루트 노드를 방문
   enqueue(Q,v);
   maxprofit = profit(v);
   while(!empty(Q)) {
      dequeue(Q,v);
      if(bound(v) > maxprofit) {   //노드가 여전히 유망한 지 점검
         for(each child u of v) {  // 각 자식 노드를 방문
            if(profit(u) > maxprofit && weight < M)
               maxprofit = profit(u); // 더 좋은 해답 발견
            if(bound(u) > maxprofit && weight < M)
               enqueue(Q,u);          // 유망하다면 enqueue
         }
      }
   }
}
```

0/1 Knapsack Problem + 한정 분기

◈ 너비 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

– 개선된 너비 우선 검색: Queue에서의 진행모습 (1/4)

Queue

maxprofit=0

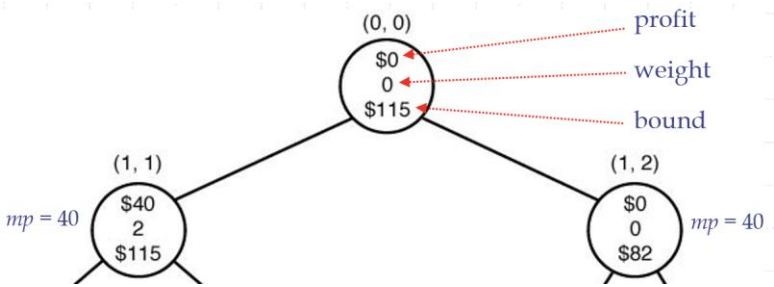
(0,0)
$0,0
$115

maxprofit=40

(1,1)	(1,2)
$40,2	$0,0
$115	$82

V

(0,0)
$0,0
$115



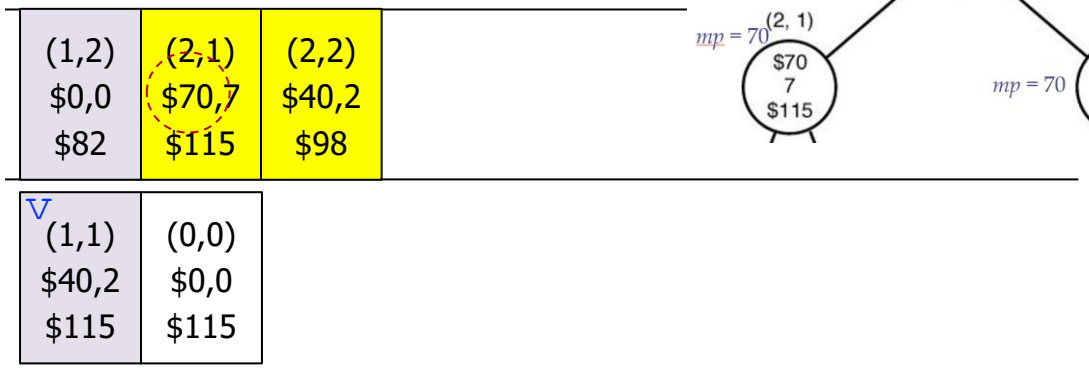

```
while(!empty(Q)) {
   dequeue(Q,v);
   if(bound(v) > maxprofit) {   //노드가 여전히 유망한 지 점검
      for(each child u of v) {   // 각 자식 노드를 방문
         if(profit(u) > maxprofit && weight < M)
            maxprofit = profit(u); // 더 좋은 해답 발견
         if(bound(u) > maxprofit && weight < M)
            enqueue(Q,u);          // 유망하다면 enqueue
      }
   }
}
```

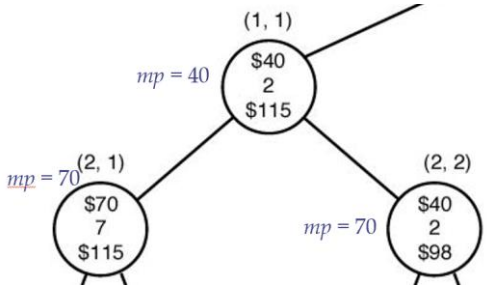
0/1 Knapsack Problem + 한정 분기

◈ 너비 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

– 개선된 너비 우선 검색: Queue에서의 진행모습 (1/4)

maxprofit=70



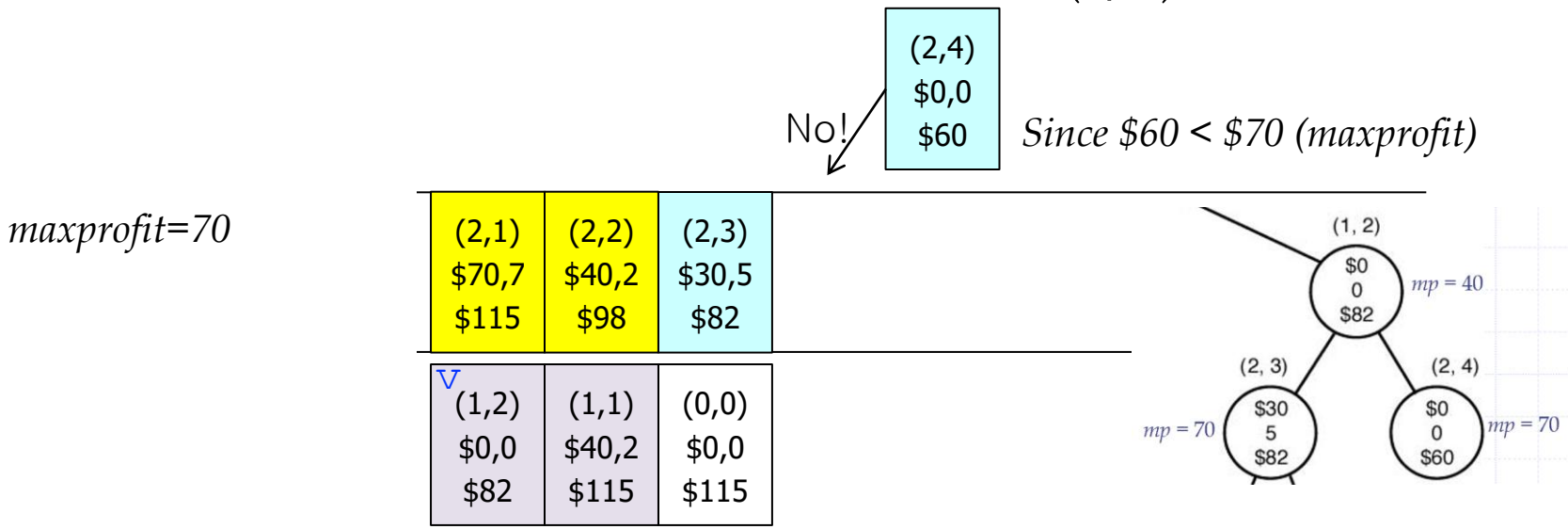



```
while(!empty(Q)) {
   dequeue(Q,v);
   if(bound(v) > maxprofit) {  //노드가 여전히 유망한 지 점검
      for(each child u of v) {  // 각 자식 노드를 방문
         if(profit(u) > maxprofit && weight < M)
            maxprofit = profit(u); // 더 좋은 해답 발견
         if(bound(u) > maxprofit && weight < M)
            enqueue(Q,u);          // 유망하다면 enqueue
      }
   }
}
```

0/1 Knapsack Problem + 한정 분기

◈ 너비 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

– 개선된 너비 우선 검색: Queue에서의 진행모습 (2/4)

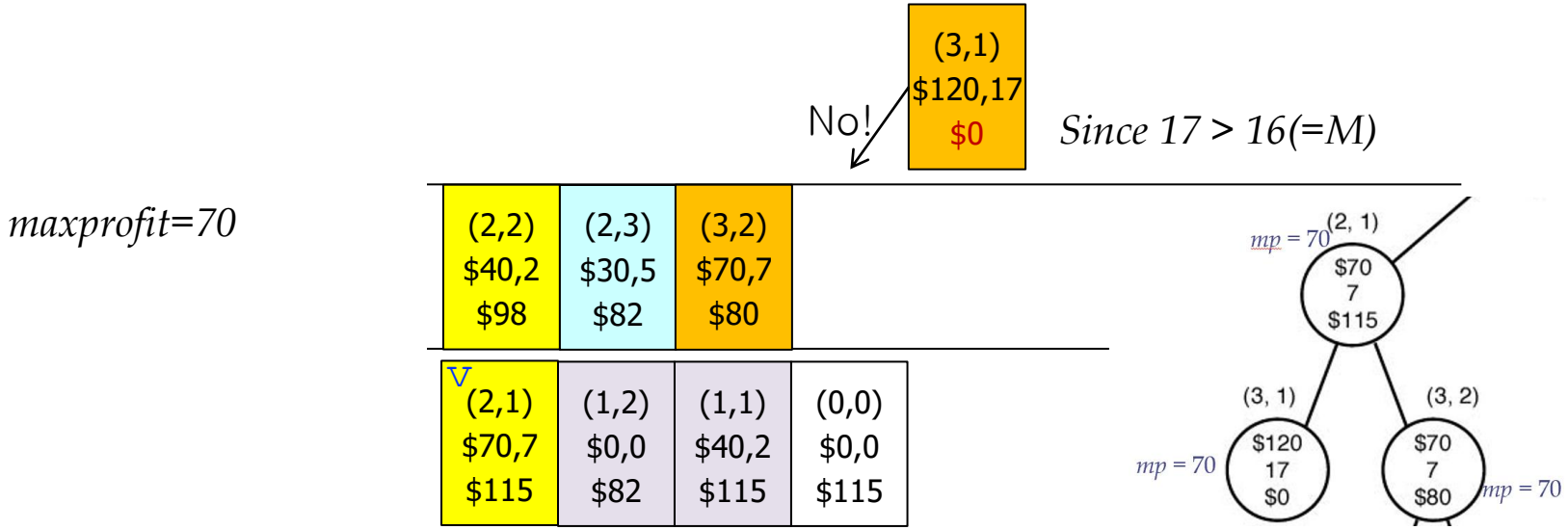



```
while(!empty(Q)) {
   dequeue(Q,v);
   if(bound(v) > maxprofit) {  //노드가 여전히 유망한 지 점검
      for(each child u of v) {  // 각 자식 노드를 방문
         if(profit(u) > maxprofit && weight < M)
            maxprofit = profit(u); // 더 좋은 해답 발견
         if(bound(u) > maxprofit && weight < M)
            enqueue(Q,u);          // 유망하다면 enqueue
      }
   }
}
```

0/1 Knapsack Problem + 한정 분기

◈ 너비 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

– 개선된 너비 우선 검색: Queue에서의 진행모습 (2/4)

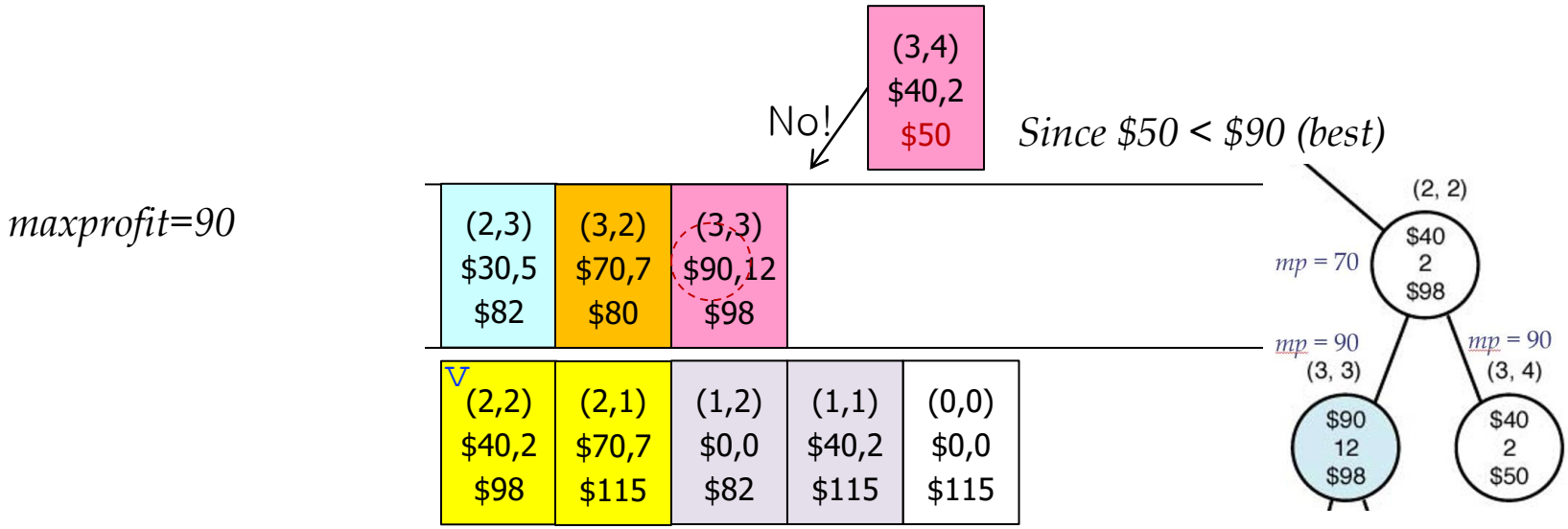



```
while(!empty(Q)) {
    dequeue(Q,v);
    if(bound(v) > maxprofit) {  //노드가 여전히 유망한 지 점검
        for(each child u of v) {  // 각 자식 노드를 방문
            if(profit(u) > maxprofit && weight < M)
                maxprofit = profit(u); // 더 좋은 해답 발견
            if(bound(u) > maxprofit && weight < M)
                enqueue(Q,u);          // 유망하다면 enqueue
        }
    }
}
```

0/1 Knapsack Problem + 한정 분기

◈ 너비 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

– 개선된 너비 우선 검색: Queue에서의 진행모습 (3/4)




```
while(!empty(Q)) {
   dequeue(Q,v);
   if(bound(v) > maxprofit) {  //노드가 여전히 유망한 지 점검
      for(each child u of v) {  // 각 자식 노드를 방문
         if(profit(u) > maxprofit && weight < M)
            maxprofit = profit(u); // 더 좋은 해답 발견
         if(bound(u) > maxprofit && weight < M)
            enqueue(Q,u);          // 유망하다면 enqueue
      }
   }
}
```

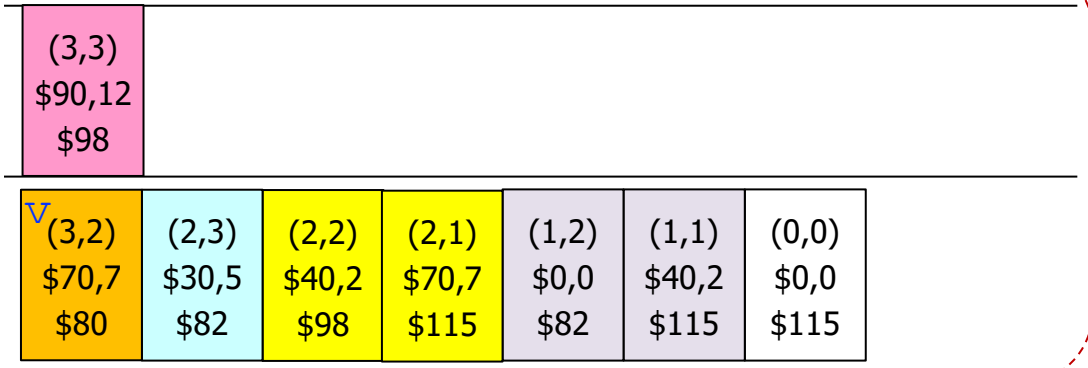
0/1 Knapsack Problem + 한정 분기

◈ 너비 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

– 개선된 너비 우선 검색: Queue에서의 진행모습 (3/4)

maxprofit=90

No Branching Since
(2, 3): $82 < $90 (maxprofit)
(3, 2): $80 < $90 (maxprofit)

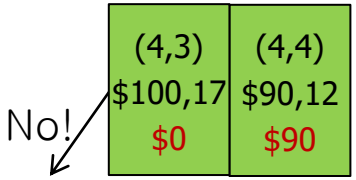



```
while(!empty(Q)) {
   dequeue(Q,v);
   if(bound(v) > maxprofit) {  //노드가 여전히 유망한 지 점검
      for(each child u of v) {  // 각 자식 노드를 방문
         if(profit(u) > maxprofit && weight < M)
            maxprofit = profit(u); // 더 좋은 해답 발견
         if(bound(u) > maxprofit && weight < M)
            enqueue(Q,u);          // 유망하다면 enqueue
      }
   }
}
```

0/1 Knapsack Problem + 한정 분기

◈ 너비 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

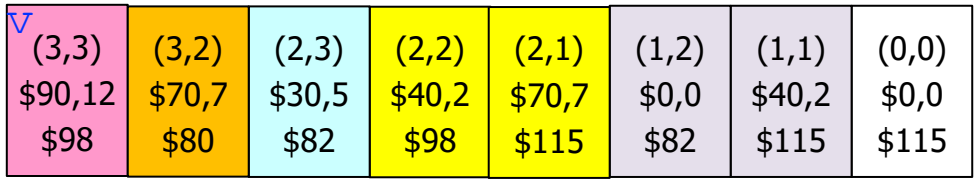
– 개선된 너비 우선 검색: Queue에서의 진행모습 (4/4)

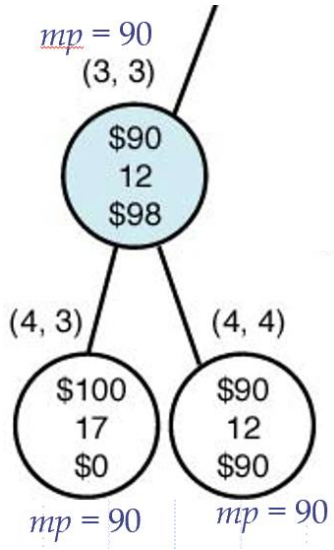


Since
17 > 16(=W)
$90 = $90 (best)

maxprofit=90

Empty!





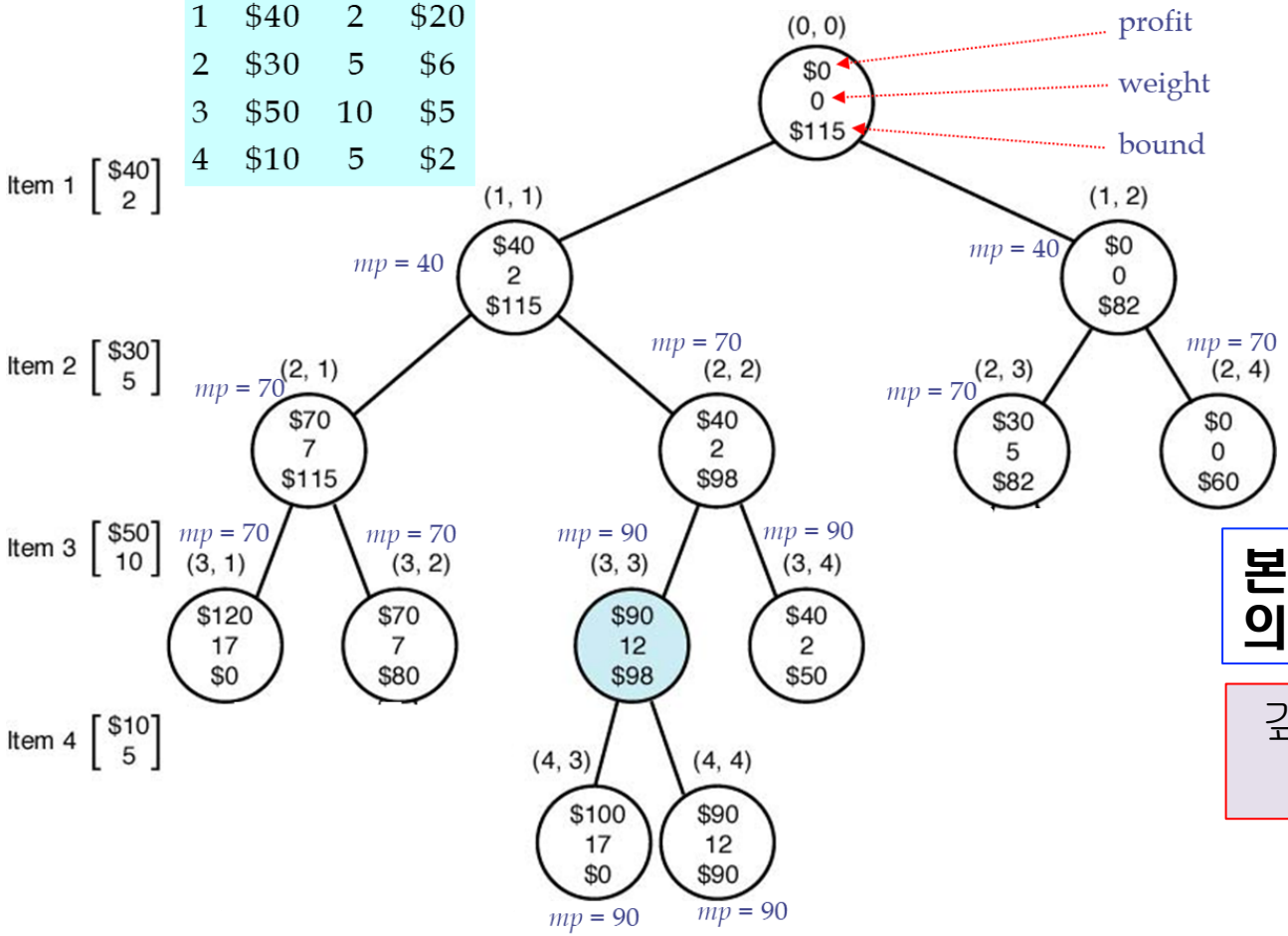

```
while(!empty(Q)) {
    dequeue(Q,v);
    if(bound(v) > maxprofit) {   //노드가 여전히 유망한 지 점검
        for(each child u of v) {   // 각 자식 노드를 방문
            if(profit(u) > maxprofit && weight < M)
                maxprofit = profit(u); // 더 좋은 해답 발견
            if(bound(u) > maxprofit && weight < M)
                enqueue(Q,u);          // 유망하다면 enqueue
        }
    }
}
```

0/1 Knapsack Problem + 한정 분기

◈ 너비 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

- 개선된 너비 우선 검색: 종합적인 상태 공간 트리 탐색 모습

i	p_i	w_i	$\frac{p_i}{w_i}$
1	$40	2	$20
2	$30	5	$6
3	$50	10	$5
4	$10	5	$2



본 예제의 경우: 점검한 노드의 총 개수는 13개

깊이 우선 검색 기반 해결책 (13개 노드 검색)과 동일

0/1 Knapsack Problem + 한정 분기

◈ 최고 우선 검색 (Best-First Search)

- 최적의 해답에 더 빨리 도달하기 위한 전략
  - 1. 주어진 노드의 모든 자식 노드를 검색한 후,
  - 2. 유망한 노드인지를 살펴보고,
  - 3. 그 중에서 가장 좋은(최고의) 한계치(bound)를 가진 노드를 우선적으로 확장하여 평가 후 큐에 삽입
    - ➢ 최고의 한계를 가진 노드를 우선적으로 선택하기 위해서 우선순위 큐(Priority Queue) 사용 ← 힙(heap)을 사용하여 구현
      - » Top Priority: bound 값이 가장 큰 노드

- 최고 우선 검색을 활용한 한정 분기 알고리즘을 최고 우선 검색 기반 분기 한정 가지치기 (Best-first search with branch-and-bound pruning) 이라 한다.

0/1 Knapsack Problem + 한정 분기

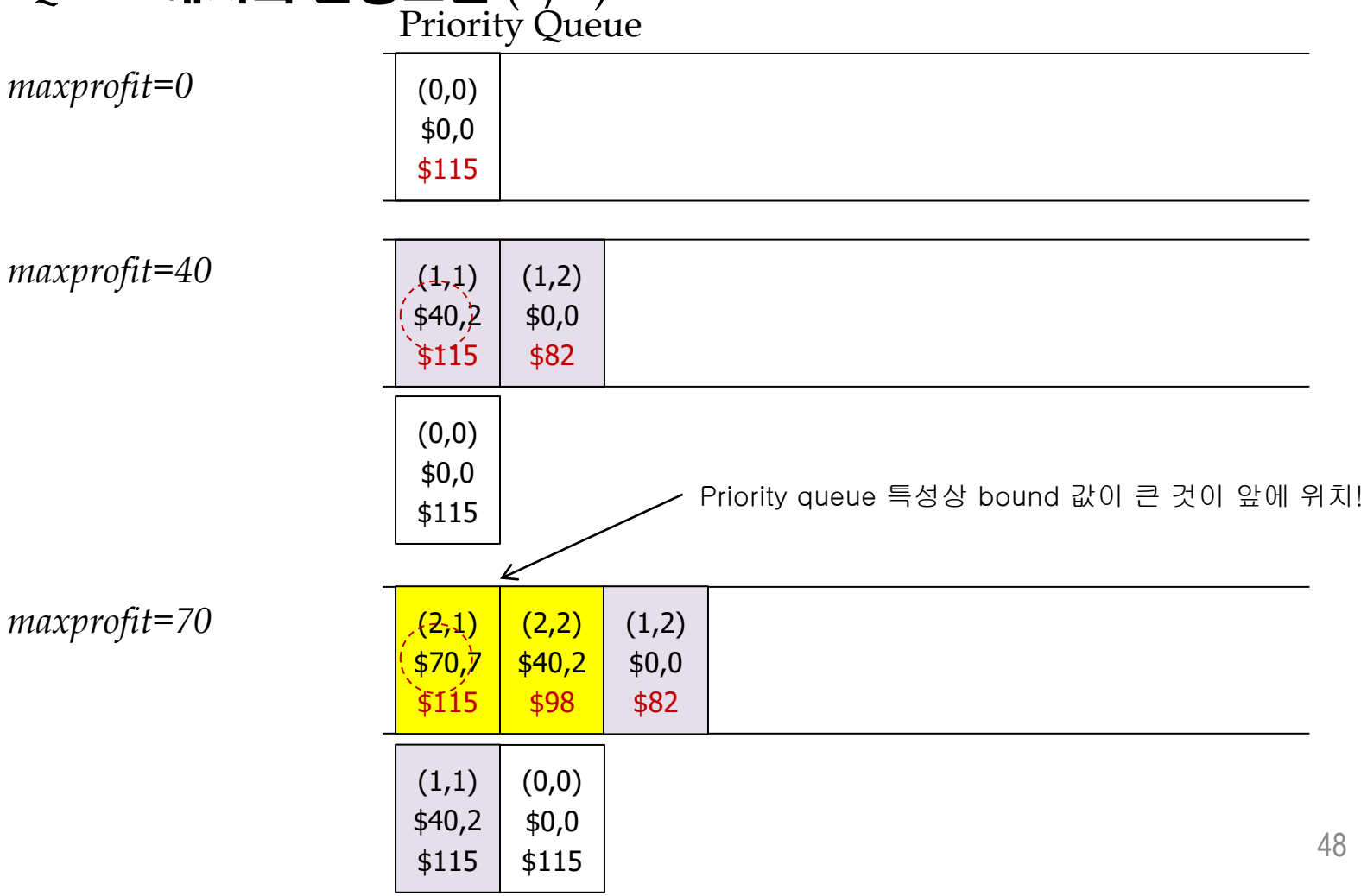
◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

```
void best_first_branch_and_bound(state_space_tree T){
  priority_queue_of_node PQ;
   node u, v;
   number maxprofit;
   initialize(PQ);      //PQ를 빈 대기열로 초기화
   v = root of T;
   maxprofit = profit(v);
   enqueue(PQ,v);
   while(!empty(PQ)) { //최고 한계 값을 가진 노드를 제거
      dequeue(PQ,v);
     if(bound(v) > maxprofit) //노드가 여전히 유망한 지 점검
         for(each child u of v) {
            if(profit(u) > maxprofit && weight < M)
                maxprofit = profit(u); // 더 좋은 해답 발견
            if(bound(u) > maxprofit && weight < M)
                insert(PQ,u);          // 유망하다면 enqueue
         }
   }
}
```

0/1 Knapsack Problem + 한정 분기

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

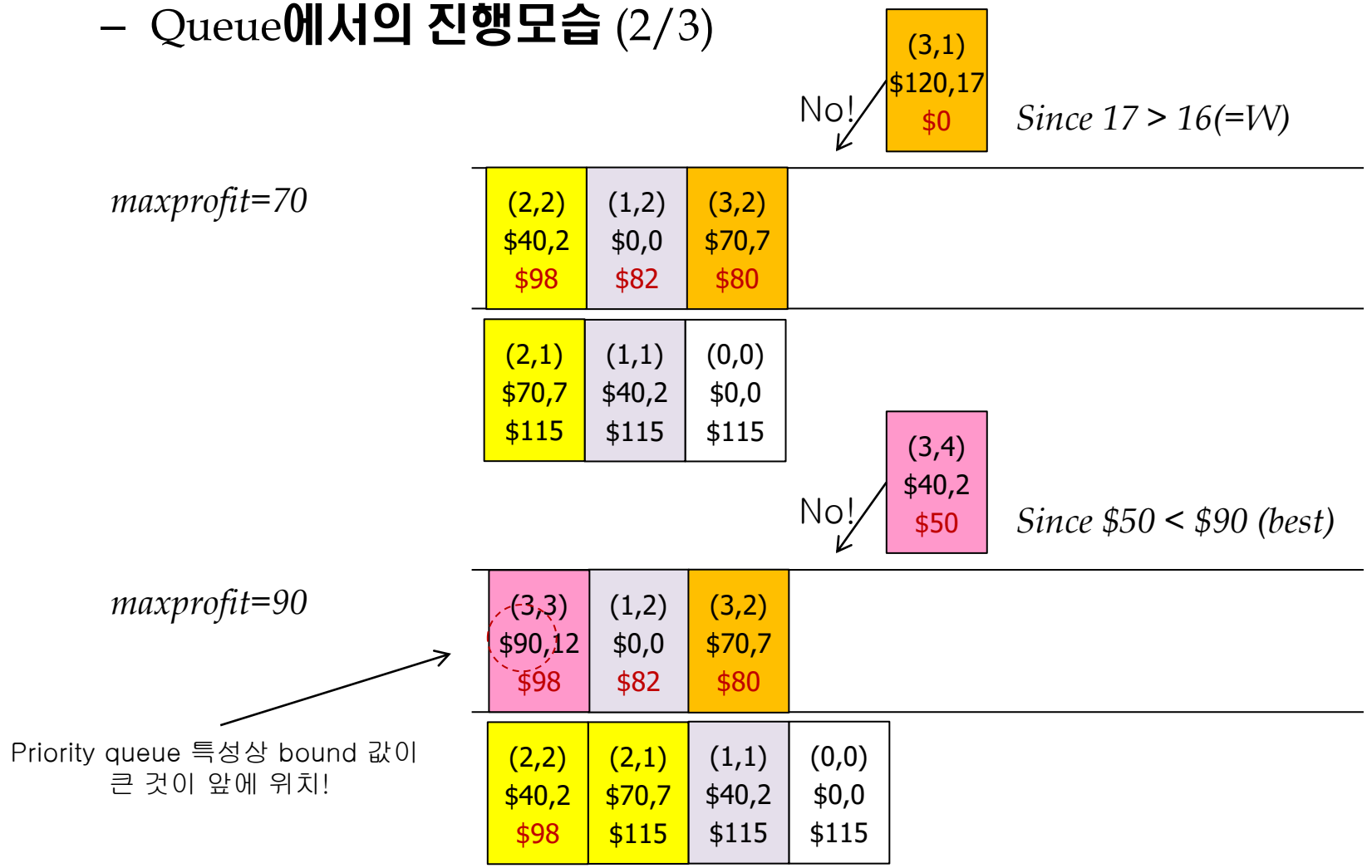
– Queue에서의 진행모습 (1/3)



0/1 Knapsack Problem + 한정 분기

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

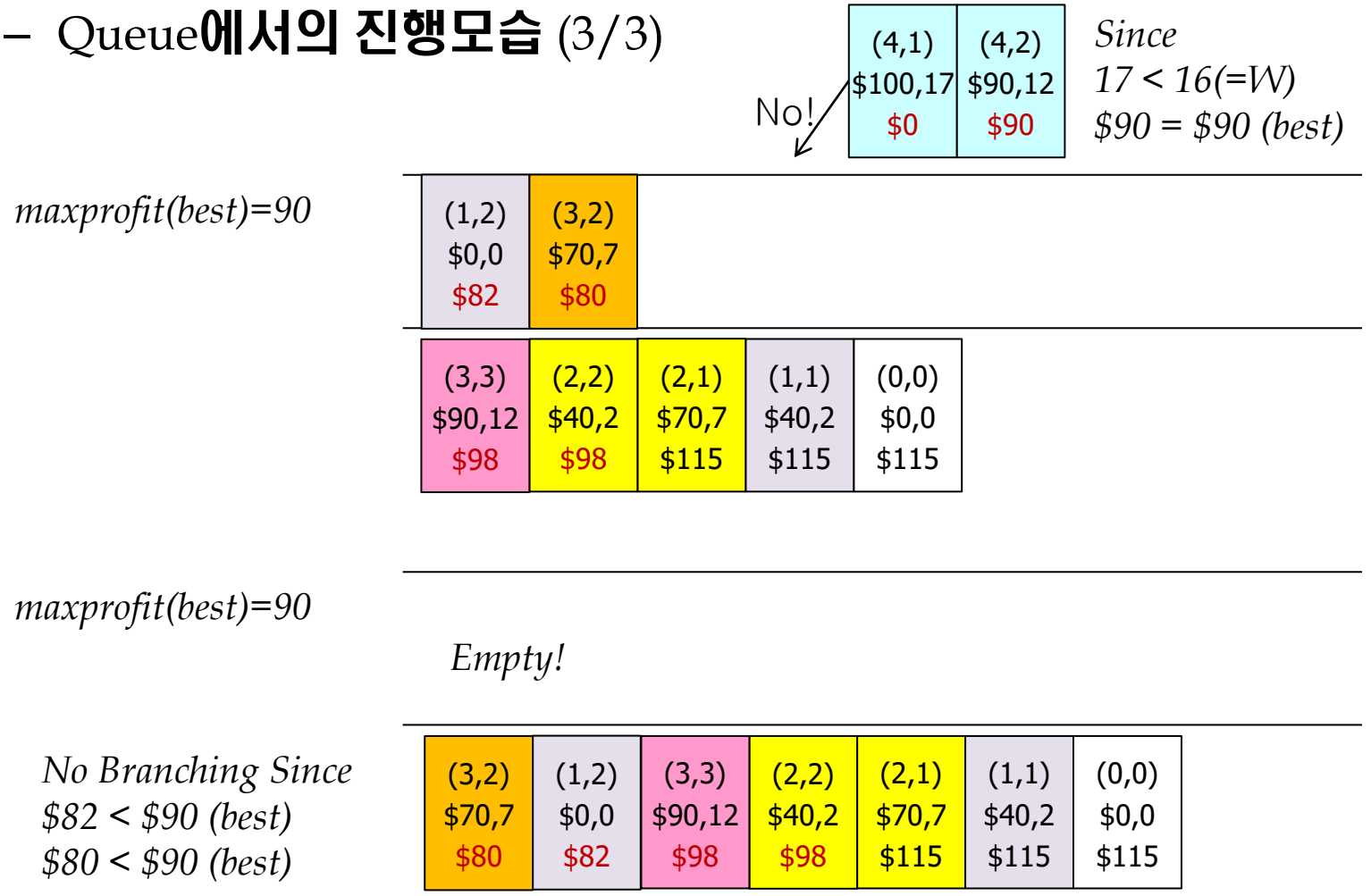
– Queue에서의 진행모습 (2/3)



0/1 Knapsack Problem + 한정 분기

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

– Queue에서의 진행모습 (3/3)

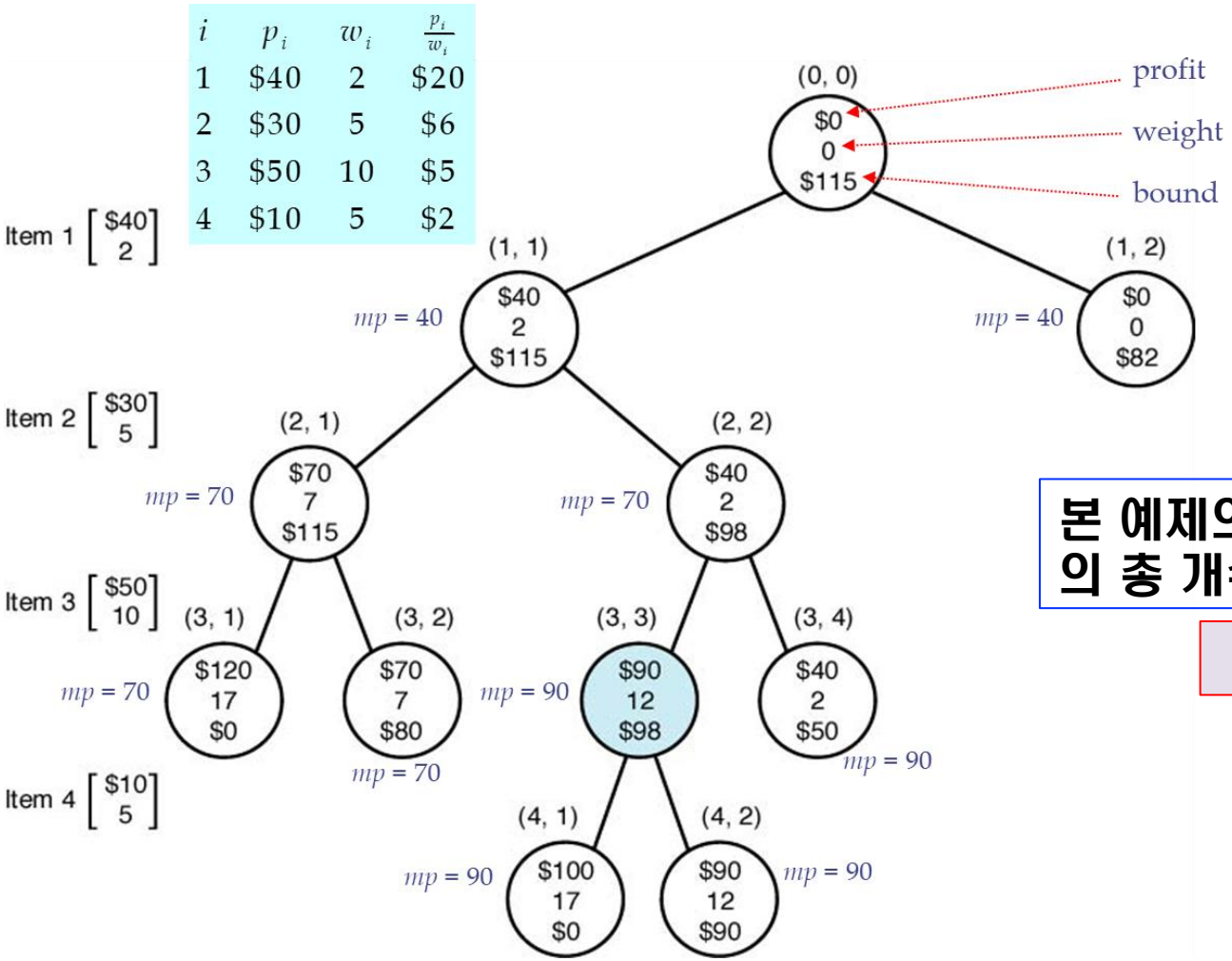


0/1 Knapsack Problem + 한정 분기

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

- 종합적인 상태 공간 트리 탐색 모습

i	p_i	w_i	$\frac{p_i}{w_i}$
1	$40	2	$20
2	$30	5	$6
3	$50	10	$5
4	$10	5	$2



본 예제의 경우: 점검한 노드의 총 개수는 11개

가장 우수!!!

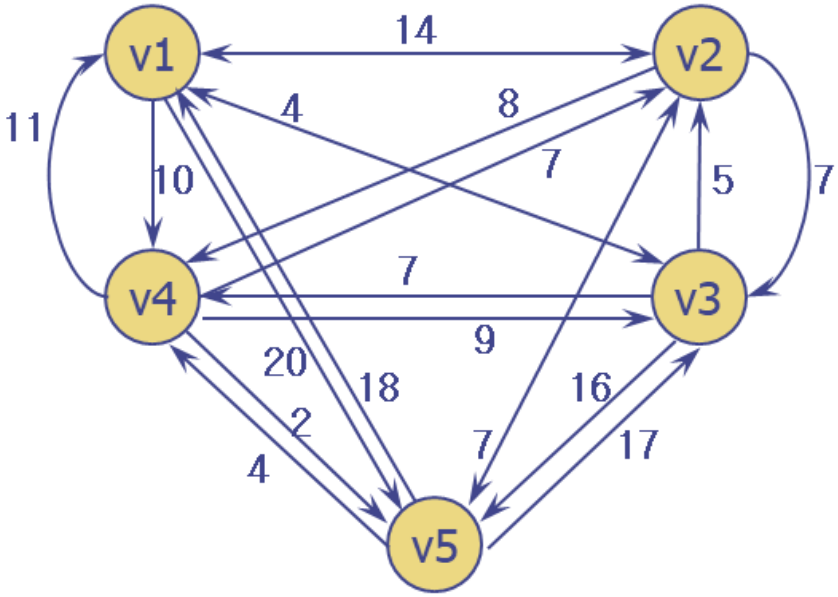
Traveling Salesman Problem

◈ 문제 정의

- 외판원이 자신의 집이 위치하고 있는 도시에서 출발하여 다른 도시들을 "각각 한번씩만 방문"하고, "다시 자기 도시로 돌아오는" 가장 짧은 일주여행경로(tour)를 결정하는 문제
- 가장 최소 비용의 해밀토니안 경로
- 일반적으로, 이 문제는 음이 아닌 가중치가 있는, 방향성 그래프를 대상으로 함

◈ 본 자료에서 사용하는 그래프

$$\begin{bmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{bmatrix}$$



Traveling Salesman Problem

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

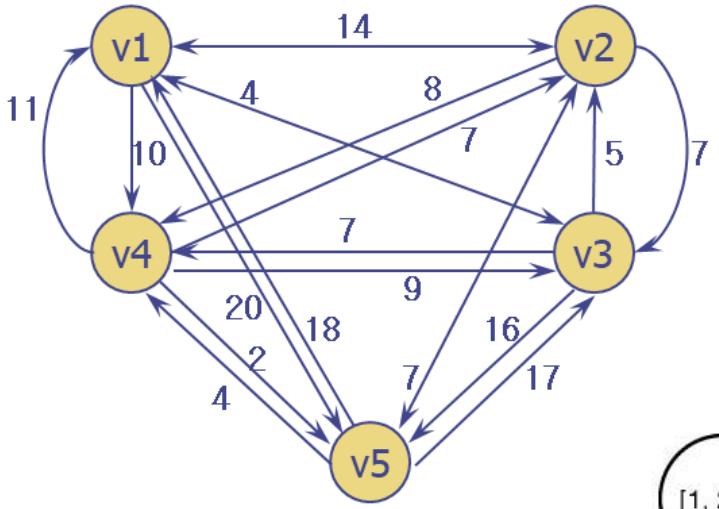
- 각 노드는 출발노드로부터의 일주여행경로를 나타냄
  - 루트노드의 여행경로는 [1]이 되고, 루트노드에서 뻗어 나가는 수준 1에 있는 여행경로는 각각 [1,2], [1,3], …, [1,5]가 된다.
  - 노드 [1,2]에서 뻗어 나가는 수준 2에 있는 노드들의 여행경로는 각각 [1,2,3], [1,2,4], [1,2,5]가 된다.
  - 이런 식으로 뻗어 나가서 단말노드에 도달하게 되면 완전한 일주여행경로를 가지게 된다.

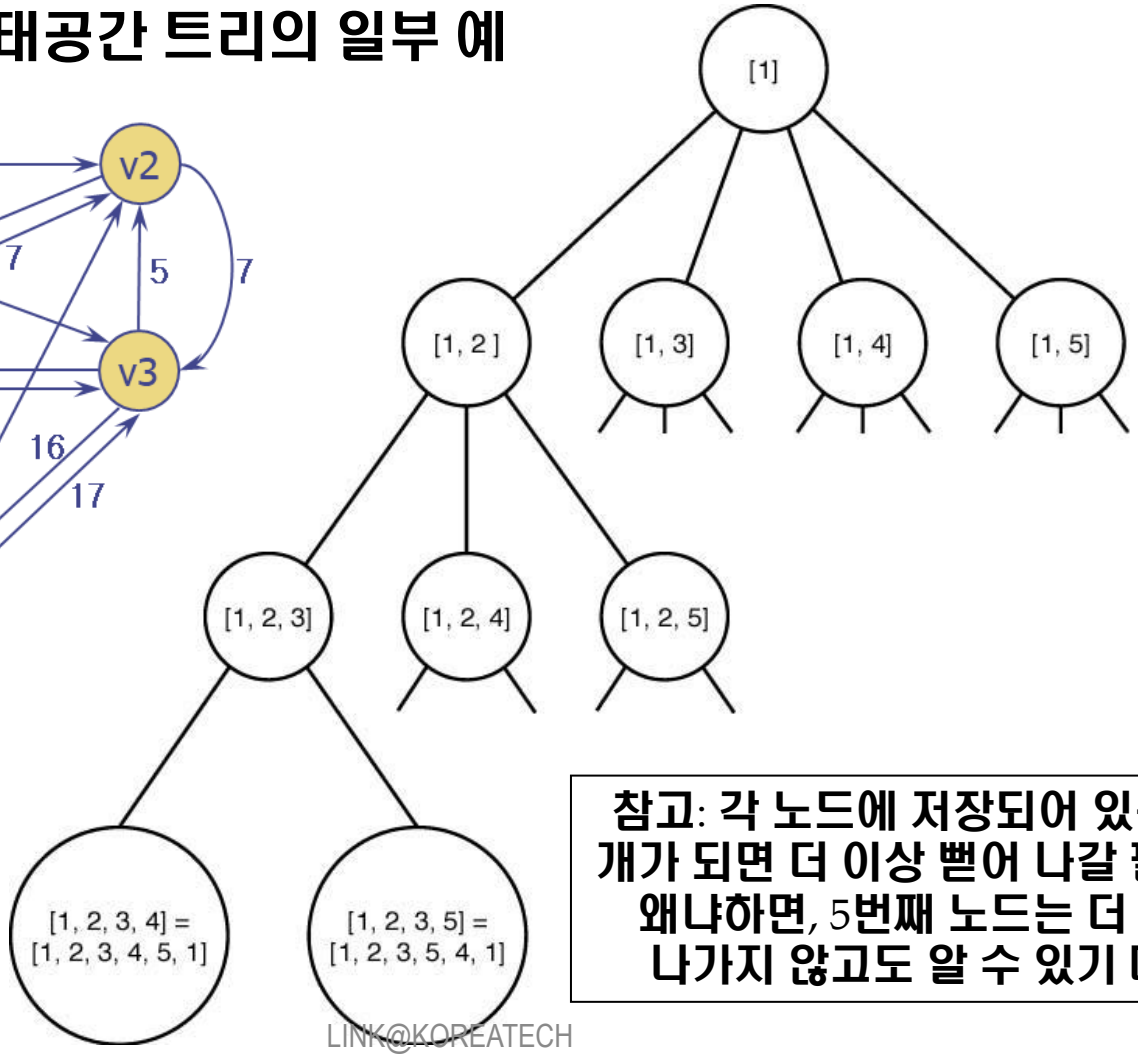
- 최적일주여행경로를 구하는 방법
  - 단말노드에 있는 일주여행경로를 모두 검사하여 그 중에서 가장 비용이 낮은 일주여행경로를 찾으면 된다. (무작정 알고리즘: Brute Force 전략)

Traveling Salesman Problem

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

– 구축된 상태공간 트리의 일부 예





참고: 각 노드에 저장되어 있는 노드가 4개가 되면 더 이상 뻗어 나갈 필요가 없다. 왜냐하면, 5번째 노드는 더 이상 뻗어 나가지 않고도 알 수 있기 때문이다.

Traveling Salesman Problem

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

- 분기한정 가지치기로 최고우선 검색을 사용하기 위해서 각 노드의 **한계치**(bound)를 구할 수 있어야 한다.
  - 이 문제에서는 주어진 노드에서 뻗어 나가서 얻을 수 있는 여행경로 길이의 **하한(최소치**, lower bound)을 구하여 한계치로 한다.

- 각 노드를 방문할 때 그 노드가 **유망** (Promising)할 조건
  - ➢ **한계치 < (현재까지 알아낸) 최소여행경로 길이**

- 한계치가 최소 여행경로 길이보다 큰 경우는 Pruning을 수행하여 검색 공간을 줄인다.

Traveling Salesman Problem

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

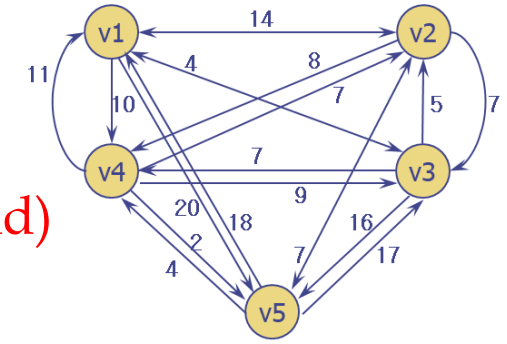
- 루트노드의 한계치(Bound, 하한) 구하기
- 전략: 어떤 일주여행경로라도 각 정점을 최소한 한번은 방문 후 떠나야 하므로, 각 정점을 떠나는 이음선 중 최소값의 합이 하한이 된다.
  - v_1 → min(14, 4, 10, 20) = 4
  - v_2 → min(14, 7, 8, 7) = 7
  - v_3 → min(4, 5, 7, 16) = 4
  - v_4 → min(11, 7, 9, 2) = 2
  - v_5 → min(18, 7, 17, 4) = 4

$$\begin{bmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{bmatrix}$$

- 따라서, 일주여행경로 길이의 하한은 21(= 4+7+4+2+4)이 된다.
- 주의할 점은 "이 길이의 일주여행경로가 있다는 말이 아니라, 이론적으로 이보다 더 짧은 일주여행경로가 있을 수 없다" 는 것이다. 그래서 하한(lower bound)이라는 말을 사용한다.

Traveling Salesman Problem

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략



– 루트노드가 아닌 다른 노드의 한계치 구하기

- 주어진 총 정점: V=[v_1, …, v_k, v_{k+1}, …, v_n]
- [v_1, …, v_k] 여행경로를 가진 노드의 한계치 (bound)

  = [v_1, …, v_k] 경로 상의 총 거리[a]

  + v_k 에서 $V - [v_1, \ldots, v_k]$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치[b]

  + v_{k+1} 에서 $V - [v_2, \ldots, v_k, v_{k+1}]$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치[c]

  + v_{k+2} 에서 $V - [v_2, \ldots, v_k, v_{k+2}]$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치[c]

  + …

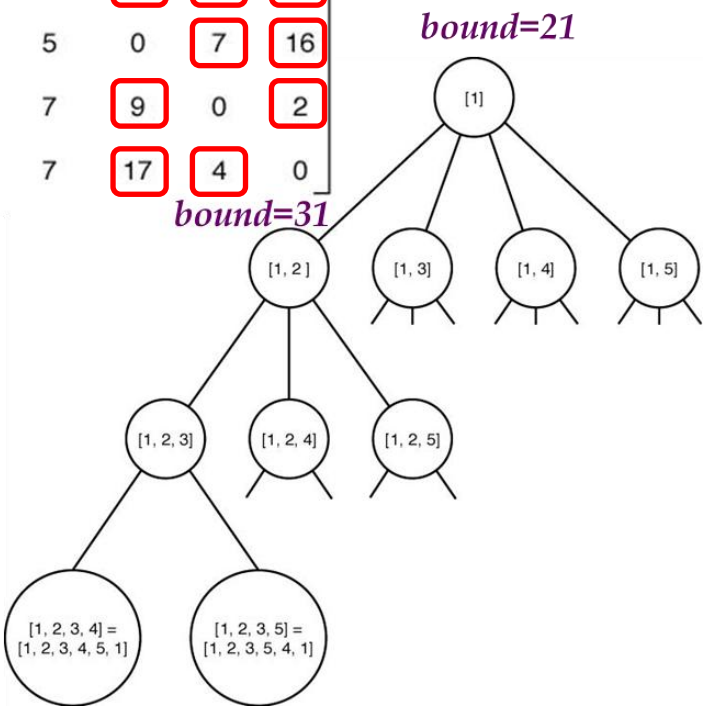
  + v_n 에서 $V - [v_2, \ldots, v_k, v_n]$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치[c]

Traveling Salesman Problem

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

- 노드 [1, 2]를 선택한 경우의 하한 구하기
- 근거: 이미 v_2를 선택하였음을 의미하므로, v_1 → v_2의 비용은 이음선의 가중치인 14가 된다. 나머지는 앞서와 동일한 방법으로 구한다.
  - v_1 → v_2 = 14[a]
  - v_2 → min(7, 8, 7) = 7[b]
  - v_3 → min(4, 7, 16) = 4[c]
  - v_4 → min(11, 9, 2) = 2[c]
  - v_5 → min(18, 17, 4) = 4[c]
- 따라서, [1, 2]를 포함하는 노드에서 확장하여 구한 일주여행경로 길이의 하한은 31(= 14+7+4+2+4)가 된다.

0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

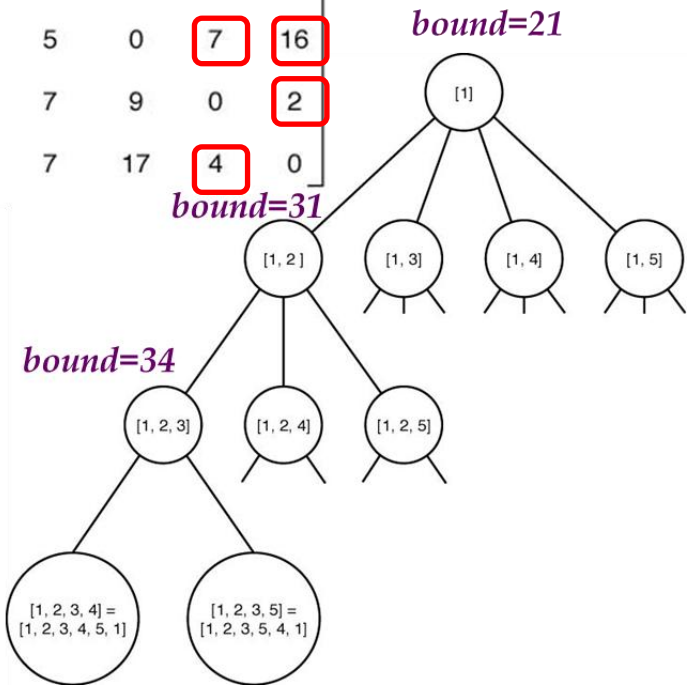


Traveling Salesman Problem

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

- 노드 [1, 2, 3]를 선택한 경우의 하한 구하기
- 근거: 이미 v_2와 v_3를 선택하였음을 의미하므로, $v_1 \rightarrow v_2 \rightarrow v_3$의 비용은 21(=14+7)이 된다. 나머지는 앞서와 동일한 방법으로 구한다.
  - $v_1 \rightarrow v_2 \rightarrow v_3$ = 21[a]
  - $v_3 \rightarrow \min(7, 16) = 7$[b]
  - $v_4 \rightarrow \min(11, 2) = 2$[c]
  - $v_5 \rightarrow \min(18, 4) = 4$[c]

0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0



- 따라서, [1, 2, 3]을 포함하는 노드에서 확장하여 구한 일주여행경로 길이의 하한은 34(= 21+7+2+4)이 된다.

Traveling Salesman Problem

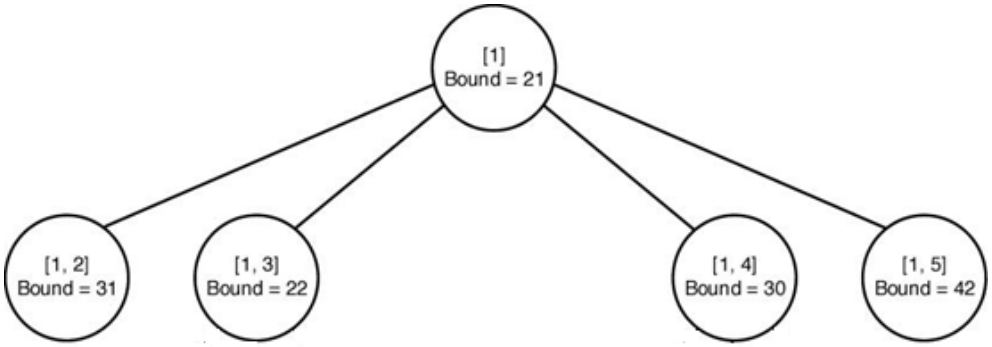
◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

- 한계치(lower bound)와 최소여행경로(Min Length)의 비교
  - 최소여행경로(Min Length) 의 초기값은 ∞로 놓는다.
  - 완전한 여행경로를 처음 얻을 때 까지 방문하는 모든 노드는 한계치가 무조건 ∞로 설정된 최소여행경로의 길이 (Min Length=∞) 보다 작으므로 그러한 모든 노드는 유망하다.
  - 완전한 여행경로를 하나라도 얻은 후에는 ∞가 아닌 최소여행경로의 길이(Min Length)를 얻게 되므로, 이후 방문하는 노드의 한계치가 그러한 최소여행경로의 길이(Min Length)보다 크면 Pruning의 효과가 발생한다.

Traveling Salesman Problem

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

- 루트노드 방문 (Lower Bound = 21, minLen = ∞)
- 기본적으로 너비우선검색 (Breadth-first search)
  - Best First Search 이므로 일반 큐가 아닌 "우선순위 큐" 사용
- 레벨 1에 대한 각 노드
  - 노드 [1, 2] (LB = 31)
  - 노드 [1, 3] (LB = 22)
  - 노드 [1, 4] (LB = 30)
  - 노드 [1, 5] (LB = 42)



- Best-First Search(BFS)에 따라
  한계 값이 가장 작은 [1, 3]을
  우선순위 큐에서 가져온다 (dequeue).

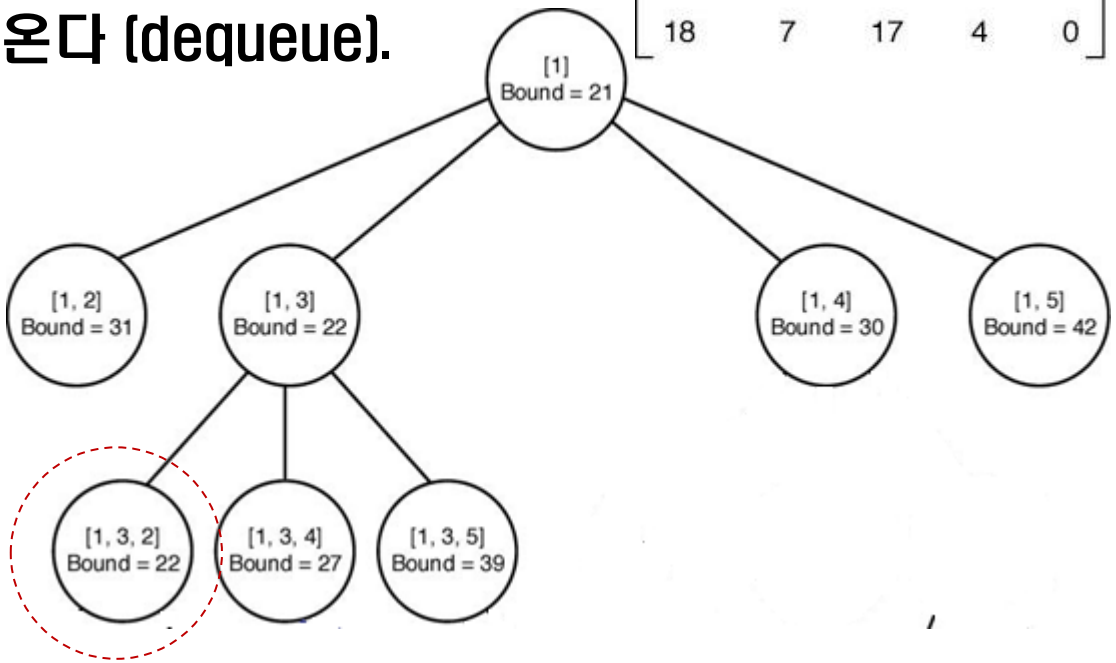
$$\begin{bmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{bmatrix}$$

Traveling Salesman Problem

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

- 노드 [1, 3, 2] (LB = 22)
- 노드 [1, 3, 4] (LB = 27)
- 노드 [1, 3, 5] (LB = 39)
- BFS에 따라 한계 값이 가장 작은 [1, 3, 2]를 우선순위 큐에서 가져온다 (dequeue).

0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

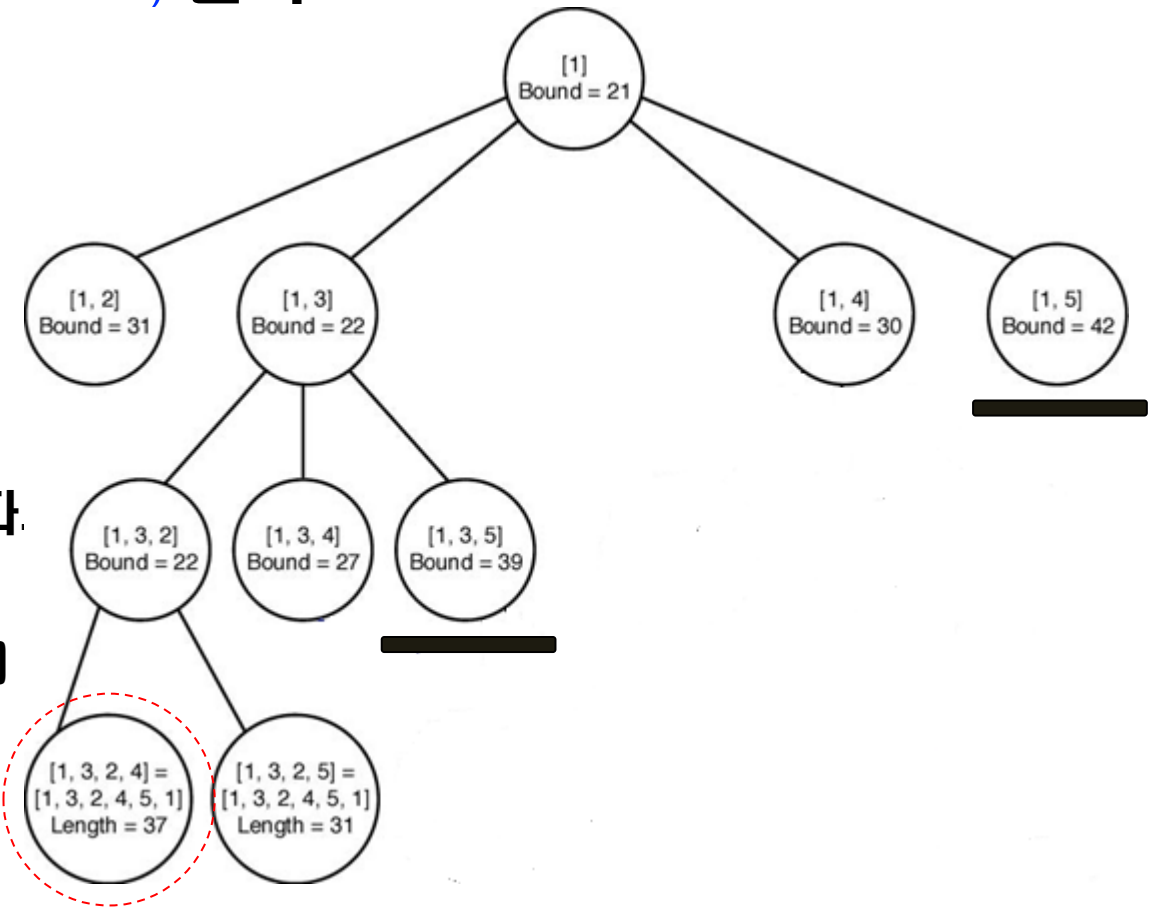


Traveling Salesman Problem

0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

- 노드 [1, 3, 2, 4]
  - 단말노드 이므로 일주여행경로의 길이를 계산한다.
  - 이 길이가 37이고, 37 < ∞ 이므로, minLen = 37이 된다.
  - [1, 5]와 [1, 3, 5]는 한계값(각각 42, 39)이 minLen보다 크므로 Pruning 할 수 있다.

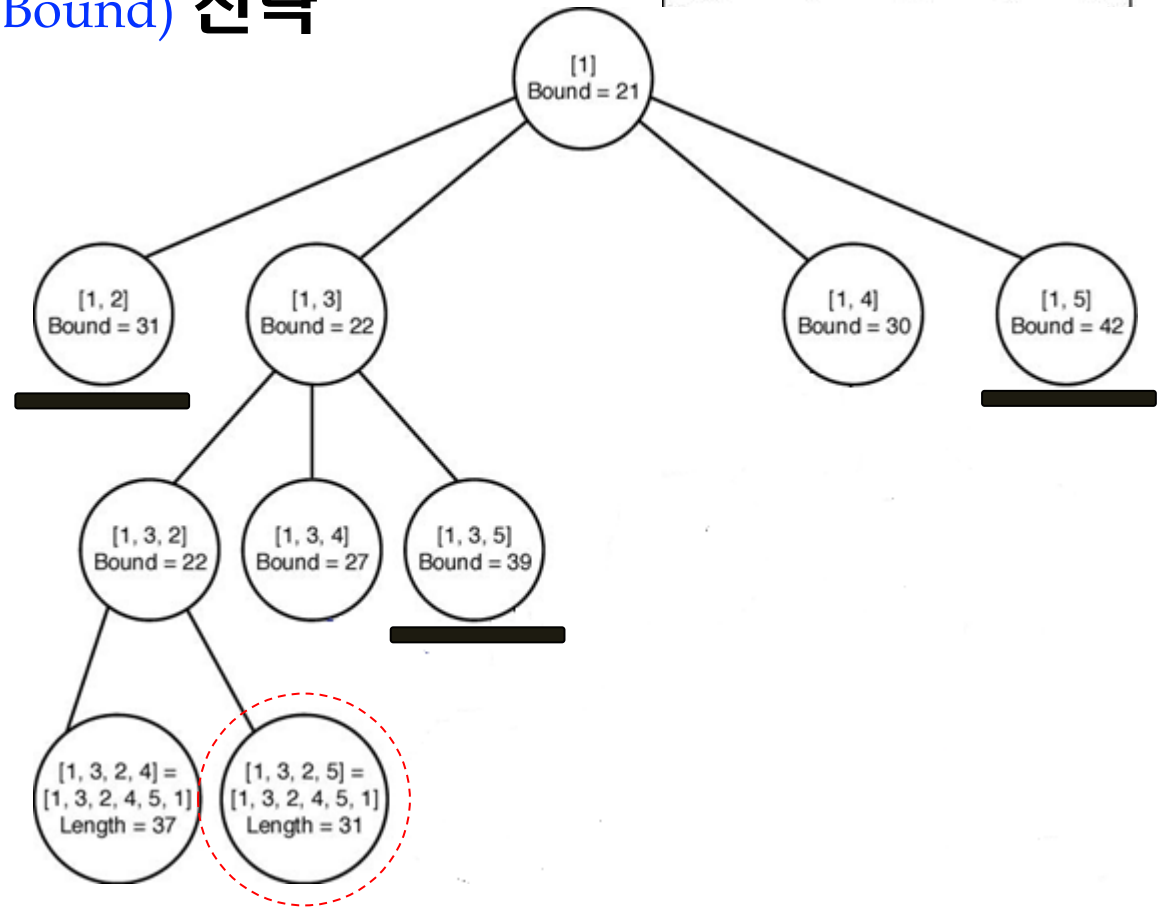


Traveling Salesman Problem

0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

- 노드 [1, 3, 2, 5]
  - 방문 결과 minLen = 31이 된다.
  - [1, 2]를 가지치기 할 수 있다.
- 다음으로, [1, 3, 4] 선택한다.
- 상기 과정을 계속 반복하면, minLen = 30을 최소 일주 경로 길이로 구할 수 있다.

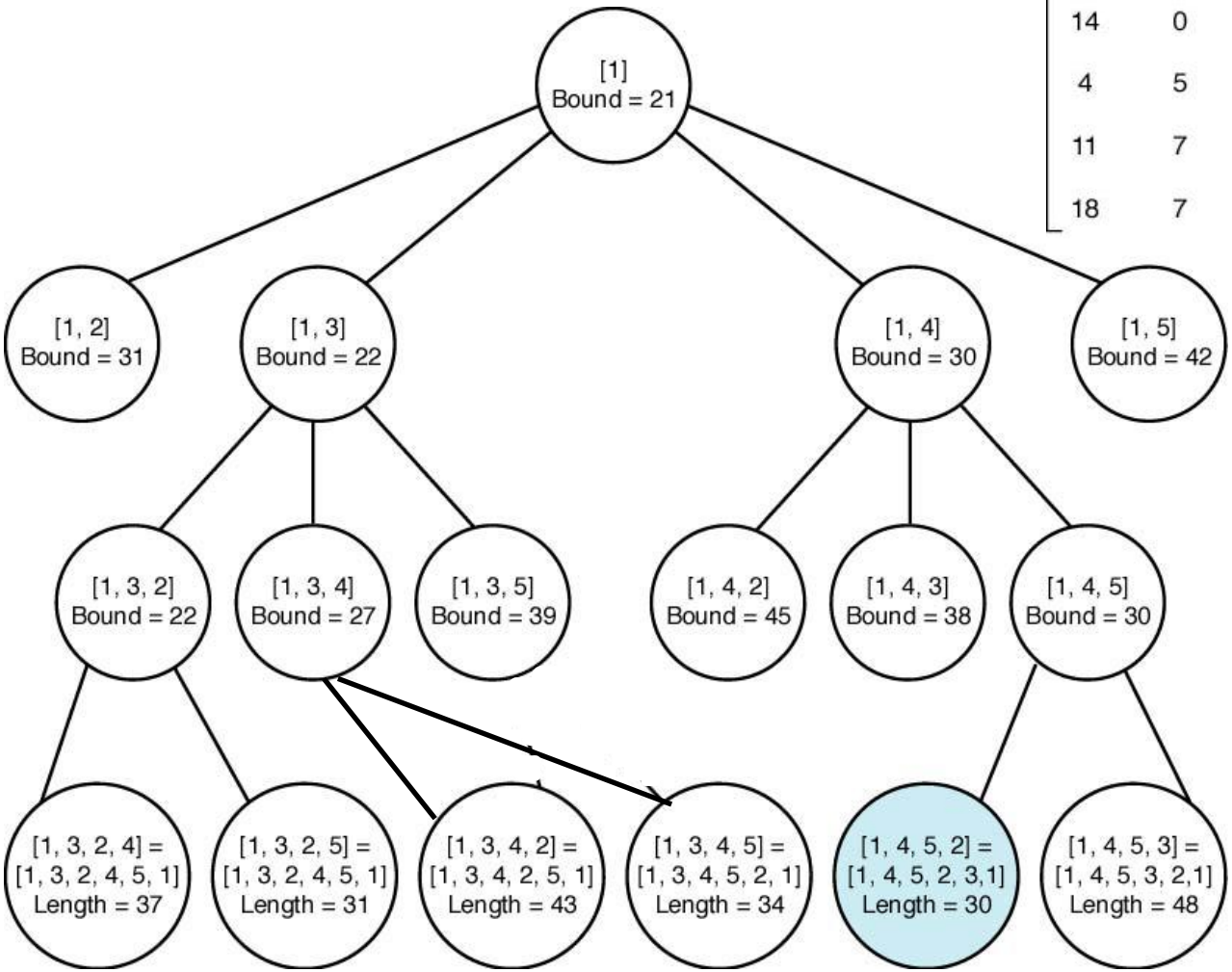


Traveling Salesman Problem

◈ 최고 우선 검색 기반 한정 분기(Branch and Bound) 전략

– 종합적인 상태 공간 트리 탐색 모습

$$\begin{bmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{bmatrix}$$



TSP 문제의 다른 예

◈ 비대칭 TSP 문제의 상태 공간 트리

- 리프 노드의 네모에 적힌 숫자는 각 헤미토니안 사이클 길이
- 모든 헤밀토니안 사이클의 개수
  - 4! = 24 (리프 노드 수)
- 총 노드의 개수
  - 41개

	1	2	3	4	5
1	0	10	10	30	25
2	10	0	14	21	10
3	10	18	0	7	9
4	8	11	7	0	3
5	14	10	10	3	0

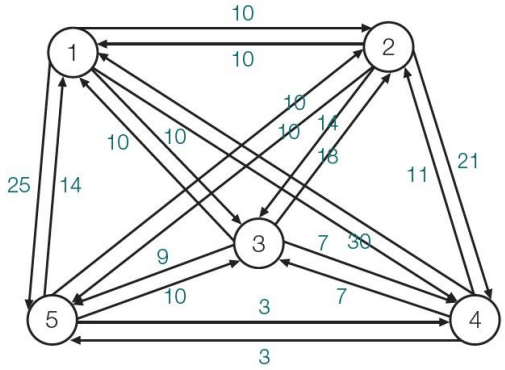


그림 14-2 비대칭 TSP 문제의 인접 행렬과 그래프

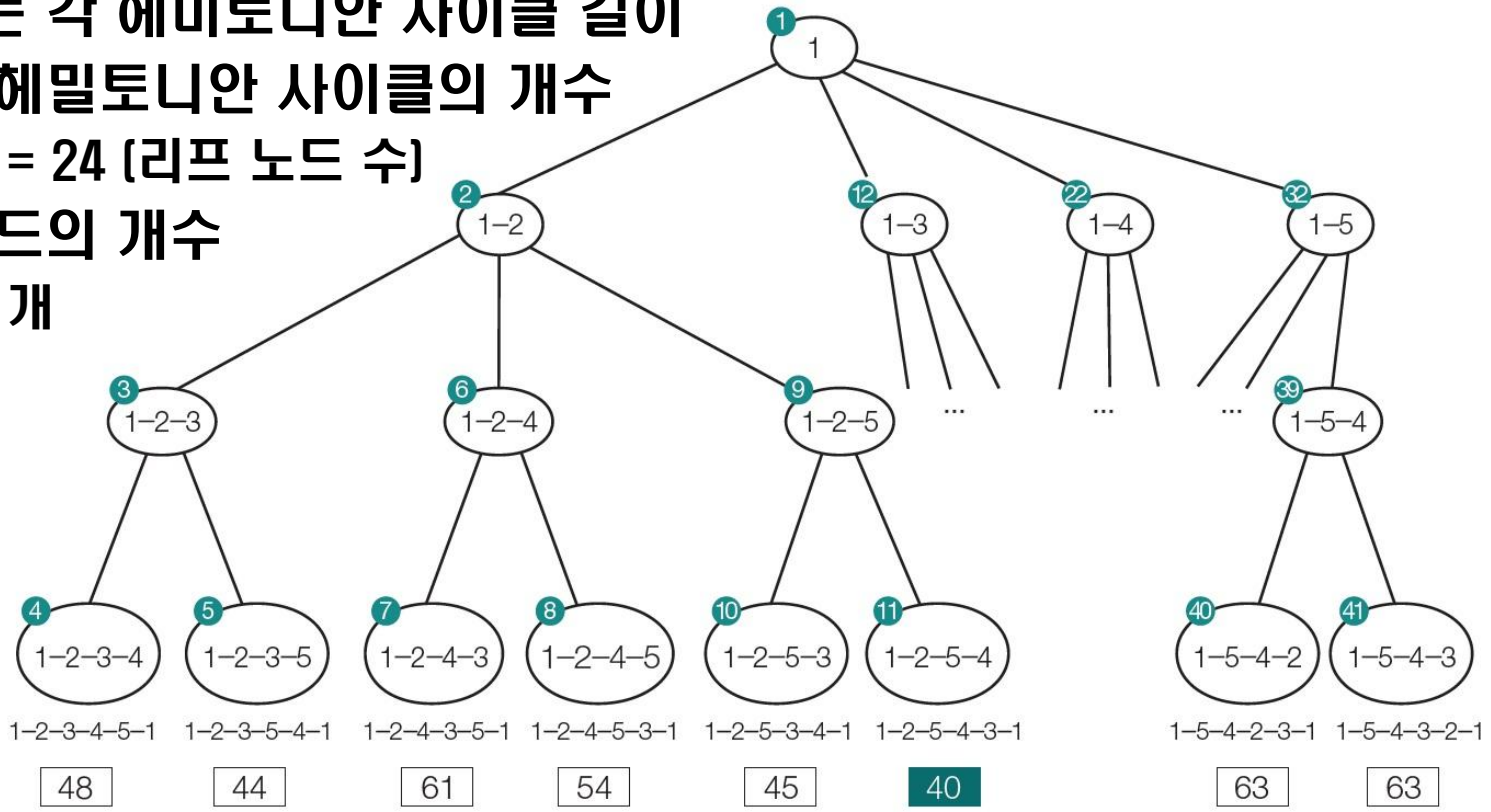


그림 14-3 [그림 14-2]의 TSP 예를 대상으로 한 사전식 탐색의 예

TSP 문제의 다른 예

◈ 최고 우선 검색을 사용한 한정 분기(Branch and Bound) 전략

모든 헤밀토니안 사이클의 개수
- 8 (리프 노드 수)

총 탐색 노드의 개수
- 18개

	1	2	3	4	5
1	0	10	10	30	25
2	10	0	14	21	10
3	10	18	0	7	9
4	8	11	7	0	3
5	14	10	10	3	0

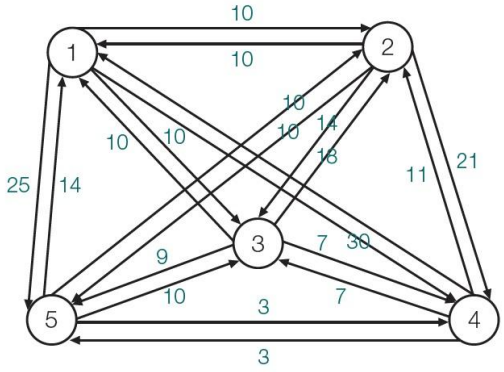


그림 14-2 비대칭 TSP 문제의 인접 행렬과 그래프

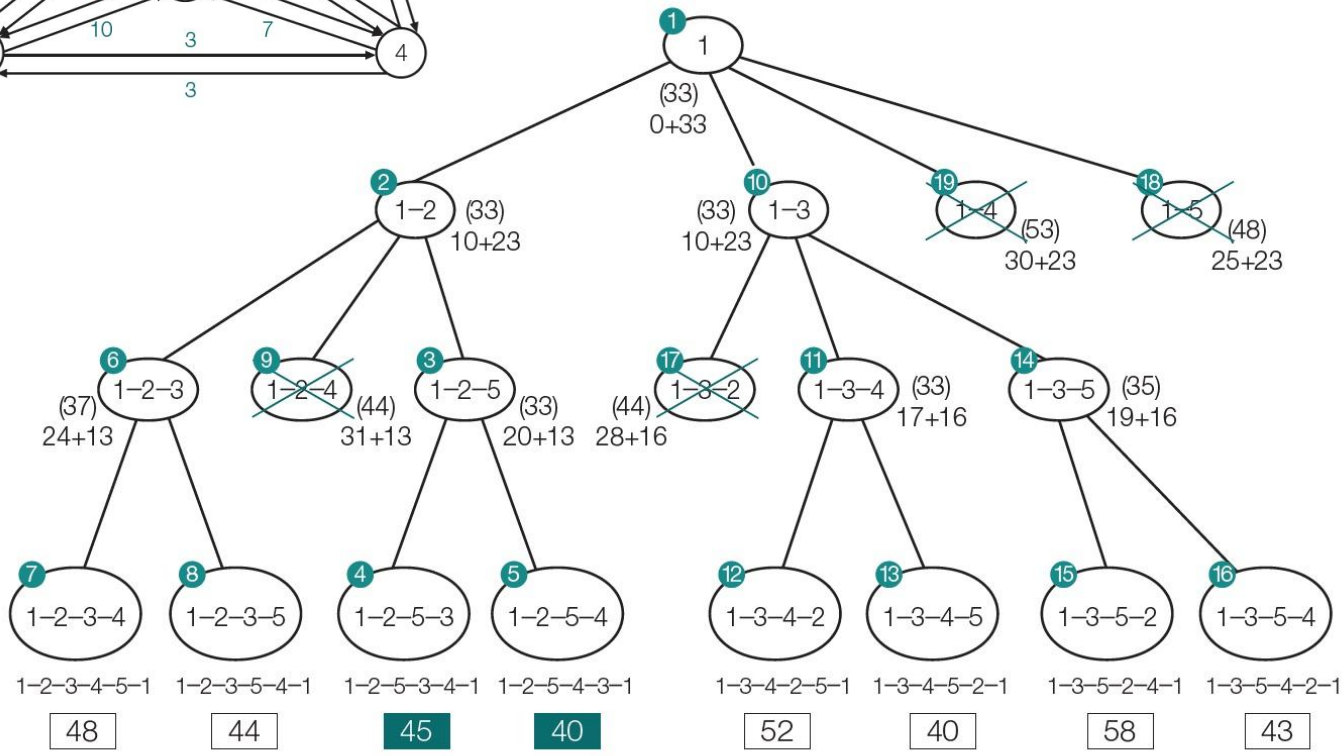


그림 14-8 [그림 14-3]의 TSP 예를 대상으로 한 한정 분기 탐색의 예(상태 공간 트리)

04. A* 알고리즘 (SKIP)

Questions & Answers