

# 분기한정법 (Branch-and-Bound)

알고리즘 해석  
강의 슬라이드 #6



# 분기 한정법 (Branch-and-Bound)

- 특징:

- ✓ 되추적 기법과 같이 상태공간트리를 구축하여 문제를 해결한다.
- ✓ 최적의 해를 구하는 문제(optimization problem)에 적용할 수 있다.
- ✓ 최적의 해를 구하기 위해서는 어차피 모든 해를 다 고려해 보아야 하므로 트리의 마디를 순회(traverse)하는 방법에 구애받지 않는다.

- 분기 한정 알고리즘의 원리

- ✓ 각 마디를 검색할 때 마다, 그 마디가 유망한지의 여부를 결정하기 위해서 한계치(bound)를 계산한다.
- ✓ 그 한계치는 그 마디로부터 가지를 뺀어나가서(branch) 얻을 수 있는 해답치의 한계를 나타낸다.
- ✓ 따라서 만약 그 한계치가 지금까지 찾은 최적의 해답치 보다 좋지 않은 경우는 더 이상 가지를 뺀어서 검색을 계속할 필요가 없으므로, 그 마디는 유망하지 않다고 할 수 있다.

# 0-1 배낭채우기 문제

- 분기한정 가지치기로 깊이우선검색 (= 되추적)
  - ✓ 상태공간트리를 구축하여 되추적 기법으로 문제를 푼다.
  - ✓ 뿌리마디에서 왼쪽으로 가면 첫번째 아이템을 배낭에 넣는 경우이고, 오른쪽으로 가면 첫번째 아이템을 배낭에 넣지 않는 경우이다.
  - ✓ 동일한 방법으로 트리의 수준 1에서 왼쪽으로 가면 두 번째 아이템을 배낭에 넣는 경우이고, 오른쪽으로 가면 그렇지 않는 경우이다.
  - ✓ 이런 식으로 계속하여 상태공간트리를 구축하면, 뿌리마디로부터 잎마디까지의 모든 경로는 해답후보가 된다.
  - ✓ 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 해를 기억해 두어야 한다.

# 최적화 문제를 풀기 위한 일반적인 되추적 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);  
}
```

- ✓ best : 지금까지 찾은 제일 좋은 해답치.
- ✓ value(v) : v 마디에서의 해답치.

# 0-1 배낭채우기: 알고리즘

## ● 알고리즘 스케치:

✓ Let:

- *profit*: 그 마디에 오기까지 넣었던 아이템의 값어치의 합.
- *weight*: 그 마디에 오기까지 넣었던 아이템의 무게의 합.
- *bound*: 마디가 수준  $i$ 에 있다고 하고, 수준  $k$ 에 있는 마디에서 총무게가  $W$ 를 넘는다고 하자. 그러면

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

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

- *maxprofit*: 지금까지 찾은 최선의 해답이 주는 값어치
- ✓  $w_i$ 와  $p_i$ 를 각각  $i$ 번째 아이템의 무게와 값어치라고 하면,  $p_i/w_i$ 의 값이 큰 것부터 내림차순으로 아이템을 정렬한다. (일종의 탐욕적인 방법이 되는 셈이지만, 알고리즘 자체는 탐욕적인 알고리즘은 아니다.)
- ✓  $maxprofit := \$0$ ;  $profit := \$0$ ;  $weight := 0$

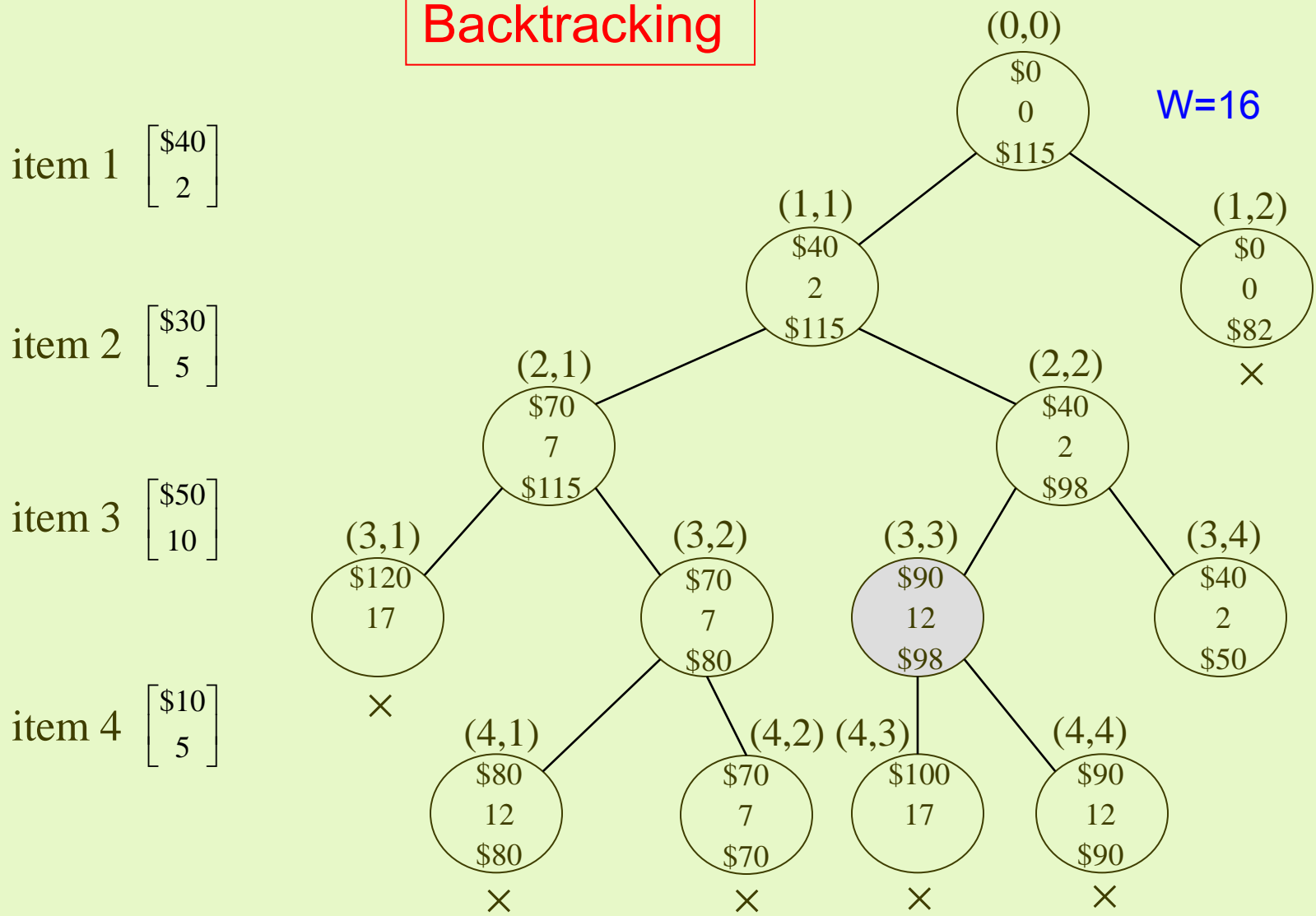
- ✓ 깊이우선순위로 각 마디를 방문하여 다음을 수행한다:
  1. 그 마디의 *profit*와 *weight*를 계산한다.
  2. 그 마디의 *bound*를 계산한다.
  3.  $weight < W$  and  $bound > maxprofit$ 이면, 검색을 계속한다; 그렇지 않으면, 되추적.
- ✓ 고찰: 최선이라고 여겼던 마디를 선택했다고 해서 실제로 그 마디로부터 최적해가 항상 나온다는 보장은 없다.

● 보기:  $n = 4$ ,  $W = 16$ 이고

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

일 때, 되추적을 사용하여 구축되는 가지친 상태공간트리를 그려 보시오.

# Backtracking



# 0-1 배낭채우기 알고리즘: 분석

- 이 알고리즘이 점검하는 마디의 수는  $\Theta(2^n)$ 이다.
- 위 보기의 경우의 분석: 점검한 마디는 13개이다. 이 알고리즘이 동적계획법으로 설계한 알고리즘 보다 좋은가?
  - ✓ 확실하게 대답하기 불가능 하다.
  - ✓ Horowitz와 Sahni(1978)는 Monte Carlo 기법을 사용하여 되추적 알고리즘이 동적계획법 알고리즘 보다 일반적으로 더 빠르다는 것을 입증하였다.
  - ✓ Horowitz와 Sahni(1974)가 분할정복과 동적계획법을 적절히 조화하여 개발한 알고리즘은  $O(2^{n/2})$ 의 시간복잡도를 가지는데, 이 알고리즘은 되추적 알고리즘 보다 일반적으로 빠르다고 한다.

# 분기 한정 알고리즘의 특성

- 분기 한정 알고리즘도 되추적 알고리즘의 한 종류이다.
  - ✓ 한계값이 *maxprofit*의 현재값보다 크지 않다면 유망함수는 *false*를 넘겨준다.
  - ✓ (깊이우선탐색에 기반한) 되추적 알고리즘은 분기한정을 사용하여 얻을 수 있는 장점을 제대로 살리지 못한다.
- 분기 한정 가지치기 최고우선검색(best-first search with branch-and-bound pruning)
  - ✓ 마디가 유망한지를 결정하기 위해 한계값을 사용하고,
  - ✓ 유망한 마디들의 한계값을 비교하여, 그 중에서 가장 좋은 한계값을 가진 마디의 자식마디를 방문한다.
  - ✓ 먼저, 분기 한정 가지치기 너비우선검색(breadth-first search with branch-and-bound pruning)을 이해하고, 이를 수정한다.

## 6.1.1 분기한정 가지치기로 너비우선검색

- 너비우선검색(Breadth-first Search)순서:
  - (1) 뿌리마디를 먼저 검색한다.
  - (2) 다음에 수준 1에 있는 모든 마디를 검색한다.  
(왼쪽에서 오른쪽으로)
  - (3) 다음에 수준 2에 있는 모든 마디를 검색한다  
(왼쪽에서 오른쪽으로)
  - (4) ...

# 일반적인 너비우선검색 알고리즘

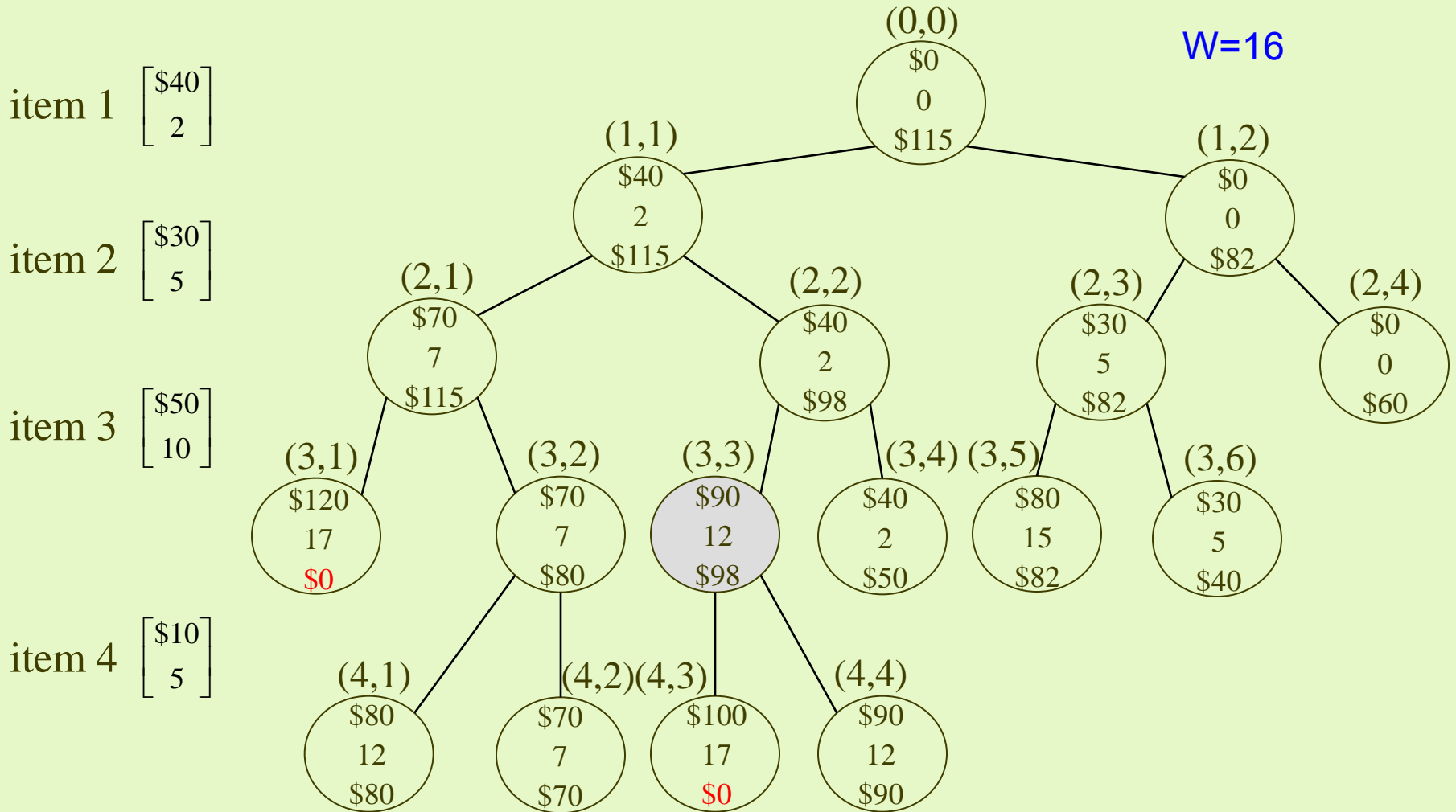
```
void breadth_first_search(tree T) {  
    queue_of_node Q;      // 대기열(queue)를 사용  
    node u, v;  
  
    initialize(Q);        // Q를 빈 대기열로 초기화  
    v = root of T;  
    visit v;  
    enqueue(Q, v);  
    while (!empty(Q)) {  
        dequeue(Q, v);  
        for (each child u of v) {  
            visit u;  
            enqueue(Q, u);  
        }  
    }  
}
```

# 분기 한정 너비우선검색 알고리즘

```
void breadth_first_branch_and_bound(state_space_tree T, number& best)
{
    queue_of_node Q;
    node u, v;

    initialize(Q); // Q는 빈 대기열로 초기화
    v = root of T; //뿌리마디를 방문
    enqueue(Q,v);
    best = value(v);
    while (!empty(Q)) {
        dequeue(Q, v);
        for (each child u of v) { // 각 자식마디를 방문
            if (value(u) is better than best)
                best = value(u);
            if (bound(u) is better than best) // 한계값이 더 좋은 경우만 분기
                enqueue(Q, u);
        }
    }
}
```

- 보기: 앞에서와 같은 예를 사용하여 **분기한정 가지치기로 너비우선검색**을 하여 가지친 상태공간트리를 그려보면 다음과 같이 된다. 이때 검색하는 마디의 개수는 **17**이다. 되추적 알고리즘보다 좋지 않다!



```

void knapsack2 (int n,
               const int p[], const int w[],
               int W,
               int& maxprofit)
{
    queue_of_node Q;
    node u, v;
    initialize(Q); // Initialize Q to be empty.
    v.level = 0; v.profit = 0; v.weight = 0; // Initialize v to be the root.
    maxprofit = 0;
    enqueue(Q, v);
    while (! empty(Q)) {
        dequeue(Q, v);
        u.level = v.level + 1; // Set u to a child of v.
        u.weight = v.weight + w[u.level]; // Set u to the child that
        u.profit = v.profit + p[u.level]; // includes the next item.
        if (u.weight <= W && u.profit > maxprofit)
            maxprofit = u.profit;
        if (bound(u) > maxprofit)
            enqueue(Q, u);
        u.weight = v.weight; // Set u to the child that
        u.profit = v.profit; // does not include the
        if (bound(u) > maxprofit) // next item.
            enqueue(Q, u);
    }
}

```

```

float bound (node u)
{
    index j, k;
    int totweight;
    float result;

    if (u.weight > = W)
        return 0;
    else {
        result = u.profit;
        j = u.level + 1;
        totweight = u.weight;
        while (j <= n && totweight + w[j] <= W){ // Grab as many items
            totweight = totweight + w[j]; // as possible.
            result = result + p[j];
            j++;
        }
        k = j; // Use k for consistency
        if (k <= n) // with formula in text.
            result = result + (W - totweight) * p[k]/w[k]; // Grab fracton of kth
        return result; // item.
    }
}

```

## 6.1.2 분기한정 가지치기로 최고우선 검색 (Best-First Search)

- 최적의 해답에 더 빨리 도달하기 위한 전략:
  1. 주어진 마디의 모든 자식마디를 검색한 후,
  2. 유망하면서 확장되지 않은(unexpanded) 마디를 살펴보고,
  3. 그 중에서 가장 좋은(최고의) 한계치(bound)를 가진 마디를 확장한다.
- 최고우선검색(Best-First Search)은 너비우선검색에 비해서 좋아짐

# 최고우선검색 전략

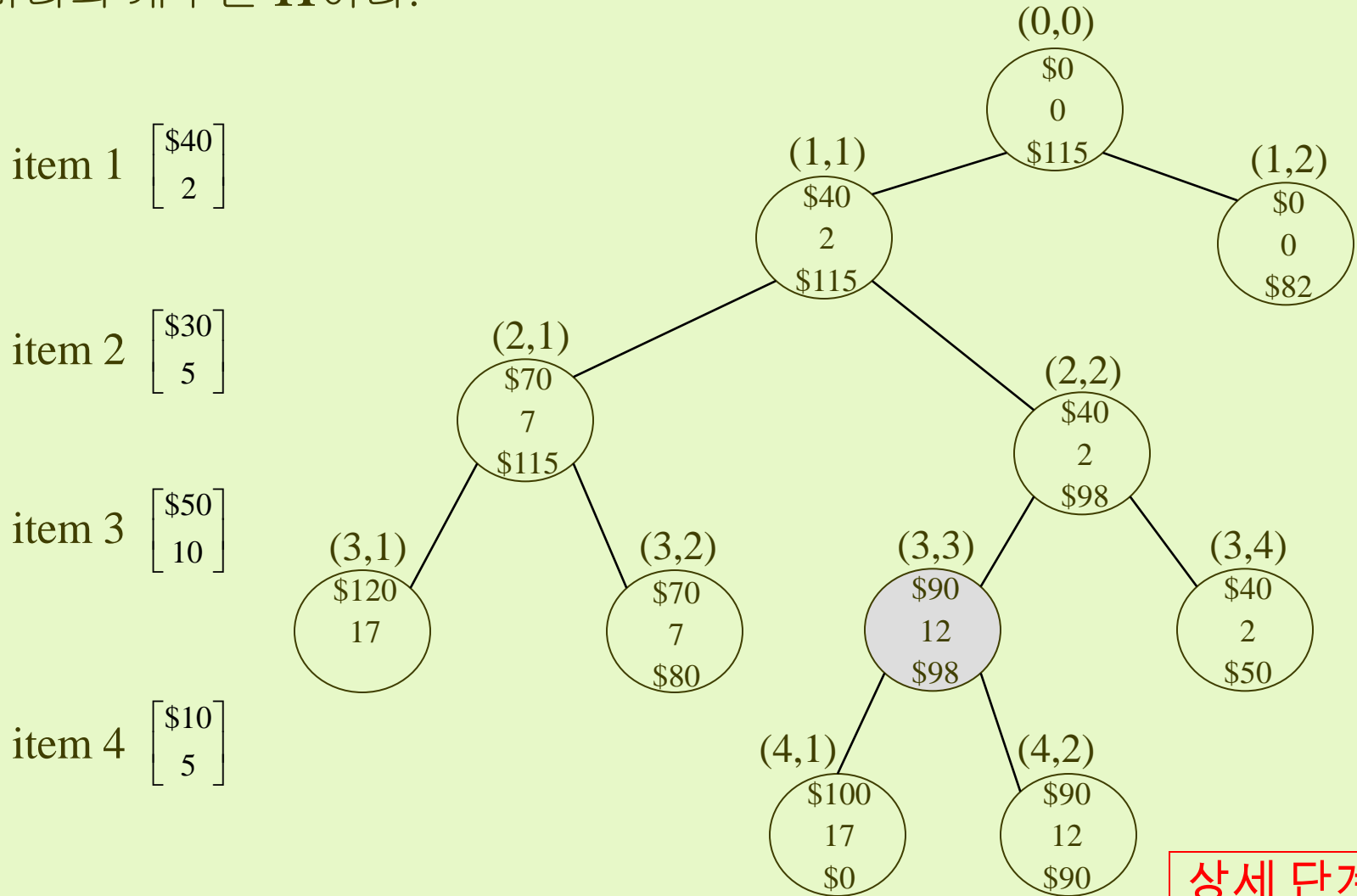
- 최고의 한계를 가진 마디를 우선적으로 선택하기 위해서 우선순위 대기열(Priority Queue)을 사용한다.
- 우선순위 대기열은 힙(heap)을 사용하여 효과적으로 구현할 수 있다.

# 분기한정 최고우선검색 알고리즘

```
void best_first_branch_and_bound(state_space_tree T, number best)
{
    priority_queue_of_node PQ; // priority queue
    node u, v;

    initialize(PQ); // PQ를 빈 대기열로 초기화
    v = root of T;
    best = value(v);
    insert(PQ, v);
    while (!empty(PQ)) { // 최고 한계값을 가진 마디를 제거
        remove(PQ, v);
        if (bound(v) is better than best) // 마디가 아직 유망한 지 점검
            for (each child u of v) {
                if (value(u) is better than best)
                    best = value(u);
                if (bound(u) is better than best)
                    insert(PQ,u);
            }
    }
}
```

- 보기: 앞에서와 같은 예를 사용하여 **분기한정 가지치기로 최고우선검색**을 하여 가지친 상태공간트리를 그려보면, 다음과 같이 된다. 이때 검색하는 마디의 개수는 **11**이다.



상세 단계 설명

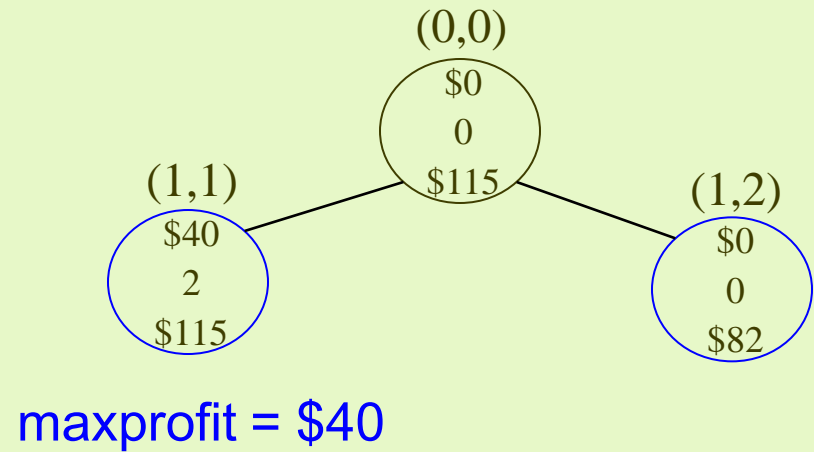
# Best-first search with branch-and-bound pruning

item 1  $\begin{bmatrix} \$40 \\ 2 \end{bmatrix}$

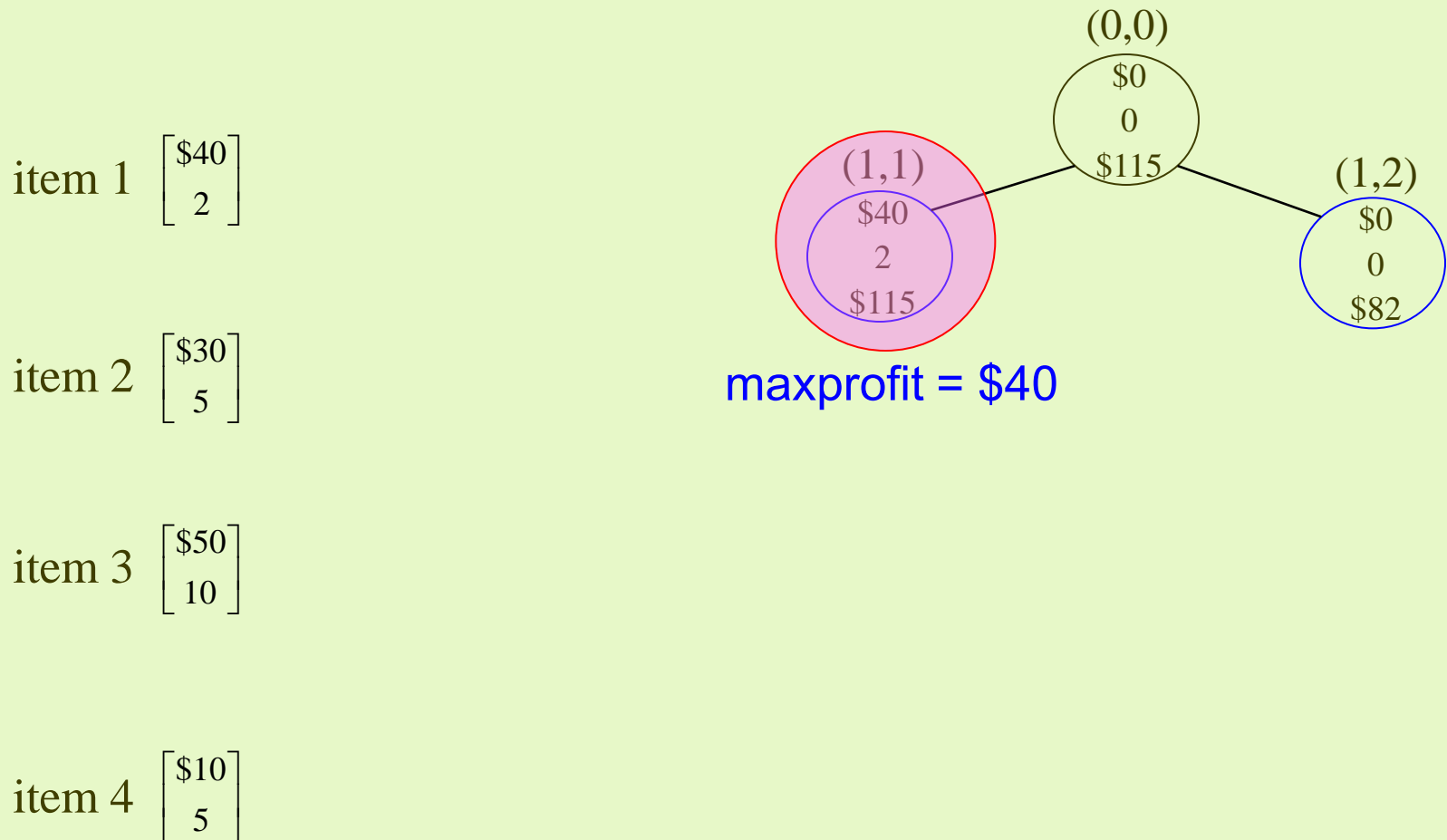
item 2  $\begin{bmatrix} \$30 \\ 5 \end{bmatrix}$

item 3  $\begin{bmatrix} \$50 \\ 10 \end{bmatrix}$

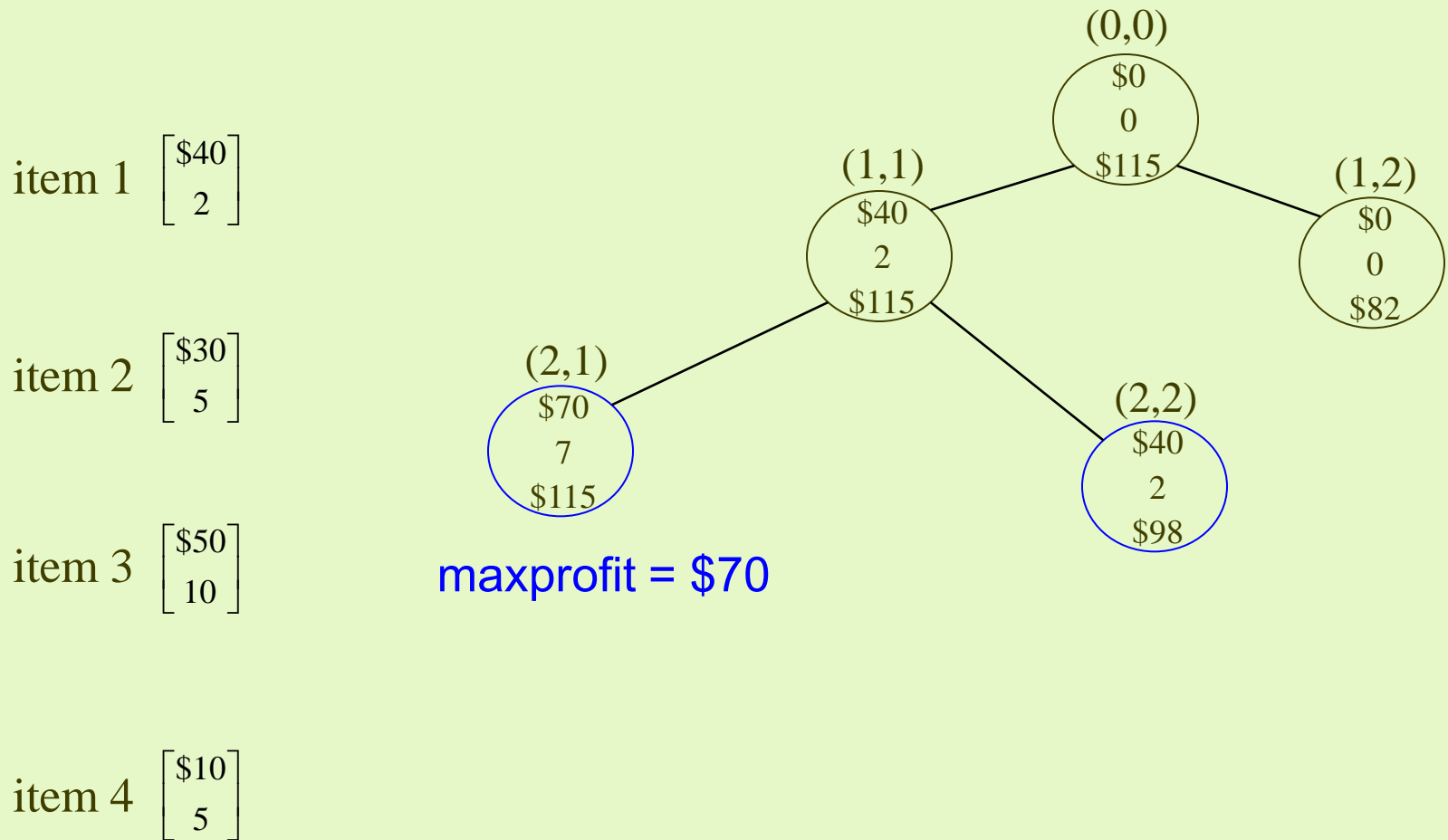
item 4  $\begin{bmatrix} \$10 \\ 5 \end{bmatrix}$



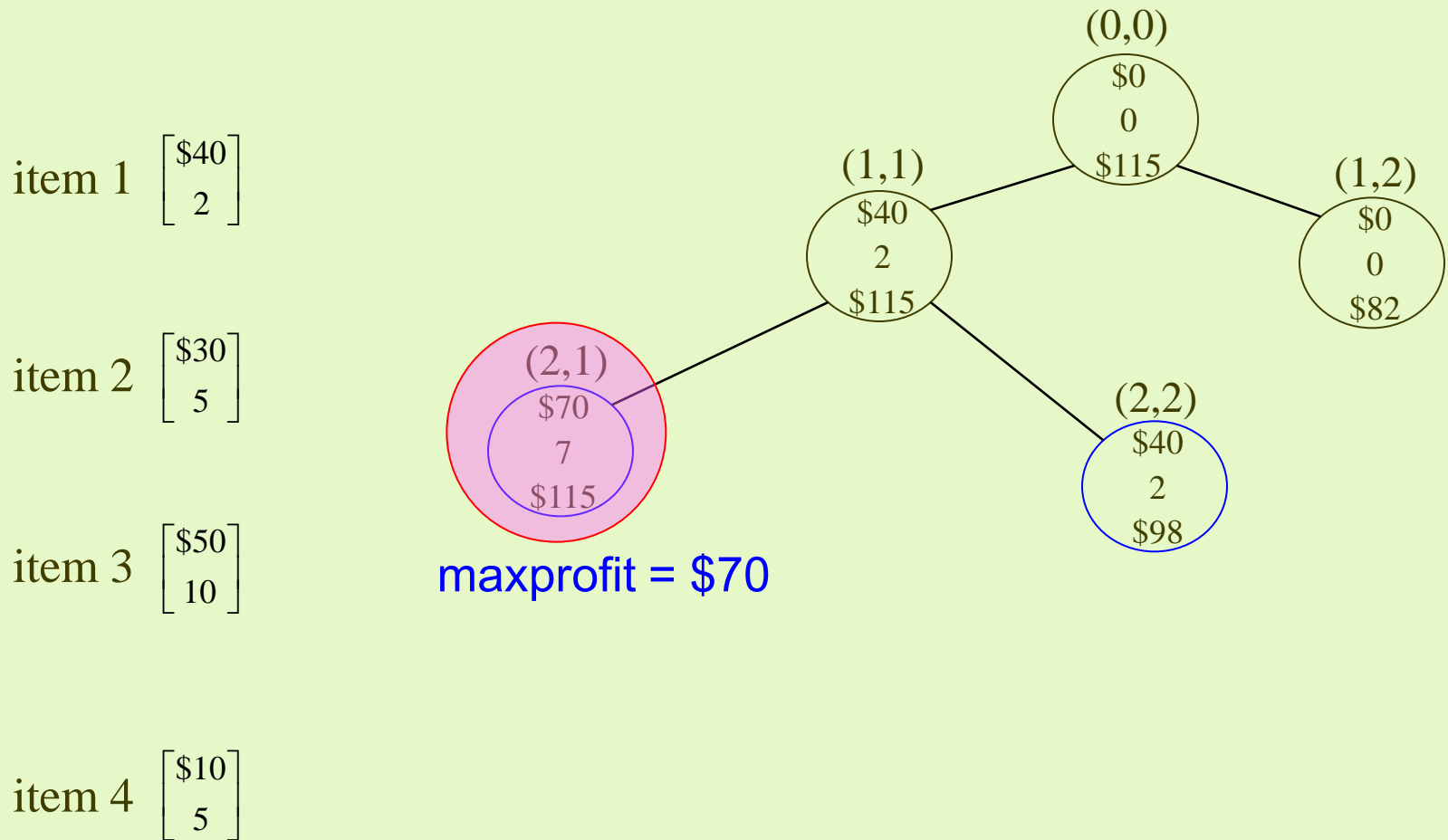
# Best-first search with branch-and-bound



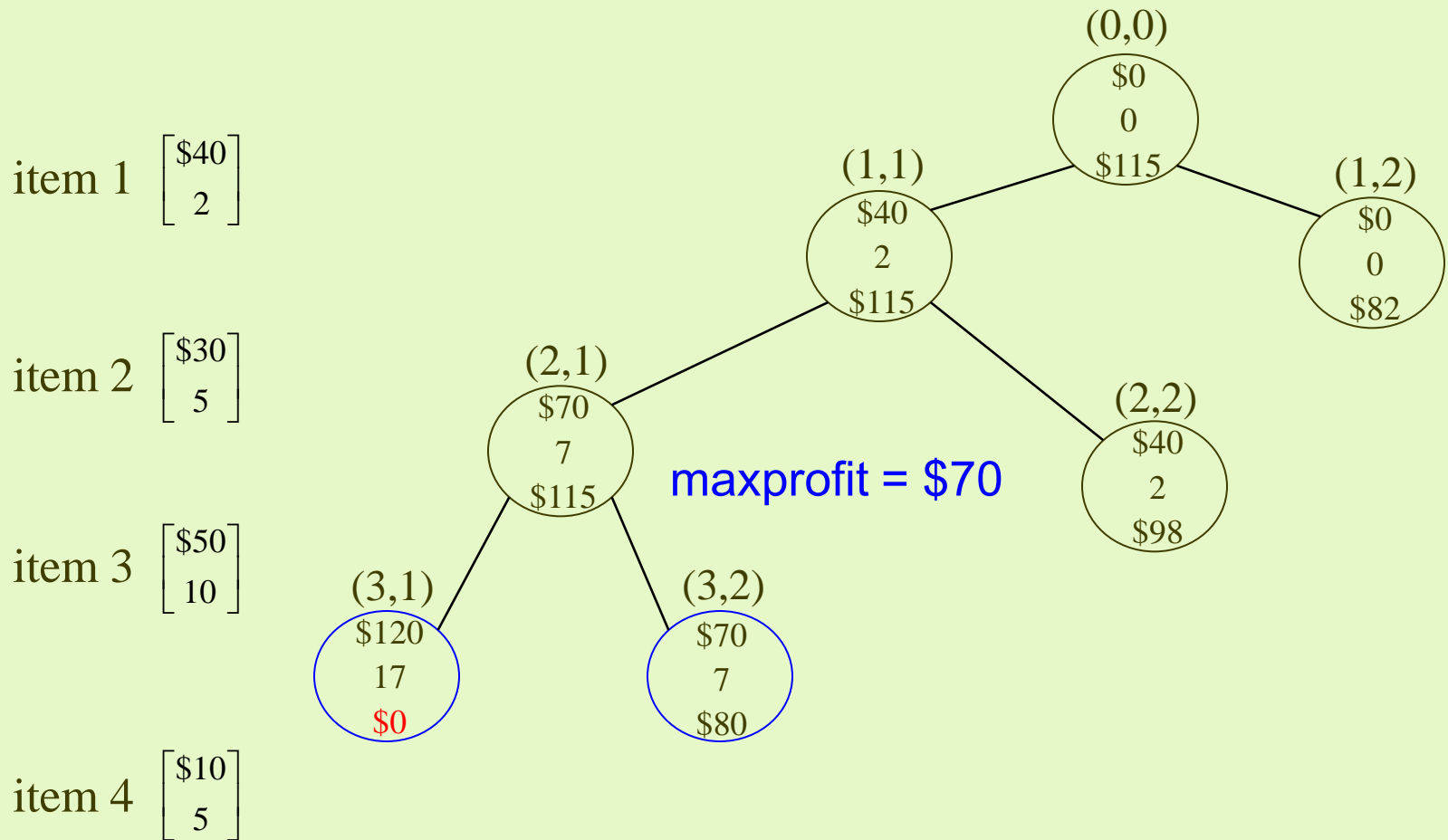
# Best-first search with branch-and-bound



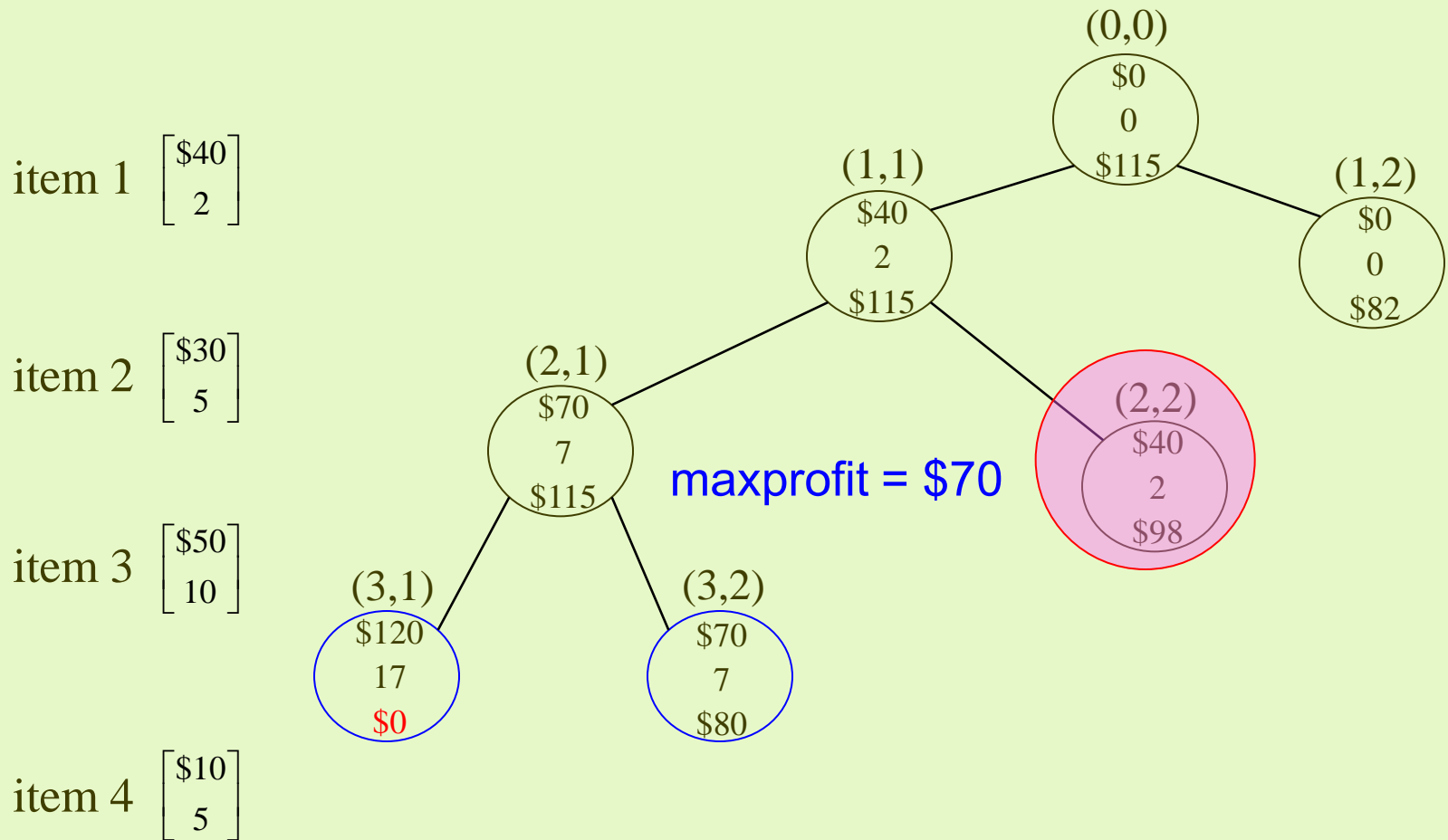
# Best-first search with branch-and-bound



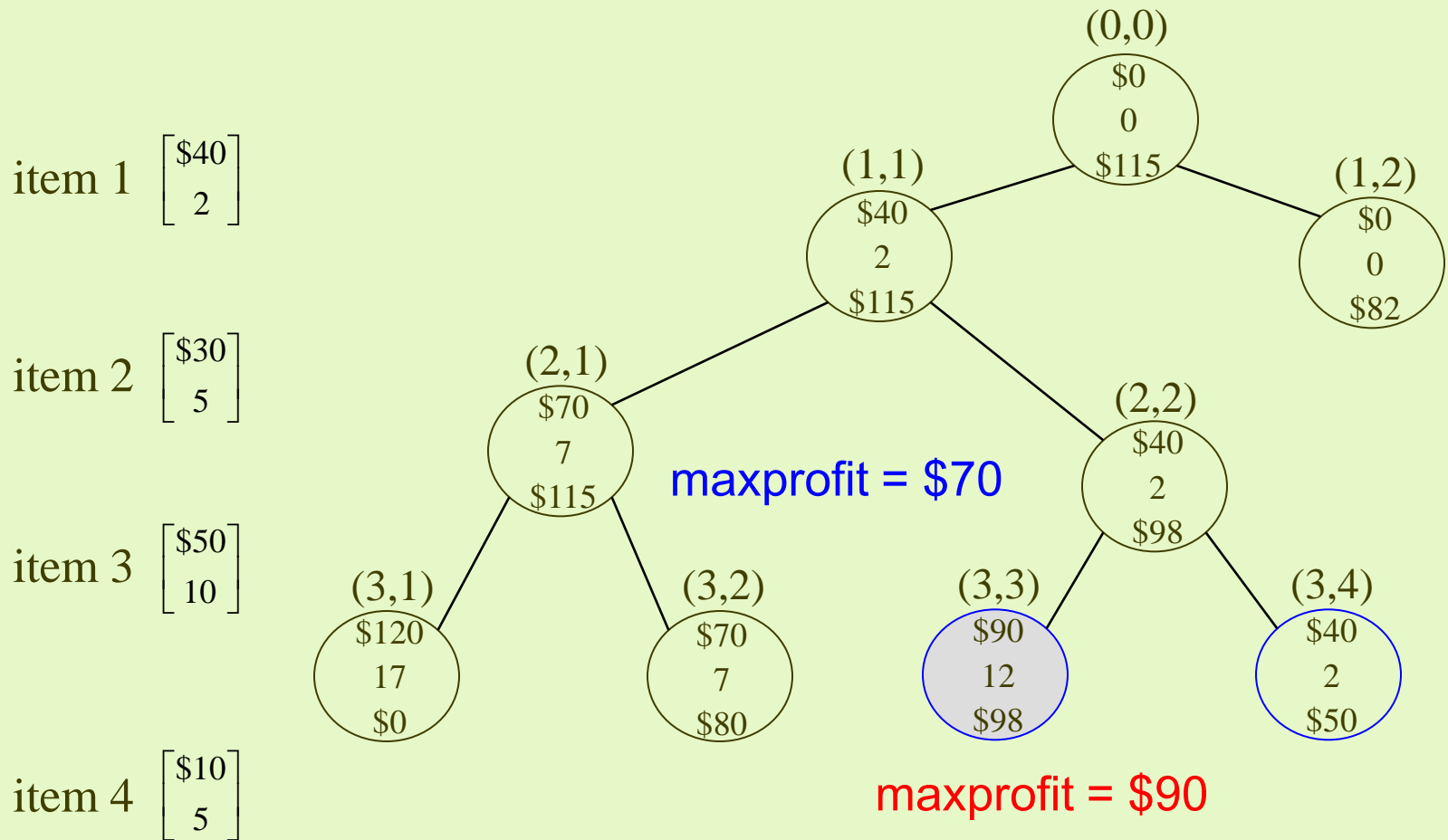
# Best-first search with branch-and-bound



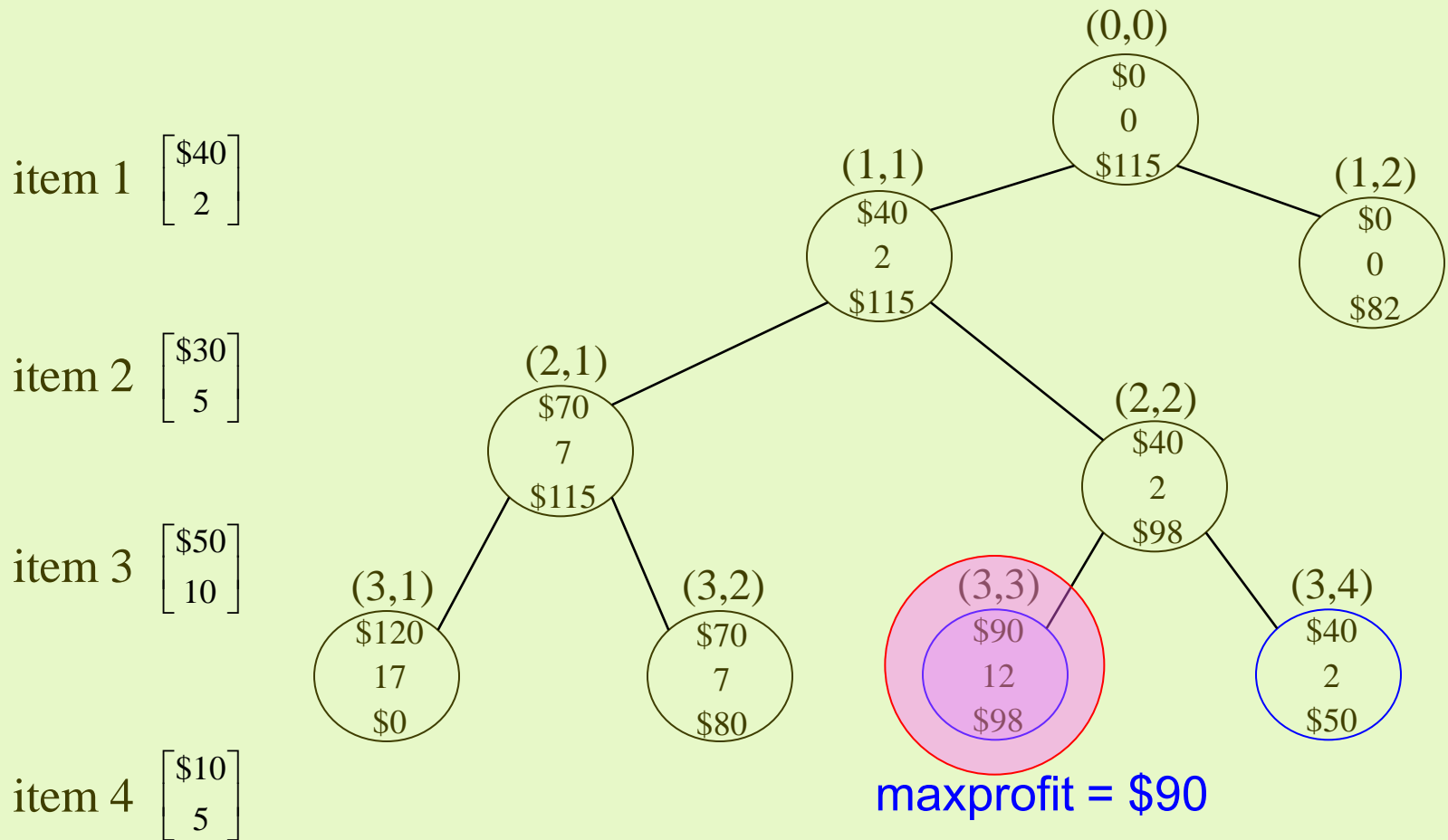
# Best-first search with branch-and-bound



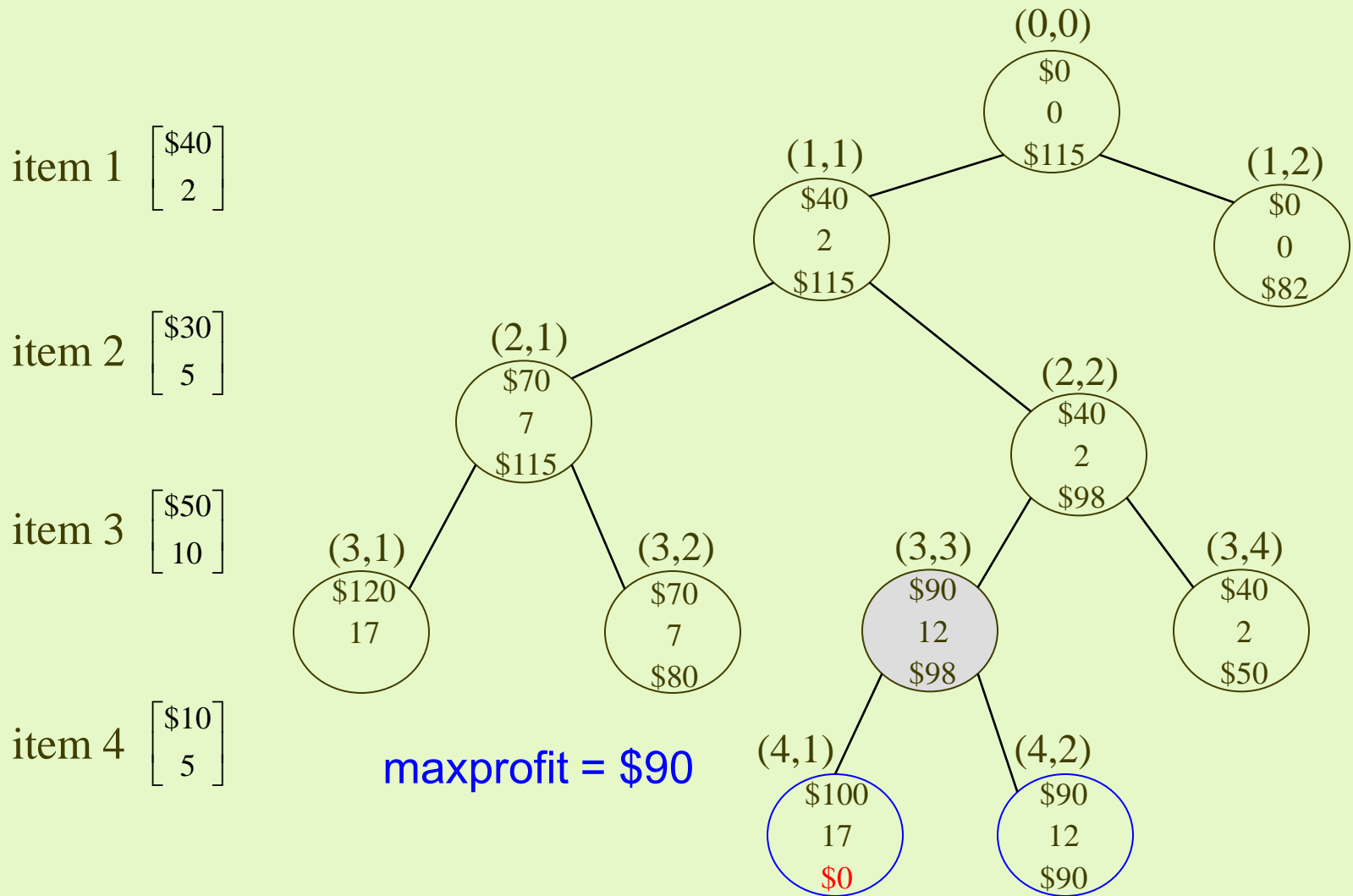
# Best-first search with branch-and-bound



# Best-first search with branch-and-bound



# Best-first search with branch-and-bound



```

void knapsack3 (int n,
                const int p[], const int w[],
                int W,
                int& maxprofit)
{
    priority_queue_of_node PQ;
    node u, v;

    initialize(PQ); // Initialize PQ to be empty.
    v.level = 0; v.profit = 0; v.weight = 0; // Initialize v to be the root.
    maxprofit = 0;
    v.bound = bound(v);
    insert(PQ, v);
    while (!empty(PQ)) { // Remove node with
        remove(PQ, v); // best bound.
        if (v.bound > maxprofit) { // Check if node is still
            u.level = v.level + 1; // promising.
            u.weight = v.weight + w[u.level]; // Set u to the child
            u.profit = v.profit + p[u.level]; // that includes the
            if (u.weight <= W && u.profit > maxprofit) // next item.
                maxprofit = u.profit;
            u.bound = bound(u);
            if (u.bound > maxprofit)
                insert(PQ, u);
            u.weight = v.weight; // Set u to the child
            u.profit = v.profit; // that does not include
            u.bound = bound(u); // the next item.
            if (u.bound > maxprofit)
                insert(PQ, u);
        }
    }
}

```

# 고찰 사항

## ● 0-1 knapsack problem

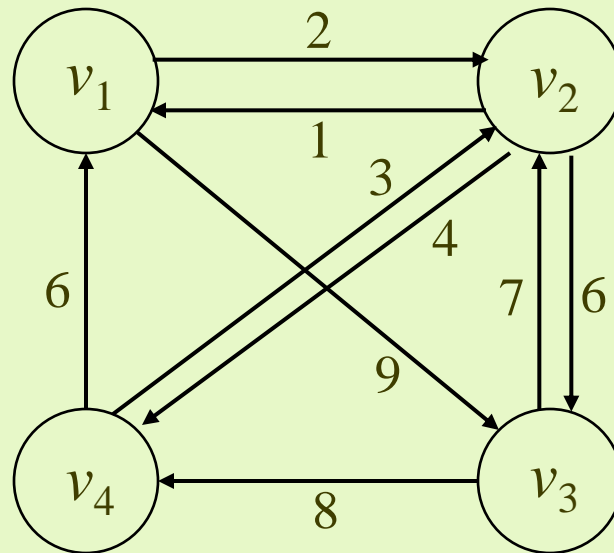
- ✓ 4.5.3 0-1 배낭채우기 문제를 푸는 동적 계획법
- ✓ 5.7.1 0-1 배낭채우기 문제를 푸는 되추적 알고리즘
- ✓ 알고리즘 6.1 0-1 배낭채우기 문제를 푸는 분기한정 가지치기 너비우선검색 알고리즘
- ✓ 알고리즘 6.2 0-1 배낭채우기 문제를 푸는 분기한정 가지치기 최고우선검색 알고리즘

# 6.2 외판원 문제

## (Traveling Saleswoman Problem)

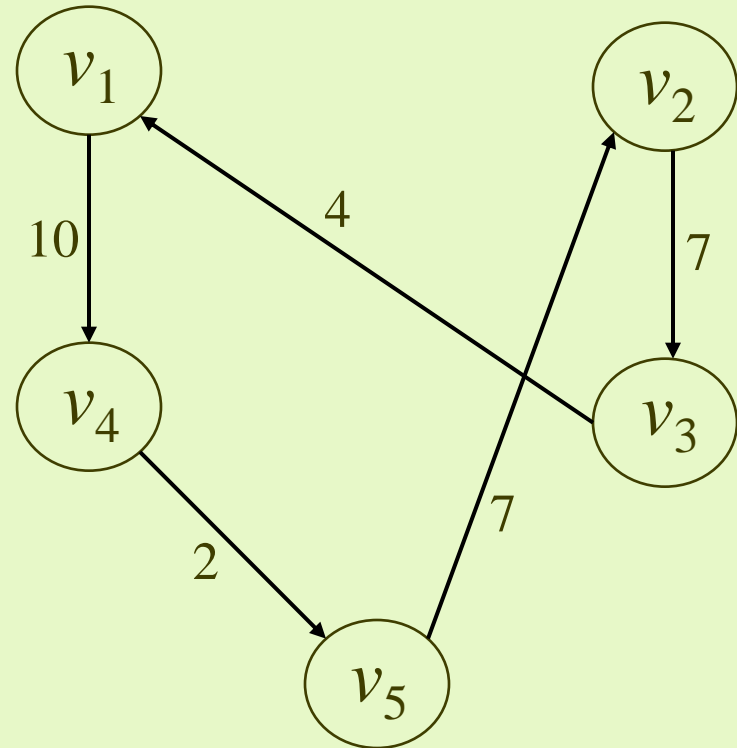
- 외판원의 집이 위치하고 있는 도시에서 출발하여 다른 도시들을 각각 한번씩 만 방문하고, 다시 집으로 돌아오는 가장 짧은 일주여행경로(tour)를 결정하는 문제.
- 이 문제는 음이 아닌 가중치가 있는, 방향성 그래프로 나타낼 수 있다.
- 그래프 상에서 일주여행경로(tour, Hamiltonian circuits)는 한 정점을 출발하여 다른 모든 정점을 한번씩 만 거쳐서 다시 그 정점으로 돌아오는 경로이다.
- 여러 개의 일주여행경로 중에서 길이가 최소가 되는 경로가 최적일주여행경로(optimal tour)가 된다.
- 무작정 알고리즘: 가능한 모든 일주여행경로를 다 고려한 후, 그 중에서 가장 짧은 일주여행경로를 선택한다. 가능한 일주여행경로의 총 개수는  $(n - 1)!$ 이다.

예제: 가장 최적이 되는 일주여행경로는?



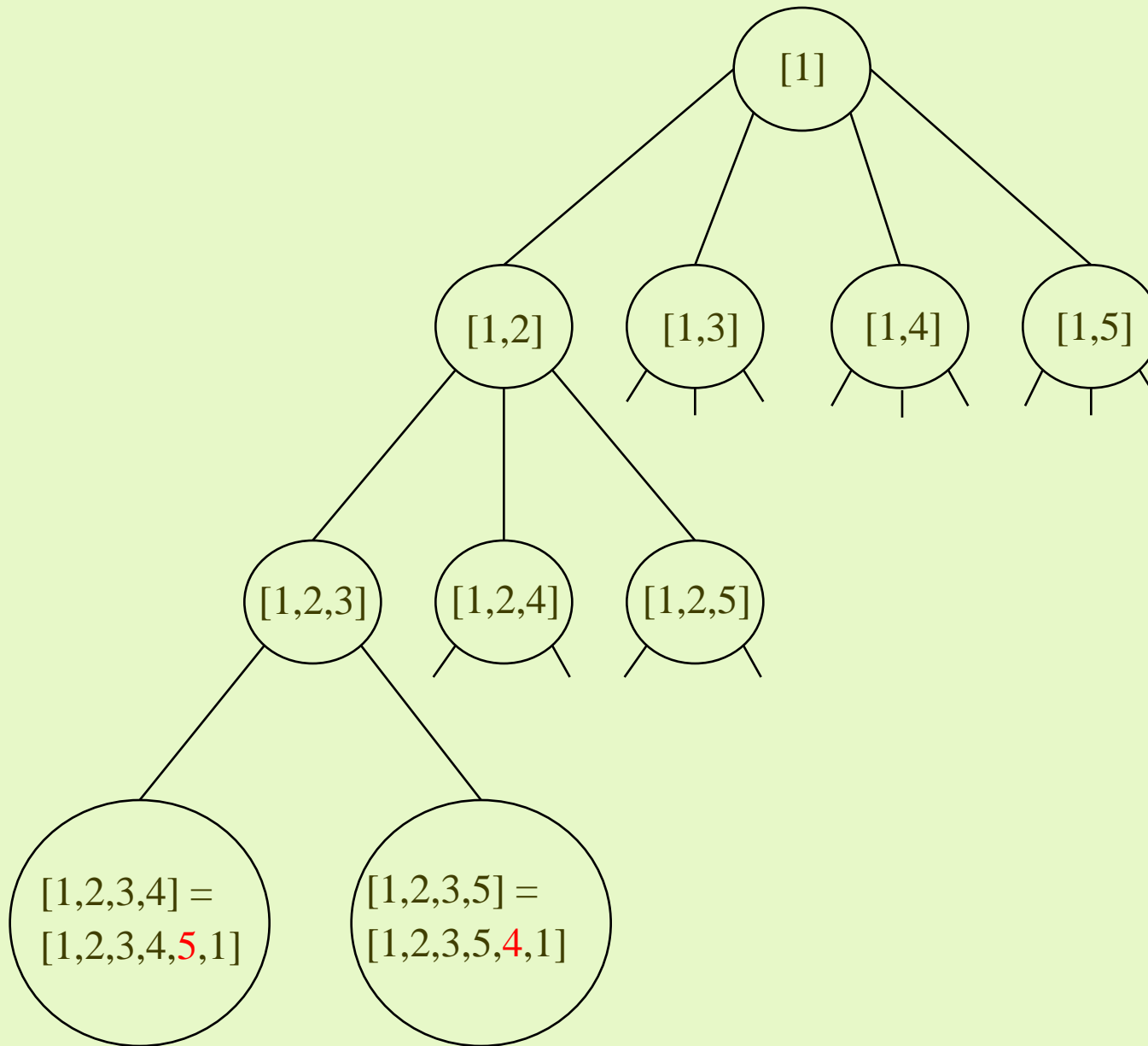
# 외판원문제: 분기한정법

- $n = 40$ 일 때, 동적계획법 알고리즘은 6년 이상이 걸린다. 그러므로 분기한정법을 시도해 본다.
- 보기: 다음 인접행렬로 표현된 그래프를 살펴보세요.

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


# 상태공간트리 구축방법

- 각 마디는 출발마디로부터의 일주여행경로를 나타내게 되는데, 몇 개만 예를 들어 보면, 뿌리마디의 여행경로는 [1]이 되고, 뿌리마디에서 뺀어 나가는 수준 1에 있는 여행경로는 각각 [1,2], [1,3], ..., [1,5]가 되고, 마디 [1,2]에서 뺀어 나가는 수준 2에 있는 마디들의 여행경로는 각각 [1,2,3], ..., [1,2,5]가 되고, 이런 식으로 뺀어 나가서 앞 마디에 도달하게 되면 완전한 일주여행경로를 가지게 된다.
- 따라서 최적일주여행경로를 구하기 위해서는 **앞 마디에 있는 일주여행 경로**를 모두 검사하여 그 중에서 가장 길이가 짧은 일주여행경로를 찾으면 된다.
- 참고: 위 예에서 각 마디에 저장되어 있는 마디가 4개가 되면 더 이상 뺀어 나갈 필요가 없다. 왜냐하면, 남은 경로는 더 이상 뺀어 나가지 않고도 알 수 있기 때문이다.



# 분기한정 최고우선검색

- 분기한정 가지치기로 최고우선 검색을 사용하기 위해서 각 마디의 한계치를 구할 수 있어야 한다. 이 문제에서는 주어진 마디에서 뺀어 나가서 얻을 수 있는 **여행경로의 길이의 하한(최소치)**을 구하여 **한계치**로 한다. 그리고 각 마디를 검색할 때 **최소여행경로의 길이 보다 한계치가 작은 경우 그 마디는 유망하다고** 한다. 최소여행경로의 초기값은  $\infty$ 로 놓는다. 따라서 완전한 여행경로를 처음 얻을 때 까지는 한계치가 무조건 최소여행경로의 길이 보다 작게 되므로 모든 마디는 유망하다.
- 각 마디의 한계치는 어떻게 구하나?

$[1, \dots, k]$ 의 여행경로를 가진 마디의 한계치는 다음과 같이 구한다. Let:

$$A = V - ([1, \dots, k] \text{ 경로에 속한 모든 마디의 집합})$$

bound =  $[1, \dots, k]$  경로 상의 총거리

+  $v_k$ 에서  $A$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치

+  $\sum_{i \in A} (v_i$ 에서  $A \cup \{v_1\} - \{v_i\}$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치)

# 마디 [1]의 bound

$v_1$	$\text{minimum}(14, 4, 10, 20) = 4$	( $v_1$ 을 떠나는 비용)
$v_2$	$\text{minimum}(14, 7, 8, 7) = 7$	( $v_2$ 를 떠나는 비용)
$v_3$	$\text{minimum}(4, 5, 7, 16) = 4$	( $v_3$ 를 떠나는 비용)
$v_4$	$\text{minimum}(11, 7, 9, 2) = 2$	( $v_4$ 를 떠나는 비용)
$v_5$	$\text{minimum}(18, 7, 17, 4) = 4$	( $v_5$ 를 떠나는 비용)

$$\text{bound} : 4 + 7 + 4 + 2 + 4 = 21$$

$[1, \dots, k]$ 의 여행경로를 가진 마디의 한계치는 다음과 같이 구한다. Let:

$A = V - ([1, \dots, k]$  경로에 속한 모든 마디의 집합)

bound =  $[1, \dots, k]$  경로 상의 총거리

+  $v_k$ 에서  $A$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치

+  $\sum_{i \in A} (v_i$ 에서  $A \cup \{v_1\} - \{v_i\}$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치)

# 마디 [1]의 bound

$$A = \{v_2, v_3, v_4, v_5\}$$

$$v_1 \quad \min(v_2, v_3, v_4, v_5) = \min(14, 4, 10, 20) = 4$$

$$v_2 \quad \min(v_1, v_3, v_4, v_5) = \min(14, 7, 8, 7) = 7$$

$$v_3 \quad \min(v_1, v_2, v_4, v_5) = \min(4, 5, 7, 16) = 4$$

$$v_4 \quad \min(v_1, v_2, v_3, v_5) = \min(11, 7, 9, 2) = 2$$

$$v_5 \quad \min(v_1, v_2, v_3, v_4) = \min(18, 7, 17, 4) = 4$$

$$\text{bound} : \quad 4 + 7 + 4 + 2 + 4 = 21$$

$[1, \dots, k]$ 의 여행경로를 가진 마디의 한계치는 다음과 같이 구한다. Let:

$A = V - ([1, \dots, k]$  경로에 속한 모든 마디의 집합)

bound =  $[1, \dots, k]$  경로 상의 총거리

+  $v_k$ 에서  $A$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치

+  $\sum_{i \in A} (v_i$ 에서  $A \cup \{v_1\} - \{v_i\}$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치)

# 마디 [1,2]의 bound

- $v_1$  14 ( $v_1 \rightarrow v_2$  로가는 비용)
- $v_2$   $\text{minimum}(7, 8, 7) = 7$  ( $v_2$ 에서  $A = \{v_3, v_4, v_5\}$ 에 속한 정점으로가는 이음선)
- $v_3$   $\text{minimum}(4, 7, 16) = 4$  ( $v_3$ 에서  $A \cup \{v_1\} - \{v_3\}$ 에 속한 정점으로가는 이음선)
- $v_4$   $\text{minimum}(11, 9, 2) = 2$  ( $v_4$ 에서  $A \cup \{v_1\} - \{v_4\}$ 에 속한 정점으로가는 이음선)
- $v_5$   $\text{minimum}(18, 17, 4) = 4$  ( $v_5$ 에서  $A \cup \{v_1\} - \{v_5\}$ 에 속한 정점으로가는 이음선)

bound:  $14 + 7 + 4 + 2 + 4 = 31$

$[1, \dots, k]$ 의 여행경로를 가진 마디의 한계치는 다음과 같이 구한다. Let:

$$A = V - ([1, \dots, k] \text{ 경로에 속한 모든 마디의 집합})$$

bound =  $[1, \dots, k]$  경로 상의 총거리

+  $v_k$ 에서  $A$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치

+  $\sum_{i \in A} (v_i$ 에서  $A \cup \{v_1\} - \{v_i\}$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치)

# 마디 [1,2]의 bound

$$A = \{v_3, v_4, v_5\}$$

$$v_1 \quad 14 \quad (v_1 \rightarrow v_2 \text{ 로가는 비용})$$

$$v_2 \quad \min(v_3, v_4, v_5) = \min(7, 8, 7) = 7 \quad (v_2 \text{에서 } A = \{v_3, v_4, v_5\} \text{에 속한 정점으로가는 이음선})$$

$$v_3 \quad \min(v_1, v_4, v_5) = \min(4, 7, 16) = 4 \quad (v_3 \text{에서 } A \cup \{v_1\} - \{v_3\} \text{에 속한 정점으로가는 이음선})$$

$$v_4 \quad \min(v_1, v_3, v_5) = \min(11, 9, 2) = 2 \quad (v_4 \text{에서 } A \cup \{v_1\} - \{v_4\} \text{에 속한 정점으로가는 이음선})$$

$$v_5 \quad \min(v_1, v_3, v_4) = \min(18, 17, 4) = 4 \quad (v_5 \text{에서 } A \cup \{v_1\} - \{v_5\} \text{에 속한 정점으로가는 이음선})$$

$$\text{bound: } 14 + 7 + 4 + 2 + 4 = 31$$

$[1, \dots, k]$ 의 여행경로를 가진 마디의 한계치는 다음과 같이 구한다. Let:

$$A = V - ([1, \dots, k] \text{ 경로에 속한 모든 마디의 집합})$$

$$\text{bound} = [1, \dots, k] \text{ 경로 상의 총거리}$$

+  $v_k$ 에서  $A$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치

+  $\sum_{i \in A} (v_i \text{에서 } A \cup \{v_1\} - \{v_i\} \text{에 속한 정점으로 가는 이음선의 길이들 중 에서 최소치})$

# 마디 [1,2,3]의 bound

$v_1$	14	( $v_1 \rightarrow v_2$ 로 가는 비용)
$v_2$	7	( $v_2 \rightarrow v_3$ 으로 가는 비용)
$v_3$	$\text{minimum}(7,16) = 7$	( $v_3$ 에서 $A = \{v_4, v_5\}$ 에 속한 정점으로 가는 이음선)
$v_4$	$\text{minimum}(11, 2) = 2$	( $v_4$ 에서 $A \cup \{v_1\} - \{v_4\}$ 에 속한 정점으로 가는 이음선)
$v_5$	$\text{minimum}(18, 4) = 4$	( $v_5$ 에서 $A \cup \{v_1\} - \{v_5\}$ 에 속한 정점으로 가는 이음선)

bound:  $14 + 7 + 7 + 2 + 4 = 34$

$[1, \dots, k]$ 의 여행경로를 가진 마디의 한계치는 다음과 같이 구한다. Let:

$A = V - ([1, \dots, k]$  경로에 속한 모든 마디의 집합)

bound =  $[1, \dots, k]$  경로 상의 총거리

+  $v_k$ 에서  $A$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치

+  $\sum_{i \in A} (v_i$ 에서  $A \cup \{v_1\} - \{v_i\}$ 에 속한 정점으로 가는 이음선의 길이들 중  
에서 최소치)

# 마디 [1,2,3]의 bound

$$A = \{v_4, v_5\}$$

$$v_1 \quad 14 \quad (v_1 \rightarrow v_2 \text{ 로가는 비용})$$

$$v_2 \quad 7 \quad (v_2 \rightarrow v_3 \text{ 으로가는 비용})$$

$$v_3 \quad \min(v_4, v_5) = \min(7, 16) = 7 \quad (v_3 \text{에서 } A = \{v_4, v_5\} \text{에 속한 정점으로 가는 이음선})$$

$$v_4 \quad \min(v_1, v_5) = \min(11, 2) = 2 \quad (v_4 \text{에서 } A \cup \{v_1\} - \{v_4\} \text{에 속한 정점으로 가는 이음선})$$

$$v_5 \quad \min(v_1, v_4) = \min(18, 4) = 4 \quad (v_5 \text{에서 } A \cup \{v_1\} - \{v_5\} \text{에 속한 정점으로 가는 이음선})$$

$$\text{bound: } 14 + 7 + 7 + 2 + 4 = 34$$

$[1, \dots, k]$ 의 여행경로를 가진 마디의 한계치는 다음과 같이 구한다. Let:

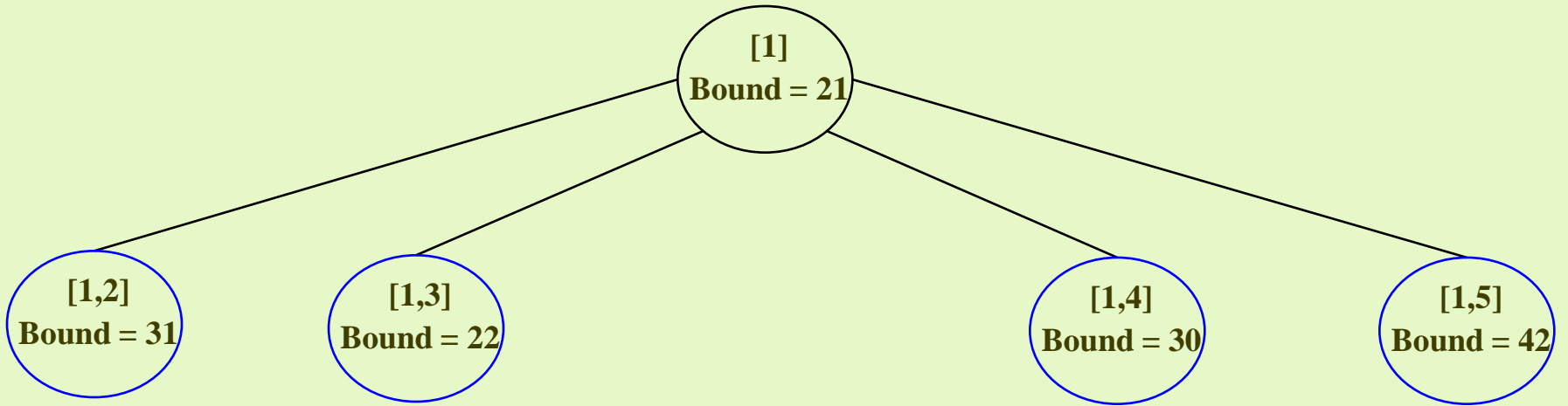
$$A = V - ([1, \dots, k] \text{ 경로에 속한 모든 마디의 집합})$$

bound =  $[1, \dots, k]$  경로 상의 총거리

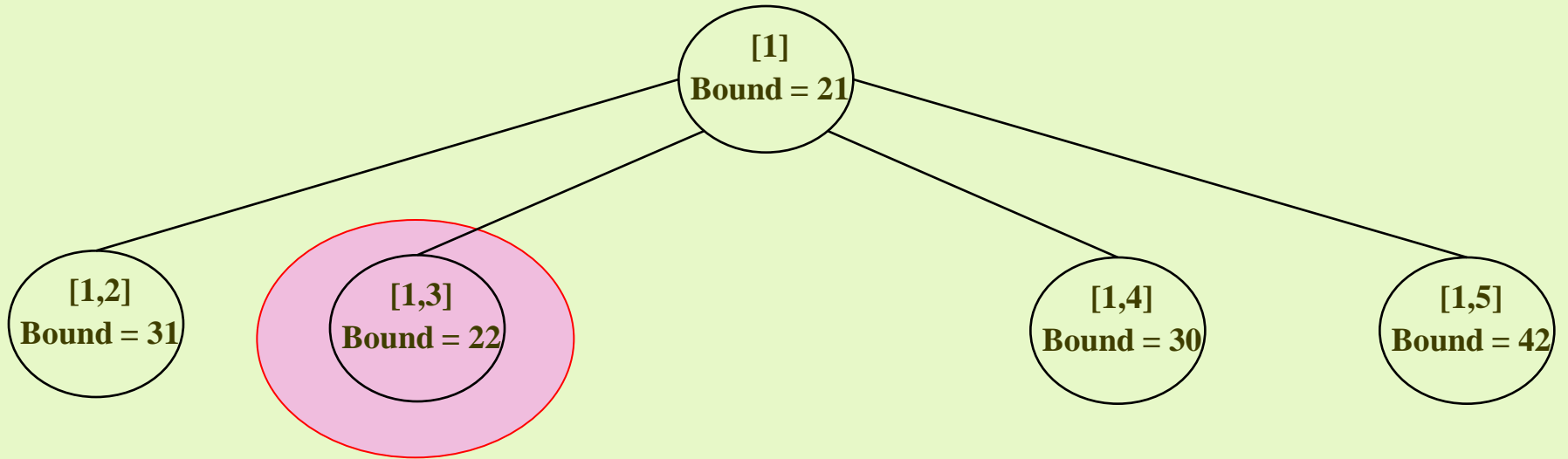
+  $v_k$ 에서  $A$ 에 속한 정점으로 가는 이음선의 길이들 중에서 최소치

+  $\sum_{i \in A} (v_i \text{에서 } A \cup \{v_1\} - \{v_i\} \text{에 속한 정점으로 가는 이음선의 길이들 중 에서 최소치})$

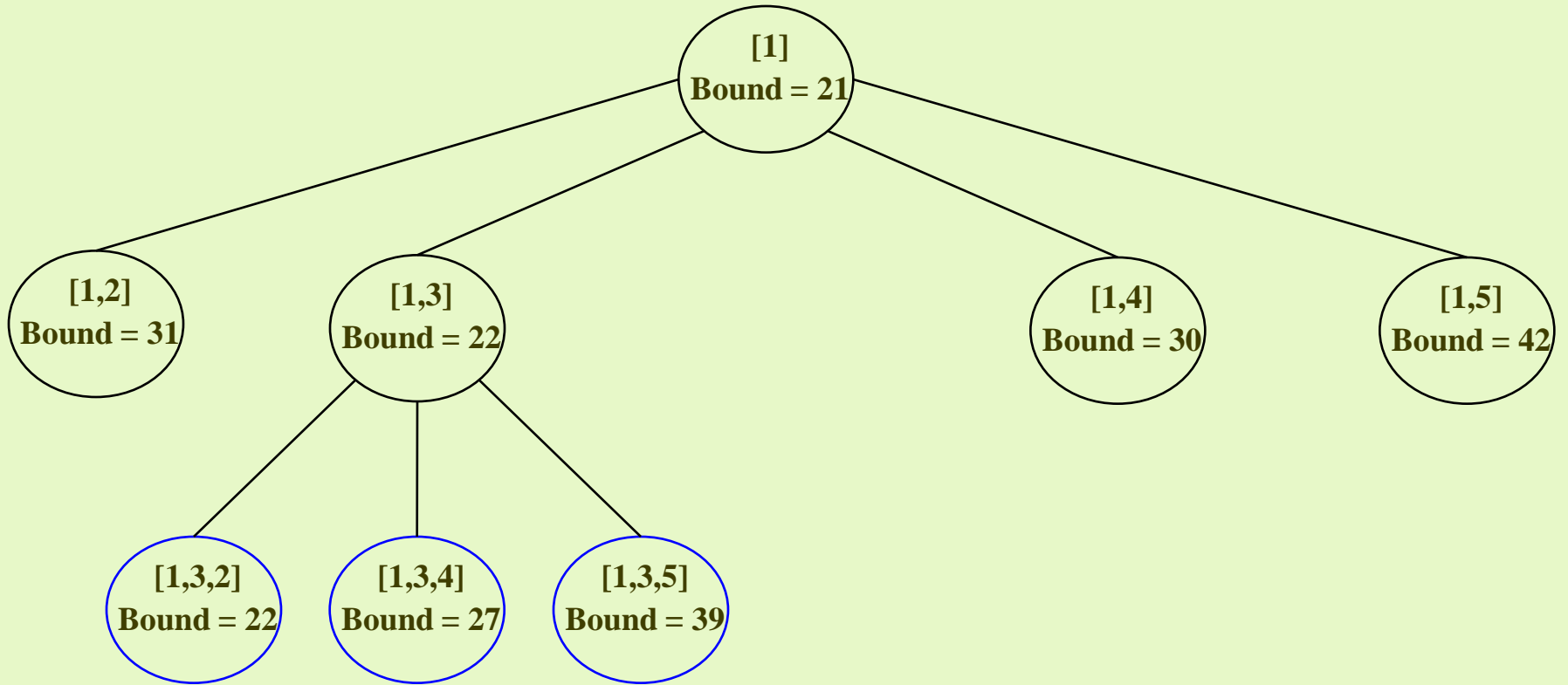
- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오



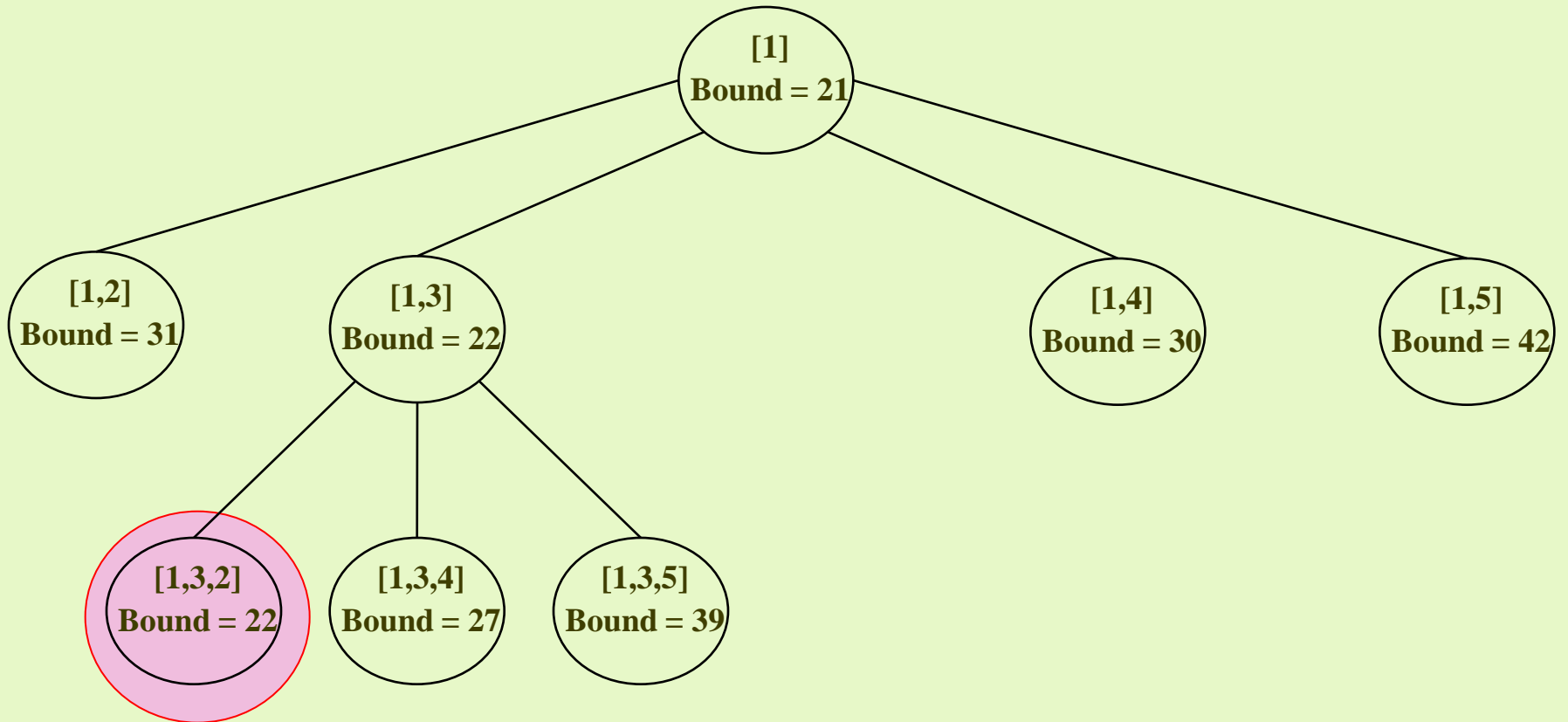
- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오



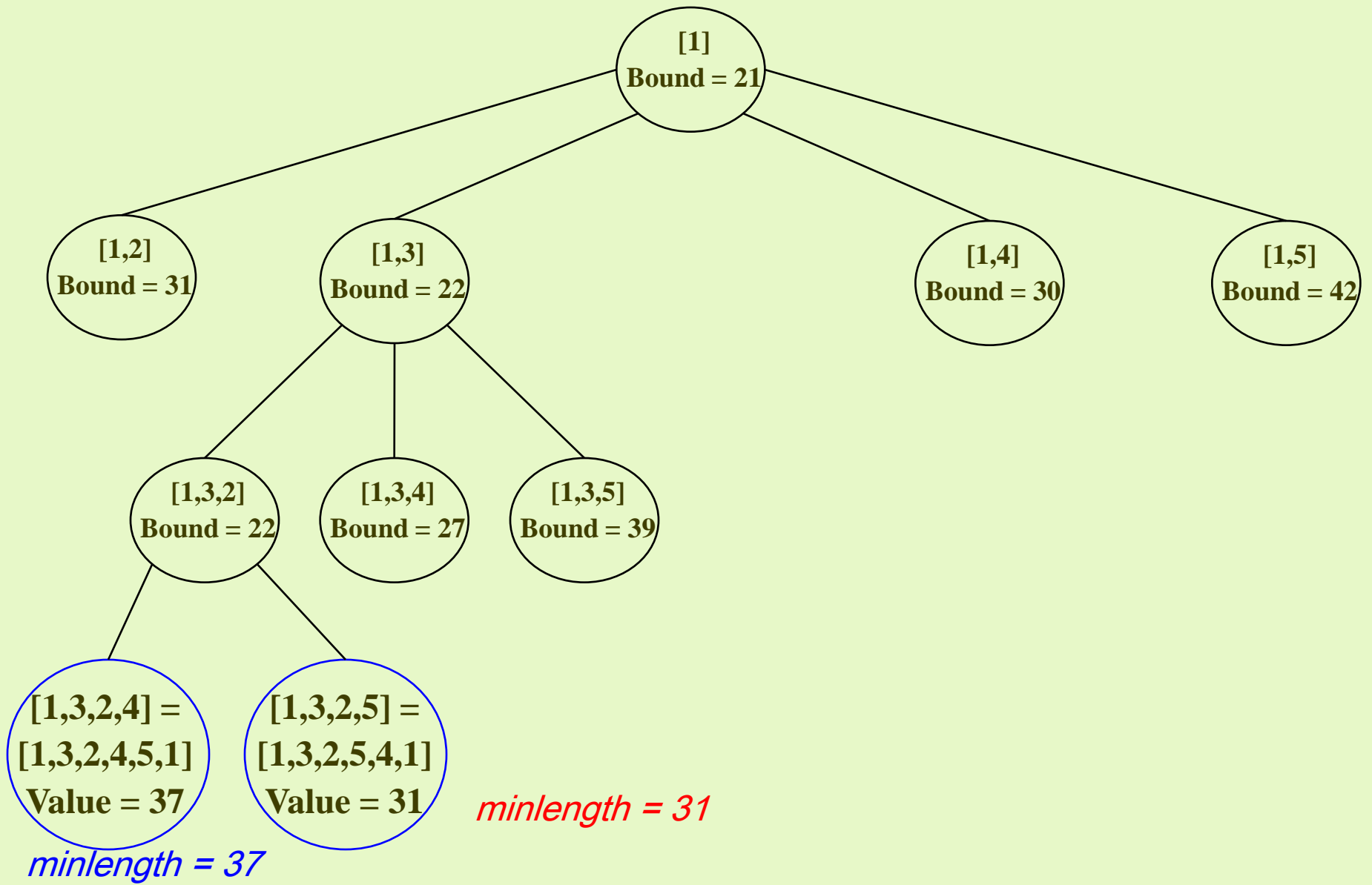
- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오



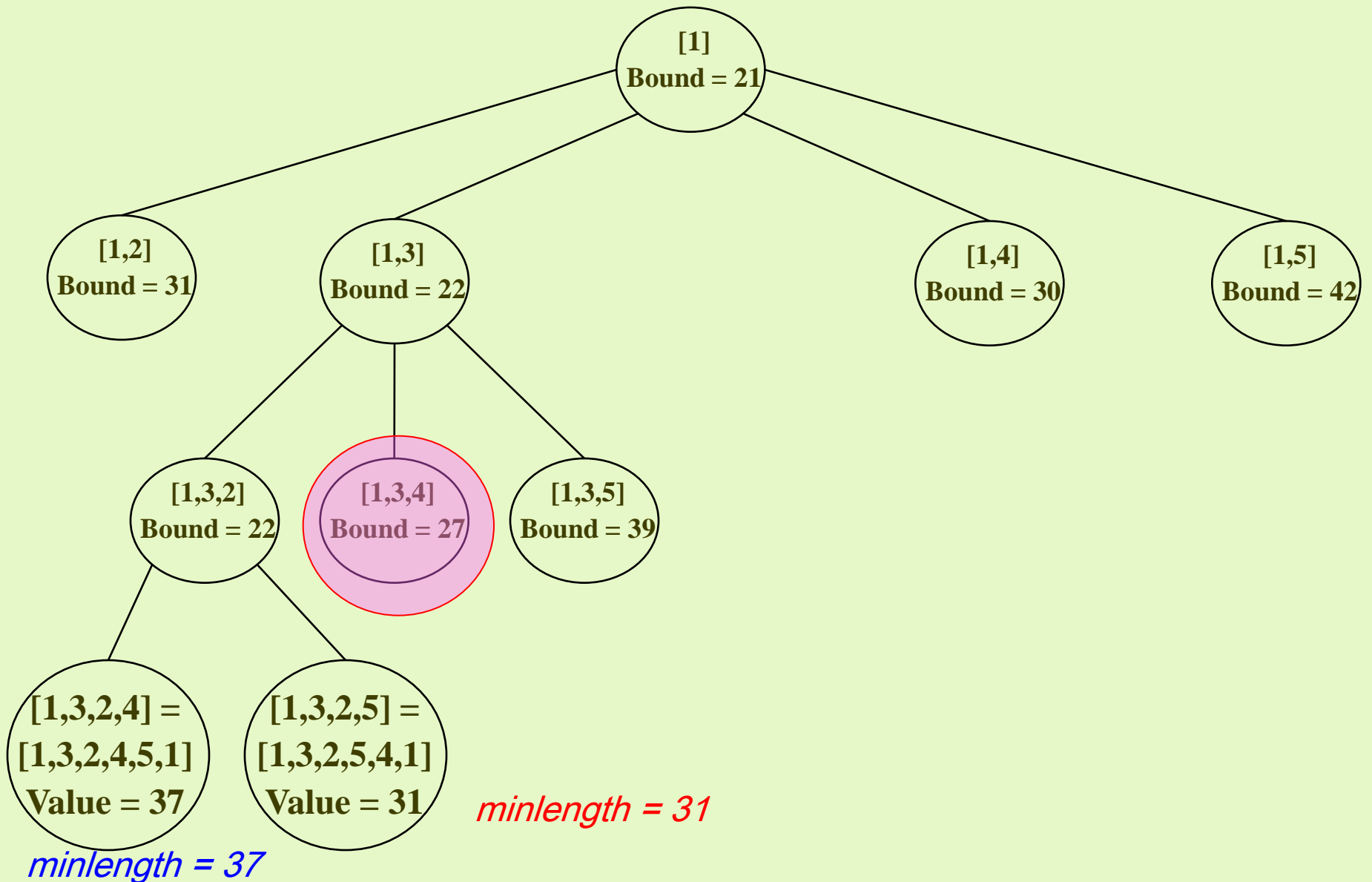
- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오



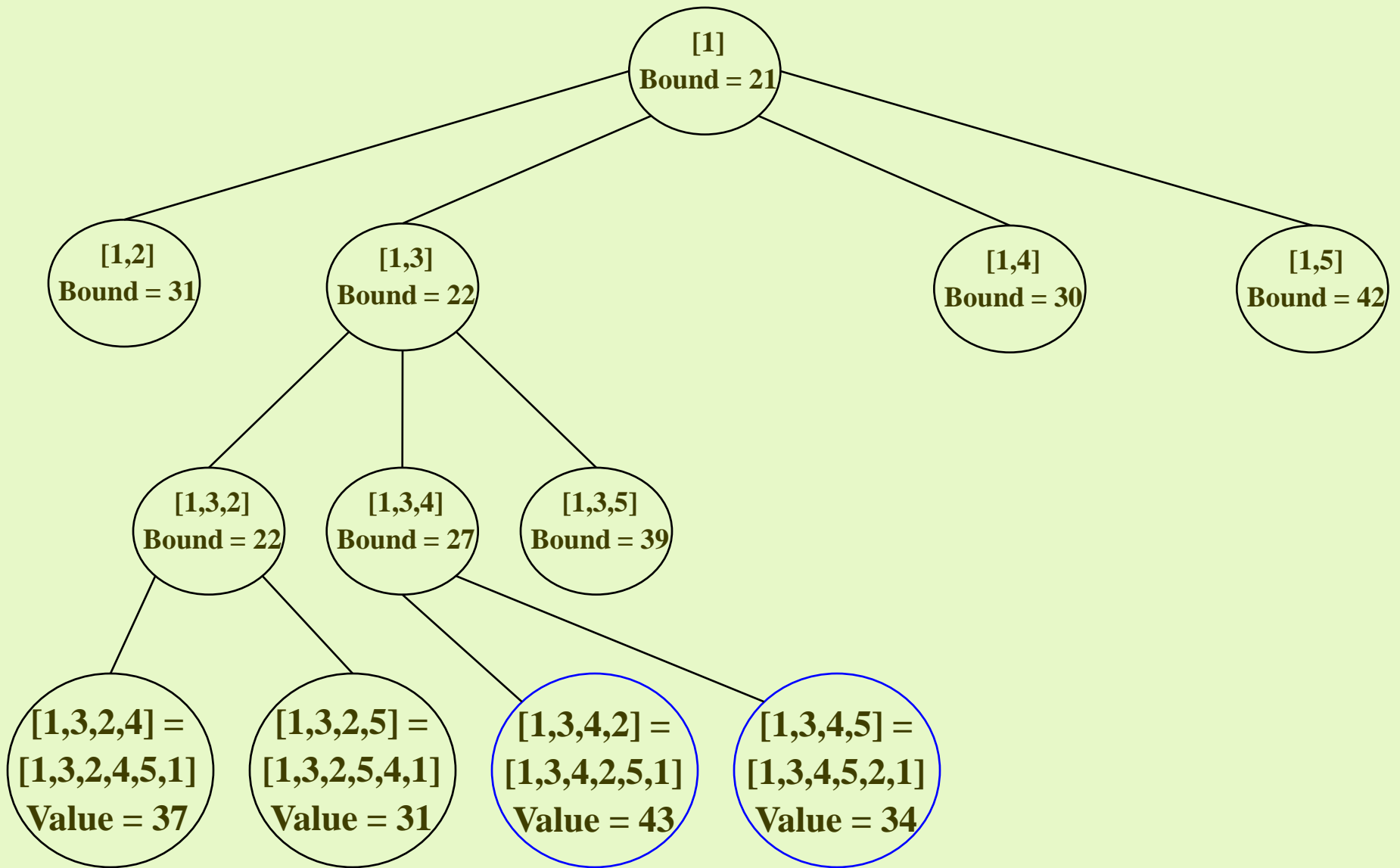
- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오



- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오

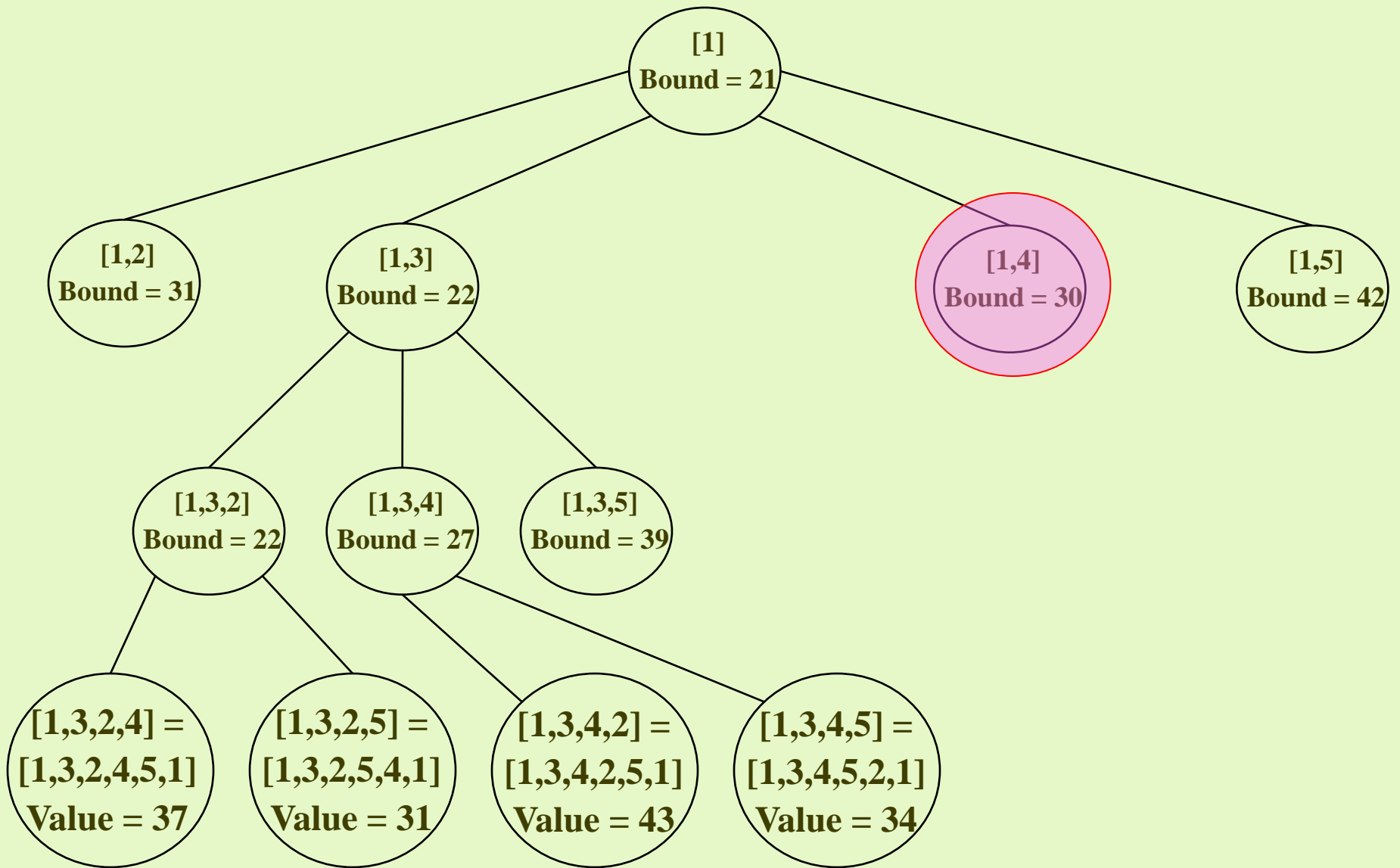


- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오



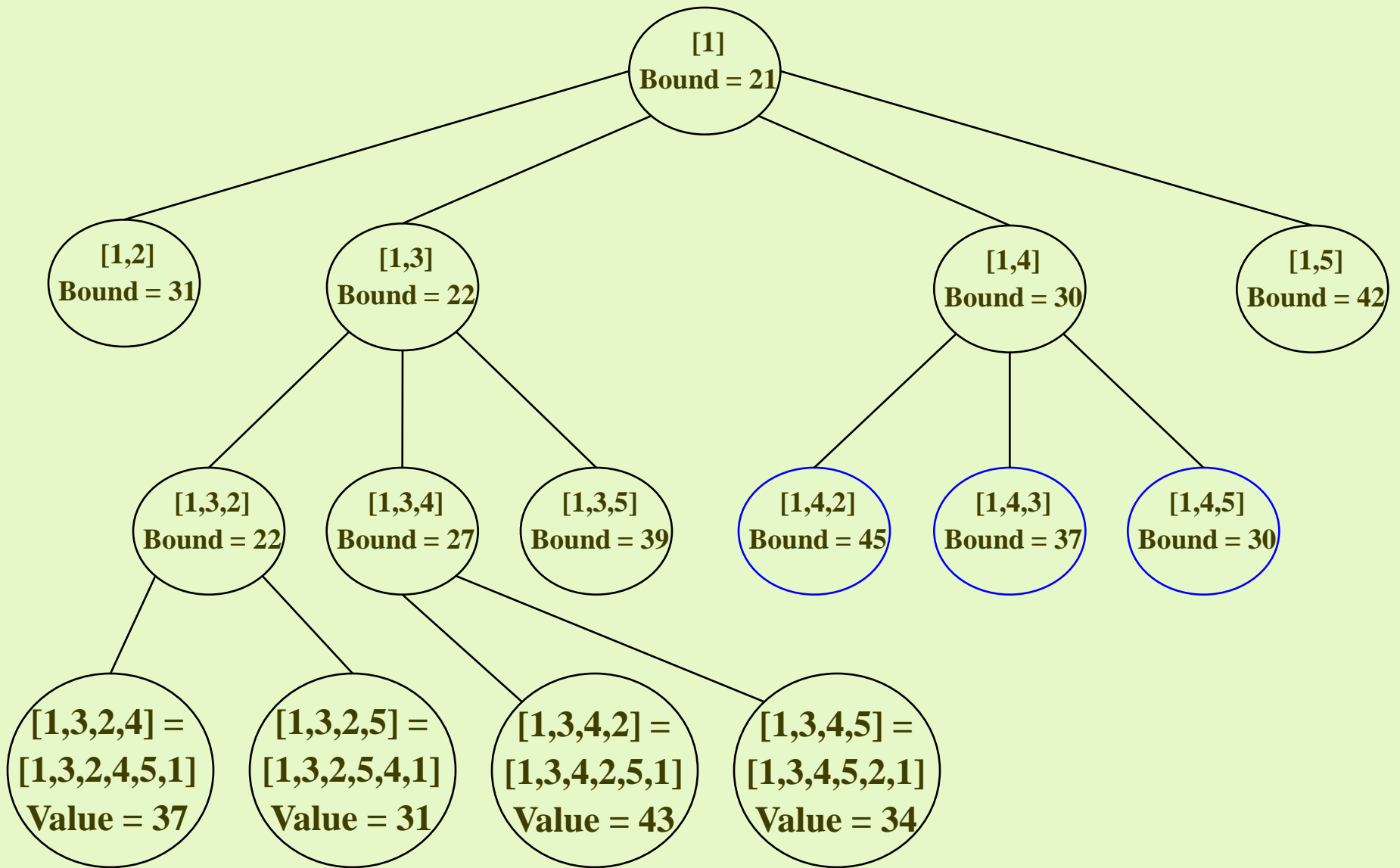
*minlength = 31*

- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오



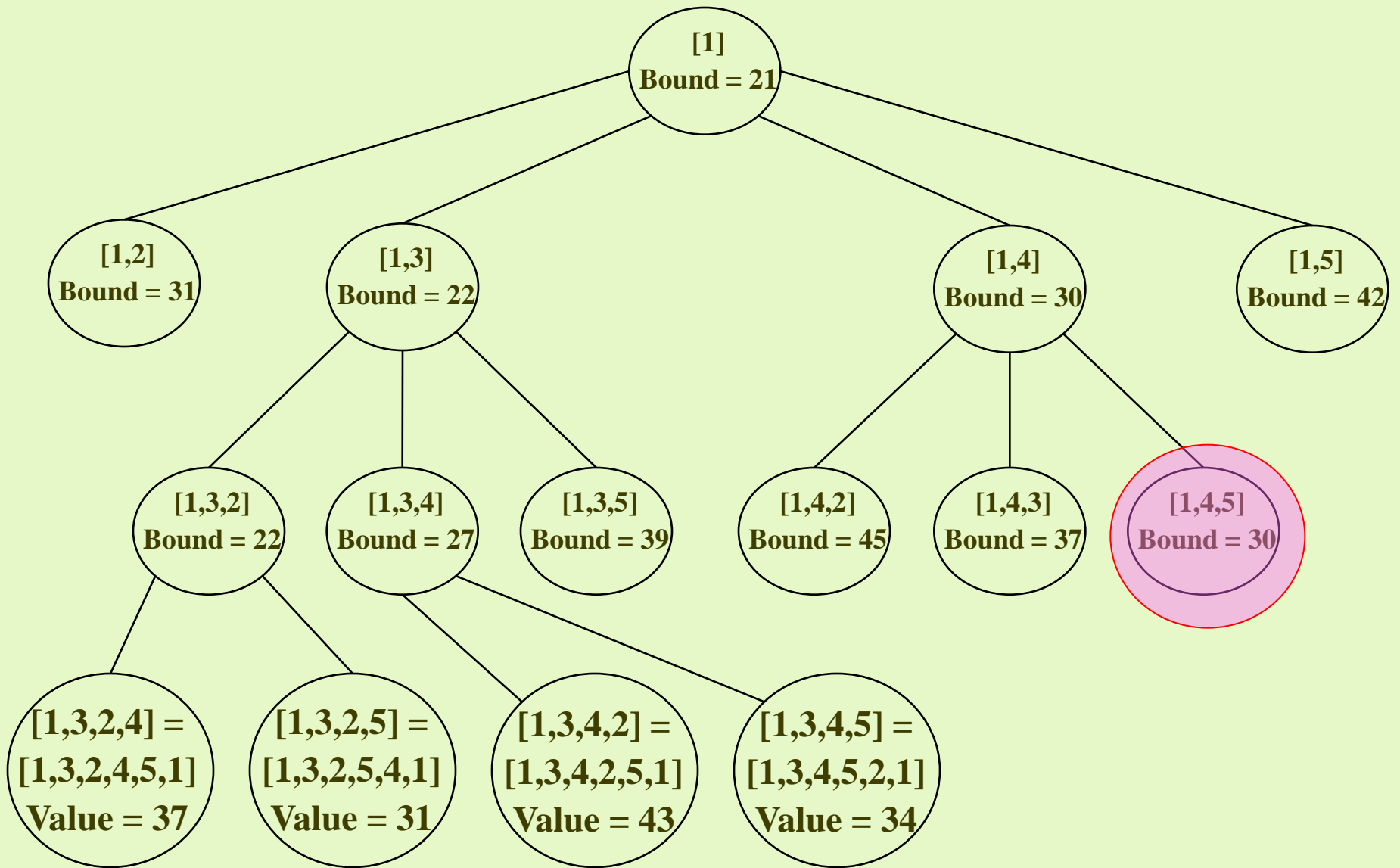
*minlength = 31*

- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오



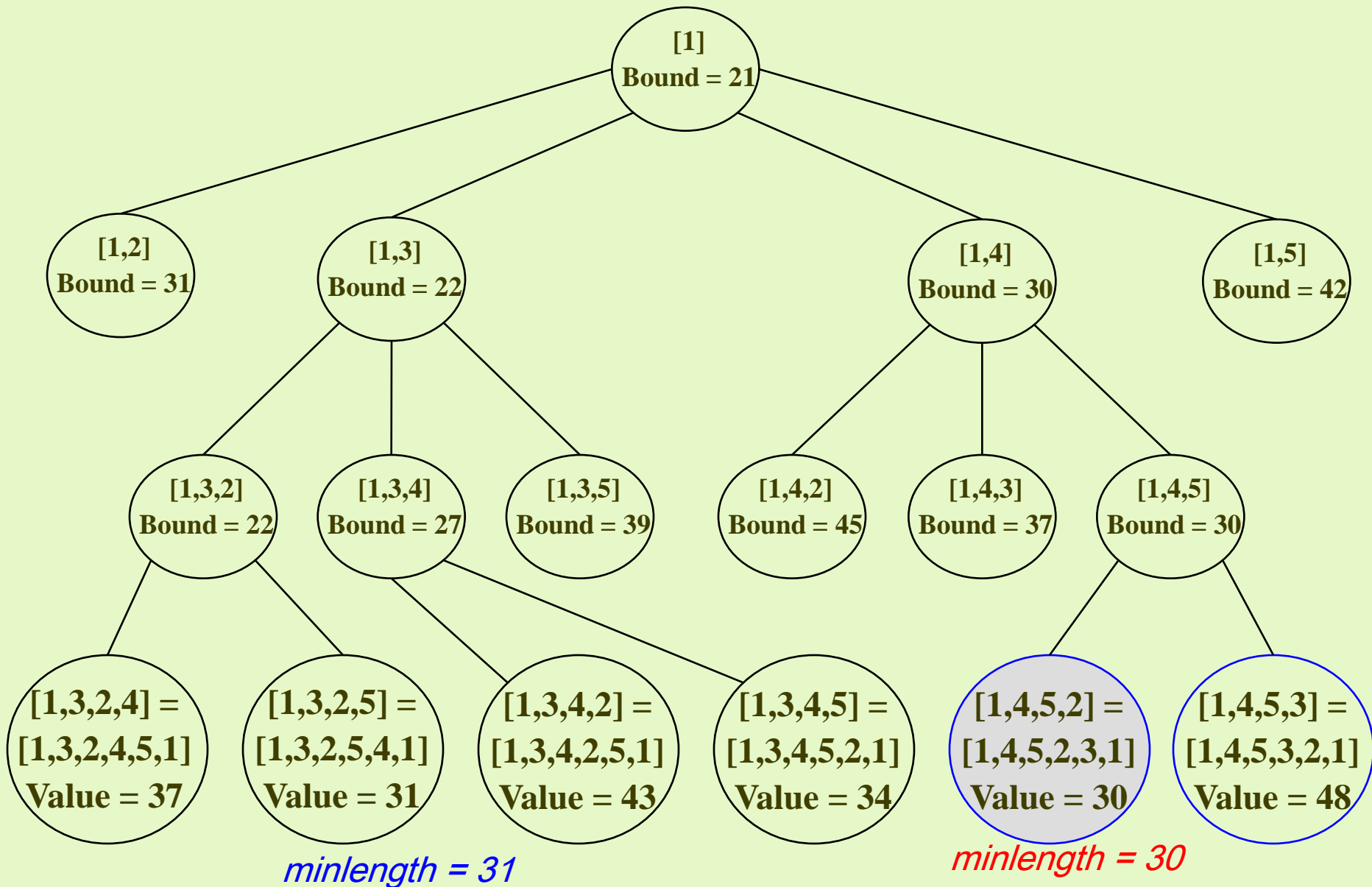
*minlength = 31*

- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오

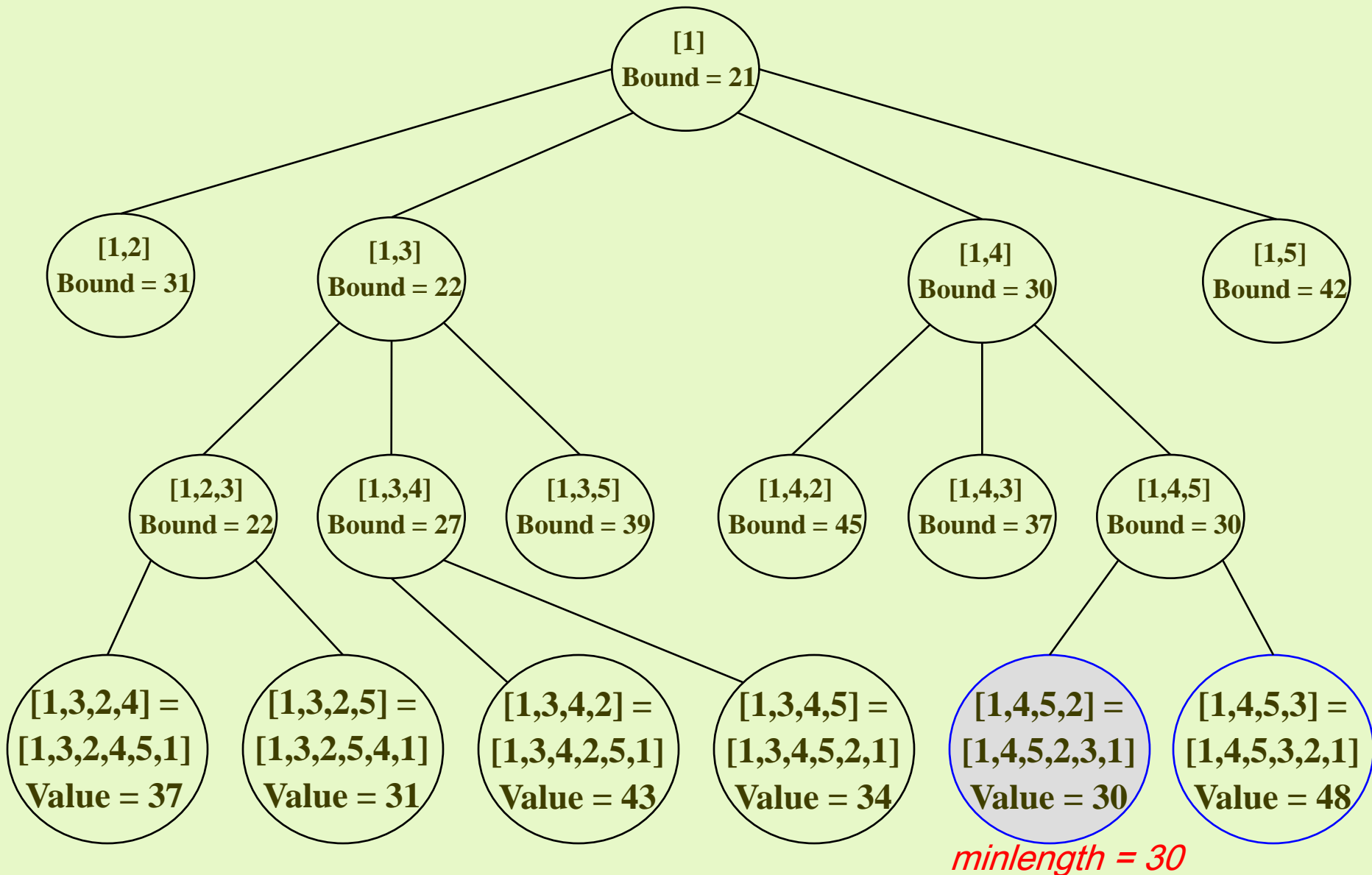


*minlength = 31*

- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오



- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오



```

void travel2 (int n, const number W[][], ordered-set & opttour, number & minlength)
{
    priority_queue_of_node PQ;
    node u, v;

    initialize(PQ);
    v.level = 0;
    v.path = [1];
    v.bound = bound(v);
    minlength = infinite_value;
    insert(PQ, v);
    while (!Empty(PQ)) {
        remove(PQ, v);
        if (v.bound < minlength) {
            u.level = v.level + 1;
            for (all i such that 2 <= i <= n && i is not in v.path) {
                u.path = v.path;
                put i at the end of u.path;
                if (u.level == n-2) {
                    put index of only vertex not in u.path at the end of u.path;
                    put 1 at the end of u.path;
                    if (length(u) < minlength) {
                        minlength = length(u);
                        opttour = u.path;
                    }
                } else {
                    u.bound = bound(u);
                    if (u.bound < minlength)
                        insert(PQ, u);
                }
            } // end of for
        } // end of if
    } // end of while
}

```

# 분석

- 이 알고리즘은 방문하는 마디의 개수가 더 적다.
- 그러나 아직도 알고리즘의 시간복잡도는 지수적이거나 그보다 못하다!
- 다시 말해서  $n = 40$ 이 되면 문제를 풀 수 없는 것과 다름없다고 할 수 있다.
- 다른 방법이 있을까?
  - ✓ 근사(approximation) 알고리즘: 최적의 해답을 준다는 보장은 없지만, 무리 없이 최적에 가까운 해답을 주는 알고리즘.