

02

강

인공지능

문제풀이(1)

컴퓨터과학과 이병래교수

학습목차

- 1 문제풀이의 개념
- 2 상태공간 탐색에 의한 문제풀이
- 3 깊이우선 탐색 및 너비우선 탐색
- 4 균일비용탐색





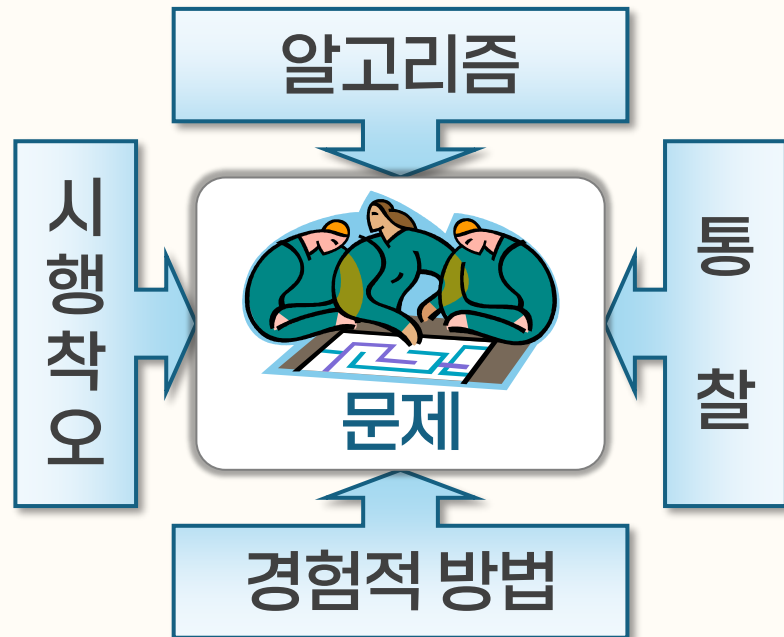
문제풀이의 개념

1. 문제풀이란 무엇인가?

■ 문제풀이

- 직관적으로 단순하게 해결할 수 없는 문제에 대해 문제를 파악하고 문제의 해에 이르는 방법을 찾아내는 일련의 과정

■ 문제풀이에 사용될 수 있는 전략

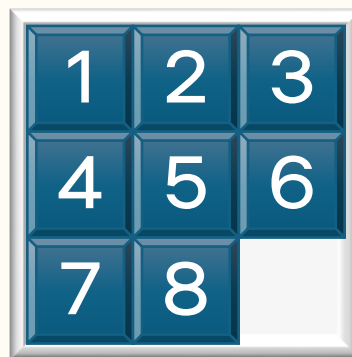


1. 문제풀이란 무엇인가?

■ 8-퍼즐 문제



퍼즐 조각의 초기배치



목표배치



8-퍼즐 문제의 풀이

- 퍼즐 조각의 배치나 조각의 이동 방식 등을 컴퓨터로 표현
- 여러 가지 방향으로 조각을 이동하는 것을 시도하는 과정에서 시행착오를 거쳐 문제 해결

2. 문제의 표현


■ 문제의 상태를 컴퓨터로 표현

예 8-퍼즐 문제

- ➔ **상태(state)**: 퍼즐 판에 나열된 퍼즐 조각 배치 형태
 - 최초로 주어진 문제의 상태: **초기상태**
 - 풀이된 결과에 해당되는 문제의 상태: **목표상태**
- 적절한 자료구조를 이용하여 컴퓨터를 통해 상태를 표현
 - ➔ **상태묘사(state description)**
- 상태를 변화시키기 위한 도구의 표현
 - ➔ **연산자(operator)**

2. 문제의 표현

■ 상태묘사

- 풀이하고자 하는 문제의 상태를 컴퓨터로 처리하기 위한 적절한 자료구조로 표현한 것
 - 문제에 따라 적절한 자료구조를 선택
 - 문제의 상태를 표현하는데 보다 자연스러운 표현 방법
 - 하나의 상태묘사에서 다른 상태묘사로 변화시키는 연산이 용이한 표현
-  기호 열, 벡터, 다차원 배열, 트리, 리스트 등

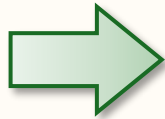
2. 문제의 표현

■ 상태묘사

예 8-퍼즐 문제: 퍼즐 판에 나열된 퍼즐 조각 배치 형태

➔ 2차원 배열로 표현

➔ 효율적인 처리를 할 수 있는 자료구조 설계



```
struct PuzzleState {  
    int  blankX, blankY;  
    char board[3][3];  
};
```


2. 문제의 표현

■ 연산자

- 역할: 어느 한 상태에서부터 변화할 수 있는 다른 상태로 변환함
- 연산자의 정의

① 변환 테이블을 정의하는 방법

- 모든 '입력' 상태묘사에 대해 각각의 상태에서부터 변화할 수 있는 모든 '출력' 상태묘사를 저장하는 목록을 만듦

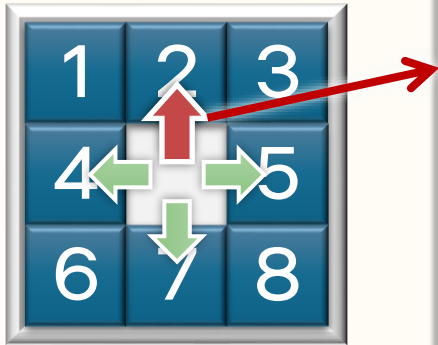
② 일반화된 변환 규칙으로 정의하는 방법

- 하나의 상태묘사를 다른 상태묘사로 변화시키는 일종의 연산 능력을 지닌 함수로 정의

2. 문제의 표현

연산자

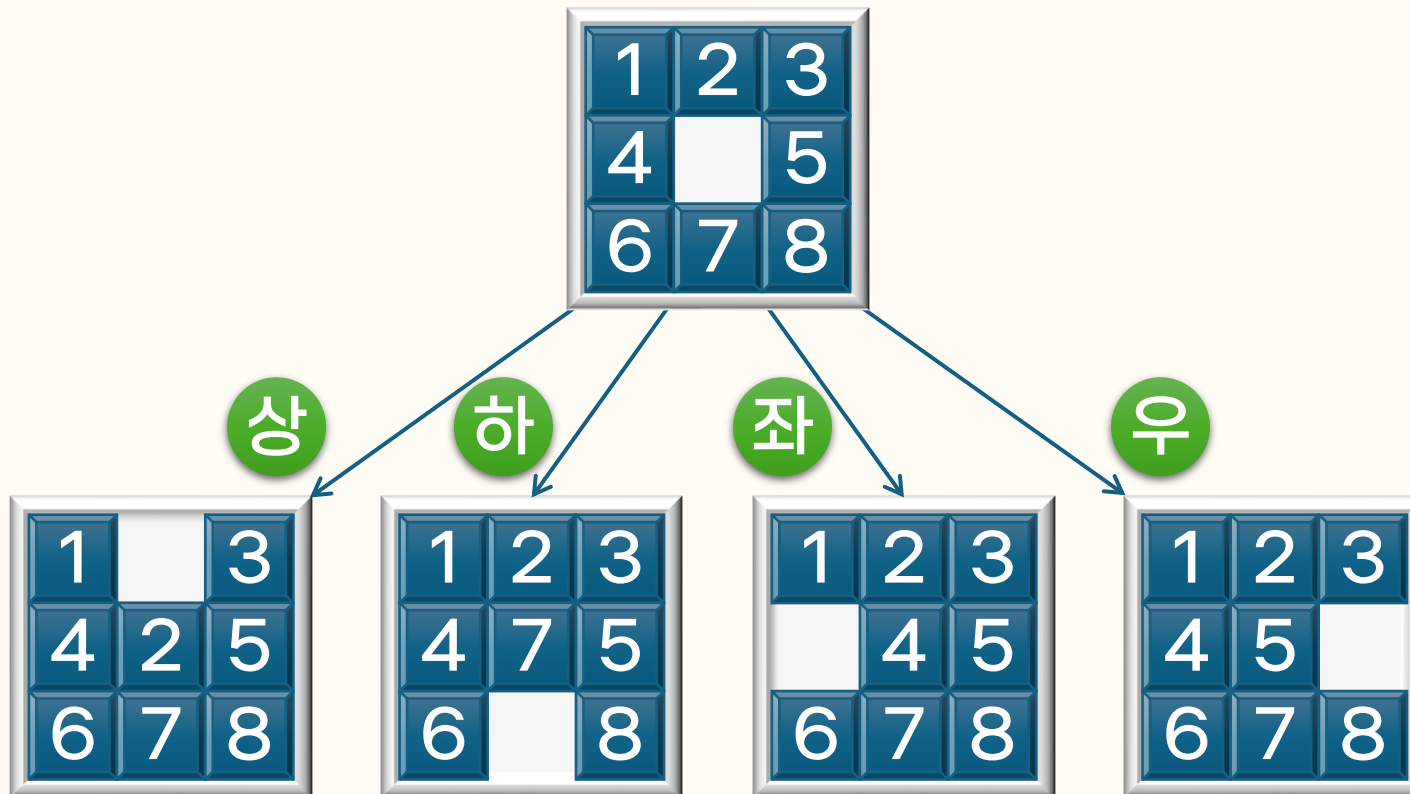
예 8-퍼즐문제: 빈칸의 상, 하, 좌, 우 이동



```
int  opMvBlnkUp(PuzzleState* s) {  
    if (s->blankY > 0) {  
        s->board[s->blankX][s->blankY] =  
            s->board[s->blankX][s->blankY-1];  
        s->board[s->blankX][--s->blankY] = ' ';  
        return 1;  
    }  
    else  
        return 0;  
}
```

2. 문제의 표현

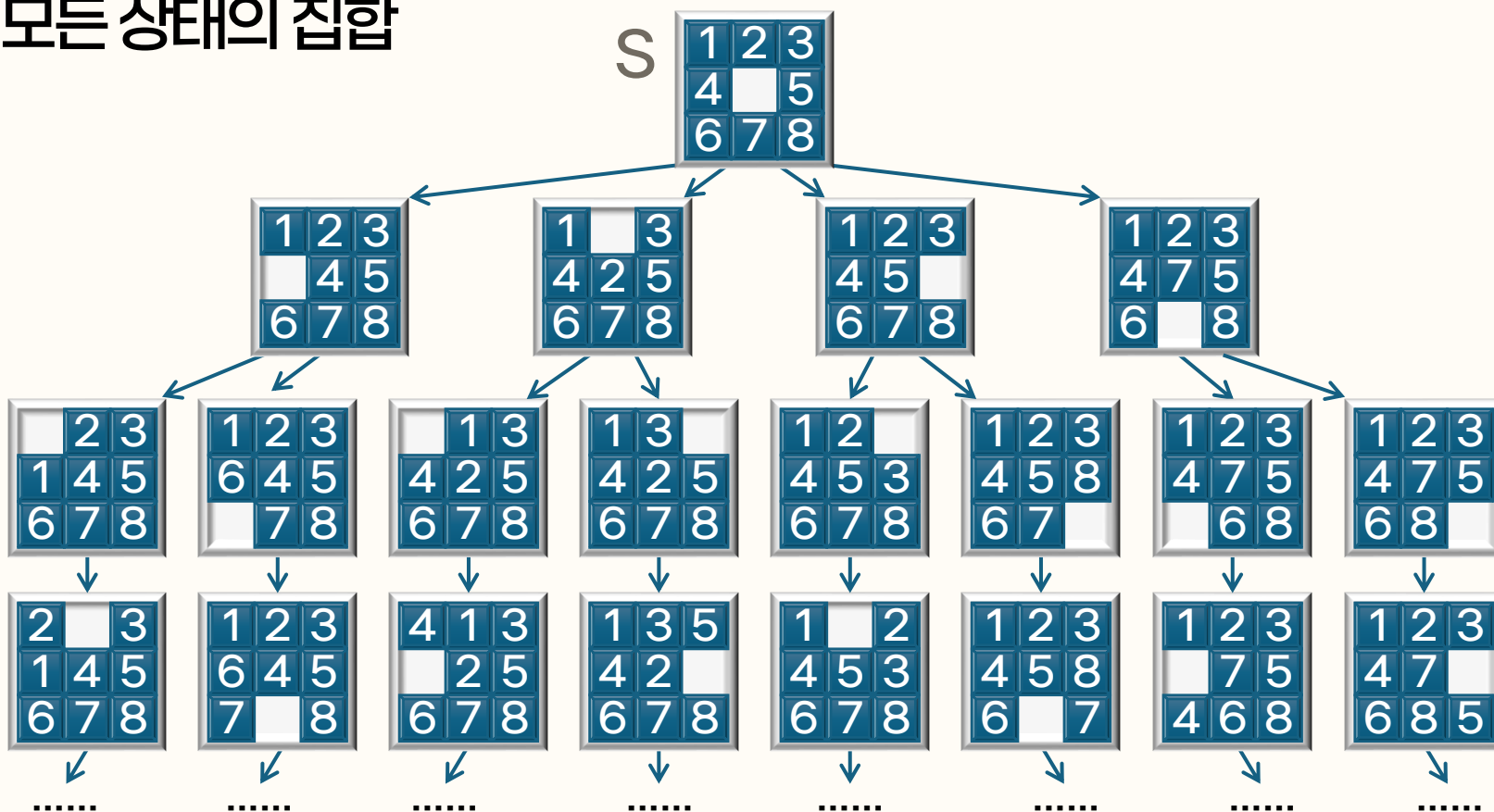
- 연산자 적용에 의한 상태 사이의 관계 표현
 - 방향성 그래프를 이용하여 부모상태와 후계상태의 관계를 표현



2. 문제의 표현

■ 상태공간(state space)

- 정의된 연산자 집합을 이용하여 초기상태로부터 얻을 수 있는 모든 상태의 집합



2. 문제의 표현

■ 상태공간에서 문제풀이를 위한 문제의 표현



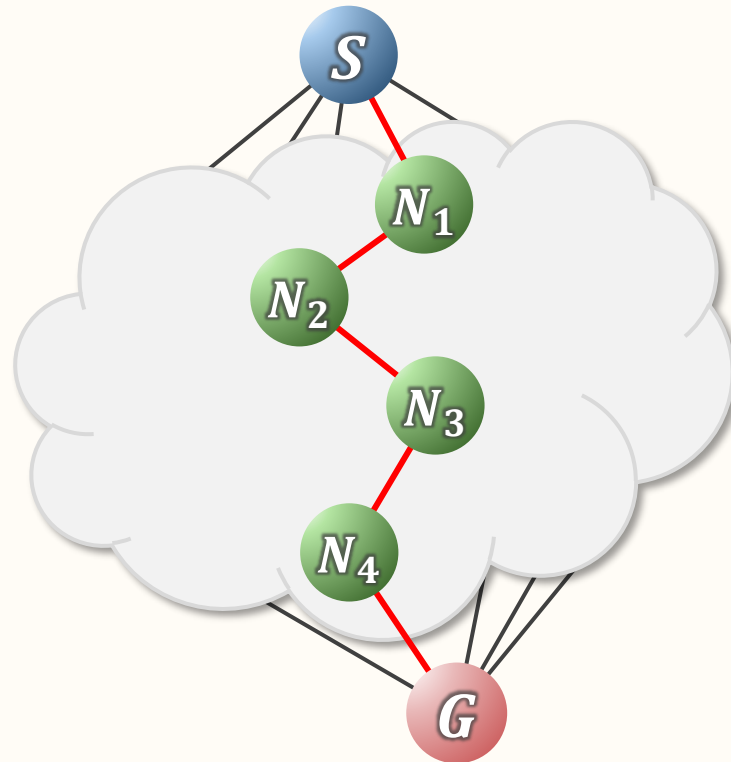
3. 상태공간 탐색에 의한 문제풀이

■ 상태공간에서의 문제풀이

- 초기상태에서 시작하여 목표상태에 도달할 수 있는 일련의 연산자를 찾는 것

➡ 그래프에서 경로의 탐색 문제

- 시행착오를 거치면서 목표상태에 도달하는 탐색 과정
- 탐색에 유용한 지식을 사용함으로써 방대한 상태공간에서 탐색 범위를 좁히려고 함

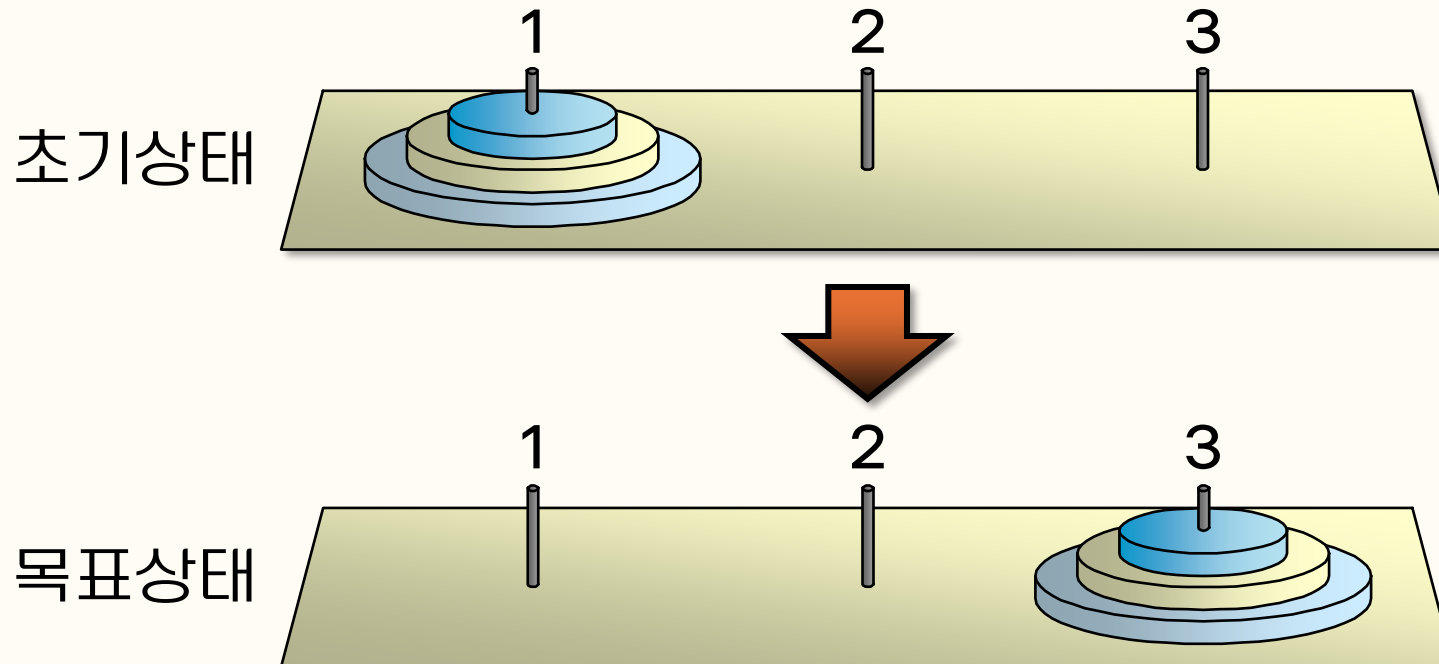


4. 문제축소에 의한 문제풀이

■ 문제축소(problem reduction)

- 주어진 문제를 부분문제로 분할하여 풀이하는 방식

예 하노이탑 문제

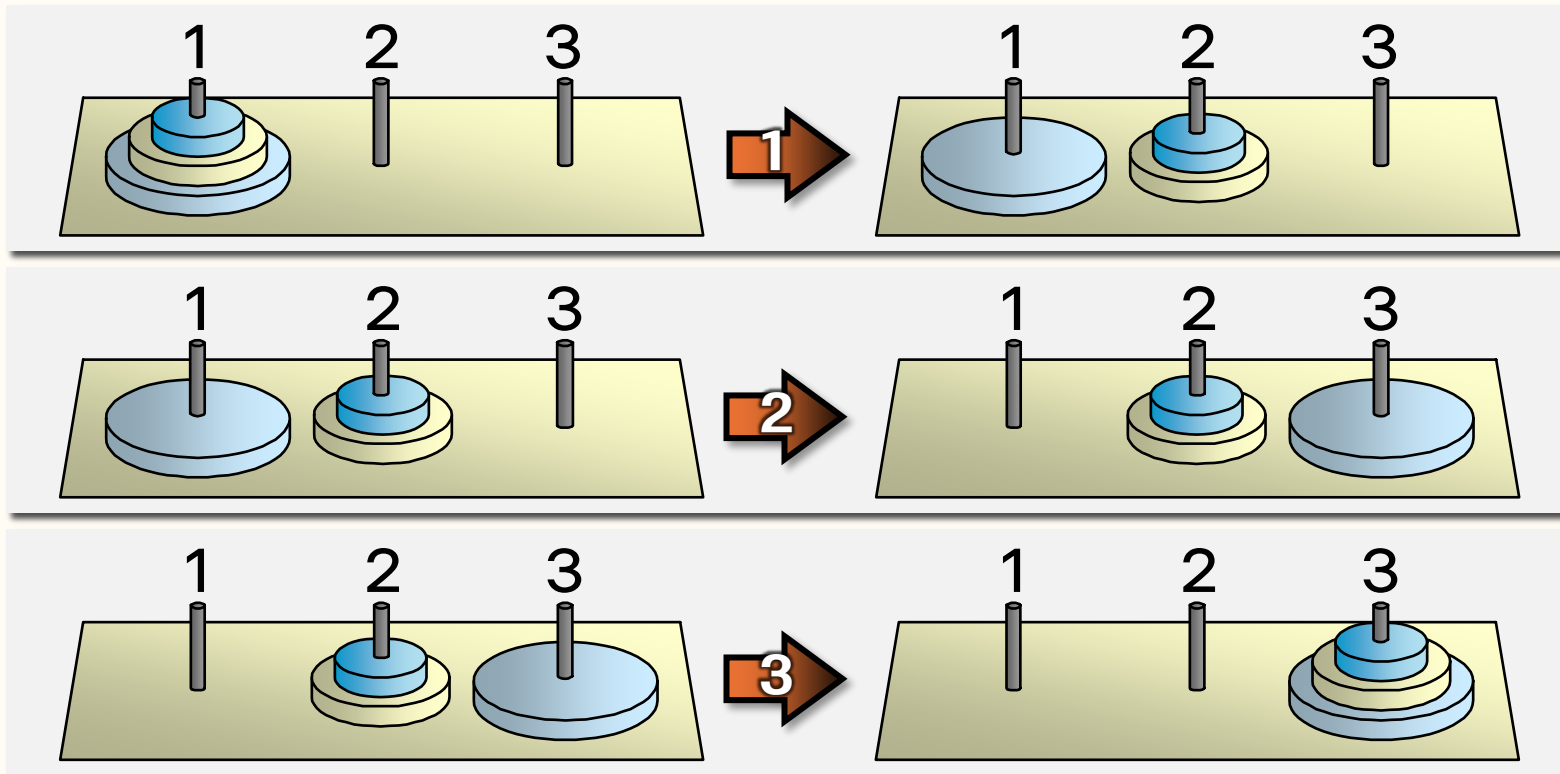


4. 문제축소에 의한 문제풀이

■ 문제축소(problem reduction)

- 주어진 문제를 부분문제로 분할하여 풀이하는 방식

예 하노이탑 문제





상태공간 탐색에 의한 문제풀이

1. 상태공간 탐색에 의한 문제풀이 개요

■ 탐색 과정

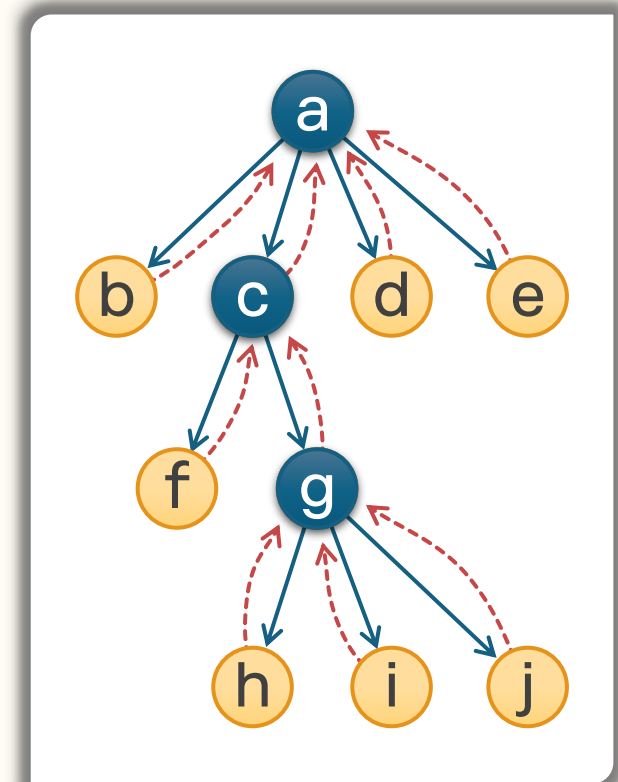
- 정해진 기준에 따라 노드를 선택
- 선택된 노드(상태)에 적용할 수 있는 모든 연산자를 가하여 모든 후계노드(후계상태)를 생성

⇒ **노드의 확장**

- 후계노드에 부모노드를 가리키는 포인터를 첨부

⇒ **탐색에 성공한 후 풀이 경로를 알 수 있게 함**

- 목표노드가 있는지 검사함
- 원하는 목표를 찾지 못했다면 정해진 기준에 따라 다음 노드를 선택하여 탐색 과정 반복



1. 상태공간 탐색에 의한 문제풀이 개요

■ 탐색 과정

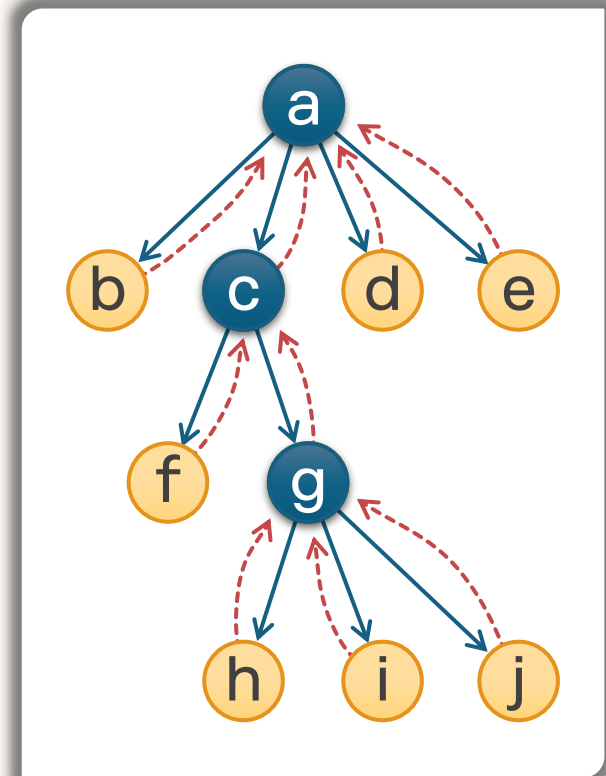
- **OPEN**: 앞으로 확장할 노드를 저장하는 리스트

$\{(b, a), (d, a), (e, a), (f, c), (h, g), (i, g), (j, g)\}$

- **CLOSED**: 이미 확장한 노드를 저장하는 리스트

$\{(a, \text{NULL}), (c, a), (g, c)\}$

- 탐색 알고리즘에 따라 정해진 순서로 OPEN에서 확장할 노드를 꺼내 CLOSED로 옮김
- 확장 결과 생성된 노드를 OPEN의 적절한 위치에 저장함



2. 탐색 방법의 종류

❑ 맹목적 탐색(blind search)

- 목표노드의 위치와 무관한 순서로 노드 확장
- 매우 소모적인 탐색을 할 가능성이 높음

❑ 경험적 탐색(heuristic search)

- 문제영역에서 사용할 수 있는 목표노드의 위치와 관련된 경험적 정보를 사용



경험적 정보: 항상 옳은 것은 아니지만, 개연성이 있어 많은 경우 잘 맞는 정보

2. 탐색 방법의 종류

■ 탐색의 목적 및 경험적 정보 사용에 따른 분류

정보사용 \ 목적	임의 경로 탐색	최적 경로 탐색
맹목적 탐색	깊이우선 탐색 너비우선 탐색	균일비용 탐색
경험적 탐색	언덕오르기 탐색 최적우선 탐색 모의 담금질	A* 알고리즘



깊이우선 탐색 및 너비우선 탐색

1. 깊이우선 탐색(depth-first search)

■ 탐색 방법

- 탐색 진행방향(깊이방향)으로 계속 전진하여 목표를 탐색하는 방법
 - ➡ 가장 최근에 생성된 노드를 가장 먼저 확장함
 - ➡ OPEN은 **스택** 구조
- 목표에 도달할 수 없는 경로를 계속 탐색하게 될 수 있음
 - ➡ **깊이제한(depth bound)**을 정할 수 있음
- 깊이제한에 도달하거나 더 이상 진행할 경로가 없을 경우 이전 상태 중 다른 경로를 선택할 수 있는 위치로 복귀하여 탐색을 계속함

1. 깊이우선 탐색(depth-first search)

예 8-퍼즐 문제

2	8	3
1	6	4
7		5

초기상태



1	2	3
8		4
7	6	5

목표상태

1. 깊이우선 탐색(depth-first search)

예 8-퍼즐 문제



1. 깊이우선 탐색(depth-first search)

깊이우선 탐색 알고리즘

```
1  출발노드를 OPEN에 삽입;
2  while not empty(OPEN) do
3      n = OPEN의 제일 앞 노드;
4      n을 OPEN에서 제거하여 CLOSED에 넣음;
5      if depth(n) < 깊이제한 then
6          노드n을 확장하여 모든 후계노드를 생성;
7          생성된 후계노드들에게 부모노드 n을 가리키는 포인터를 첨부;
8          if 후계노드 중 목표노드가 존재 then
9              그 후계노드로부터 포인터를 역으로 추적하여 풀이경로 구성;
10             return 탐색성공;
11         else
12             후계노드를 OPEN의 앞에 넣음;
13         end-if;
14     end-if
15 end-while
16 return 탐색실패;
```

2. 너비우선 탐색(breadth-first search)

■ 탐색 방법

- 트리의 레벨 순서에 따라 노드를 확장함

➡ 생성된 순서에 따라 노드를 확장함

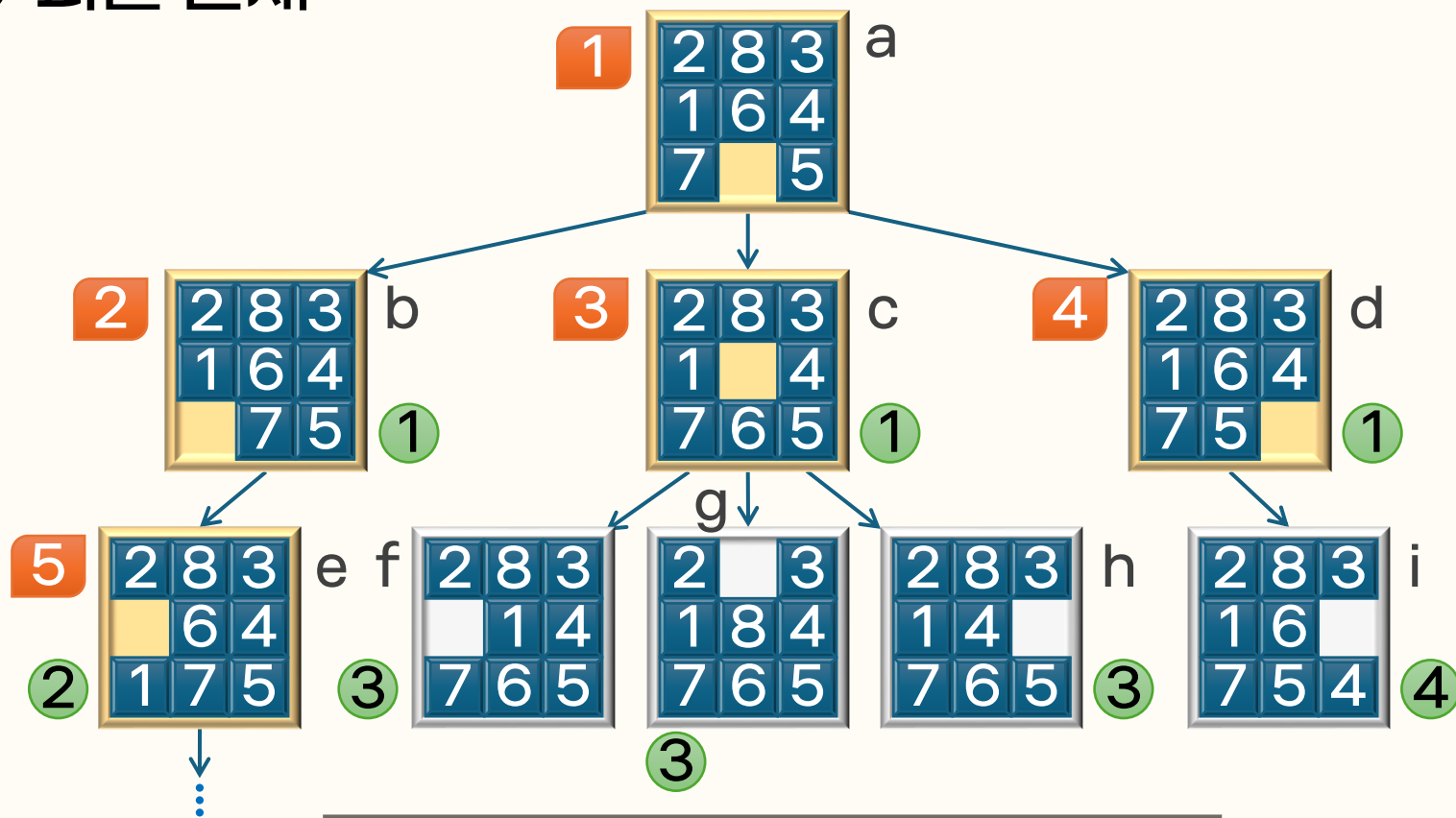
➡ OPEN은 큐 구조



만일 해가 존재한다면 출발노드에서 목표노드까지 도달하는
최단길이 경로를 찾는 것을 보장함

2. 너비우선 탐색(breadth-first search)

예 8-퍼즐 문제



2. 너비우선 탐색(breadth-first search)

너비우선 탐색 알고리즘

```
1 출발노드를 OPEN에 삽입;
2 while not empty(OPEN) do
3     n = OPEN의 제일 앞 노드;
4     n을 OPEN에서 제거하여 CLOSED에 넣음;
5     노드n을 확장하여 모든 후계노드를 생성;
6     생성된 후계노드에게 부모노드 n을 가리키는 포인터를 첨부;
7     if 후계노드 중 목표노드가 존재 then
8         그 후계노드로부터 포인터를 역으로 추적하여 풀이경로 구성;
9         return 탐색성공;
10    else
11        후계노드를 OPEN의 뒤에 넣음;
12    end-if;
13 end-while;
14 return 탐색실패;
```



균일비용 탐색

1. 균일비용 탐색(uniform-cost search)의 개념

■ 노드 사이의 경로비용

- 한 상태에서 다른 상태로 이동하기 위한 필요한 '비용'

예 8퍼즐 문제: 퍼즐 조각을 이동하여 원하는 배열을 얻고자 함

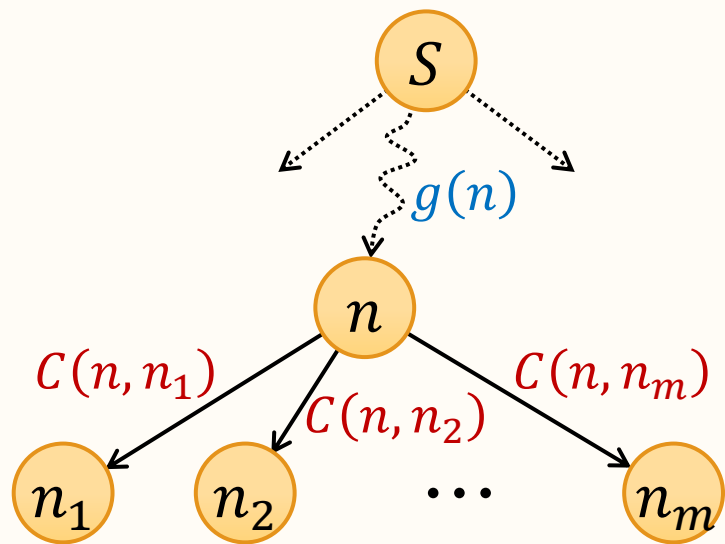
➡ 퍼즐 조각을 이동하는 횟수를 비용으로 고려할 수 있음

예 지도상의 경로 탐색 문제

➡ 한 지점에서 다음 지점으로 이동하는 거리를 비용으로 고려할 수 있음

1. 균일비용 탐색(uniform-cost search)의 개념

- 출발노드 S 로부터 노드 n 의 후계노드 n_i 까지의 경로비용



$$g(n_i) = g(n) + C(n, n_i), \\ i = 1, 2, \dots, m$$

- OPEN의 노드들 중 **출발노드로부터의 경로비용이 최소인 노드**를 선택하여 확장함
- **최소비용 경로**를 탐색할 수 있음

2. 균일비용 탐색 알고리즘

균일비용 탐색 알고리즘

```
1 출발노드에 경로비용 0을 지정하여 OPEN에 삽입함;
2 while not empty(OPEN) do
3     n = OPEN의 제일 앞 노드;
4     n을 OPEN에서 제거하여 CLOSED에 넣음;
5     if n == 목표노드 then
6         노드 n으로부터 포인터를 역으로 추적하여 풀이경로 구성;
7         return 탐색성공;
8     end-if;
9     노드 n을 확장하여 후계노드  $n_1, n_2, \dots, n_m$ 을 생성;
10    for i=1 to m do
11         $n_i$ 에 부모노드인 n을 가리키는 포인터 첨부;
12         $n_i$ 의 경로비용  $g(n_i)$ 를 계산;
13        if  $\exists n_{old} (n_{old} \in OPEN \text{ and } n_{old} == n_i)$  then
14            .....
```

2. 균일비용 탐색 알고리즘

균일비용 탐색 알고리즘

```
2 while not empty(OPEN) do
3   n = OPEN의 제일 앞 노드;
...   .....
9   노드 n을 확장하여 후계노드  $n_1, n_2, \dots, n_m$ 을 생성;
10  for i=1 to m do
11     $n_i$ 에 부모노드인 n을 가리키는 포인터 첨부;
12     $n_i$ 의 경로비용  $g(n_i)$ 를 계산;
13    if  $\exists n_{old} (n_{old} \in OPEN \text{ and } n_{old} == n_i)$  then
14      if  $g(n_{old}) > g(n_i)$  then
15         $n_{old}$ 를  $n_i$ 로 대체; // 비용이 더 큰 기존 경로를 제거
16      else
17         $n_i$ 를 제거; // 비용이 더 큰 새로운 경로를 제거
18      end-if;
19    else if .....
```

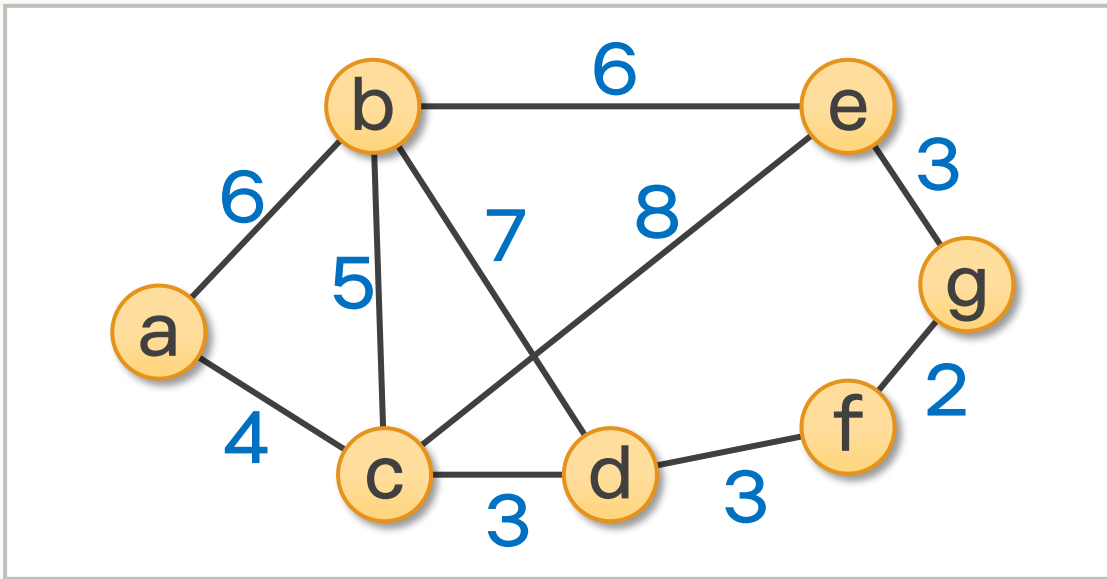
2. 균일비용 탐색 알고리즘

균일비용 탐색 알고리즘

```
2 while not empty(OPEN) do
3   n = OPEN의 제일 앞 노드;
...   .....
10  for i=1 to m do
...    .....
19    else if  $\exists n_{old} (n_{old} \in \text{CLOSED and } n_{old} == n_i)$  then
20       $n_i$ 를 제거; //  $g(n_i)$ 는  $g(n_{old})$ 보다 작을 수 없음
21    else
22       $n_i$ 를 OPEN에 추가;
23    end-if;
24  end-for;
25  OPEN을 경로비용 g의 오름차순으로 정렬;
26 end-while;
27 return 탐색실패;
```

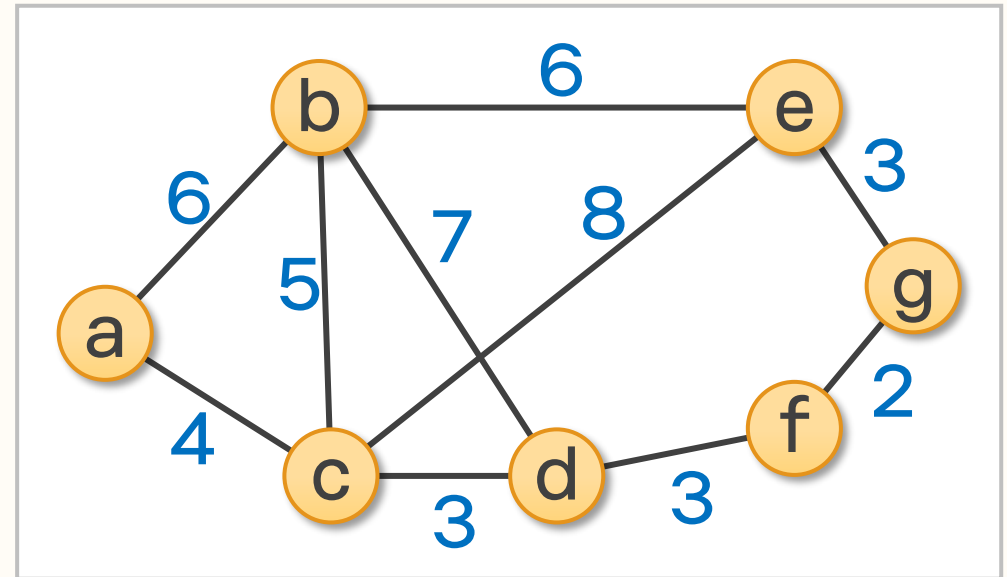
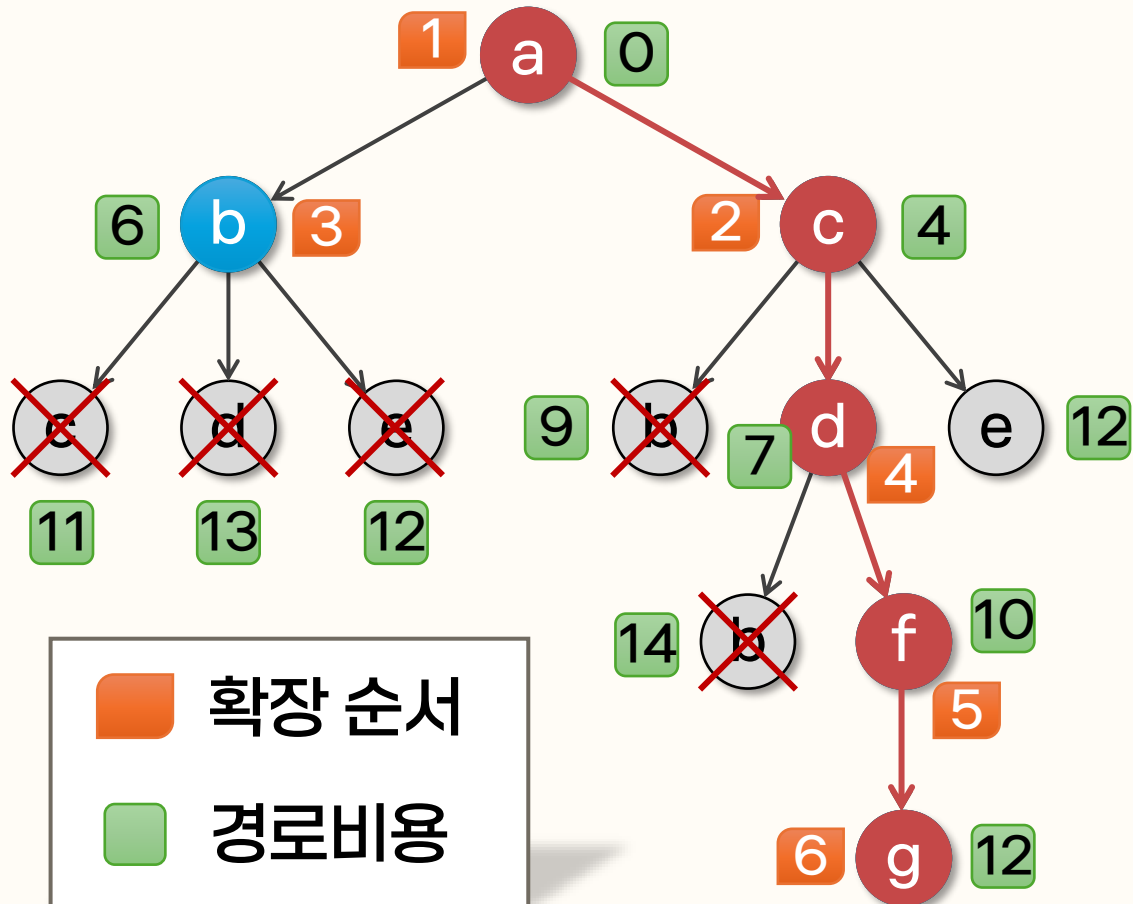
3. 예제 : 최단길이 경로 탐색 문제

a, b, c, d, e, f, g라는 7개의 도시를 연결하는 도로망이 건설되어 있다. 어떤 여행자가 도시a를 출발하여 도시g까지 가는 경로를 찾고자 한다. 다음 그림은 각 도시를 연결하는 도로와 그 거리를 그래프로 표현한 것이다. 균일비용 탐색을 하여 최단길이 경로를 탐색하라.



3. 예제 : 최단길이 경로 탐색 문제

탐색트리



정리하기

- ✓ 상태묘사란 풀이하고자 하는 문제의 상태를 컴퓨터로 처리하기 위한 적절한 자료구조로 표현한 것이다.
- ✓ 연산자는 문제의 어느 한 상태를 다른 상태로 변환하는 역할을 한다.
- ✓ 상태공간은 정의된 연산자 집합을 이용하여 초기상태로부터 얻을 수 있는 모든 상태의 집합으로, 그래프 형태로 표현할 수 있다.
- ✓ 상태묘사 및 초기상태의 정의, 목표상태의 정의, 연산자의 정의를 함으로써 상태공간에서 문제를 표현한다.
- ✓ 상태공간 탐색에 의한 문제풀이 방식은 시행착오를 통해 초기상태로부터 목표상태에 도달하는 경로를 탐색한다.

정리하기

- ✔ 맹목적 탐색은 다음 확장할 노드를 선택할 때 목표노드의 위치에 대한 정보를 사용하지 않는다.
- ✔ 깊이우선 탐색에서는 가장 최근에 생성된 노드를 가장 먼저 확장함으로써 탐색 진행방향(깊이 방향)으로 계속 전진하여 목표를 탐색한다.
- ✔ 너비우선 탐색에서는 생성된 순서에 따라 노드를 확장함으로써, 트리의 레벨 순서에 따라 노드를 확장한다. 만일 해가 존재한다면 출발노드에서 목표노드까지의 최단길이 경로를 찾는 것을 보장한다.
- ✔ 균일비용 탐색에서는 출발노드로부터의 경로비용이 가장 작은 노드를 먼저 확장한다. 최소비용 경로를 탐색할 수 있다.

03강

다음시간안내 ▶▶▶

문제풀이(2)