2020 Spring AI HW-3, Professor: Minkoo Kim

1. A* algorithm를 여러분의 말로 설명하는데 반드시 evaluation function을 언급하여 정확하게 설명하시오. 그리고 A* algorithm을 이용하여 다음의 8-puzzle problem을 해결하는데 매 단계마다 evaluation function이 어떻게 되어 다음 단계로 옮겨지는 정확하게 설명하시오. (10 Points)

1	2	
5	6	3
4	7	8

<start state>

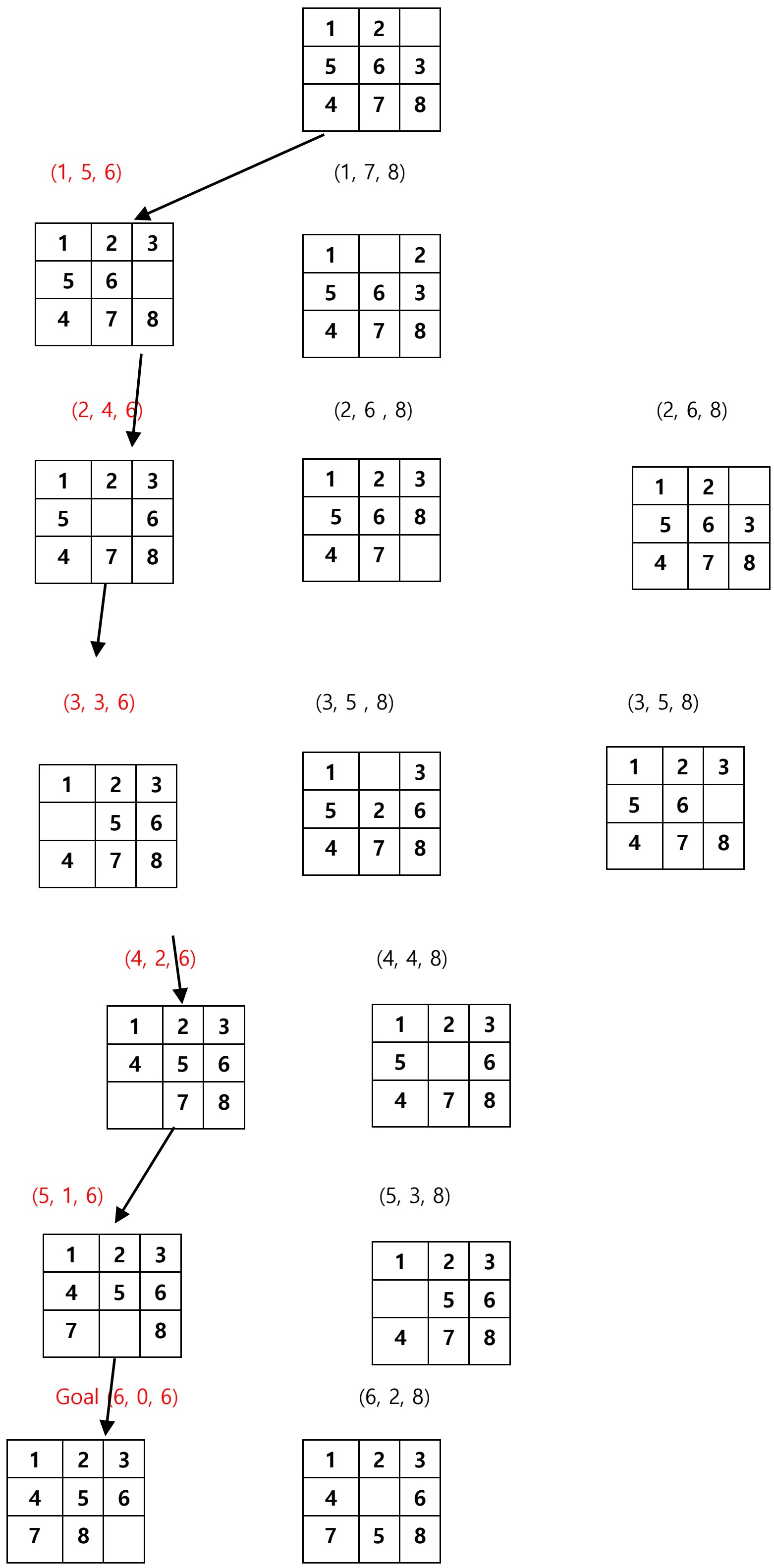
1	2	3
4	5	6
7	8	

<goal state>

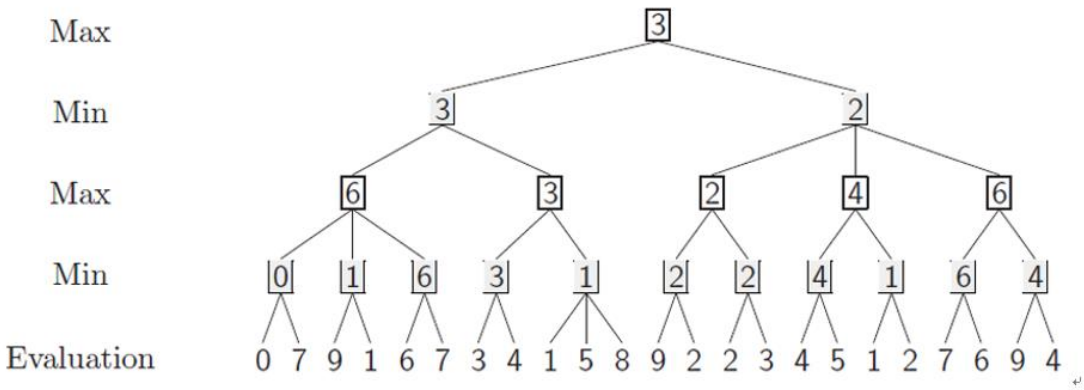
(Solution)

A* search is the best-first search with the following heuristic evaluation function f(n): f(n) = g(n) + h(n) where g(n) is the actual cost from the start node to the current node n and h(n) is the estimated cost from the current node n to the goal node. If f is admissible (that is h is not overestimated), we can find an optimal solution using the A* search algorithm.

I would like to use Manhattan distance as a heuristic function h to solve the above problem. The values in the below (   ) are g(n), h(n), and f(n), respectively.


(1, 5, 6)
(1, 7, 8)
(2, 4, 6)
(2, 6 , 8)
(2, 6, 8)
(3, 3, 6)
(3, 5 , 8)
(3, 5, 8)
(4, 2, 6)
(4, 4, 8)
(5, 1, 6)
(5, 3, 8)
Goal (6, 0, 6)
(6, 2, 8)

2. 강의노트에 있는 α-β pruning algorithm을 여러분의 말로 설명하고 다음 문제를 구체적으로 알고리즘을 따라 가면서 α, β 값이 어떻게 변하여 알고리즘이 작동되는지 보이시오. (10 Points)



(Solution)

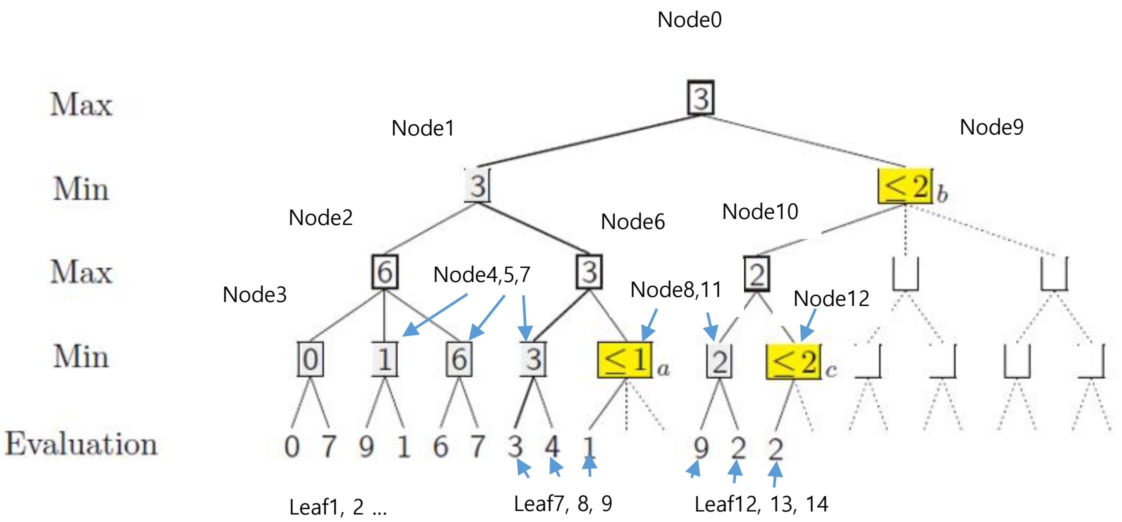
AlphaBetaMax(Node, α, β)

If DepthLimitReached(Node) **Return**(Rating(Node))
NewNodes = Successors(Node)
While NewNodes $\neq \emptyset$
  α = Maximum(α, AlphaBetaMin(First(NewNodes), α, β)
  If $\alpha \geq \beta$ **Return**(β)
  NewNodes = Rest(NewNodes)
Return(α)

AlphaBetaMin(Node, α, β)

If DepthLimitReached(Node) **Return**(Rating(Node))
NewNodes = Successors(Node)
While NewNodes $\neq \emptyset$
  β = Minimum(β, AlphaBetaMax(First(NewNodes), α, β)
  If $\beta \leq \alpha$ **Return**(α)
  NewNodes = Rest(NewNodes)
Return(β)

말로 설명하는 것은 생략되었음.



The nodes are numbered in depth-first manner and leaf nodes are numbered from left to the right.

AlphaBetMax(Node0,−∞, ∞)

AlphaBetMin(Node1,−∞, ∞)

AlphaBetMax(Node2,−∞, ∞)

AlphaBetMin(Node3,−∞, ∞)

AlphaBetMax(Leaf1,−∞, ∞)

Return(0)

β = 0

AlphaBetMax(Leaf2,−∞, 0)

Return(7)

Return(0)

α = 0

AlphaBetMin(Node4, 0, ∞)

…

Return(1)

α = 1

AlphaBetMin(Node5, 1, ∞)

…

Return(6)

Return(6)

β = 6

AlphaBetMax(Node6,−∞, 6)

```
    AlphaBetMin(Node7, −∞, 6)
       AlphaBetMax(Leaf7, −∞, 6)
         Return(3)
       β = 3
       AlphaBetMax(Leaf8, −∞, 3)
         Return(4)
       Return(3)
    α = 3
    AlphaBetMin(Node8, 3, 6)
        AlphaBetMax(Leaf9, 3, 6)
          Return(1)
        β = 1; if β ≤ α then
    Return(3)
  Return(3)
β = 3
  Retrun(3)
α = 3
AlphaBetMin(Node9, 3, ∞)
   AlphaBetMax(Node10, 3, ∞)
      AlphaBetMin(Node11, 3, ∞)
         AlphaBetMax(Leaf12, 3, ∞)
           Return(9)
         AlphaBetMax(Leaf13, 3, ∞)
           Return(2)
         Return(2)
     α = 2
     AlphaBetMin(Node12, 2, 3)
         AlphaBetMax(Leaf14, 2, 3)
           Return(2)
         β = 2; if β ≤ α then
    Return(2)
  β = 2
```

```
    if β ≤ α then Return(2)
Return(3); /*Max{3, 2} = 3*/
```