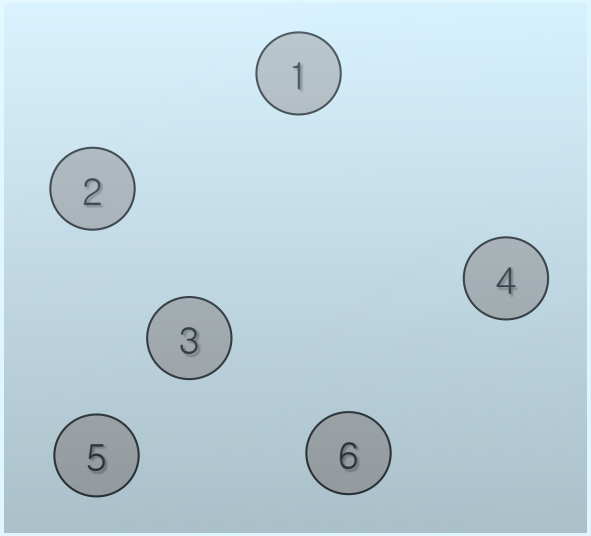
상태공간 트리의 탐색

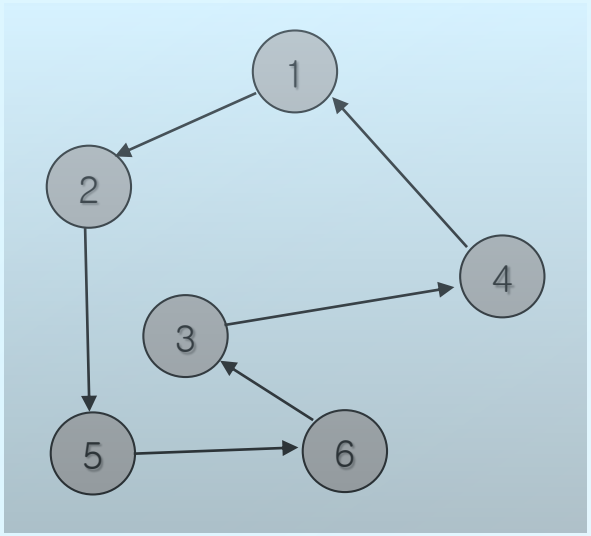
State-Space Tree

- 문제 해결 과정의 중간 상태를 각각 한 노드로 나타낸 트리
- 이 장에서 배우는 세가지 상태공간 탐색 기법
  - 백트래킹
  - 분기한정
  - A^* 알고리즘

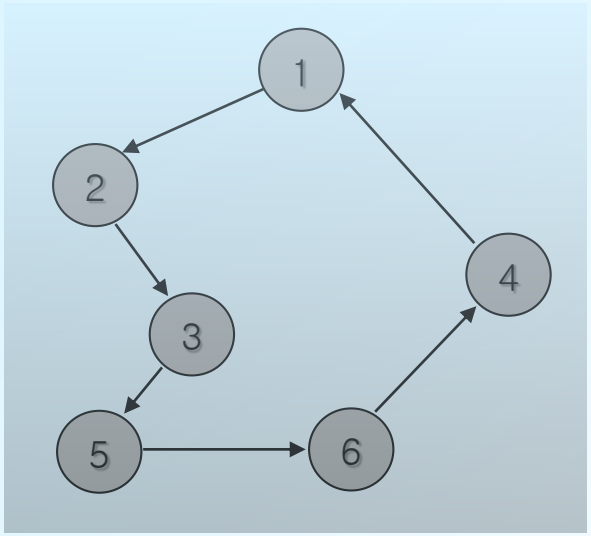
TSP의 예



TSP 예제

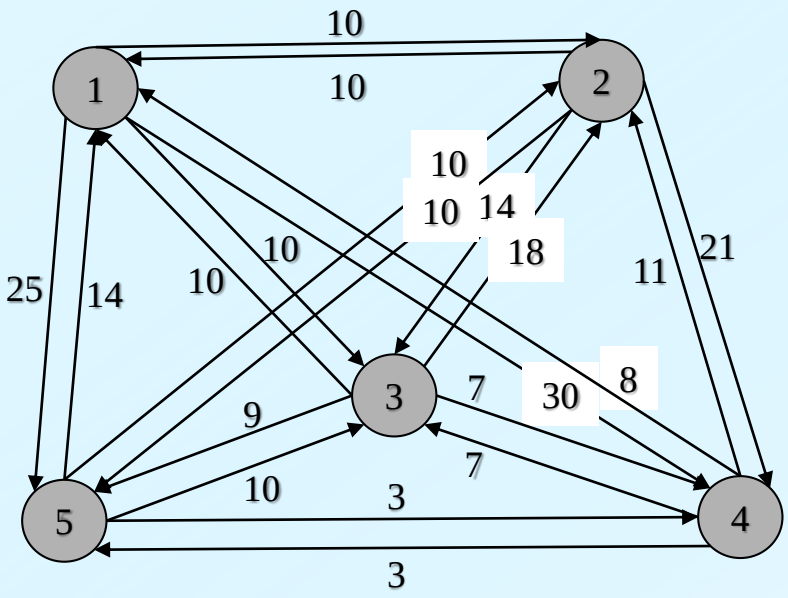


해의 예



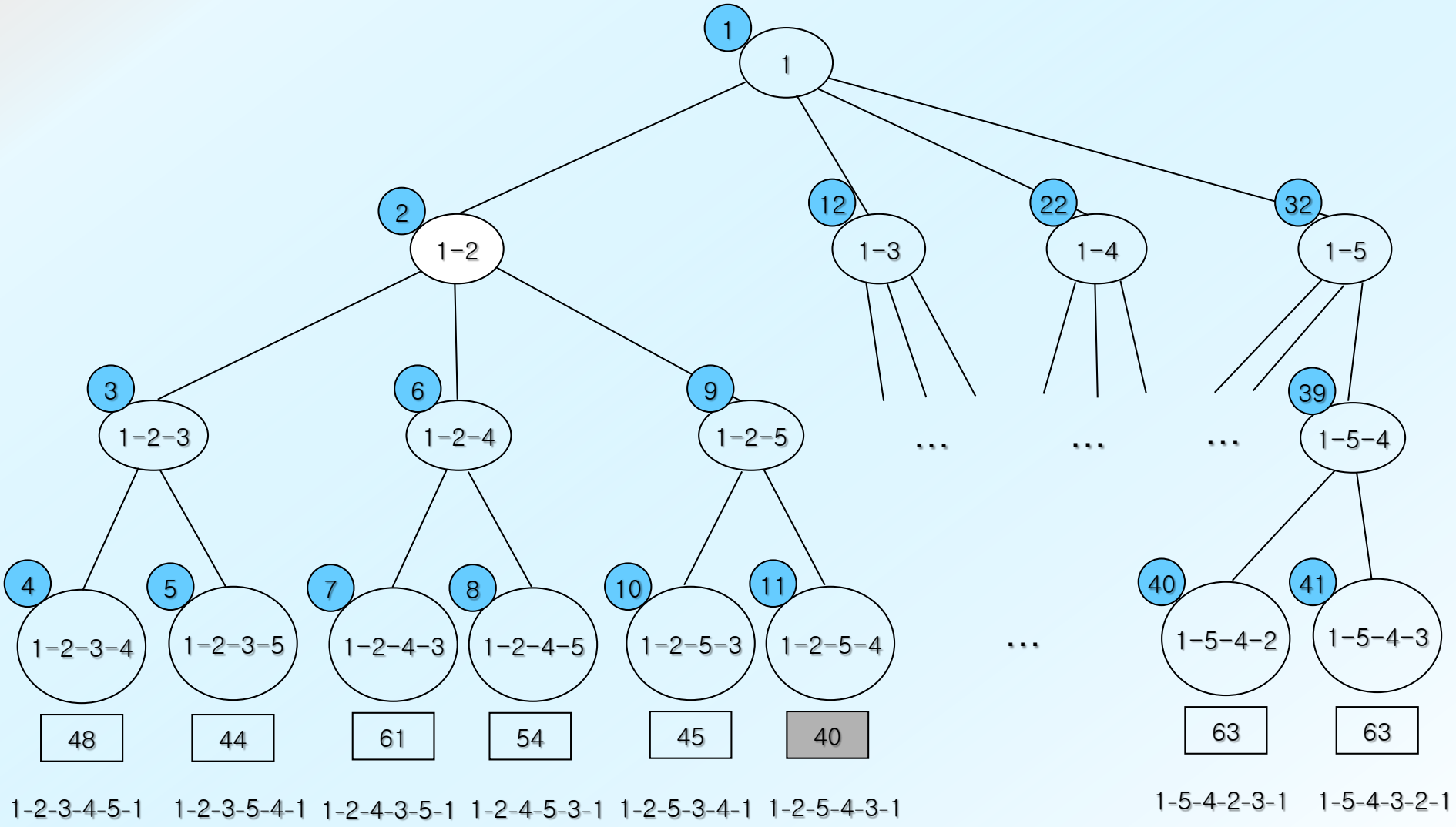
최적해

TSP와 Adjacency Matrix의 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

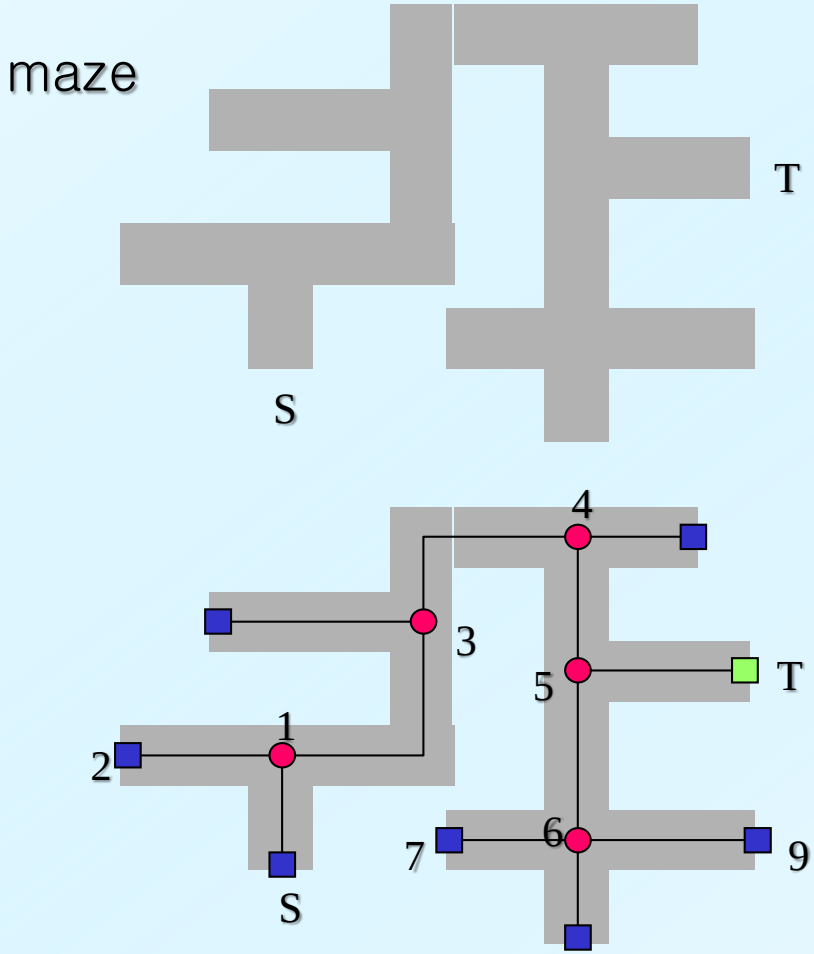
사전적 탐색의 State-Space Tree



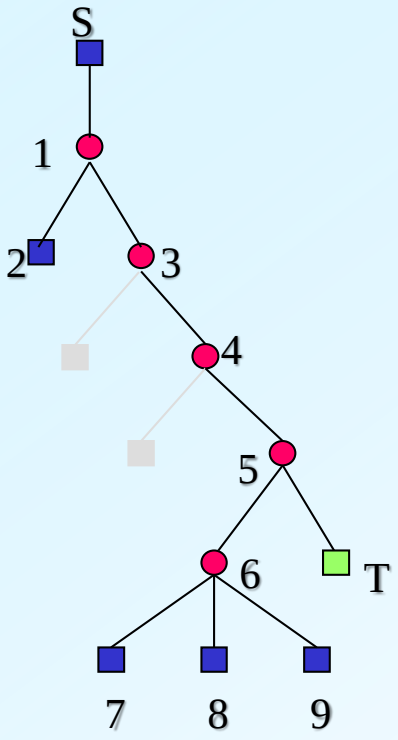
Backtracking

- DFS 또는 그와 같은 스타일의 탐색을 총칭한다
- Go as deeply as possible, backtrack if impossible
  - 가능한 지점까지 탐색하다가 막히면 되돌아간다
- 예
  - Maze, 8-Queens problem, Map coloring, …

Maze



그래프로 모델링한 미로



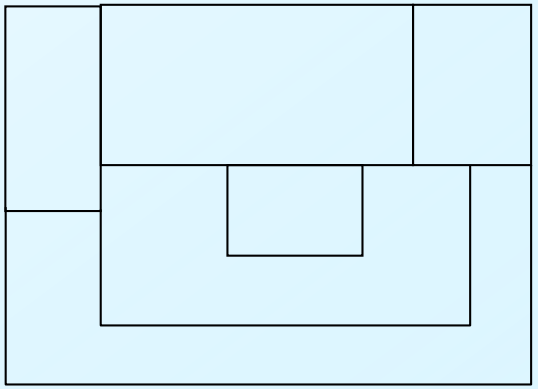
Branching tree

```
maze(v):
        visited[v] ← YES
        if (v = T)
                print "성공!"
                exit( )
        for each x ∈ L(v)  ▷ L(v) : vertex v의 adjacent vertex 집합
                if (visited[x] = NO)
                        prev[x] ← v
                        maze(x)
```

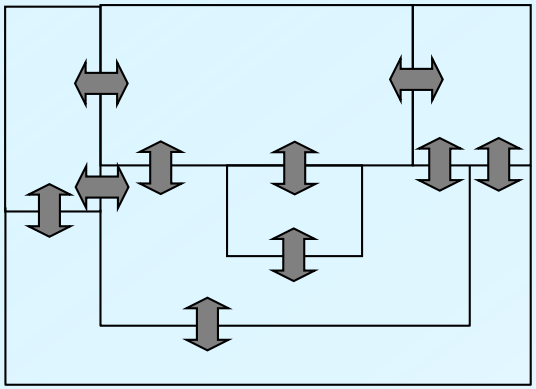
Coloring Problem

- Graph에서
  - 인접한 vertex는 같은 색을 칠할 수 없다
  - k 개의 색상을 사용해서 전체 graph를 칠할 수 있는가?
  - k가 2 를 넘으면 NP-Hard
  - 3-colorable 문제를 4 color로 칠하는 것도 NP-Hard
  - Planar graph는 4-colorable

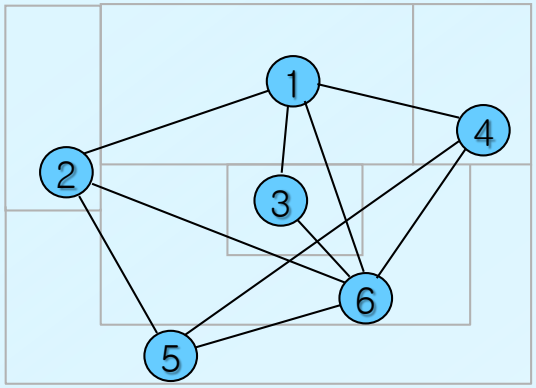
Coloring Problem의 예: Map Coloring



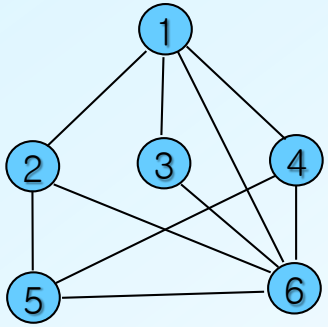
(a) 지도



(b) 구역간의 인접관계



(c) 연결관계를 정점과 간선으로 나타낸 것



(d) (c)와 동일한 그래프

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

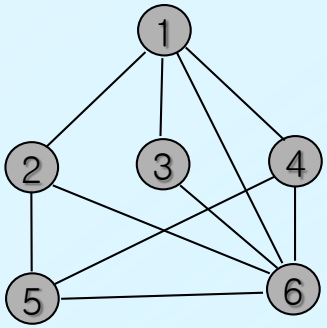
valid(i, c):

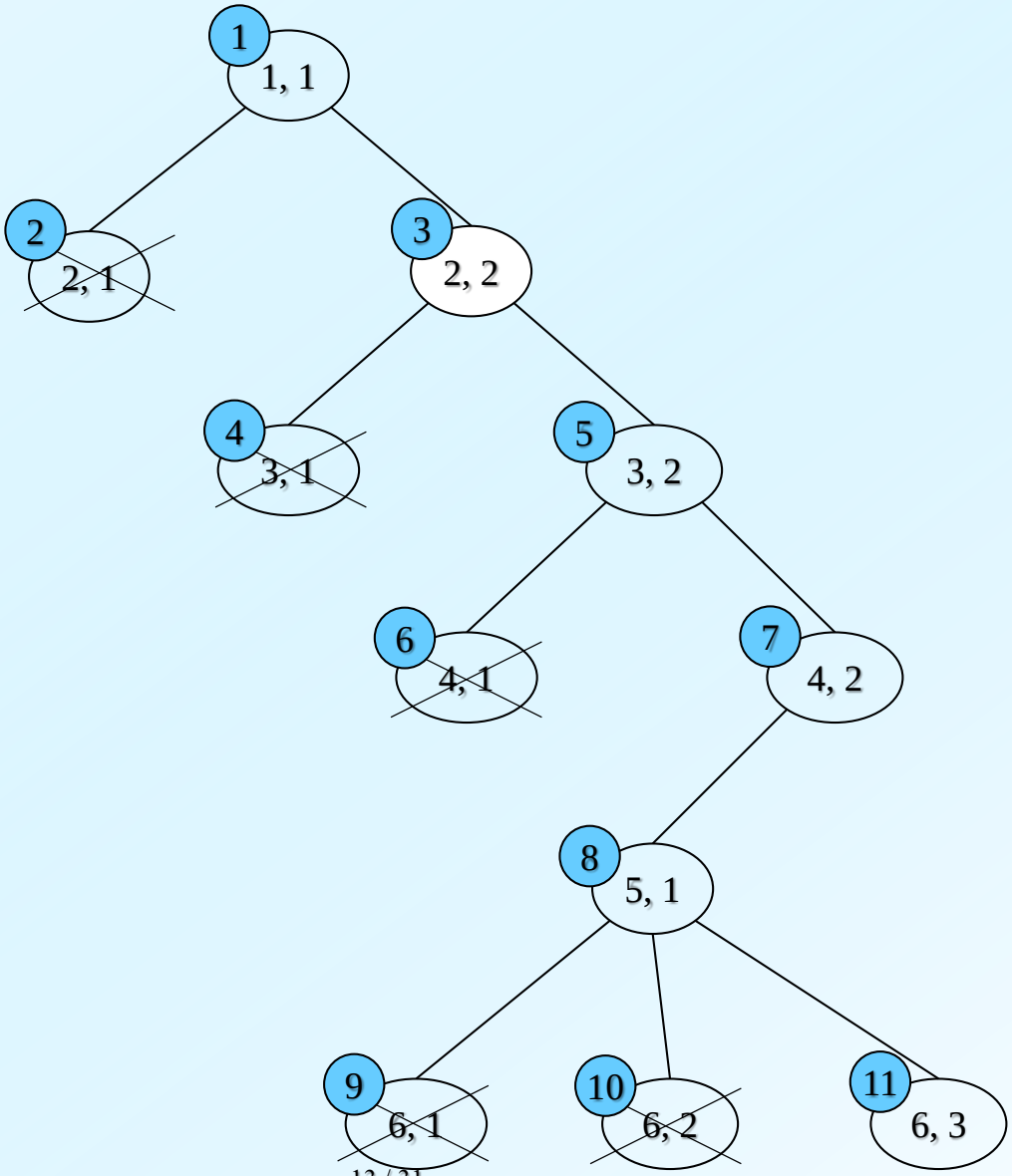
▷ i: vertex, c: color

▷ 질문: vertex i-1까지는 제대로 칠이 된 상태에서 vertex i를 색 c로 칠하려면 이들과 색이 겹치지 않는가?

```
for j ← 1 to i-1
        ▷ vertex i와 j 사이에 edge가 있고, 두 vertex가 같은 색이면 안된다
    if ((i, j) ∈ E and color[j] = c) return FALSE;
return TRUE;
```

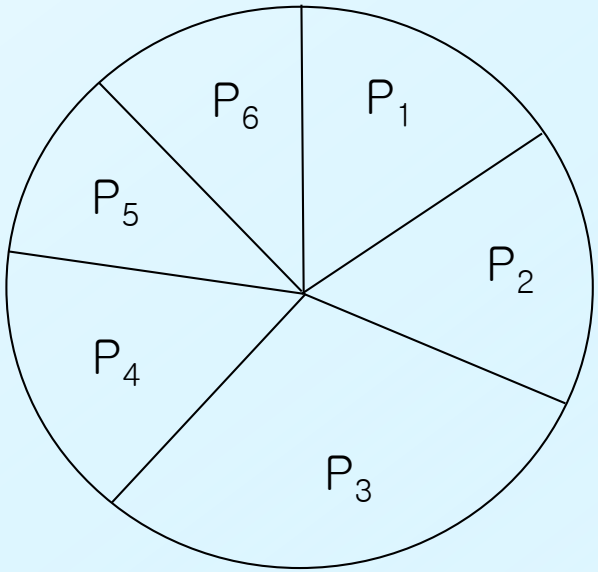
State-Space Tree



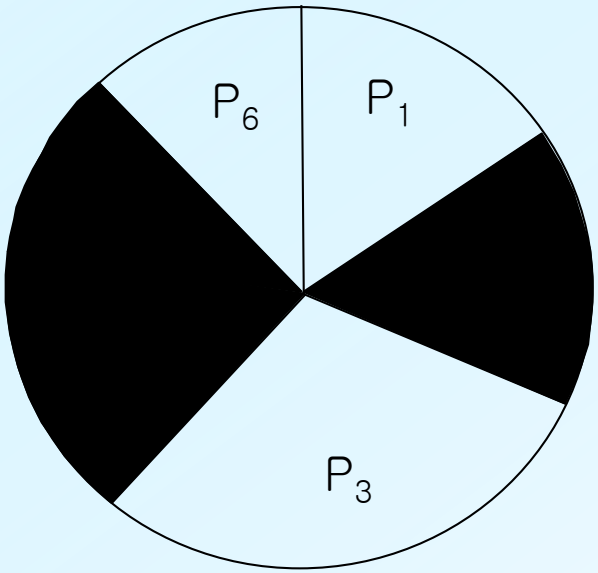


Branch-and-Bound

- 분기branch와 한정bound의 결합
  - 분기를 한정시켜 쓸데없는 시간 낭비를 줄이는 방법
- Backtracking과 공통점, 차이점
  - 공통점
    - 경우들을 차례로 나열하는 방법 필요
  - 차이점
    - Backtracking – 가보고 더 이상 진행이 되지 않으면 돌아온다
    - Branch-&-Bound – 최적해를 찾을 가능성이 없으면 분기는 하지 않는다



어느 시점에 가능한 선택들

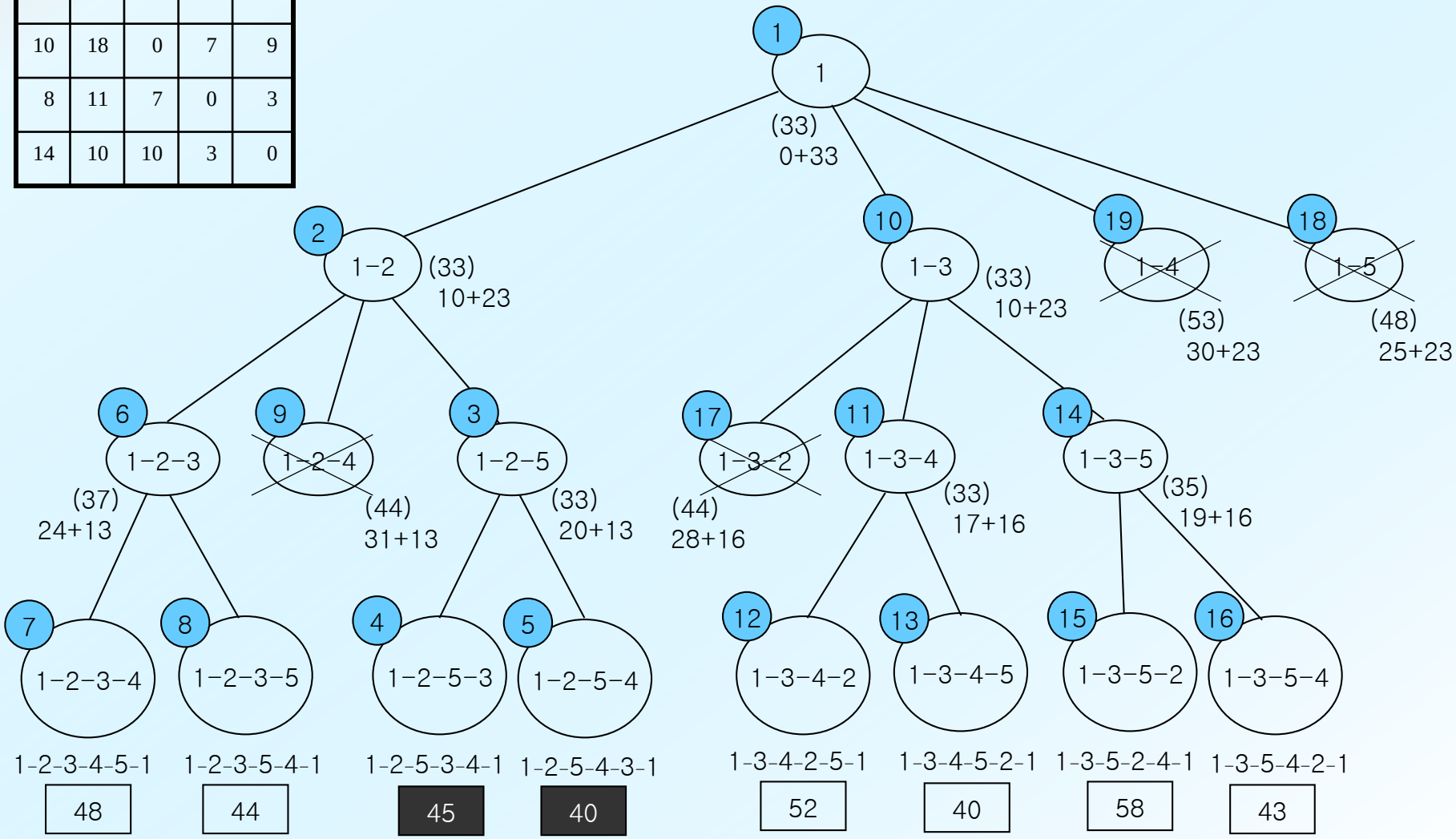


최적해를 포함하지 않아 제외하는 선택들

TSP 예제를 대상으로 한 Branch-and-Bound 탐색의 예

(State-Space Tree)

0	10	10	30	25
10	0	14	21	10
10	18	0	7	9
8	11	7	0	3
14	10	10	3	0



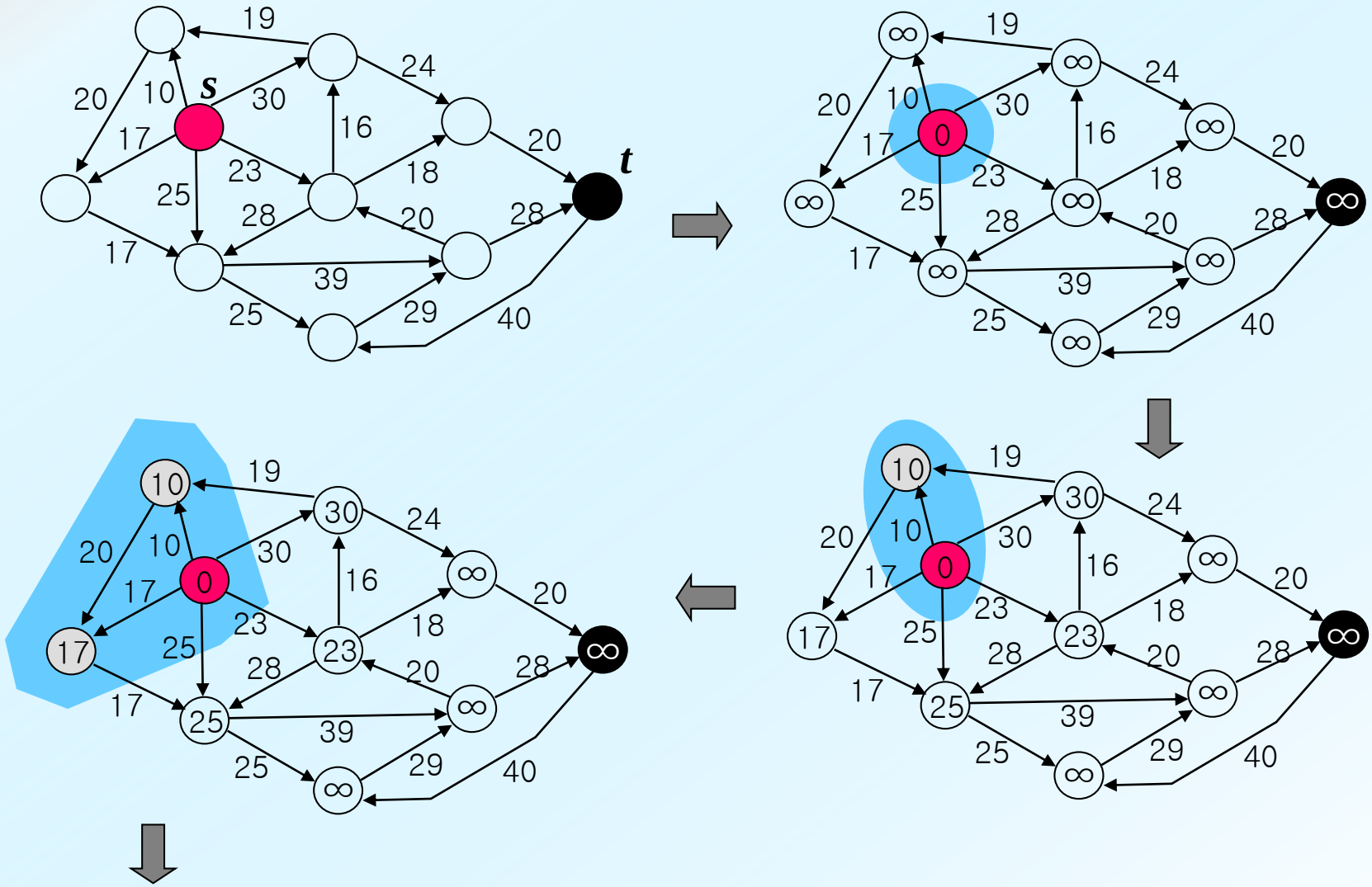
A* Algorithm

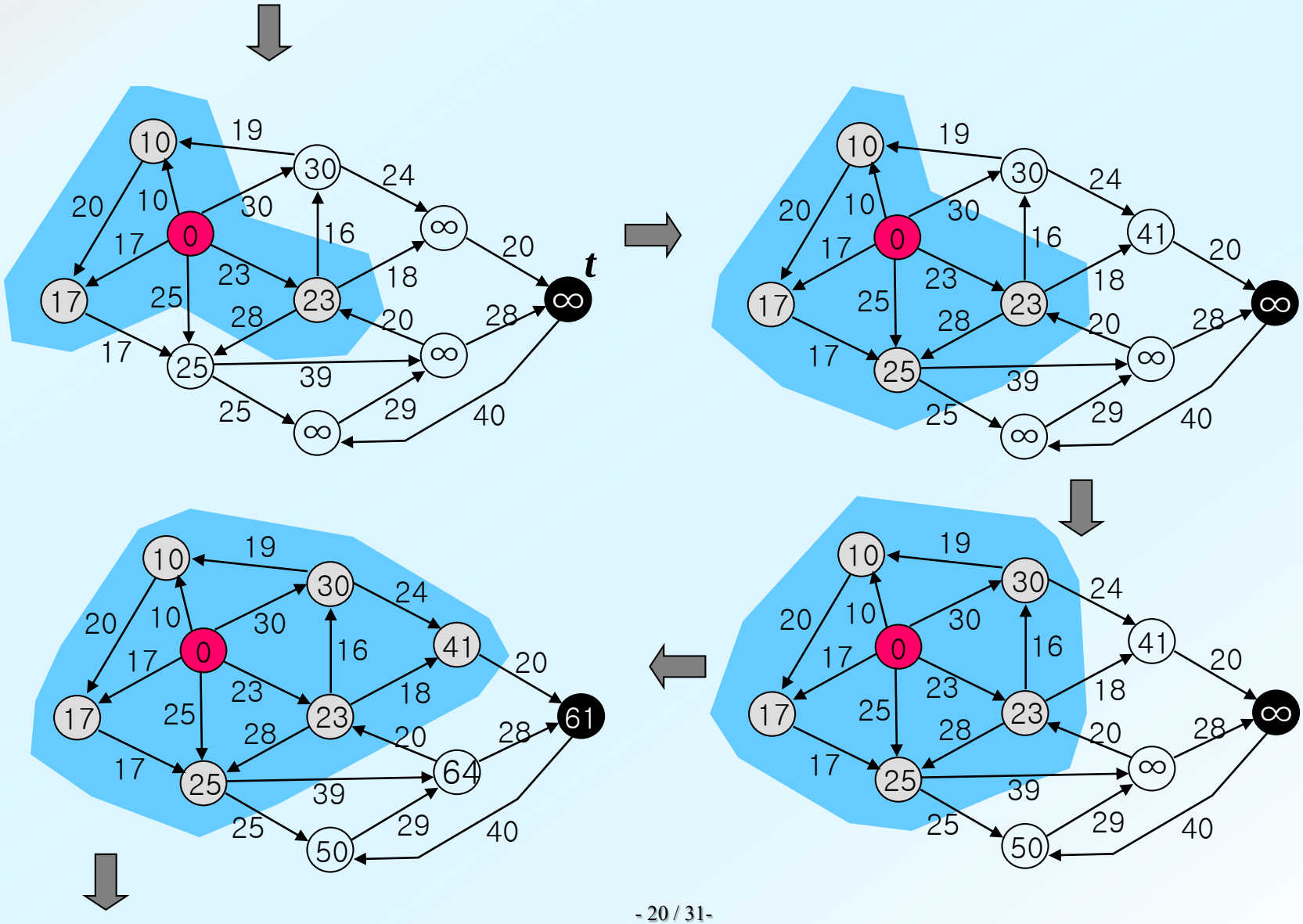
- Best-First-Search
  - 각 정점이 매력함수값 $g(x)$를 갖고 있다
  - 방문하지 않은 정점들 중 $g(x)$ 값이 가장 매력적인 것부터 방문한다
- A* algorithm은 best-first search에 목적점에 이르는 잔여추정거리 $h(x)$를 고려하는 알고리즘이다
  - "$g(x) + h(x)$"가 가장 매력적인 것부터
  - (Vertex x로부터 목적점에 이르는) 잔여거리의 추정치 $h(x)$는 실제치보다 크면 안된다
  - 추정치 전제 조건(단조성monotonicity): $h(x) \leq w_{xy} + h(y)$

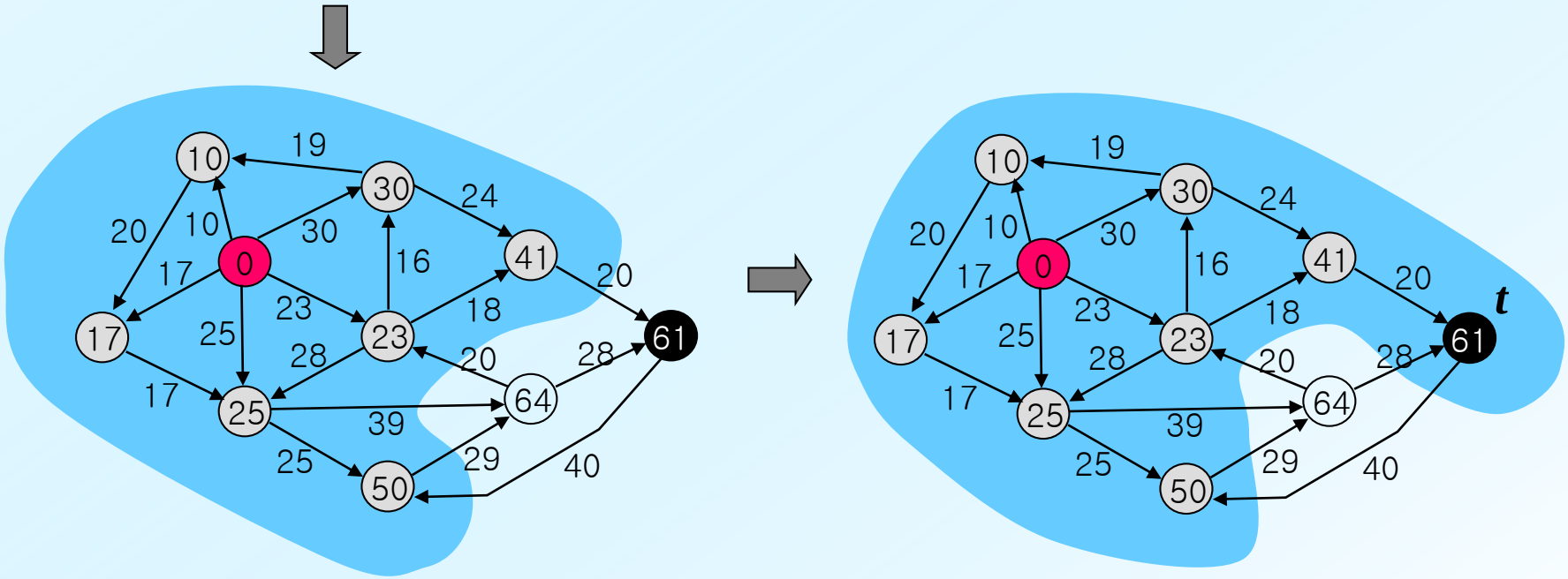
Shortest Path 문제

- Remind: Dijkstra algorithm
  - 시작점은 하나
  - 시작점으로부터 다른 모든 vertex에 이르는 최단경로를 구한다 (목적점이 하나가 아니다)
- A* algorithm에서는 목적점이 하나다

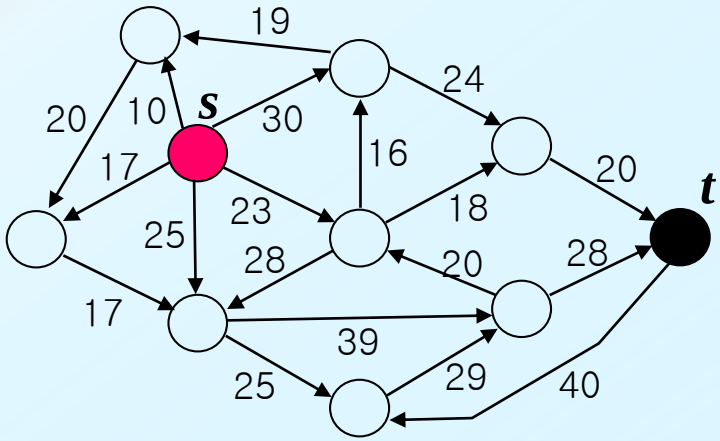
Dijkstra Algorithm의 작동 예

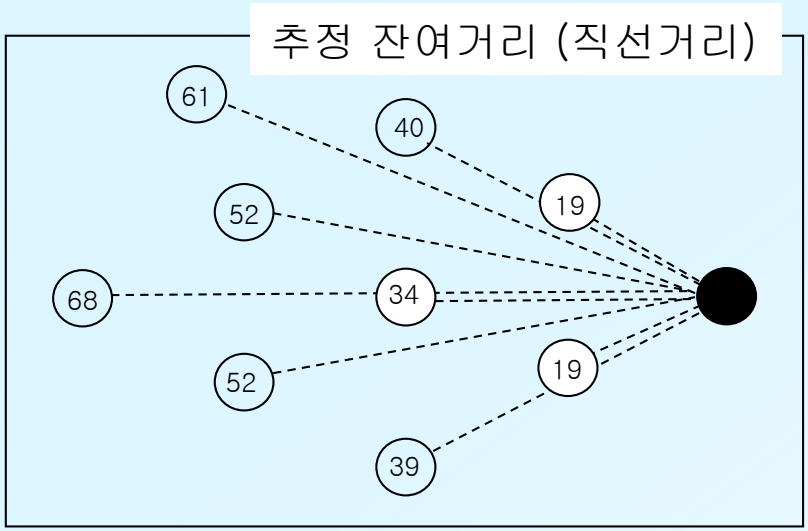


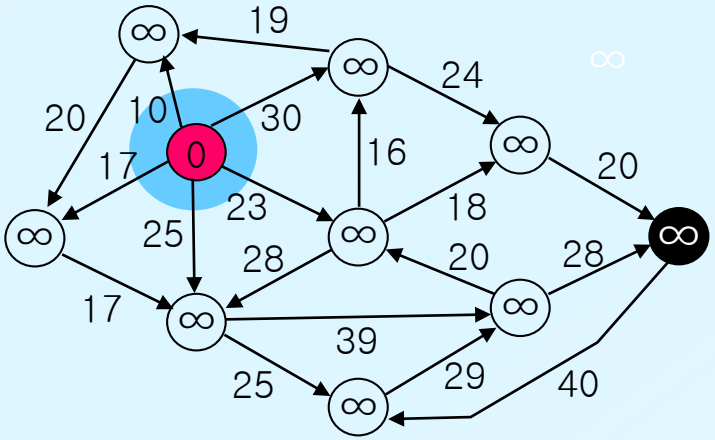

10
19
30
24
20
10
30
∞
16
20
17
t
23
18
17
25
28
23
20
28
17
∞
25
39
25
29
∞
40
61
64
50

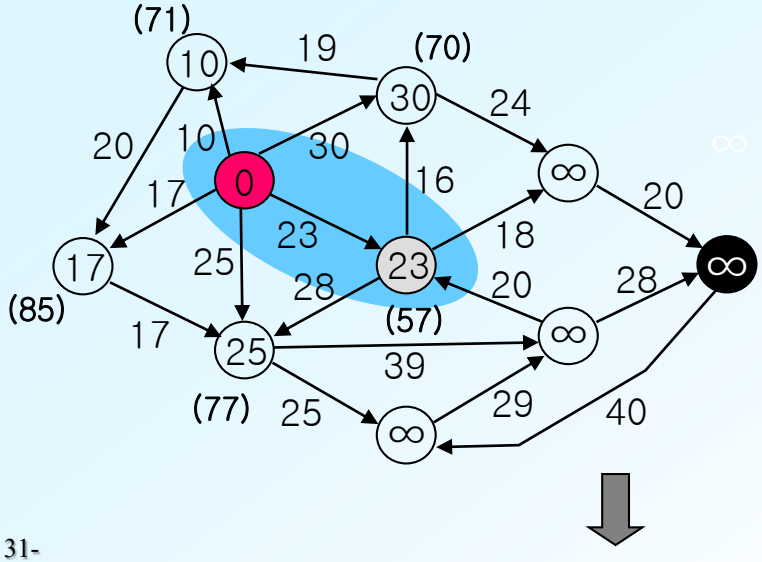

10
19
30
24
20
10
30
0
41
20
17
16
23
18
17
25
23
61
28
20
28
17
64
25
39
25
29
40
50
t

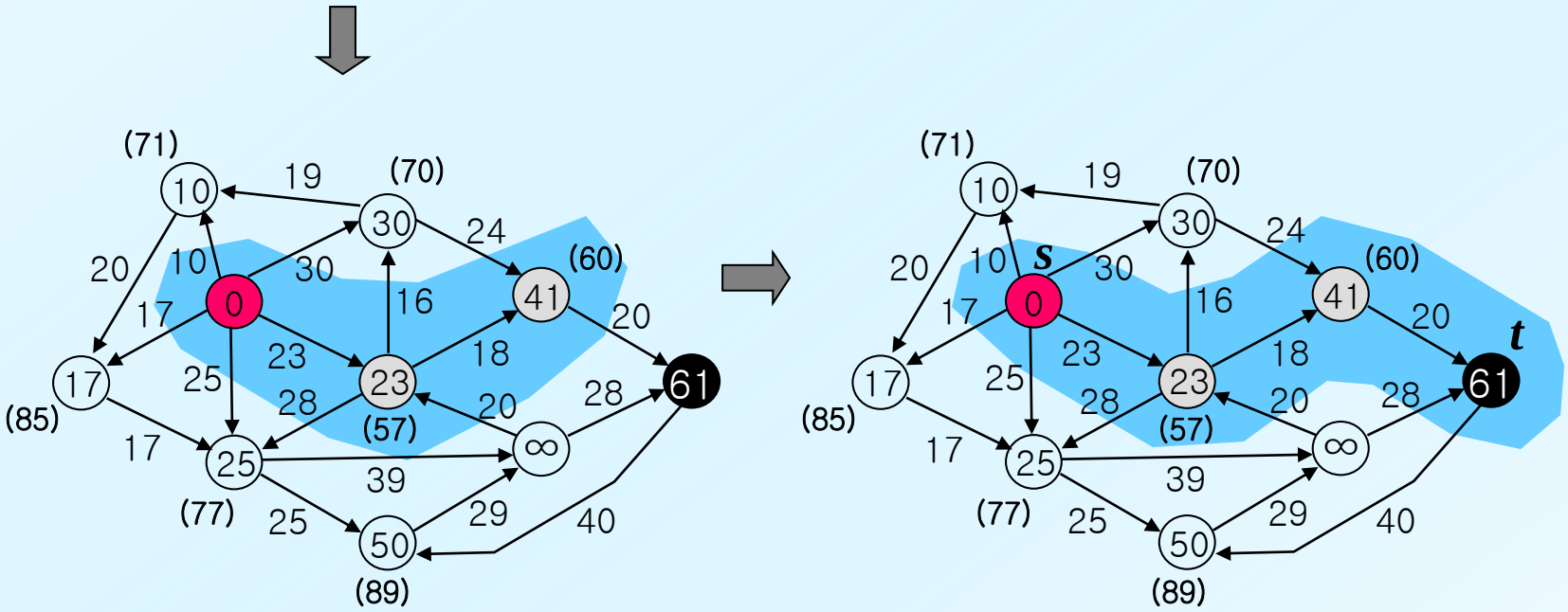
A* Algorithm의 작동 예









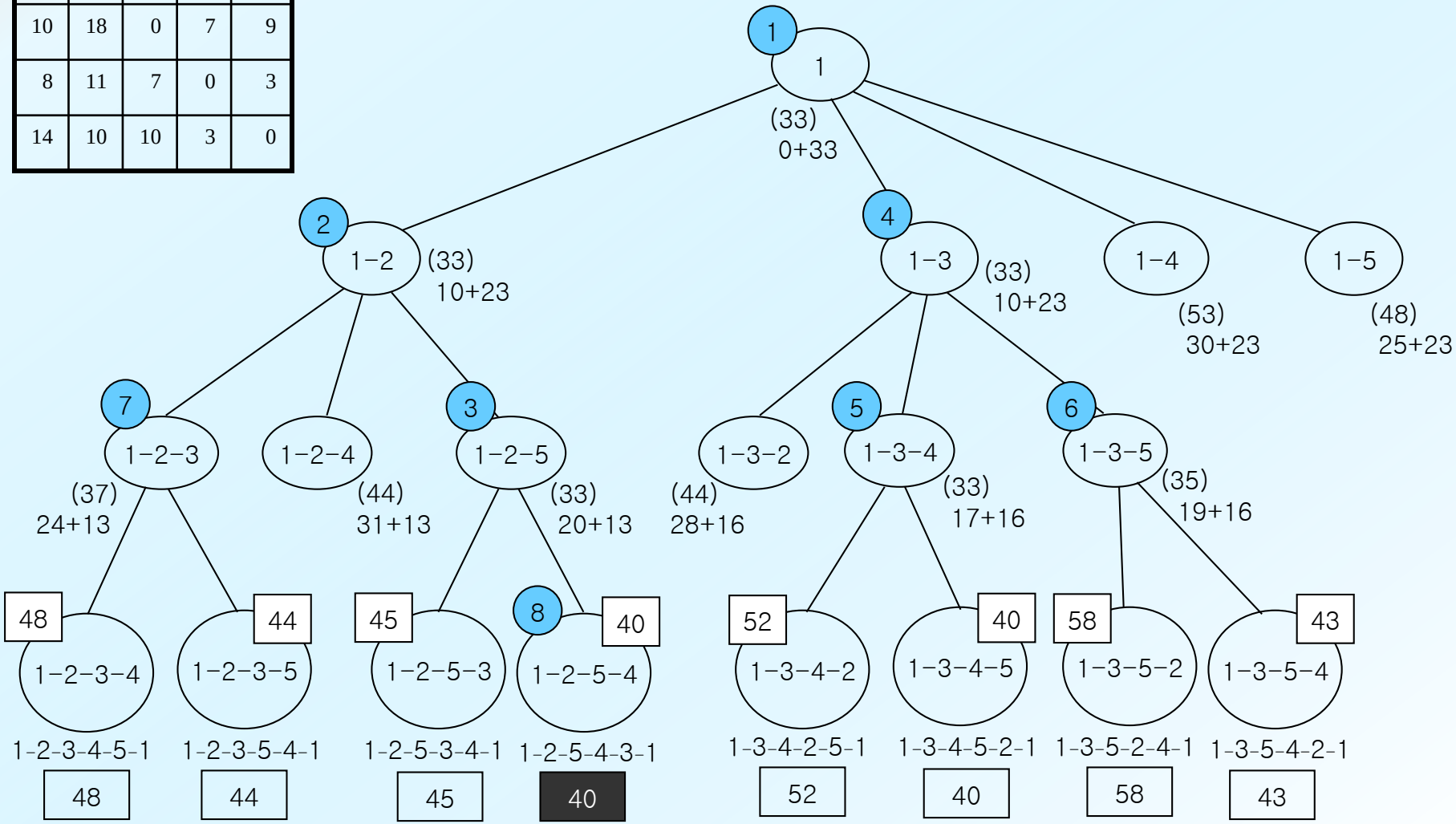


✓ 추정잔여거리를 사용함으로써 탐색의 단계가 현저히 줄었다

✓ 전제 조건: $\forall x, y \in V, \mathrm{h}(x) \leq \mathrm{w}(x, y) + \mathrm{h}(y)$

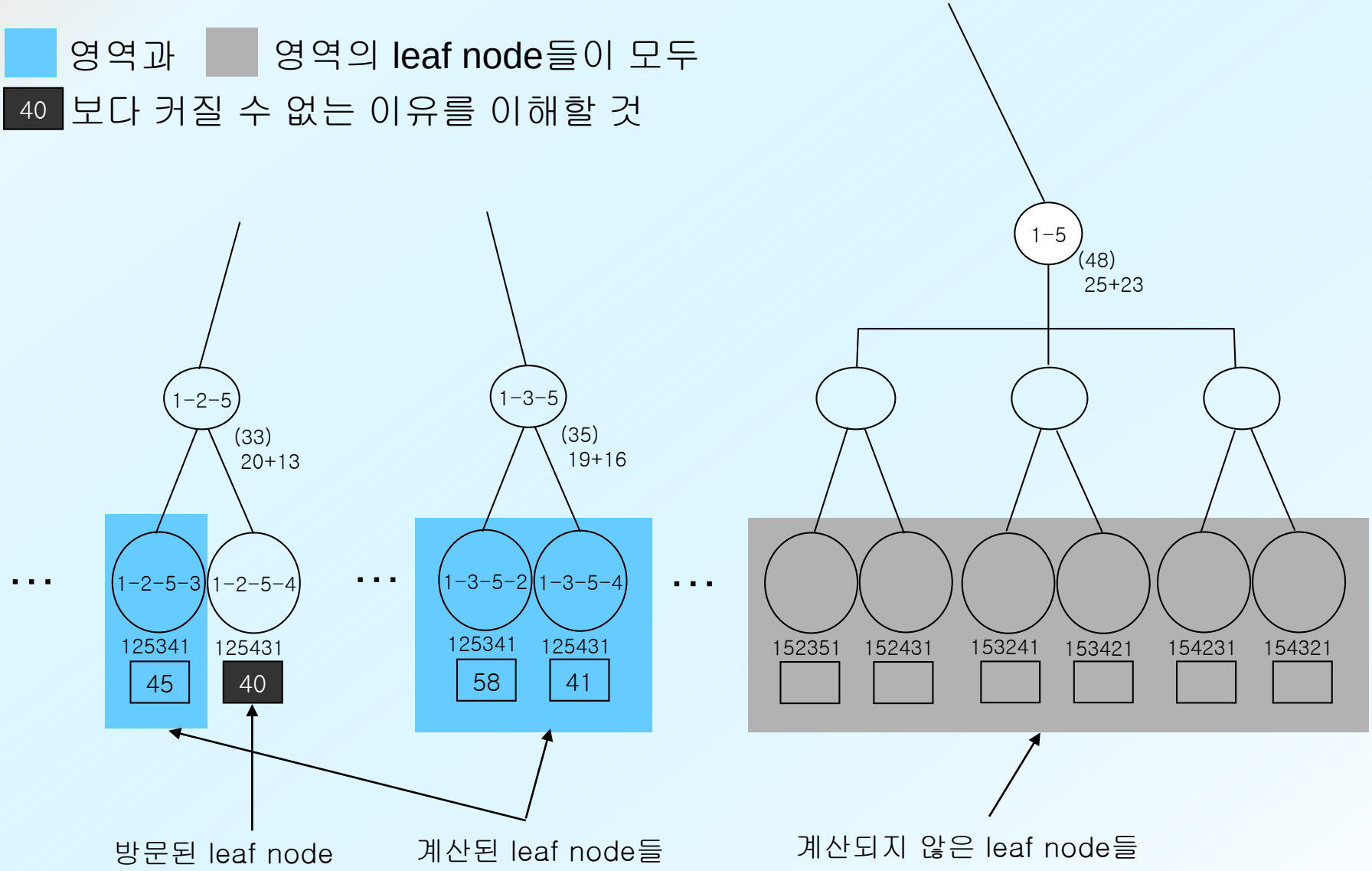
TSP 예제를 대상으로 한 A* Algorithm 탐색의 예

0	10	10	30	25
10	0	14	21	10
10	18	0	7	9
8	11	7	0	3
14	10	10	3	0



A* Algorithm이 첫 Leaf Node를 방문하는 순간 종료되는 이유

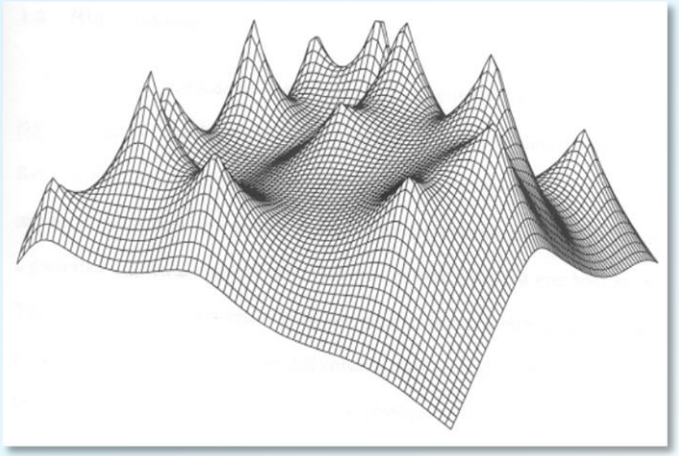
영역과 영역의 leaf node들이 모두 40 보다 커질 수 없는 이유를 이해할 것

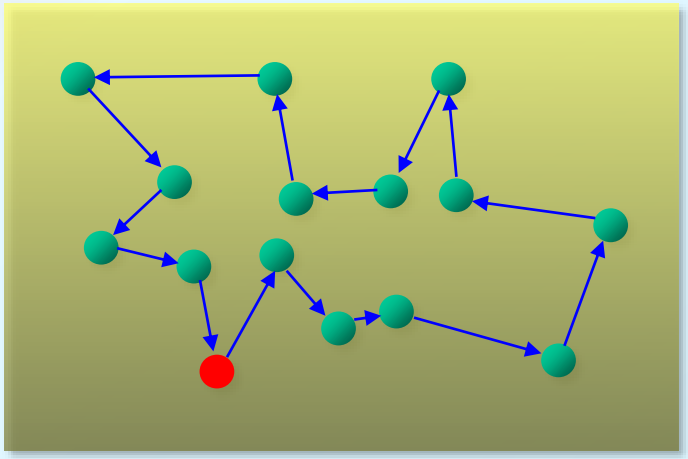


문제 공간의 방대함

문제는 공간을 이루고,
알고리즘은 공간을 여행한다

각각의 해는 자신의 품질을 갖고,
이들은 지형landscape을 만든다.

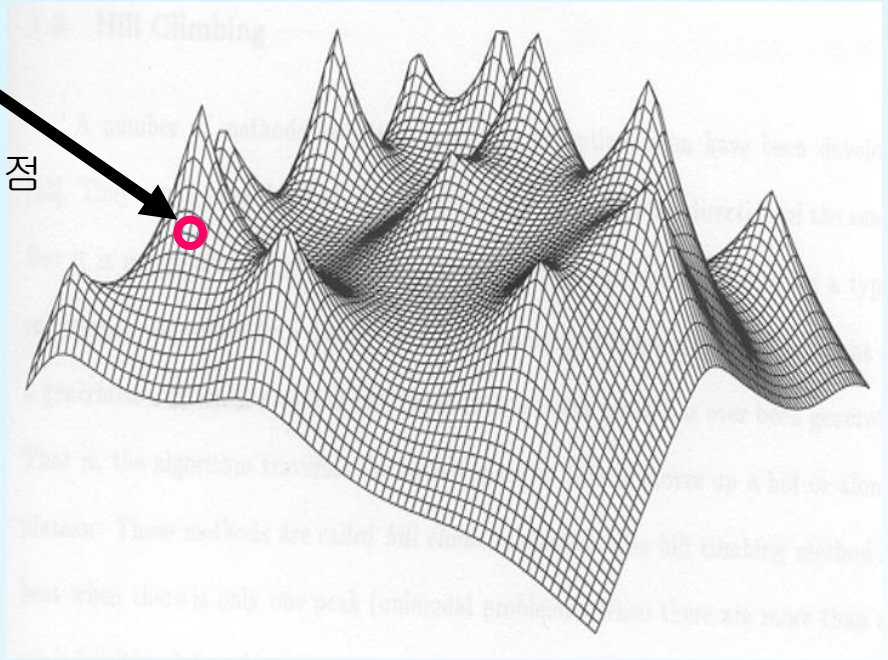




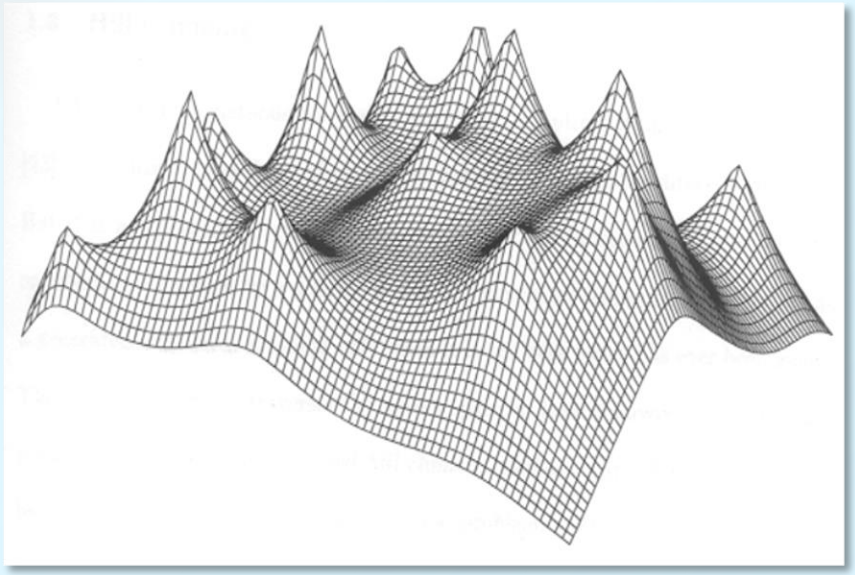
하나의 해

비유를 위한 그림
실제로는 3차원보다 훨씬 차원이 높다

지형도에서 한 점



공간의 여행

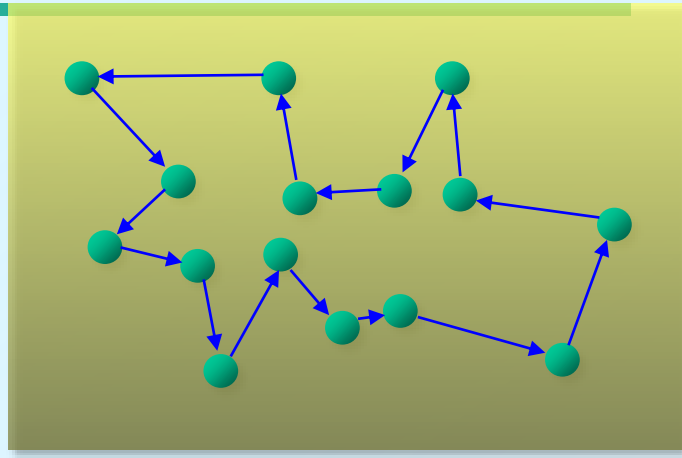


문제 = 공간

알고리즘 = 여행수단




도대체 얼마나 복잡하길래: TSP



질문:

(컴퓨터가 1초에 150만 경우의 수를 평가한다면)
25개 지점을 다 방문하고 돌아오는
모든 경우($24!=6.2*10^{23}$)를 보는데 드는 시간?

1분 1시간 1일 1달 1년 1000년 10억년 1조년

300억년

- ✓ 27명이면 10조년
- ✓ 요즘은 십만개 짜리 문제도 다룬다!!

끌개 (Attractor)

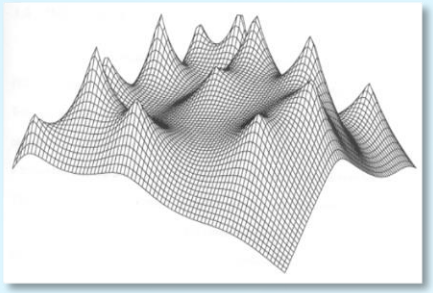
Attractor = 끌개
- 문제공간 상에서의 국소적 최적점
- 공간탐색의 목표이자 장애물

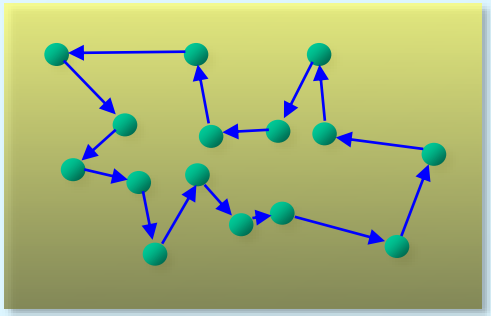
끌개의 예
- 생태계의 종: 개나리, 질경이, 치타, 가젤, …
- 인류사의 조직, 제도: 가족, 부족, 국가, 학교, 대통령제, …
- 인간의 고정 관념, 사고 체계: 시각, 습관, 편집증, …
- 시장에서 정착되는 제품들
- 알고리즘이 만들어내는 주식투자전략
- 알파고의 가치망
- 테니스의 스윙폼
- …

Optimization은 가장 수준 높은 끌개를 찾는 것
- 저수준 끌개(low-quality local optimum)에 고착되어 버리지 않도록
- 다양한 끌개에 접할 수 있도록 넓은 탐색 기능 필요

방대하고 황량한 공간: TSP의 끌개수

문제 크기(정점수)	끌개의 수(평균)
10	4
20	170
100	3.4×10^{16} (3경 4천조)





모든 솔루션의 수: 9.3×10^{157}
끌개의 비율: 3×10^{141} 솔루션당 하나씩의 끌개

- ✓ 지금은 8천개짜리 문제를 푼다
- ✓ 최적화 알고리즘은 이런 공간을 돌아다니는 교통수단이다