

1. 각각에 대하여 탐색 순서(노드 확장 순서)를 표기한 탐색트리를 작성한다.

<p>HC 알고리즘</p>	<p>HC 알고리즘</p>
<p>A* 알고리즘</p>	<p>A* 알고리즘</p>

## 2. 탐색 시 확장된 노드의 수에 어떤 차이가 있는지 설명하고 그 차이가 어디에서 오는지 설명하시오.

이번 과제의 시작 노드[2,4,3,1,0,6,7,5,8]로 진행하면 HC 알고리즘은  $g(n)$ 을 고려하지 않아 계속 아래쪽으로 내려가기 때문에 Local Maximum에 갇히게 되고 그로 인해 확장된 노드의 수가 같은 노드까지 도달하고 다시 다른 단계의 노드로 돌아가 진행하여 결국 Global Maximum에 도달하는 A\* 알고리즘에 비해 노드의 수가 적다.

## 3. 두 방법이 최적의 해를 찾아내는지 여부를 표기하고 그 이유를 설명하시오.

HC 알고리즘은 A\* 알고리즘의 가장 큰 차이점은 A\* 알고리즘은 Global Maximum에 도달할 것을 보장하지만 HC 알고리즘은 Local Maximum 안에 갇힐 수도 있다는 점이다. 이러한 특징으로 인해 속도는 빠르지만 A\* 알고리즘과는 달리 optimal한 결과를 보장하지 못한다는 점이다.

HC 알고리즘과는 다르게 A\* 알고리즘은  $h(n)$ :(꼭지점  $n$ 으로부터 목표 꼭지점까지의 추정 경로 가중치)  $g(n)$ :(출발 꼭지점으로부터 꼭지점  $n$ 까지의 경로 가중치)을 고려하여 진행하기 때문에 optimal한 결과를 보장한다.

이번 과제의 시작 노드[2,4,3,1,0,6,7,5,8]로 진행하면 A\* 알고리즘은 optimal을 보장하며 Global Maximum에 도달하지만 HC 알고리즘을 사용하면 Global Maximum이 아닌 Local Maximum에 갇히게 된다.

4. 파이썬으로 작성된 A\* 알고리즘 프로그램을 작성하여 실행해 보고 결과화면을 캡처하시오. 언덕오르기 방법으로 프로그램을 수정하여 실행해 보고 화면을 캡처하여 리포트에 포함시킨다.

#### 8puzzle\_HC 알고리즘\_소스코드

```
import queue

# 상태를 나타내는 클래스, f(n) 값을 저장한다.
class State:
    def __init__(self, board, goal):
        self.board = board
        self.goal = goal

    # i1과 i2를 교환하여서 새로운 상태를 반환한다.
    def get_new_board(self, i1, i2):
        new_board = self.board[:]
        new_board[i1], new_board[i2] = new_board[i2], new_board[i1]
        return State(new_board, self.goal)

    # 자식 노드를 확장하여서 리스트에 저장하여서 반환한다.
    def expand(self):
        result = []
        i = self.board.index(0)          # 숫자 0(빈칸)의 위치를 찾는다.
        if not i in [0, 1, 2]:           # UP 연산자
            result.append(self.get_new_board(i, i-3))
        if not i in [0, 3, 6]:           # LEFT 연산자
            result.append(self.get_new_board(i, i-1))
        if not i in [2, 5, 8]:           # RIGHT 연산자
            result.append(self.get_new_board(i, i+1))
        if not i in [6, 7, 8]:           # DOWN 연산자
            result.append(self.get_new_board(i, i+3))
        return result

    # 휴리스틱 함수 값인 h(n)을 계산하여 반환한다.
    # 현재 제 위치에 있지 않은 타일의 개수를 리스트 함축으로 계산한다.
    def h(self):
        return sum([1 if self.board[i] != self.goal[i] else 0 for i in range(8)])
```

<pre> # 상태와 상태를 비교하기 위하여 less than 연산자를 정의한다. def __lt__(self, other):     return self.h() &lt; other.h()  # 객체를 출력할 때 사용한다. def __str__(self):     return "----- h(n)=" + str(self.h()) + "\n"+\            str(self.board[:3]) + "\n"+\            str(self.board[3:6]) + "\n"+\            str(self.board[6:]) + "\n"+\            "-----"  # 초기 상태 puzzle = [2, 4, 3,           1, 0, 6,           7, 5, 8]  # 목표 상태 goal = [1, 2, 3,         4, 5, 6,         7, 8, 0]  # open 리스트는 우선순위 큐로 생성한다. open_queue = queue.PriorityQueue() open_queue.put(State(puzzle, goal)) closed_queue = [ ] while not open_queue.empty():     current = open_queue.get()     print(current)     if current.board == goal:         print("탐색 성공")         break     for state in current.expand():         if state not in closed_queue:             open_queue.put(state)             closed_queue.append(current)         else:             print('탐색 실패') </pre>
<p style="text-align: center;"><b>8puzzle_HC 알고리즘_실행화면</b></p>
<pre> ----- h(n)=5 [2, 4, 3] </pre>

```
[1, 0, 6]
[7, 5, 8]
```

```
-----
----- h(n)=4
```

```
[2, 4, 3]
[1, 5, 6]
[7, 0, 8]
```

```
-----
----- h(n)=3
```

```
[2, 4, 3]
[1, 5, 6]
[7, 8, 0]
```

```
-----
----- h(n)=4
```

```
[2, 4, 3]
[1, 5, 0]
[7, 8, 6]
```

```
-----
----- h(n)=3
```

```
[2, 4, 3]
[1, 5, 6]
[7, 8, 0]
```

```
-----
----- h(n)=4
```

```
[2, 4, 3]
[1, 5, 6]
[7, 0, 8]
```

```
-----
```

### 8puzzle\_A \* 알고리즘\_소스코드

```
import queue
```

```
# 상태를 나타내는 클래스, f(n) 값을 저장한다.
```

```
class State:
```

```
    def __init__(self, board, goal, moves=0):
```

```
        self.board = board
```

```
        self.moves = moves
```

```
        self.goal = goal
```

```
# i1과 i2를 교환하여서 새로운 상태를 반환한다.
```

```
def get_new_board(self, i1, i2, moves):
```

```
    new_board = self.board[:]
```

```
    new_board[i1], new_board[i2] = new_board[i2], new_board[i1]
```

```
    return State(new_board, self.goal, moves)
```

```
# 자식 노드를 확장하여서 리스트에 저장하여서 반환한다.
```

```
def expand(self, moves):
```

```

result = []
i = self.board.index(0)      # 숫자 0(빈칸)의 위치를 찾는다.
if not i in [0, 1, 2] :      # UP 연산자
    result.append(self.get_new_board(i, i-3, moves))
if not i in [0, 3, 6] :      # LEFT 연산자
    result.append(self.get_new_board(i, i-1, moves))
if not i in [2, 5, 8]:        # RIGHT 연산자
    result.append(self.get_new_board(i, i+1, moves))
if not i in [6, 7, 8]:        # DOWN 연산자
    result.append(self.get_new_board(i, i+3, moves))
return result

# f(n)을 계산하여 반환한다.
def f(self):
    return self.h()+self.g()

# 휴리스틱 함수 값인 h(n)을 계산하여 반환한다.
# 현재 제 위치에 있지 않은 타일의 개수를 리스트 함축으로 계산한다.
def h(self):
    return sum([1 if self.board[i] != self.goal[i] else 0 for i in range(8)])

# 시작 노드로부터의 경로를 반환한다.
def g(self):
    return self.moves

# 상태와 상태를 비교하기 위하여 less than 연산자를 정의한다.
def __lt__(self, other):
    return self.f() < other.f()

# 객체를 출력할 때 사용한다.
def __str__(self):
    return "----- f(n)=" + str(self.f()) + "\n" + \
           "----- h(n)=" + str(self.h()) + "\n" + \
           "----- g(n)=" + str(self.g()) + "\n" + \
           str(self.board[:3]) + "\n" + \
           str(self.board[3:6]) + "\n" + \
           str(self.board[6:]) + "\n" + \
           "-----"

# 초기 상태

```

```

puzzle = [2, 4, 3,
          1, 0, 6,
          7, 5, 8]
# 목표 상태
goal = [1, 2, 3,
        4, 5, 6,
        7, 8, 0]

# open 리스트는 우선순위 큐로 생성한다.
open_queue = queue.PriorityQueue()
open_queue.put(State(puzzle, goal))
closed_queue = [ ]
moves = 0
while not open_queue.empty():
    current = open_queue.get()
    print(current)
    if current.board == goal:
        print("탐색 성공")
        break
    moves = current.moves+1
    for state in current.expand(moves):
        if state not in closed_queue:
            open_queue.put(state)
            closed_queue.append(current)
        else:
            print ('탐색 실패')

```

### 8puzzle\_A\* 알고리즘\_실행화면

```

----- f(n)=5
----- h(n)=5
----- g(n)=0
[2, 4, 3]
[1, 0, 6]
[7, 5, 8]
-----
----- f(n)=5
----- h(n)=4
----- g(n)=1
[2, 4, 3]
[1, 5, 6]
[7, 0, 8]
-----
----- f(n)=5
----- h(n)=3
----- g(n)=2

```

[2, 4, 3]  
[1, 5, 6]  
[7, 8, 0]

-----  
----- f(n)=6  
----- h(n)=5  
----- g(n)=1

[2, 0, 3]  
[1, 4, 6]  
[7, 5, 8]

-----  
----- f(n)=6  
----- h(n)=5  
----- g(n)=1

[2, 4, 3]  
[0, 1, 6]  
[7, 5, 8]

-----  
----- f(n)=6  
----- h(n)=4  
----- g(n)=2

[0, 2, 3]  
[1, 4, 6]  
[7, 5, 8]

-----  
----- f(n)=6  
----- h(n)=3  
----- g(n)=3

[1, 2, 3]  
[0, 4, 6]  
[7, 5, 8]

-----  
----- f(n)=6  
----- h(n)=2  
----- g(n)=4

[1, 2, 3]  
[4, 0, 6]  
[7, 5, 8]

-----  
----- f(n)=6  
----- h(n)=1  
----- g(n)=5

[1, 2, 3]  
[4, 5, 6]  
[7, 0, 8]

-----  
----- f(n)=6  
----- h(n)=0  
----- g(n)=6

[1, 2, 3]  
[4, 5, 6]  
[7, 8, 0]

-----  
탐색 성공