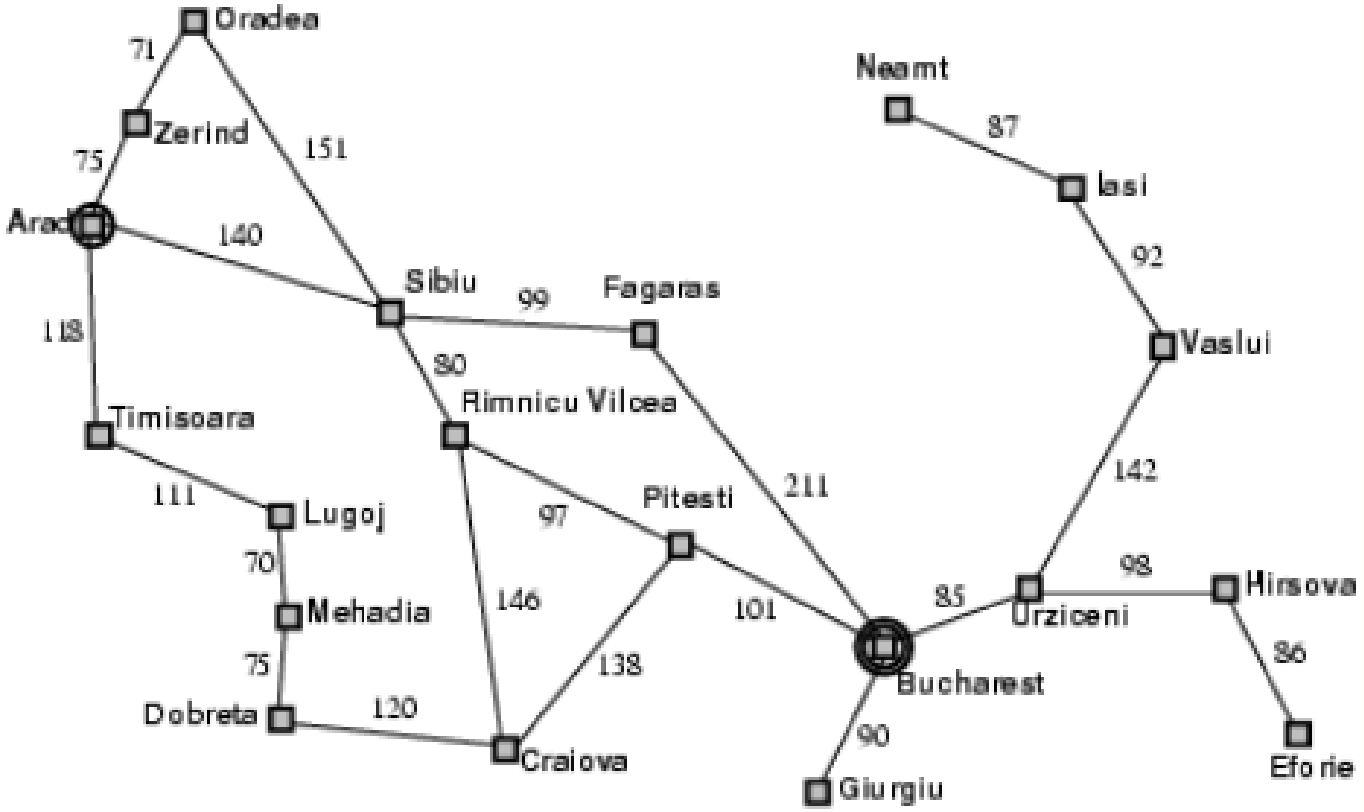
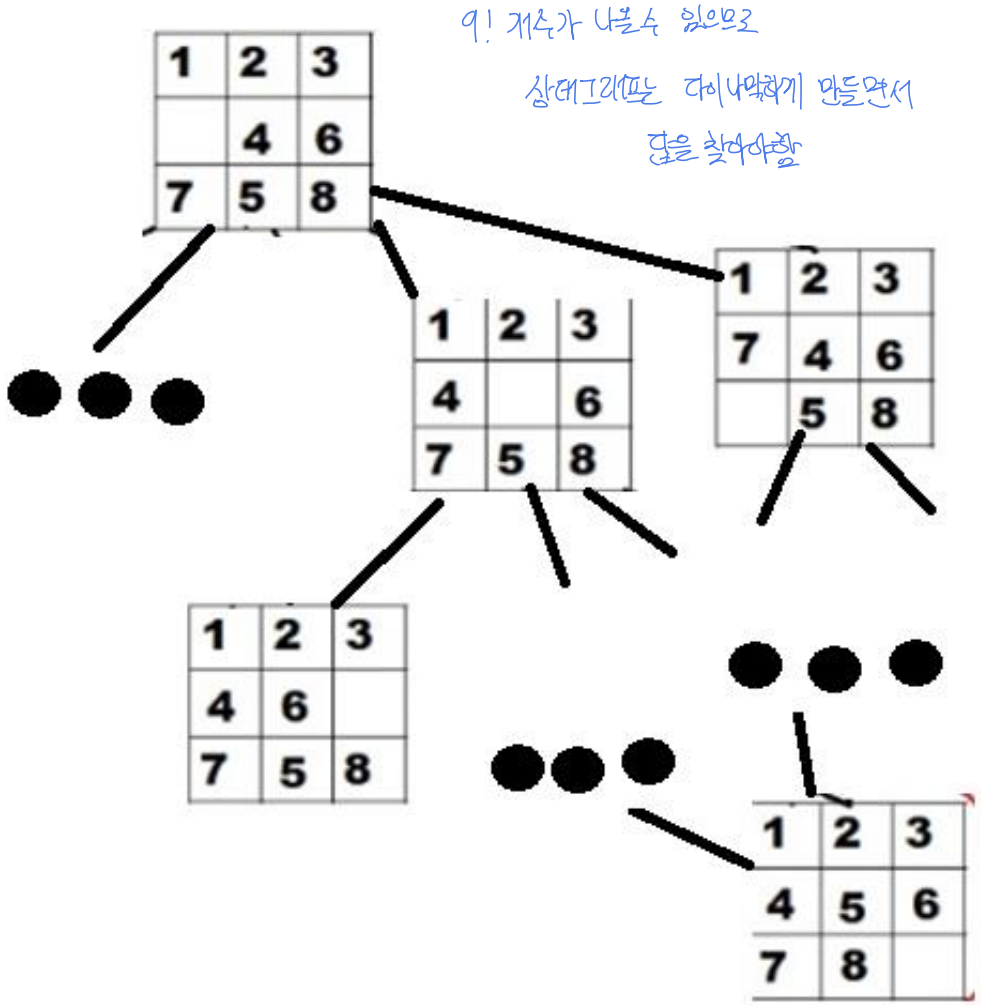
# A* algorithm

# A* Search Algorithm

- A* 알고리즘 이란?
  - 출발지에서 목적지까지 가는 최단 경로를 찾는 알고리즘
  - 구현이 간단하며 효과적인 길찾기 알고리즘
  - 대부분의 길찾기 알고리즘이 A*를 기반으로 만들어짐
  - point-to-point 길찾기를 위해 디자인되어짐
    - 다익스트라 알고리즘과 달리 가장 짧은 길이 아닐 수 있음

  - **(Definition from Wikipedia) A***(pronounced as "A star") is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently directed path between multiple points, called "nodes".


Oradea
71
Neamt
Zerind
87
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Pitesti
211
Lugoj
97
98
70
Hirsova
Mehadia
146
101
85
Urziceni
75
138
Bucharest
86
Dobreta
120
90
Craiova
Efo rie
Giurgiu




node as a state

- A* 알고리즘은 출발 꼭짓점(start node)으로부터 목표 꼭짓점(goal node)까지의 최적 경로를 탐색하기 위한 것이다. 이를 위해서는 각각의 꼭짓점(node)에 대한 평가 함수를 정의해야 한다.
- node n에 대한 평가 함수 f(n)은:

$$f(n) = g(n) + h(n)$$

- $g(n)$ : 출발 꼭짓점으로부터 꼭짓점 $n$까지의 경로 가중치
- $h(n)$ : 꼭짓점 $n$으로부터 목표 꼭짓점까지의 추정 경로 가중치

**경로 가중치**: 경로(path)를 구성하는 모든 edge(link)들의 가중치 합(sum of edge weights)

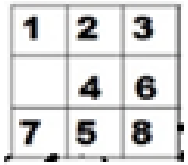
# 용어 설명

- Open node
  - 방문(visted)은 했지만 아직 처리(expansion)는 하지 않음
  - 진행되어질 수 있는 후보
- Closed node
  - 이미 처리(expansion)를 한 것
  - 이미 지나왔거나 살펴보지 않아도 되는 노드
- Unvisited node
  - 아직 살펴보지 않은 노드

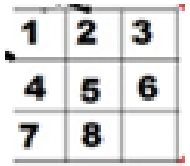
# 용어 설명

- Heuristic
  - Rule of thumb (엄지의 법칙. 이론적 아니라 경험 등에 입각한 법칙)
  - 의사결정 과정을 단순화하여 빠른 시간 안에 답을 찾는 것
  - A*는 이동할 때마다 최단거리가 '될 것 같은' 지점을 선택
  - '될 것 같은' 지점을 선택하는데 heuristic 사용
  - heuristic 계산이 정확하면 효과적이고 그렇지 않으면 Dijstra's Algorithm보다 성능이 안좋음

- 8-Puzzle Problem Example




start node

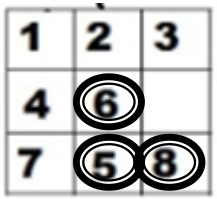



goal node

# 8-puzzle problem에서 h(n) 예제

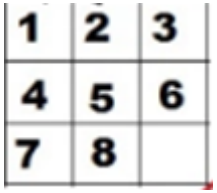
- h(n) = number of misplaced tiles in node n.

ex)




node n

h(n) = 3

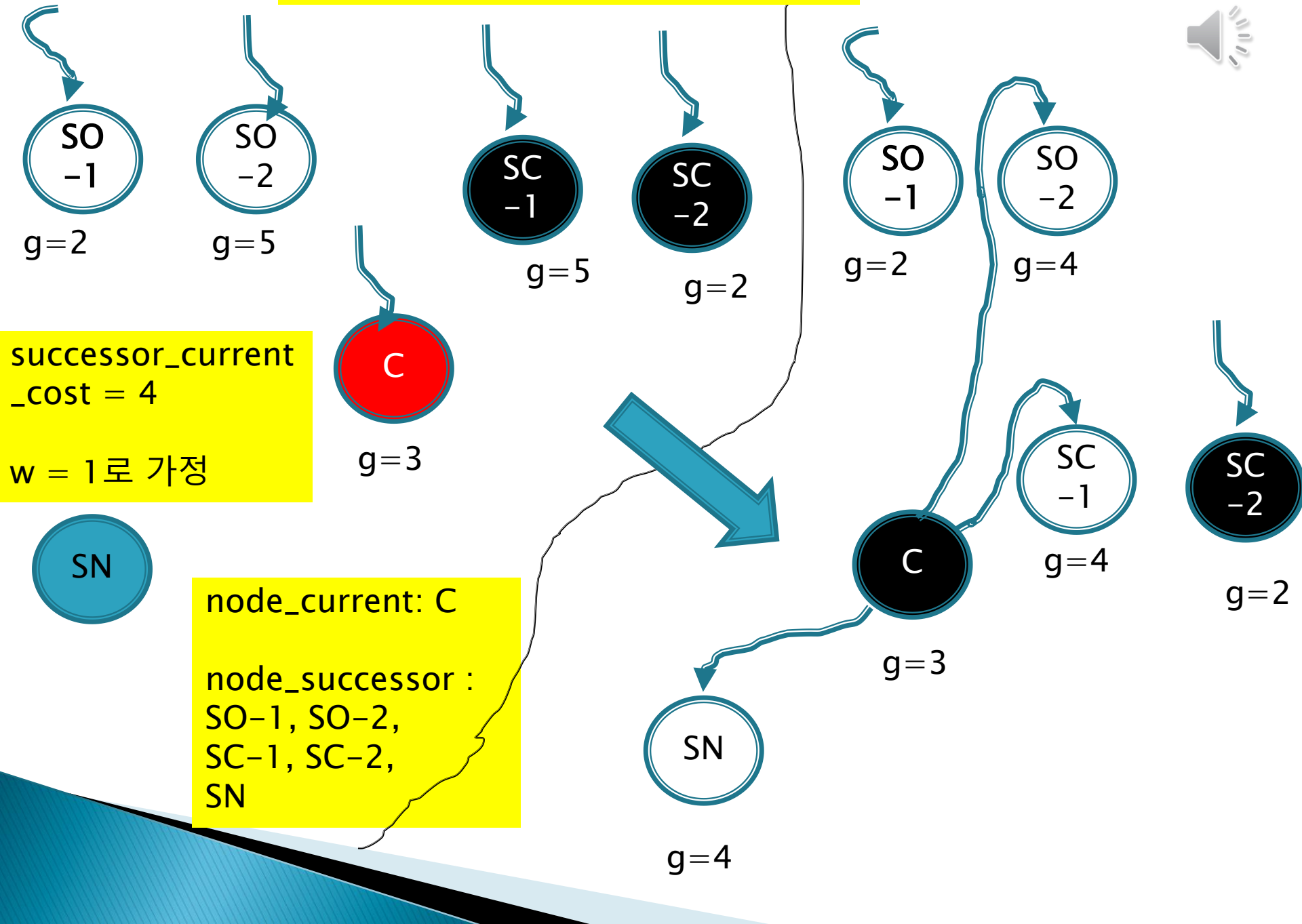



goal node

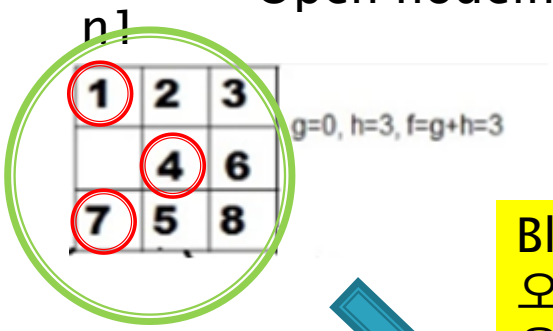
# A* Algorithm: search a good path from node_start to node_goal

```
1  Put node_start in the OPEN list with f(node_start) = h(node_start) (initialization)
2  while the OPEN list is not empty {
3    Take from the open list the node node_current with the lowest
4         f(node_current) = g(node_current) + h(node_current)
5    if node_current is node_goal we have found the solution; break
6    Generate each state node_successor that come after node_current
7    for each node_successor of node_current {
8      Set successor_current_cost = g(node_current) + w(node_current, node_successor)
9      if node_successor is in the OPEN list {
10       if g(node_successor) ≤ successor_current_cost continue (to line 20)
11     } else if node_successor is in the CLOSED list {
12       if g(node_successor) ≤ successor_current_cost continue (to line 20)
13       Move node_successor from the CLOSED list to the OPEN list
14     } else {
15       Add node_successor to the OPEN list
16       Set h(node_successor) to be the heuristic distance to node_goal
17     }
18     Set g(node_successor) = successor_current_cost
19     Set the parent of node_successor to node_current
20   }
21   Add node_current to the CLOSED list
22 }
23 if(node_current != node_goal) exit with error (the OPEN list is empty)
```

# Line 5 ~ Line 21 설명 (예제를 통해)



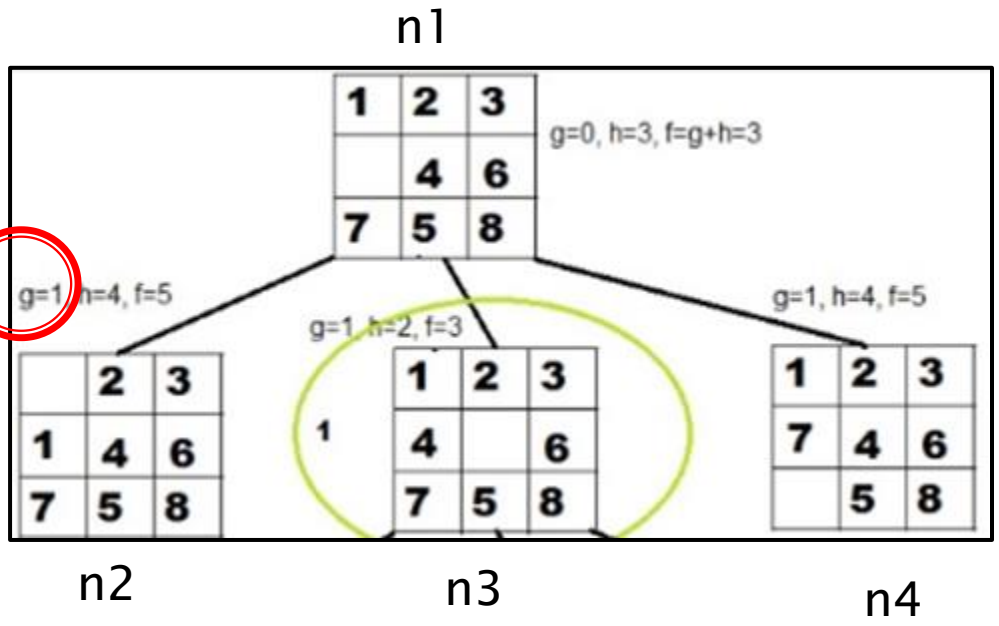
Open node:n1

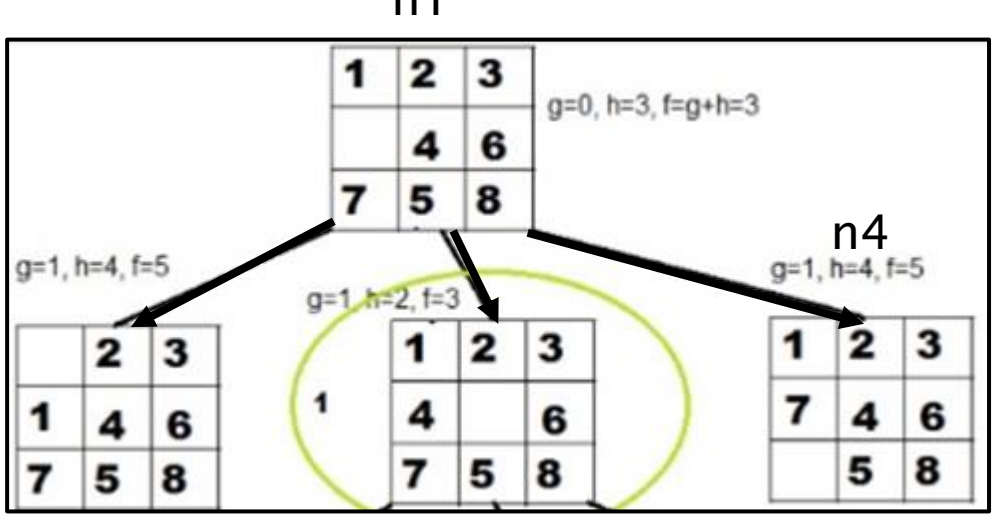



Blank(빈 칸)를 위로, 오른쪽으로, 아래로 움직일 수 있음

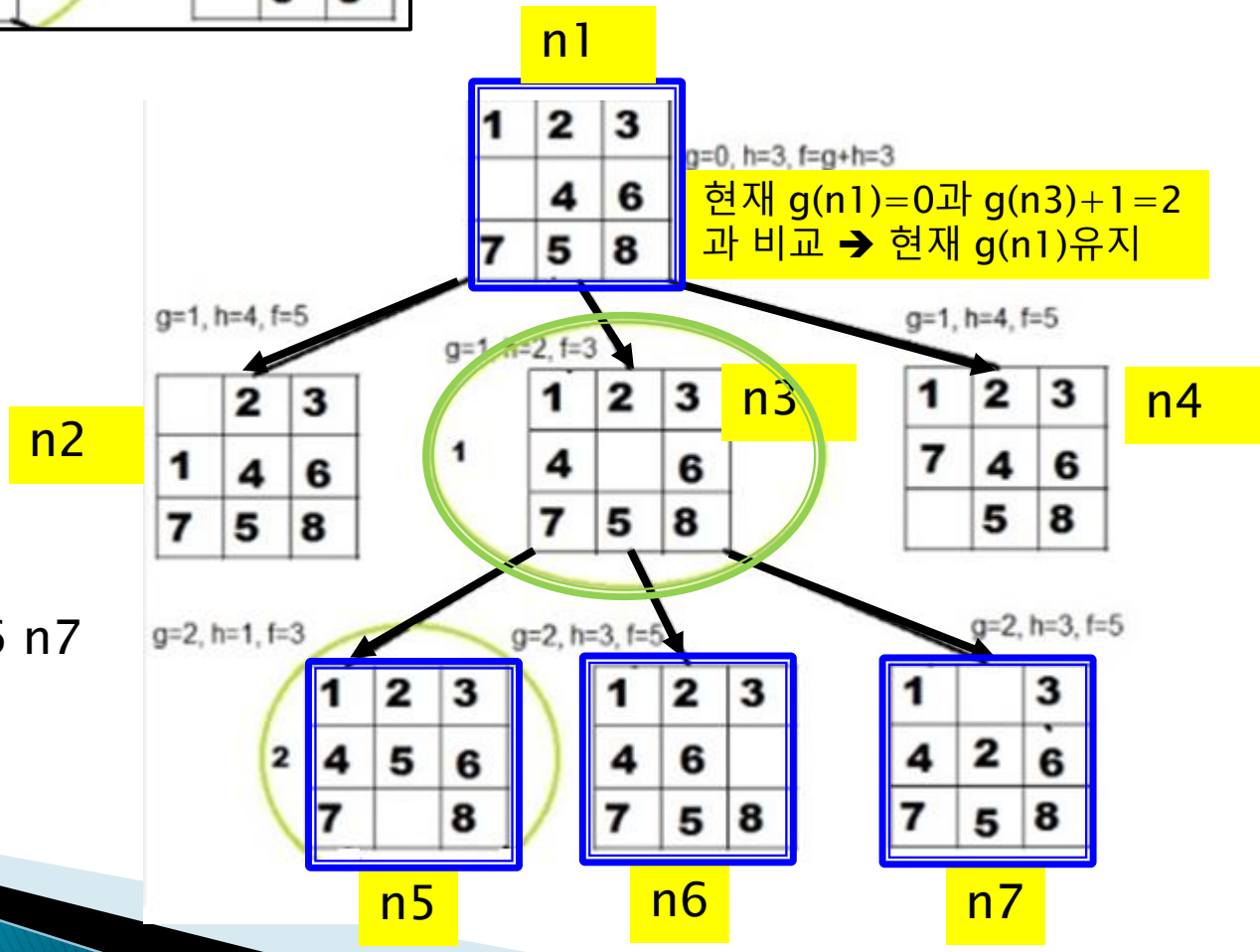
g(n2) = g(n1) + 1

Open node: n2 n3 n4
Close node: n1



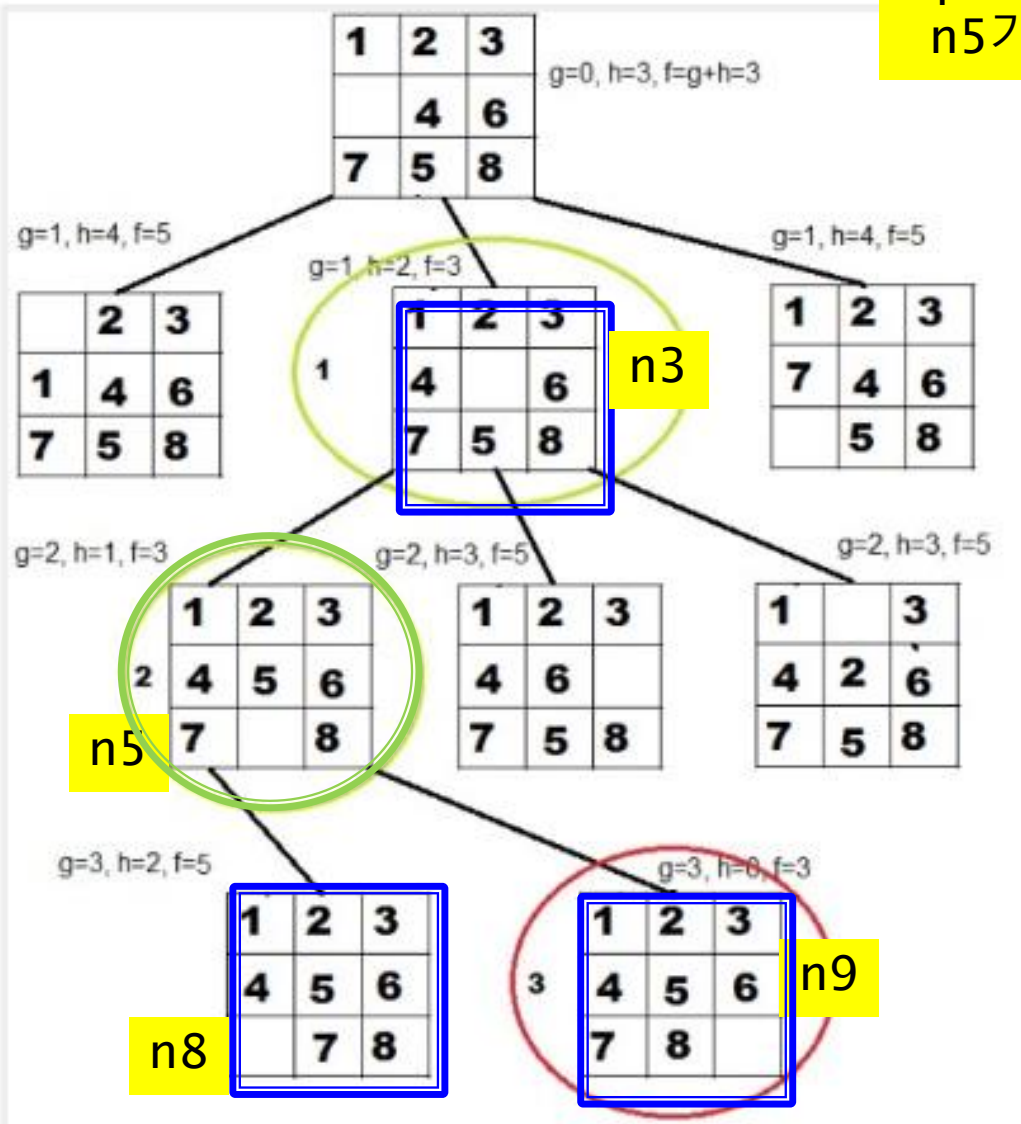



Open node: n2 n3 n4
Close node: n1



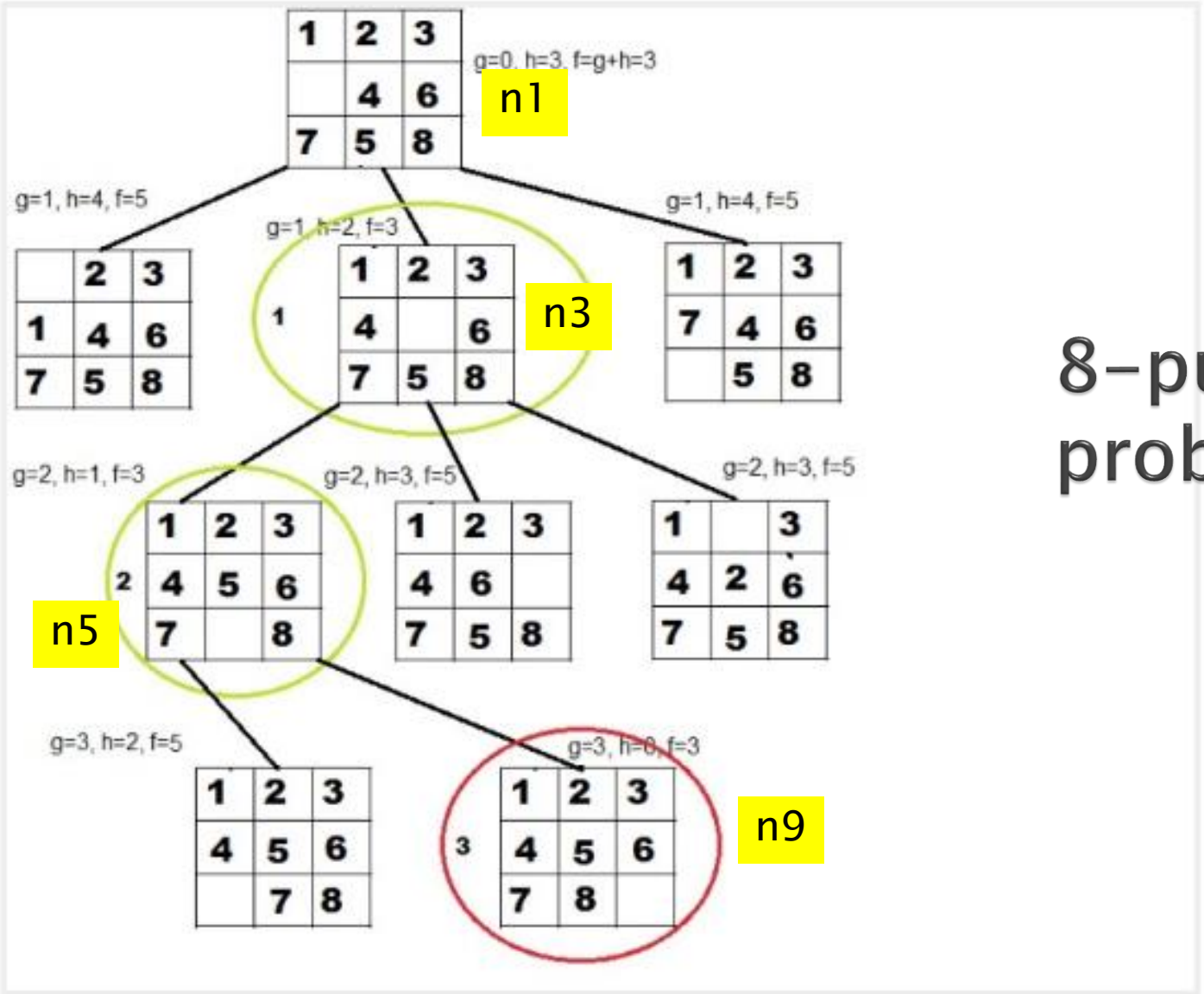

Open node: n2 n4 n5 n6 n7
Close node: n1 n3

Open node n2 n4 n5 n6 n7 중에서
n5가 선택됨




# 8-puzzle problem

* 현재 g(n3)과 g(n5)+1과 비교 ➔ 현재 g(n3)유지




# 8-puzzle problem

# A* 처리

- 모든 노드는 최종 추정치를 가지게 됨
  - 최종 추정치 = 시작점 부터의 가중치 + 목적지 까지의 예상치(heuristic 값)
- 각 노드는 어디서 부터 진행되었는지를 기록해둠

- A*는 일정한 행동의 반복으로 진행됨
  - 현재 노드와 인접하고 unvisited인 노드는 open 노드가 되며 최종 추정치를 계산
  - open 노드 중 최종 추정치가 가장 낮은 노드로 진행
  - 선택된 노드는 closed 노드가 됨
    - 이 때 시작점 부터의 가중치를 다시 계산하며, 변경되었다면 open 노드가 됨

# A* 처리

- 최종적으로 goal에 도달 시 이전 노드로 되돌아 가면서 길 도출
  - ~~이전 예제에서는 EG – CE – AC 이므로 A-C-E-G가 길~~
  - 이전 예제에서는 n1 ➔ n3 ➔ n5 ➔ n9

# A* Algorithm 장단점

- A* 알고리즘의 장점
  - 상대적으로 빠른 시간안에 결과를 도출
- A* 알고리즘의 단점
  - 기본 A* 알고리즘만 사용시에 찾은 길이 최단거리가 아닐 수 있음
    - 검색을 시작하는 방향에 따라 결과가 바뀔 수 있음
- A* 사용 예시
  - 스타크래프트
    - 실제 이동할 수 없는 지형이라도 우선 이동 -> 정확성 보다는 속도가 중요하므로
  - 바둑 게임
    - 속도보다는 승부가 중요하므로 정확성이 높은 알고리즘 사용

음성 설명 없음

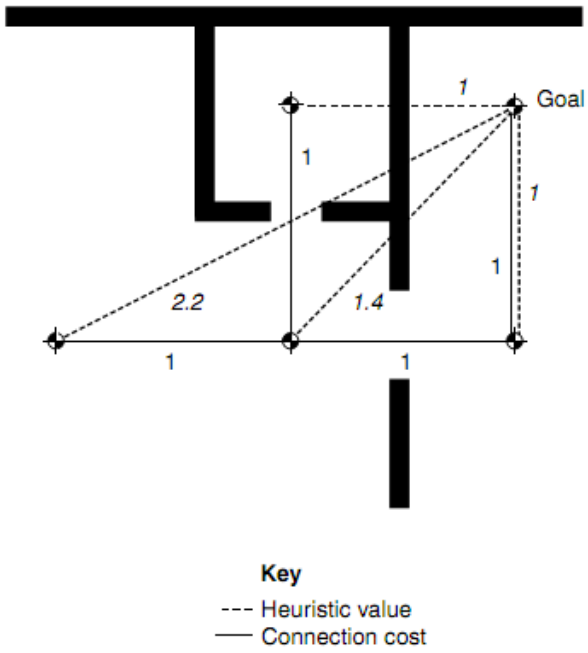
# Heuristic 선택

# Heuristic 선택

- 휴리스틱 값이 실제 거리보다 과소평가 된다면?
  (A heuristic function is said to be **admissible** if it never overestimates the cost of reaching the goal. A* gives an optimal solution if the heuristic is *admissable*)
  - 목표 지점 로 빨리 가기 보다는, 시작 노드 또는 선택된 노드 (node_current) 근처의 노드를 더욱 많이 검사하고, 더욱 많은 경우의 수를 검사하게 됨
    - 정확도는 증가하지만 시간도 증가
    - 만약 h(n)=0이면, 다익스트라 알고리즘과 같게 됨
- 휴리스틱 값이 실제 거리보다 과대평가 된다면?
  - 정확도는 떨어지지만 시간 감소
- 게임은 그렇게 정확하지 않아도 됨
  - 적당한 선에서 타협
    - 바둑 게임에서 한 수씩 둘 때 마다 모든 경우의 수를 계산한다면 몇 천년 이상의 시간이 소요됨

# Euclidean Distance Heuristic

- 목적지 까지의 거리를 h (즉 휴리스틱 함수) 값으로 사용
  - 장애물이 많은 곳에서 성능 저하
    - 시간, 정확도 모두
  - 장애물이 없을 경우  정확함

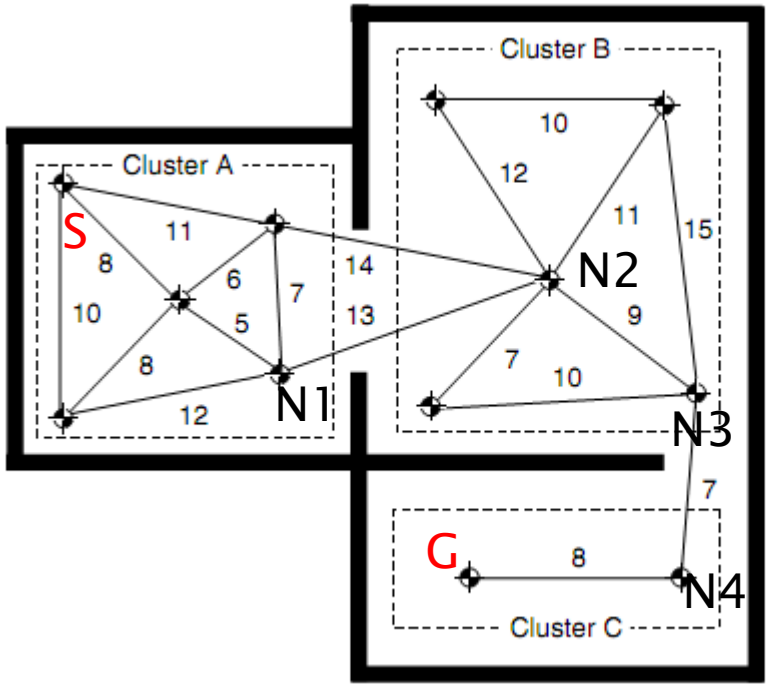


# Cluster Heuristic

- 클러스터링
  - 데이터의 성질이 유사한 것끼리 묶는 것

- 클러스터 휴리스틱
  - A* 에서는 비슷한 지역의 노드끼리 묶음
  - 그래프 클러스터링 알고리즘으로 자동으로 묶거나 레벨 디자인을 할 때 직접 묶음
  - Lookup Table
    - 각 클러스터들 사이의 가장 짧은 경로를 가지는 테이블
  - 복잡한 곳에서 Euclidean Distance보다 좋은 성능

# Cluster Heuristic

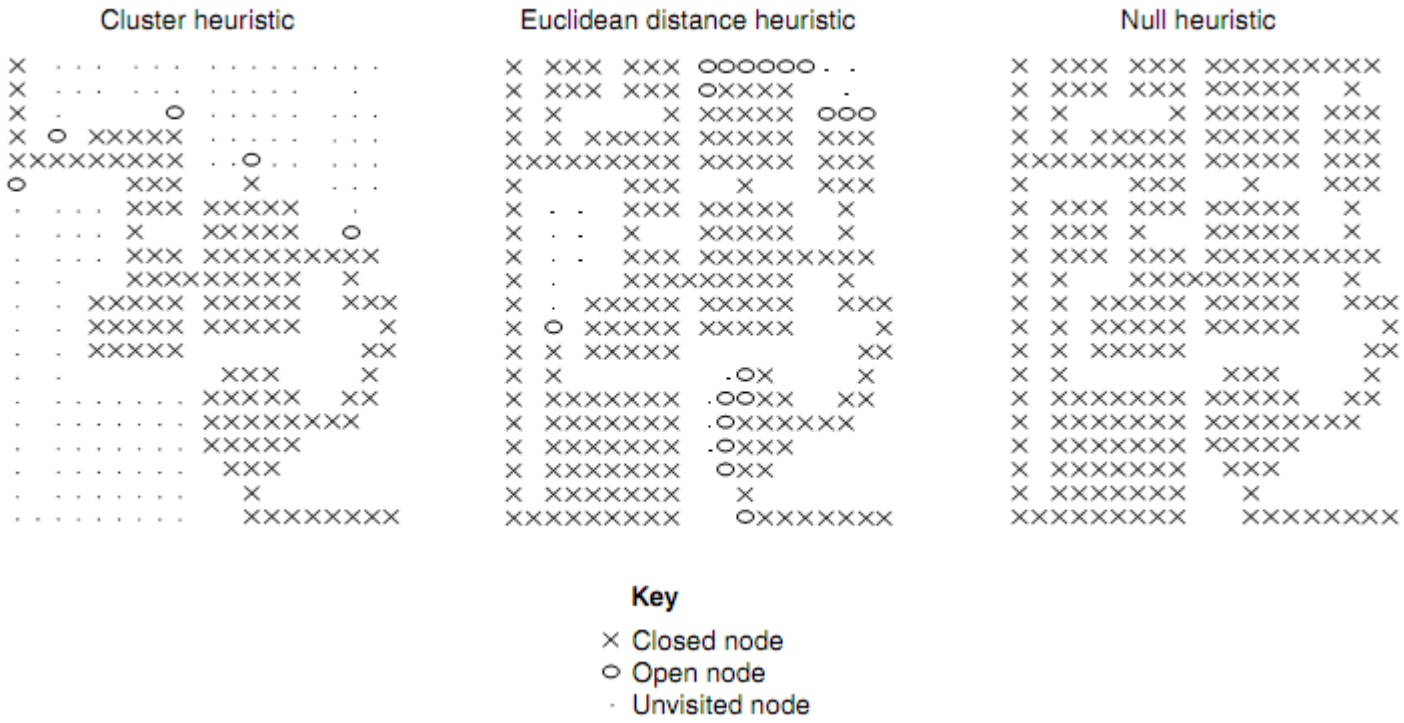



| | A | B | C |
|---|---|---|---|
| A | x | 13 | 29 |
| B | 13 | x | 7 |
| C | 29 | 7 | x |

Lookup table

음성 설명 없음

# 휴리스틱 비교 (같은 path finding 문제에 대해, 각 휴리스틱별로 생성된 closed, open node들의 개수 및 위치)




전체 지면을 타일(tile 또는 cell)들로 나누고, 각 타일 을 node로, 이웃 node들 사이에 edge로 연결하여 graph를 만듬.

음성 설명 종료