Tree Search

김성영교수
국립금오공과대학교
컴퓨터공학부

※본 자료는 국립금오공과대학교 컴퓨터공학부 황준하교수님의 자료를 참조 및 수정한 것임

Contents

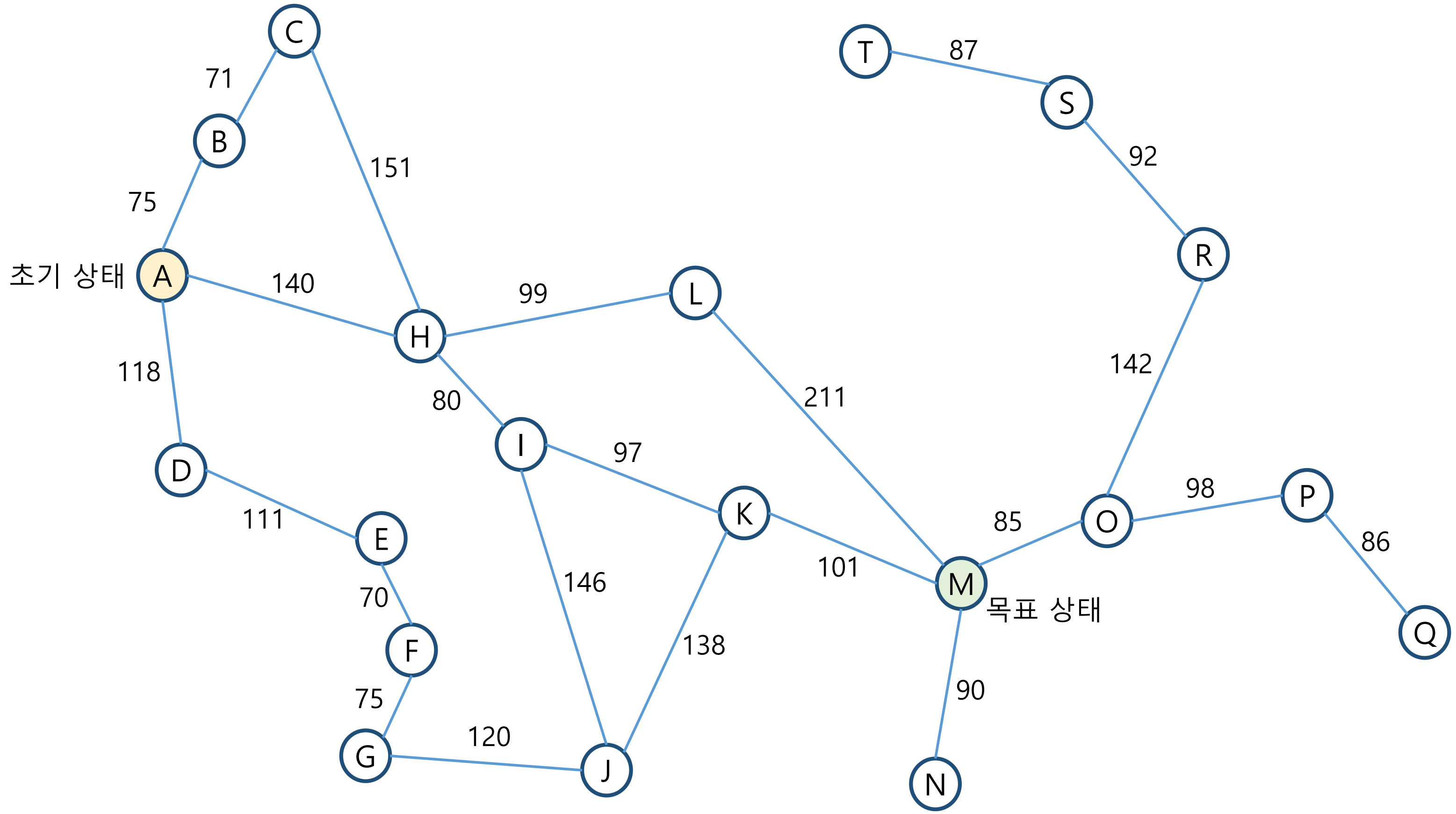
Introduction

Tree 탐색이란?

- 탐색문제의 문제 정의는 초기 상태와 목표 상태를 포함한 상태를 정의
- 상태 공간 상에서 탐색 tree를 만들어가는 과정
  - 상태 공간(state space): 모든 상태(초기 상태, 목표 상태 등)들을 포함하고 있는 공간
  - Root 노드: 초기 상태
    - 연산자를 적용하여 현재 상태를 확장(expanding)하여 새로운 상태 노드들 생성
  - 목표 상태가 나올 때까지 반복적으로 수행 → 하나의 해 도출
- 어떤 노드를 먼저 펼칠 것인가? (=탐색 전략)
  - 다양한 tree 탐색 알고리즘
  - depth-first 탐색, breadth-first 탐색 등

Tree 탐색 이해를 위한 예제

- 최단 경로 찾기 문제
  - A로부터 출발하여 M까지 가는 최단 경로 찾기

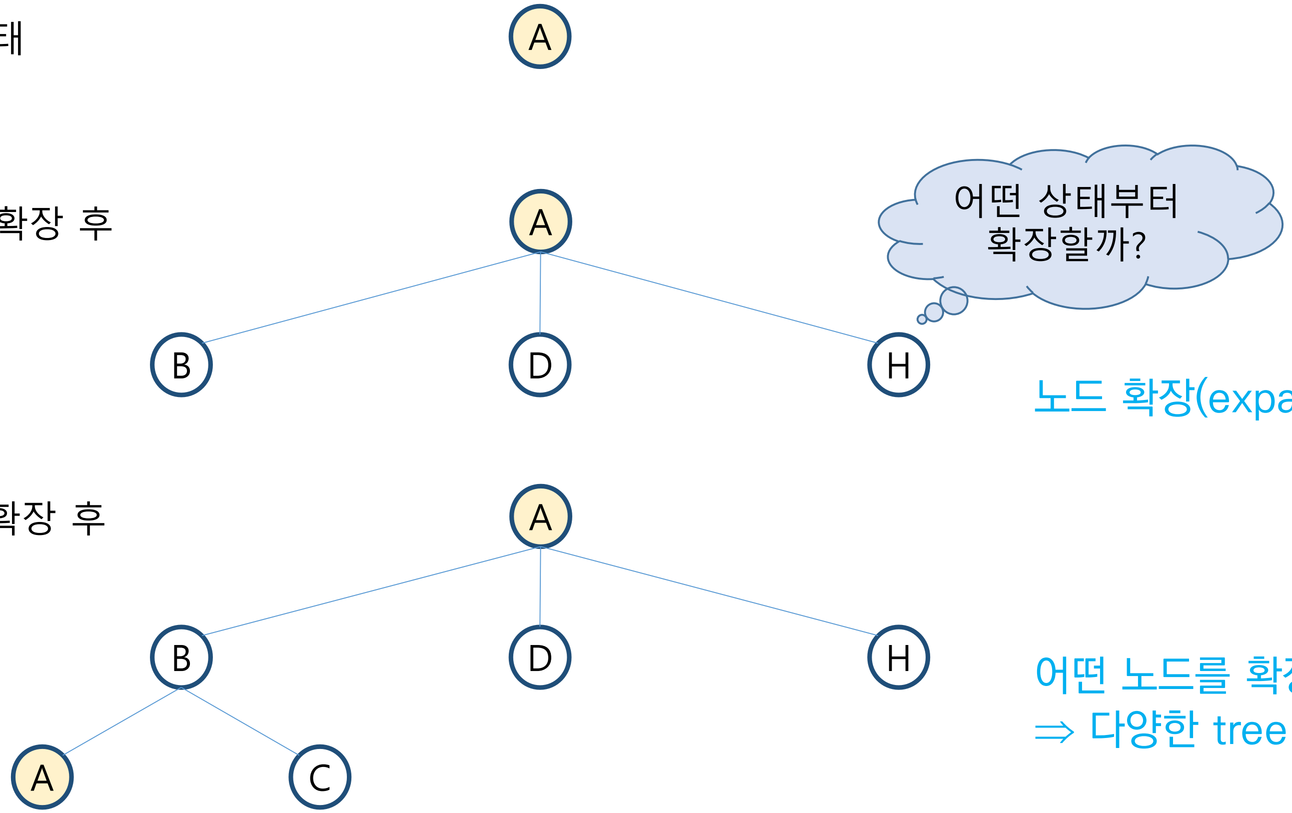


최단 경로 찾기 문제를 위한 tree 탐색 과정

(a) 초기 상태

(b) 상태 A 확장 후

(c) 상태 B 확장 후



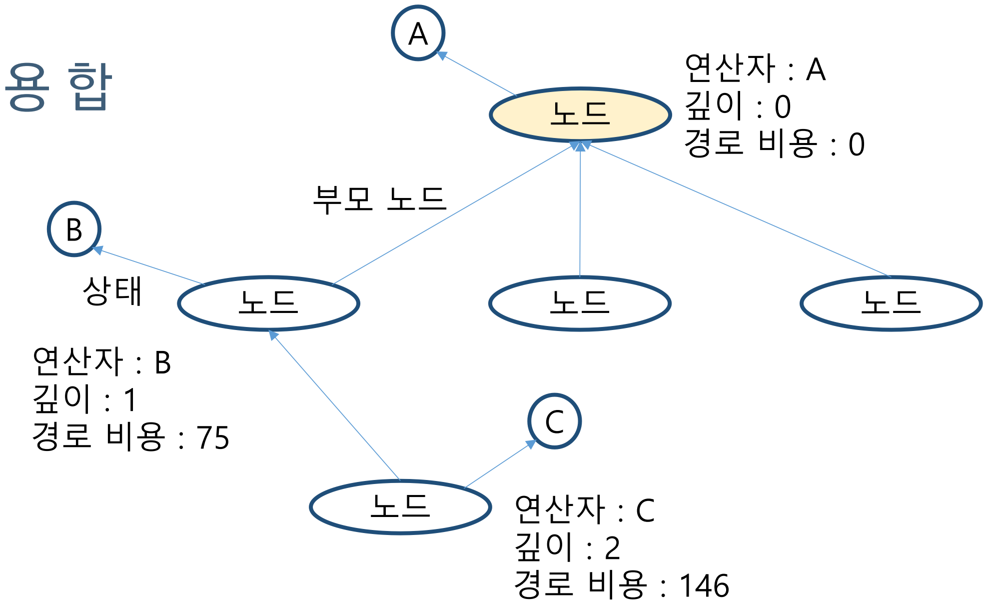
노드 확장(expanding)

어떤 노드를 확장?
⇒ 다양한 tree 탐색 알고리즘이 가능

실제 적용 시에는 이미 등장했던 상태는 제외(무한 반복 방문 회피)

일반적인 tree 탐색 알고리즘 (1)

- 노드: tree 구성을 위한 상태를 포함한 정보들
  - 상태(state) : 현재 상태
  - 부모 노드(parent node) : 부모 노드를 통해 초기 노드까지 추적
  - 연산자(operator) : 부모 상태에서 현재 상태로의 이동 연산자
  - 깊이(depth) : tree 상에서의 깊이로 root 노드의 깊이는 0
  - 경로 비용(path cost) : 초기 상태로부터 현재 상태까지의 비용 합
- fringe (=frontier)
  - 노드들이 확장 전에 저장되는 공간
  - 확장을 위해 선택된 노드는 fringe에서 제거
  - Depth-first 탐색: 스택으로 구현
  - Breadth-first 탐색: 큐로 구현



일반적인 Tree

```
Function TreeSearch(problem, fringe) return a solution or failure
    initial_node = MakeNode(initial_state(problem))
    Insert initial_node to fringe
    repeat
        if fringe is empty
            return failure
        node = Get a node from fringe
        if State[node] is the goal
            return solution(node)
        expanded_nodes = Expand(node, problem)
        Insert all nodes in expanded_node to fringe
```

부모 NULL, 연산자 NULL, 깊이 0, 경로 비용 0

```
Function Expand(node, problem) return a set of nodes
    successors = empty set
    for <operator, next_state> SuccesorFunction(problem, State[node])
        n = a new Node
        State[n] = next_state
        ParentNode[n] = node
        Operator[n] = operator
        Depth[n] = Depth[node] + 1
        PathCost[n] = PathCost[node] + StepCost(node, operator, n)
        add n to successors
    return successors
```

일반적인 Tree 탐색 알고리즘

Tree 탐색 알고리즘의 성능 평가

- 해의 도출 여부(completeness)
  - 하나의 해를 찾을 수 있다는 보장이 있는가?
- 최적해 도출 여부(optimality)
  - 반환하는 해가 최적해임을 보장하는가?
- 시간 복잡도(time complexity)
  - 하나의 해를 찾는 데 최악의 경우 얼마나 많은 시간이 소요되는가?
    = 하나의 해를 찾는 데 최악의 경우 확장되는 노드의 수가 몇 개인가?
- 공간 복잡도(space complexity)
  - 하나의 해를 찾는 데 최악의 경우 얼마나 많은 메모리를 필요로 하는가?
    - 어떤 한 순간에 필요로 하는 가장 많은 메모리의 크기는?
    - 하나의 해를 찾기까지 총 1,000개의 노드가 확장된다 하더라도 어떤 순간에 최대로 저장해야 되는 노드의 수가 100개라면 공간 복잡도는 O(100)이 됨

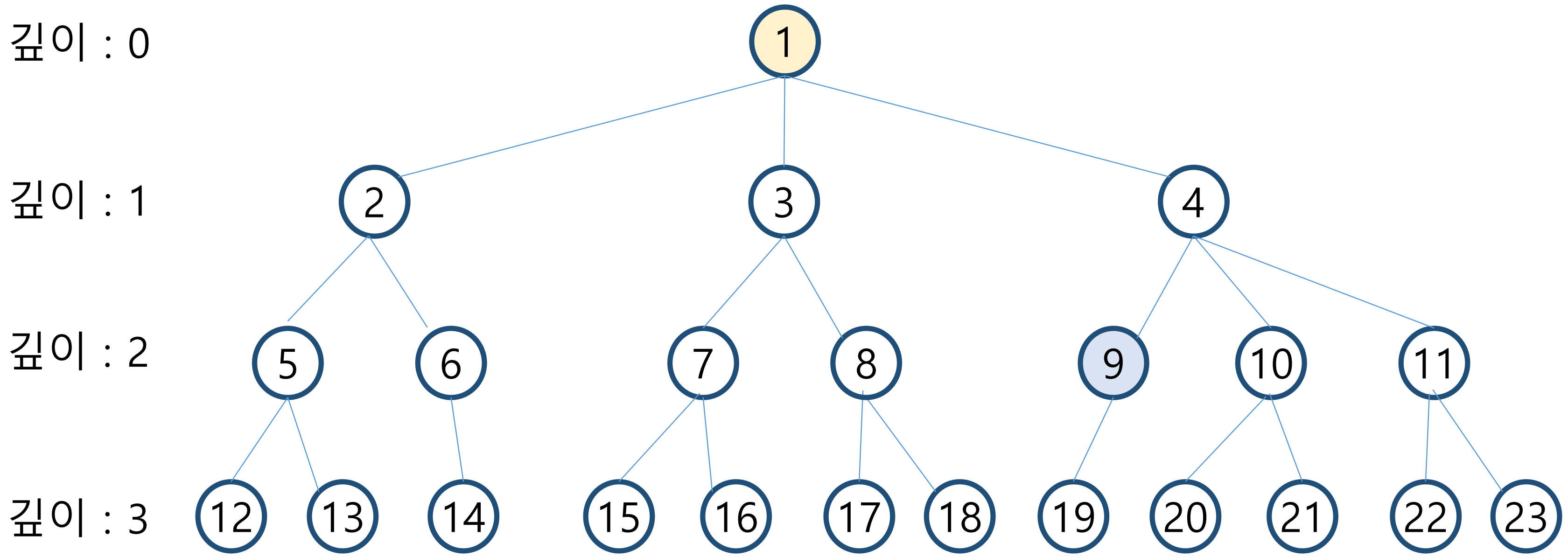
Tree 탐색 알고리즘의 분류

- Uninformed Search (= Blind Search)
  - 확장 노드 선택: 초기 상태부터 현재 상태까지의 경로 정보만 사용
  - Breadth-first Search, Uniform Cost Search, Depth-first Search, Depth-limited Search, Iterative Deepening Search
- Informed Search (= Heuristic Search)
  - 현재 상태부터 목표 상태까지의 경로 정보 사용 (어떻게?)
    - 추정치 활용
    - 최단 경로 찾기 문제 : 상태 A로부터 상태 B, D, H 중 어떤 것부터 확장할 것인가?
  - Greedy Best-first Search, A* Search

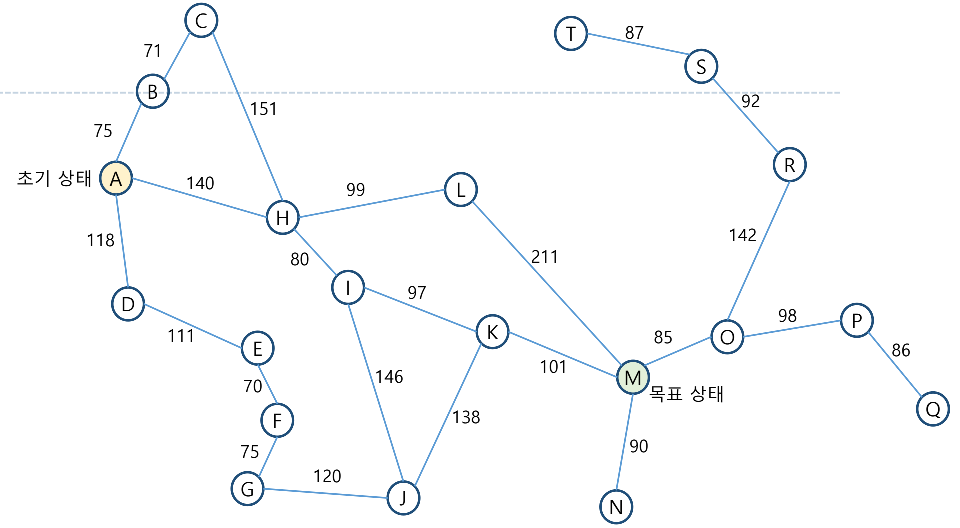
Uninformed Search

너비 우선 탐색 (Breadth-first Search)

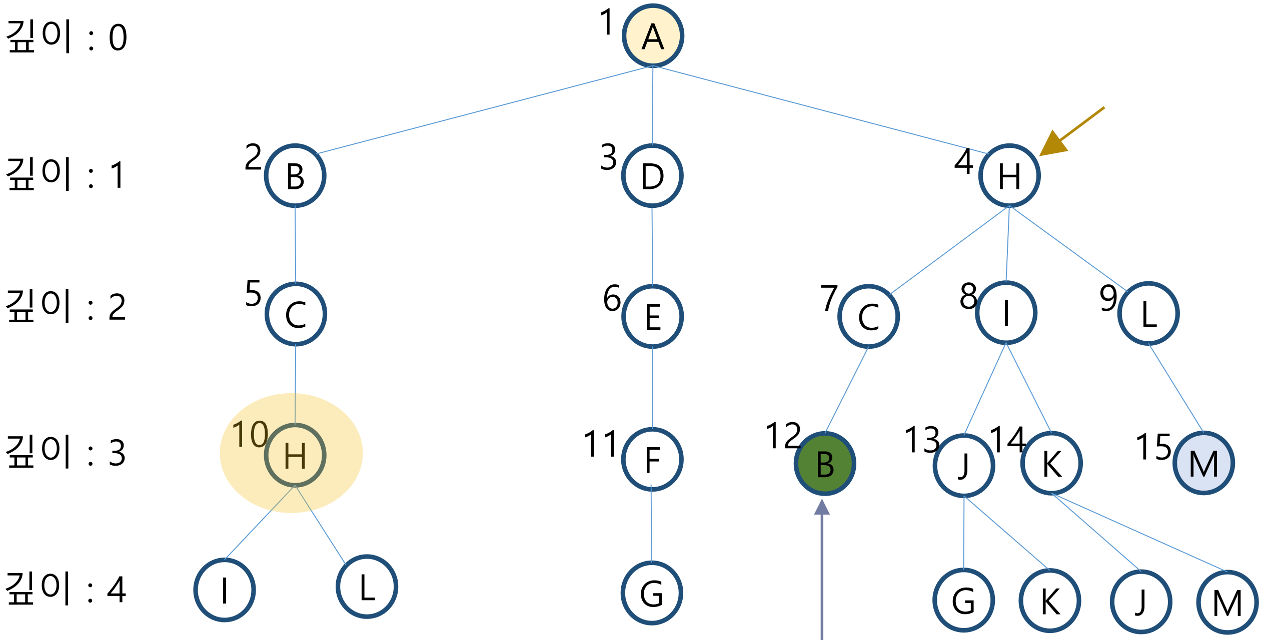
- 너비 우선 탐색의 노드 확장 순서
  - 깊이가 얕은 노드부터 확장, Queue로 구현 가능
  - 예 : 9번 노드가 목표 상태라면?
    - 1, 2, 3, 4, 5, 6, 7, 8, 9



너비 우선 탐색 : 최단 경로 찾기 문제 적용



- 최단 경로 찾기 문제에 적용한 결과
  - □가정 : 이미 지나온 상태 이동 X
  - □상태 확장 순서
    - A, B, D, H, C, E, C, I, L, H, F, B, J, K, M
  - □최종해 : A, H, L, M → 최적해?



12(B) : dead end(막다른 길)
더 이상 갈 곳이 없음

너비 우선 탐색 : 성능 평가 (1)

- 해의 도출 여부 : O
  - □시간이 충분하다면 해의 도출 보장
- 최적해의 도출 여부 : △
  - □깊이 기준 최적해 O
  - □거리 기준 최적해 X
- 시간 복잡도 : $O(b^d)$
  - □확장 노드의 개수 = $1 + b + b^2 + b^3 + \cdots + b^d$
    - 평균적으로 확장되어 나오는 노드의 개수(branching factor) : b
    - 해가 있는 깊이 : d
- 공간 복잡도 : $O(b^d)$
  - □모든 노드를 저장하고 있어야 노드 확장 가능

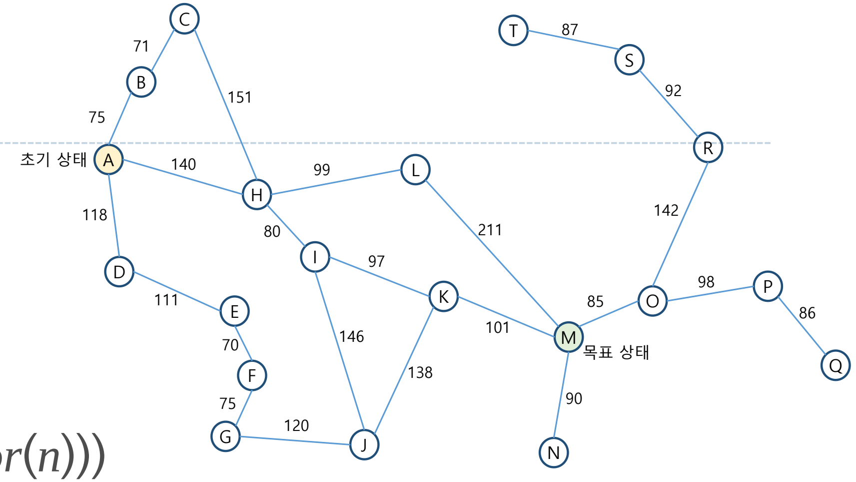
너비 우선 탐색 : 성능 평가 (2)

- 해가 존재하는 깊이에 따른 소요 시간 및 메모리
  - □branching factor : 10
  - □하나의 노드 평가 시간 : 1밀리초(ms)
  - □하나의 노드 메모리 : 100byte

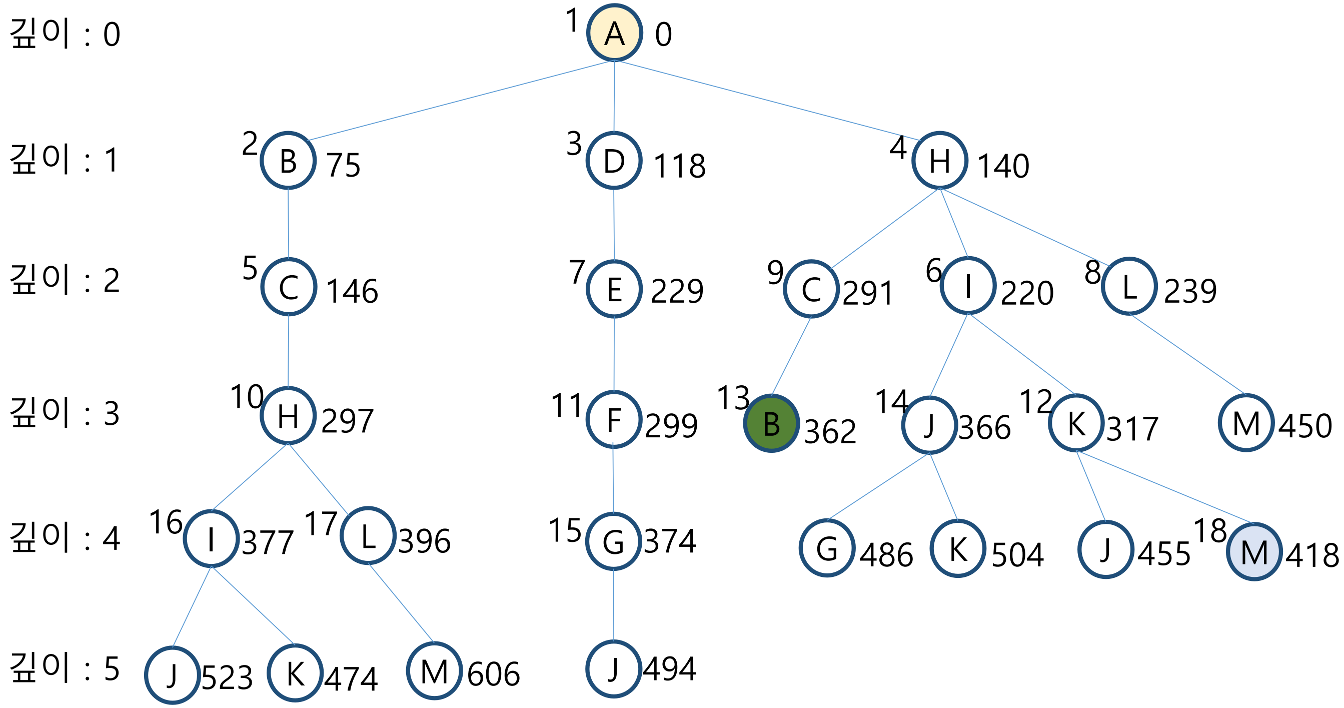
깊이	확장 노드수	소요 시간	소요 메모리
0	1	1 밀리초	100 B
2	111(≈10^2)	1 초	10 KB
4	11,111(≈10^4)	11 초	1 MB
6	10^6	18 분	100 MB
8	10^8	31 시간	10 GB
10	10^{10}	128 일	1 TB
12	10^{12}	35 년	100 TB
14	10^{14}	3500 년	10 PB

- [프로그램 2.1] 최단 경로 찾기 문제를 위한 너비 우선 탐색

균일 비용 탐색 (Uniform Cost Search)



- 경로 비용이 가장 적은 노드를 먼저 확장
  - 경로 비용 측면에서 최적해의 도출 보장
    - 단, 노드 n의 비용 값($g(n)$) 〈 n 노드 확장 이후의 경로 비용 값($g(operator(n))$)
      - 실세계 문제 대부분은 이 조건을 만족 : 최단 경로 찾기 문제 등
- 최단 경로 찾기 문제 적용 결과

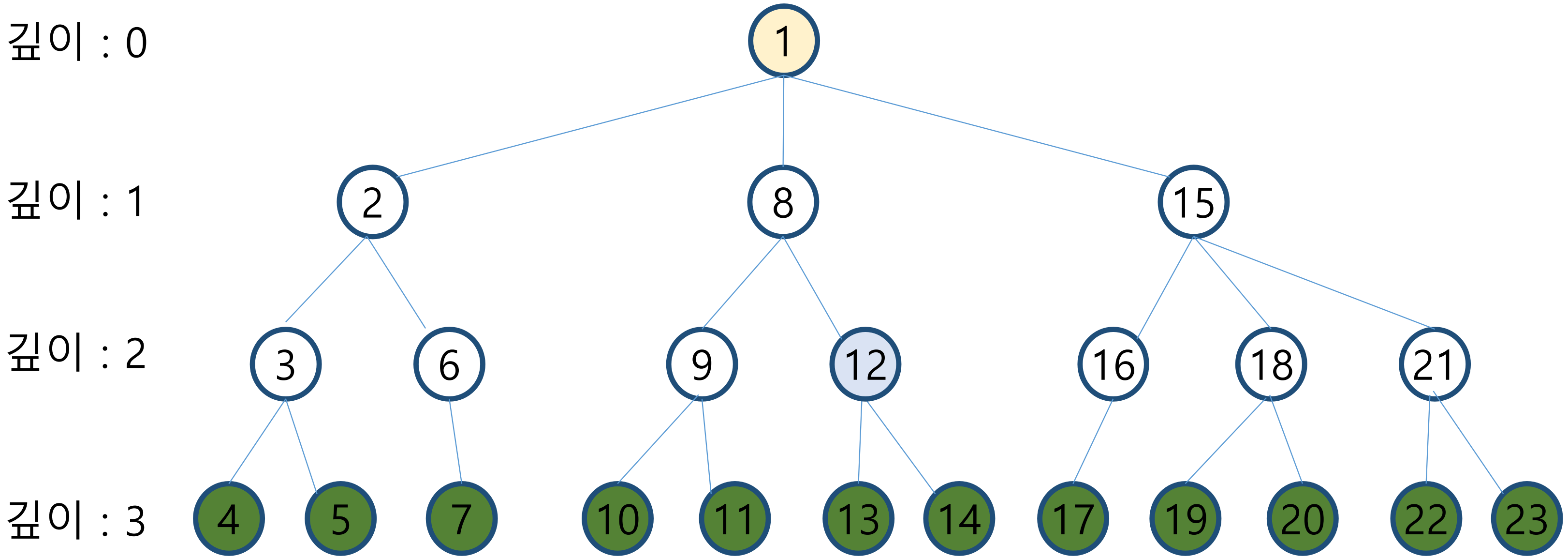


균일 비용 탐색 : 성능 평가

- 해의 도출 여부 : O
  - 시간이 충분하다면 해의 도출 보장
- 최적해의 도출 여부 : O
  - 거리 기준 최적해 O
  - 깊이 기준 최적해 X
- 시간 복잡도 : $O(b^d)$
  - 확장 노드의 개수 = $1 + b + b^2 + b^3 + \cdots + b^d$
    - 평균적으로 확장되어 나오는 노드의 개수(branching factor) : b
    - 해가 있는 깊이 : d
- 공간 복잡도 : $O(b^d)$
  - 모든 노드를 저장하고 있어야 노드 확장 가능
- <u>[프로그램 2.2] 최단 경로 찾기 문제를 위한 균일 비용 탐색</u>

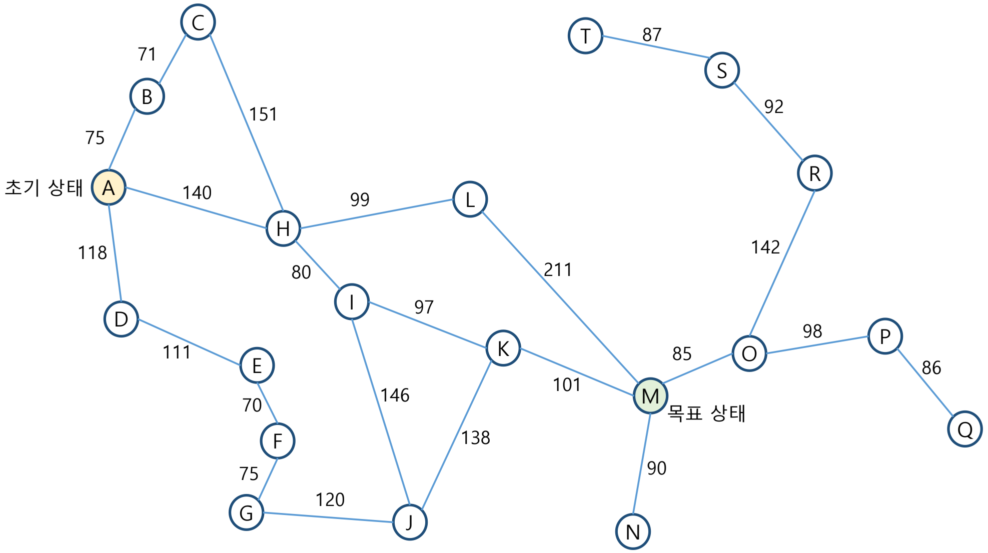
깊이 우선 탐색 (Depth-first Search)

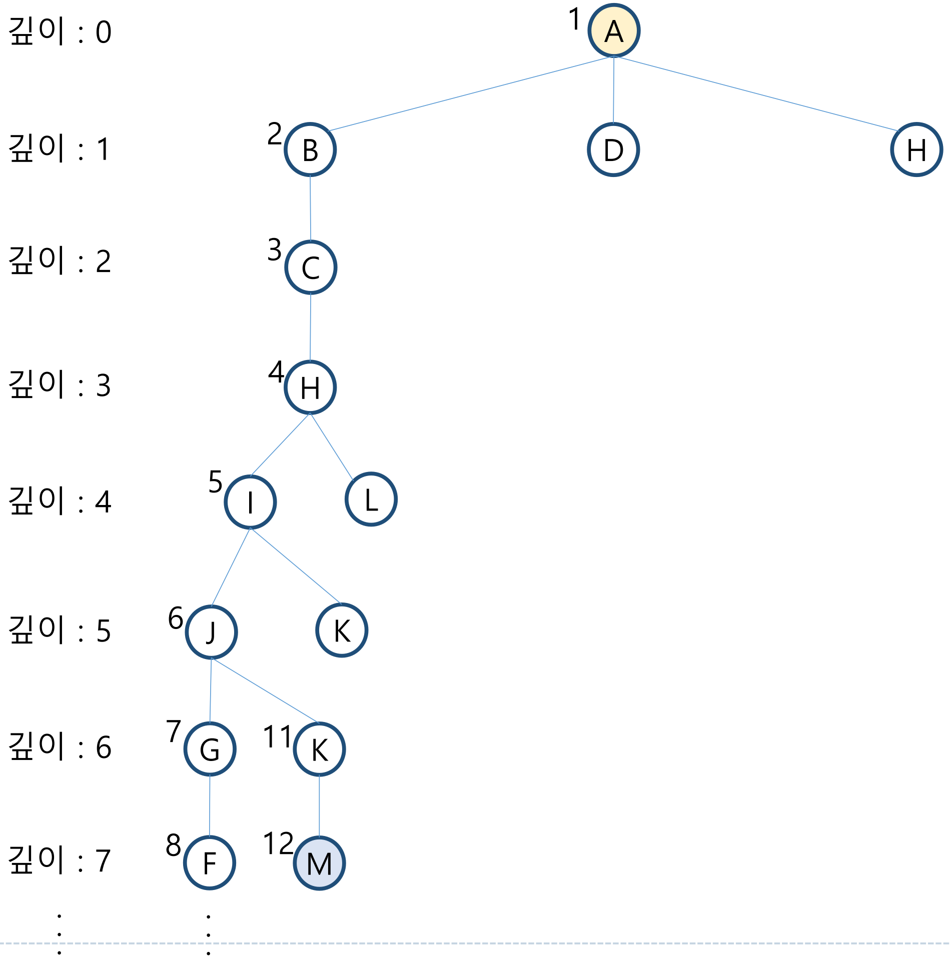
- 깊이 우선 탐색의 노드 확장 순서
  - 확장 중인 노드의 자식 노드부터 확장, Stack으로 구현 가능
  - 예 : 12번 노드가 목표 상태라면?
    - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12



깊이 우선 탐색 : 최단 경로 찾기 문제 적용

- 상태 확장 순서 : A, B, C, H, I, J, G, F, E, D, K, M
- 최종해 : A, B, C, H, I, J, K, M



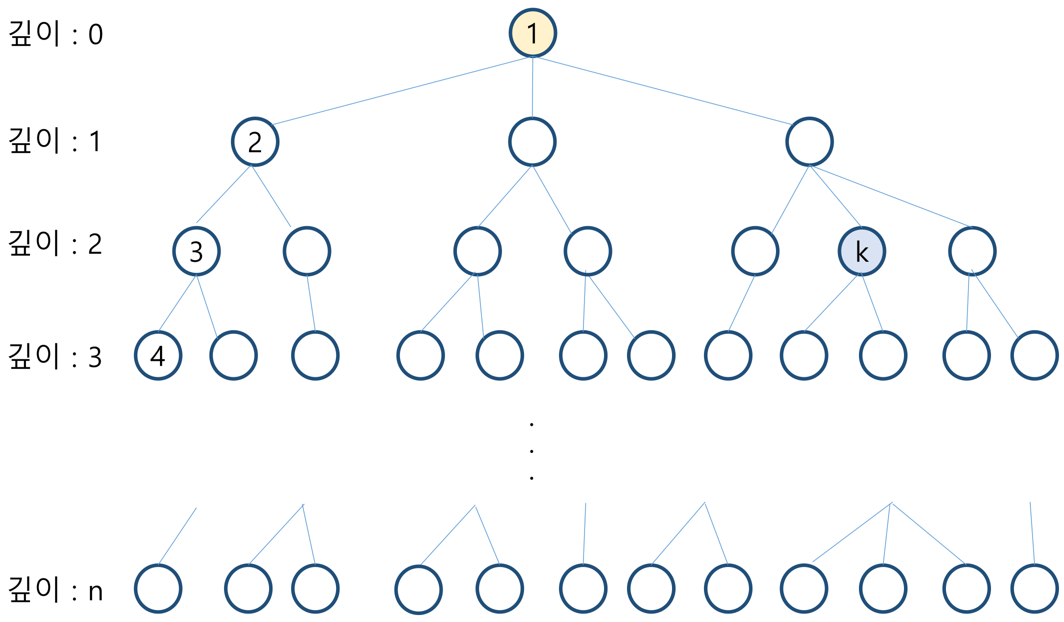


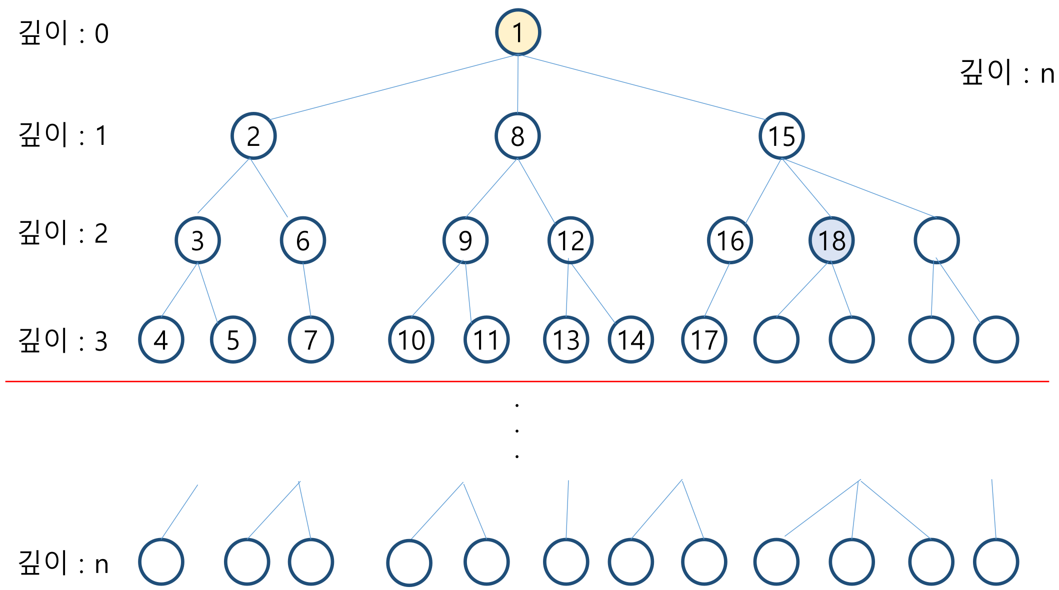
깊이 우선 탐색 : 성능 평가

- 해의 도출 여부 : X
  - 알고리즘 설계 시 주의 필요
  - 최단 경로 찾기 문제에서 지나온 경로 상의 상태로 다시 갈 수 있다고 가정한다? 무한 반복 상황 발생 가능
- 최적해의 도출 여부 : X
  - 거리 기준 최적해 X
  - 깊이 기준 최적해 X
- 시간 복잡도 : $O(b^m)$
  - m : 탐색 트리의 최대 깊이
- 공간 복잡도 : O(bm)
  - 한 순간에는 해당 깊이로의 노드들만 저장하면 됨
    - 이미 지나간 트리에는 해가 없다는 것을 확인 → 저장 필요 없음
- [프로그램 2.3] 최단 경로 찾기 문제를 위한 깊이 우선 탐색

깊이 제한 탐색 (Depth Limited Search)

- 해 노드 k가 다음과 같이 위치할 경우
  - 깊이 2까지만 확장하면 끝!
- 깊이 3까지 확장하는 것으로 제한한다면?
  - 깊이 제한 탐색

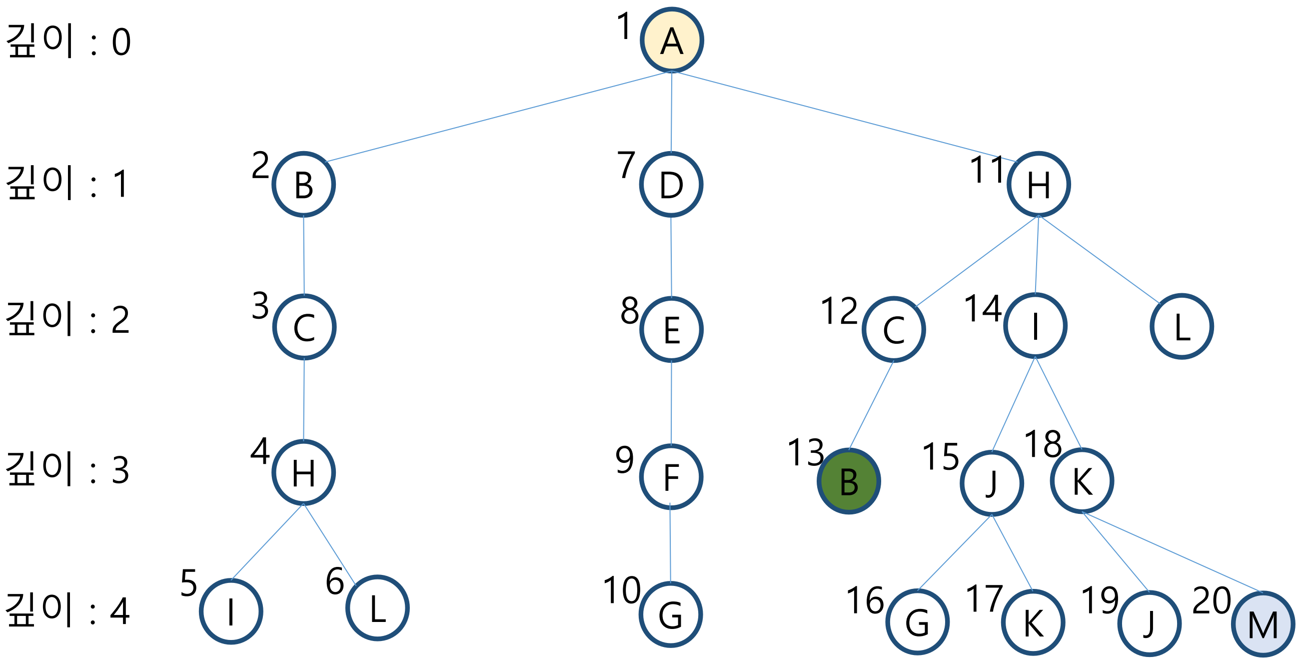




depth limit을 어떻게 설정할 것인가?

깊이 제한 탐색 : 최단 경로 찾기 문제 적용

- depth limit = 4
- 상태 확장 순서
  - A, B, C, H, I, L, D, E, F, G, H, C, B, I, J, G, K, K, J, M
- 최종해 : A, H, I, K, M

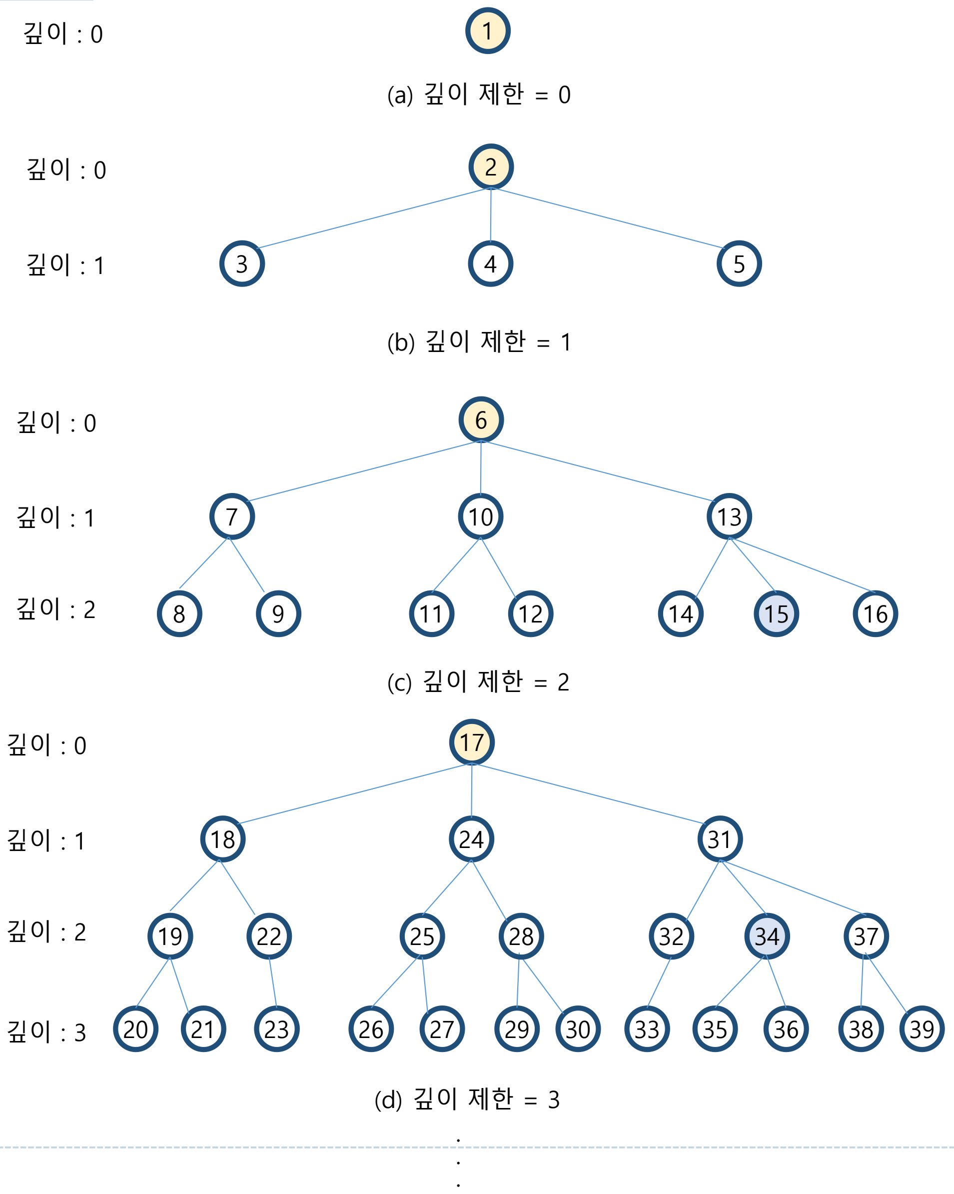


깊이 제한 탐색 : 성능 평가

- 해의 도출 여부 : O
  - 단, depth limit을 해의 깊이와 같거나 크게 설정해야 함
- 최적해의 도출 여부 : X
  - 거리 기준 최적해 X
  - 깊이 기준 최적해 X
- 시간 복잡도 : $O(b^l)$
  - l : depth limit
- 공간 복잡도 : $O(bl)$
  - 내부적으로 깊이 우선 탐색이 실행됨
- [프로그램 2.4] 최단 경로 찾기 문제를 위한 깊이 제한 탐색

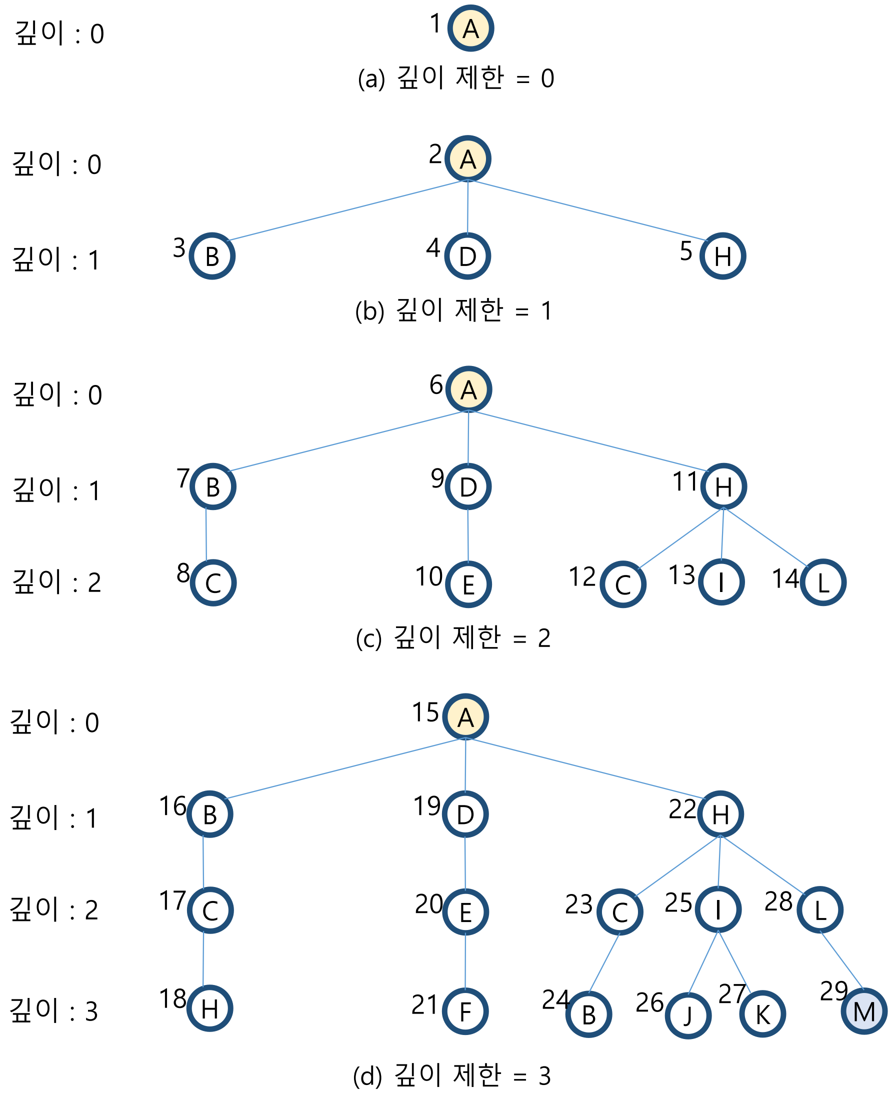
반복적 깊이 증가 탐색 (Iterative Deepening Search)

- 깊이 제한 탐색을 반복적으로 수행
  - depth limit 0, 1, 2, 3, 4, ...
    - 깊이 제한이 증가할 때마다 이전 트리 모두 삭제 후 다시 수행
  - 내부적으로 깊이 우선 탐색



반복적 깊이 증가 탐색 : 최단 경로 찾기 문제 적용

- 상태 확장 순서
  - (A), (A, B, D, H), (A, B, C, D, E, H, C, I, L), (A, B, C, H, D, E, F, H, C, B, I, J, K, L, M)
- 최종해 : A, H, L, M



(a) 깊이 제한 = 0

(b) 깊이 제한 = 1

(c) 깊이 제한 = 2

(d) 깊이 제한 = 3

반복적 깊이 증가 탐색 : 성능 평가 (1)

- 해의 도출 여부 : O
  - 너비 우선 탐색과 동일
- 최적해의 도출 여부 : △
  - 너비 우선 탐색과 동일
  - 깊이 기준 최적해 O
  - 거리 기준 최적해 X

반복적 깊이 증가 탐색 : 성능 평가 (2)

- 시간 복잡도 : $O(b^l)$
  - l : depth limit
  - 이전 depth limit의 노드들 모두 삭제 → 시간 훨씬 많이 소요?
    - 깊이 제한 탐색과 반복적 깊이 증가 탐색의 확장 노드 개수 비교
    - b = 10, l=5라면
      - 깊이 제한 탐색 : 111,111개
      - 반복적 깊이 증가 탐색 : 123,456개 (약 10% 증가)

$$1 + b + b^2 + b^3 + \cdots + b^{l-1} + b^l$$

$$(l+1)1 + (l)b + (l-1)b^2 + (l-2)b^3 + \cdots + (2)b^{l-1} + (1)b^l$$

- 공간 복잡도 : $O(bl)$
  - 내부적으로 깊이 우선 탐색이 실행됨
- [프로그램 2.5] 최단 경로 찾기 문제를 위한 반복적 깊이 증가 탐색

Uninformed Search 알고리즘 요약 및 비교

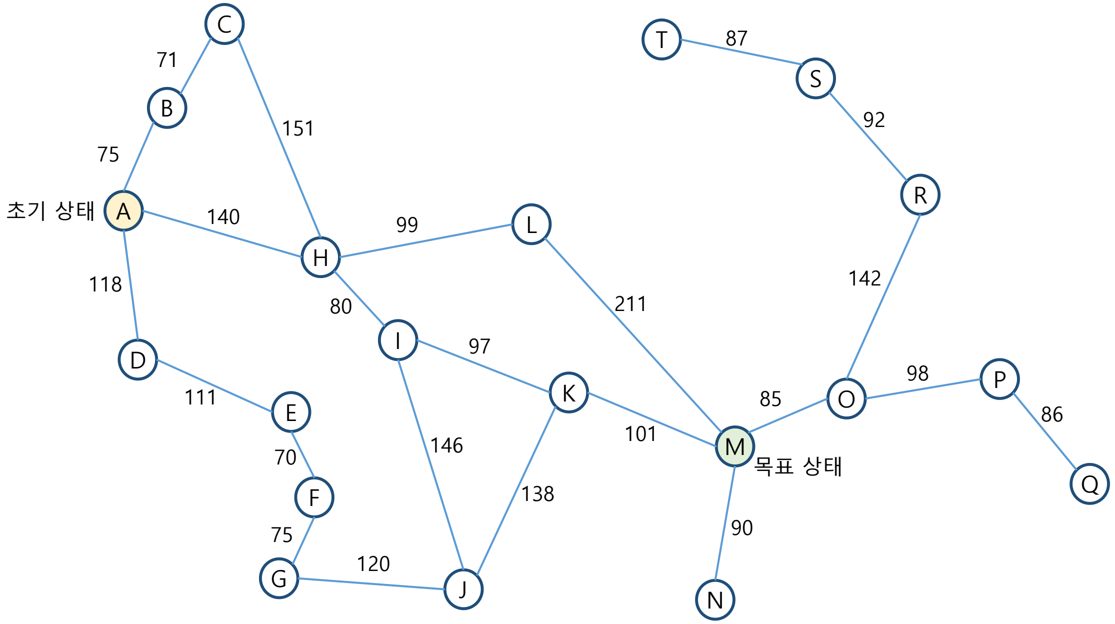
- b : branching factor
- d : 해가 위치하는 깊이
- m : 탐색 트리의 최대 깊이
- l : 깊이 제한 탐색 시 깊이 제한

	너비 우선 탐색	균일 비용 탐색	깊이 우선 탐색	깊이 제한 탐색	반복적 깊이 증가 탐색
해의 도출 여부 (Complete?)	Yes	Yes	No	Yes (if $d \leq l$)	Yes
최적해 도출 여부 (Optimal?)	Yes (깊이)	Yes (비용)	No	No	Yes (깊이)
시간 복잡도 (Time Complexity)	b^d	b^d	b^m	b^l	b^d
공간 복잡도 (Space Complexity)	b^d	b^d	bm	bl	bd

Informed Search

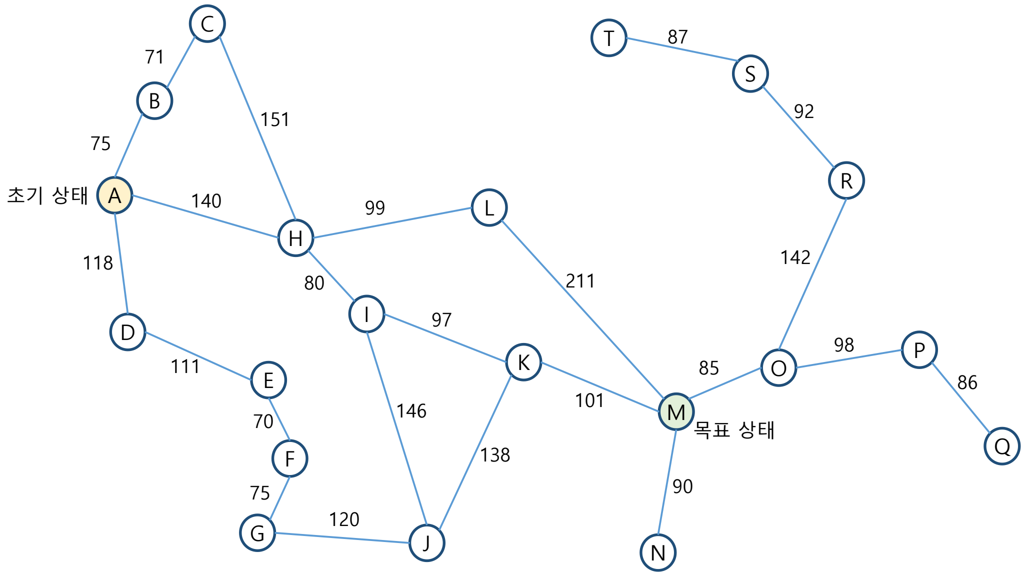
Informed Search : Introduction

- 확장 노드 선택 순서 : $f(n)$ 값이 가장 좋은(작은) 것부터
  - □ $f(n) = g(n) + h(n)$
    - $g(n)$: 초기 상태부터 현재 상태 n까지의 정보
      - 최단 경로 찾기 문제 : 실제 이동 거리
    - $h(n)$: 현재 상태 n부터 목표 상태까지의 정보 (휴리스틱)
      - 최단 경로 찾기 문제 : 추정 거리
- Uninformed Vs. Informed Search
  - □ Uninformed Search : $h(n) = 0$
  - □ Informed Search
    - Greedy Best-first Search : $g(n) = 0$
    - A* Search : $g(n)$, $h(n)$ 모두 사용



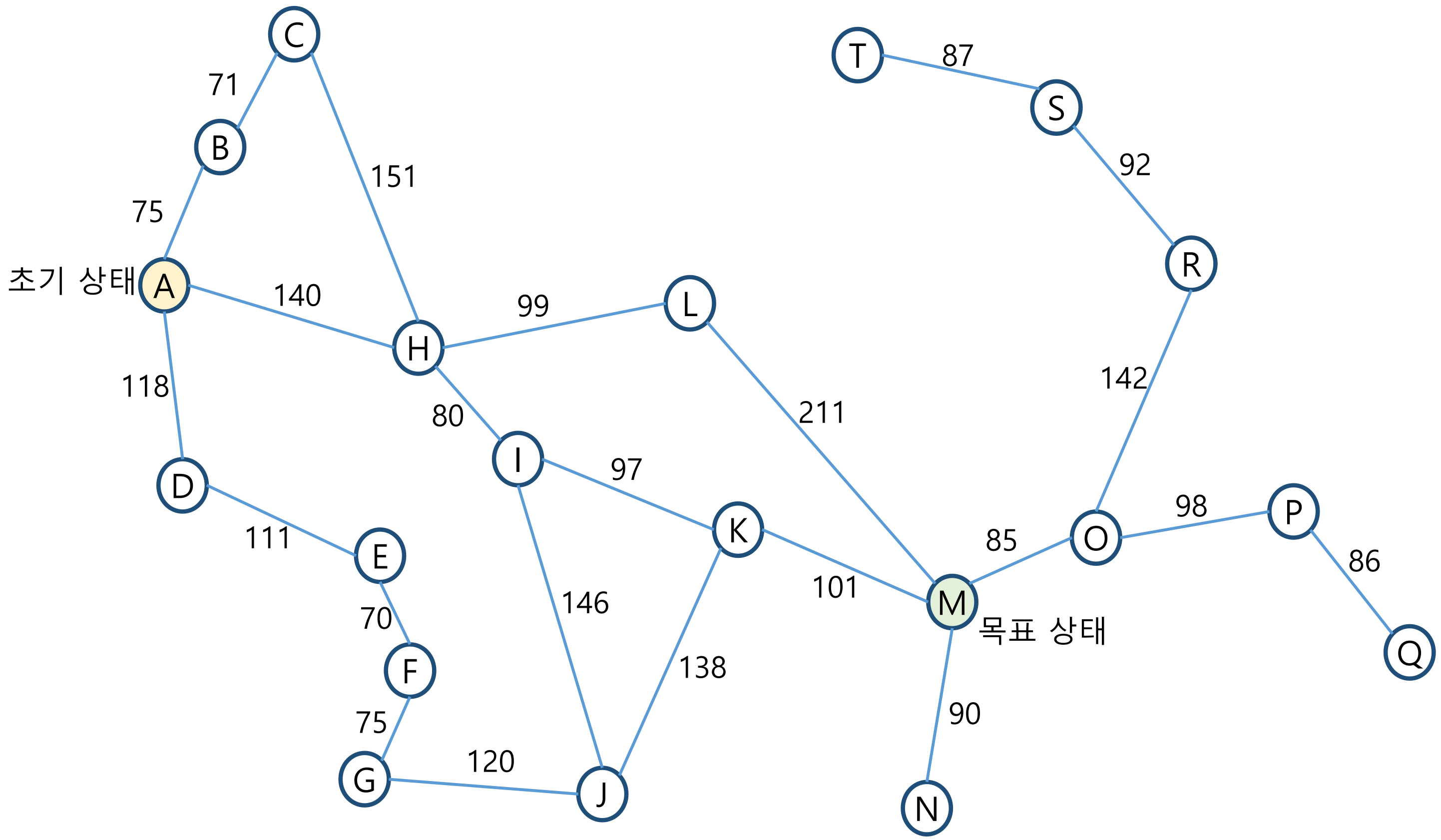
Informed Search : 휴리스틱 함수 h(n) (1)

- $h(n)$: 노드 n에서 목표 노드까지의 최저 비용에 대한 추정 비용
  - 예) 최단 경로 찾기 문제 : 노드 n의 도시에서 목표 도시까지의 최단 거리에 대한 추정 거리
    - 가장 좋은 상황 : 실제 최단 거리 반영 → 그러나 실제 최단 거리는 알 수 없음
  - 어떻게 추정할 것인가? → 문제에 대한 지식 활용
- 최단 경로 찾기 문제에서의 h(n)
  - A ~ M 거리(실제 최단 거리 418) :
    - 0으로 추정? 100으로? 200으로? ok
    - 500으로 추정? No!
      - 과대평가(overestimate)해서는 안됨
  - 과대평가하지 않으면서 가능한 최단 거리로 추정
    - "**직선 거리**" 사용 가능



Informed Search : 휴리스틱 함수 h(n) (2)

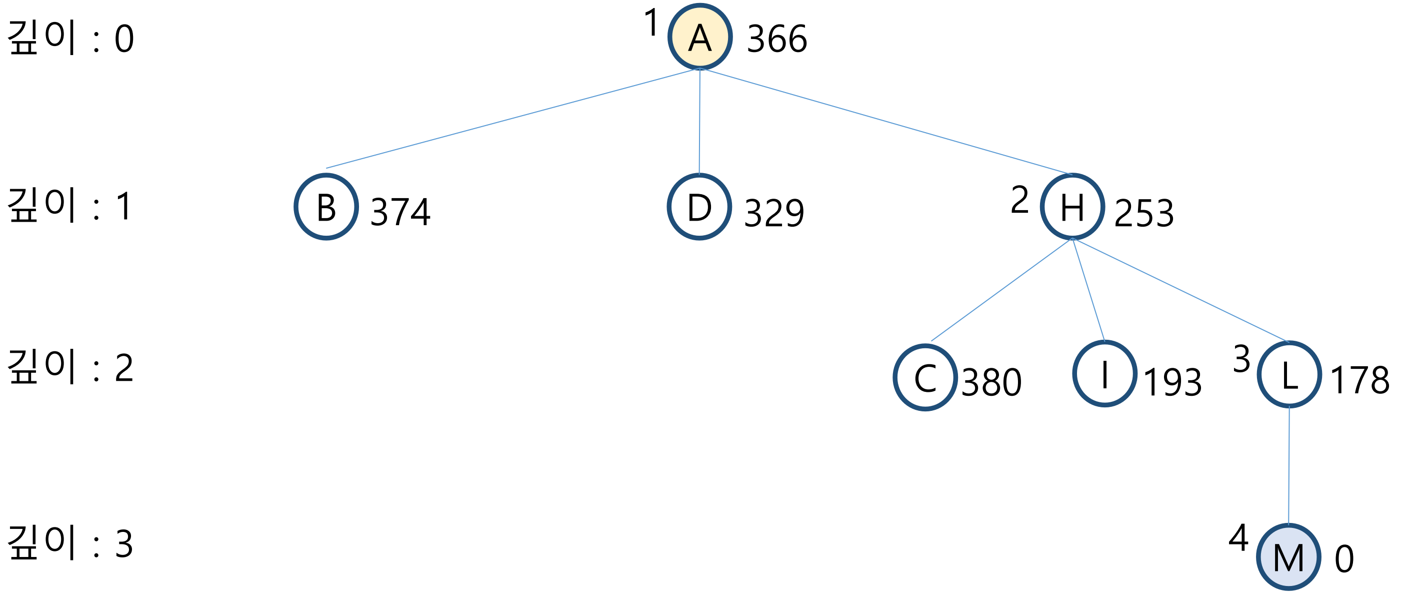
- 최단 경로 찾기 문제의 h(n)
  - 목표 도시 M까지의 직선 거리



도시	h(n)
A	366
B	374
C	380
D	329
E	244
F	241
G	242
H	253
I	193
J	160
K	98
L	178
M	0
N	77
O	80
P	151
Q	161
R	199
S	226
T	234

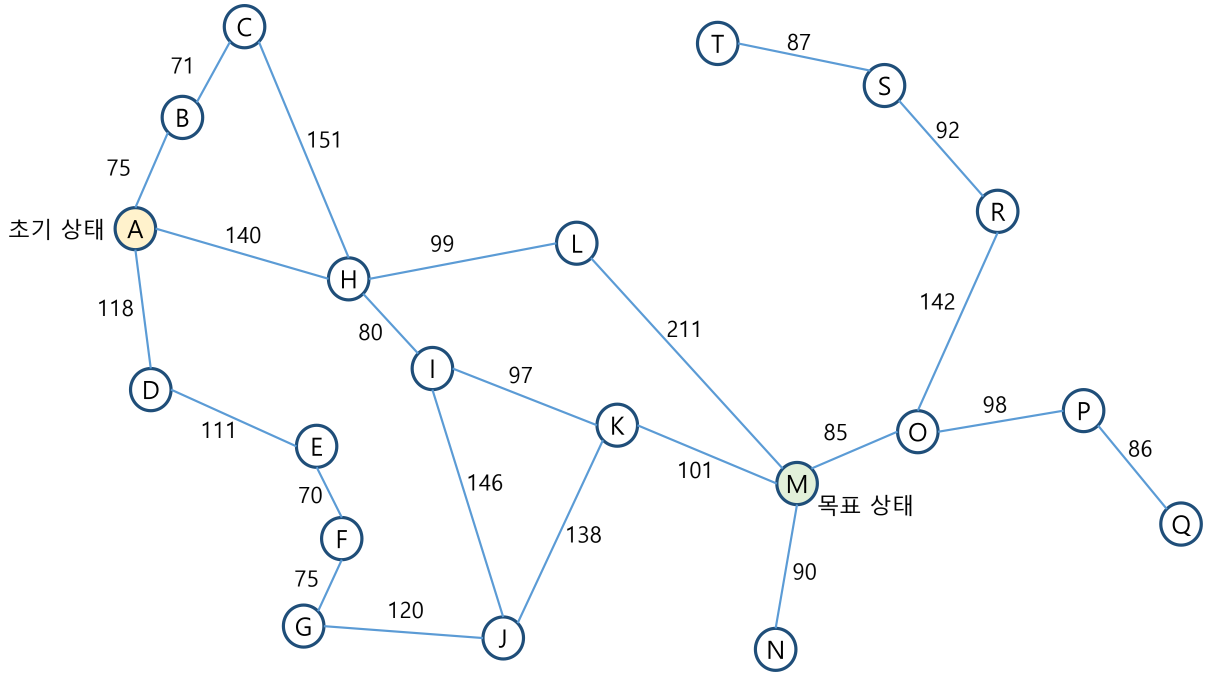
Greedy Best-first Search

- 목표 상태와 가장 가까운 노드를 먼저 확장
  - $f(n) = h(n)$
- 최단 경로 찾기 문제로의 적용
  - 노드 확장 순서 : A, H, L, M
  - 최종해 : A, H, L, M



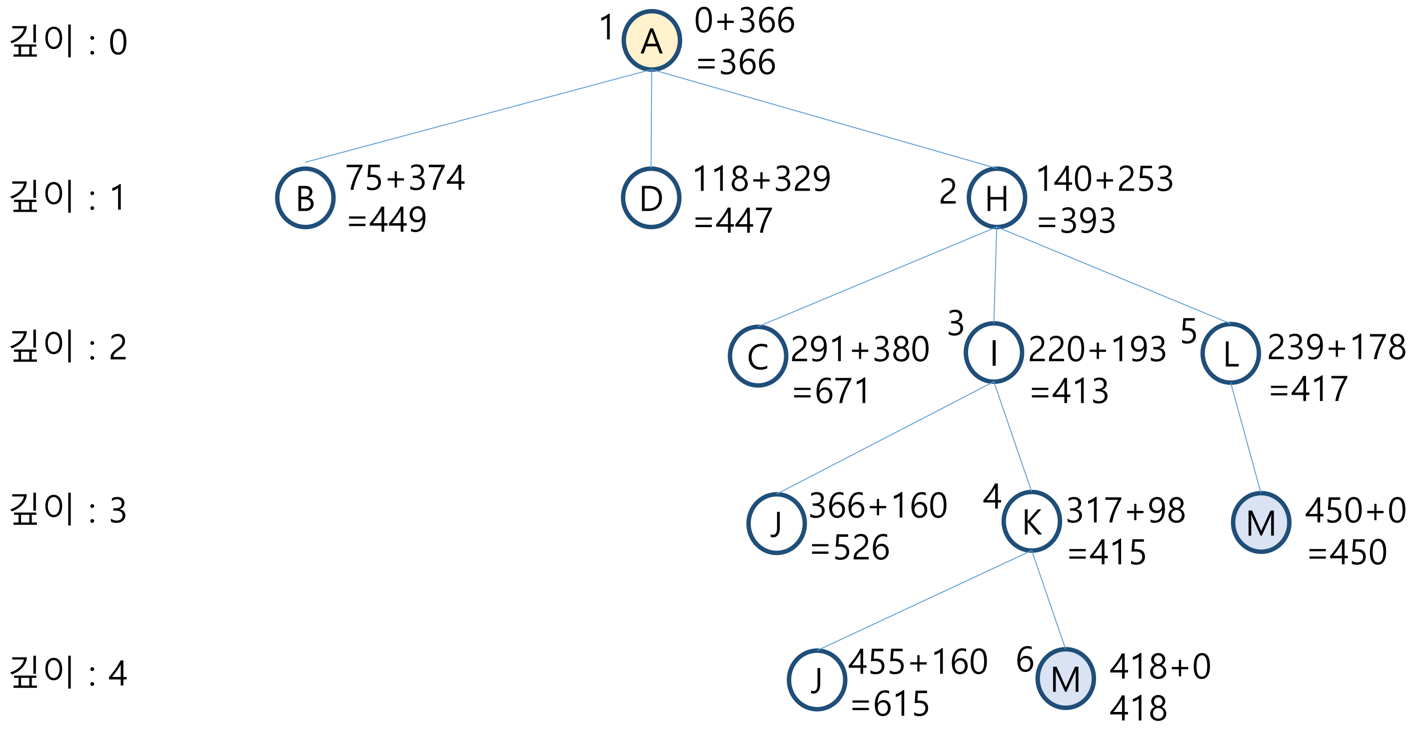
Greedy Best-first Search : 성능 평가

- 해의 도출 여부 : O
  - 다만, 이미 나왔던 상태에 대한 제어가 되지 않을 경우 보장 X
    - 예) 초기 도시 S, 목표 도시 C
- 최적해의 도출 여부 : X
- 시간 복잡도 : $O(b^m)$
  - m : 탐색 트리의 최대 깊이
  - dead end에서 백트랙 상황 발생 가능
- 공간 복잡도 : $O(b^m)$
  - 탐색 도중 언제 어떤 노드든 확장 가능
- [프로그램 2.6] 최단 경로 찾기 문제를 위한 Greedy Best-first 탐색



A* Search

- 초기 상태부터 현재 상태까지의 정보, 현재 상태에서 목표 상태까지의 정보 모두 사용
  - $f(n) = g(n) + h(n)$
- 최단 경로 찾기 문제로의 적용
  - 노드 확장 순서 : A, H, I, K, L, M
  - 최종해 : A, H, I, K, M



A* Search : 성능 평가

- 해의 도출 여부 : O
- 최적해의 도출 여부 : O
  - 단, h(n) : 과대평가되어서는 안됨. 직선 거리는 가능
  - h(n)이 항상 최단 거리로 추정한다면?
    - 탐색이 필요 없음. 최단 경로로만 확장

최단 경로 찾기 문제에서 f 값이 418 이하인 노드는 모두 확장됨.
결국 418 이하인 노드가 존재하지 않으며 최적해 보장

- 시간 복잡도 : $O(b^m)$
  - m : 탐색 트리의 최대 깊이
- 공간 복잡도 : $O(b^m)$
  - 탐색 도중 언제 어떤 노드든 확장 가능
- [프로그램 2.7] 최단 경로 찾기 문제를 위한 A* Search

휴리스틱 함수 h(n)의 중요성 (1)

- 최단 경로 찾기 문제 Vs. 8-puzzle 문제

	최단 경로 찾기 문제	8-puzzle 문제
상태	도시	현재 퍼즐 모양
연산자	다른 도시로의 이동	공백 타일 이동 (동, 서, 남, 북)
깊이	도시 이동 시 마다 1 증가	공백 타일 이동 시 마다 1 증가
경로 비용	초기 도시로부터 현재 도시까지의 이동 거리	초기 상태부터 현재 상태까지 공백 타일 이동 횟수
함수 f	g(경로 비용) + h(직선 거리)	g(경로 비용) + h(?)

1		5
4	3	2
7	8	6

초기 상태

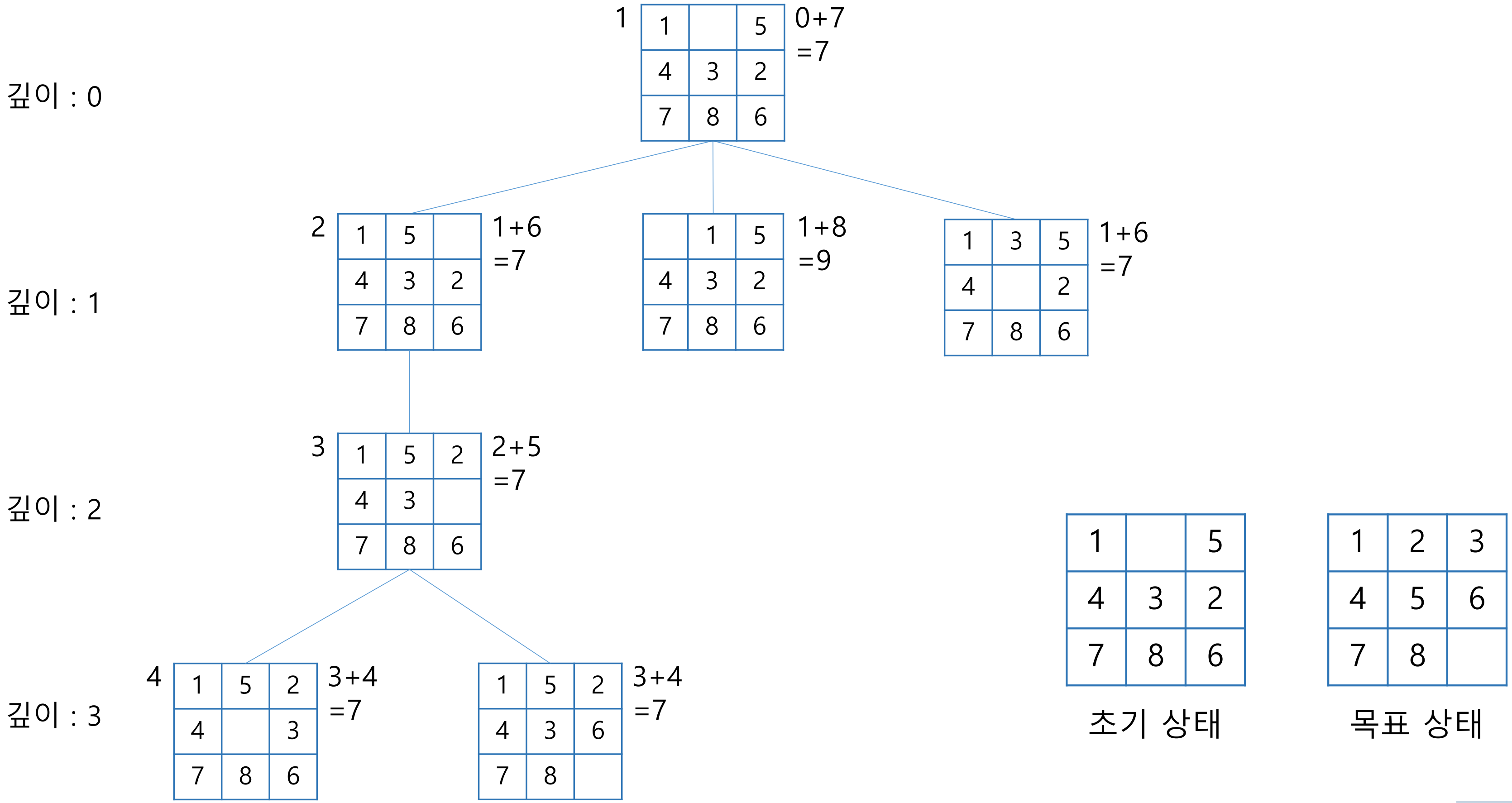
1	2	3
4	5	6
7	8	

목표 상태

- 8-puzzle 문제의 휴리스틱 함수 h(n)
  - h1: 목표 상태와 동일한 위치에 있지 않은 숫자 타일의 개수
  - h2: 각 숫자 타일이 다른 숫자와 관계없이 목표 상태에 있는 자신 위치로 이동하기 위한 최소 이동 횟수 합

휴리스틱 함수 h(n)의 중요성 (2)

- h2 기준 노드 확장 순서
  - g(n) : 지금까지의 공백 타일 이동 횟수



휴리스틱 함수 h(n)의 중요성 (3)

- h1과 h2에 대한 실험 결과
  - 해당 깊이에 최적해가 있는 경우 각각 노드 확장 개수

깊이 \ 탐색 알고리즘	반복적 깊이 증가 탐색	A* Search (h1)	A* Search (h2)
2	10	6	6
4	112	13	12
5	680	20	18
6	6384	39	25
10	47127	93	39
12	364404	227	73
14	3473941	539	113
16	-	1301	211
18	-	3056	363
20	-	7276	676
22	-	18094	1219
24	-	39135	1641

요약

- 트리 탐색은 초기 상태를 시작으로 목표 상태를 찾을 때까지 트리를 만드는 과정이며 노드 확장(expand) 전략에 따라 다양한 탐색 기법이 가능
- 탐색 알고리즘의 성능 평가를 위해 해의 도출 여부, 최적해 도출 여부, 시간 복잡도, 공간 복잡도에 대한 확인 필요
- 트리 탐색은 uninformed search와 informed search로 구분
- Uninformed search에는 breadth-first search, depth-first search, iterative deepening search 등이 있음
- Informed search의 하나인 A* search에서는 현재 노드까지의 평가값(g(n))뿐만 아니라 현재 노드에서 목표 노드까지의 추정치(h(n))를 함께 사용
- h(n)은 과대평가되어서는 안되며 실제 최적값과 가까울 수록 더 빨리 최적해를 찾을 수 있음