**8 PUZZLE SOLVER**

**A**

**Mini Project Report**

**Submitted for the course**

**Design and Analysis of Algorithms Lab**

**Semester-IV, CSE-B**

**By**

**JAIDEV SHARMA 1602-18-733-078**

**SAAD SHAIK 1602-18-733-093**



**Department of Computer Science and Engineering**

**VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)**

**9-5-81, Ibrahimbagh, Hyderabad-31**

**2019-20**

**ACKNOWLEDGEMENT**

We take this opportunity with pride and enormous gratitude, to express the deeply embedded feeling and gratefulness to our respectable guide **Mrs. V Sireesha,** Department of Computer Science and Engineering. Whose guidance was unforgettable and innovative ideas as well as his constructive suggestions has made the presentation of my thesis a grand success.

Finally at last but not least express our heart full thanks to the management of our college, **Vasavi College of Engineering** for providing the necessary arrangements and support to complete my project work successively.

**TABLE OF CONTENTS**

**S.no. Topic name Pg.no.**

**1. Project Abstract 5**

**2. Description 6-7**

**3. Code 8-15**

**4. Output 16-17**

**5. Constraints &Test Cases 18-21**

**6. Conclusion 22**

**7. References 23**

**PROJECT -ABSTRACT**

**8 PUZZLE SOLVER**

This web application allows you to view a graphical representation of a range of different graph search algorithms, whilst solving your choice of 8-puzzle problems.

**Search Algorithm**

N-Puzzle supports five different Graph-based Search Algorithms. The first three are *Uninformed* Search Algorithms:

* Breadth-first Search
* Depth-first Search
* Iterative Deepening Search

The other two are *Informed* Search Algorithms:

* A\* Search
* Greedy Search

If you choose an Informed Search Algorithm, then you will also need to select a Heuristic Function.

**Heuristic Function**

N-Puzzle supports three different Heuristic Functions:

* Euclidean Distance
* Manhattan Distance (City-Block distance)
* Tiles Out-of-place

**Single-Step or Burst Mode**

N-Puzzle can be used in two modes. The default is Single-Step mode, which allows you to 'rewind' a search, one step at a time. This is useful for getting a better understanding of how a Search Algorithm works.

The other mode is Burst Mode. Once started, Burst Mode continues running a search until the goal state has been found. A Burst Mode search can be paused, but cannot be 'rewound'.

**CODE->**

**/\* DESIGN & ANALYSIS OF ALGORITHMS MINIPROJECT**

**8 PUZZLE SOLVER**

**STRATEGY USED:A\* search with heuristic approach**

**BY:JAIDEV SHARMA(1602-18-733-078)**

**SAAD SHAIK(1602-18-733-093)\**

**\*/**

**//Declaration of header files**

**#include<stdio.h>**

**#include<malloc.h>**

**#include<string.h>**

**typedef struct node\_array**

**{**

**int a[3][3];**

**int depth;**

**int f;**

**char text[11];**

**struct node\_array \*ch1;**

**struct node\_array \*ch2;**

**struct node\_array \*ch3;**

**struct node\_array \*ch4;**

**struct node\_array \*parent;**

**struct node\_array \*next;**

**} node;**

**node \*front=NULL,\*rear=NULL,\*x,\*front\_list=NULL,\*rear\_list=NULL;**

**int up[3][3],dn[3][3],rt[3][3],lt[3][3],goal[3][3],depth\_flag=0,f\_count=0;**

**//function to initialise array elements to 0**

**void set\_zero(int a[3][3])**

**{**

**int i,j;**

**for(i=0;i<3;i++)**

**for(j=0;j<3;j++)**

**a[i][j]=0;**

**}**

**//function to copy the contents of an array to another**

**void copy\_array(int a[3][3],int b[3][3])//copy b to a**

**{**

**int i,j;**

**for(i=0;i<3;i++)**

**for(j=0;j<3;j++)**

**a[i][j]=b[i][j];**

**}**

**//function to check whether 2 arrays are equal or not**

**int is\_equal(int a[3][3],int b[3][3])**

**{**

**int i,j,flag=1;**

**for(i=0;i<3;i++)**

**for(j=0;j<3;j++)**

**if(a[i][j]!=b[i][j])**

**flag=0;**

**return flag;**

**}**

**//function to swap 2 elements**

**void swap(int \*a,int \*b)**

**{**

**int temp;**

**temp=\*a;**

**\*a=\*b;**

**\*b=temp;**

**}**

**//function to print elements of array**

**void print\_array(int a[3][3])**

**{**

**int i,j;**

**for(i=0;i<3;i++)**

**{ for(j=0;j<3;j++)**

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

**printf("\n");**

**}**

**}**

**//function to check the empty space block**

**int check\_zero(int a[3][3])//returns 1 if 0**

**{**

**int i,j,flag=1;**

**for(i=0;i<3;i++)**

**{ for(j=0;j<3;j++)**

**if(a[i][j]!=0)**

**flag=0;**

**}**

**return flag;**

**}**

**int check\_list(node \*nd)//returns 0 if it doesnt match the list**

**{**

**node \*new\_node=NULL;**

**if(front\_list==NULL)**

**return 0;**

**else**

**{**

**new\_node=front\_list;**

**while(new\_node!=NULL)**

**{**

**if(is\_equal(new\_node->a,nd->a)==1)**

**return 1;**

**new\_node=new\_node->next;**

**}**

**return 0;**

**}**

**}**

**//function to insert element into a queue**

**void insert\_queue(node \*nd)**

**{**

**nd->next=NULL;**

**if(front==NULL)**

**{**

**front=nd;**

**rear=nd;**

**}**

**else**

**{**

**rear->next=nd;**

**rear=nd;**

**rear->next=NULL;**

**}**

**}**

**//function to insert element in a list**

**void insert\_list(node \*nd)**

**{**

**nd->next=NULL;**

**if(front\_list==NULL)**

**{**

**front\_list=nd;**

**rear\_list=nd;**

**}**

**else**

**{**

**rear\_list->next=nd;**

**rear\_list=nd;**

**rear\_list->next=NULL;**

**}**

**}**

**//function to return the absolute value of a number**

**int mod(int a)**

**{**

**if(a>=0)**

**return a;**

**else**

**return (-1\*a);**

**}**

**//function to perform heuristic calculate**

**int calculate\_heuristic(node \*nd)**

**{**

**int i,j,k,l,h=0;**

**for(i=0;i<=2;i++)**

**{**

**for(j=0;j<=2;j++)**

**{**

**if(nd->a[i][j]!=0)**

**{for(k=0;k<=2;k++)**

**{**

**for(l=0;l<=2;l++)**

**{**

**if(nd->a[i][j]==goal[k][l])**

**goto xy;**

**}**

**}**

**xy:**

**h+=(mod(i-k)+mod(j-l));}**

**}**

**}**

**return h;**

**}**

**//function to decide the next move**

**void next\_move(node \*nd)**

**{**

**int i,j,x,y;**

**for(i=0;i<3;i++)**

**{**

**for(j=0;j<3;j++)**

**{**

**if(nd->a[i][j]==0)**

**goto label;**

**}**

**}**

**label:**

**x=i;**

**y=j;**

**if(y+1 >2)**

**nd->ch4=NULL;**

**else**

**{**

**nd->ch4=(node\*)malloc(sizeof(node));**

**copy\_array(nd->ch4->a,nd->a);**

**swap(&nd->ch4->a[x][y],&nd->ch4->a[x][y+1]);**

**if(check\_list(nd->ch4)==1)**

**{**

**nd->ch4=NULL;**

**free(nd->ch4);**

**}**

**else**

**{**

**nd->ch4->parent=nd;**

**nd->ch4->depth=nd->depth+1;**

**strcpy(nd->ch4->text,"move right");**

**nd->ch4->f=nd->depth+calculate\_heuristic(nd->ch4)+1;**

**insert\_list(nd->ch4);**

**insert\_queue(nd->ch4);**

**}**

**}**

**if(y-1 <0)**

**nd->ch3=NULL;**

**else**

**{**

**nd->ch3=(node\*)malloc(sizeof(node));**

**copy\_array(nd->ch3->a,nd->a);**

**swap(&nd->ch3->a[x][y],&nd->ch3->a[x][y-1]);**

**if(check\_list(nd->ch3)==1)**

**{**

**nd->ch3=NULL;**

**free(nd->ch3);**

**}**

**else**

**{**

**nd->ch3->parent=nd;**

**strcpy(nd->ch3->text,"move left");**

**nd->ch3->depth=nd->depth+1;**

**nd->ch3->f=nd->depth+calculate\_heuristic(nd->ch3)+1;**

**insert\_list(nd->ch3);**

**insert\_queue(nd->ch3);**

**}**

**}**

**if(x+1 >2)**

**nd->ch2=NULL;**

**else**

**{**

**nd->ch2=(node\*)malloc(sizeof(node));**

**copy\_array(nd->ch2->a,nd->a);**

**swap(&nd->ch2->a[x][y],&nd->ch2->a[x+1][y]);**

**if(check\_list(nd->ch2)==1)**

**{**

**nd->ch2=NULL;**

**free(nd->ch2);**

**}**

**else**

**{**

**nd->ch2->parent=nd;**

**strcpy(nd->ch2->text,"move down");**

**nd->ch2->depth=nd->depth+1;**

**nd->ch2->f=nd->depth+calculate\_heuristic(nd->ch2)+1;**

**insert\_list(nd->ch2);**

**insert\_queue(nd->ch2);**

**}**

**}**

**if(x-1 <0)**

**nd->ch1=NULL;**

**else**

**{**

**nd->ch1=(node\*)malloc(sizeof(node));**

**copy\_array(nd->ch1->a,nd->a);**

**swap(&nd->ch1->a[x][y],&nd->ch1->a[x-1][y]);**

**if(check\_list(nd->ch1)==1)**

**{**

**nd->ch1=NULL;**

**free(nd->ch1);**

**}**

**else**

**{**

**nd->ch1->parent=nd;**

**strcpy(nd->ch1->text,"move up");**

**nd->ch1->depth=nd->depth+1;**

**nd->ch1->f=nd->depth+calculate\_heuristic(nd->ch1)+1;**

**insert\_list(nd->ch1);**

**insert\_queue(nd->ch1);**

**}**

**}**

**}**

**int is\_goal(node \*nd) //return 1 if goal**

**{**

**return is\_equal(nd->a,goal);**

**}**

**//delete element from queue**

**node \*pop\_queue()**

**{**

**node \*nd;**

**nd=front;**

**front=front->next;**

**if(front==NULL)**

**rear=NULL;**

**return nd;**

**}**

**void final\_print(node \*nd)**

**{**

**while(nd!=NULL)**

**{**

**printf("\n");**

**print\_array(nd->a);**

**printf("\n\n%s\n",nd->text);**

**nd=nd->parent;**

**f\_count++;**

**}**

**printf("\nNo of steps %d",f\_count-1);**

**}**

**int count\_queue()**

**{**

**int count=0;**

**node \*temp;**

**temp=front;**

**while(temp!=NULL)**

**{**

**temp=temp->next;**

**count++;**

**}**

**return count;**

**}**

**void arrange()**

**{**

**node \*t1,\*t2,\*t3;**

**int i,j;**

**i=count\_queue();**

**j=i;**

**int k = 0, l = 1;**

**for(k=0;k<i;k++)**

**{**

**t1=front;**

**t2=front->next;**

**for(l=1;l<j;l++)**

**{**

**if(t1->next->f>t2->next->f)**

**{**

**t3=t2;**

**t1->next=t2->next;**

**t3->next=t3->next->next;**

**t1->next->next=t2;**

**}**

**j--;**

**t1=t1->next;**

**t2=t2->next;**

**}**

**}**

**t1=front;**

**while(t1->next!=NULL)**

**t1=t1->next;**

**rear=t1;**

**rear\_list=t1;**

**}**

**//function to perform A\*star search**

**void A\_star(node \*root)**

**{**

**{**

**node \*nd;**

**insert\_queue(root);**

**insert\_list(root);**

**nd=root;**

**while(1)**

**{**

**nd=front;**

**if(is\_goal(nd)==1)**

**{**

**final\_print(nd);**

**break;**

**}**

**next\_move(nd);**

**arrange();**

**nd=pop\_queue();**

**}**

**}**

**}**

**int main()**

**{**

**node \*n;**

**int i,j;**

**n=(node\*)malloc(sizeof(node));**

**x=(node\*)malloc(sizeof(node));**

**printf("Enter the goal state\n");**

**for(i=0;i<3;i++)**

**for(j=0;j<3;j++)**

**scanf("%d",&n->a[i][j]);**

**n->parent=NULL;**

**n->next=NULL;**

**n->ch1=NULL;**

**n->ch2=NULL;**

**n->ch3=NULL;**

**n->ch4=NULL;**

**n->depth=0;**

**printf("Enter the current state\n");**

**for(i=0;i<3;i++)**

**for(j=0;j<3;j++)**

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

**n->f=calculate\_heuristic(n);**

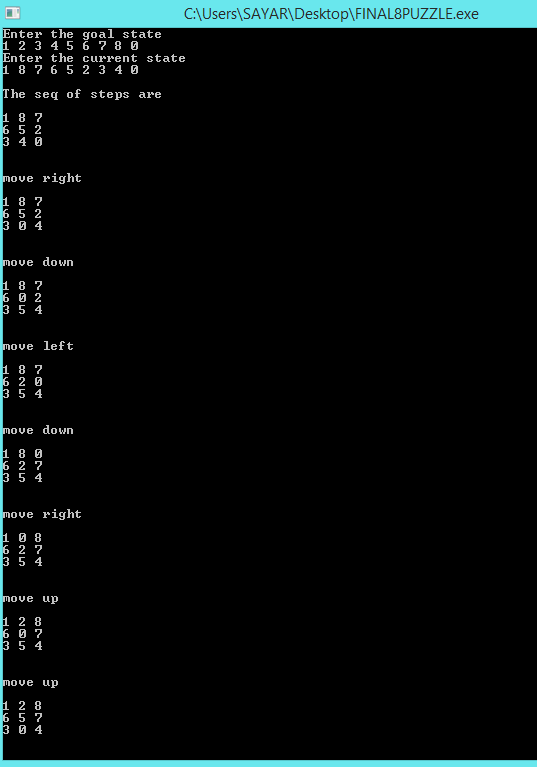
**printf("\nThe seq of steps are\n",f\_count);**

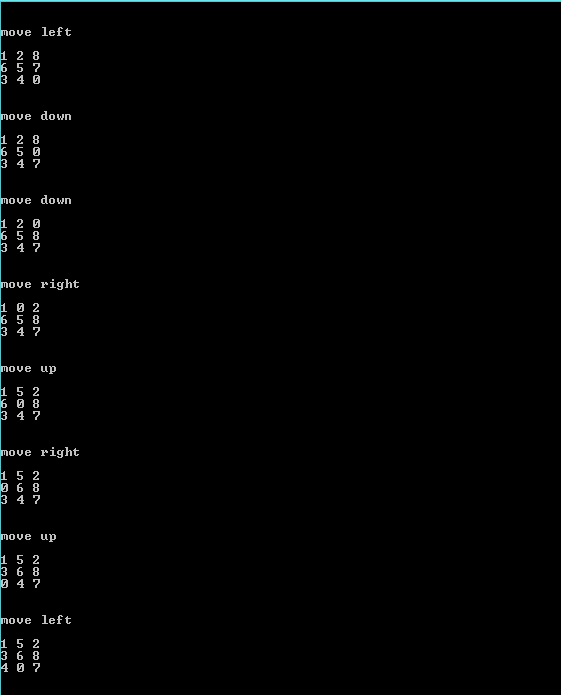
**A\_star(n);**

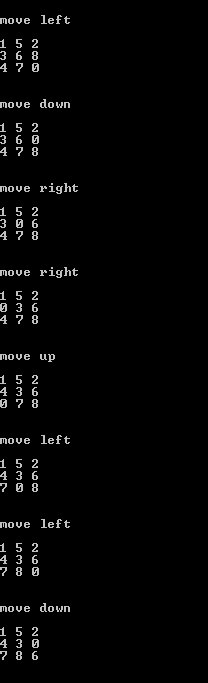
**scanf("%d",&i);**

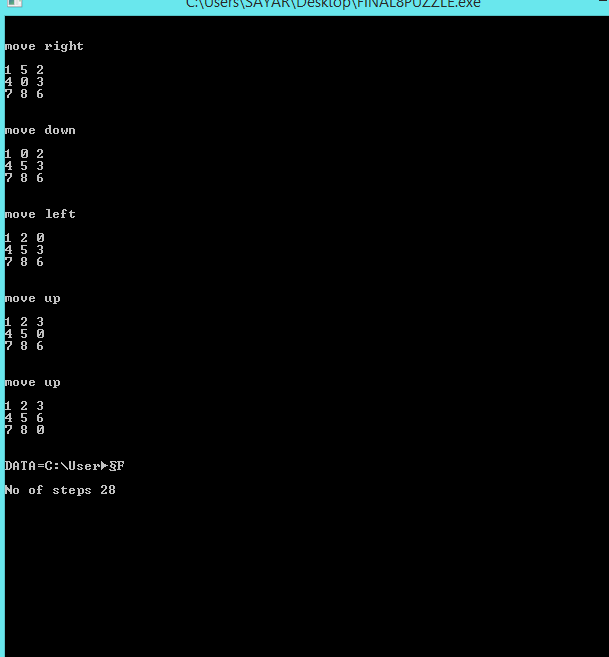
**return 0; }**

**TEST CASES**









**CONCLUSION**

The project deals with puzzle solving. It changes the input array to the target array choosing the most efficient method each case.

**REFERENCES->**

|  |
| --- |
| https://mail.google.com/mail/u/0/images/cleardot.gif  https://mail.google.com/mail/u/0/images/cleardot.gif |
|

* <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>
* <https://www.tutorialspoint.com/c_standard_library/c_function_rand.html>
* TEXTBOOK REFERRED- DESIGN AND ANALYSIS OF ALGORITHMS BY ELLIS HOROWITZ AND SATRAJ SAHANI