Bicol University College of Science

CS-IT Department

1st Semester 2021-2022

Artificial Intelligence

A Written Report in Solving 8 Puzzle using Blind Search and Heuristics Search Strategy

(Iterative Deepening Search and A Search)*

Prepared by:

Arvy Llave

Aira Lumawag

Alliana Ermino

Romelyn Banzuela

BSCS-3A

Arlene Satuito

Professor

This program is intended to solve an 8-puzzle problem using a specific search strategy which are the Heuristic Search and the Iterative Deepening Search. In Heuristic search the code was implemented using A Search and the Manhattan Distance for the Heuristic Search. An 8 puzzle problem is a 3x3 board with 8 tiles corresponds to a numbers 1 to 8 and one 0 or one empty space. The goal of the problem is to place all the numbers in tiles and the given inputted initial state must be matched to the goal state configuration. The implementation of the program was done using C++.*

Soirce Code Imlementation:

The program initiated with the different data structure and a pointer.An ***struct singlestate*** *is used to store several data's such as the array for a given tile arrangements, the level/depth from the chosen starting state, the pointer to the parentState and for the preferred/previous moved and the heuristicValue. An* ***struct listofStates*** *are designated for the new set of data or a series of a linked list for the states since an 8 puzzle games corresponds to a certain state.*

```
struct singlestate{
        int blankTileX; //blanktile index row
        int blankTileY; //blanktile index column
        int tiles[SIZE][SIZE];
        string chosenmove;
        int level;
        singlestate* parentState;
        singlestate* up;
        singlestate* down;
        singlestate* left;
        singlestate* right;
        int heuristicValue;
};
struct listofstates{
        singlestate* state;
        listofstates* fringeNext;
};
```

Functions declaration for A Search Strategy and Iterative Deepening Search Strategy.*

```
void iterativeDeepening(singlestate* initialState);
void astarsearch(singlestate* initialState);
```

Functions for accessing and for inserting into a list in which it picks first the preferred states then insert into the list, checked if it is not yet expanded and pick State with the lowest heuristic value.

```
singlestate* pickfirststate(listofstates** list);
void insertintothelist(listofstates** list, singlestate* state);
bool notYetExpanded(listofstates* list, singlestate* state);
singlestate* pickStateWithLowestHeuristic(listofstates** list);
```

Implementation of A* Search

*In the function **void astarsearch(singlestate* initialState)**, we declared a two nodes. First is **listofstates* openlist;** it contains those nodes that have been evaluated by the heuristic function but have not been expanded into successors yet and second is the **listofstates* closelist** that contains those nodes that have already been visited. A while loop is created in the function and exits when openlist == NULL. In the while loop there is a if statement that check if the current state/node is equal to goal node and if it equals the function exit, otherwise the loop will continue to expand the node and it removes the node with the smallest value of f(n) from OPEN and move it to list CLOSED. In getting the lowest heuristic value a function pickStateWithLowestHeuristic(listofstates** list) is used.*

Implementation of IDS Search

Given function iterativeDeepening(singlestate initialState). It composed of a 2 while expression in which the first while expressions refers to the increase of level. It is basically same with the idea of Depth first Search. But the difference is there are also 2 list of states that are mentioned, the visited list and the idsFringe. The purpose of the visited list is to make sure that the previous expanded nodes aren't be expanded again, since if we expand again those previous expanded nodes it will always tends to iterate or result to infinite loop. In idsFringe it specify that the while(idsFringe != NULL) is not satisfies it is always continue in expanding the node, implementing the chosen moves like up, down or left until if the moves are valid it can be inserted in the idsfringe. If the goal state is cannot be reached it will continue in expanding the node. In terms of limit, given that we have a 2 while in this IDS, it refers in to two ideas in which the first while is for incrementing the limit and the other one is for the DFS so since the limit started at level 0 the dfs is also in a level 0 or the root node then it exits at inner while loop since it finished*

the level 0.Then if the outer while loop is satisfies again it repeat the process but the limit is incremented or increases.

The void insertintothelist function refers only to the function that insert nodes into the list.

```
void insertintothelist(listofstates** list, singlestate* state)
{
        listofstates* currentList = new(listofstates);
        currentList->state = state;
        currentList->fringeNext = NULL;
        if(*list != NULL)
        {
                currentList->fringeNext = *list;
        }
        *list = currentList;
}
```

This function checks the goal or if the corresponding IDS or A search found the respective goal state. The main purpose of this is to compare the array of the goal state and the array of the initial state if there are similar to the arrangements of the tile.*

```
bool checkGoal(singlestate* state1)
{
        for (int i = 0; i < SIZE; i++)
        {
                for (int j = 0; j < SIZE; j++)
                {
                        if(state1->tiles[i][j] != goalState[i][j])
                                return false;
                }
        }
        return true;
}
```

This function simply prints the path and return the cost.

```
int moveSequence(singlestate* state)
{
        if(state != NULL)
        {
                int i;
                i = moveSequence(state->parentState) + 1;
                cout << " "<<state->chosenmove<<" >";
                return i;
        }else{
                return 0;
        }
}
```

Initial State				*IDS*	*A**
Easy			*Solution Path*	*Up-Right-Up-Left-Down*	*Up-Right-Up-Left-Down*
1	3	*4*	*Solution Cost*	*5*	*5*
8	*6*	*2*	*Number of nodes Expanded*	*117*	*5*
7		*5*	*Running Time*	*0*	*0*
Medium			*Solution Path*	*Up-Right-Right-Down-Left-Left-Up-Right-Down*	*Up-Right-Right-Down-Left-Left-Up-Right-Down*
2	*8*	*1*	*Solution Cost*	*9*	*9*
	4	*3*	*Number of nodes Expanded*	*992*	*17*
7	*6*	*5*	*Running Time*	*0.015625*	*0*
Hard			*Solution Path*	*Left-Left-Up-Right-Down-Left-Up-Right-Down-Left-Up-Up-Right-Right-Down-Left-Left-Up-Right-Down*	*Left-Up-Left-Up-Right-Right-Down-Left-Left-Up-Right-Down*
2	*8*	*1*	*Solution Cost*	*20*	*12*
4	*6*	*3*	*Number of nodes Expanded*	*23848*	*26*

7	*5*		*Running Time*	*5.35938*	*0*
Worst			*Solution Path*	*Left-Down-Right-Right-Up-Up-Left-Left-Down-Down-Right-Right-Up-Up-Left-Left-Down-Down-Right-Right-Up-Up-Left-Left-Down-Down-Right-Right-Up-Left*	*Up-Left-Down-Down-Right-Right-Up-Up-Left-Left-Down-Down-Right-Right-Up-Up-Left-Left-Down-Down-Right-Right- Up-Up-Left-Left-Down-Down-Right-Up*
5	*6*	*7*	*Solution Cost*	*30*	*30*
4	*0*	*8*	*Number of nodes Expanded*	*213565*	*940*
3	*2*	*1*	*Running Time*	*555.281*	*0.0625*
Your preferred initial configuration			*Solution Path*	*Right-Up-Left-Down-Right-Down-Left-Left-Up-Right-Up-Left-Down-Right*	*Right-Up-Left-Down-Right-Down-Left-Left-Up-Right-Up-Left-Down-Right*
2	*3*	*1*	*Solution Cost*	*14*	*14*
7	*0*	*8*	*Number of nodes Expanded*	*6640*	*66*
6	*5*	*4*	*Running Time*	*0.359375*	*0*

After the execution of the program, certain similarities and differences of the Iterative Deepening Search and A Search Strategy are noticed. Based on the given table the result shows some advantages and consistency in A* especially in computing their heuristics cost probably because it is an Informed Search Algorithm compare to IDS in computing their heuristic cost its quite hard since*

IDS is a Blind Search Algorithm. In general, their both good Search strategy in solving an 8 puzzle problem but in terms of worst configuration in finding the IDS it took a couple of time to get the corresponding output.

Sample Run Program:

EASY

```
yhanaermino@DESKTOP-OVF7RFK:~$ g++ mp.cpp -o mp
yhanaermino@DESKTOP-OVF7RFK:~$ ./mp

---------------------------------------
                EXAMPLE
---------------------------------------

If this is your desired initial state:

                1 2 3
                8 0 4
                7 6 5

Enter: 1 2 3 8 0 4 7 6 5

Only the integers 0 to 8 are allowed
and each can only be entered once.
---------------------------------------

Please enter the values for the Initial State: 1 3 4 8 6 2 7 0 5

A* search

Move Sequence:  START > Up > Right > Up > Left > Down >
Solution Cost: 5
Expansion: 5
Running Time: 0



Iterative Deepening search

Move Sequence:  START > Up > Right > Up > Left > Down >
Solution Cost: 5
Expansion: 117
Running time: 0
yhanaermino@DESKTOP-OVF7RFK:~$
```

MEDIUM

```
yhanaermino@DESKTOP-OVF7RFK:~$ ./mp

----------------------------------------
                EXAMPLE
----------------------------------------

If this is your desired initial state:

                1 2 3
                8 0 4
                7 6 5

Enter: 1 2 3 8 0 4 7 6 5

Only the integers 0 to 8 are allowed
and each can only be entered once.
----------------------------------------

Please enter the values for the Initial State: 2 8 1 0 4 3 7 6 5

A* search

Move Sequence:  START > Up > Right > Right > Down > Left > Left > Up > Right > Down >
Solution Cost: 9
Expansion: 17
Running Time: 0



Iterative Deepening search

Move Sequence:  START > Up > Right > Right > Down > Left > Left > Up > Right > Down >
Solution Cost: 9
Expansion: 992
Running time: 0.015625
yhanaermino@DESKTOP-OVF7RFK:~$
```

HARD

```
yhanaermino@DESKTOP-OVF7RFK:~$ ./mp

--------------------------------------
             EXAMPLE
--------------------------------------

If this is your desired initial state:

             1 2 3
             8 0 4
             7 6 5

Enter: 1 2 3 8 0 4 7 6 5

Only the integers 0 to 8 are allowed
and each can only be entered once.
--------------------------------------

Please enter the values for the Initial State: 2 8 1 4 6 3 7 5 0

A* search

Move Sequence:  START > Left > Up > Left > Up > Right > Right > Down > Left > Left > Up > Right > Down >
Solution Cost: 12
Expansion: 26
Running Time: 0



Iterative Deepening search

Move Sequence:  START > Left > Left > Up > Right > Down > Left > Up > Right > Down > Left > Up > Up > Right > Right > Down > Left > Left > Up > Right > Down >
Solution Cost: 20
Expansion: 23848
Running time: 5.35938
yhanaermino@DESKTOP-OVF7RFK:~$
```

WORST

```
yhanaermino@DESKTOP-OVF7RFK:~$ ./mp

--------------------------------------
              EXAMPLE
--------------------------------------

If this is your desired initial state:

              1 2 3
              8 0 4
              7 6 5

Enter: 1 2 3 8 0 4 7 6 5

Only the integers 0 to 8 are allowed
and each can only be entered once.
--------------------------------------

Please enter the values for the Initial State: 5 6 7 4 0 8 3 2 1

A* search

Move Sequence:  START > Up > Left > Down > Down > Right > Right > Up > Up > Left > Left > Down > Down > Right > Right > Up > Up > Left > Left > Down > Down > Right > R
ght > Up > Up > Left > Left > Down > Down > Right > Up >
Solution Cost: 30
Expansion: 940
Running Time: 0.0625



Iterative Deepening search

Move Sequence:  START > Left > Down > Right > Right > Up > Up > Left > Left > Down > Down > Right > Right > Up > Up > Left > Left > Down > Down > Right > Right > Up >
p > Left > Left > Down > Down > Right > Right > Up > Left >
Solution Cost: 30
Expansion: 213565
Running time: 555.281
yhanaermino@DESKTOP-OVF7RFK:~$
```

PREFFERED INITIAL CONFIGURATION

```
yhanaermino@DESKTOP-OVF7RFK:~$ ./mp1

--------------------------------------
        THE 8 PUZZLE GAME
--------------------------------------

If this is your initial state:

        1 2 3
        8 0 4
        7 6 5

Your input should be: 1 2 3 8 0 4 7 6 5

Represent the tiles from 0 to 8.
Start inputting from left to right of each row.
Starting from the top most.
--------------------------------------

Enter the Initial State: 2 3 1 7 0 8 6 5 4

A* search

Move Sequence:  START > Right > Up > Left > Down > Right > Down > Left > Left > Up > Right > Up > Left > Down > Right >
Solution Cost: 14
Expansion: 66
Running Time: 0



Iterative Deepening search

Move Sequence:  START > Right > Up > Left > Down > Right > Down > Left > Left > Up > Right > Up > Left > Down > Right >
Solution Cost: 14
Expansion: 6640
Running time: 0.359375
```

Participation and Contributions

Arvy Llave: *Coding and Contributed for A* Algorithm*

Allianna Ermino: *Contributed for IDS Algorithm and Documentation*

Aira Daet Lumawag: *Documentation and Code for finding the heuristic using Manhattan Distance.*

Romelyn Azueta Banzuela: *Running and Testing the Code for the Table of Analysis and Comparison of the two Algorithms, Documentation.*