14. State-Space Search

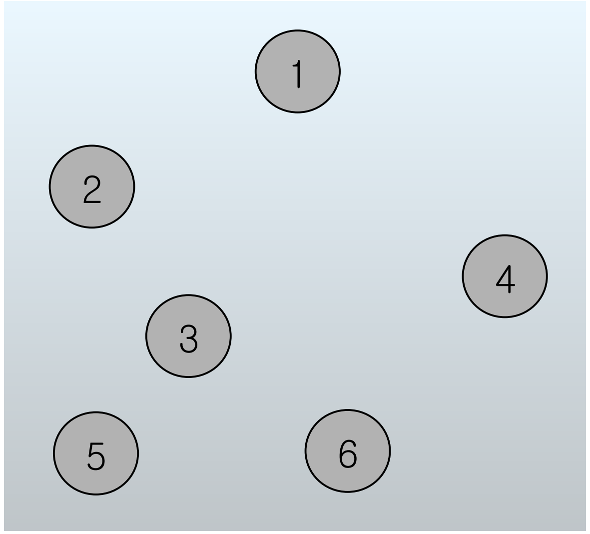
Goals

- Understand state-space search.
- Understand state-space tree.
- Learn backtracking.
- Learn branch-and-bound.
- Learn A* algorithm.

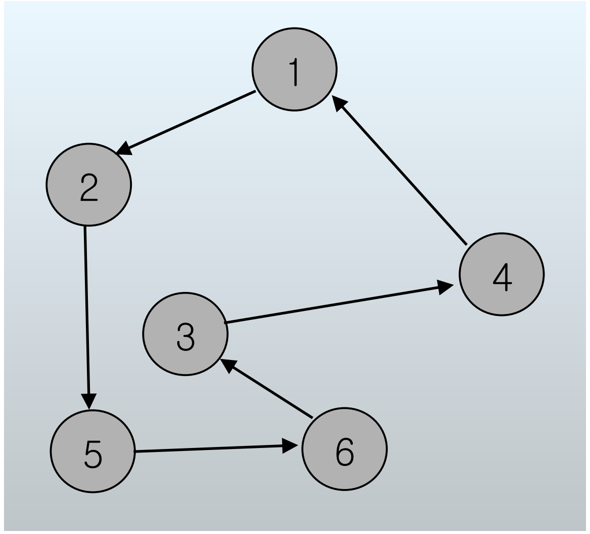
State-Space Tree

- State space: set of states that are generated in problem solving process
- State-space tree: tree where each node represents a state of problem solving process
- Search techniques for state space
  - Backtracking
  - Branch-and-bound
  - A^* algorithm

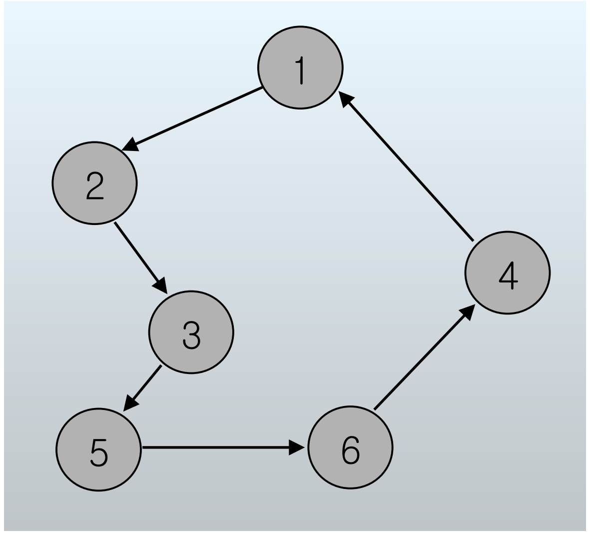
TSP



(a) instance of TSP



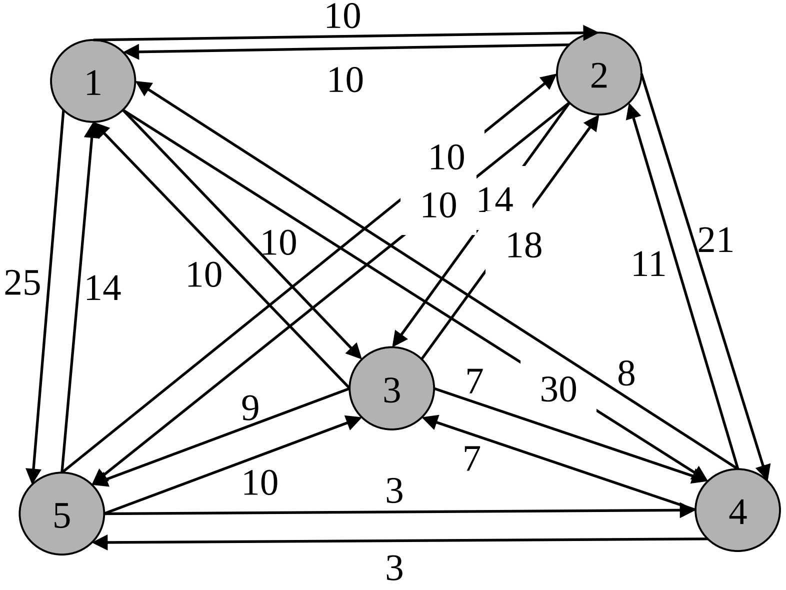
(b) a solution



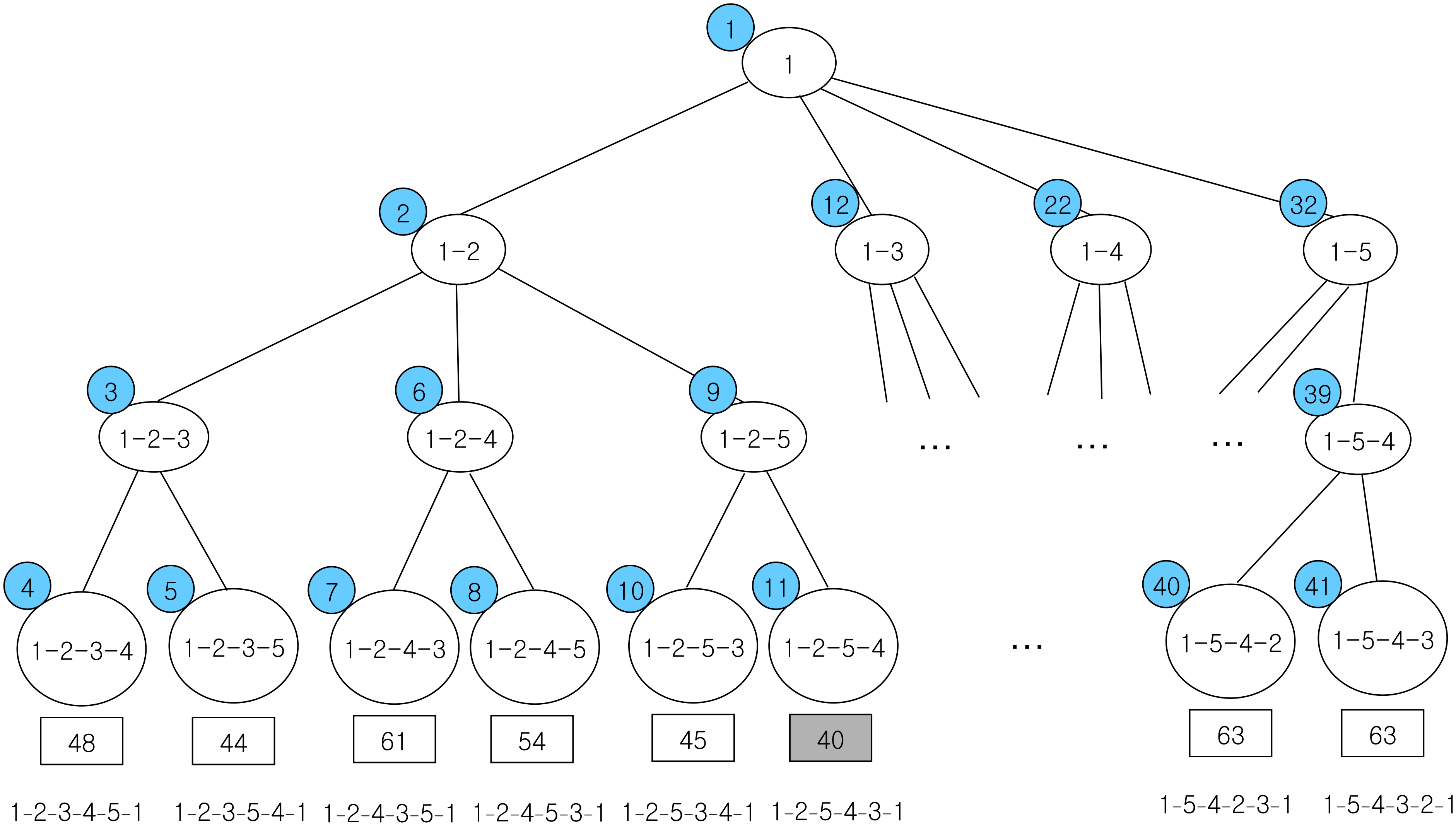
(c) optimal solution

TSP and Adjacency Matrix

	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



Lexicographic order search of state space

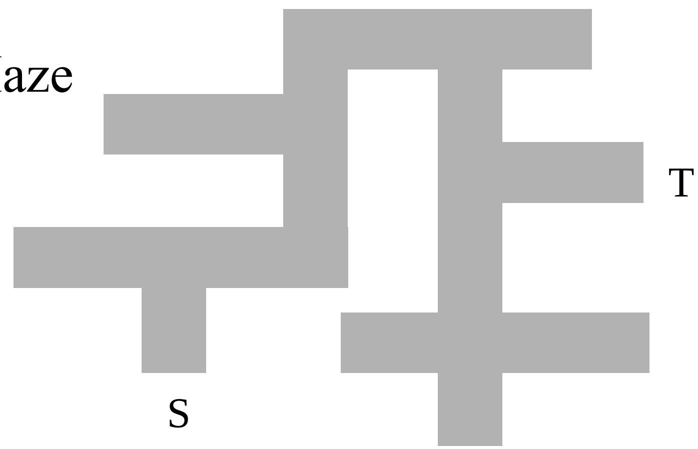


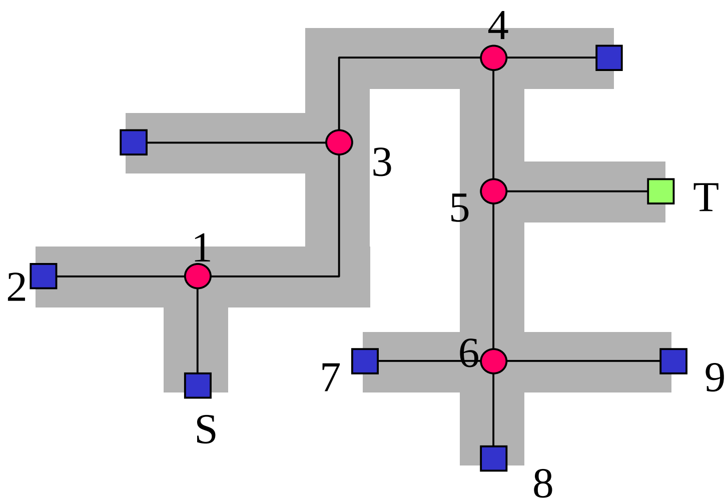
Backtracking

- Refers to DFS-like search
- Go as deeply as possible, backtrack if impossible
- Examples
  - Maze search, 8-Queens problem, map coloring, …

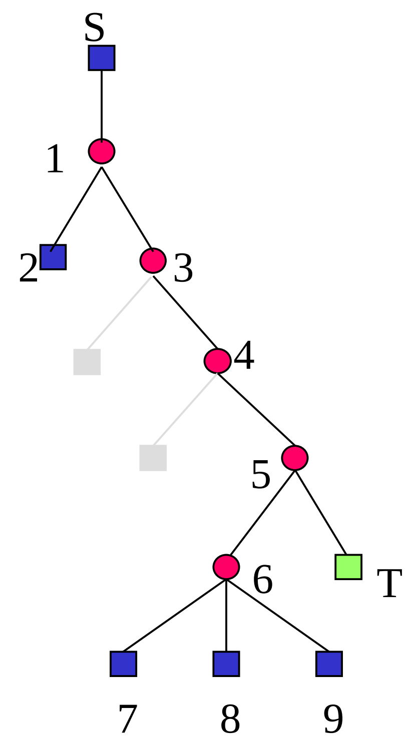
Maze Search

(a) Maze





(b) Graph modeling of maze



(c) Maze tree

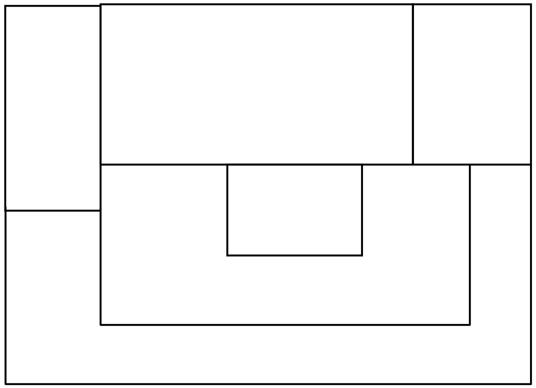
Backtracking Algorithm for Maze Search

```
maze(v)
{
        visited[v] ← YES;
        if (v = T) then {print "success!";}    ▷ terminate
        for each x ∈ L(v)                      ▷ L(v): vertices adjacent to v
                if (visited[x] = NO) then {
                        prev[x] ← v;
                        maze(x);
                }
}
```

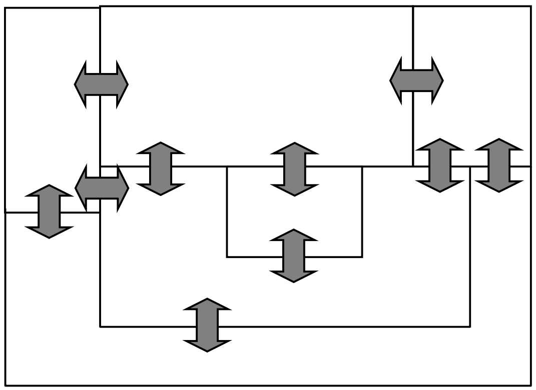
Graph Coloring

- k-coloring of a graph
  - Adjacent vertices cannot be colored with the same color
  - Can the graph be colored with *k* colors?

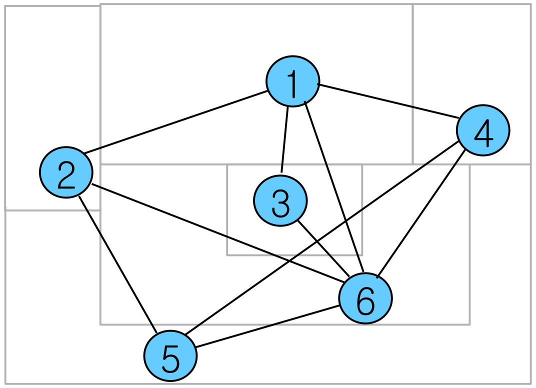
Map Coloring



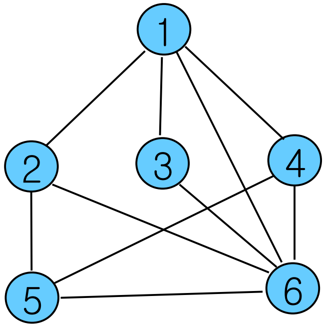
(a) Map



(b) Adjacent regions



(c) Graph modeling of map

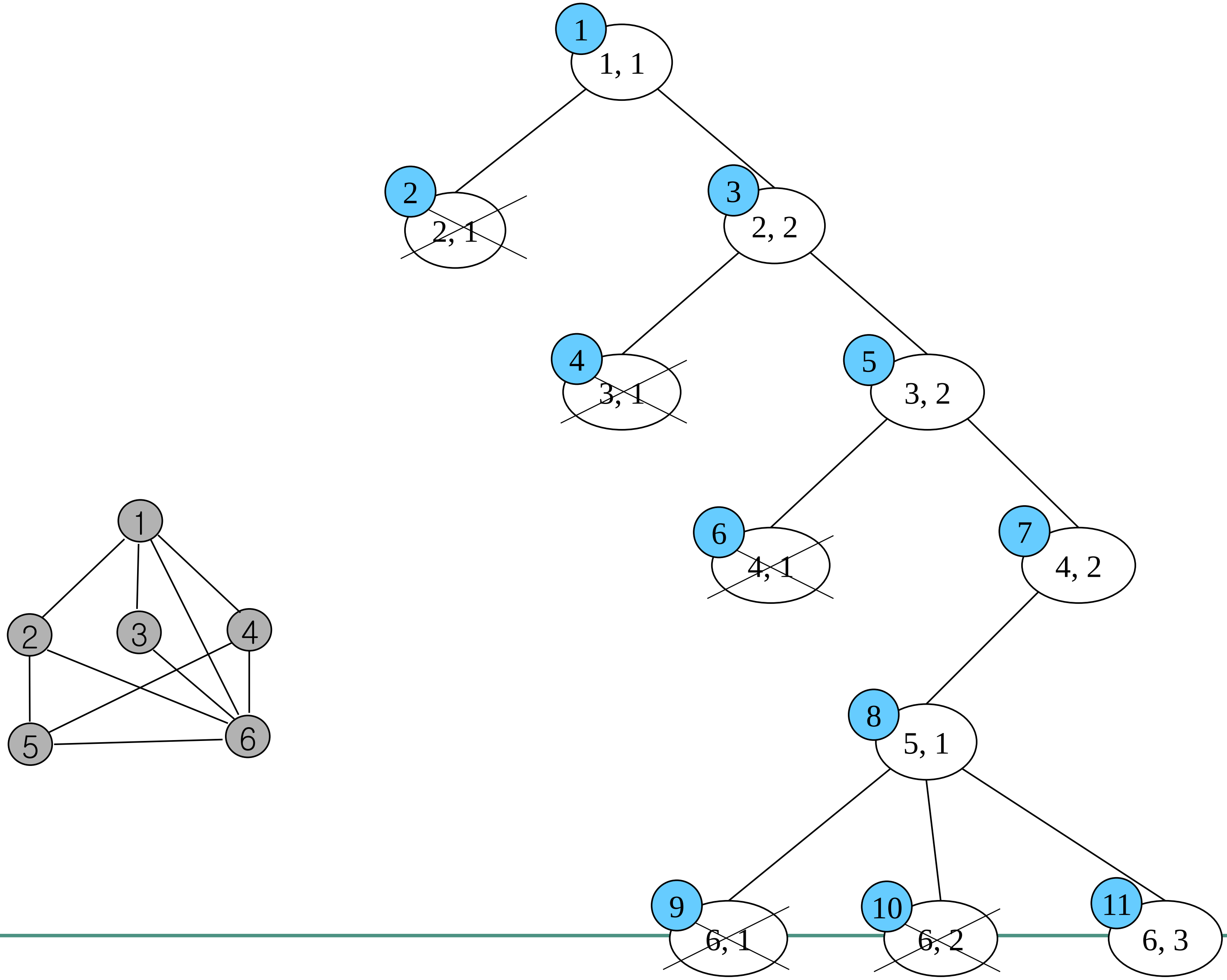


(d) Graph of (c)

```
kColoring(i, c)
▷ i: vertex, c: color
▷ When vertices 1…i-1 are colored, can we color vertex i with color c?
{
    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: next vertex
                            d++;
                    }
            }
            return result;
    } else {return FALSE;}
}
```

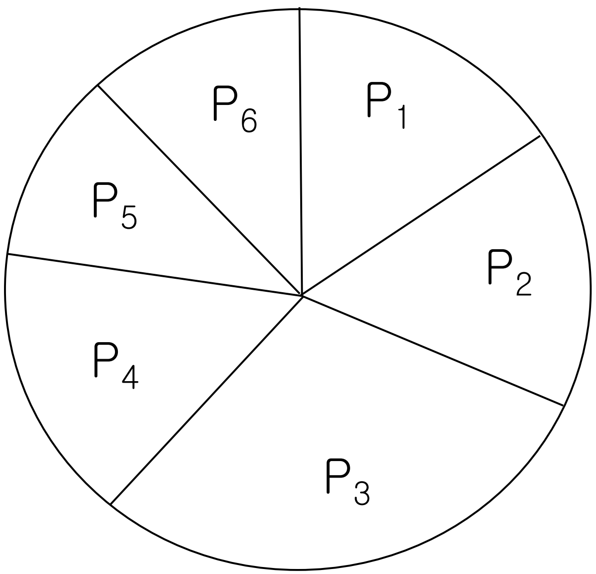
```
valid(i, c)
▷ i: vertex, c: color
▷ When vertices 1…i-1 are colored, can we color vertex i with color c?
{
    for j ← 1 to i-1 {
            ▷ No if there is an edge (i, j) and i, j have the same color
            if ((i, j) ∈ E and color[j] = c) then return FALSE;
    }
    return TRUE;
}
```

State-Space Tree of Backtracking Algorithm

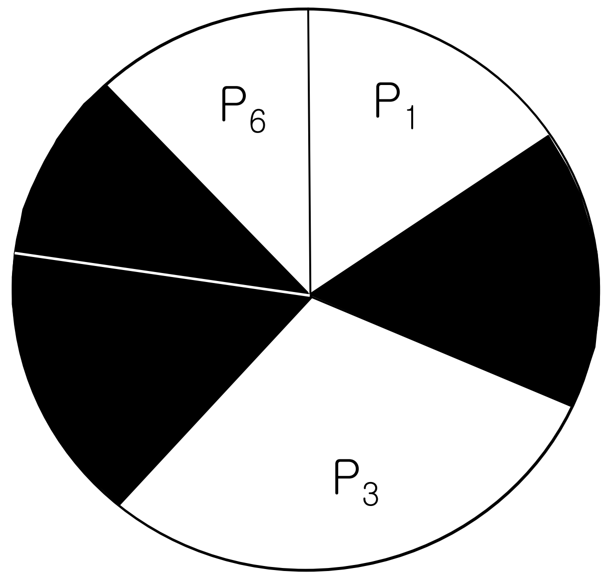


Branch-and-Bound

- Combination of ‘branch’ and ‘bound’
  - Save time by bounding branchings.
- Comparison with backtracking
  - common
    - Require a method to list cases
  - different
    - Backtracking – backtrack when there is no further way to go
    - Branch-and-bound – don’t branch if it is guaranteed that there is no optimal solution in that branch



(a) Choices at one point

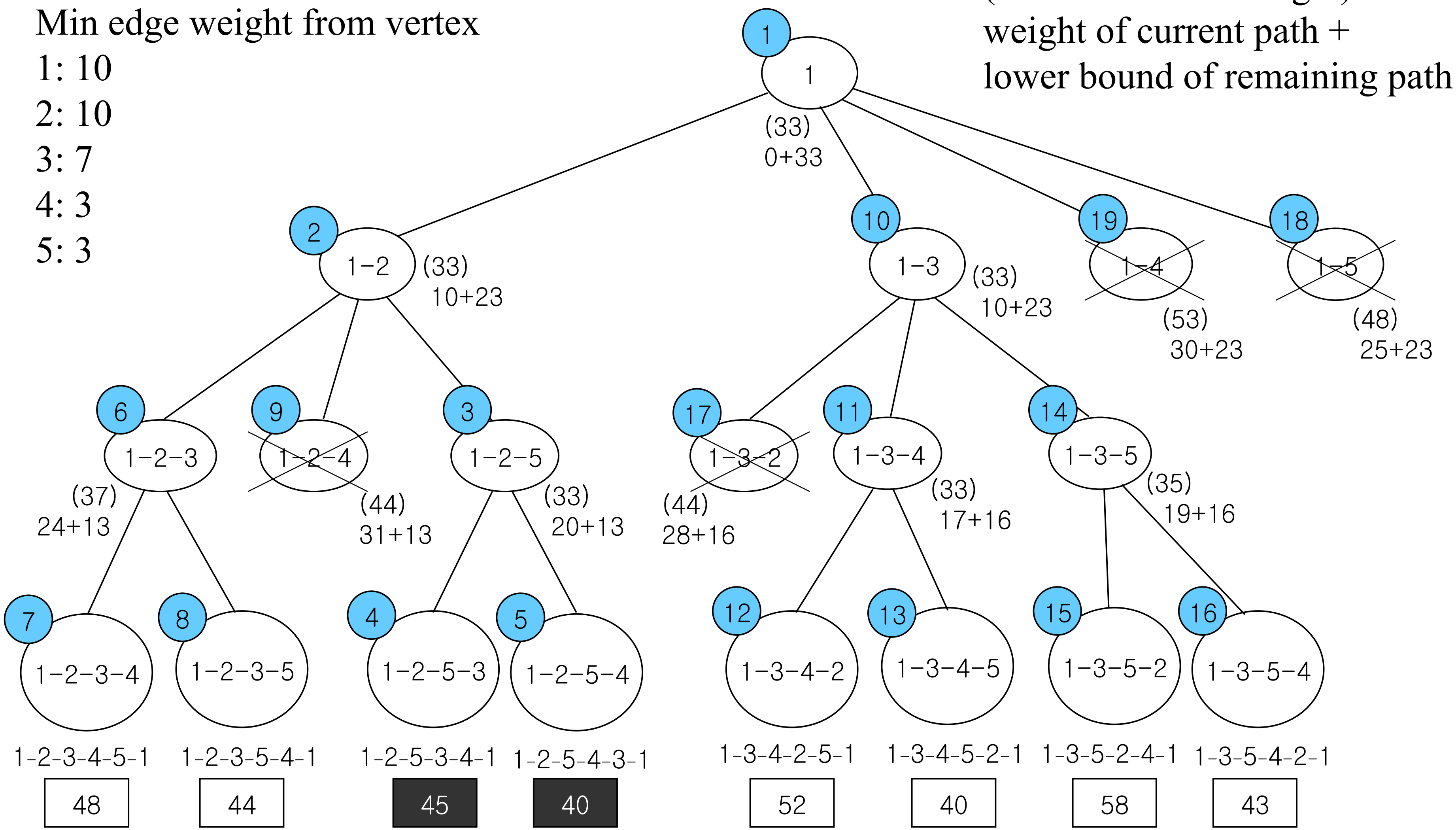


(b) Choices that don’t contain optimal solutions are excluded

State-Space Tree of Branch-and-Bound for TSP

Min edge weight from vertex
1: 10
2: 10
3: 7
4: 3
5: 3

(lower bound of weight)
weight of current path +
lower bound of remaining path



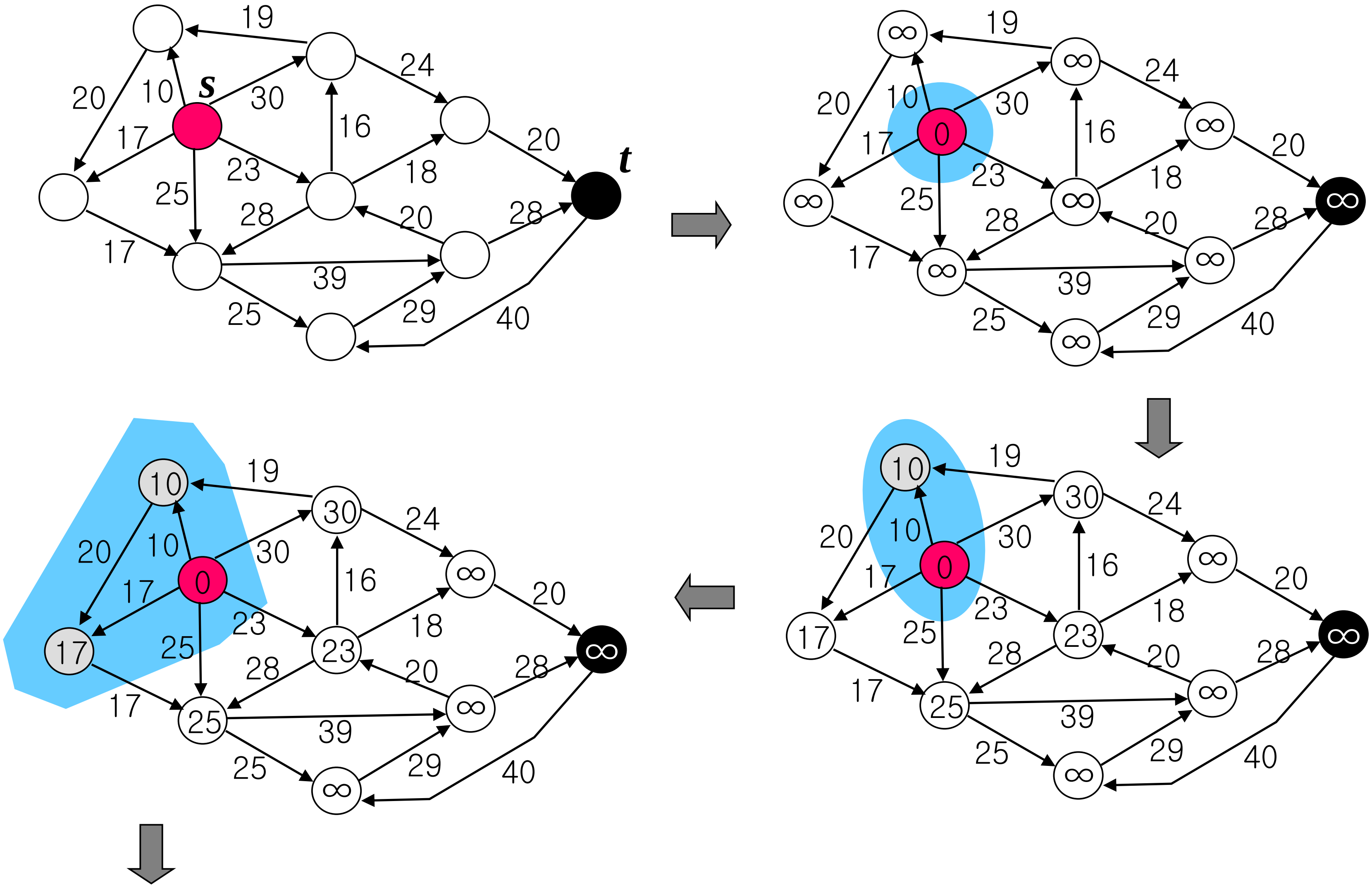
A^* Algorithm

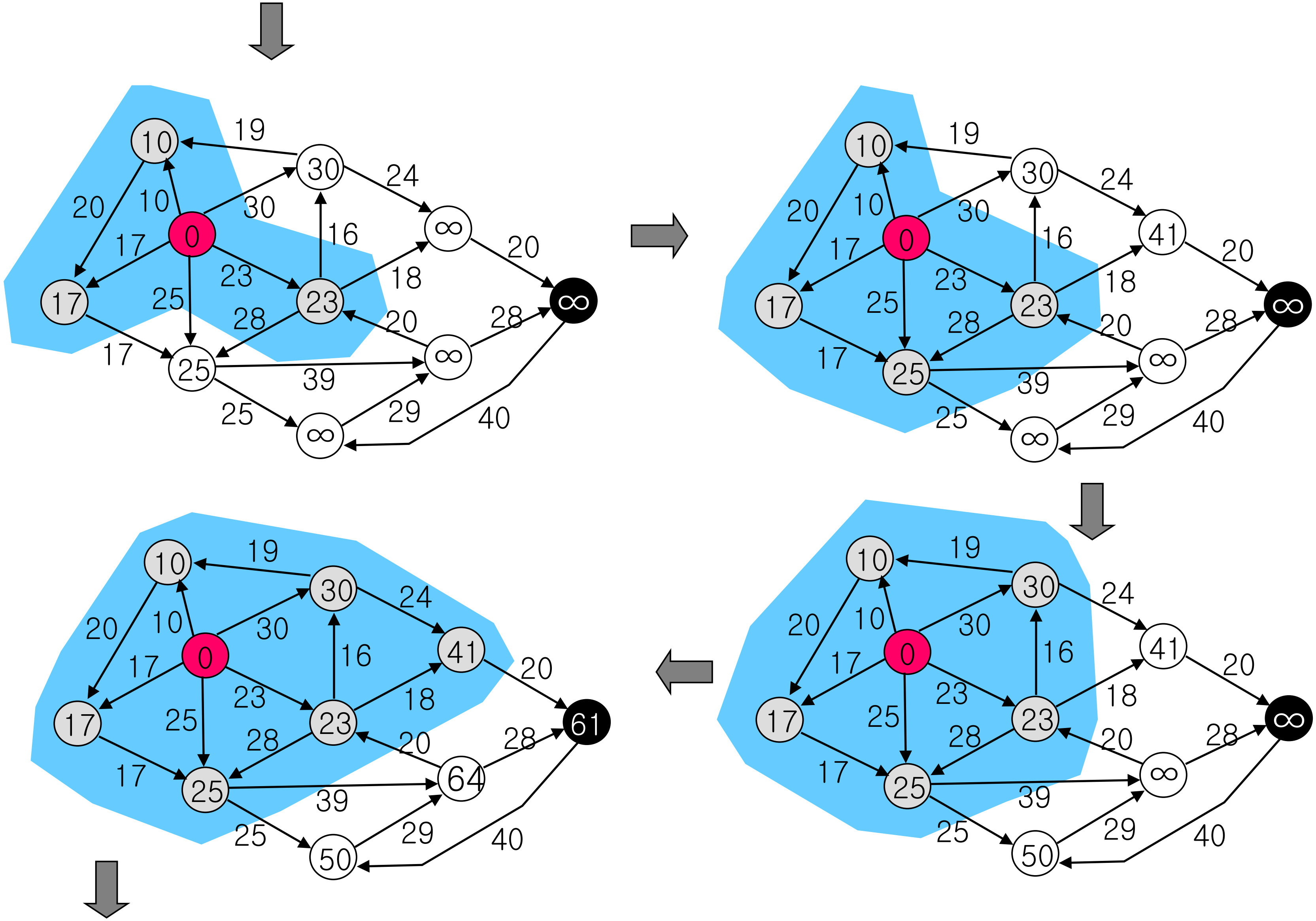
- Find the shortest path from a source to a destination
- Can be applied to NP-hard and P problems
- cf. Dijkstra algorithm
  - Single source
  - All destinations

A* Algorithm

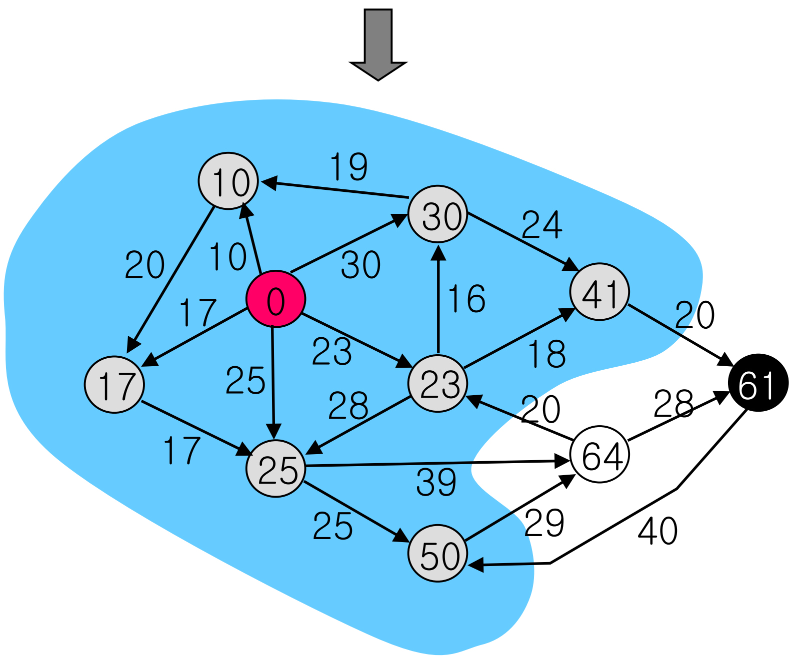
- Best-first search
  - Each vertex x has g(x): cost (shortest path weight) from source to x
  - Each vertex x has h(x): estimate of cost from x to destination
  - h(x) must be less than or equal to actual cost from x to destination
  - For all x, y, $h(x) \leq w(x,y) + h(y)$
  - A* always selects a vertex x that minimizes $g(x) + h(x)$

Dijkstra Algorithm

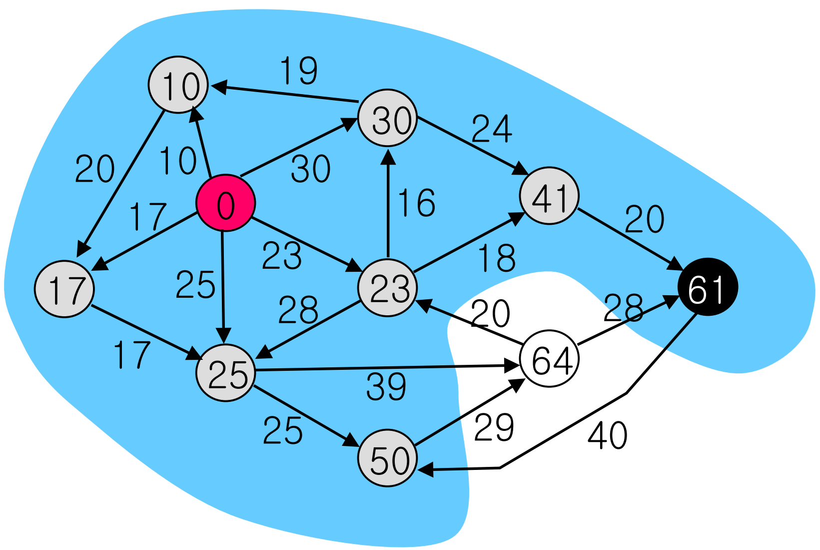




10 19 30 24 ∞ 20 ∞
20 10 0 30 16 17 23 18
17 25 23 28 20 28 ∞
17 25 39 25 ∞ 29 40
10 19 30 24 41 20 ∞
20 10 0 30 16 17 23 18
17 25 23 28 20 28 ∞
17 25 39 25 ∞ 29 40
10 19 30 24 41 20 ∞
20 10 0 30 16 17 23 18
17 25 23 28 20 28 ∞
17 25 39 25 50 29 40
10 19 30 24 41 20 61
20 10 0 30 16 17 23 18
17 25 23 28 20 28 64
17 25 39 25 50 29 40

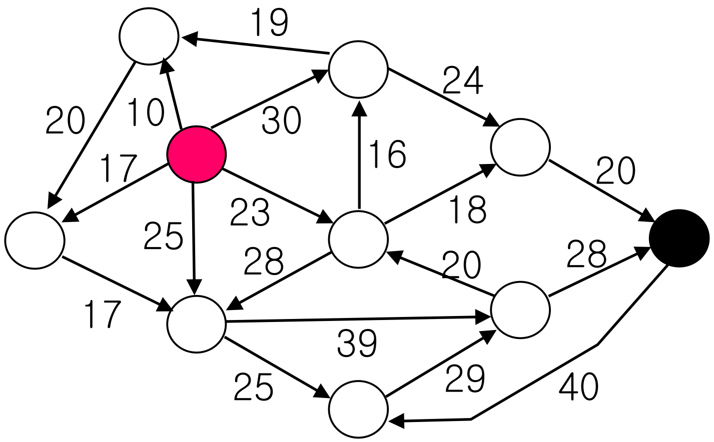


10
19
30
24
20
10
0
30
16
41
20
17
17
23
18
61
25
28
23
20
28
17
25
39
64
25
29
40
50

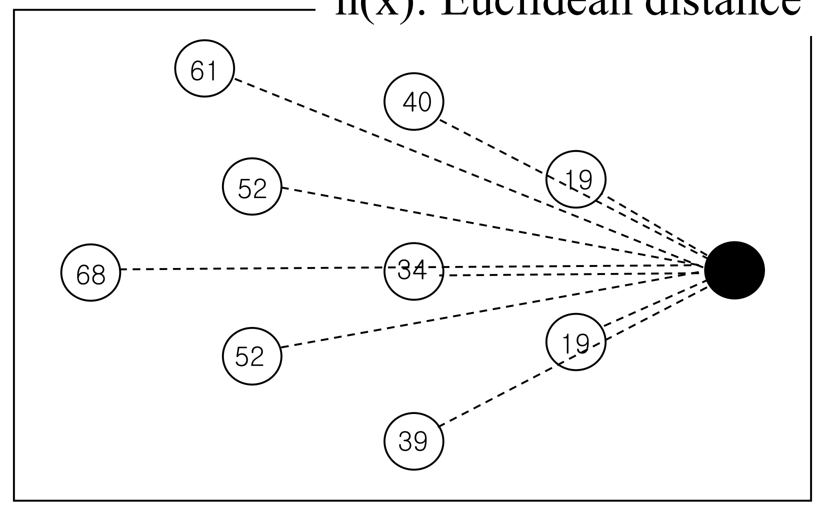


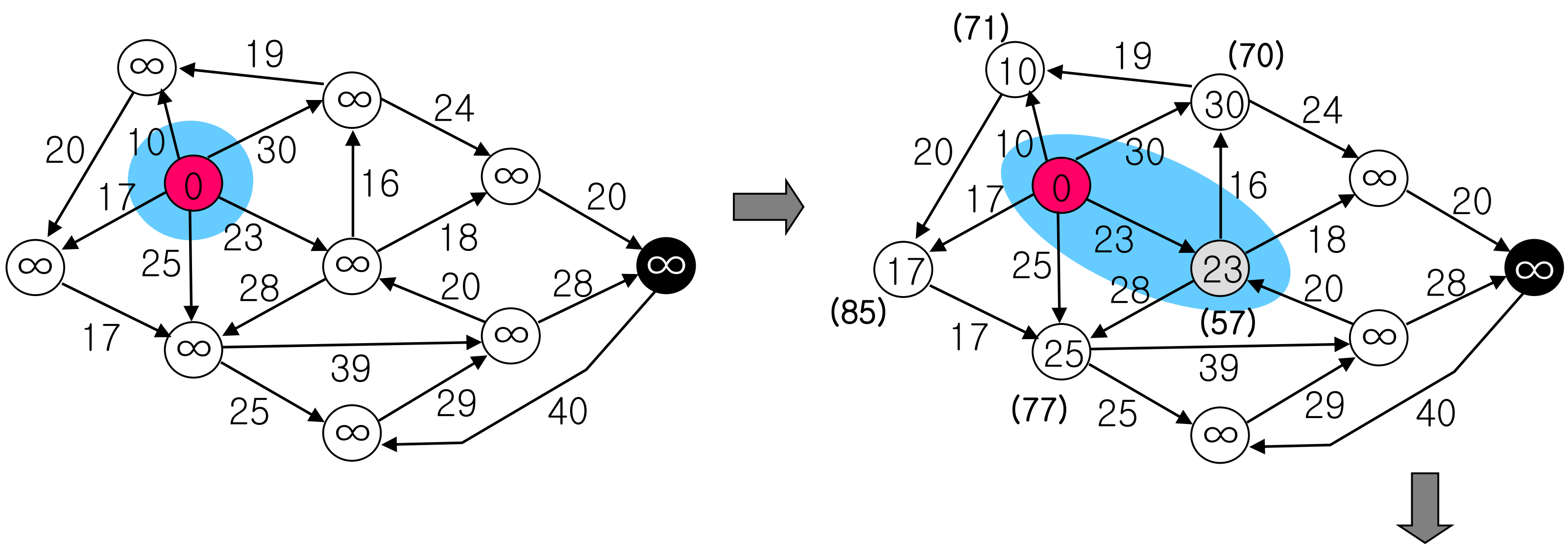
10
19
30
24
20
10
0
30
16
41
20
17
17
23
18
61
25
28
23
20
28
17
25
39
64
25
29
40
50

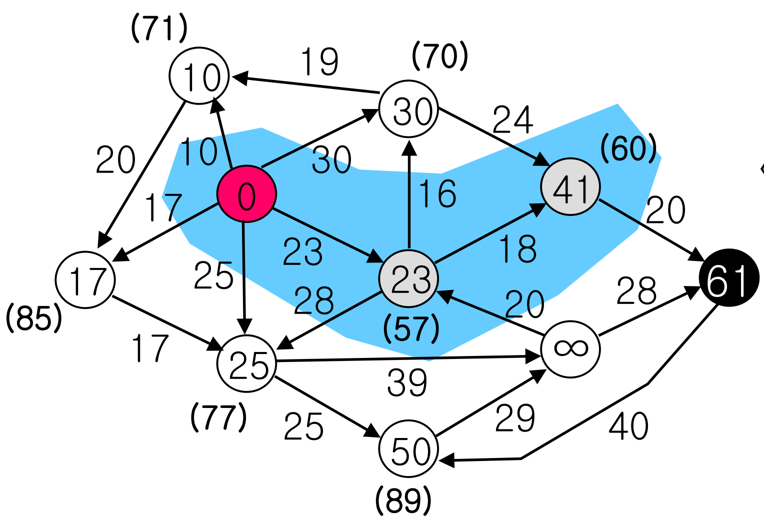
A* Algorithm

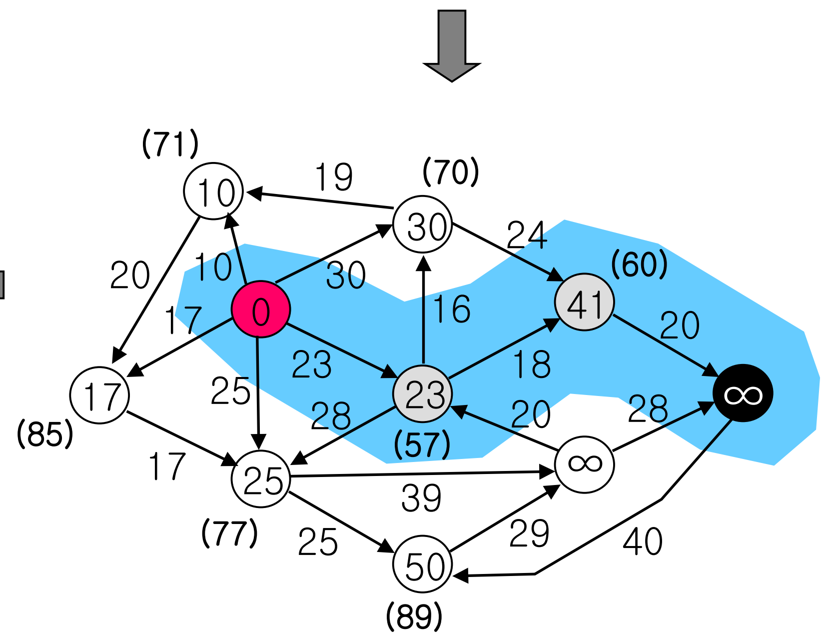


h(x): Euclidean distance





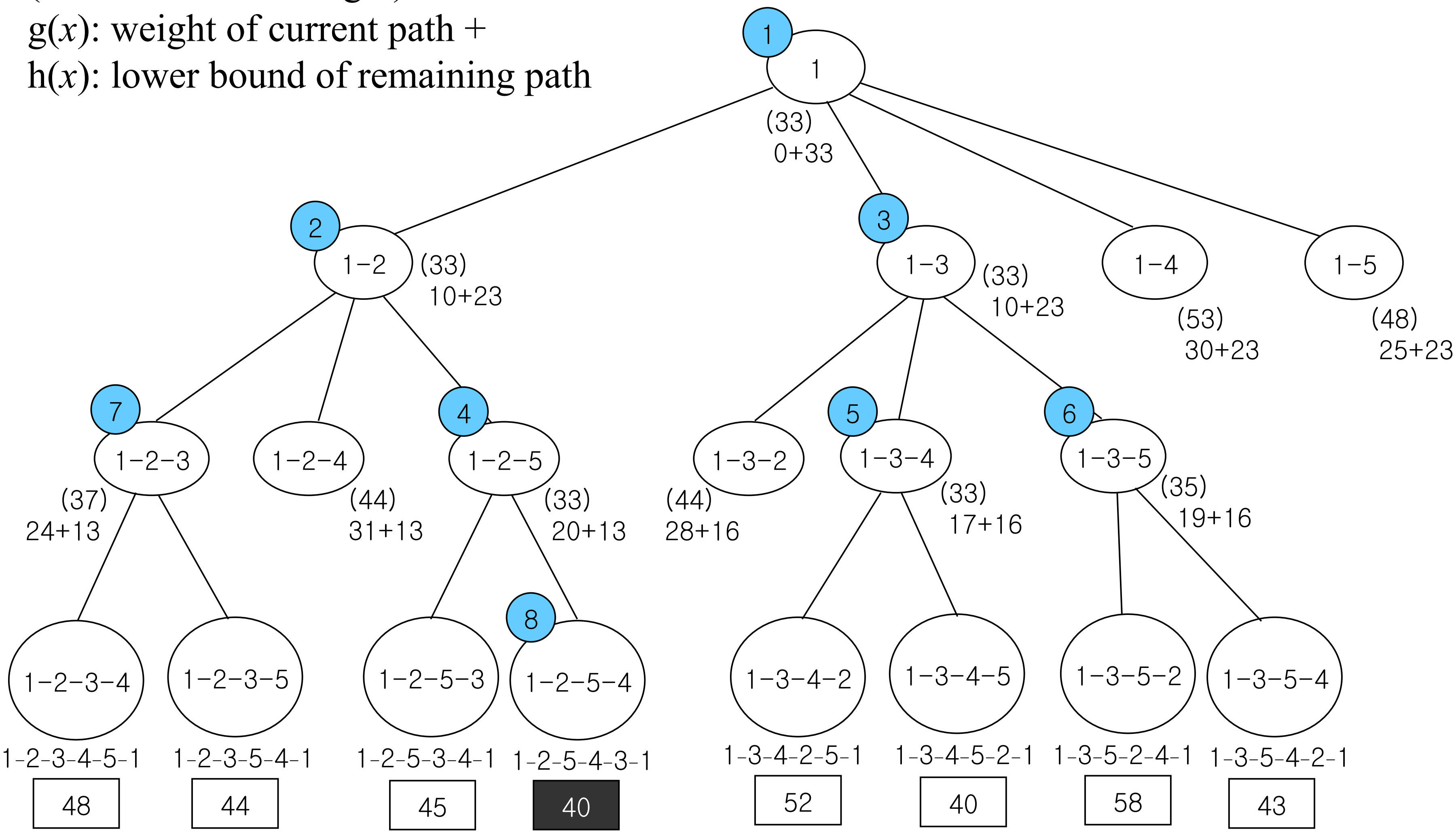




- ✓ A* is faster than Dijkstra by using h(x)

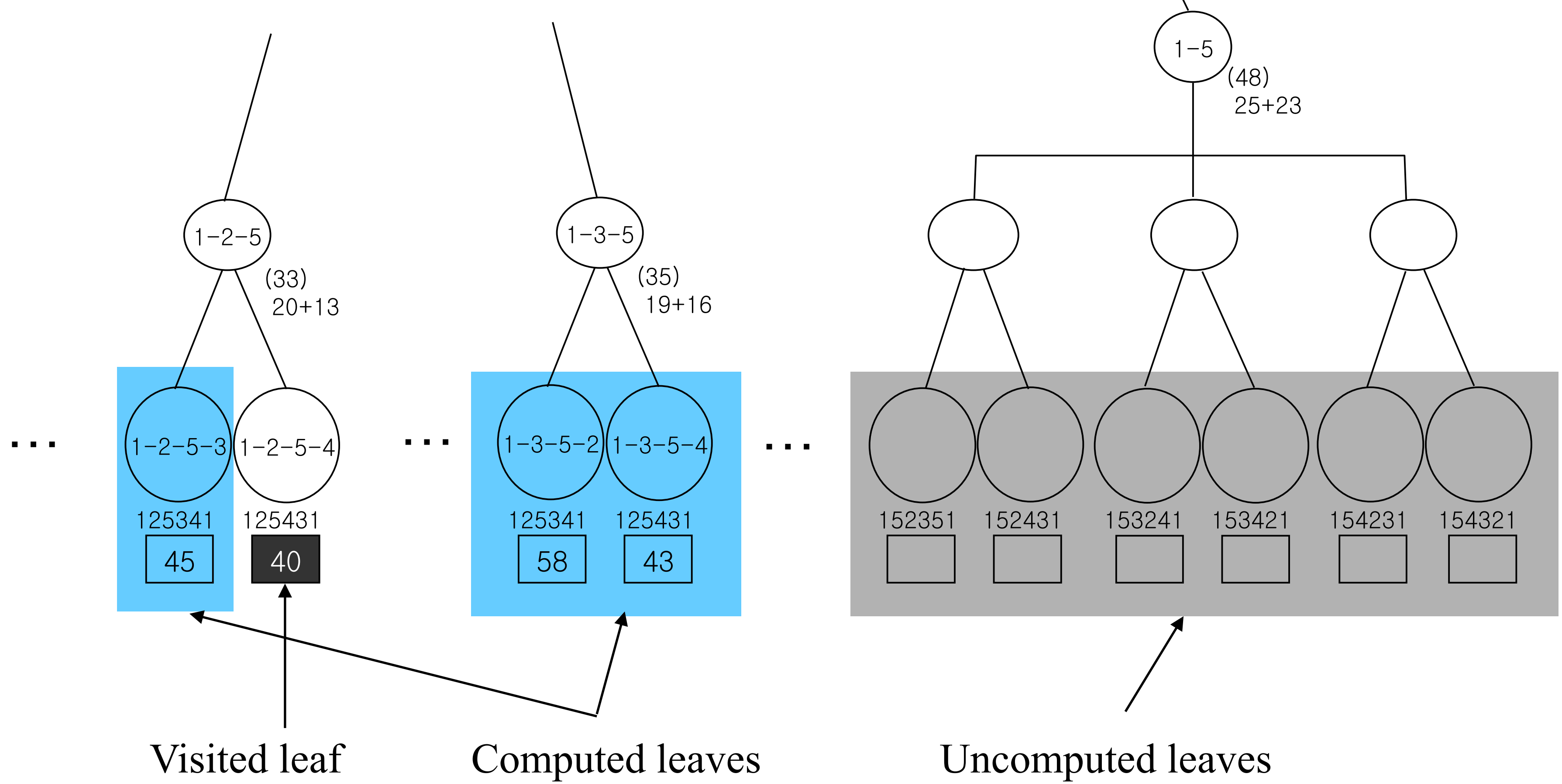
State-Space Tree of A* Algorithm for TSP

(lower bound of weight)
g(x): weight of current path +
h(x): lower bound of remaining path



A* Algorithm Terminates When It Visits First Leaf

leaves and leaves cannot be smaller than 40



Thank you