A* Algorithm_1
15 slides puzzle problem

Introduction

This problem is to solve the 15 slides puzzle using the A* algorithm. Heuristics to be used are:

- NUMBER OF MISPLACED TILES.
- SUM OF “NUMBER OF MOVES EACH TILE IS AWAY FROM ITS GOAL POSITION”.

Search terminologies:

Searching is the universal technique of problem solving in AI. There are some single-player games such as tile games, Sudoku, crossword, etc. The search algorithms help you to search for a particular position in such games.

- **Problem Space** − It is the environment in which the search takes place. (A set of states and set of operators to change those states)
- **Problem Instance** − It is Initial state + Goal state.
- **Problem Space Graph** − It represents problem state. States are shown by nodes and operators are shown by edges.

- **Depth of a problem** − Length of a shortest path or shortest sequence of operators from Initial State to goal state.
- **Space Complexity** − The maximum number of nodes that are stored in memory.
- **Time Complexity** − The maximum number of nodes that are created.
- **Admissibility** − A property of an algorithm to always find an optimal solution.
- **Branching Factor** − The average number of child nodes in the problem space graph.
- **Depth** − Length of the shortest path from initial state to goal state.

A* Algorithm

- Initialize the open list
- Initialize the closed list, put the starting node on the open list(you can leave its f at zero) , initialize the closed list, put the starting node on the open list(you can leave its f at zero
- while the open list is not empty
    - Find the node with the least f on the open list, call it”q”.
    - Pop q off the open list
    - Generate q’s 8 successors and set their parents to q
    - For each successor
        - If successor is the goal, stop search
        - successor.g = q.g + distance between sucessor and q.Successor.h = distance from goal to successor.

          successor.**f** = successor.**g** + successor.**h**
        - If a node with the same position as successor is in the OPEN list which has a lower f thean successor, skip this successor.
        - If a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip the successor otherwise,ass the node to the open list

          end(for loop)
    - Push q on the closed list. End (while loop).

Example to explain A* Algorithm:

Now let us apply the algorithm on the above search tree and see what it gives us. We will go through each iteration and look at the final output. Each element of the priority queue is written as [path,f(n)]. We will use h1 as the heuristic, given in the diagram below.

Initialization: { [S , 4] }

Iteration 1: { [S->A , 3] , [S->G , 12] }

Iteration 2: { [S->A->C , 4] , [S->A->B , 10] , [S->G , 12] }

Iteration 3: { [S->A->C->G , 4] , [S->A->C->D , 6] , [S->A->B , 10] , [S->G , 12] }

Iteration 4: gives the final output as S->A->C->G with a cost of 4.

Explanation of code:

- **FUNCTIONS USED:**
    - ***print(node * temp)*** :
    - ***noofmisplaced(int arr[4][4]):***
    - ***findtile(int no,int x,int y,int arr[4][4]):***
    - ***noawayfromgoal(int arr[4][4]):***
    - ***checkw(int board[4][4]):***
    - ***print1(int board[4][4]):***
    - ***addleft(int board[4][4],int x,int y,int level):***

- ***addright(int board[4][4],int x,int y,int level):***
- ***addup(int board[4][4],int x,int y,int level):***
- ***addown(int board[4][4],int x,int y,int level):***
- ***expand(node * temp):***
- ***getinversionspairs(int arr[]):***
- ***findxpositiofrombottom(int board[4][4]):***
- ***issolvable(int board[4][4]):***
- ***main():***

Source Code:

1. **NUMBER OF MISPLACED TILES**

```
//AI ASSIGNMENT

//TEAM MEMBERS:-

//(1) ANIRUDH KANNAN V P (201601004)

//(2) SAHITHI KRISHNA KOTTE (201601045)



#include <time.h>

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


int totalnoofnodes=0;


//TOTAL TIME
```

```
clock_t start,end;


//NODE STRUCTURE

typedef struct node{

            int cost,level;

            int board[4][4];

            int visited;

            struct node * next;

   struct node * parent;

} node;




struct node * closed;

struct node * open;

struct node *rearopen;

struct node *rearclosed;

struct node *headopen;

struct node *headclosed;

struct node *rearclosed1;


//NO OF MISPLACED 1ST HEURISTIC

int noofmisplaced(int arr[4][4]){

            int completed[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,0}};

            int i,j,c=0;
            for (i = 0; i < 4; ++i)

            {

                        for (j = 0; j < 4; ++j)

                        {

                                    if(completed[i][j]!=arr[i][j])

                                                c+=1;

                        }

            }

            return c;

}




node * newnode;


//CHECKING WHETHER DUPLICATE EXISTS BY CHECKING MATRIX

bool checkw(int board[4][4]){

   node * temp=headclosed;

   while(temp!=NULL){

      int k=0,j,i;

      for(i=0;i<4;i++){

         for(j=0;j<4;j++){

            if(board[i][j]==temp->board[i][j]){

               k++;

            }
            }
        }
        if(k==16)
            return true;
        temp=temp->next;
    }
return false;
}
//PRINTING THE BOARD
void print1(int board[4][4]){
    int i,j;


    printf("\n-----------------\n");


    for(i=0;i<4;i++){
        for(j=0;j<4;j++){
            printf("%d ",board[i][j]);
        }
        printf("\n");
    }


    printf("\n-----------------\n");
return;
}
```

```
void printc(){
 int completed[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,0}},i,j;
            for (i = 0; i < 4; ++i)
            {
                        for (j = 0; j < 4; ++j)
                        {
                                    printf("%d ",completed[i][j]);
                        }
                        printf("\n");
            }
return;
}


//ADD LEFT TILE.
void addleft(int board[4][4],int x,int y,int level,node * temp){
    int i,j,stemp;
    if(y==0)return;
    newnode=(node*)malloc(sizeof(node));
        for(i=0;i<4;i++){
            for(j=0;j<4;j++){
                newnode->board[i][j]=board[i][j];
            }
         }
         stemp=newnode->board[x][y];
         newnode->board[x][y]=newnode->board[x][y-1];

         newnode->board[x][y-1]=stemp;

         newnode->level=level+1;

         newnode->cost=noofmisplaced(newnode->board)+newnode->level;

         newnode->next=NULL;

         newnode->parent=temp;

if(checkw(newnode->board))

    return;

totalnoofnodes+=1;

node *temphead=headopen;

node *temphead1=headopen;

int yu=0;

while(temphead!=NULL){

        yu+=1;

        //HIGHER LIMIT TO AVOID SEGMENTATION ERROR

       if(yu==237){

        printc();

        end=clock();

        double totaltime=((double)(end-start))/CLOCKS_PER_SEC;

        printf("TOTAL TIME IS:- %lf\n",totaltime);

        printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);

        exit(0);

       }

    if(temphead->cost>newnode->cost && temphead==headopen){
        newnode->next=temphead;
        headopen=newnode;
    }
    if(temphead->cost<=newnode->cost){;
            temphead1=temphead;
            temphead=temphead->next;
    }
    else{
            temphead1->next=newnode;
            newnode->next=temphead;
            print1(newnode->board);
            return;
    }
}
temphead1->next=newnode;
newnode->next=NULL;
print1(newnode->board);
return;
}


//ADD RIGHTTILE
void addright(int board[4][4],int x,int y,int level,node * temp){
    int i,j,stemp;
    if(y==3)return;
    newnode=(node*)malloc(sizeof(node));
        for(i=0;i<4;i++){
            for(j=0;j<4;j++){
                newnode->board[i][j]=board[i][j];
            }
         }
         stemp=newnode->board[x][y];
         newnode->board[x][y]=newnode->board[x][y+1];
         newnode->board[x][y+1]=stemp;
         newnode->level=level+1;
         newnode->cost=noofmisplaced(newnode->board)+newnode->level;
         newnode->next=NULL;
         newnode->parent=temp;
if(checkw(newnode->board))
    return;
totalnoofnodes+=1;
node *temphead=headopen;
node *temphead1=headopen;
int yu=0;
while(temphead!=NULL){
        yu+=1;
        //HIGHER LIMIT TO AVOID SEGMENTATION ERROR
       if(yu==237){
        printc();
        end=clock();

        double totaltime=((double)(end-start))/CLOCKS_PER_SEC;

        printf("TOTAL TIME IS:- %lf\n",totaltime);

        printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);

        exit(0);

       }

    if(temphead->cost>newnode->cost && temphead==headopen){

        newnode->next=temphead;

        headopen=newnode;

    }

    if(temphead->cost<=newnode->cost){

            temphead1=temphead;

            temphead=temphead->next;

    }

    else{

            temphead1->next=newnode;

            newnode->next=temphead;

            print1(newnode->board);

            return;

    }

}

temphead1->next=newnode;

newnode->next=NULL;

newnode->next=NULL;
```

```
print1(newnode->board);

return;

}




//ADD UP TILE

void addup(int board[4][4],int x,int y,int level,node * temp){

  int i,j,stemp;

  if(x==0)return;

  newnode=(node*)malloc(sizeof(node));

    for(i=0;i<4;i++){

      for(j=0;j<4;j++){

        newnode->board[i][j]=board[i][j];

      }

     }

     stemp=newnode->board[x][y];

     newnode->board[x][y]=newnode->board[x-1][y];

     newnode->board[x-1][y]=stemp;

     newnode->level=level+1;

     newnode->cost=noofmisplaced(newnode->board)+newnode->level;

     newnode->next=NULL;

     newnode->parent=temp;

if(checkw(newnode->board))

  return;
```

```
totalnoofnodes+=1;

node *temphead=headopen;

node *temphead1=headopen;

int yu=0;

while(temphead!=NULL){

        yu+=1;

        //HIGHER LIMIT TO AVOID SEGMENTATION ERROR

       if(yu==237){

        printc();

        end=clock();

        double totaltime=((double)(end-start))/CLOCKS_PER_SEC;

        printf("TOTAL TIME IS:- %lf\n",totaltime);

        printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);

        exit(0);

       }

    if(temphead->cost>newnode->cost && temphead==headopen){

        newnode->next=temphead;

        headopen=newnode;

    }

    if(temphead->cost<=newnode->cost){

            temphead1=temphead;

            temphead=temphead->next;

    }

    else{
            temphead1->next=newnode;

            newnode->next=temphead;

            print1(newnode->board);

            return;

    }

}

temphead1->next=newnode;

newnode->next=NULL;

print1(newnode->board);

return;

}


//ADD DOWN TILE

void addown(int board[4][4],int x,int y,int level,node * temp){

    int i,j,stemp;

    if(x==3)return;

    newnode=(node*)malloc(sizeof(node));

        for(i=0;i<4;i++){

            for(j=0;j<4;j++){

                newnode->board[i][j]=board[i][j];

            }

          }

         stemp=newnode->board[x][y];

         newnode->board[x][y]=newnode->board[x+1][y];
         newnode->board[x+1][y]=stemp;

         newnode->level=level+1;

         newnode->cost=noofmisplaced(newnode->board)+newnode->level;

         newnode->next=NULL;

         newnode->parent=temp;

if(checkw(newnode->board))

    return;

totalnoofnodes+=1;

node *temphead=headopen;

node *temphead1=headopen;

int yu=0;

while(temphead!=NULL){

        yu+=1;

        //HIGHER LIMIT TO AVOID SEGMENTATION ERROR

       if(yu==237){

        printc();

        end=clock();

        double totaltime=((double)(end-start))/CLOCKS_PER_SEC;

        printf("TOTAL TIME IS:- %lf\n",totaltime);

        printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);

        exit(0);

       }

    if(temphead->cost>newnode->cost && temphead==headopen){

        newnode->next=temphead;
        headopen=newnode;
    }
    if(temphead->cost<=newnode->cost){
            temphead1=temphead;
            temphead=temphead->next;
    }
    else{
            temphead1->next=newnode;
            newnode->next=temphead;
            print1(newnode->board);
            return;
    }
}
temphead1->next=newnode;
newnode->next=NULL;
print1(newnode->board);
return;
}


// EXPANDING NODES CHILDREN
void expand(node * temp){
    int board[4][4],i,j,x,y,stemp;
    for(i=0;i<4;i++){
        for(j=0;j<4;j++){
            board[i][j]=temp->board[i][j];

            if(board[i][j]==0){

                x=i;

                y=j;

            }

        }

    }

    //HIGHER LIMIT TO AVOID SEGMENTATION ERROR

    if(totalnoofnodes==123){

        printc();

        end=clock();

        double totaltime=((double)(end-start))/CLOCKS_PER_SEC;

        printf("TOTAL TIME IS:- %lf\n",totaltime);

        printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);

        exit(0);

    }

    addleft(board,x,y,temp->level,temp);

    addright(board,x,y,temp->level,temp);

    addown(board,x,y,temp->level,temp);

return;

}


//CHECKING WHETHER 2 MATRIX ARE EQUAL

bool checkmatrixn(node *a,node *b){
```

```
int i,j,k=0;

for(i=0;i<4;i++){

    for(j=0;j<4;j++){

        if(a->board[i][j]==b->board[i][j]){

            k++;

        }

    }

}

if(k==16)

    return true;

return false;

}


//TO CHECK WHETHER SOLUTION EXISTS OR NOT

int getinversionspairs(int arr[]){

    int invpairs_count=0,i,j;

    for(i=0;i<15;i++){

        for(j=i+1;j<16;j++){

            if(arr[j] && arr[i] && arr[i]>arr[j])

                invpairs_count+=1;

        }

    }

return invpairs_count;

}
```

```
//TO CHECK WHETHER SOLUTION EXISTS OR NOT

int findxpositiofrombottom(int board[4][4]){

    int i,j;

    for(i=3;i>=0;i--){

        for(j=3;j>=0;j--){

            if(board[i][j]==0)

                return 4-i;

        }

    }

}


//TO CHECK WHETHER SOLUTION EXISTS OR NOT

bool issolvable(int board[4][4]){

int invcount=getinversionspairs((int *) board);

int position=findxpositiofrombottom(board);

if(position & 1)

    return !(invcount & 1);

else

    return invcount & 1;

}



int main(){
```

```
//clock_t start,end;

start=clock();

double totaltime;

        //int initial[4][4]={{1,0,3,4},{6,2,7,8},{5,10,11,12},{9,13,14,15}};

int initial[4][4],i,j;

printf("ENTER THE INITIAL MATRIX TO COMPUTE THE SOLUTION USING A* ALGORITHM: \n");

for(i=0;i<4;i++){

   for(j=0;j<4;j++){

      scanf("%d",&initial[i][j]);

   }

 }

if(!issolvable(initial)){

   printf("NO SOLUTION EXISTS\n");

   end=clock();

   totaltime=((double)(end-start))/CLOCKS_PER_SEC;

   printf("TOTAL TIME IS:- %lf\n",totaltime);

   printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);

   return 0;

}

printf("\n\n\nSOLUTION IS:-\n\n\n");

closed=(node*)malloc(sizeof(node));

rearclosed=closed;

headclosed=closed;

open=(node*)malloc(sizeof(node));
```

```
open->cost=noofmisplaced(initial)+0;

open->level=0;

open->parent=NULL;

for(i=0;i<4;i++){

    for(j=0;j<4;j++){

        open->board[i][j]=initial[i][j];

    }

}

open->next=NULL;

headopen=open;

rearopen=open;

while(noofmisplaced(headopen->board)){

    expand(headopen);

    rearclosed1=(node*)malloc(sizeof(node));

    for(i=0;i<4;i++){

        for(j=0;j<4;j++){

            rearclosed1->board[i][j]=headopen->board[i][j];

        }

    }

    rearclosed1->cost=headopen->cost;

    rearclosed1->level=headopen->level;

    rearclosed1->next=NULL;

    rearclosed1->parent=headopen->parent;

    rearclosed->next=rearclosed1;
```

```
        rearclosed=rearclosed1;

        headopen=headopen->next;

}

    end=clock();

    totaltime=((double)(end-start))/CLOCKS_PER_SEC;

    printf("TOTAL TIME IS:- %lf\n",totaltime);

    printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);


return 0;

}
```

2.NUMBER OF MOVES EACH TILE IS AWAY FROM ITS GOAL POSITION

```
//AI ASSIGNMENT

//TEAM MEMBERS:-

//(1) ANIRUDH KANNAN V P (201601004)

//(2) SAHITHI KRISHNA KOTTE (201601045)



#include <time.h>

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>
```

```
int totalnoofnodes=0;


//TOTAL TIME

clock_t start,end;


//NODE STRUCTURE

typedef struct node{

            int cost,level;

            int board[4][4];

            int visited;

            struct node * next;

   struct node * parent;

} node;




struct node * closed;

struct node * open;

struct node *rearopen;

struct node *rearclosed;

struct node *headopen;

struct node *headclosed;

struct node *rearclosed1;
```

```
//FIND THE TILE-(2 ND ) HEURISTIC

int findtile(int no,int x,int y,int arr[4][4]){

   int i,j;

          for (i = 0; i < 4; ++i)

          {

                    for (j = 0; j < 4; ++j)

                    {

                              if(arr[i][j]==no){

                                        return abs(x-i)+abs(y-j);

                              }

                    }

          }

          return 0;

}




// NO OF TILES AWAY FROM GOAL- (2)ND HEURISTIC

int noawayfromgoal(int arr[4][4]){

          int c=0,temp;

          int completed[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,0}};

   int i,j;

          for (i = 0; i<4 ;++i)

          {
```

```
                    for (j = 0; j < 4 ; ++j)
                    {
                              temp=completed[i][j];
                              c+=findtile(temp,i,j,arr);
                    }
          }
          return c;
}

node * newnode;

//CHECKING WHETHER DUPLICATE EXISTS BY CHECKING MATRIX
bool checkw(int board[4][4]){
   node * temp=headclosed;
   while(temp!=NULL){
      int k=0,j,i;
      for(i=0;i<4;i++){
         for(j=0;j<4;j++){
            if(board[i][j]==temp->board[i][j]){
               k++;
            }
         }
      }
      if(k==16)
```

```
            return true;

        temp=temp->next;

    }

return false;

}

//PRINTING THE BOARD

void print1(int board[4][4]){

    int i,j;



    printf("\n-----------------\n");



    for(i=0;i<4;i++){

        for(j=0;j<4;j++){

            printf("%d ",board[i][j]);

        }

        printf("\n");

    }



    printf("\n-----------------\n");

return;

}

void printc(){

 int completed[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,0}},i,j;

             for (i = 0; i < 4; ++i)
            {
                        for (j = 0; j < 4; ++j)
                        {
                                    printf("%d ",completed[i][j]);
                        }
                        printf("\n");
            }
return;
}

//ADD LEFT TILE.
void addleft(int board[4][4],int x,int y,int level,node * temp){
    int i,j,stemp;
    if(y==0)return;
    newnode=(node*)malloc(sizeof(node));
        for(i=0;i<4;i++){
            for(j=0;j<4;j++){
                newnode->board[i][j]=board[i][j];
            }
         }
         stemp=newnode->board[x][y];
         newnode->board[x][y]=newnode->board[x][y-1];
         newnode->board[x][y-1]=stemp;
         newnode->level=level+1;
         newnode->cost=noawayfromgoal(newnode->board)+newnode->level;

         newnode->next=NULL;

         newnode->parent=temp;

if(checkw(newnode->board))

    return;

totalnoofnodes+=1;

node *temphead=headopen;

node *temphead1=headopen;

int yu=0;

while(temphead!=NULL){

        yu+=1;

        //HIGHER LIMIT TO AVOID SEGMENTATION ERROR

       if(yu==237){

        printc();

        end=clock();

        double totaltime=((double)(end-start))/CLOCKS_PER_SEC;

        printf("TOTAL TIME IS:- %lf\n",totaltime);

        printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);

        exit(0);

       }

    if(temphead->cost>newnode->cost && temphead==headopen){

        newnode->next=temphead;

        headopen=newnode;

    }
    if(temphead->cost<=newnode->cost){;
            temphead1=temphead;
            temphead=temphead->next;
    }
    else{
            temphead1->next=newnode;
            newnode->next=temphead;
            print1(newnode->board);
            return;
    }
}
temphead1->next=newnode;
newnode->next=NULL;
print1(newnode->board);
return;
}


//ADD RIGHTTILE
void addright(int board[4][4],int x,int y,int level,node * temp){
    int i,j,stemp;
    if(y==3)return;
    newnode=(node*)malloc(sizeof(node));
        for(i=0;i<4;i++){
            for(j=0;j<4;j++){
                newnode->board[i][j]=board[i][j];

            }

          }

         stemp=newnode->board[x][y];

         newnode->board[x][y]=newnode->board[x][y+1];

         newnode->board[x][y+1]=stemp;

         newnode->level=level+1;

         newnode->cost=noawayfromgoal(newnode->board)+newnode->level;

         newnode->next=NULL;

         newnode->parent=temp;

if(checkw(newnode->board))

    return;

totalnoofnodes+=1;

node *temphead=headopen;

node *temphead1=headopen;

int yu=0;

while(temphead!=NULL){

        yu+=1;

        //HIGHER LIMIT TO AVOID SEGMENTATION ERROR

       if(yu==237){

        printc();

        end=clock();

        double totaltime=((double)(end-start))/CLOCKS_PER_SEC;

        printf("TOTAL TIME IS:- %lf\n",totaltime);
        printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);
        exit(0);
       }
    if(temphead->cost>newnode->cost && temphead==headopen){
        newnode->next=temphead;
        headopen=newnode;
    }
    if(temphead->cost<=newnode->cost){
            temphead1=temphead;
            temphead=temphead->next;
    }
    else{
            temphead1->next=newnode;
            newnode->next=temphead;
            print1(newnode->board);
            return;
    }
}
temphead1->next=newnode;
newnode->next=NULL;
newnode->next=NULL;
print1(newnode->board);
return;
}
```

```
//ADD UP TILE

void addup(int board[4][4],int x,int y,int level,node * temp){

    int i,j,stemp;

    if(x==0)return;

    newnode=(node*)malloc(sizeof(node));

        for(i=0;i<4;i++){

            for(j=0;j<4;j++){

                newnode->board[i][j]=board[i][j];

            }

          }

         stemp=newnode->board[x][y];

         newnode->board[x][y]=newnode->board[x-1][y];

         newnode->board[x-1][y]=stemp;

         newnode->level=level+1;

         newnode->cost=noawayfromgoal(newnode->board)+newnode->level;

         newnode->next=NULL;

         newnode->parent=temp;

if(checkw(newnode->board))

    return;

totalnoofnodes+=1;

node *temphead=headopen;

node *temphead1=headopen;
```

```
int yu=0;
while(temphead!=NULL){
        yu+=1;
        //HIGHER LIMIT TO AVOID SEGMENTATION ERROR
       if(yu==237){
        printc();
        end=clock();
        double totaltime=((double)(end-start))/CLOCKS_PER_SEC;
        printf("TOTAL TIME IS:- %lf\n",totaltime);
        printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);
        exit(0);
       }
    if(temphead->cost>newnode->cost && temphead==headopen){
        newnode->next=temphead;
        headopen=newnode;
    }
    if(temphead->cost<=newnode->cost){
            temphead1=temphead;
            temphead=temphead->next;
    }
    else{
            temphead1->next=newnode;
            newnode->next=temphead;
            print1(newnode->board);
```

```
          return;
    }
}
temphead1->next=newnode;
newnode->next=NULL;
print1(newnode->board);
return;
}


//ADD DOWN TILE
void addown(int board[4][4],int x,int y,int level,node * temp){
    int i,j,stemp;
    if(x==3)return;
    newnode=(node*)malloc(sizeof(node));
        for(i=0;i<4;i++){
            for(j=0;j<4;j++){
                newnode->board[i][j]=board[i][j];
            }
          }
         stemp=newnode->board[x][y];
         newnode->board[x][y]=newnode->board[x+1][y];
         newnode->board[x+1][y]=stemp;
         newnode->level=level+1;
         newnode->cost=noawayfromgoal(newnode->board)+newnode->level;
        newnode->next=NULL;

        newnode->parent=temp;

if(checkw(newnode->board))

    return;

totalnoofnodes+=1;

node *temphead=headopen;

node *temphead1=headopen;

int yu=0;

while(temphead!=NULL){

        yu+=1;

        //HIGHER LIMIT TO AVOID SEGMENTATION ERROR

       if(yu==237){

        printc();

        end=clock();

        double totaltime=((double)(end-start))/CLOCKS_PER_SEC;

        printf("TOTAL TIME IS:- %lf\n",totaltime);

        printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);

        exit(0);

       }

    if(temphead->cost>newnode->cost && temphead==headopen){

        newnode->next=temphead;

        headopen=newnode;

    }

    if(temphead->cost<=newnode->cost){
            temphead1=temphead;

            temphead=temphead->next;

    }

    else{

            temphead1->next=newnode;

            newnode->next=temphead;

            print1(newnode->board);

            return;

    }

}

temphead1->next=newnode;

newnode->next=NULL;

print1(newnode->board);

return;

}


// EXPANDING NODES CHILDREN

void expand(node * temp){

    int board[4][4],i,j,x,y,stemp;

    for(i=0;i<4;i++){

        for(j=0;j<4;j++){

            board[i][j]=temp->board[i][j];

            if(board[i][j]==0){

                x=i;
            y=j;
         }
      }
   }
   //HIGHER LIMIT TO AVOID SEGMENTATION ERROR
   if(totalnoofnodes==123){
      printc();
      end=clock();
      double totaltime=((double)(end-start))/CLOCKS_PER_SEC;
      printf("TOTAL TIME IS:- %lf\n",totaltime);
      printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);
      exit(0);
   }
   addleft(board,x,y,temp->level,temp);
   addright(board,x,y,temp->level,temp);
   addown(board,x,y,temp->level,temp);
return;
}


//CHECKING WHETHER 2 MATRIX ARE EQUAL
bool checkmatrixn(node *a,node *b){
int i,j,k=0;
for(i=0;i<4;i++){
   for(j=0;j<4;j++){
        if(a->board[i][j]==b->board[i][j]){

            k++;

        }

    }

}

if(k==16)

    return true;

return false;

}


//TO CHECK WHETHER SOLUTION EXISTS OR NOT

int getinversionspairs(int arr[]){

    int invpairs_count=0,i,j;

    for(i=0;i<15;i++){

        for(j=i+1;j<16;j++){

            if(arr[j] && arr[i] && arr[i]>arr[j])

                invpairs_count+=1;

        }

    }

return invpairs_count;

}


//TO CHECK WHETHER SOLUTION EXISTS OR NOT

int findxpositiofrombottom(int board[4][4]){
    int i,j;

    for(i=3;i>=0;i--){

        for(j=3;j>=0;j--){

            if(board[i][j]==0)

                return 4-i;

        }

    }

}


//TO CHECK WHETHER SOLUTION EXISTS OR NOT

bool issolvable(int board[4][4]){

int invcount=getinversionspairs((int *) board);

int position=findxpositiofrombottom(board);

if(position & 1)

    return !(invcount & 1);

else

    return invcount & 1;

}




int main(){

    //clock_t start,end;

    start=clock();

    double totaltime;
```

```
        //int initial[4][4]={{1,0,3,4},{6,2,7,8},{5,10,11,12},{9,13,14,15}};
int initial[4][4],i,j;
printf("ENTER THE INITIAL MATRIX TO COMPUTE THE SOLUTION USING A* ALGORITHM: \n");
for(i=0;i<4;i++){
  for(j=0;j<4;j++){
    scanf("%d",&initial[i][j]);
  }
 }
if(!issolvable(initial)){
  printf("NO SOLUTION EXISTS\n");
  end=clock();
  totaltime=((double)(end-start))/CLOCKS_PER_SEC;
  printf("TOTAL TIME IS:- %lf\n",totaltime);
  printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);
  return 0;
}
printf("\n\n\nSOLUTION IS:-\n\n\n");
closed=(node*)malloc(sizeof(node));
rearclosed=closed;
headclosed=closed;
open=(node*)malloc(sizeof(node));
open->cost=noawayfromgoal(initial)+0;
open->level=0;
open->parent=NULL;
```

```
    for(i=0;i<4;i++){
        for(j=0;j<4;j++){
            open->board[i][j]=initial[i][j];
        }
    }
    open->next=NULL;
    headopen=open;
    rearopen=open;
    while(noawayfromgoal(headopen->board)){
        expand(headopen);
        rearclosed1=(node*)malloc(sizeof(node));
        for(i=0;i<4;i++){
            for(j=0;j<4;j++){
                rearclosed1->board[i][j]=headopen->board[i][j];
            }
        }
        rearclosed1->cost=headopen->cost;
        rearclosed1->level=headopen->level;
        rearclosed1->next=NULL;
        rearclosed1->parent=headopen->parent;
        rearclosed->next=rearclosed1;
        rearclosed=rearclosed1;
        headopen=headopen->next;
}
```

```
    end=clock();

    totaltime=((double)(end-start))/CLOCKS_PER_SEC;

    printf("TOTAL TIME IS:- %lf\n",totaltime);

    printf("TOTAL NO OF NODES IS: %d\n",totalnoofnodes);


return 0;

}
```

Conclusion:

A* algorithm is the best known form of Best First Search. It avoids expanding paths that are already expensive, but expands most promising paths first.

Complexity:

The time complexity of A* depends on the heuristic. The complexity is $O(b^d)$ where d is the depth of the solution(the shortest path) and b is the branching factor(the average number of successors per state).

Applications:

A* is commonly used for the common pathfinding problem in applications such as games, but was originally designed as a general graph traversal algorithm. It finds applications to diverse problems, including the problem of parsing using stochastic grammars in NLP.