**Full Name: Nguyễn Thế Huy**

**Class: K57CA**

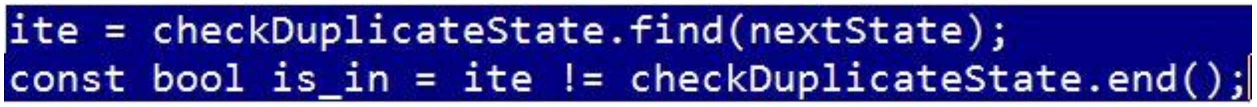
**Student Code: 12020174**

# Assignment Report – Artificial Intelligence

**Note: When compile, please add –Ofast in parameter to optimize speed of application**

1. **<u>How to detect and avoid repeated state in A* implementation ?</u>**
   In my solution, I use set (a kind of data structure is supported in C++) to check duplicate Sets of C++ are containers that store unique elements following a specific order and implemented in binary search tree.

   Here is my code to check if a state existed in set:



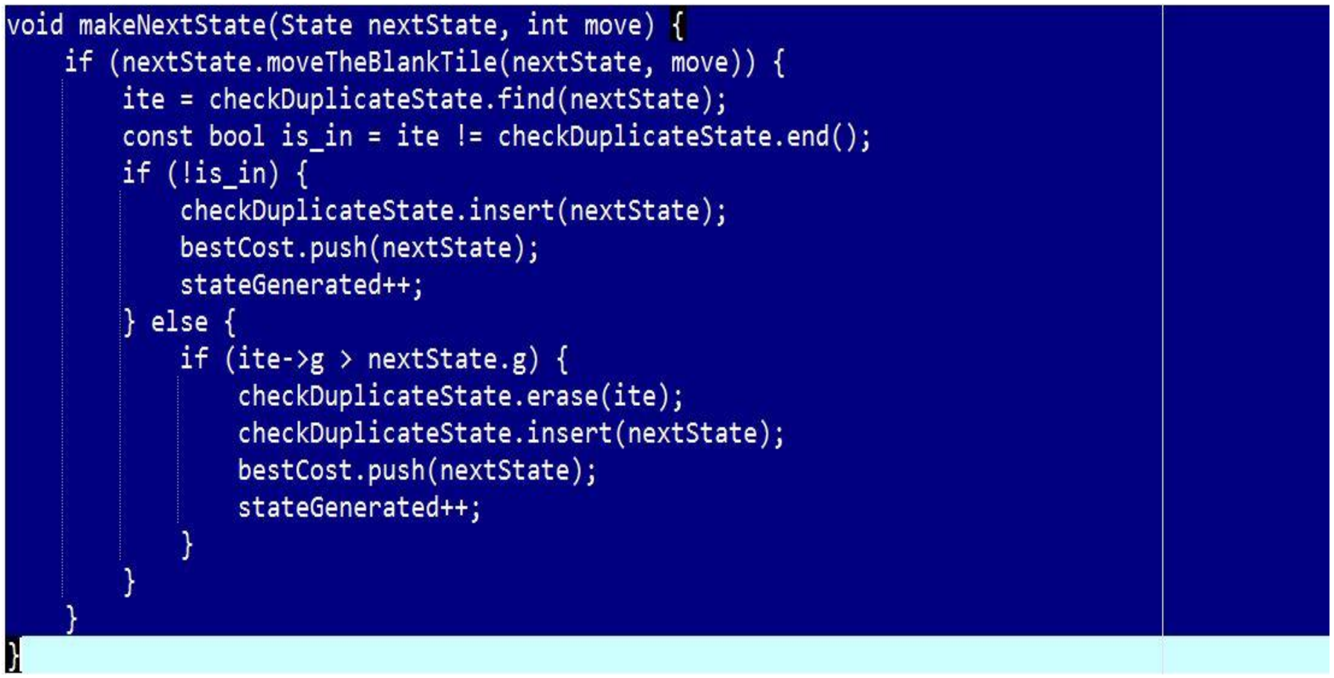
```
ite = checkDuplicateState.find(nextState);
const bool is_in = ite != checkDuplicateState.end();
```

   Ite is an iterator for set:

```
set<State, compareTwoStateInSet> checkDuplicateState; // a set to
store generated state and check duplicate state
set<State, compareTwoStateInSet>::iterator ite; //iterator for set
```

   **<u>Explain:</u>**
   We check whether a state is duplicated by its value in board. In a set of C++, it provides find() method to find a specific element. If found, find() return pointer to element is found. If no, find() return pointer to the last element of set. So, we can check: if find(state) is not equal to last element (provided by end() method), this state existed in our set. If no duplicate, we can insert it to the set. But if  we have the same state (means that the same board), we have to check g (cost from initial state to current state). If new state has g better than current state in set, we delete old state and insert new state

   Full code:



```
void makeNextState(State nextState, int move) {
    if (nextState.moveTheBlankTile(nextState, move)) {
        ite = checkDuplicateState.find(nextState);
        const bool is_in = ite != checkDuplicateState.end();
        if (!is_in) {
            checkDuplicateState.insert(nextState);
            bestCost.push(nextState);
            stateGenerated++;
        } else {
            if (ite->g > nextState.g) {
                checkDuplicateState.erase(ite);
                checkDuplicateState.insert(nextState);
                bestCost.push(nextState);
                stateGenerated++;
            }
        }
    }
}
```

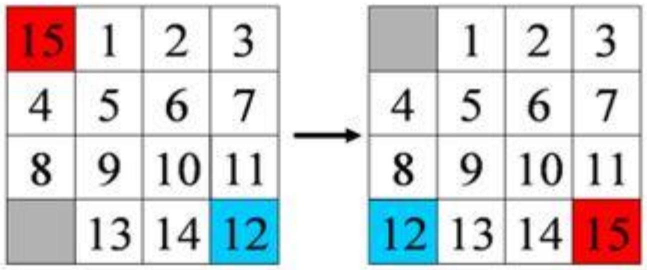
Because of implementing in binary search tree, set is fast to find and check duplicated element with time complexity average: O(log n)

2. **<u>My heuristic is by flying distance:</u>**

**<u>The formula:</u>**

$$h(s)=\sum_{i=1}^{n}\left(\sqrt{\left(x_i(s)-\overline{x}_i\right)^2+\left(y_i(s)-\overline{y}_i\right)^2}\right)$$

We consider this example:



The flying distance in this example $\sqrt{3^2+3^2}+\sqrt{0^2+3^2}=\sqrt{18}+\sqrt{9}$

In a right triangle, the side opposite the right angle is equal to the sum of the squares of the other two sides (Pythagorean theorem) and less than sum of them.
Flying distance represents the length of the hypotenuse and Manhantan distance is the sum of the right triangle's other two sides. So: flying distance always less than Manhantan

distance. On the other hand, manhantan distance is admissible. So we have proved that, flying distance is always admissible

Flying distance is worse than Manhantan distance, but better than missed place