**Problem**

* **nodeExpanded**: số trạng thái đã duyệt qua
* **startState** (*list*): trạng thái ban đầu
* **nonePosition**: vị trí nút “0” trong state
* **dicGoal** (*dict*): một dictionary chứa vị trí (x, y) của mỗi nút trong trạng thái đích (dùng để tính giá trị heuristic)

Funtion

* **findNonePosition**(*matrix*): tìm vị trí “0” trong trạng thái hiện tại (matrix 3x3)
  + Trả về một tupe (x, y)
* **setStartState**(*matrix*): đặt trạng thái ban đầu bằng ma trận được đưa vào và đặt startState bằng một tupe gồm (matrix, nonePosition). Giá trị nonePosition được tìm thông qua hàm findNonePosition.
* **setGoalState**(*matrix*): đặt trạng thái đích và lưu các vị trí (x, y) của từng nút vào dicGoal.
* **getStartState**(): trả về một tupe (startState, nonePosition)
* **getGoalState**(): trả về một tupe (goalState, dicGoal)
* **isGoalState**(*state*):
  + state = (matrix, nonePosition)
  + So sánh trạng thái hiện tại với trạng thái đích. Nếu giống trả về True, ngược lại trả về False.
* **isValidPosition**(*x, y*): nếu vị trí (x, y) vượt ra khỏi ma trận 3x3 trả về False, ngược lại trả về True.
* **getSuccessor**(*state*): trả về một danh sách chứa những trạng thái kế tiếp của trạng thái hiện tại.
  + Mỗi trạng thái sẽ có nhiều nhất 4 trạng thái kết tiếp. Tương ứng với vị trí “0” di chuyển theo 4 hướng “North, East, South, West”
  + Một khi hàm **getSuccessor** được gọi, giá trị nodeExpanded sẽ tăng lên 1.

**PriorityQueu**:

**Direction**:

**Solve\_Problem**

Function

* **aStarSearch**(*problem*):
  + Hàm sử dụng thuật toán A\* để giải bài toán.
  + Trả về một tupe chứa (state, moveDirection, cost, nonePositionMove)
    - state = (matrix, nonePosition)
    - cost: tổng số bước đi từ trạng thái ban đầu đến trạng thái đích.
    - nonePositionMove: danh sách chứa vị trí của “0” tại mỗi bước đi.
* **getAnswer**(*problem*):
  + Sử dụng hàm **aStarSearch** và trả về một tupe chứa (nonePositionMove, cost, nodeExpanded)
* **printState**(*state*): in trạng thái trong console
  + state = matrix
* **printGoalState**(*problem*): in trạng thái đích trong console
* **printStartState**(*problem*): in trạng thái bắt đầu trong console
* **printAnswer**(*problem*):
  + in answer trong console
* **heuristicFunction**(*state, problem*):
  + Hàm ước lượng giá trị heuristic h(x) từ trạng thái (state) đến trạng thái đích.
* **doAction**(*problem*):
  + in từng bước đi từ trạng thái ban đầu đến trạng thái đích trong console.

**Hàm tính giá trị heuristic:**

*Ta có:*

* x, y là vị trí của nút thứ i trong trạng thái hiện tại với i = 0-8
* dx, dy là vị trí của nút i tương ứng trong trạng thái đích.
* temp = |x – dx|2 + |y – dy|2
* temp = temp + 0.1\*temp
  + Qua thực nghiệm hằng số 0.1 cho giá trí tối ưu trong nhiều trường hợp.
* Tổng của các giá trị temp của tất cả 9 nút là giá trị heuristic cần tìm.

Theo công thức ước lượng giá trị heuristic trên, cho ra kết quả tối ưu về mặt thời gian nhưng kết quả không phải là số bước nhỏ nhất để giải bài toán 8-puzzle trong một số trường hợp.