5장. 문제해결 방법 (Problem solving)

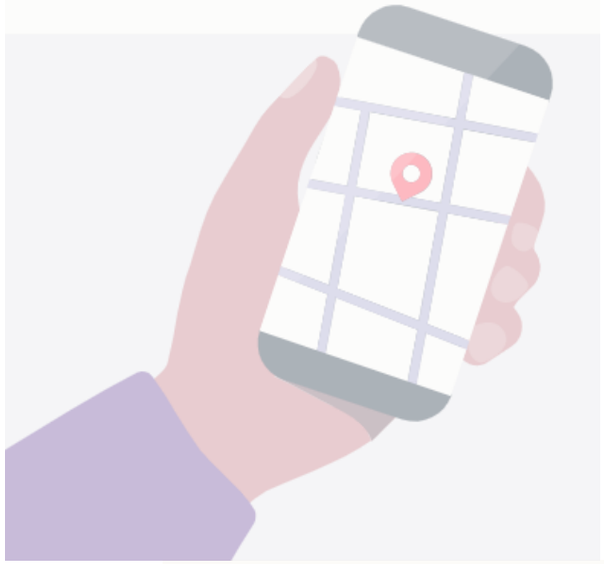
숙명여자대학교
SOOKMYUNG WOMEN'S UNIVERSITY

Contents

I. 탐색 및 문제해결

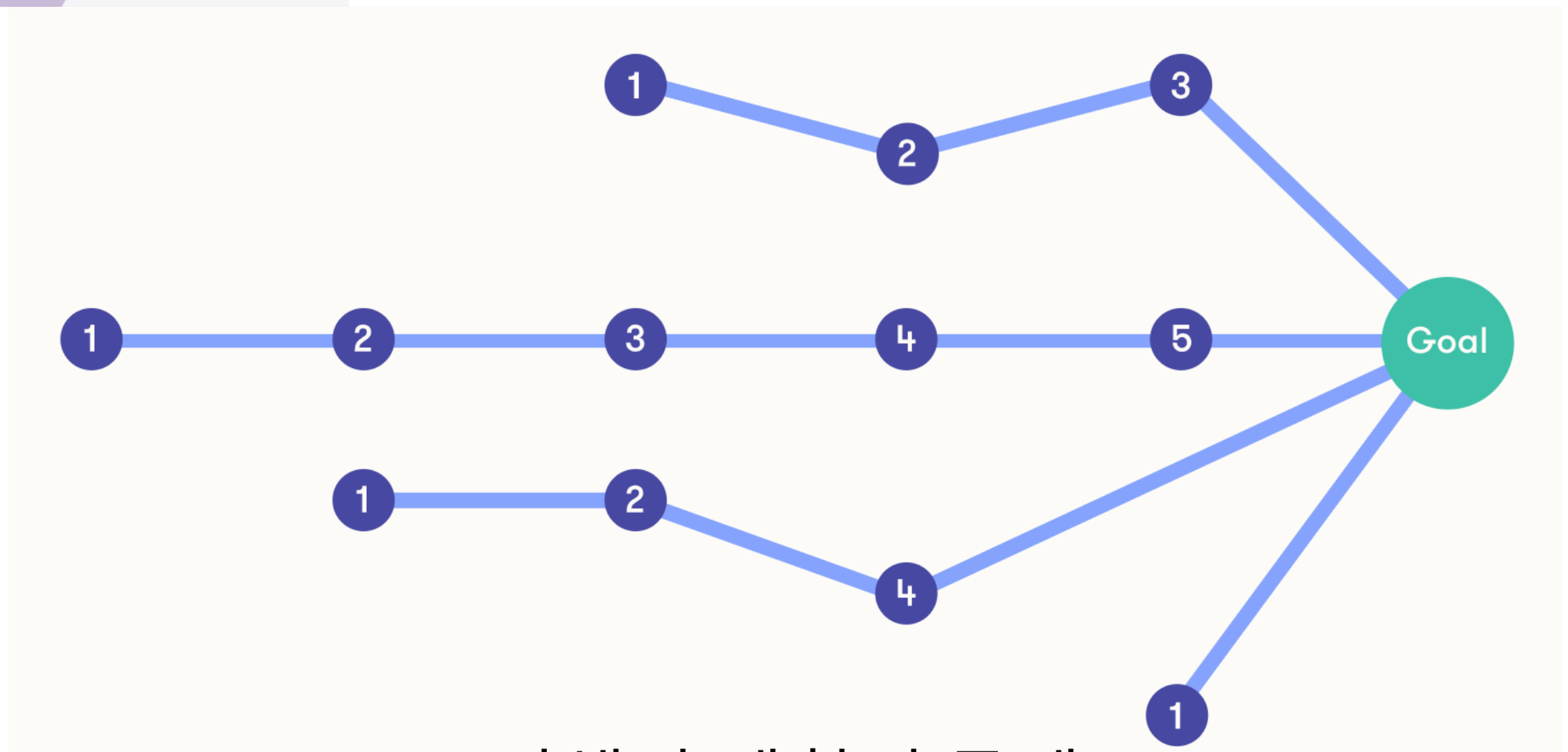
II. AI 탐색 기법

III. 탐색과 게임



일상에서의 탐색

A에서 B로 가기



검색과 계획의 문제

탐색 문제 (1/2)

- 탐색문제들
  - 네비게이션
  - 자율주행차
  - 체스게임 : 상대방으로부터 말을 안전하게 유지하기 위해
  - ➢ 시간, 노력, 비용 등 여러 면에서 굉장히 효율적인 알고리즘들이 존재함

- 궁극적인 목적이 탐색 알고리즘 그 자체는 아니지만, 문제 해결 과정의 첫 단계가 선택과 그 결과를 정의하는 것 ⇒ 여기에 탐색 기법이 필요함

탐색 문제 (2/2)

- 탐색문제의 2가지 유형
  - 하나의 에이전트로 정적 환경에서의 탐색 문제
  - 두 에이전트가 서로 경쟁하는 게임에서의 탐색 문제

[참고]

- 에이전트(Agent) : 미리 결정된 목표를 달성하기 위해 작동하는 코드
  - 인공지능에서 프로그램을 부르는 용어
  - 챗봇, 자율주행자동차에 들어가 있는 AI 프로그램들

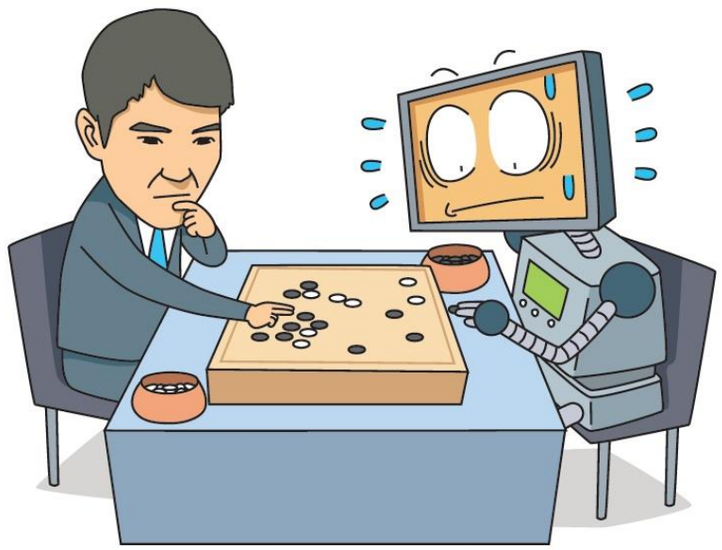
알파고는 수를 어떻게 읽었을까? (1/2)

- 상대의 수를 읽고, 가능한 수에 대해 탐색을 한 후, 가장 최적의 수를 선택

➢ 문제는 가능한 경우의 수가 너무 많아서 제한된 시간내에 완료할 수 없음

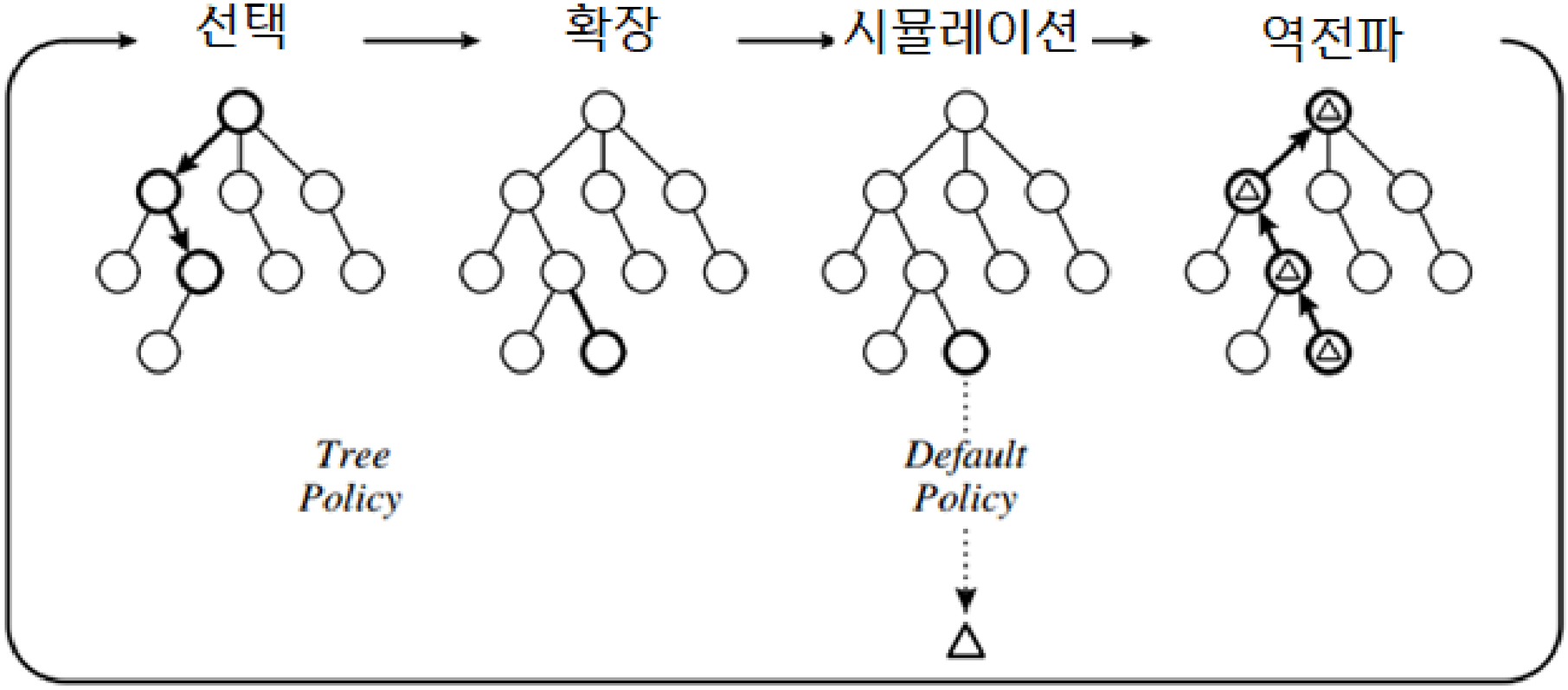
➢ 딥러닝을 사용해서 가장 가능성 있는 수를 추려내고 이들 수에 대해서만 탐색을 함

- 몬테카를로 트리 탐색 : 선택지 중 가장 유리한 선택을 하도록 돕는 알고리즘



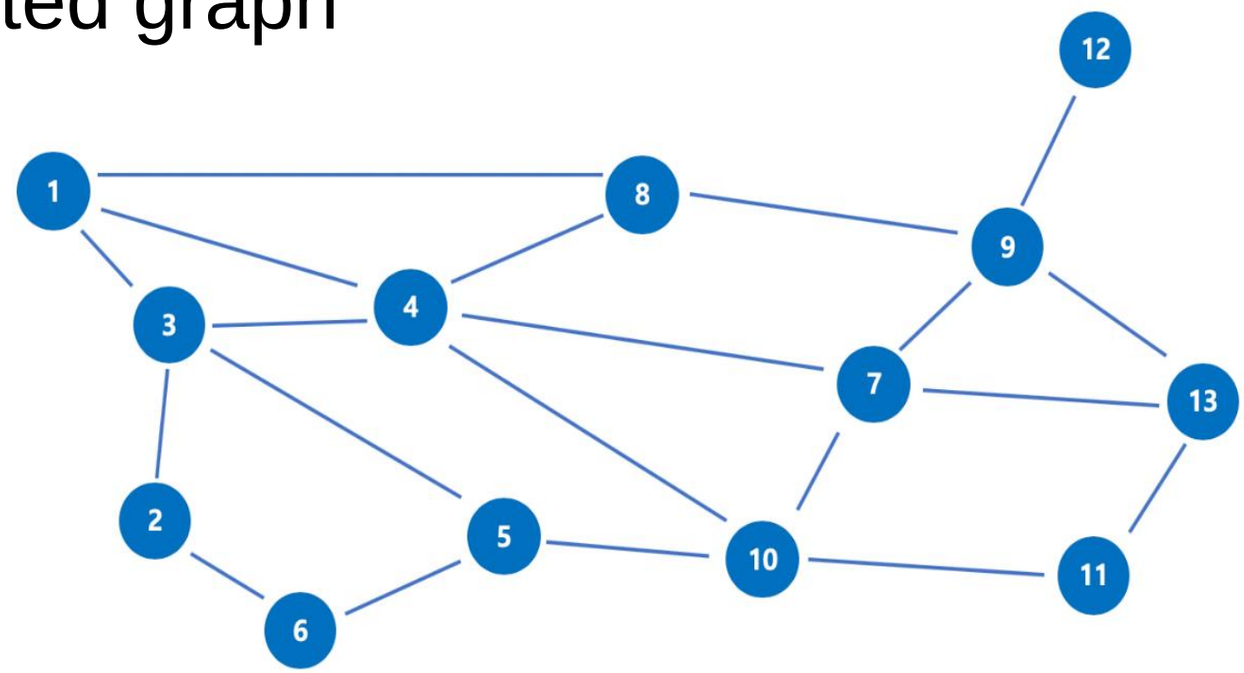
알파고는 수를 어떻게 읽었을까? (2/2)

- 몬테카를로 트리 탐색(MCTS) 알고리즘
  - MCTS의 목적은 가장 최선의 움직임을 분석하는 데에 있으며, 무작위로 이동하면서 샘플을 얻은 것으로 트리를 확장



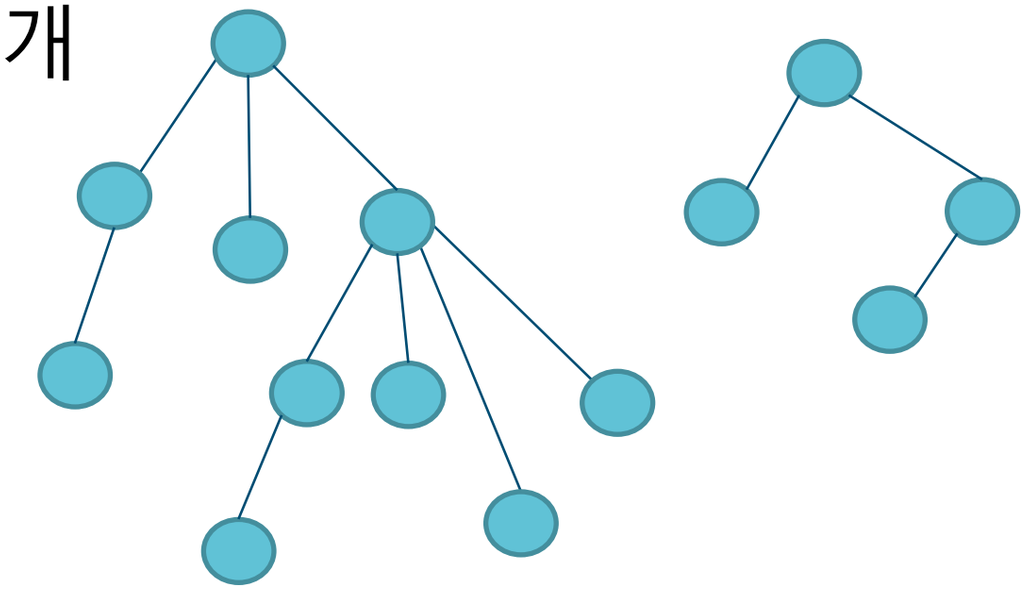
[참고] 그래프와 트리 (1/2)

- 그래프(Graph)
  - 노드(vertex)와 노드 간을 연결하는 간선(edge)으로 구성된 자료 구조
  - 순환구조, 비순환구조
  - directed graph, undirected graph
  - 일종의 네트워크 모델



[참고] 그래프와 트리 (2/2)

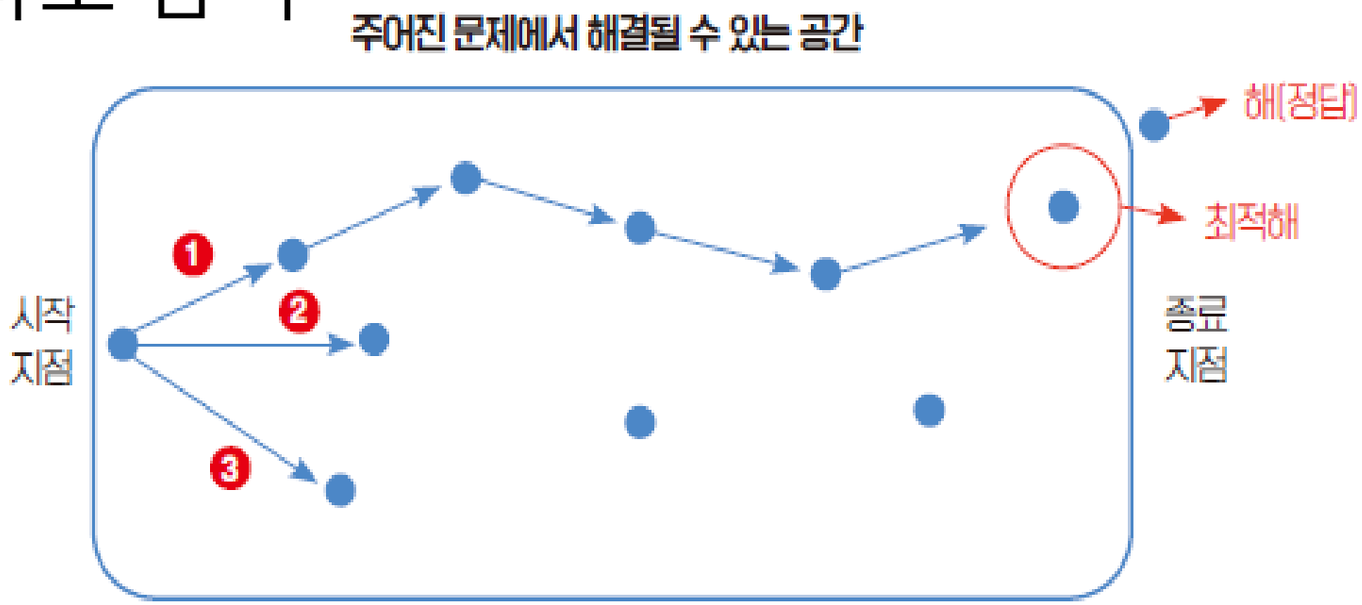
- 트리(Tree)
  - 두 개의 노드 사이에 반드시 1개의 경로만을 가지며, 사이클이 존재하지 않는 그래프
  - 부모-자식 관계가 존재
  - 레벨이 존재 (최상위 노드 = Root)
  - 노드가 N개 이면, 간선은 N-1개



탐색(search)이란 ? (1/2)

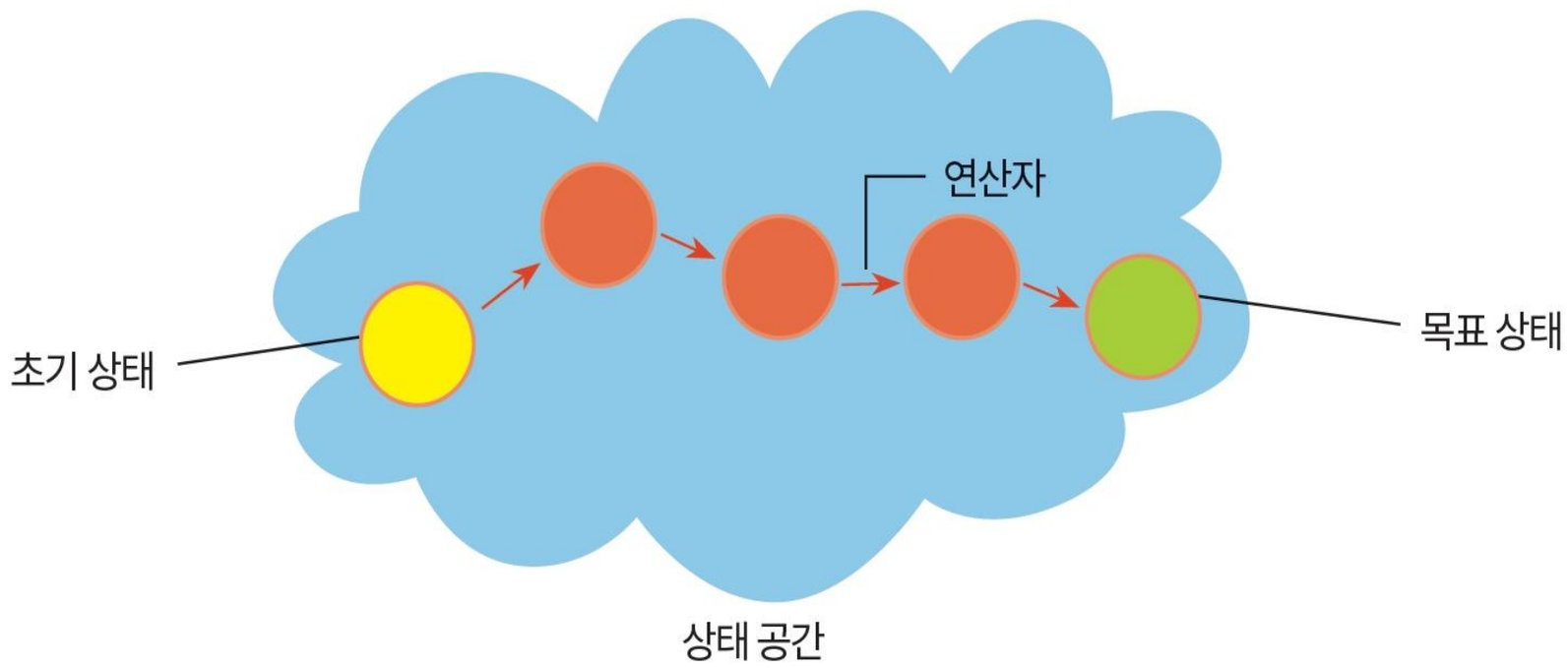
- 인공지능에서의 탐색 개념
  - 주어진 문제를 해결하기 위해 처음 시작부터 우리가 최종적으로 원하는 상태까지 진행하는 과정을 통해 최적의 해답(정답)을 찾음

➢ 최종적으로 원하는 상태까지 갈 수 있도록 진행하는 과정이 바로 탐색



탐색(search)이란 ? (2/2)

- 상태공간에서 시작상태에서 목표 상태까지의 경로를 찾는 것
  - 상태공간(state space)
  - 연산자 : 다음 상태를 생성하는 것
  - 초기상태
  - 목표상태

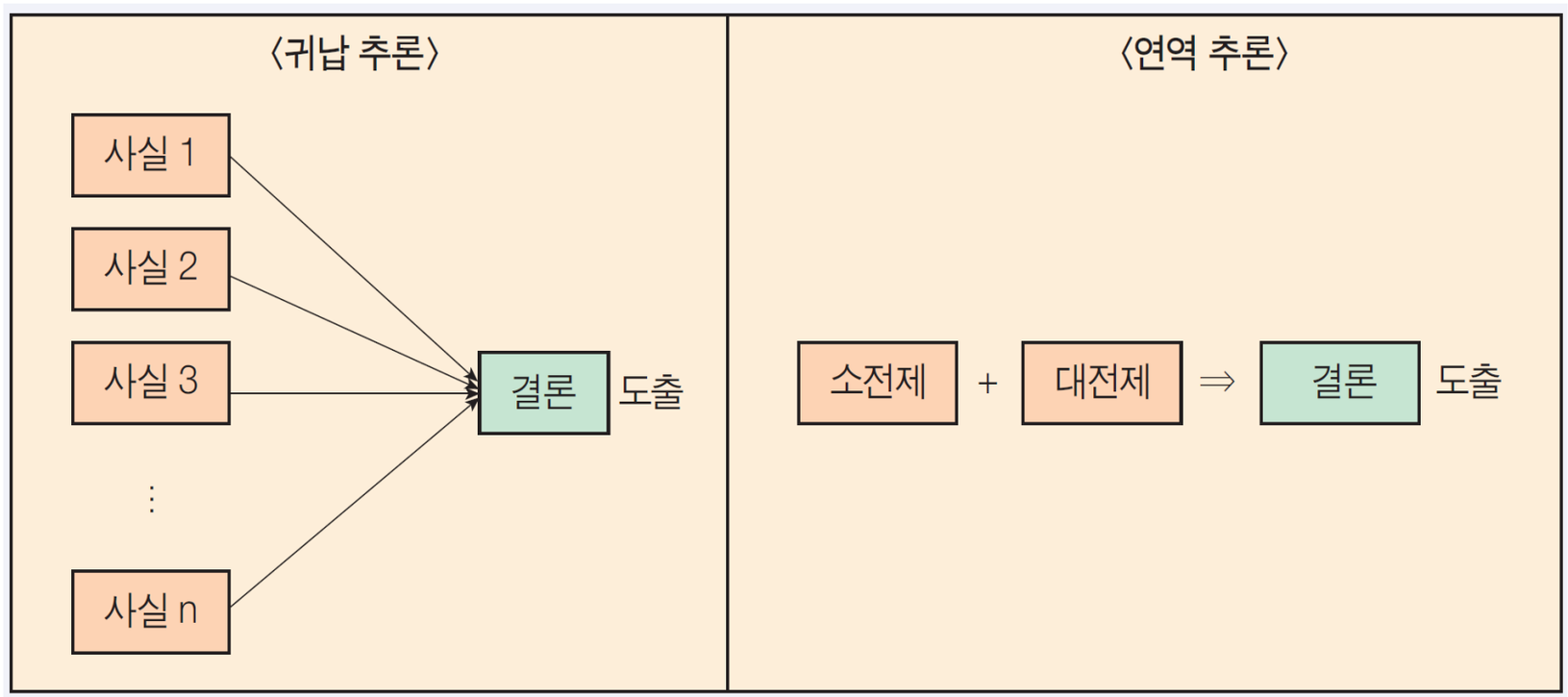


탐색으로 추론하기 (1/5)

- 추론(Inference)
  - 이미 알고 있는 정보로부터 논리적인 결론을 도출하는 것
  - 탐색으로 추론하기 : 탐색을 통해 문제 해결 방법을 찾아가는 과정
- 인공지능에서의 추론
  - 규칙기반 인공지능은 논리바탕의 규칙을 통해 추론 => 주어진 몇가지 사실과 규칙을 바탕으로 새로운 사실을 생성해 낼 수 있기 때문에 추론이 중요함
  - 논리와 추론은 전문가 시스템 등의 응용에 필수적인 기반 지식

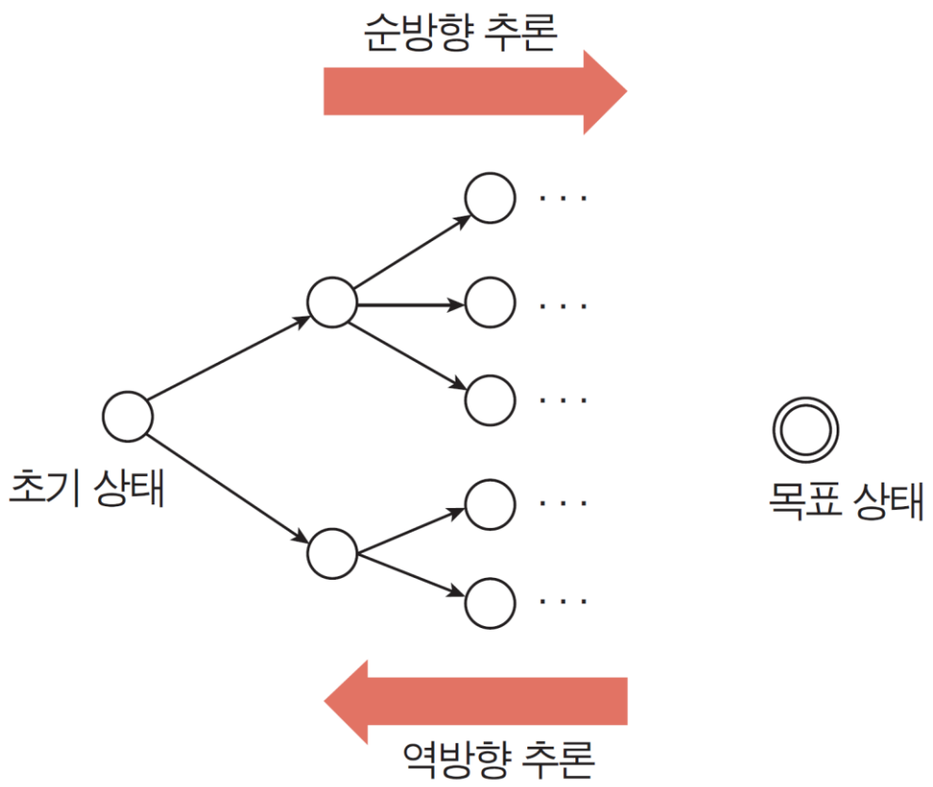
탐색으로 추론하기 (2/5)

- 추론의 종류
  - 귀납추론(induction)
  - 연역추론(deduction)
  - 유비추론(abduction, analogy, 유추)



탐색으로 추론하기 (3/5)

- 많은 초기의 AI 프로그램이 기본탐색 알고리즘을 사용
  - 미로를 탐색하는 것처럼 단계별로 진행
  - 막다른 곳에 도달할 때마다 백트래킹
- 추론을 통한 탐색
  - 순방향 추론은 출발 상태에서 목표 상태로 진행
  - 역방향 추론은 목표 상태에서 출발 상태로 진행



탐색으로 추론하기 (4/5)

- 조합폭발의 문제
  - 탐색과정에서 가능한 선택의 수가 폭발적으로 증가하는 것

7		2
8	5	4
6	3	1

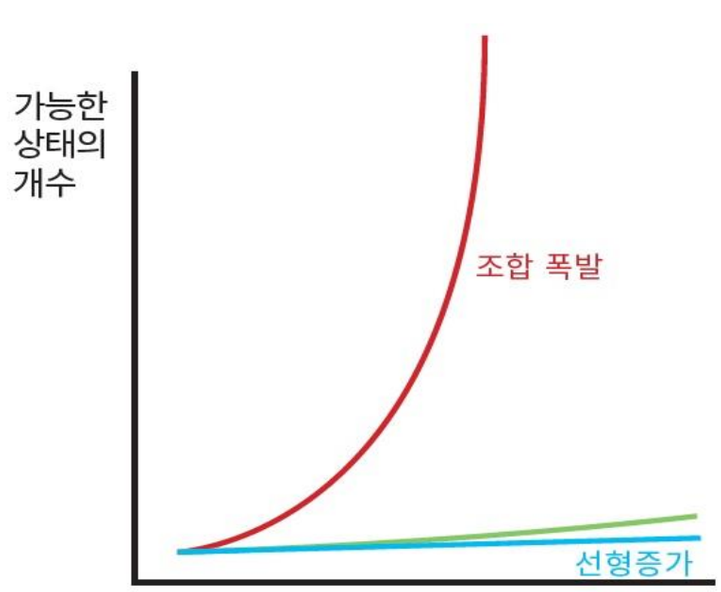
10^5개의 상태

4	10	15	3
1		13	7
6	2	9	5
8	14	11	12

10^{13}개의 상태

2	6	3	14	17
18	1	13	8	10
16	11	15		24
21	23	9	12	4
7	5	22	20	19

10^{25}개의 상태



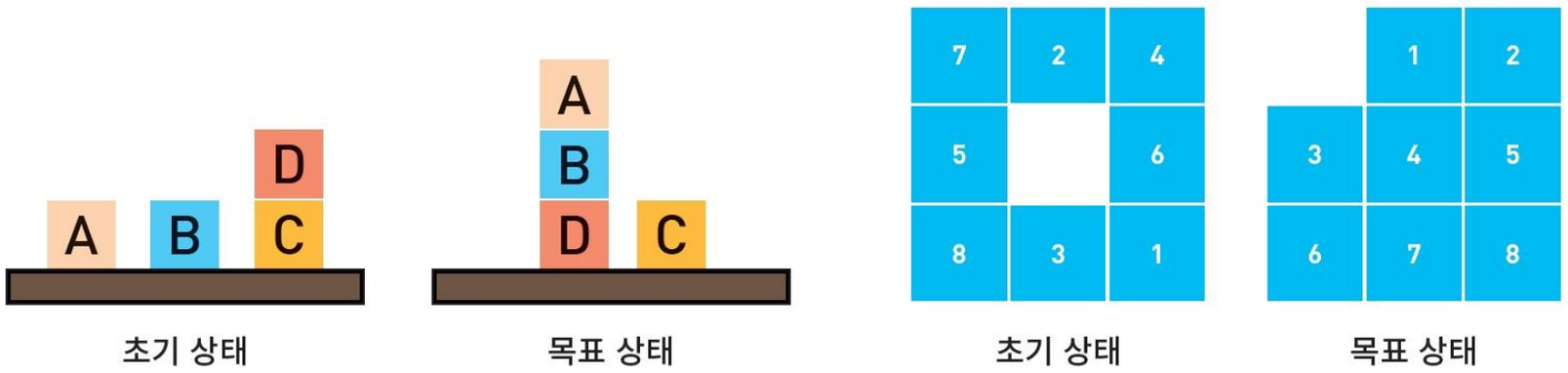
➢ 조합폭발의 문제로 현실에서는 문제 해결이 어려워짐

탐색으로 추론하기 (5/5)

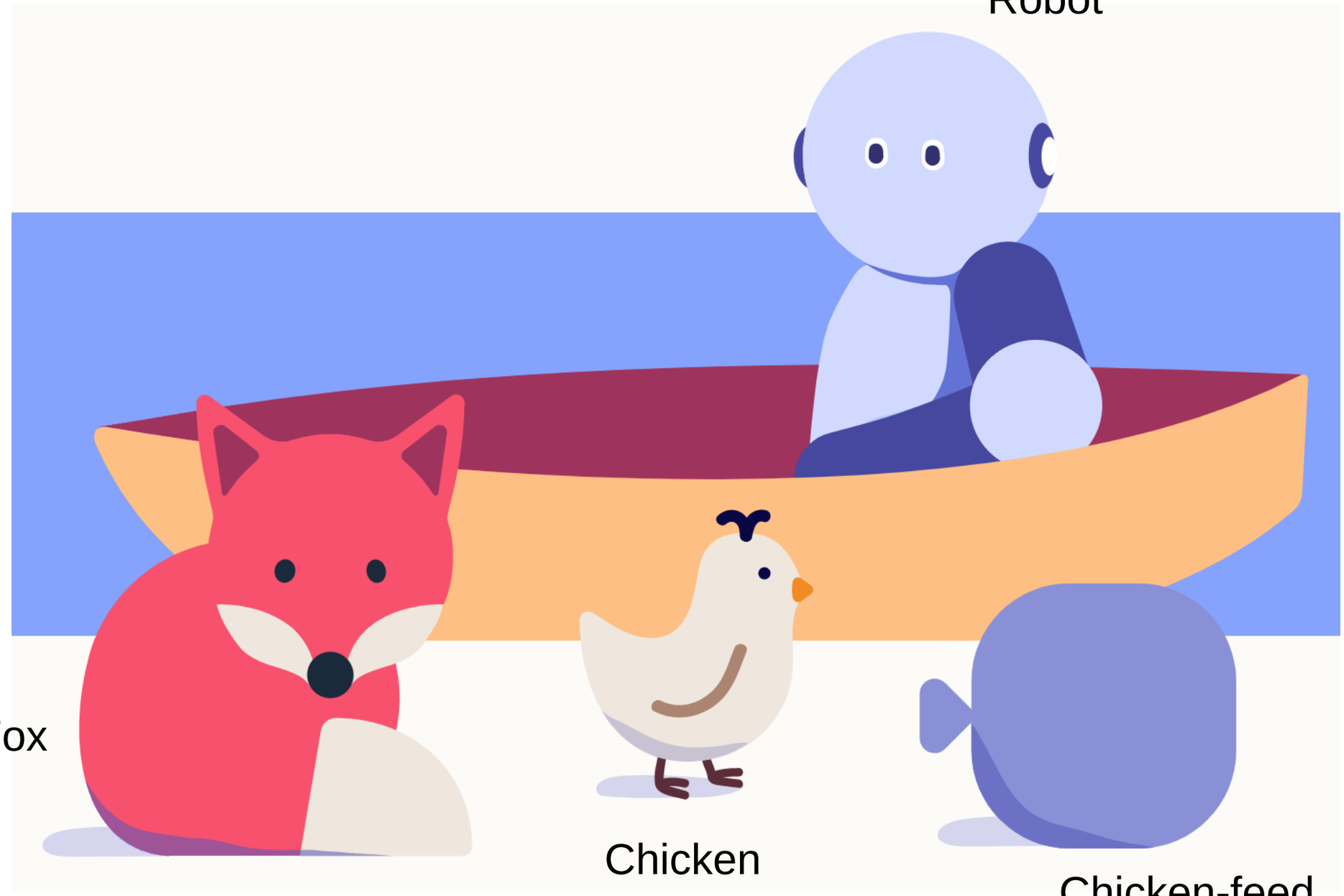
- 1970~80년대 초기 인공지능 연구개발자들의 시도
  - 탐색으로 추론하기 알고리즘의 일반화된 버전 만들기 : 이세상에 존재하는 모든 문제를 하나의 일반화된 알고리즘으로 해결이 가능하도록
  - 문제가 복잡해지면 연산자가 너무 많아져서 처리를 도저히 할 수가 없었고, 너무 많은 시간과 메모리가 필요함
  - ➢ 문제를 간단하게 축소를 해서 문제 해결 방법을 찾고, 그걸 현실문제로 확장하는 방식으로 해결하고자 함
  - ➢ 장난감 문제(Toy problem)

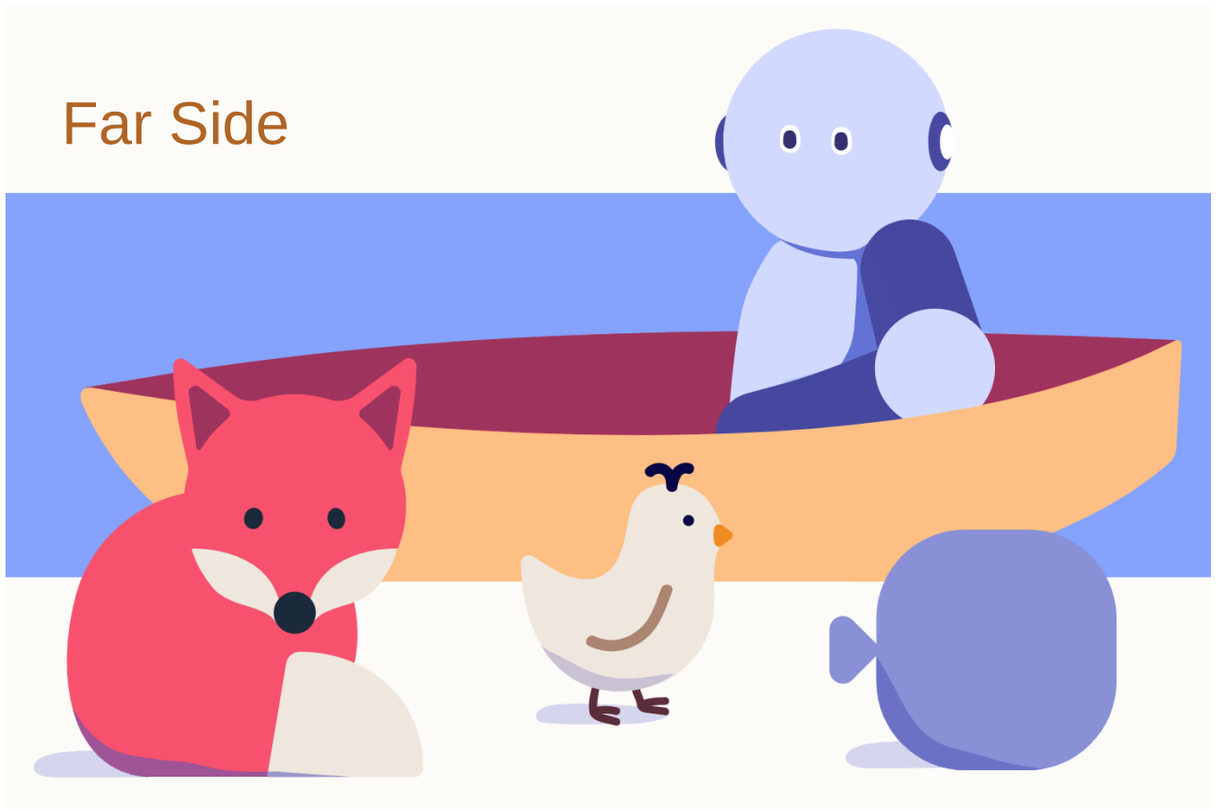
장난감 문제 (Toy problem)

- 현실의 문제를 아주 간단하게 만든 문제
- 인위적으로 문제를 쉽게 축소하여 이를 해결하는 방법을 설명하는 것
- AI 알고리즘은 거의 완벽하게 장난감 문제를 해결
- 현실로 가져오면 ? 조합폭발(combinatorial explosion)



장난감 문제 : Chicken Crossing Puzzle



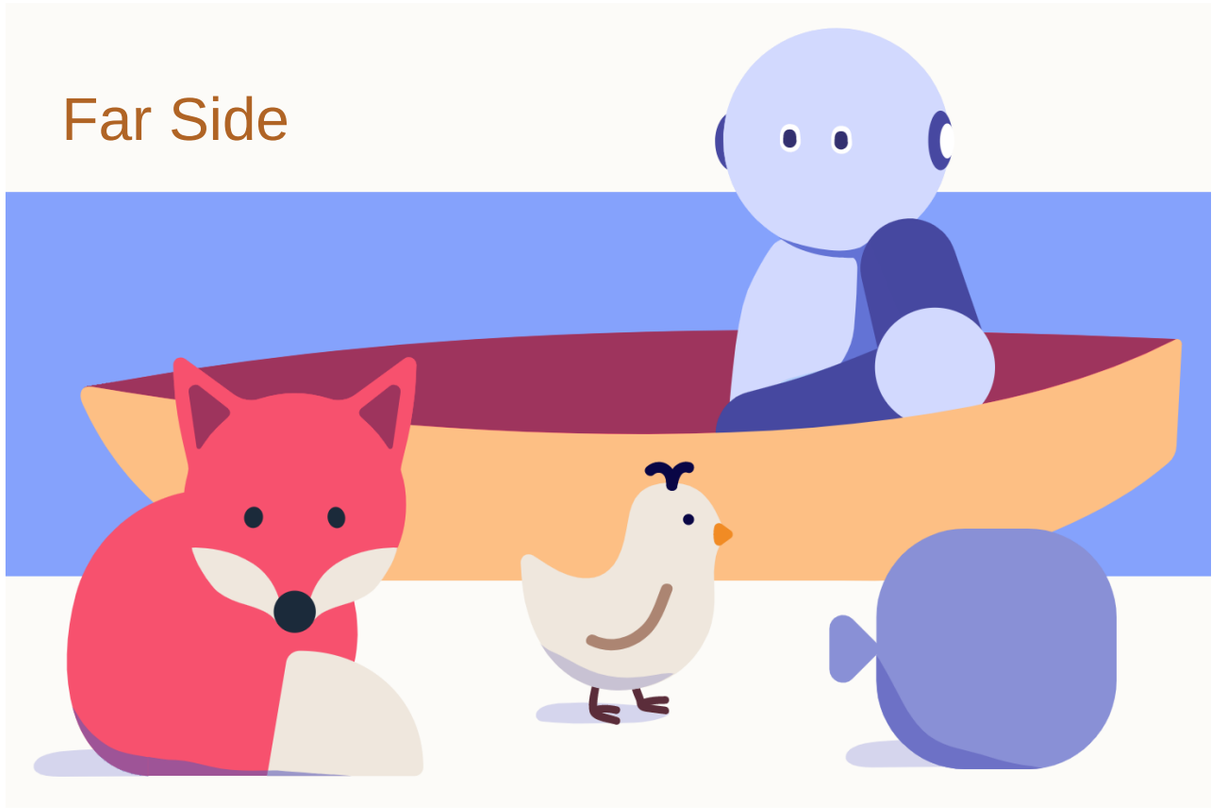


초기상태
(Starting States)

목표상태
(Goal States)

Possible States

State	Robot	Fox	Chicken	Chicken-feed
NNNN	Near side	Near side	Near side	Near side
NNNF	Near side	Near side	Near side	Far side
NNFN	Near side	Near side	Far side	Near side
NNFF	Near side	Near side	Far side	Far side
NFNN	Near side	Far side	Near side	Near side
NFNF	Near side	Far side	Near side	Far side
NFFN	Near side	Far side	Far side	Near side
NFFF	Near side	Far side	Far side	Far side
FNNN	Far side	Near side	Near side	Near side
FNNF	Far side	Near side	Near side	Far side
FNFN	Far side	Near side	Far side	Near side
FNFF	Far side	Near side	Far side	Far side
FFNN	Far side	Far side	Near side	Near side
FFNF	Far side	Far side	Near side	Far side
FFFN	Far side	Far side	Far side	Near side
FFFF	Far side	Far side	Far side	Far side

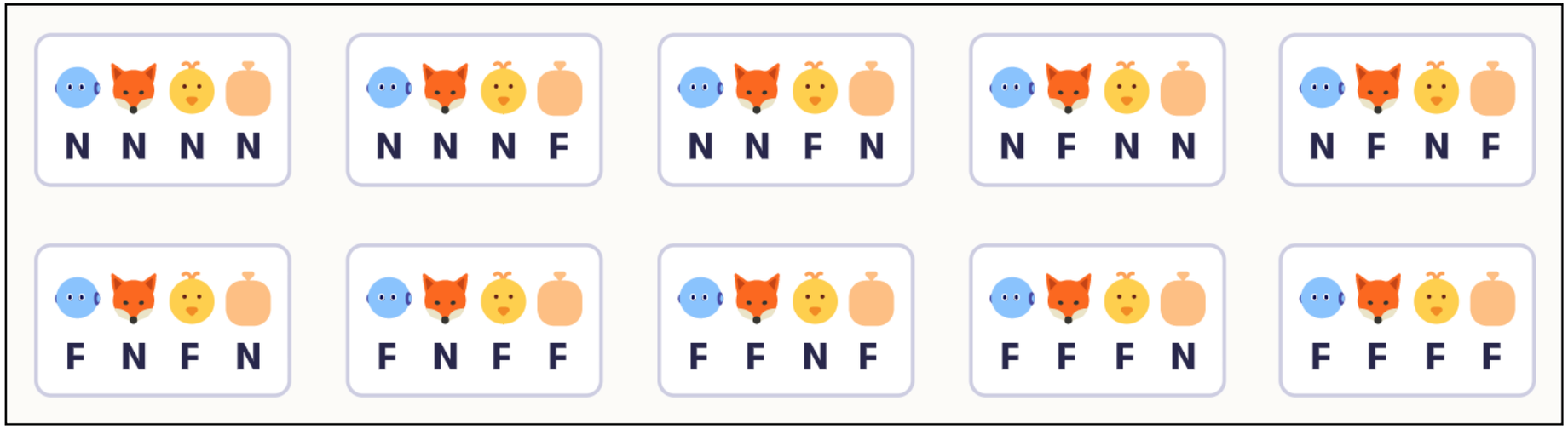


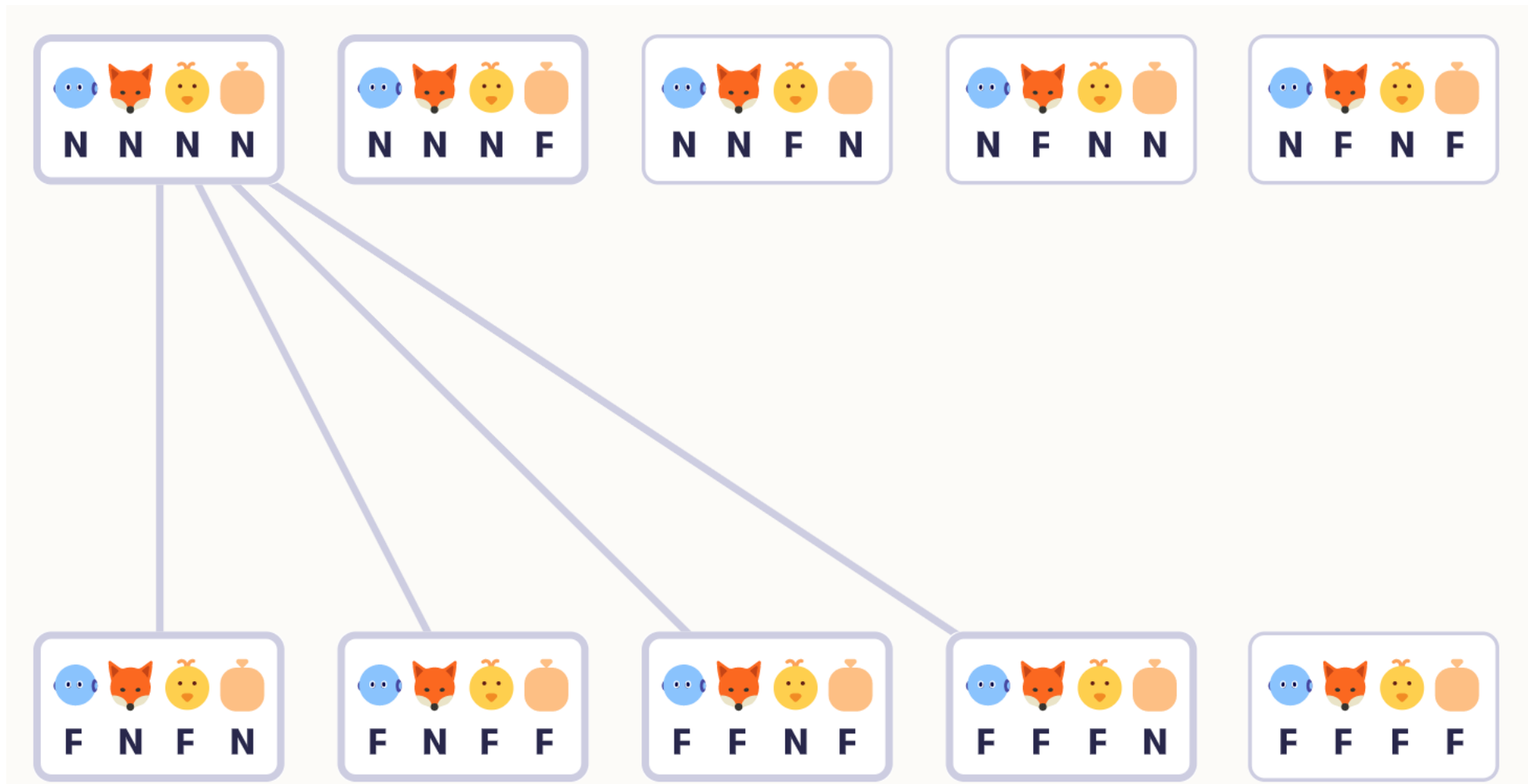
초기상태
(Starting States)

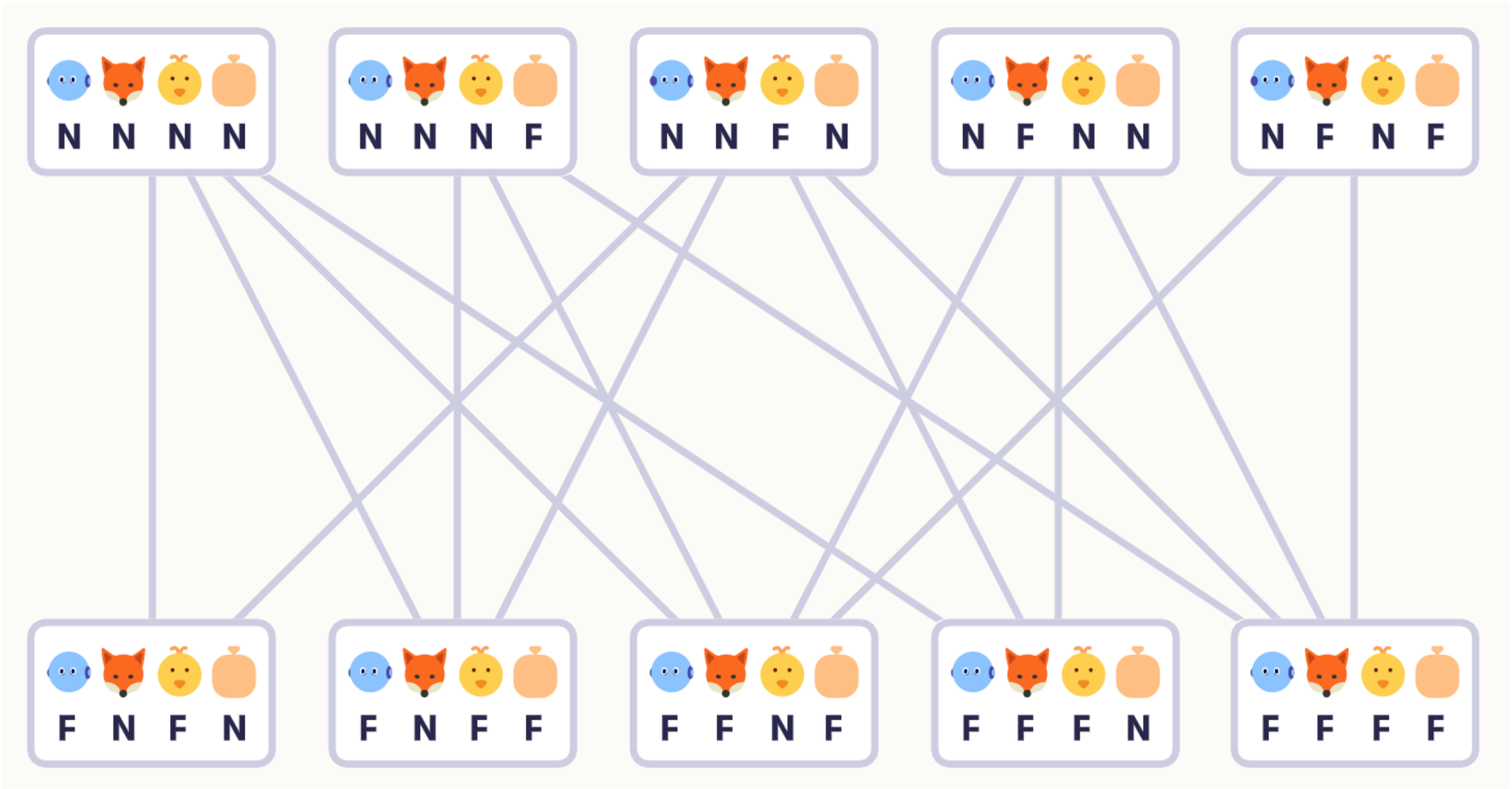
Possible States

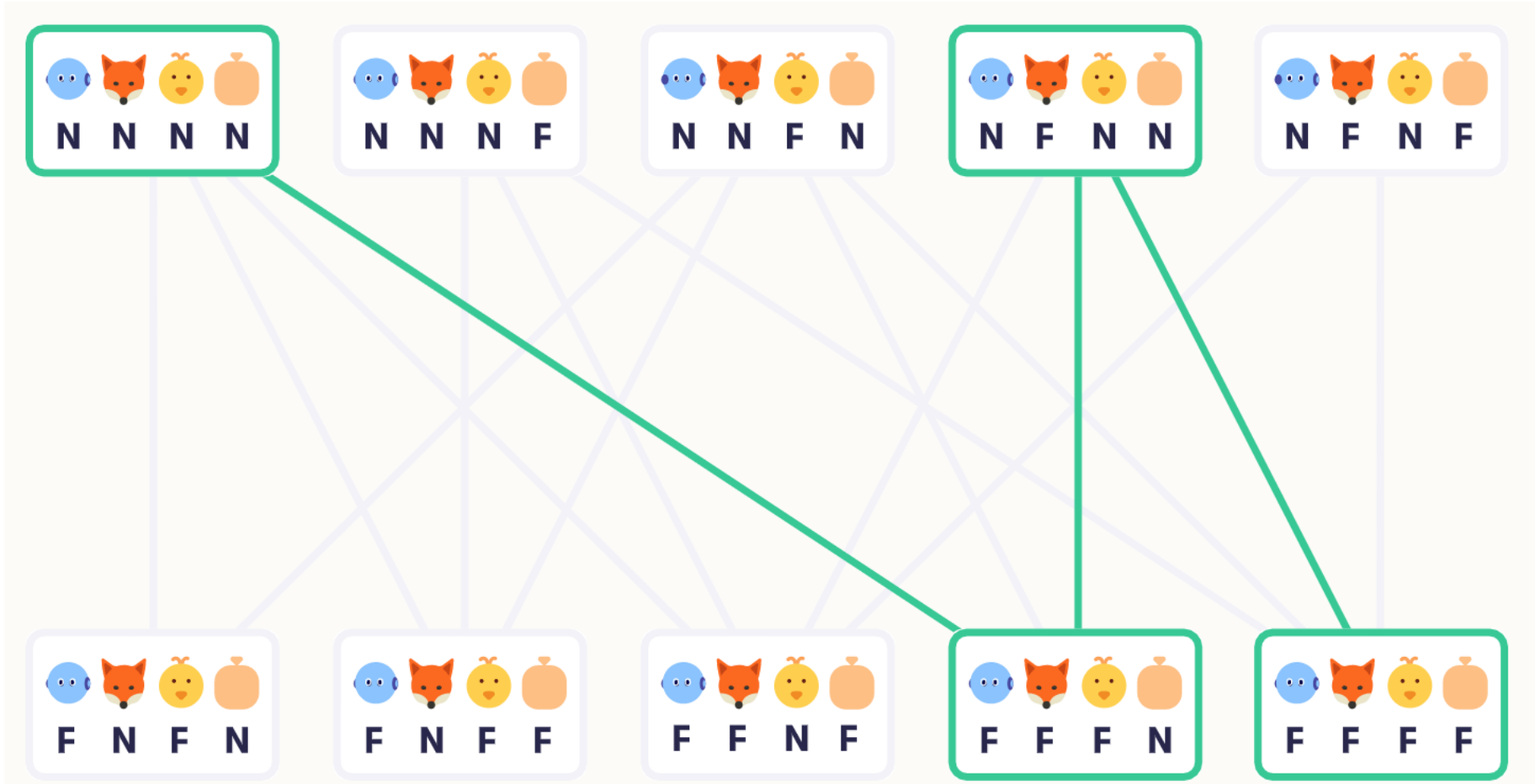
State	Robot	Fox	Chicken	Chicken-feed
NNNN	Near side	Near side	Near side	Near side
NNNF	Near side	Near side	Near side	Far side
NNFN	Near side	Near side	Far side	Near side
~~NNFF~~	~~Near side~~	~~Near side~~	~~Far side~~	~~Far side~~
NFNN	Near side	Far side	Near side	Near side
NFNF	Near side	Far side	Near side	Far side
~~NFFN~~	~~Near side~~	~~Far side~~	~~Far side~~	~~Near side~~
~~NFFF~~	~~Near side~~	~~Far side~~	~~Far side~~	~~Far side~~
~~FNNN~~	~~Far side~~	~~Near side~~	~~Near side~~	~~Near side~~
~~FNNF~~	~~Far side~~	~~Near side~~	~~Near side~~	~~Far side~~
FNFN	Far side	Near side	Far side	Near side
FNFF	Far side	Near side	Far side	Far side
~~FFNN~~	~~Far side~~	~~Far side~~	~~Near side~~	~~Near side~~
FFNF	Far side	Far side	Near side	Far side
FFFN	Far side	Far side	Far side	Near side
FFFF	Far side	Far side	Far side	Far side

목표상태
(Goal States)

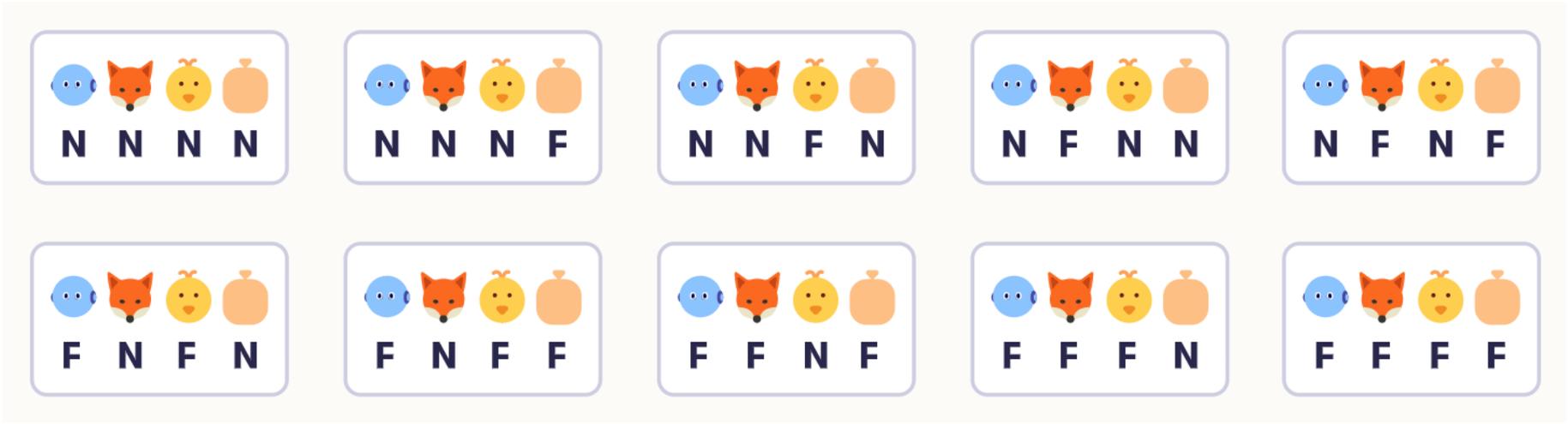

N N N N
N N N F
N N F N
N F N N
N F N F
F N F N
F N F F
F F N F
F F F N
F F F F


N N N N
N N N F
N N F N
N F N N
N F N F
F N F N
F N F F
F F N F
F F F N
F F F F


N N N N
N N N F
N N F N
N F N N
N F N F
F N F N
F N F F
F F N F
F F F N
F F F F

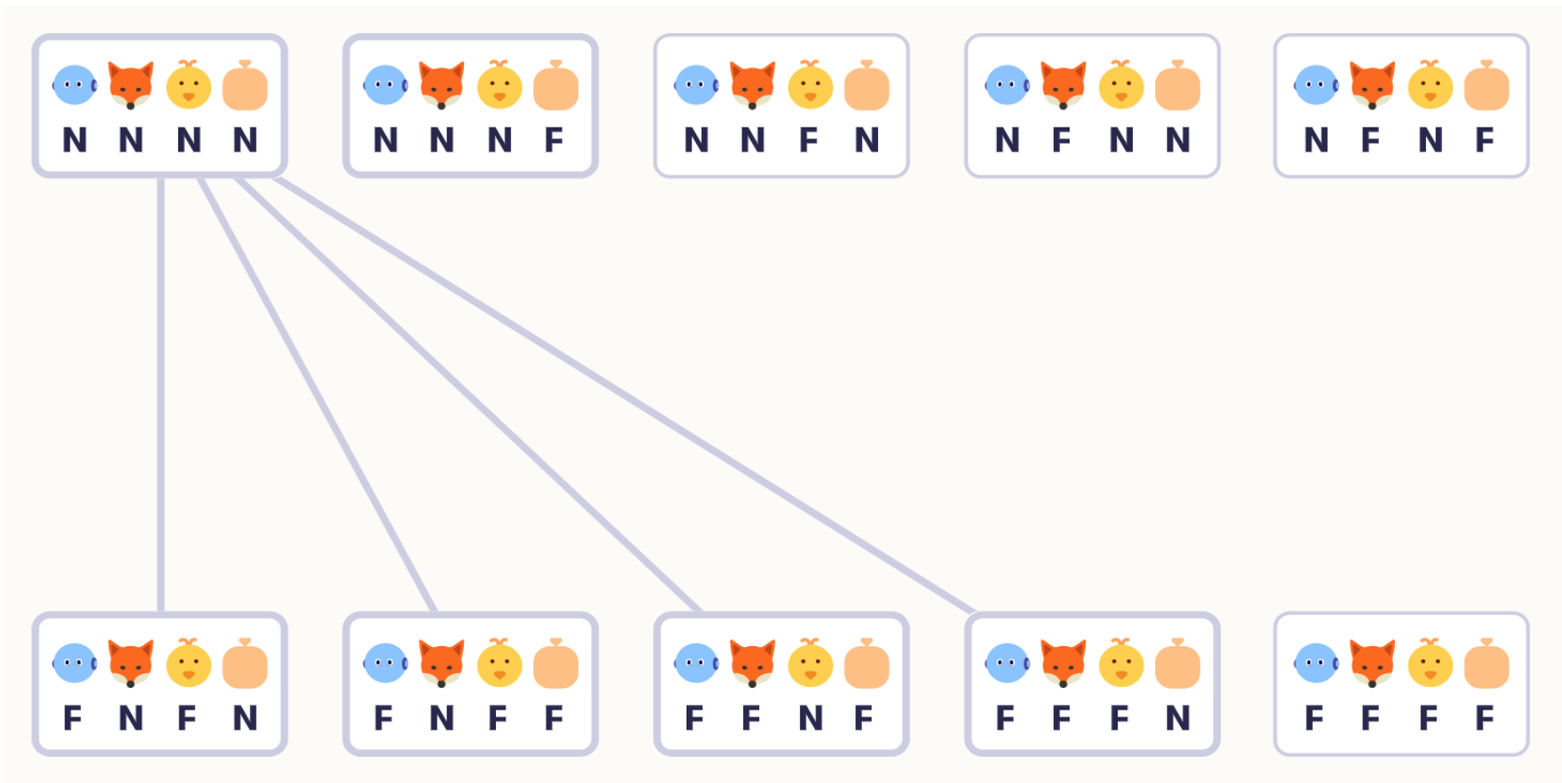

N N N N
N N N F
N N F N
N F N N
N F N F
F N F N
F N F F
F F N F
F F F N
F F F F

상태공간(State space)



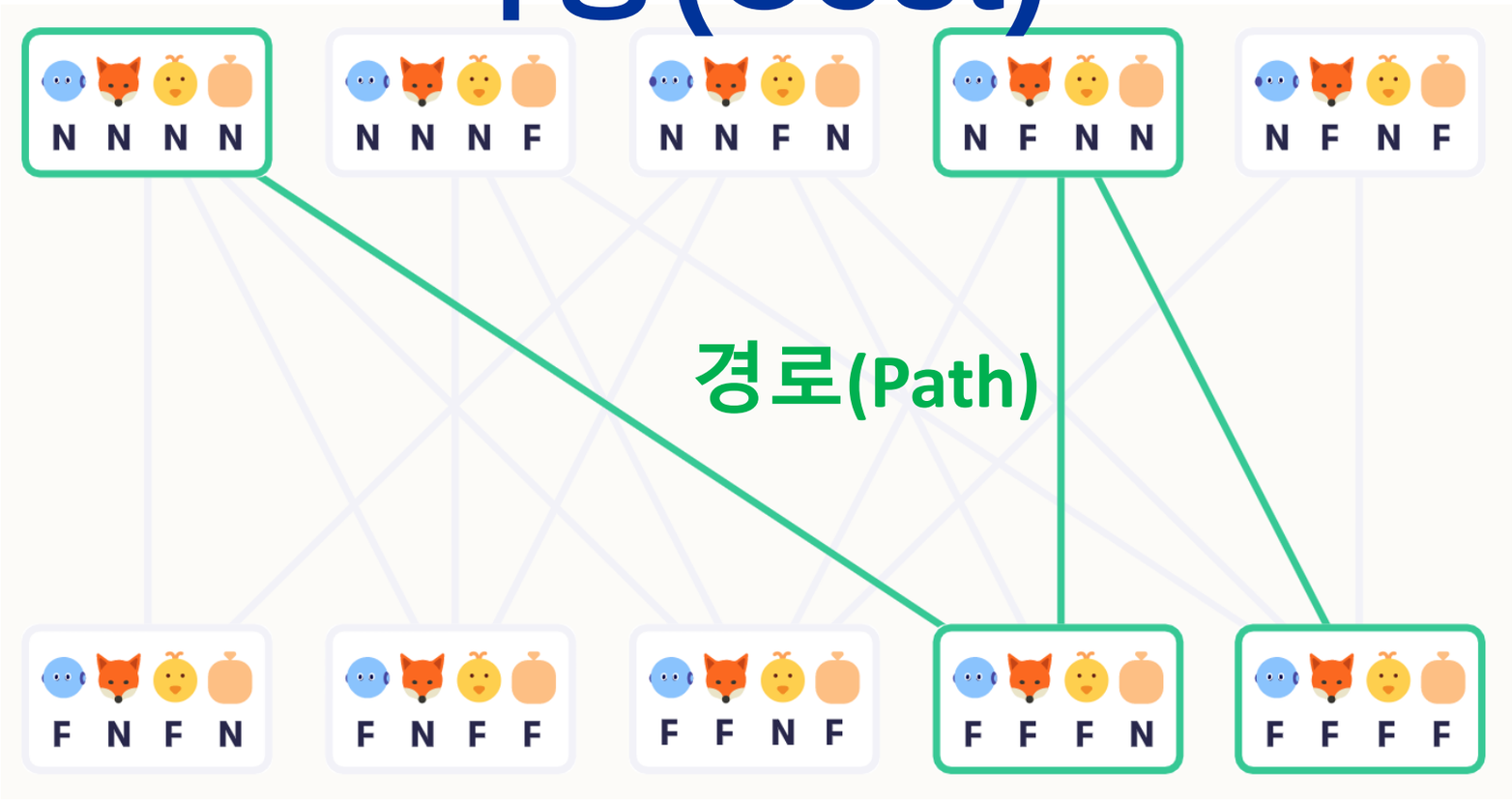
- 가능한 상태의 집합
  - NNNN에서 FFFF까지 허용되는 10개의 상태
  - 퍼즐 규칙이 허용하지 않는 상태는 포함 안됨
  - A에서 B로 이동하는 경우 : 시작점 A에서 도달할 수 있는 (x, y)좌표로 정의된 위치 집합

전이(Transition)



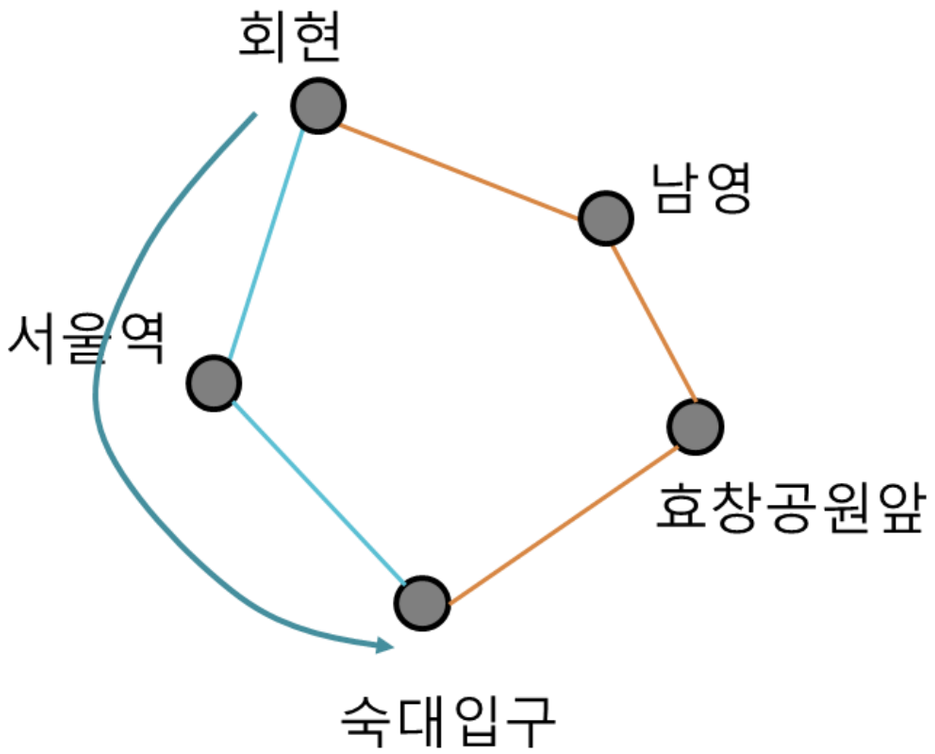
- 한 상태에서 다른 상태 사이에서의 가능한 이동
- 단일 작업으로 수행되는 직접 전환(direct transition)
- NNNN에서 FNFN으로 전이

비용(Cost)

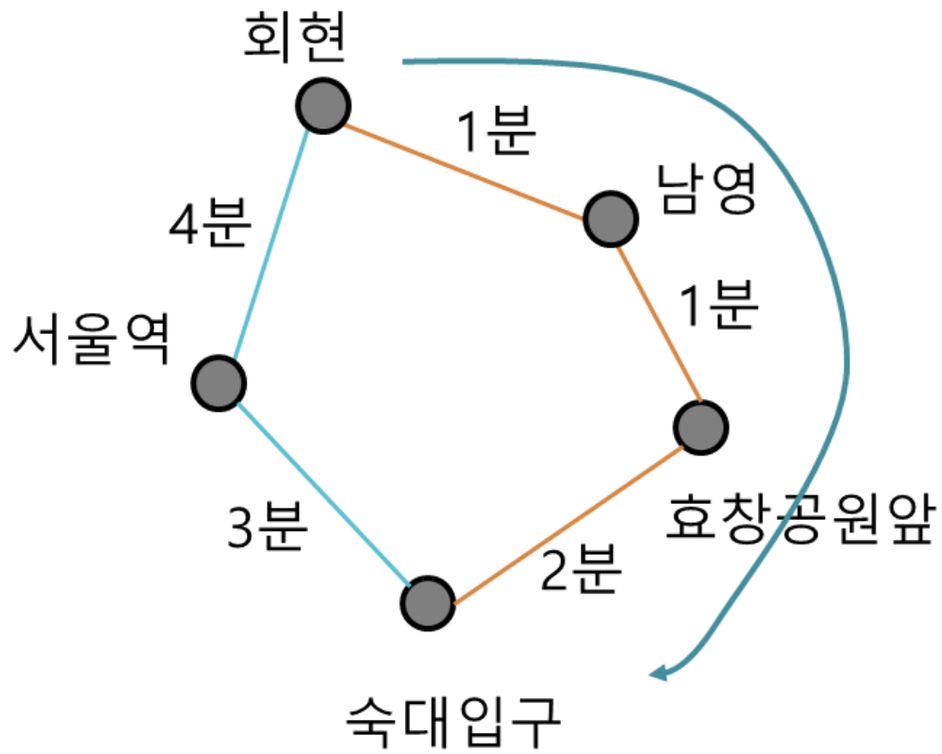


- 여러 가능한 경로 중 어떤 경로를 선택하게 되는 기준이 될 수 있음
  - 각 전이가 이동 거리 또는 소요 시간 등을 표현한다면, 더 적은 비용이 드는 경로를 선택하게 됨.

경로에서 비용의 의미



The shortest path :
회현 – 서울역 – 숙대입구



The shortest path :
회현 – 남영 – 효창공원앞 – 숙대입구

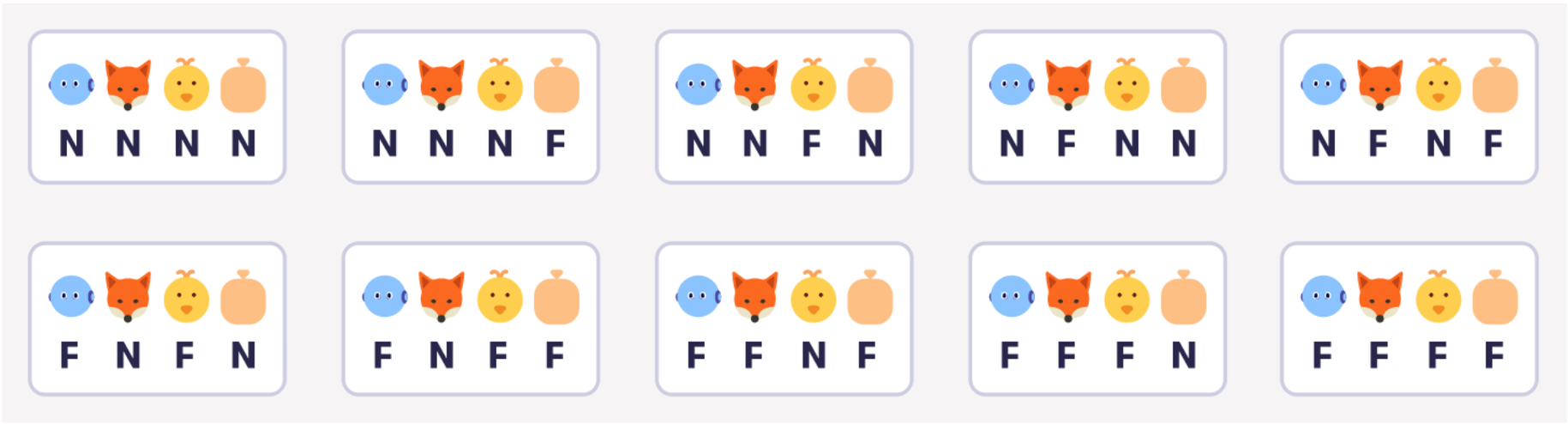
연습문제

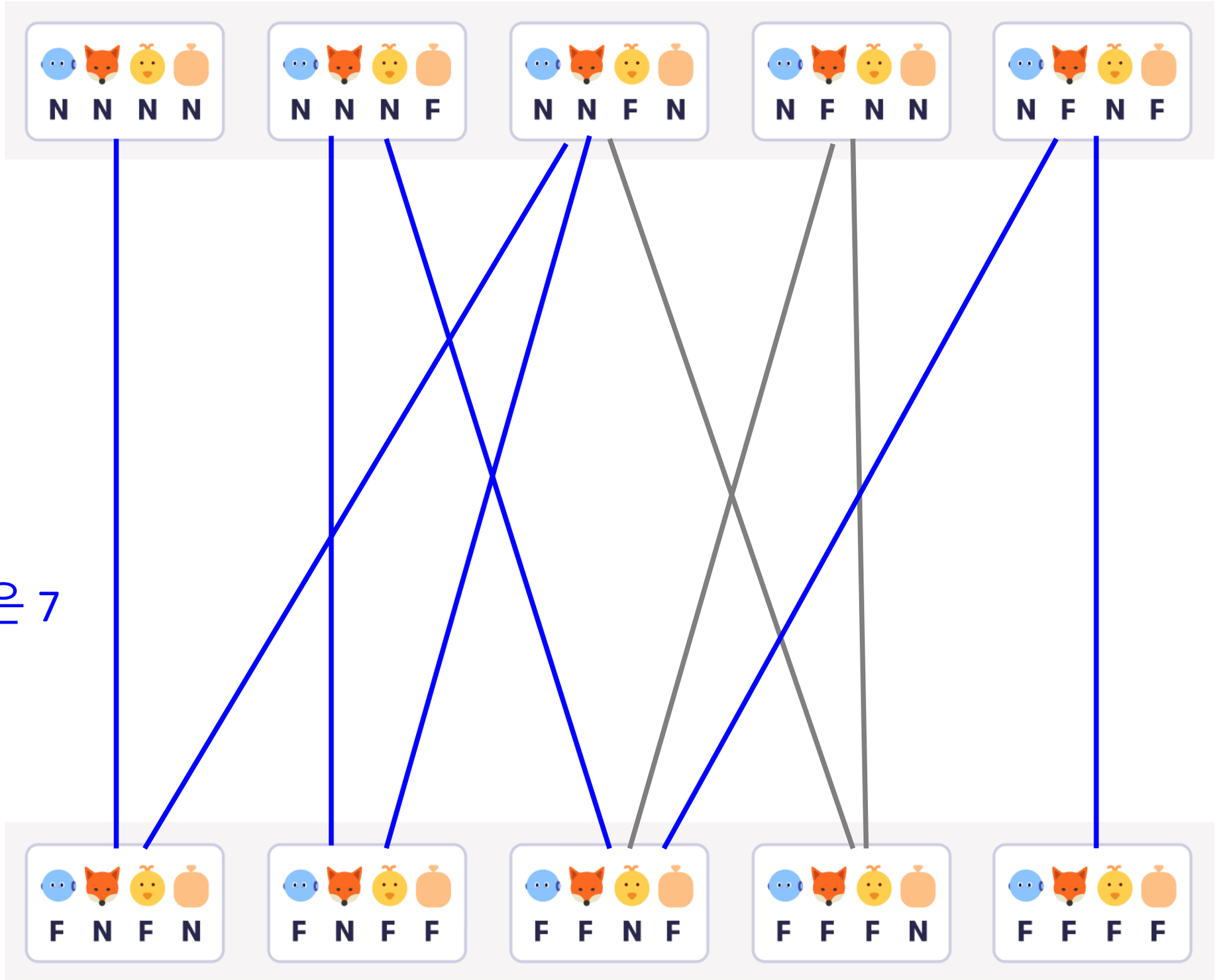
더 작은 보트

로봇이 보트에 딱 하나의 객체만 실을 수 있다면 ?

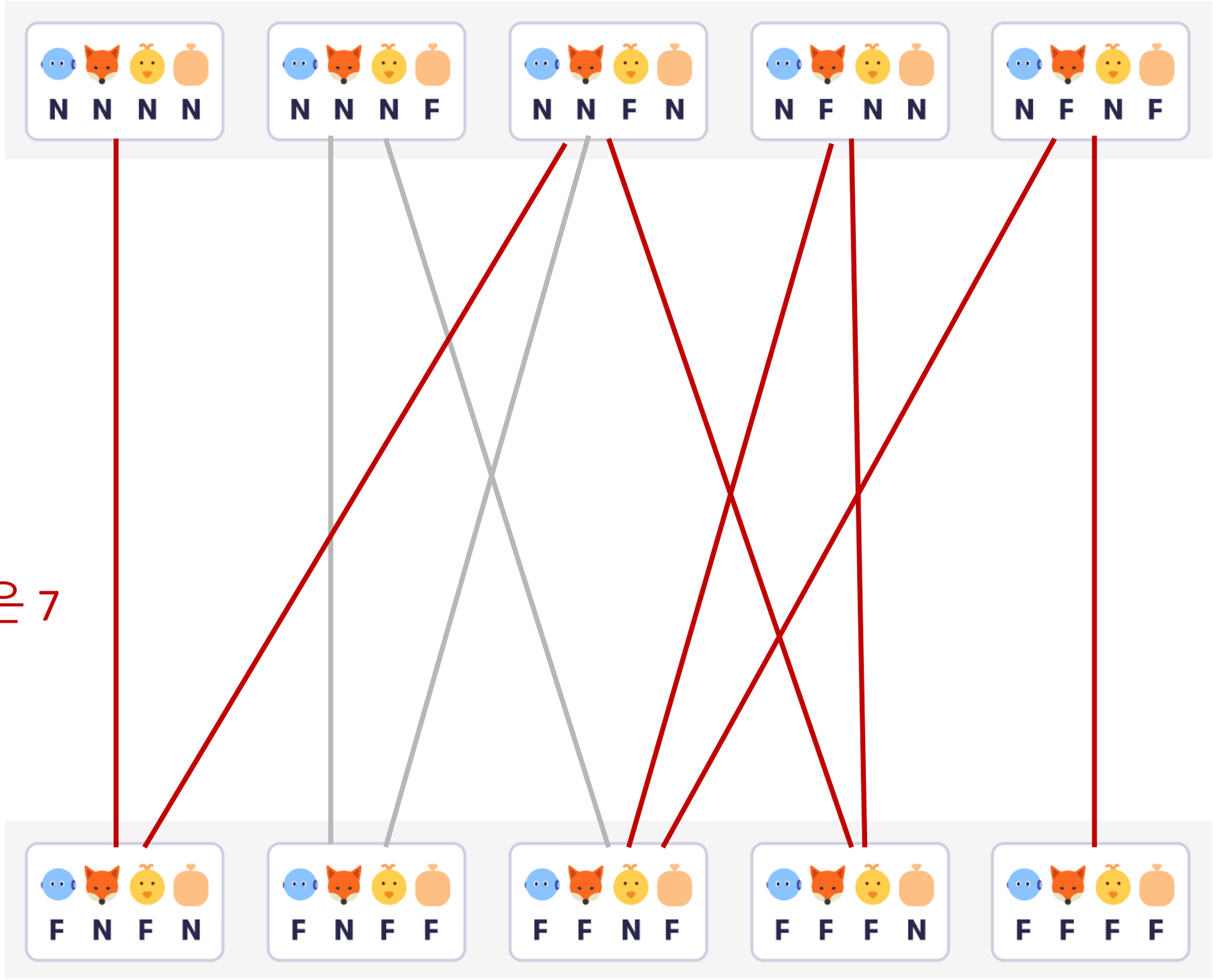
문제

NNNN에서 FFFF까지의 최단 경로를 찾고 그 경로의 전이 횟수를 계산해 보기



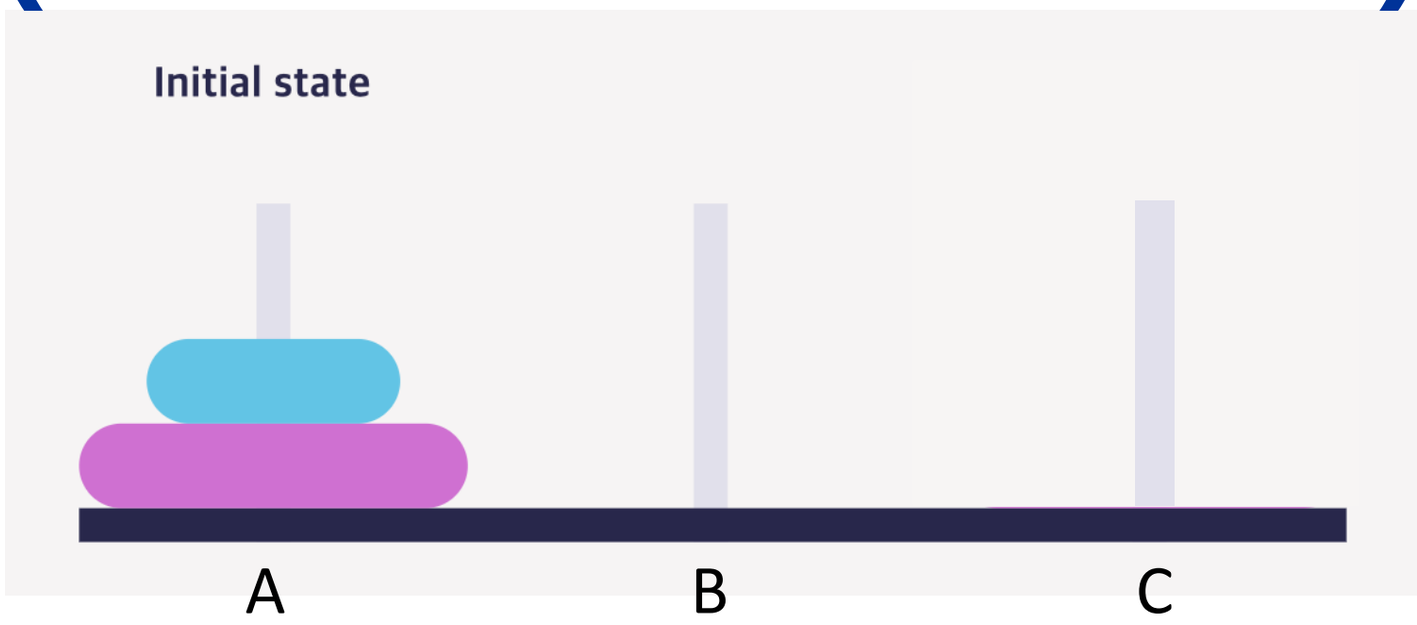


정답은 7



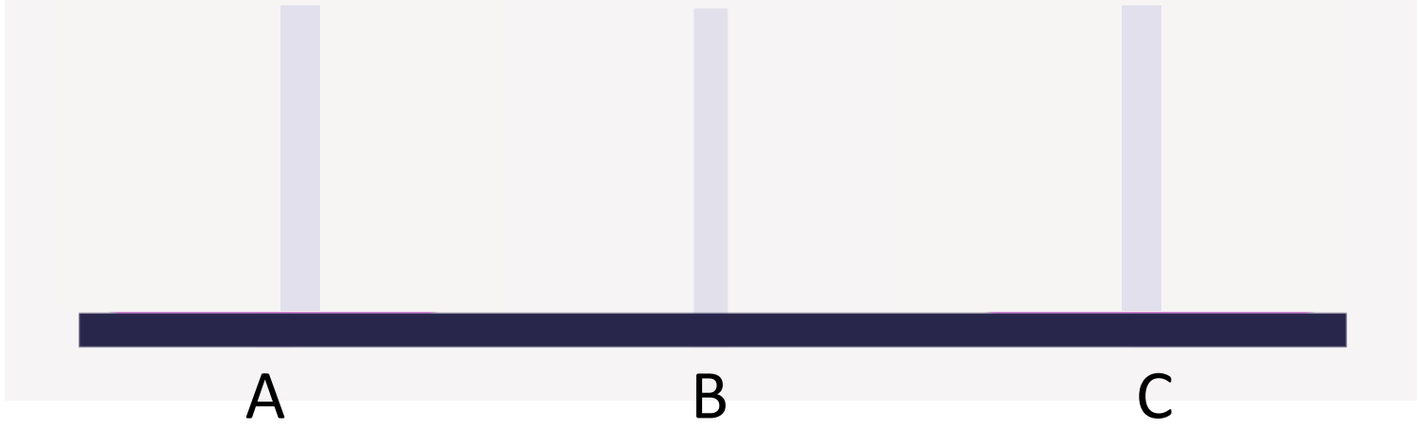
정답은 7

하노이탑 (1/2)
(The Towers of Hanoi)



- 이동 규칙
  - 한번에 한 개의 원판만 옮길 수 있음
  - 가장 위에 있는 원판만 이동할 수 있음
  - 큰 원판이 작은 원판 위에 있어서는 안됨

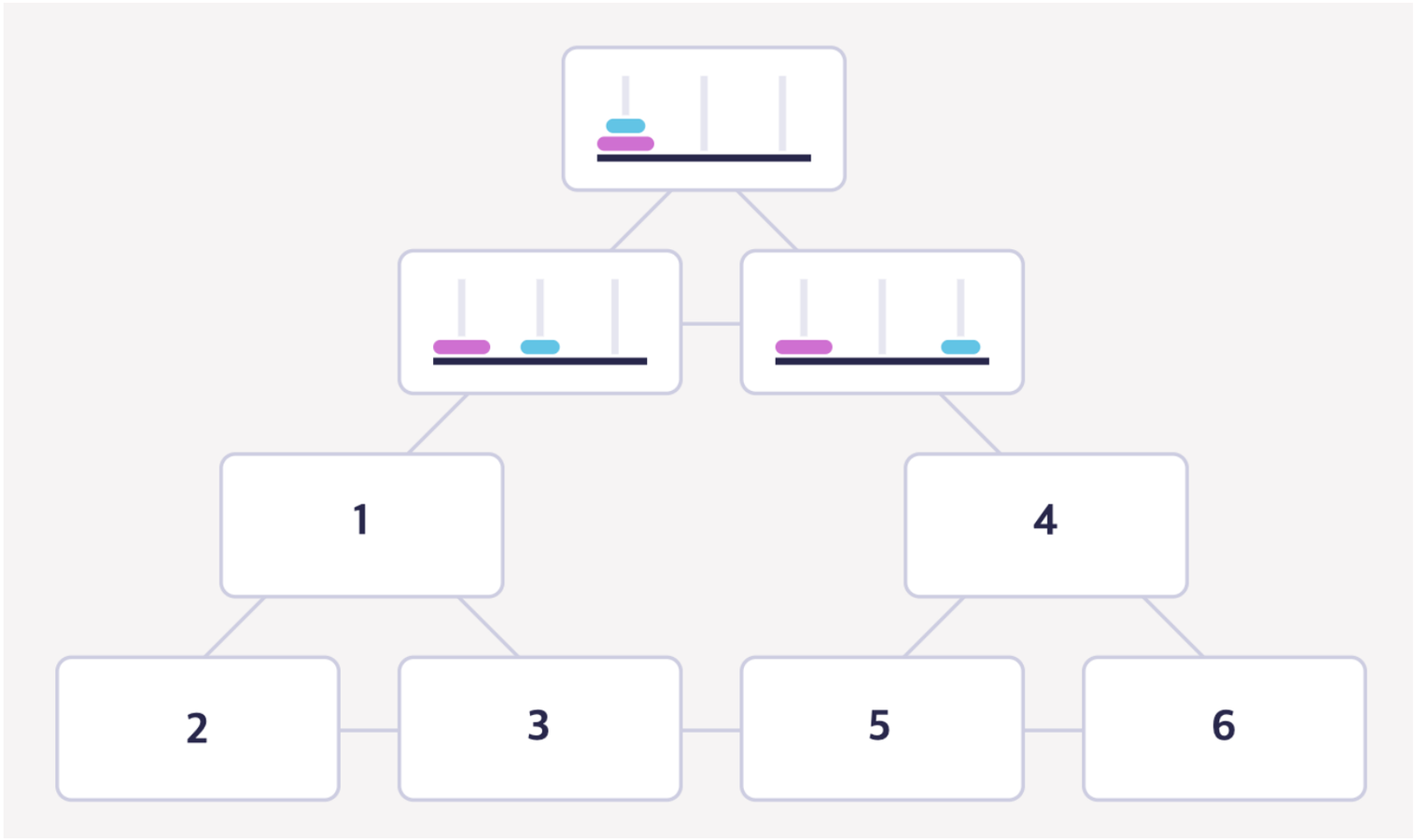
하노이탑 (2/2)
(The Towers of Hanoi)



- 상태공간 $S = \{(a_1, a_2) | \ a_i \in \{A, B, C\}\}$
- 초기 상태 $I = (A, A)$, 목표 상태 $G = (C, C)$
- 연산자

$$O = \{m(which, where) | \ which \in \{1, 2\}, where \in \{A, B, C\}\}$$

연습문제 : 하노이탑의 상태 전이도



정답

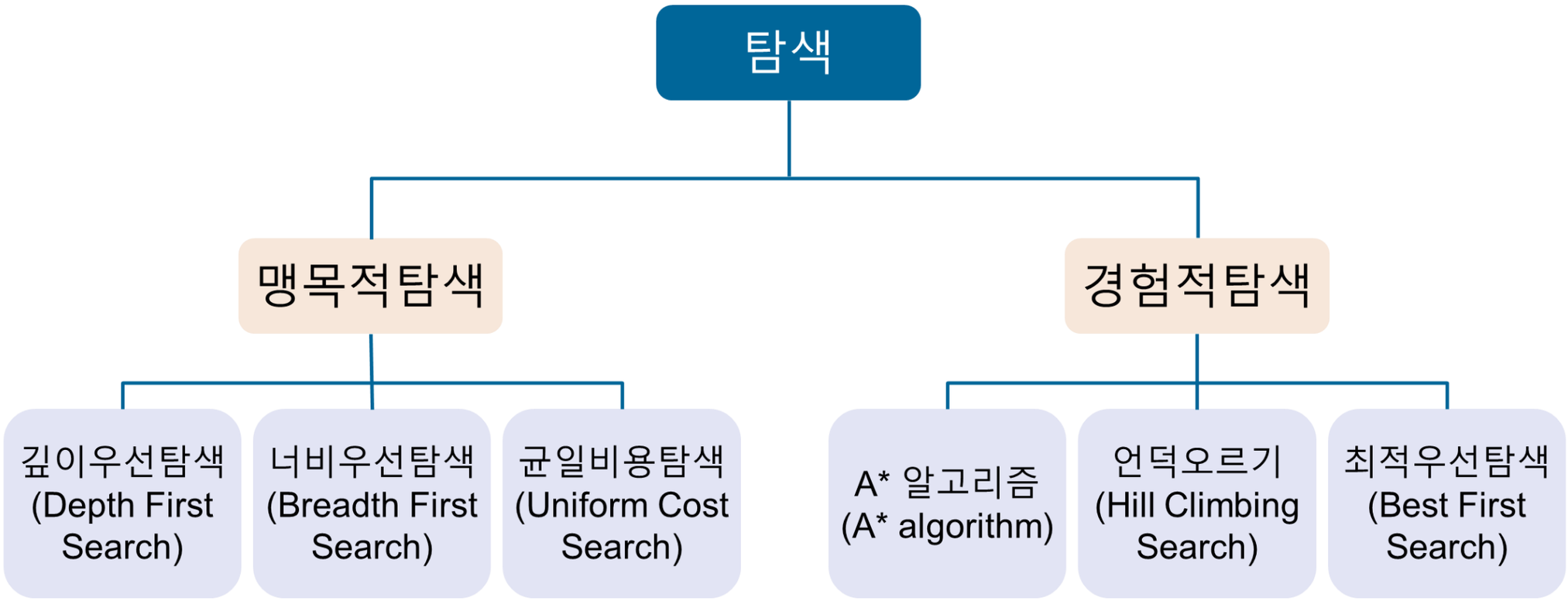
Contents

I. 탐색 및 문제해결

II. AI 탐색 기법

III. 탐색과 게임

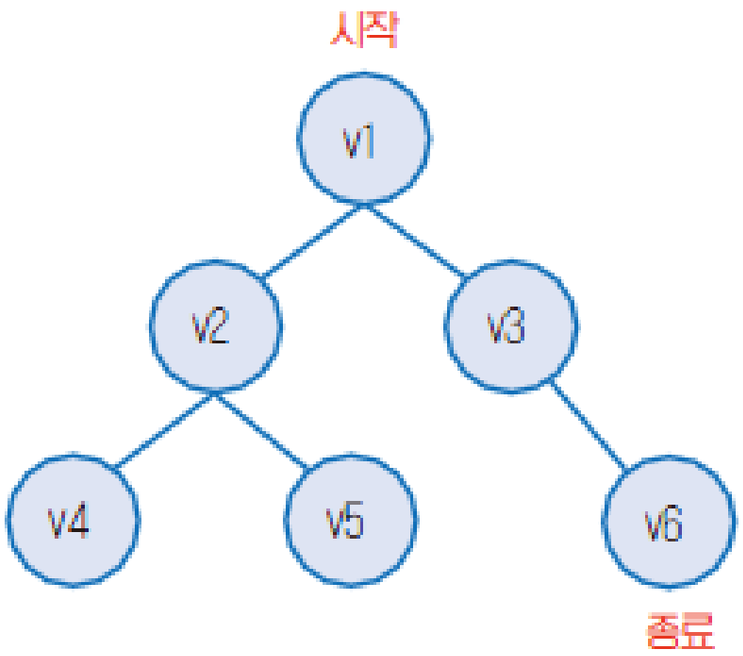
기본적인 탐색 기법



- 맹목적인 탐색 : 목표 노드에 대한 정보를 이용하지 않고 기계적인 순서로 노드를 확장하는 방법
- 경험적(heuristic) 탐색 : 목표 노드에 대한 경험적 정보를 이용

너비우선 탐색 (1/5)

- BFS(Breadth-First Search)
  - 시작 지점을 기준으로 가장 근접한 지점(직접 연결)들을 검색하고 그 다음으로 가까운 지점을 탐색하는 방식



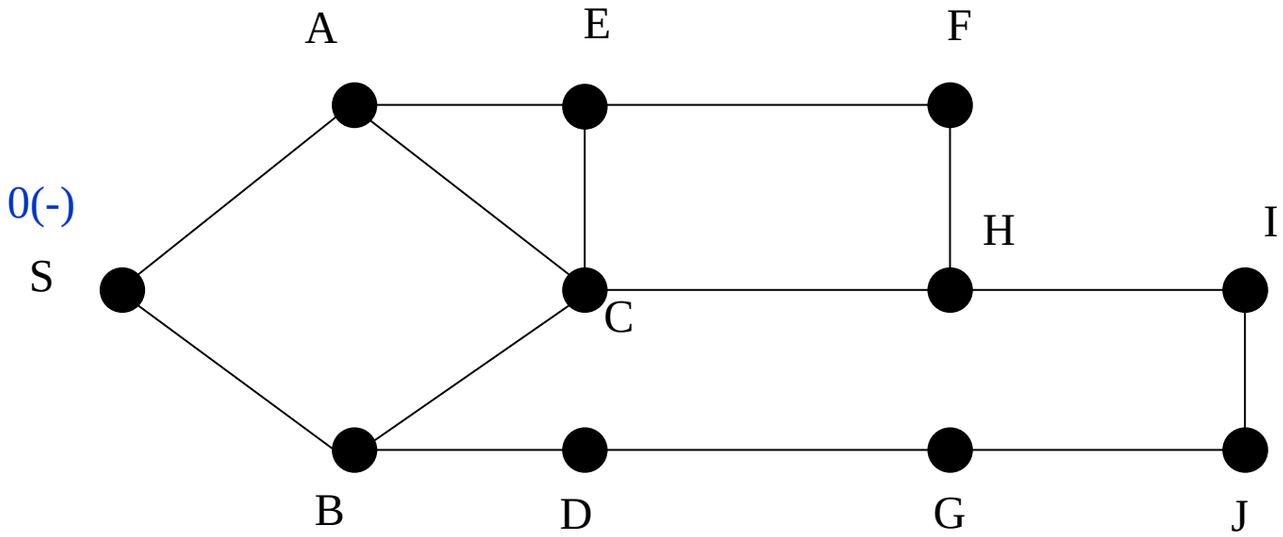
v1 → v2 → v3 → v4→ v5 → v6

너비우선 탐색 (2/5)

- 한 정점(v)을 먼저 탐색하고, v에 인접한 모든 정점들을 차례로 탐색
- 더 이상 방문할 정점이 없는 경우, v에 인접한 정점 가운데 맨 처음으로 방문한 정점과 인접한 정점들을 차례로 방문, v에 인접한 정점 중 두 번째로 방문한 정점과 인접한 정점들을 차례로 방문하는 과정을 반복
- 모든 정점들을 방문한 후 탐색을 종료

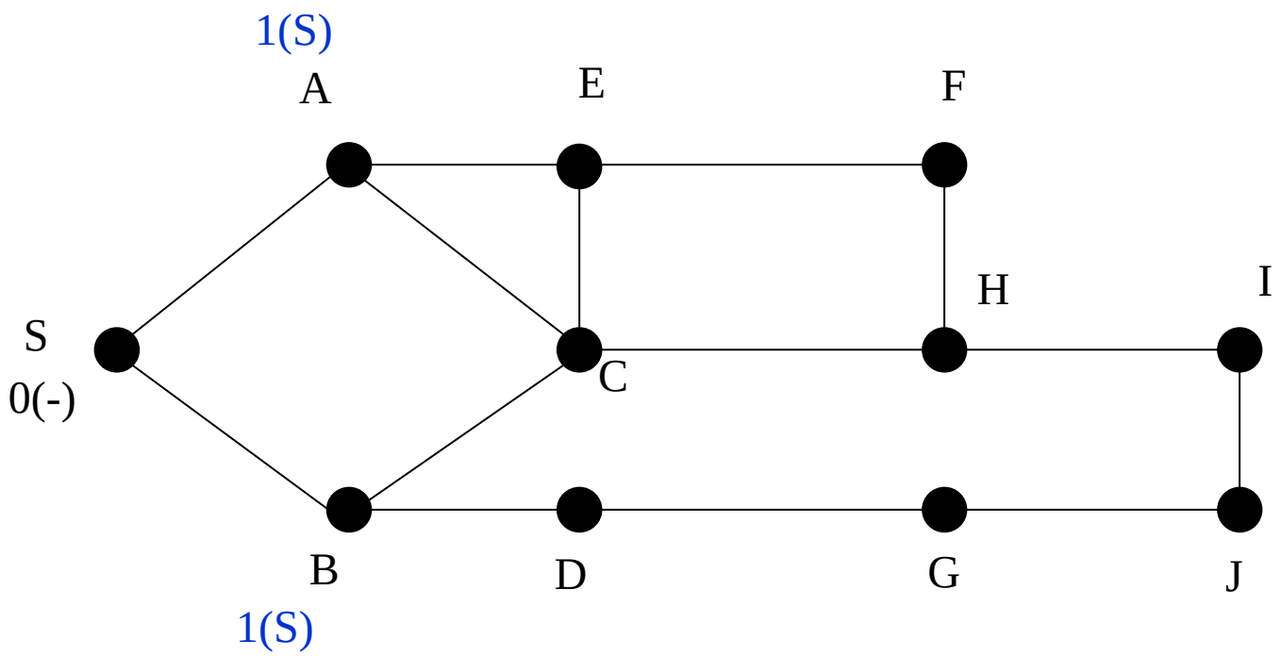
너비우선 탐색 (3/5)

- L은 레이블이 붙은 정점의 집합
- 레이블 k=0부터 시작

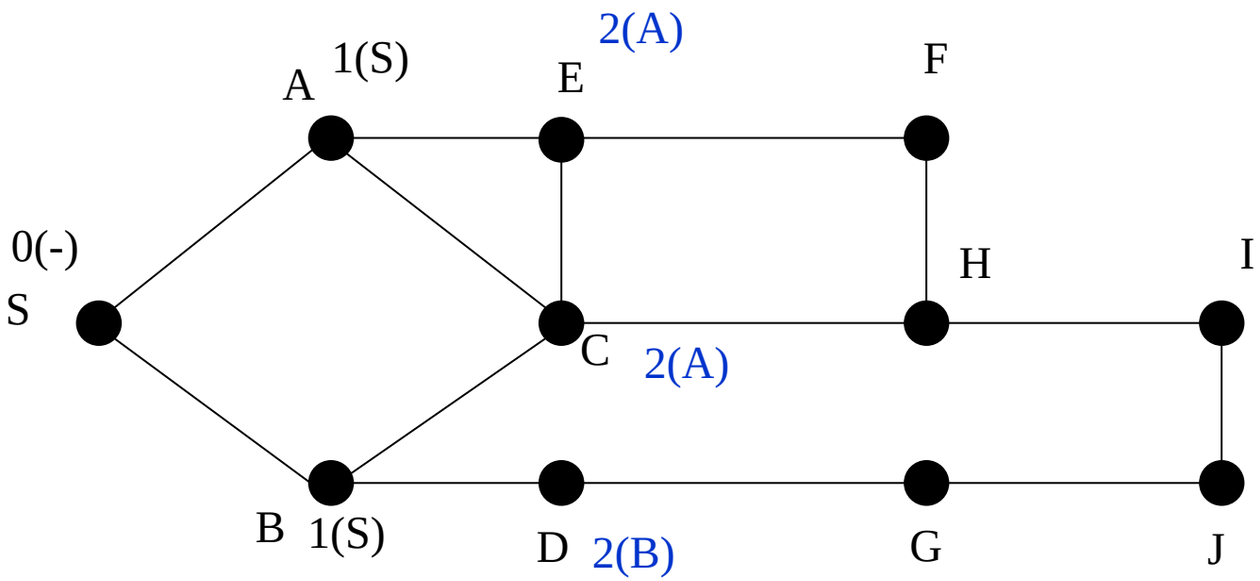


$L = \{S\}$ and $k = 0$

너비우선 탐색 (4/5)



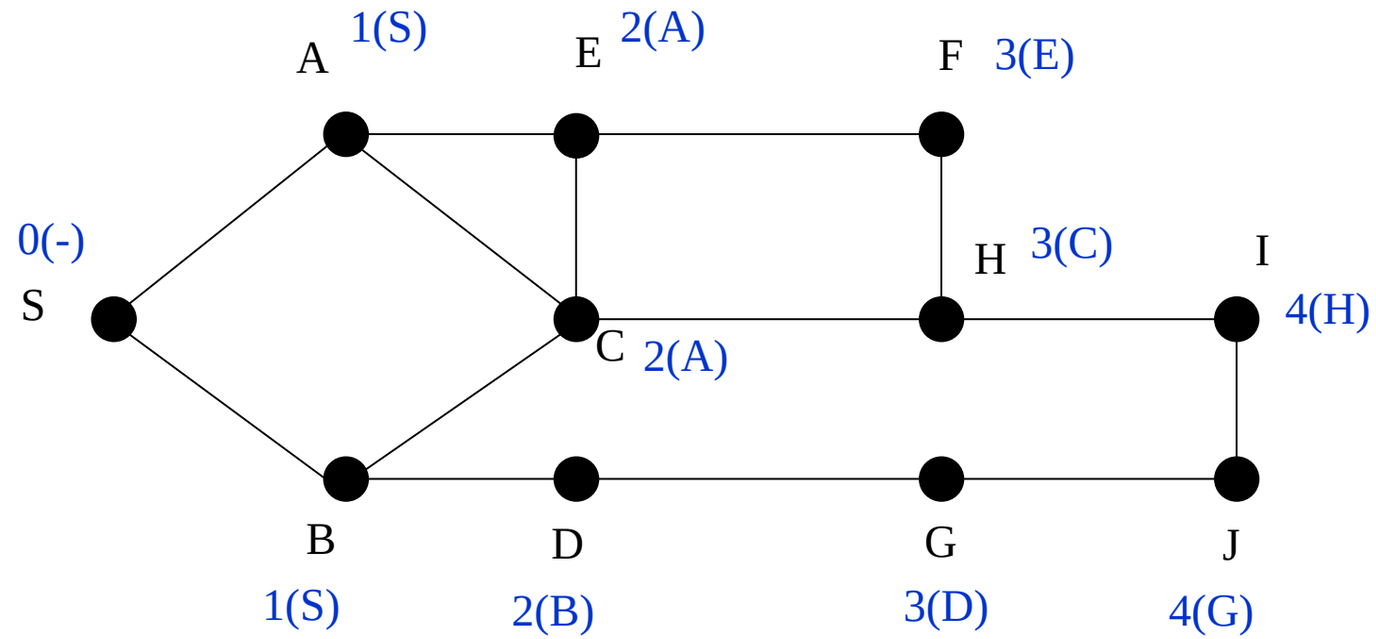
$k = 1$ and $L = \{S, A, B\}$



$k = 2$ and $L = \{S, A, B, E, C, D\}$

너비우선 탐색 (5/5)

- S로부터 각 점으로 가는 경로의 거리

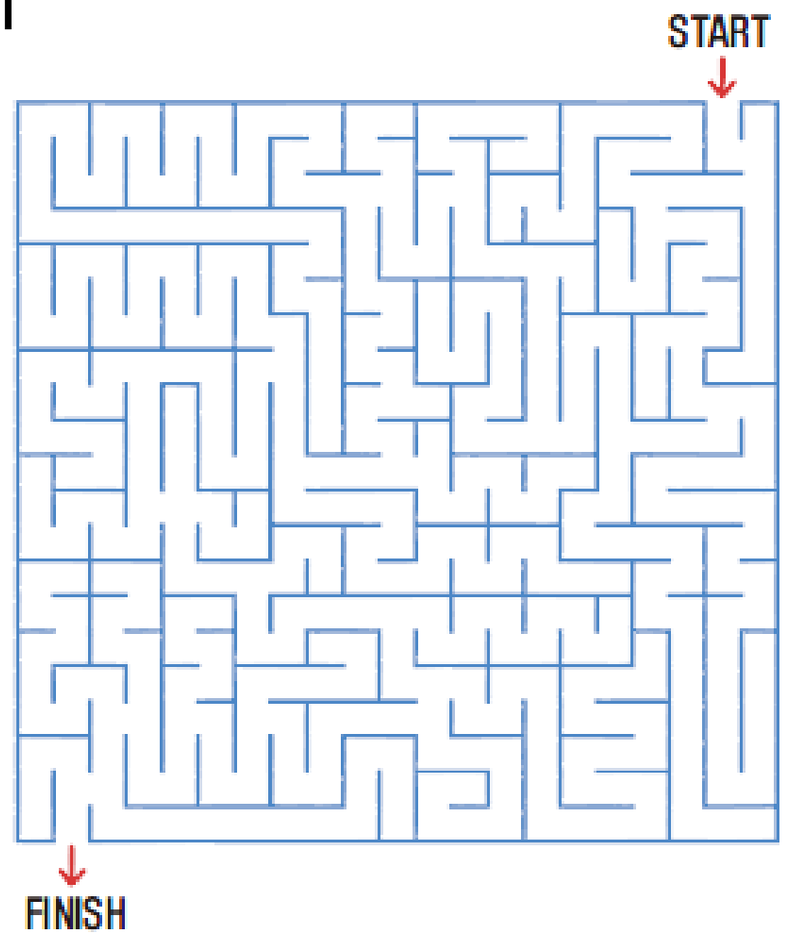


최단 경로는 I부터 선행자를 따라 S ← A ← C ← H ← I

- *S*에서 *I*로의 최단거리 : *S, A, C, H, I*

깊이우선 탐색 (1/9)

- DFS(Depth First Search)
- 갔다가 되돌아 오기

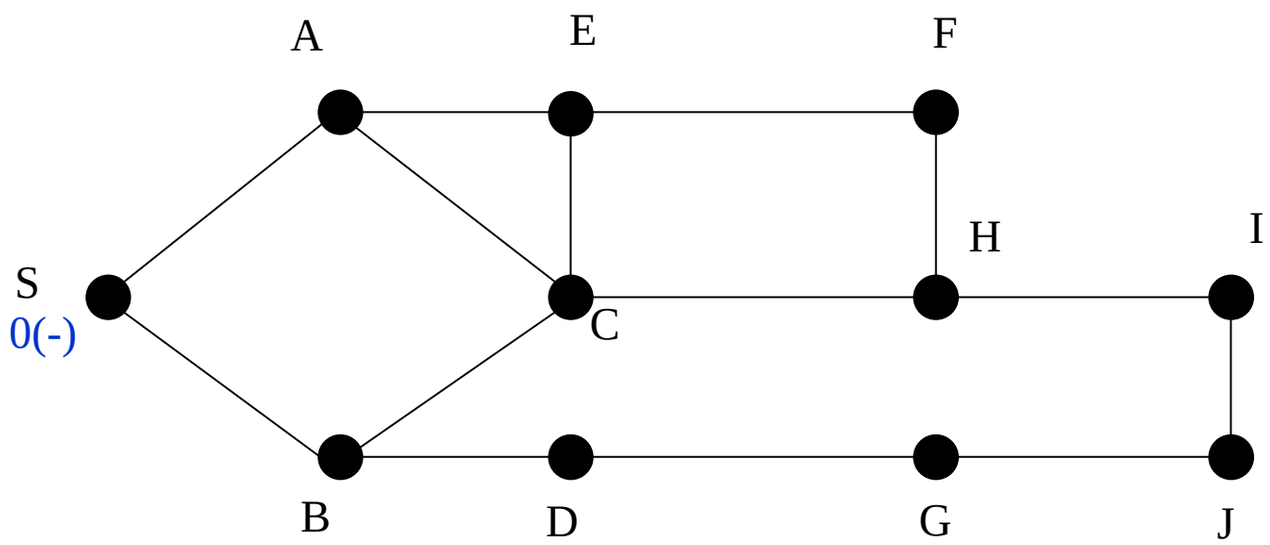


깊이우선 탐색 (2/9)

- 시작점 v부터 방문함
- v에 인접한 정점 중에서 방문하지 않은 정점 w를 방문하고 다시 w로부터 탐색을 시작함
- 어떤 정점 u를 방문하고 u에 인접한 모든 정점들을 이미 방문한 경우에는 그 전에 마지막으로 방문한 정점으로 되돌아가서 위의 과정들을 반복함
- 모든 정점들을 방문한 후 탐색을 종료함

깊이우선 탐색 (3/9)

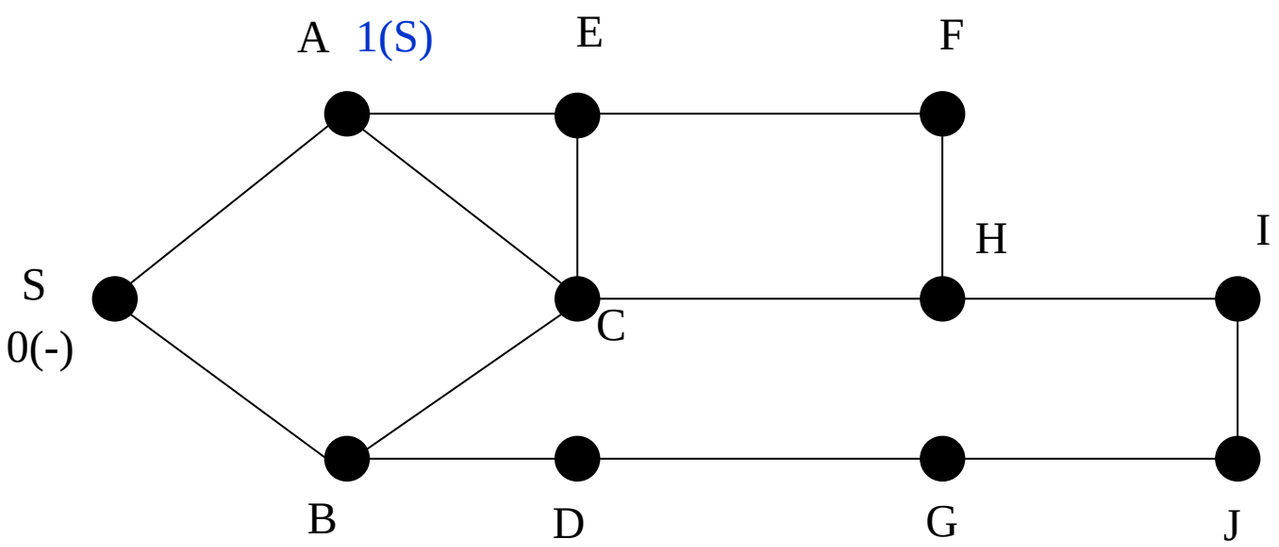
- L은 레이블이 붙은 정점의 집합
- 레이블 k=0부터 시작해서 일련의 정수들

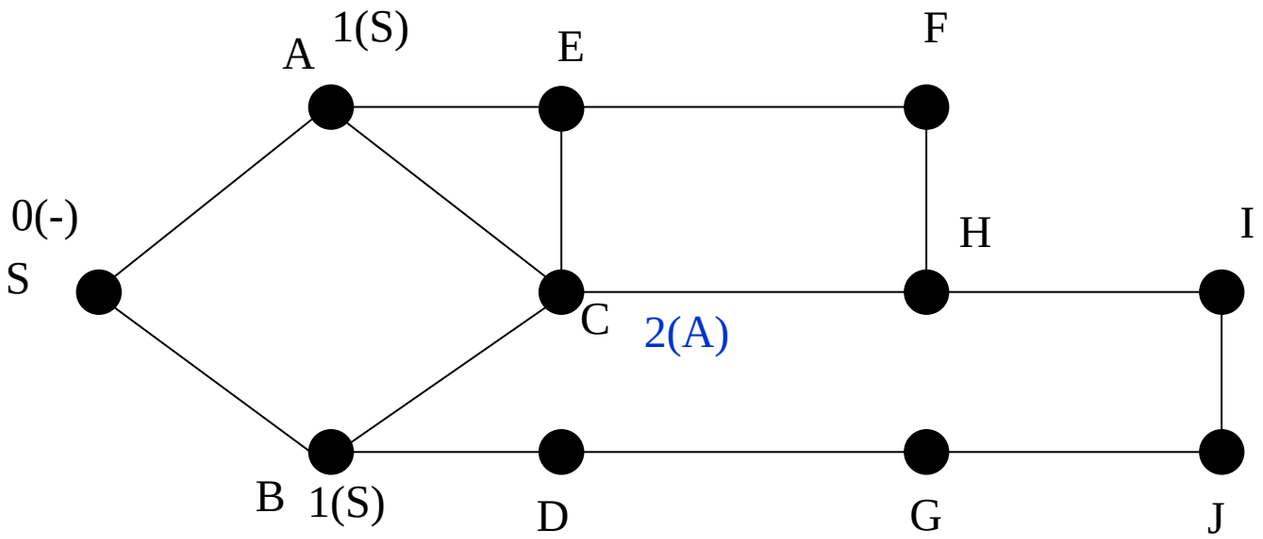


$L = \{S\}$ and $k = 0$

깊이우선 탐색 (4/9)

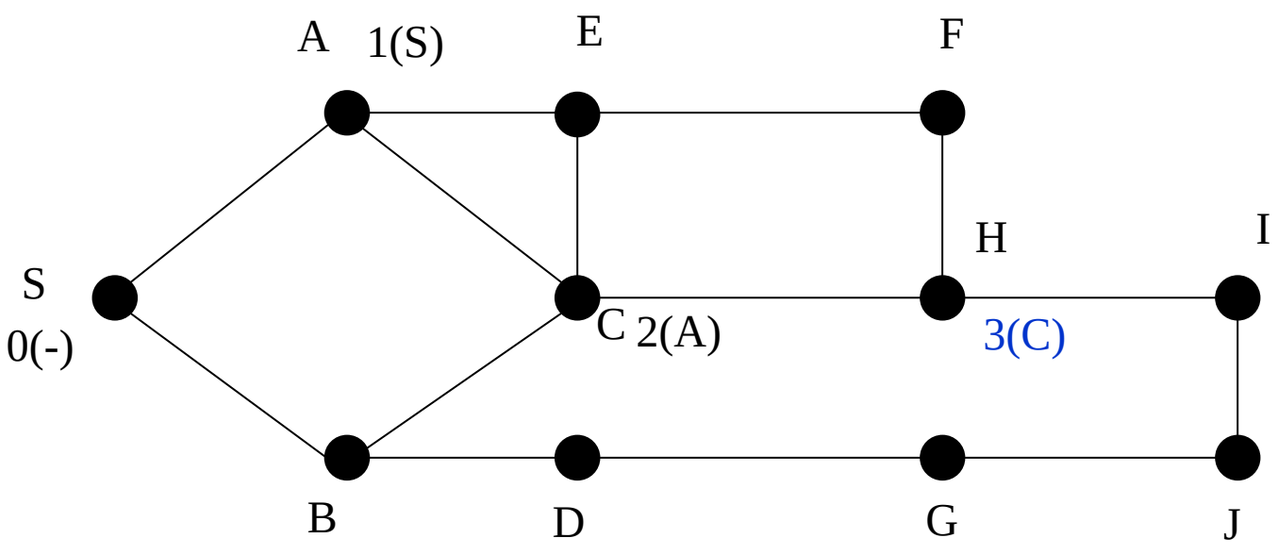
$k = 1$ and $L = \{S, A\}$



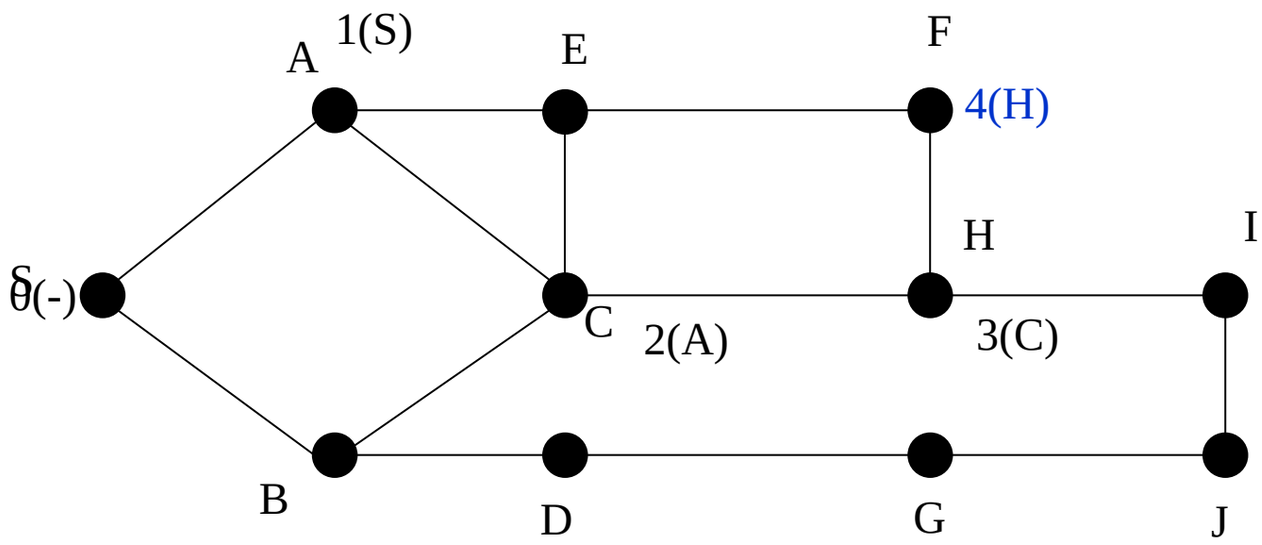


$k = 2$ and $L = \{S, A, C\}$

깊이우선 탐색 (5/9)

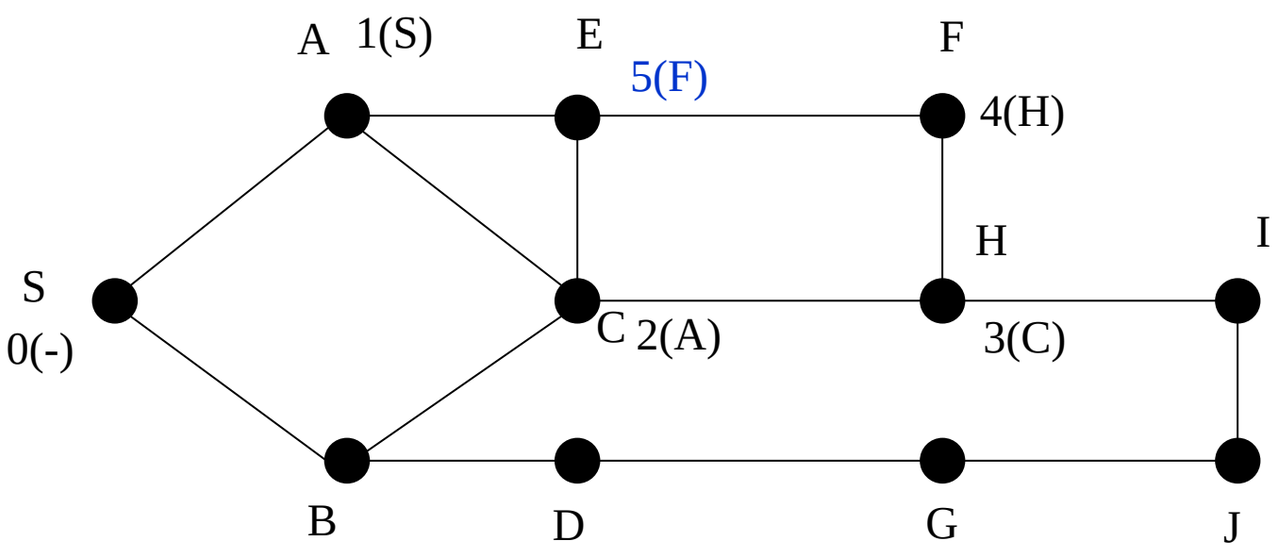


$k = 3$ and $L = \{S, A, C, H\}$

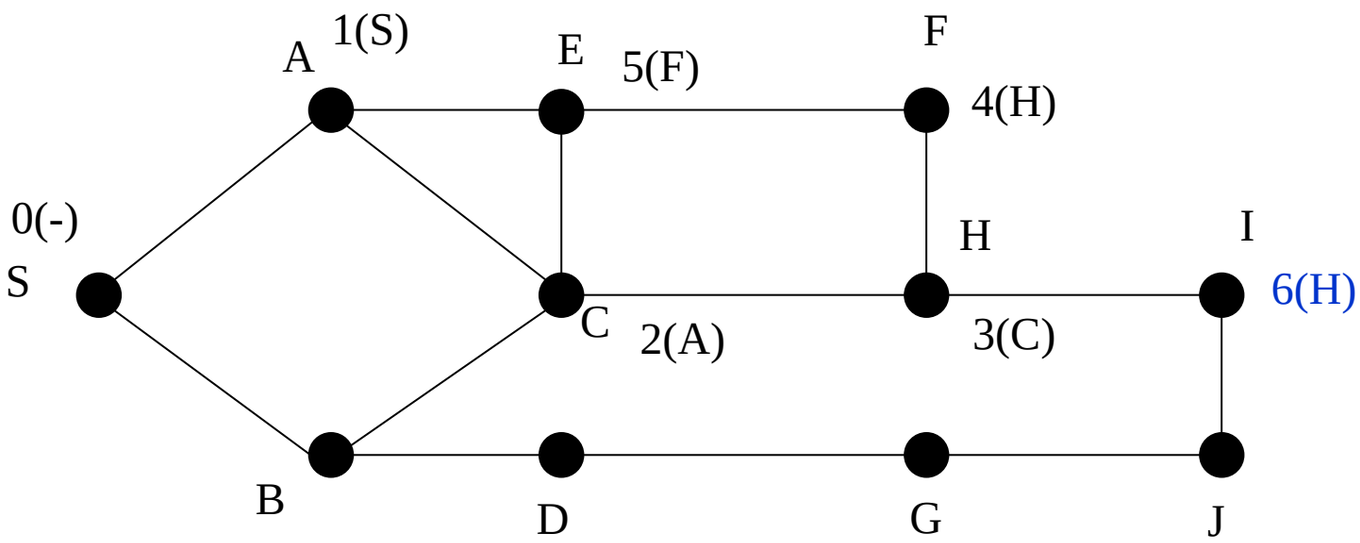


$k = 4$ and $L = \{S, A, C, H, F\}$

깊이우선 탐색 (6/9)

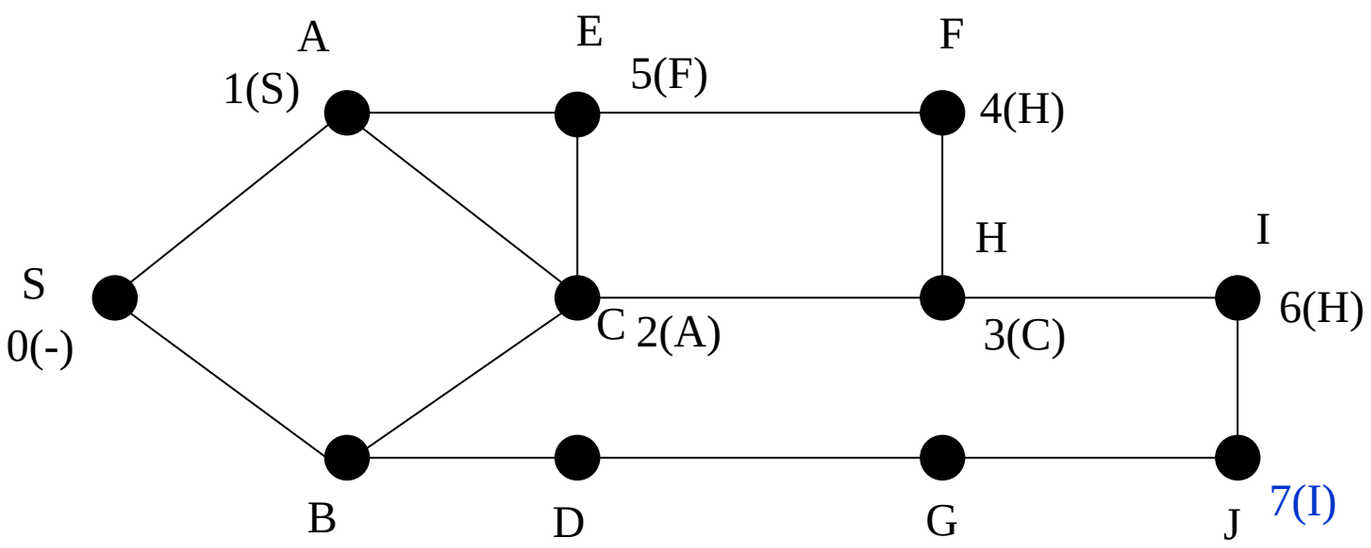


$k = 5$ and
$L = \{S, A, C, H, F, E\}$

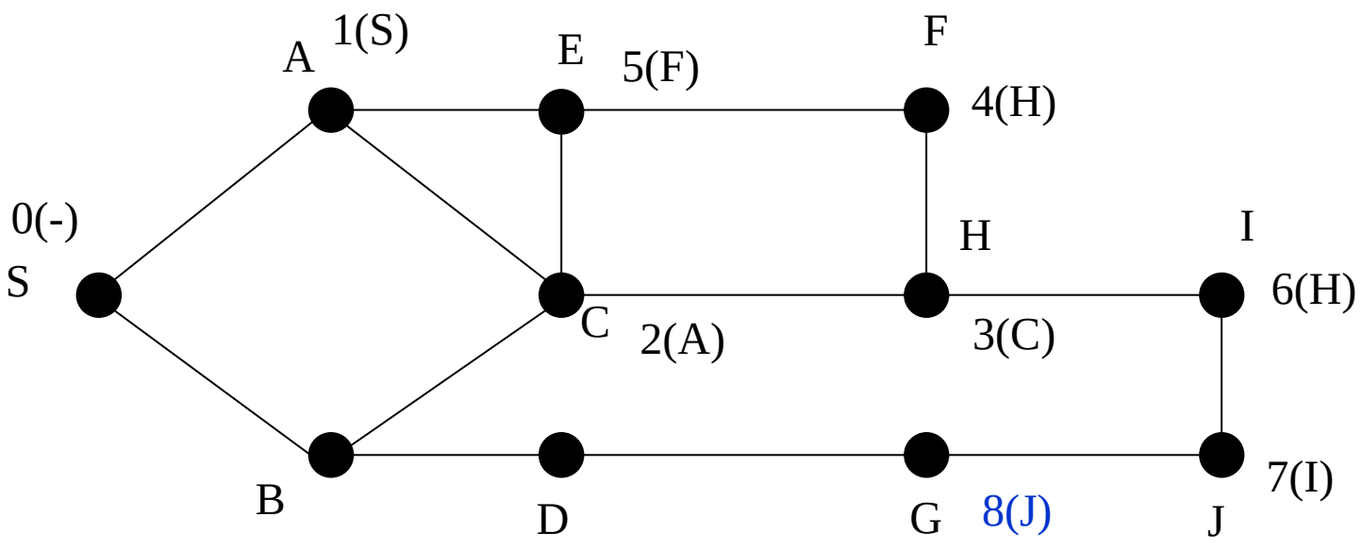


$k = 6$ and
$L = \{S, A, C, H, F, E, I\}$

깊이우선 탐색 (7/9)

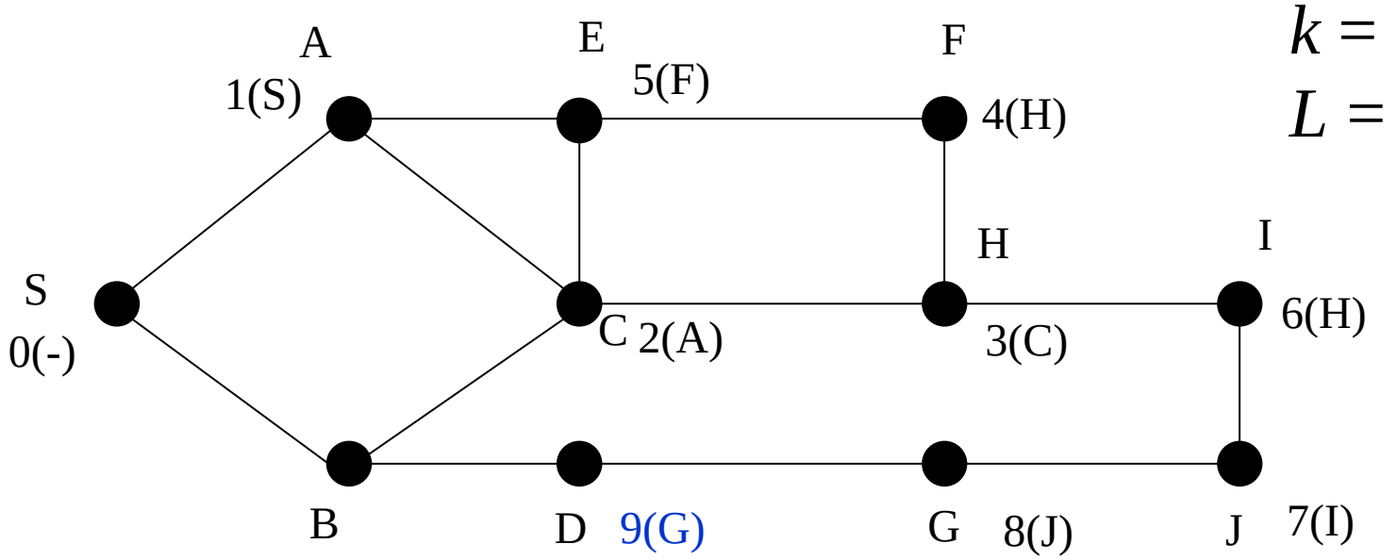


$k = 7$ and
$L = \{S, A, C, H, F, E, I, J\}$

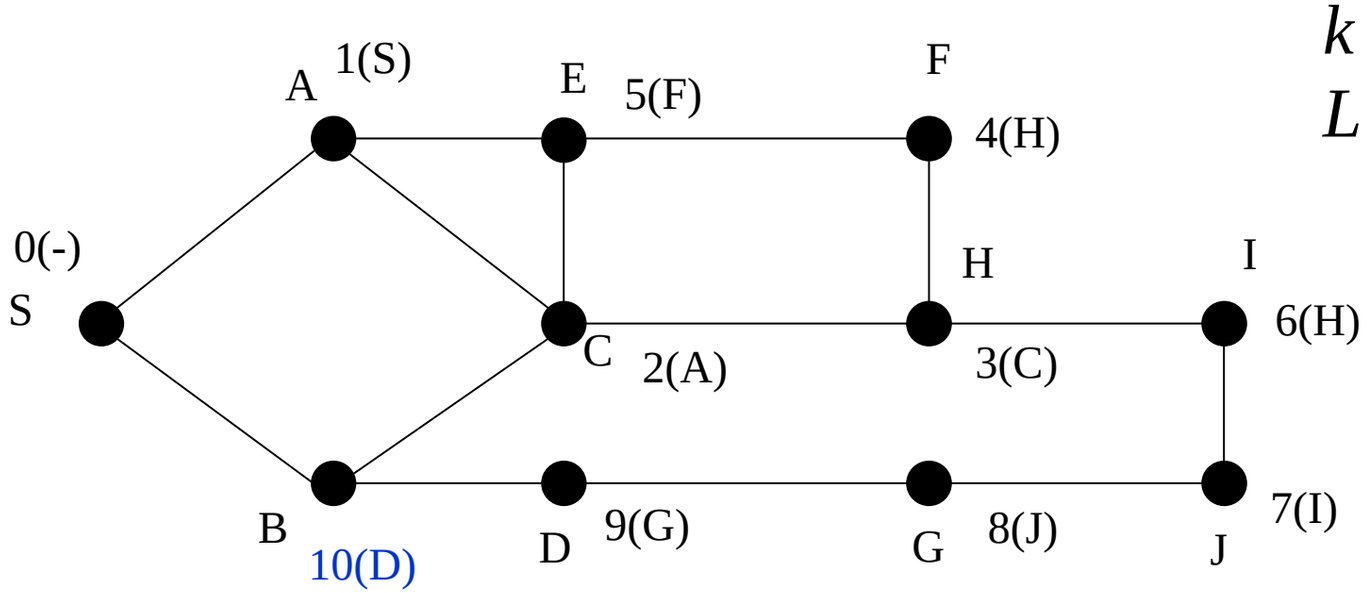


$k = 8$ and
$L = \{S, A, C, H, F, E, I, J, G\}$

깊이우선 탐색 (8/9)



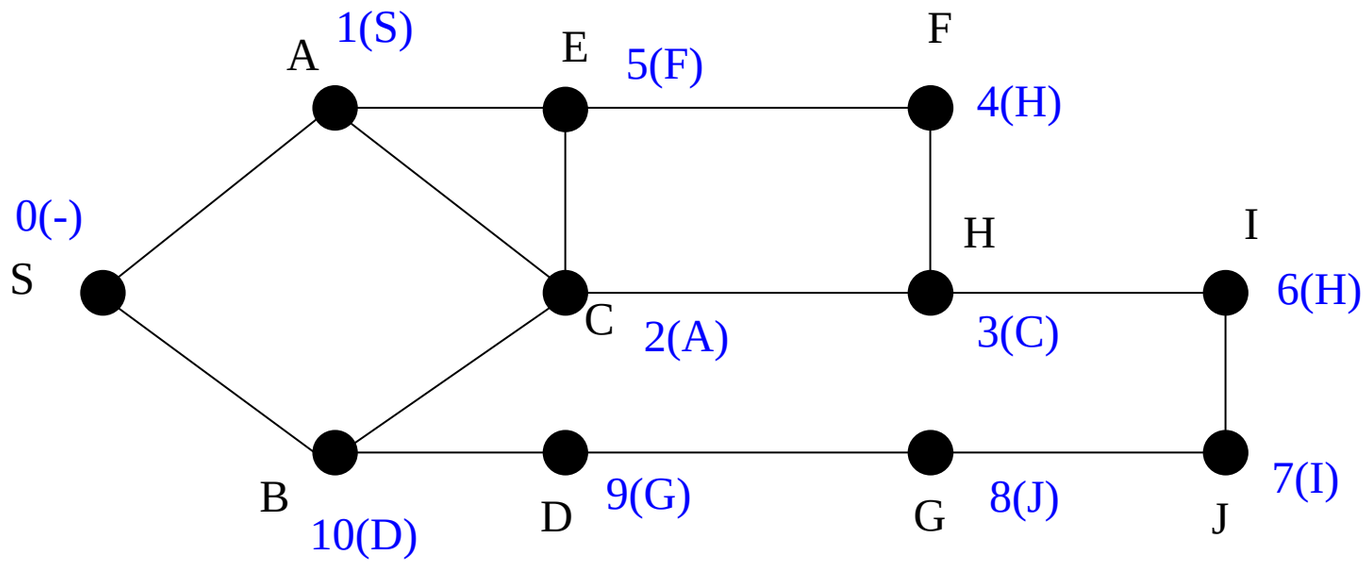
$k = 9$ and
$L = \{S, A, C, H, F, E, I, J, G, D\}$



$k = 10$ and
$L = \{S, A, C, H, F, E, I, J, G, D, B\}$

깊이우선 탐색 (9/9)

- S로부터 각 점으로 가는 경로의 거리

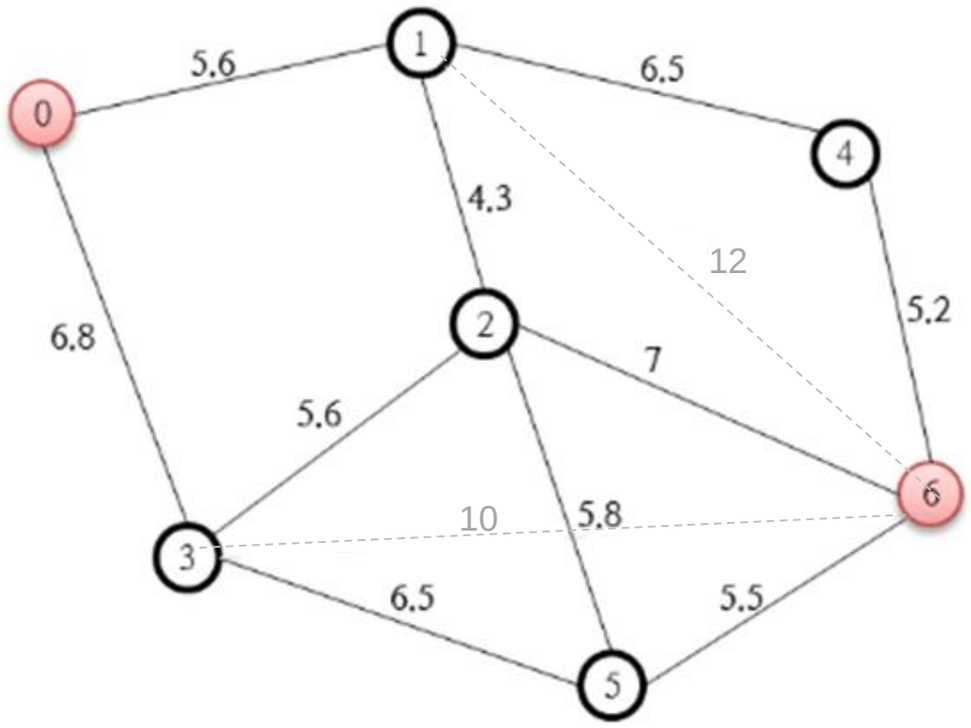


A* 탐색 알고리즘 (1/7)

- 주어진 출발점에서부터 목표점까지 가는 최단 경로를 찾아내는 그래프 탐색 알고리즘
- 휴리스틱 추정 값을 통해 알고리즘을 개선
  - 휴리스틱 추정 값의 제공 방식에 따라 얼마나 빨리 최단 경로를 파악할 수 있는지가 결정
  - F = G + H
    - G는 시작점에서 해당 점까지의 실제 소요 경비 값
    - H는 휴리스틱 추정값: 해당 점에서 최종 목적지까지 도달하는데 소요될 것으로 추정되는 값
  - Parent node : 해당 점에 도달하기 직전에 거치는 노드

A* 탐색 알고리즘 (2/7)

- O (Open List, 열린 목록) : O저장소에는 최단 경로를 분석하기 위한 상태 값들이 계속 갱신 됨
- C (Close List, 닫힌 목록) : C 저장소는 처리가 완료된 노드를 넣어 두기 위한 목적으로 사용
- 0번 노드에서 6번 노드까지의 최단 경로를 산출해 보기



$O =$

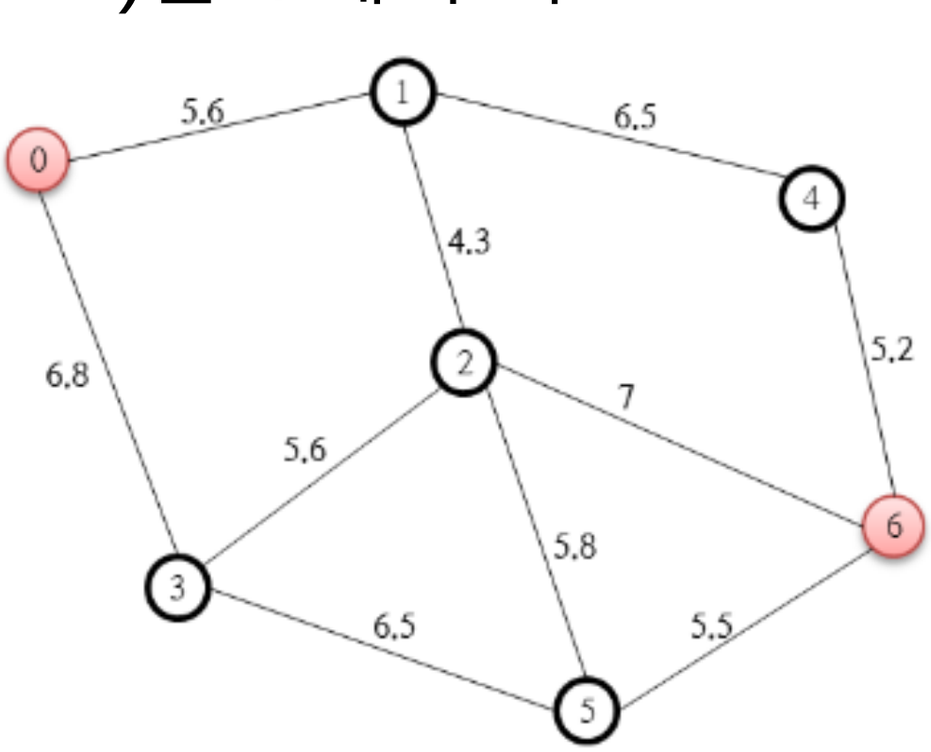
Node ID	1	3
F Score	17.6	16.8
G Score	5.6	6.8
H Score	12	10
Parent Node	0	0

$C =$

Node ID	0
F Score	0
G Score	0
H Score	0
Parent Node	-

A* 탐색 알고리즘 (3/7)

- O에서 F 값이 가장 작은 노드(3번노드)를 선택해서 C에 추가
- 선택된 노드와 연결된 노드 중 C에 존재하지 않는 노드 (2번, 5번 노드)를 O에 추가



O =

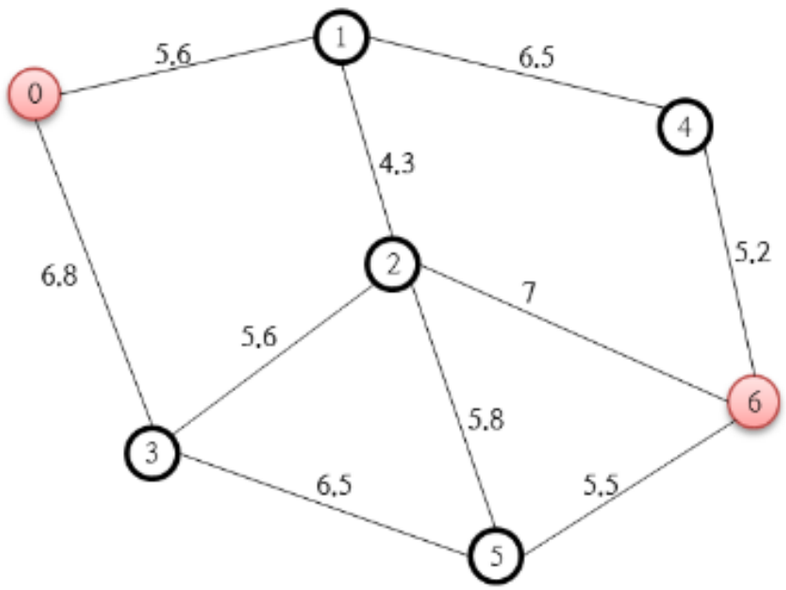
Node ID	1	2	5
F Score	17.6	19.4	18.8
G Score	5.6	12.4	13.3
H Score	12	7	5.5
Parent Node	0	3	3

C =

Node ID	0	3
F Score	0	16.8
G Score	0	6.8
H Score	0	10
Parent Node	-	0

A* 탐색 알고리즘 (4/7)

- O에서 F 값이 가장 작은 노드(1번노드)를 선택해서 C에 추가
- 선택된 노드와 연결된 노드 중 C에 존재하지 않는 노드(4번노드)를 O에 추가
- 이미 O에 있던 연결된 노드(2번노드)는 새롭게 계산된 값과 비교해서 변경



원래 가지고 있던 값

2
19.4
12.4
7
3

O =

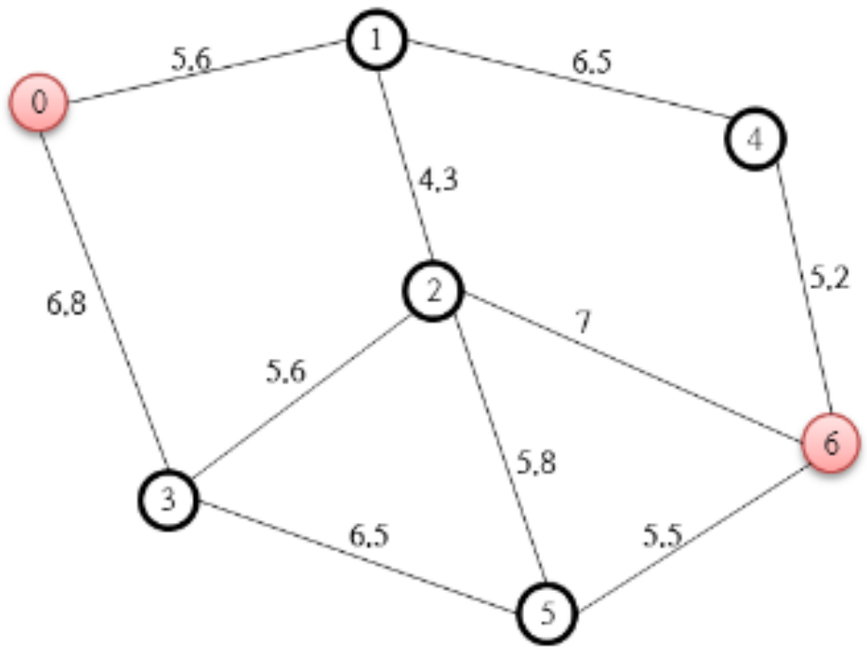
Node ID	2	5	4
F Score	16.9	18.8	17.3
G Score	9.9	13.3	12.1
H Score	7	5.5	5.2
Parent Node	1	3	1

C =

Node ID	0	3	1
F Score	0	16.8	17.6
G Score	0	6.8	5.6
H Score	0	10	12
Parent Node	-	0	0

A* 탐색 알고리즘 (5/7)

- O에서 F 값이 가장 작은 노드(2번노드)를 선택해서 C에 추가
- 선택된 노드와 연결된 노드 중 C에 존재하지 않는 노드(6번노드)를 O에 추가
- 이미 O에 있던 연결된 노드(5번노드)는 새롭게 계산된 값과 비교해서 변경(또는 유지)



O =

Node ID	5	4	6
F Score	18.8	17.3	16.9
G Score	13.3	12.1	16.9
H Score	5.5	5.2	0
Parent Node	3	1	2

C =

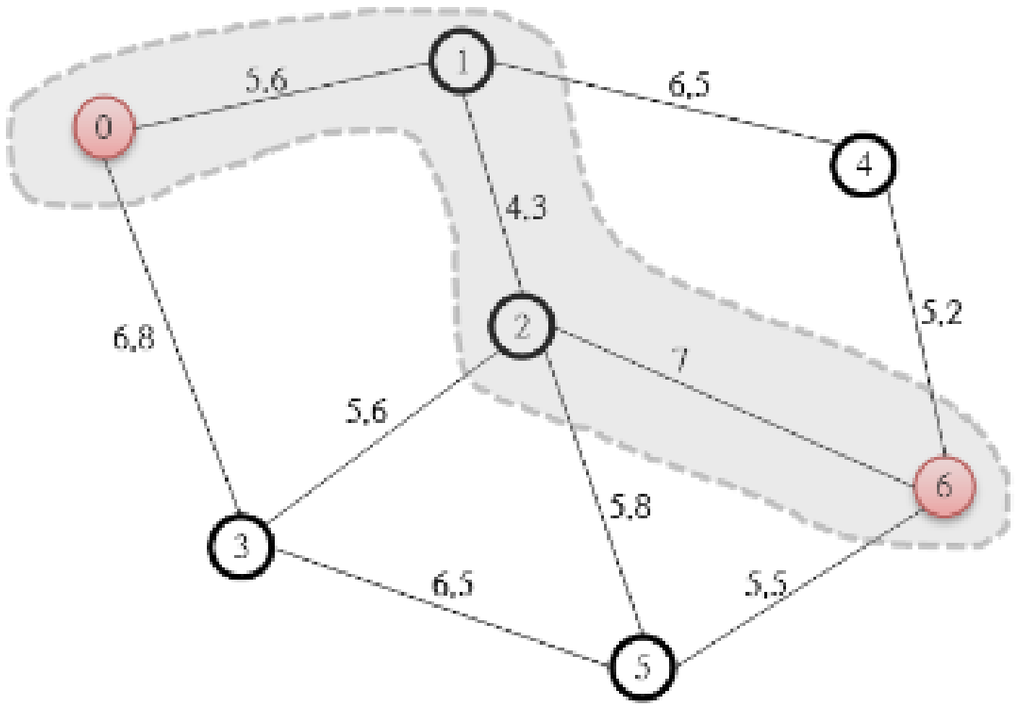
Node ID	0	3	1	2
F Score	0	16.8	17.6	16.9
G Score	0	6.8	5.6	9.9
H Score	0	10	12	7
Parent Node	-	0	0	1

A* 탐색 알고리즘 (6/7)

- O에서 F 값이 가장 작은 노드(6번노드)를 선택해서 C에 추가
- 최종 목표점이 C에 추가되면, A* 알고리즘은 종료
- 지금까지 만들어진 C를 토대로, 출발점에서 목표점까지 최단 경로를 파악할 수 있음
  - parent node를 따라가면, 6번 노드의 Parent Node는 2번 이고, 2번 노드의 Parent Node는 1번…이므로
  - 최단 경로는 6번 노드←2번 노드←1번 노드←0번 노드

➢ 0 – 1 – 2 – 6

A* 탐색 알고리즘 (7/7)



$O =$

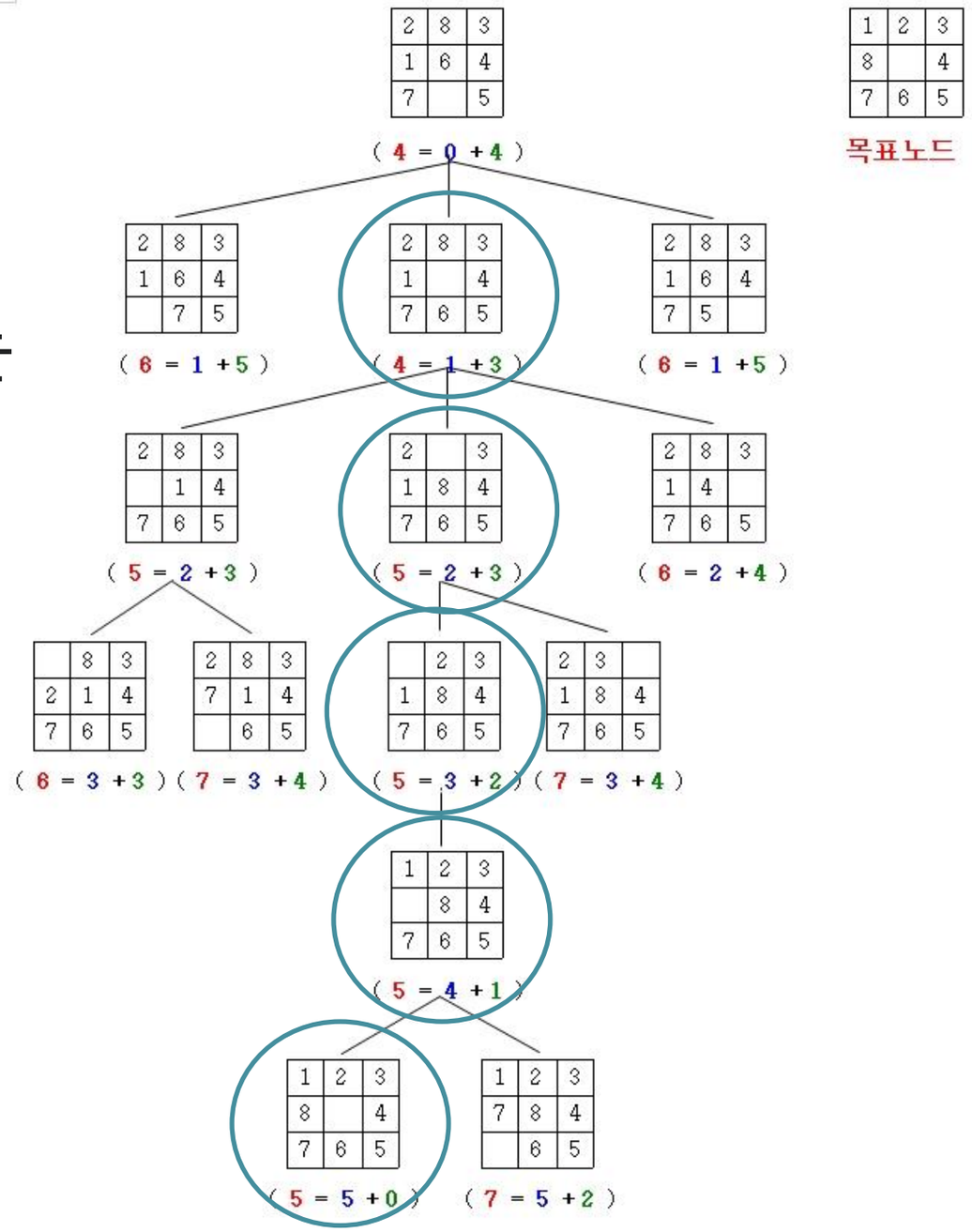
Node ID	5	4
F Score	18.8	17.3
G Score	13.3	12.1
H Score	5.5	5.2
Parent Node	3	1

$C =$

Node ID	0	3	1	2	6
F Score	0	16.8	17.6	16.9	16.9
G Score	0	6.8	5.6	9.9	16.9
H Score	0	10	12	7	0
Parent Node	-	0	0	1	2

A*알고리즘으로 풀기: 8-Queen 문제

- 3 x 3의 숫자판에 1~8까지의 숫자와 빈칸이 주어져 있음
  - 숫자를 인접한 빈칸으로 옮기는 작업을 반복
- 목표가 되는 숫자 배열로 만들기(총 이동 횟수 최소화)
- F = G + H
  - G : 지금까지 움직인 횟수
  - H : 제자리에 있지 않은 퍼즐 수



Contents

I. 탐색 및 문제해결

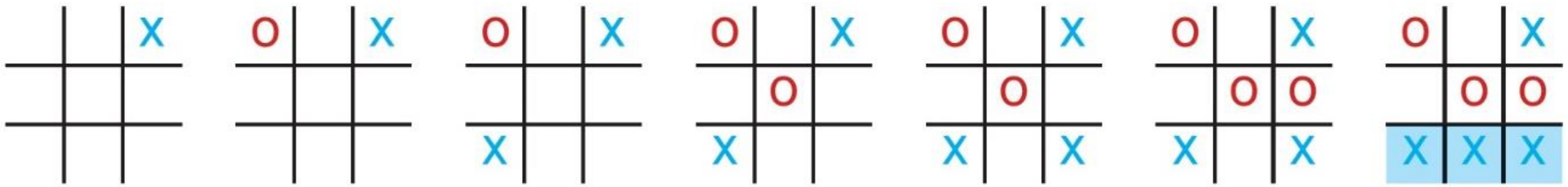
II. AI 탐색 기법

III. 탐색과 게임

Tic-Tac-Toe 게임 (1/2)

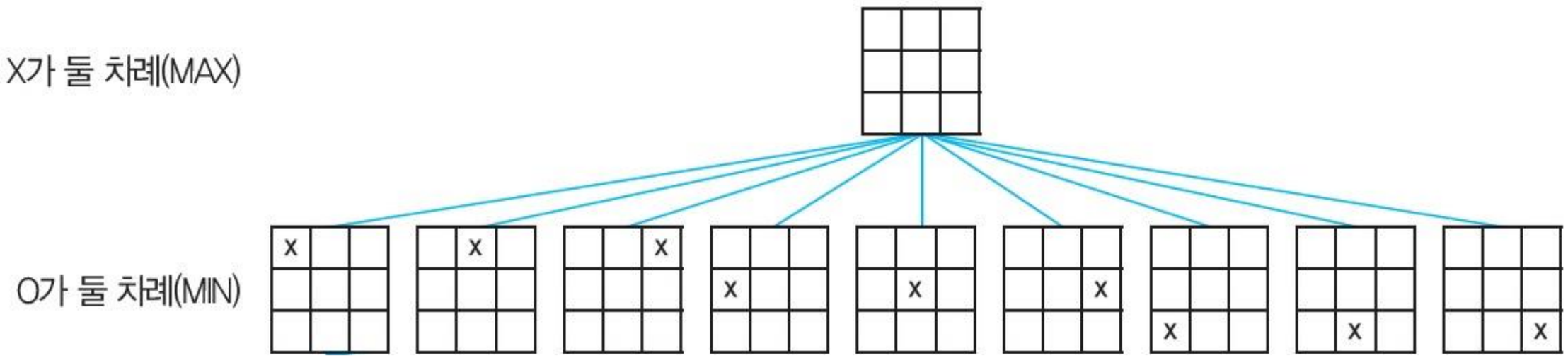
Tic-Tac-Toe 게임 (2/2)

- 최저(Min)와 최고(Max)가 번갈아 가면서 게임을 함
- Max는 x, Min은 o로 정하고, 항상 Max가 먼저 수를 둔다고 가정



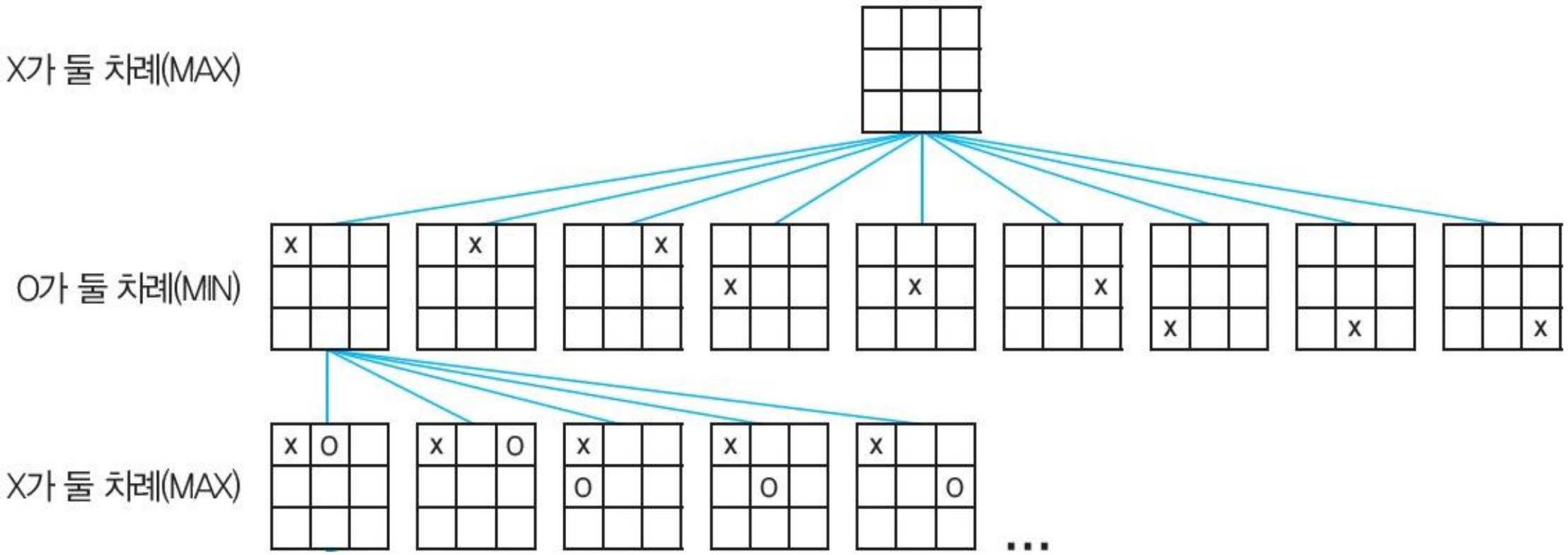
게임 트리 (1/5)

- 상태 = 노드(node)
- 초기 상태 = 루트 노드
- 연산자 = 간선(edge)

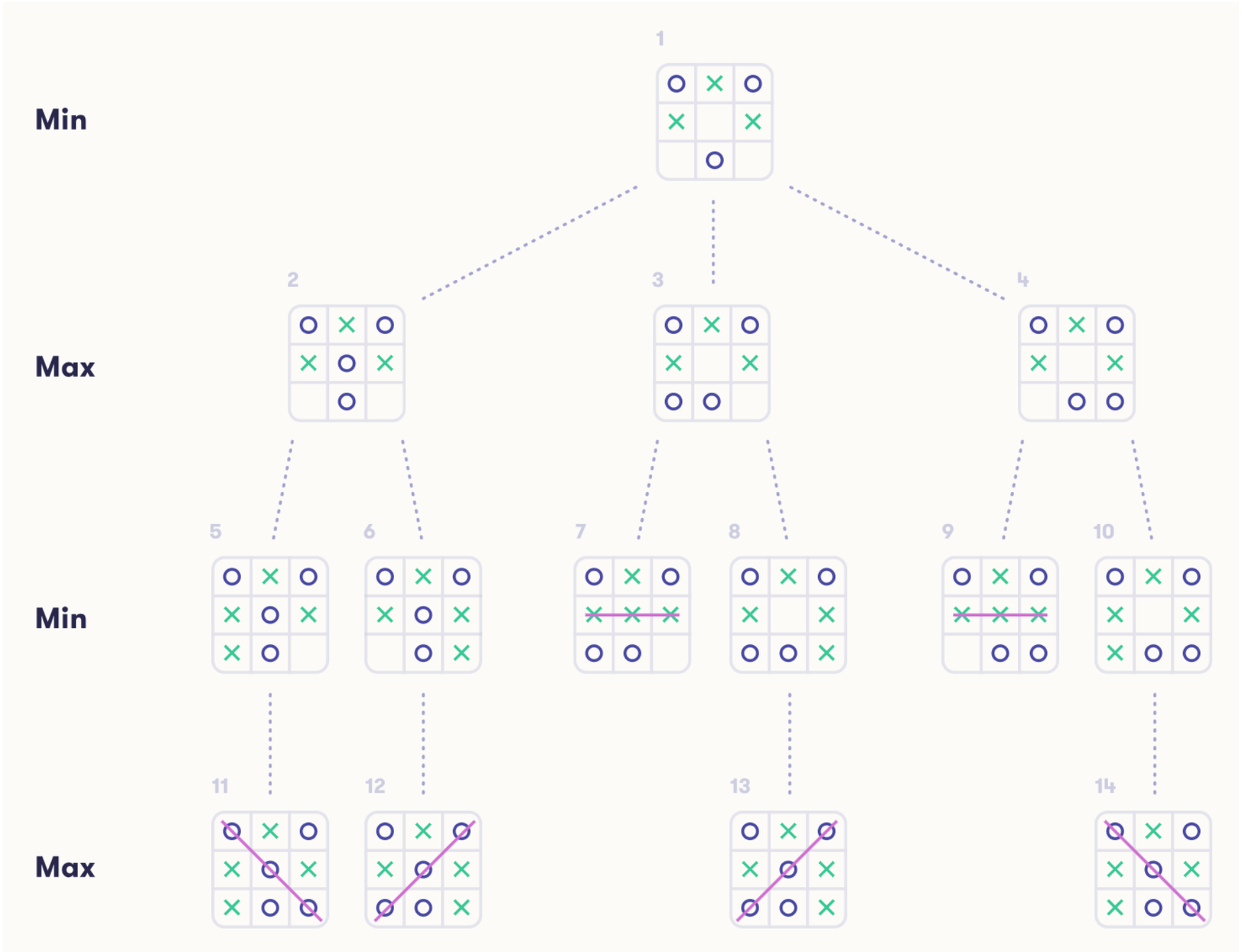


게임 트리 (2/5)

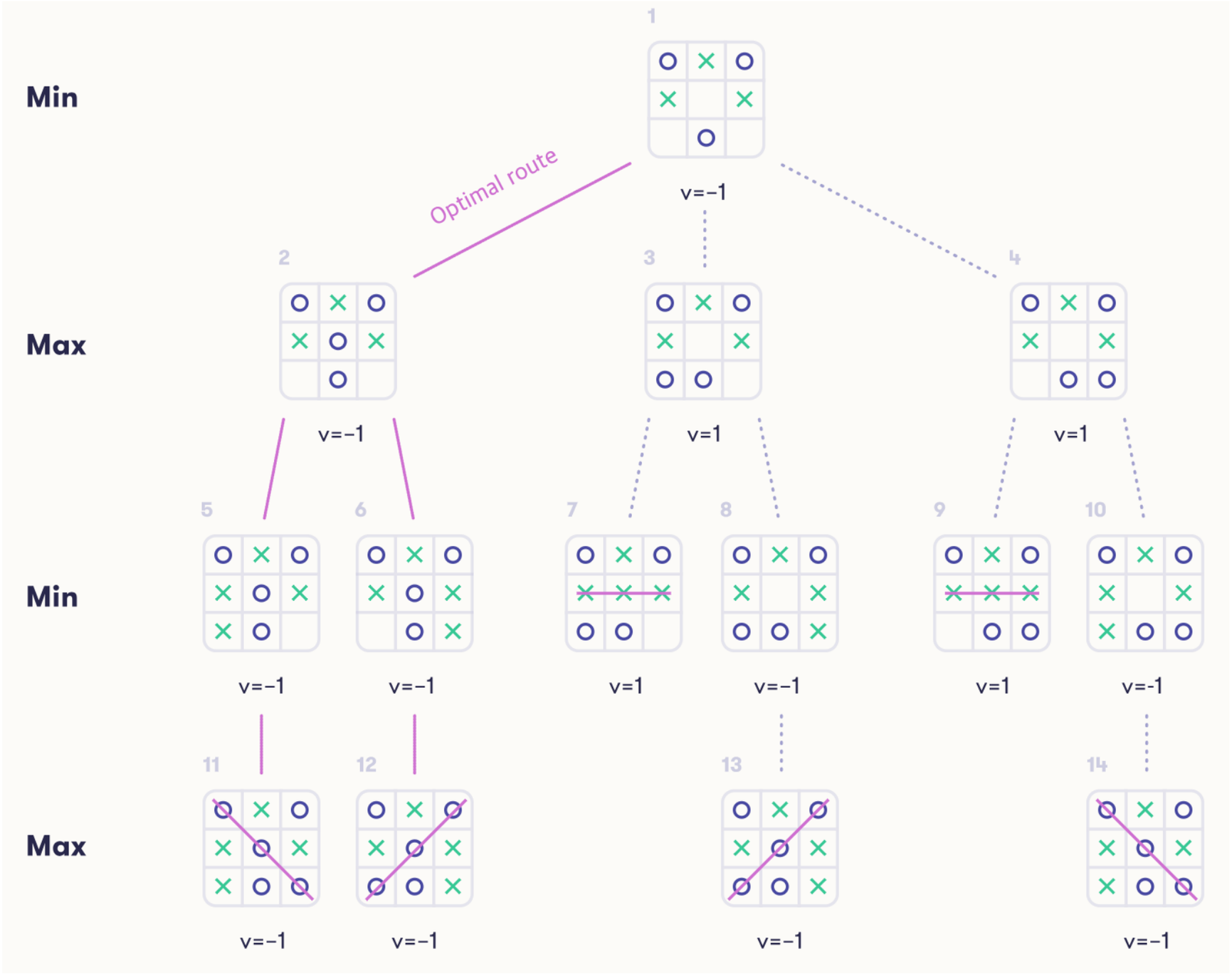
- 연산자를 적용하기 전까지는 게임 트리는 미리 만들어져 있지 않음!



게임 트리 (3/5)



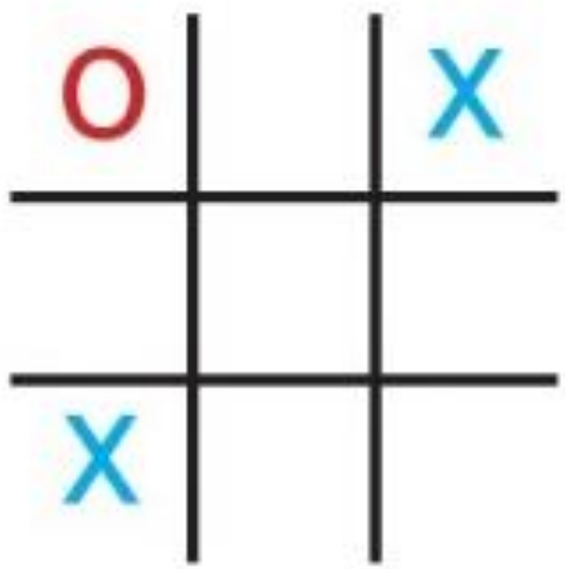
게임 트리 (4/5)


Min
Max
Min
Max
Optimal route
1
v=−1
2
3
4
v=−1
v=1
v=1
5
6
7
8
9
10
v=−1
v=−1
v=1
v=−1
v=1
v=-1
11
12
13
14
v=−1
v=−1
v=−1
v=−1

게임 트리 (5/5)

- Tic-Tac-Toe의 게임 트리는 크기가 얼마나 될까?

$$9 \times 8 \times 7 \times \ldots \times 1 = 9! = 362{,}880$$

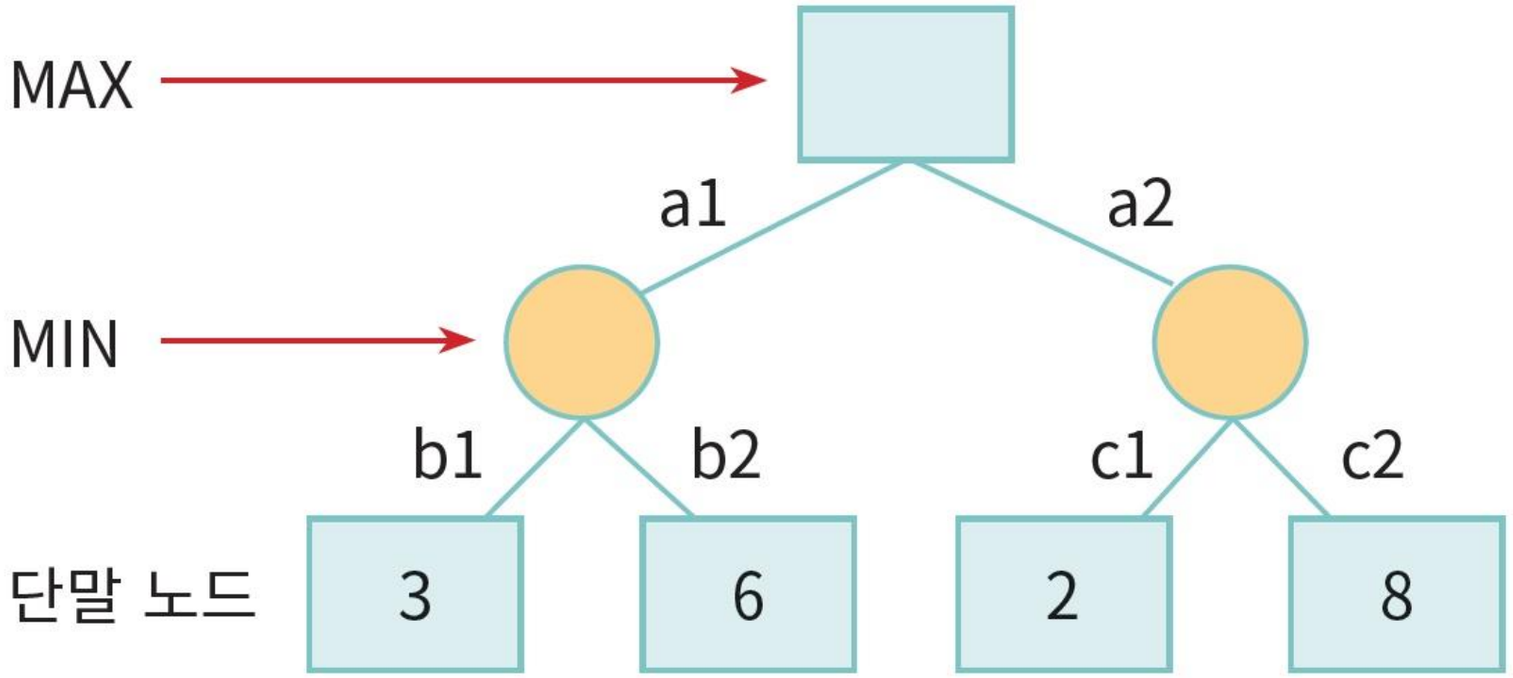


Minimax 알고리즘 (1/6)

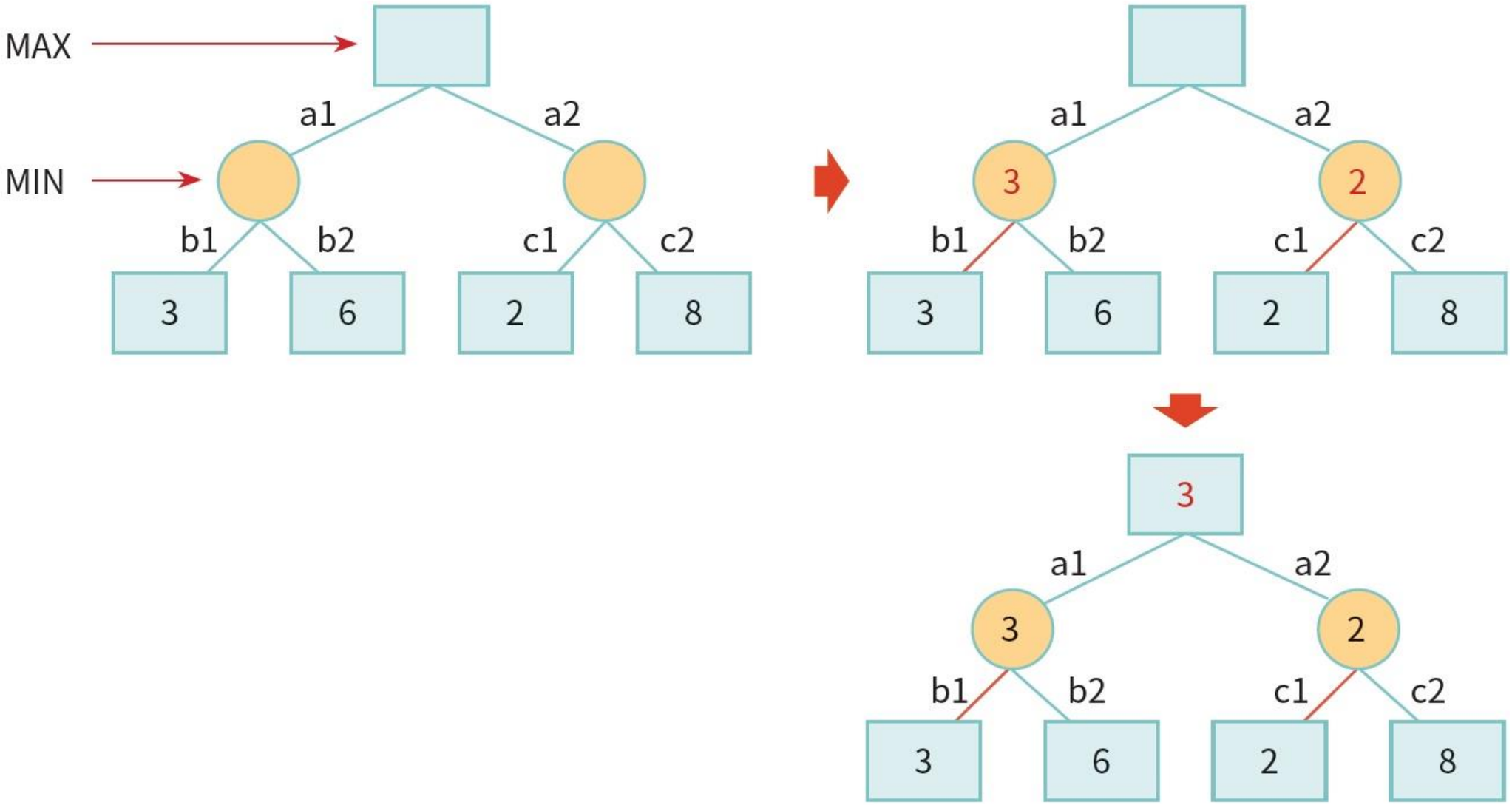
- 게임트리에서의 목표 상태는 승리에 해당되는 단말 상태
- 상대방이 탐색에 영향을 주게 됨
- 미니맥스 알고리즘
  - 나에게 좋은 수는 상대방에게 불리한 수가 되기 때문에, 내가 이길 가능성(큰 수)과 상대방이 질 수 있는 가능성(작은 수)를 교대로 선택하게 하는 방법
  - 최종 게임이 끝나고 일정 깊이까지의 값을 계산하여 가장 최적의 경로를 탐색하는 기법

Minimax 알고리즘 (2/6)

- 안전하게 하려면 상대방이 최선의 수를 둔다고 생각
  - 단말노드에 적힌 숫자는 평가 값으로 높으면 Max가 유리, 낮으면 Min이 유리

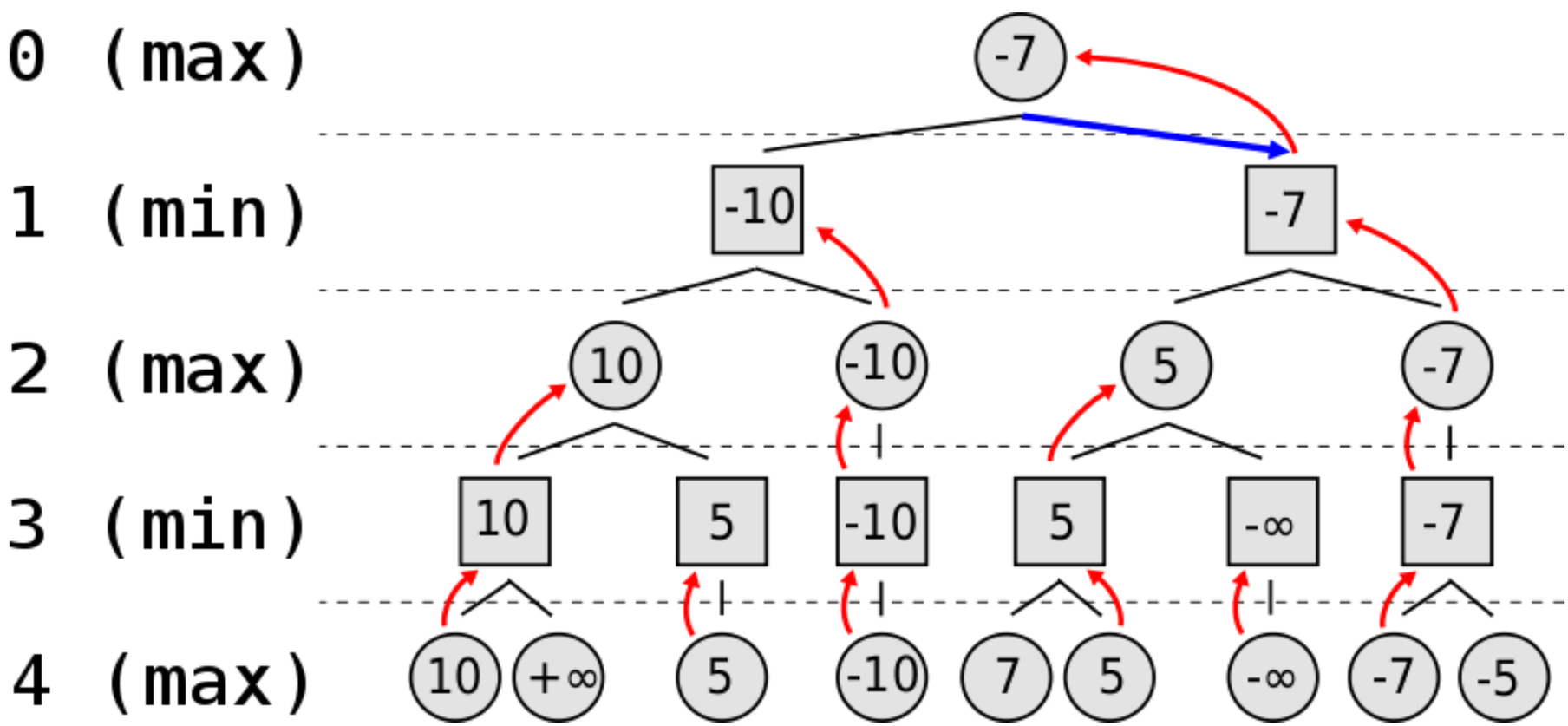


Minimax 알고리즘 (3/6)



Minimax 알고리즘 (4/6)

- 좀 더 복잡한 예제



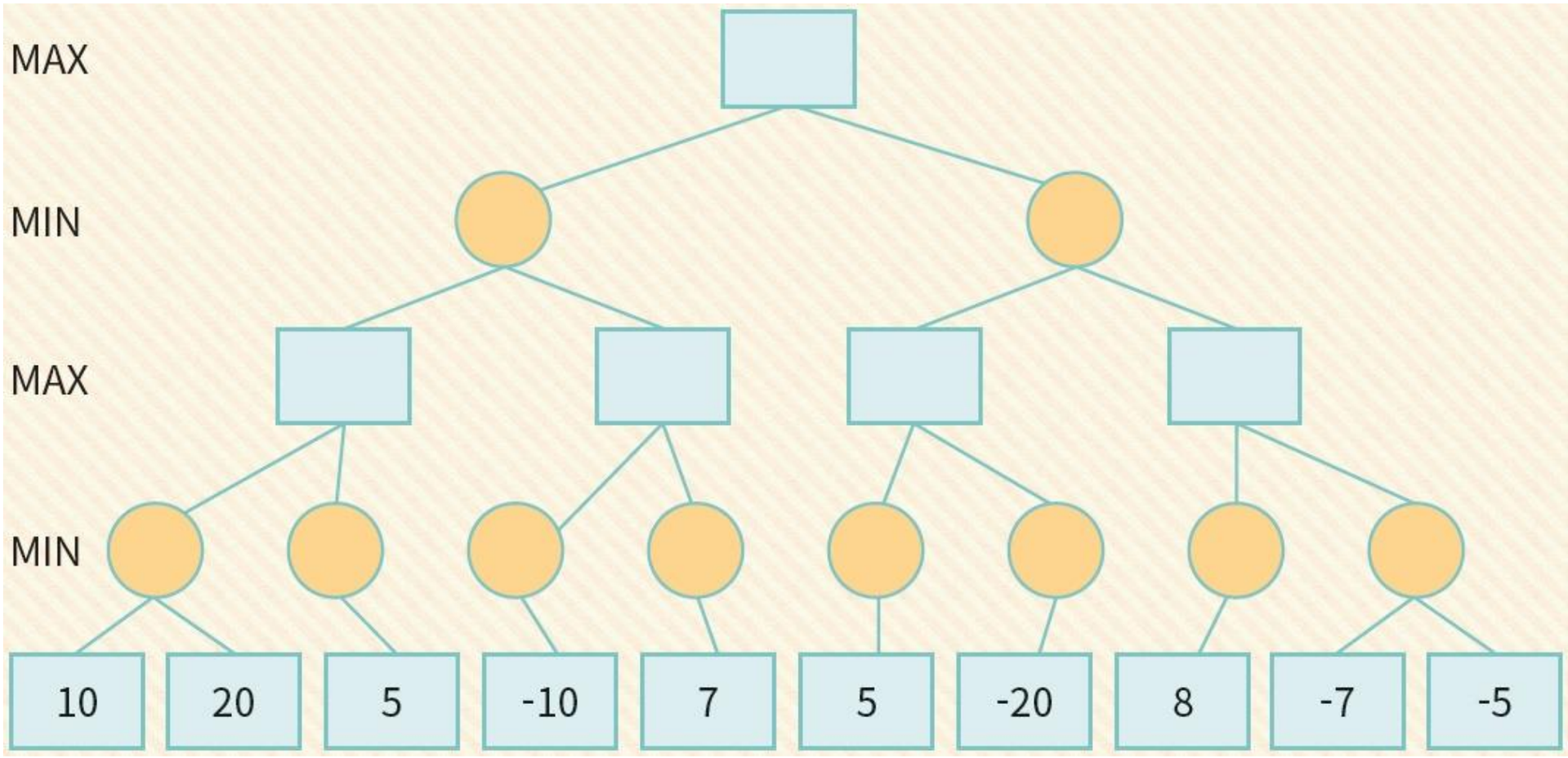
Minimax 알고리즘 (5/6)

- Minimax 알고리즘은 이론적으로는, 모든 결정론적이고 완전한 정보가 주어진 2인용 제로섬 게임에서 최적의 게임 플레이를 보장
- 게임의 상태가 주어지면 알고리즘은 단순히 주어진 상태의 하위 노드(자식 노드) 값을 계산하고, Max의 차례이면 최대값을, Min의 차례이면 최소값을 선택
- Tic-Tac-Toe, Chess, Go 등에서 작동

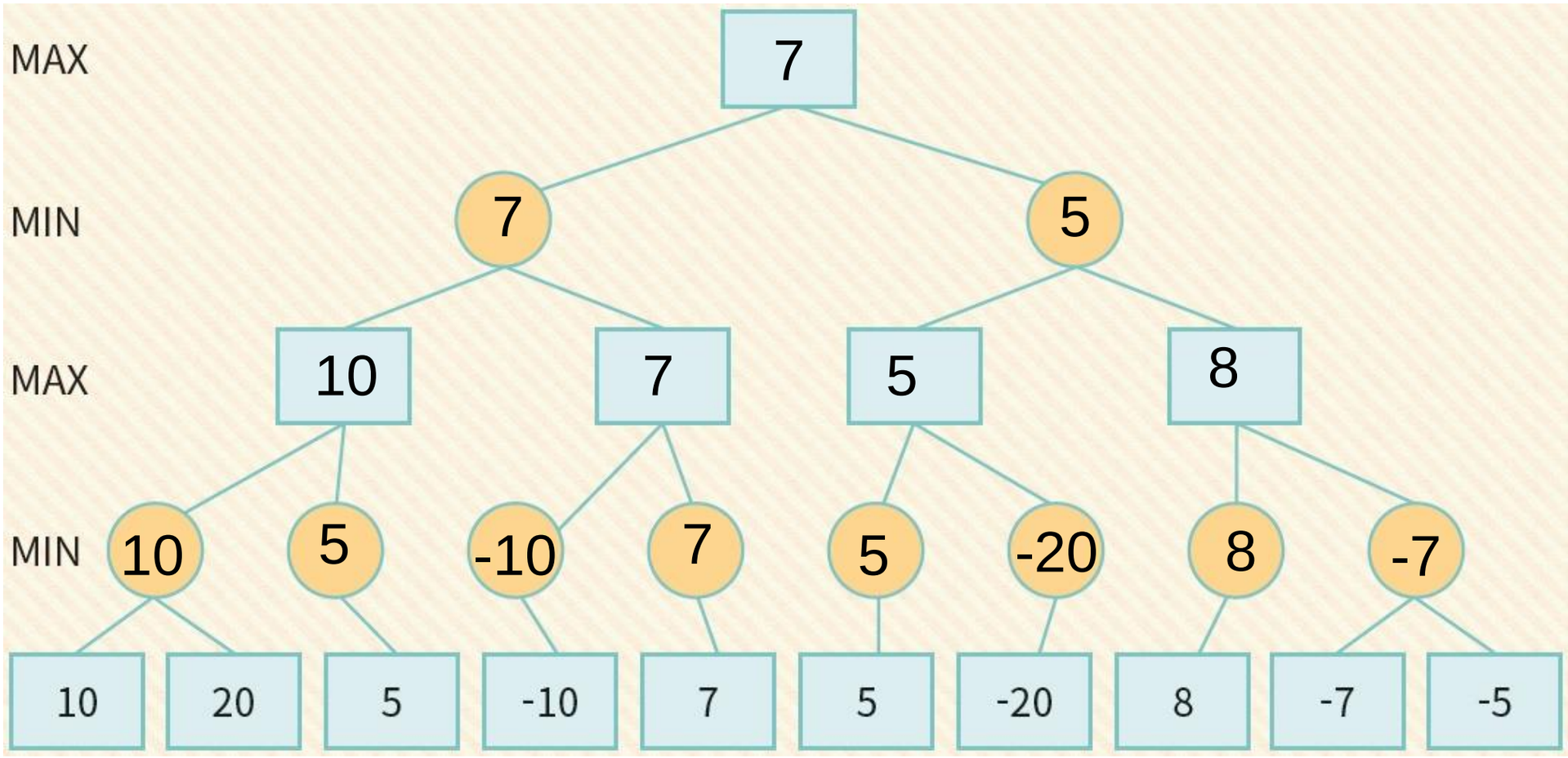
Minimax 알고리즘 (6/6)

- 미니맥스 알고리즘은 게임 트리에 대하여 완벽한 깊이 우선 탐색을 수행
- 만약 트리의 최대 깊이가 m이고 각 노드에서의 가능한 수가 b개라면 미니맥스 알고리즘의 시간 복잡도는 $O(b^m)$
- 바둑은 경우의 수가 약 316!
  - 노드 당 평균 branching factor가 최대 250개 정도
  - 계산해보면 약 10^{761}로 추산
  - 전체 우주는 약 10^{80}개의 원자 만을 포함하는 것으로 추정

연습문제 : Minimax 알고리즘



연습문제 : Minimax 알고리즘

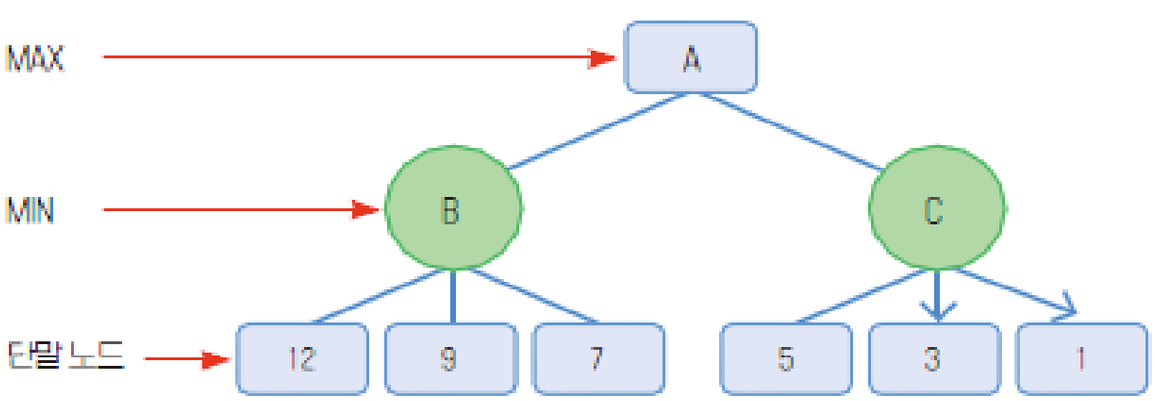


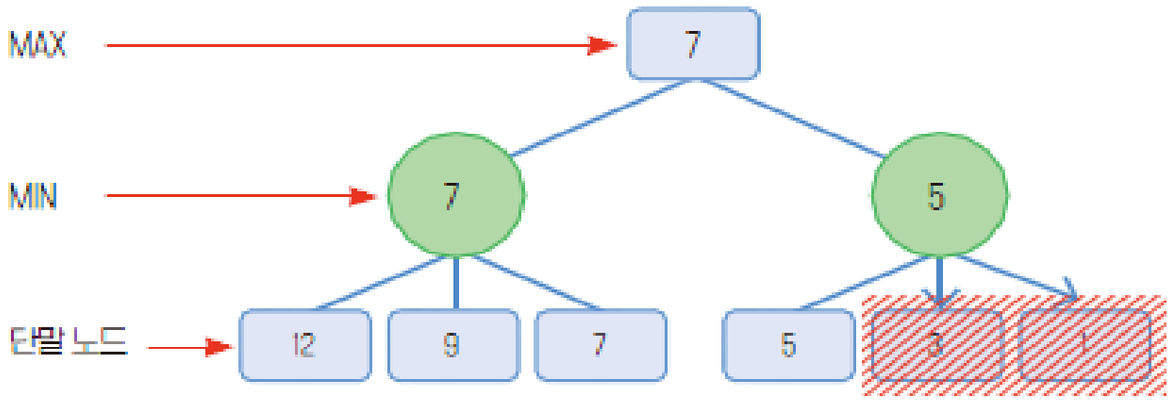
게임트리 탐색 (1/2)

- 알파베타 가지치기(α-β Pruning)
  - 최대한 탐색 시간을 줄이기 위해 특정 노드에 들어가기 전에 이 노드가 정답이 있을 확률이 높은지, 아니면 정답이 있을 확률이 낮은지를 판단하여 이 노드를 탐색할지 아니면 탐색하지 않을지를 결정
  - 기본으로 미니맥스 탐색 알고리즘으로 경로를 탐색하는데, 이때 중요하지 않은 경로는 무시하며 진행

게임트리 탐색 (2/2)

- 알파베타 가지치기(α-β Pruning)의 과정





- 노드 A는 Max의 노드이므로 7보다 같거나 커지게 됨
- C의 왼쪽 노드가 5이므로 C는 5보다 같거나 작아짐
- 노드 A가 7인데, C가 아무리 커봐야 5라면 탐색의 의미가 없어짐 => C의 나머지 노드는 탐색대상에서 제외