Shubhan Singh

SE-Comps B/Batch C

2022300118

DAA Experiment 9: Fifteen puzzle problem

Problem statement:

Find a solution for a 15 puzzle problem using the minimum number of moves

Pseudocode :

```
Procedure FifteenPuzzle(arr, coordi, coordj, scores, inf, UP, DOWN, RIGHT, LEFT, count):
    // Swap elements and update scores based on the current position
    If coordi > 0 Then
        // Code for UP direction
    Else
        scores[UP] = inf
    End If

    If coordi < 3 Then
        // Code for DOWN direction
    Else
        scores[DOWN] = inf
    End If

    If coordj < 3 Then
        // Code for RIGHT direction
    Else
        scores[RIGHT] = inf
```

```
    End If

    If coordj > 0 Then
        // Code for LEFT direction
    Else
        scores[LEFT] = inf
    End If

    // Find the direction with the minimum score
    minindex = -1
    min = inf
    For i = 0 to 3 Do
        If scores[i] <= min Then
            minindex = i
            min = scores[i]
        End If
    End For

    // Swap elements and recursively calculate based on the minimum score
    If minindex == UP Then
        // Swap UP and recursively calculate
    Else If minindex == DOWN Then
        // Swap DOWN and recursively calculate
    Else If minindex == RIGHT Then
        // Swap RIGHT and recursively calculate
    Else If minindex == LEFT Then
        // Swap LEFT and recursively calculate
    End If
End Procedure
```

Source code(C language):

```c
#include<stdio.h>
#include<stdlib.h>

#define UP 0
#define DOWN 1
#define RIGHT 2
#define LEFT 3

#define inf 0x7fffffff

int count=0;

void swap(int* a, int* b){
    int temp=*a;
    *a=*b;
    *b=temp;
}

void prarr(int **arr){
    if(count>0)
    printf("Step %d :\n",count);
    for(int i=0;i<4;i++){
        for(int j=0;j<4;j++){
            if(arr[i][j]!=16){
                printf("%-4d",arr[i][j]);
            }
            else{
                printf("*   ");
            }
        }
        printf("\n");
    }
    printf("\n");
}

int score(int** arr){
    int sc=0;
    for(int i=0;i<4;i++){
        for(int j=0;j<4;j++){
            if(arr[i][j]!=(4*i+j+1) && arr[i][j]!=16){
```

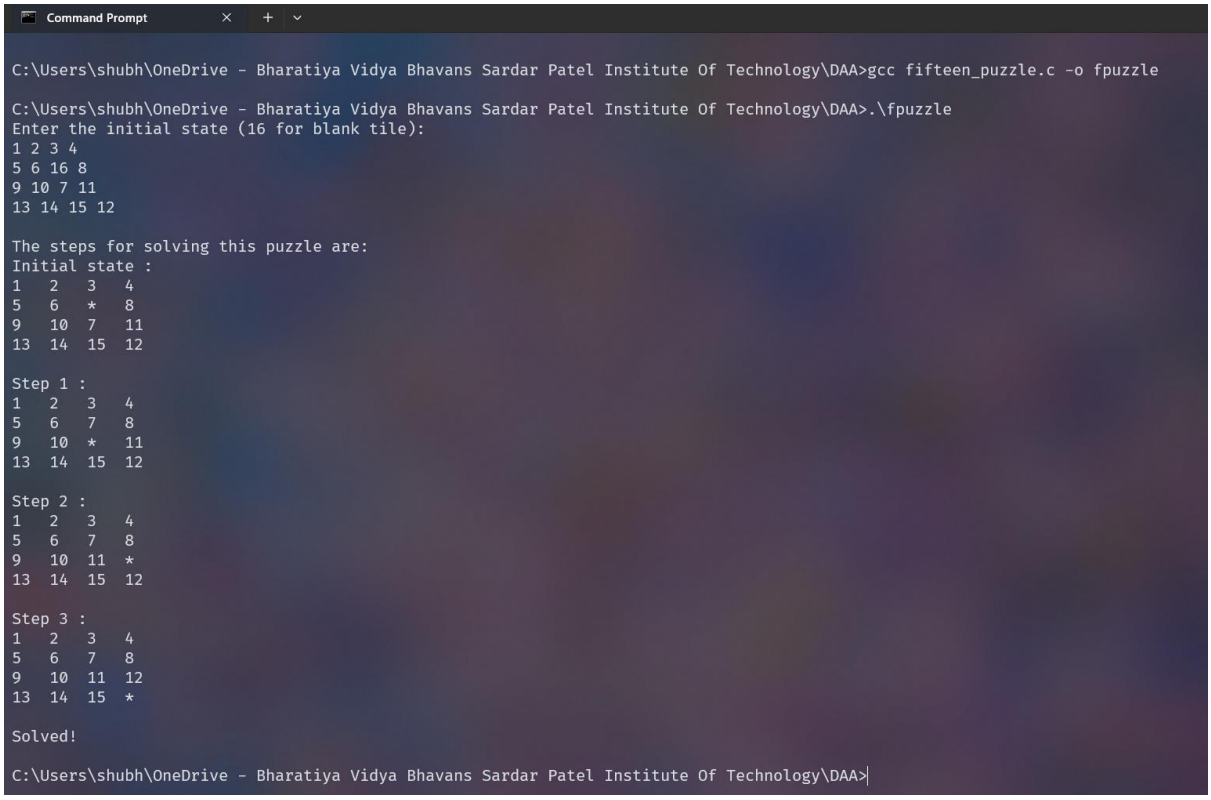
```
                sc++;
            }
        }
    }
    return sc;
}

void calculate(int** arr, int coordi,int coordj, int
direction){
    if(score(arr)==0){
        printf("Solved!\n");
        return;
    }
    int scores[4];
    if(direction!=DOWN){
        if(coordi>0){
            swap(&arr[coordi][coordj],&arr[coordi-
1][coordj]);
            scores[UP]=score(arr);
            swap(&arr[coordi][coordj],&arr[coordi-
1][coordj]);
        }
        else{
            scores[UP]=inf;
        }
    }
    else{
        scores[UP]=inf;
    }
    if(direction!=UP){
        if(coordi<3){
            swap(&arr[coordi][coordj],&arr[coordi+1][coordj])
;
            scores[DOWN]=score(arr);
            swap(&arr[coordi][coordj],&arr[coordi+1][coordj])
;
        }
        else{
            scores[DOWN]=inf;
        }
    }
    else{
```

```
        scores[DOWN]=inf;
    }
    if(direction!=LEFT){
        if(coordj<3){
            swap(&arr[coordi][coordj],&arr[coordi][coordj+1])
;
            scores[RIGHT]=score(arr);
            swap(&arr[coordi][coordj],&arr[coordi][coordj+1])
;
        }
        else{
            scores[RIGHT]=inf;
        }
    }
    else{
        scores[RIGHT]=inf;
    }
    if(direction!=RIGHT){
        if(coordj>0){
            swap(&arr[coordi][coordj],&arr[coordi][coordj-
1]);
            scores[LEFT]=score(arr);
            swap(&arr[coordi][coordj],&arr[coordi][coordj-
1]);
        }
        else{
            scores[LEFT]=inf;
        }
    }
    else{
        scores[LEFT]=inf;
    }
    int minindex=-1,min=inf;
    for(int i=0;i<4;i++){
        if(scores[i]<=min){
            minindex=i;
            min=scores[i];
        }
    }
    if(minindex==UP){
        swap(&arr[coordi][coordj],&arr[coordi-1][coordj]);
        count++;
```

```
        prarr(arr);
        calculate(arr,coordi-1,coordj,UP);
    }
    else if(minindex==DOWN){
        swap(&arr[coordi][coordj],&arr[coordi+1][coordj]);
        count++;
        prarr(arr);
        calculate(arr,coordi+1,coordj,DOWN);
    }
    else if(minindex==RIGHT){
        swap(&arr[coordi][coordj],&arr[coordi][coordj+1]);
        count++;
        prarr(arr);
        calculate(arr,coordi,coordj+1,RIGHT);
    }
    else if(minindex==LEFT){
        swap(&arr[coordi][coordj],&arr[coordi][coordj-1]);
        count++;
        prarr(arr);
        calculate(arr,coordi,coordj-1,LEFT);
    }
return;
}


int main(){
    int **arr;
    arr=malloc(4*sizeof(int*));
    for(int i=0;i<4;i++){
        arr[i]=malloc(4*sizeof(int));
    }
    int arr2[16];
    int sigma=0,x=0;
    int coordi,coordj;
    printf("Enter the initial state (16 for blank tile):\n");
    for(int i=0;i<4;i++){
        for(int j=0;j<4;j++){
            scanf("%d",&arr[i][j]);
            arr2[4*i+j]=arr[i][j];
            if(arr[i][j]==16){
                coordi=i;
                coordj=j;
```

```
                x=(i+j)%2==1;
            }
        }
    }
    sigma+=x;
    for(int i=0;i<16;i++){
        for(int j=i+1;j<16;j++){
            if(arr2[j]<arr2[i]){
                sigma++;
            }
        }
    }
    if(sigma%2==1){
        printf("The goal state is unreachable.\n");
        return 0;
    }
    printf("\nThe steps for solving this puzzle are: \n");
    printf("Initial state : \n");
    prarr(arr);
    if(score(arr)==0){
        printf("Already solved.\n");
        return 0;
    }
    calculate(arr,coordi,coordj,-1);

    for(int i=0;i<4;i++){
        free(arr[i]);
    }
    free(arr);

return 0;
}
```

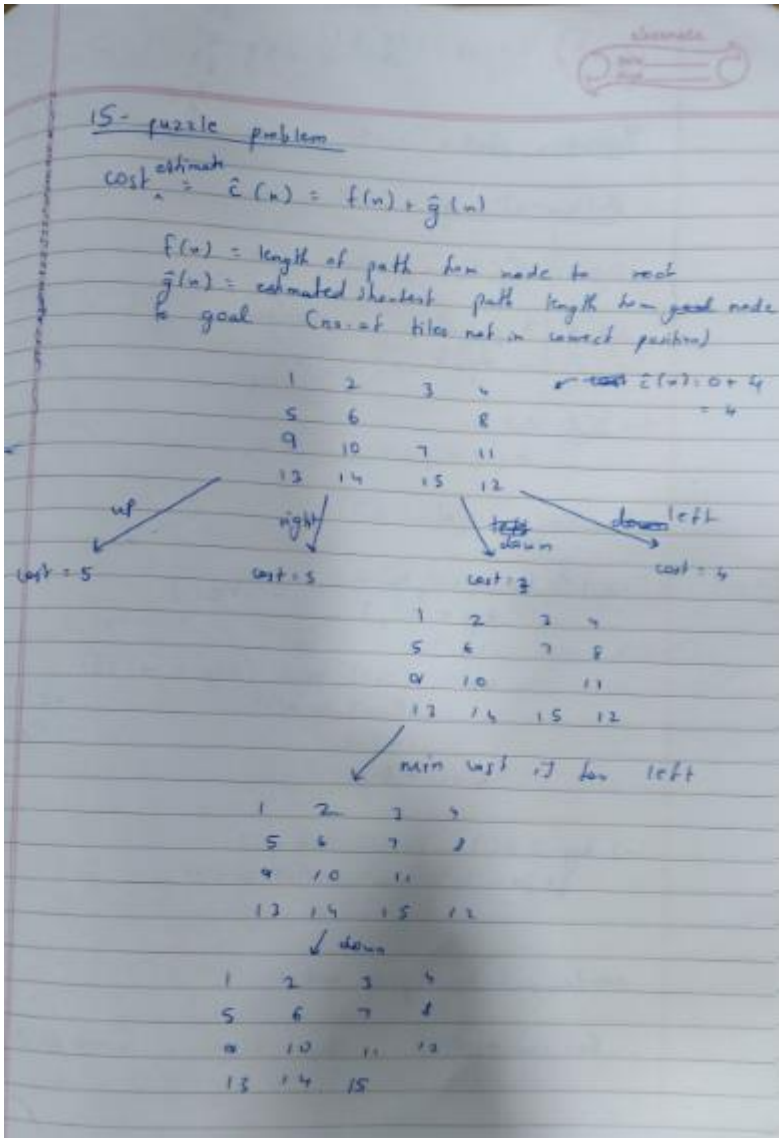
Output:



Conclusion:

- We solved the 15 puzzle problem using branch and bound technique.
- We greedily chose the node closest to the solution and killed all other nodes, and applied this recursively.
- We checked beforehand whether the puzzle was solvable or not to prevent infinite recursion.

Rough Working :



Theory:

The Fifteen Puzzle, also known as Game of Fifteen or 15-Puzzle, is a sliding puzzle that consists of a frame of numbered square tiles in random order with one tile missing. The puzzle also exists in other sizes, creating a category of similar sliding puzzles. The object of the puzzle is to place the tiles in order by making sliding moves that use the empty space.

Solvability of a Fifteen Puzzle

Not all arrangements of the tiles in the Fifteen Puzzle are solvable. To determine whether a given instance of a Fifteen Puzzle is solvable, we can use the concept of "inversions".

An inversion is when a tile precedes another tile with a lower number on it. The position of the blank space is also important. If the grid width is odd, then every solvable state has an even number of inversions. If the grid width is even, then the number of inversions in a solvable situation is odd if the blank is on an even row counting from the bottom (second-last, fourth-last, etc.) and even if the blank is on an odd row counting from the bottom (last, third-last, fifth-last, etc.).

Principles to Solve the Fifteen Puzzle

The Fifteen Puzzle can be solved using various search algorithms, including:

1. **Depth-First Search (DFS)**: DFS is a strategy that searches "deeper" in the tree before searching "wider". However, DFS is not suitable for the Fifteen Puzzle because it can get stuck going down a wrong path for a long time.
2. **Breadth-First Search (BFS)**: BFS is a strategy that searches "wider" in the tree before going "deeper". BFS is guaranteed to find a solution if one exists, but it can be slow and require a lot of memory.

In the provided code, the heuristic used is the number of misplaced tiles. The code calculates the score for each possible move (up, down, left, right), chooses the move with the lowest score, and recursively applies the same process to the new state.