Algoritmo A* con resolucion heuristica de Mahantan, para la solucion del
ocho−puzle.

Se puede compilar en Borland C++, y en VC++

```
////////////////////////////////////////////////////////////////////////////////

//////////  METODO A* UTILIZANDO LA ESTIMACION EURISTICA DE MANHATAN
//////////
////////////////////////////////////////////////////////////////////////////////

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define MAX_L 3

struct nodo{
  struct nodo *ant;//apunta al nodo anterior
  unsigned short int puzle[MAX_L][MAX_L];//contiene el puzle
  unsigned short int blanco[2];//tiene la posicion x e y donde esta el
blanco
  struct nodo *sig;//apunta al nodo siguiente
};

void pide_estado(struct nodo **g);
void A_asterisco(struct nodo **g,struct nodo **f);
struct nodo *expansion(struct nodo *g,int d);
void inserta(struct nodo **e,struct nodo **n);
void borrar(struct nodo **e);
struct nodo *evalua_nodo(struct nodo **e,struct nodo **f);//se usa el
metodo de nº de casillas mal colocadas
void ensena_graf(const struct nodo *g);
void retorn_x_y(const struct nodo **f,unsigned short int *i_fin,
    unsigned short int *j_fin,unsigned short int num);


void coge_ini(struct nodo **g);
void coge_fin(struct nodo **g);


void pide_estado(struct nodo **g)
{
 struct nodo *nuevo;
 unsigned short int i,j,a_i,a_j;

 if(!(nuevo=(struct nodo *)malloc(sizeof(struct nodo))))
  printf("no queda memoria \n");
 else
 {
```

```
      printf("   Mete un 0 en la posicion en blanco \n");
 for(i=0;i!=MAX_L;i++)
  for(j=0;j!=MAX_L;j++)
  {
   printf("\n Mete el nº de la posicion [%d,%d]=",i,j);
   scanf("%d",&nuevo->puzle[i][j]);
   if(nuevo->puzle[i][j]==0)
   {
    a_i=i;
    a_j=j;
   }
  }
 nuevo->blanco[0]=a_i;
 nuevo->blanco[1]=a_j;
}
nuevo->ant=NULL;
nuevo->sig=NULL;
   *g=nuevo;
}


void A_asterisco(struct nodo **g,struct nodo **f)
{
 int terminado=0;
 struct nodo *expan=NULL,*aus=NULL,*nuevo=NULL;

 while(! terminado)
 {
  if((*g)->blanco[1]!=MAX_L-1)
  {
   nuevo=expansion(*g,1);
   inserta(&expan,&nuevo);
  }
  if((*g)->blanco[1]!=0)
  {
   nuevo=expansion(*g,2);
   inserta(&expan,&nuevo);
  }
  if((*g)->blanco[0]!=MAX_L-1)
  {
   nuevo=expansion(*g,3);
   inserta(&expan,&nuevo);
  }
  if((*g)->blanco[0]!=0)
  {
   nuevo=expansion(*g,4);
   inserta(&expan,&nuevo);
  }

  aus=evalua_nodo(&expan,f);

  if(aus==NULL)//fin del proceso
```

```
  {
   terminado=1;
   inserta(g,f);//inserta en el grafo el nodo final
  }
  else
   inserta(g,&aus);

  borrar(&expan);
 }
}

struct nodo *expansion(struct nodo *g,int d)
{
 unsigned short int aus=0,i,j;
 struct nodo *n;

 if(!(n=(struct nodo *)malloc(sizeof(struct nodo))))
  printf("no queda memoria \n");
 else
 {
  n->sig=NULL;
  n->ant=NULL;

  for(i=0;i!=MAX_L;i++)
   for(j=0;j!=MAX_L;j++)
    n->puzle[i][j]=g->puzle[i][j];
  i=g->blanco[0];
  j=g->blanco[1];

  n->blanco[0]=i;
  n->blanco[1]=j;

  if(d==1)//movimiento del 0 a la derecha
  {
   aus=n->puzle[i][j+1];
   n->puzle[i][j+1]=0;
   n->blanco[0]=i;
   n->blanco[1]=j+1;
  }
  else
      if(d==2)//movimiento del 0 a la izquierda
   {
       aus=n->puzle[i][j-1];
       n->puzle[i][j-1]=0;
    n->blanco[0]=i;
    n->blanco[1]=j-1;
   }
   else
         if(d==3)//movimiento del 0 abajo
    {
           aus=n->puzle[i+1][j];
          n->puzle[i+1][j]=0;
```

```
    n−>blanco[0]=i+1;
    n−>blanco[1]=j;
   }
   else
          if(d==4)//movimiento del 0 arriba
    {
           aus=n−>puzle[i−1][j];
           n−>puzle[i−1][j]=0;
     n−>blanco[0]=i−1;
     n−>blanco[1]=j;
    }

 n−>puzle[i][j]=aus;
}
return n;
}

void inserta(struct nodo **e,struct nodo **n)
{

if(*e != NULL)
{
 (*e)−>sig=*n;
 (*n)−>ant=*e;
 (*n)−>sig=NULL;
}
*e=*n;
*n=NULL;
}

void borrar(struct nodo **e)
{
struct nodo *borra;

while(*e != NULL)
{
 borra=*e;
 *e=(*e)−>sig;
 free(borra);
}
}


struct nodo *evalua_nodo(struct nodo **e,struct nodo **f)//se usa el
metodo de nº de casillas mal colocadas
{
struct nodo *p_mejor=NULL,*p_mejor_aus=NULL;
unsigned short int i,j,mal,mal_aus=0,mejor=999,opcion,i_f,j_f;

while((*e) != NULL)
{
 mal=0;
```

```
  for(i=0;i!=MAX_L;i++)//comparacion de los nodos
   for(j=0;j!=MAX_L;j++)
    if(((*e)->puzle[i][j] != (*f)->puzle[i][j]) && ((*e)->puzle[i][j] !=
0))
    {////se aplica Manhatan
     retorn_x_y(f,&i_f,&j_f,(*e)->puzle[i][j]);
     mal=mal+(abs(i-i_f)+abs(j-j_f));
    }

  if(mal < mejor)
  {
   mejor=mal;
   p_mejor=*e;// el mejor nodo
  }
  else
   if(mal == mejor)
   {
    mal_aus=mal;
    p_mejor_aus=*e;
   }
  *e=(*e)->ant;
 }

 opcion=0;
 if((mejor==mal_aus) && (mejor != 0))// en el caso de que halla 2 nodos
posibles
 {
  printf("\n\n¡PROBLEMA! Hay dos nodos con solo %d casilla mal
colocadas",mejor);
  printf("\n Pulse ( 1 ) o ( 2 ) para elegir uno de los dos nodos al
azar --> ");
  scanf("%d",&opcion);
        fflush(stdin);
 }
 if(opcion==2)// en el caso de que halla 2 nodos posibles
  p_mejor=p_mejor_aus;

 if(mejor==0)//fin del proceso
  return NULL;
 else
 {
  if(p_mejor->sig != NULL)
  {
   p_mejor->sig->ant=p_mejor->ant;
      p_mejor->sig=NULL;
  }
  if(p_mejor->ant != NULL)
  {
   p_mejor->ant->sig=p_mejor->sig;
   p_mejor->ant=NULL;
  }
  printf("\n se escoge este");
```

```
 ensena_graf(p_mejor);
 return p_mejor;
 }
}

void retorn_x_y(const struct nodo **f,unsigned short int *i_fin,
   unsigned short int *j_fin,unsigned short int num)
{///se pasan i_fin y j_fin  por referencia
 unsigned short int i,j;

 for(i=0;i!=MAX_L;i++)//comparacion de los nodos
  for(j=0;j!=MAX_L;j++)
   if((*f)->puzle[i][j] == num)
   {
    *i_fin=i;
    *j_fin=j;
   }
}


void ensena_graf(const struct nodo *g)
{
 unsigned short int i,j,cont_nodo=0;

 for(;g->ant != NULL;g=g->ant);/// me posiciono en el primer nodo para
luego sacar la lista
         /// en el orden correcto
 while (g != NULL)/// saco el contenido de la lista
 {
  cont_nodo++;
  printf("\n          NODO %d",cont_nodo);
  for(i=0;i!=MAX_L;i++)
  {
   printf("\n");
   for(j=0;j!=MAX_L;j++)
    printf("[%d,%d]=%d <> ",i,j,g->puzle[i][j]);
  }
  g=g->sig;
 }
}

void coge_ini(struct nodo **g)/// para coger el nodo inicial sin meterlo
por teclado
{
 struct nodo *nuevo;

 if(!(nuevo=(struct nodo *)malloc(sizeof(struct nodo))))
  printf("no queda memoria \n");
 else
 {
  nuevo->blanco[0]=1;
  nuevo->blanco[1]=2;
```

```
 nuevo->puzle[0][0]=1;
 nuevo->puzle[0][1]=2;
 nuevo->puzle[0][2]=3;
 nuevo->puzle[1][0]=6;
 nuevo->puzle[1][1]=4;
 nuevo->puzle[1][2]=0;
 nuevo->puzle[2][0]=8;
 nuevo->puzle[2][1]=7;
 nuevo->puzle[2][2]=5;
 }
 nuevo->ant=NULL;
 nuevo->sig=NULL;
   *g=nuevo;
}

void coge_fin(struct nodo **g)/// para coger el nodo final sin meterlo
por teclado
{
 struct nodo *nuevo;

 if(!(nuevo=(struct nodo *)malloc(sizeof(struct nodo))))
  printf("no queda memoria \n");
 else
 {
 nuevo->blanco[0]=1;
 nuevo->blanco[1]=1;

 nuevo->puzle[0][0]=1;
 nuevo->puzle[0][1]=2;
 nuevo->puzle[0][2]=3;
 nuevo->puzle[1][0]=8;
 nuevo->puzle[1][1]=0;
 nuevo->puzle[1][2]=4;
 nuevo->puzle[2][0]=7;
 nuevo->puzle[2][1]=6;
 nuevo->puzle[2][2]=5;
 }
 nuevo->ant=NULL;
 nuevo->sig=NULL;
   *g=nuevo;
}



void main()
{
 struct nodo *grafo=NULL,*final=NULL;
 unsigned short int opcion;

 printf("\n\n          Metodo A* con estimacion heuristica Manhatan
\n\n");
```

```
 printf("\n PULSE 1− Si desea meter el contenido de los nodos por
teclado");
 printf("\n PULSE 2− Si desea usar los nodos del ejemplo de clase \n");
 do
      scanf("%d",&opcion);
 while ((opcion!=1) && (opcion!=2));

 if (opcion==1)
 {
    printf("\n Introduce la tabla del estado inicial \n");
    pide_estado(&grafo);
      printf("\n Introduce la tabla del estado final \n");
       pide_estado(&final);
 }
 else
 {
     coge_ini(&grafo);
     coge_fin(&final);
 }
 A_asterisco(&grafo,&final);///realiza el proceso
 printf("\n\n\n   PROCESO TERMINADO, ESTE ES EL CAMINO HASTA LA
SOLUCION");
 ensena_graf(grafo);
}
```