KHOA CÔNG NGHỆ THÔNG TIN

ĐẠI HỌC BÁCH KHOA-ĐẠI HỌC ĐÀ NẴNG

Bài Thực hành

Trí Tuệ Nhân Tạo

Giáo viên hướng dẫn : Võ Đức Hoàng
Sinh viên thực hiện : Võ Thị Thu Giang
Lớp : 07T4
Nhóm : 9B

Đà Nẵng,11/2010

I) Thuật toán BFS (Best-First-Search)

Best First Search(hay tìm kiếm theo lựa chọn tốt nhất) là một giải thuật tìm kiếm trên đồ thị bằng cách mở rộng nút có triển vọng nhất được lựa chọn theo một luật xác định.

Judea Pearl (sinh năm 1936, nhà triết học và khoa học máy tính) đã mô tả: Best First Search ước lượng độ tốt của một node bằng một "hàm ước lượng heuristic".

Nhiều tác giả đã sử dụng Best First Search để mô tả cách tìm kiếm với heuristic, trong đó cố gắng dự đoán xem đoạn cuối của một đường đi đã gần với lời giải hay chưa, và con đường nào được đánh giá là gần với lời giải hơn sẽ mở rộng trước. Loại tìm kiếm đặc trưng này được gọi là Greedy Best First Search.

Người ta thường dùng hàng đợi ưu tiên (priority queue) để chọn ra trạng thái tốt nhất và mở rộng nó.

II) Thuật toán A*

1) Khái quát chung :

A* là một thuật toán tìm kiếm trong đồ thị và là phiên bản đặc biệt của tìm kiếm theo kiểu BFS.

Thuật toán này tìm một đường đi từ một nút khởi đầu tới một nút đích cho trước (hoặc tới một nút thỏa mãn một điều kiện đích). Thuật toán này sử dụng một "đánh giá heuristic" để xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút đó. Thuật toán này duyệt các nút theo thứ tự của đánh giá heuristic này

2) Mô tả thuật toán :

A* lưu giữ một tập các lời giải chưa hoàn chỉnh,nghĩa là các đường đi qua đồ thì bắt đầu từ nút xuất phát.Tập lời giải này được lưu trong một hàng đợi ưu tiên gọi là priority queue.Thứ tự ưu tiên gán cho một đường đi x được quyết định bởi hàm:

$$f(x) = g(x) + h(x).$$

Trong đó, g(x) là chi phí của đường đi cho đến thời điểm hiện tại, nghĩa là tổng trọng số của các cạnh đã đi qua. h(x) là hàm đánh giá heuristic về chi phí nhỏ nhất để đến đích từ x. Ví dụ, nếu "chi phí" được tính là khoảng cách đã đi qua, khoảng cách đường chim bay giữa hai điểm trên một bản đồ là một đánh giá heuristic cho khoảng cách còn phải đi tiếp.

Hàm f(x) có giá trị càng thấp thì độ ưu tiên của x càng cao

3) Cài đặt thuật toán :

A* sử dụng :

- tập OPEN chứa trạng thái khởi đầu
- tập CLOSE lưu trữ những trường hợp đã được xét đến
- mỗi trạng thái T(i) có một trạng thái cha gọi là Dad(Ti)
- trạng thái đích là TG

1) Ban đầu,tập OPEN chỉ chứa T0.
   Đặt g(T0)=0,h(T0) = 0,f(T0) = 0
   CLOSE là tập rỗng
2) Lặp lại các bước sau đến khi gặp điều kiện ngừng
   a) Nếu OPEN là rỗng : bài toán vô nghiệm,thoát
   b) Ngược lại chọn Tmax trong OPEN sao cho f(Tmax) nhỏ nhất
      - Lấy Tmax ra khỏi OPEN và đưa Tmax vào CLOSE
      - Nếu Tmax là TG thì thoát và thông báo lời giải là Tmax
      - Nếu Tmax không phải TG,tạo ra danh sách các trạng thái kế tiếp của Tmax.Gọi một trạng thái này là Tk.Với mỗi Tk thực hiện các bước sau :
        - Tính g(Tk) = g(Tmax) + cost (Tmax,Tk)
        - Nếu tốn tại Tk’ trong OPEN trùng với Tk.Nếu Tk’ có g(Tk) < g(Tk’) thì đặt g(Tk’) = g(Tk) và cập nhật lại f(Tk’).Đặt Dad(Tk’) = Tmax
        - Nếu tồn tại Tk’ trong CLOSE trùng với Tk.Nếu g(Tk) < g(Tk’) thì đặt lại g(Tk')=g(Tk) .Tính lại f(Tk)..Đặt Dad(Tk’)=Tmax
        - Nếu chưa xuất hiện Tk trong cả OPEN lẫn CLOSE thì : thêm Tk vào OPEN và tính lại f(Tk).

      Lan truyền sự thay đổi giá trị g, f’ cho tất cả các trạng thái kế tiếp của Ti (ở tất cả các cấp) đã được lưu trữ trong CLOSE và OPEN.

4) <u>Các tính chất của thuật toán :</u>

   a) *<u>Tính tối ưu :</u>*

Cũng như tìm kiếm theo chiều rộng (breadth-first search), A* là thuật toán đầy đủ theo nghĩa rằng nó sẽ luôn luôn tìm thấy một lời giải nếu bài toán có lời giải.

Nếu hàm heuristic h có tính chất thu nạp được , nghĩa là nó không bao giờ đánh giá cao hơn chi phí nhỏ nhất thực sự của việc đi tới đích, thì bản thân A* có tính chất thu nạp được (hay tối ưu) nếu sử dụng một tập đóng. Nếu không sử dụng tập đóng thì hàm h phải có tính chất đơn điệu (hay nhất quán) thì A* mới có tính chất tối ưu. Nghĩa là nó

không bao giờ đánh giá chi phí đi từ một nút tới một nút kề nó cao hơn chi phí thực. Phát biểu một cách hình thức, với mọi nút x,y trong đó y là nút tiếp theo của x:

$$h(x) \leq g(y) - g(x) + h(y)$$

A* còn có tính chất hiệu quả một cách tối ưu với mọi hàm heuristic h, có nghĩa là không có thuật toán nào cũng sử dụng hàm heuristic đó mà chỉ phải mở rộng ít nút hơn A*, trừ khi có một số lời giải chưa đầy đủ mà tại đó h dự đoán chính xác chi phí của đường đi tối ưu.

b) *Độ phức tạp :*

Độ phức tạp thời gian của A* phụ thuộc vào đánh giá heuristic. Trong trường hợp xấu nhất, số nút được mở rộng theo hàm mũ của độ dài lời giải, nhưng nó sẽ là hàm đa thức khi hàm heuristic h thỏa mãn điều kiện sau:

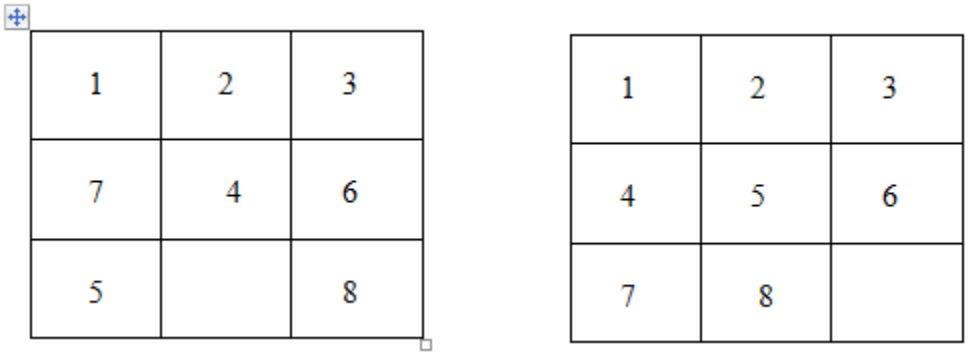
$$|h(x) - h^*(x)| \leq O(\log h^*(x))$$

trong đó h * là heuristic tối ưu, nghĩa là hàm cho kết quả là chi phí chính xác để đi từ x tới đích. Nói cách khác, sai số của h không nên tăng nhanh hơn lôgarit của "heuristic hoàn hảo" h * - hàm trả về khoảng cách thực từ x tới đích.

III) Bài toán ứng dụng

1) Giới thiệu trò chơi quân cờ (Cờ ta canh)

Tám (8) quân cờ được chỉ ra trong hình, gồm một bảng kích thước 3x3 với 8 quân cờ dược đánh số từ 1 đến 8 và một ô trống. Một quân cờ đứng cạnh ô trống có thể đi vào ô rống. Mục tiêu là luôn luôn tiến tới vị trí các quân cờ như ở trong hình bên phải (trạng thái đích).



1	2	3
7	4	6
5		8

1	2	3
4	5	6
7	8	

Trạng thái ban đầu Trạng thái đích

Cho đến nay ngoại trừ 2 phương pháp vét cạn và tìm kiếm Heuristic ,ta vẫn chưa tìm ra một thuật toán chính xác,tối ưu để giải bài toán này.Tuy nhiên,cách giải theo thuật toán A* đơn giản hơn và thường tìm được lời giải.Bài toán này có cấu trúc hoàn toàn thích hợp để có thể giải bằng phương pháp A*.

2) Cài đặt :

Ta sử dụng 3 hàm f,g,h.Trong đó :

- g là bước thực hiện
- h là tổng khoảng cách các ô ở vị trí sai lệch để trở về vị trí đúng.
- f=g+h

Mỗi trạng thái được đặc trưng bởi một cấu trúc như sau :

```
typedef struct
{
      int index; // chỉ số của trạng thái
      int f,g,h;
      int dad; // trạng thái trước đó phát sinh ra nó
} state;
```

Thuật toán thực hiện :

```
  stop = 0

while (!stop)
  {
      Loại bỏ trạng thái tốt nhất ra khỏi OPEN và đặt
   nó vào CLOSE;
     if (trạng thái tốt nhất == trạng thái cuối)
```

```
            {
               Thông báo đã tìm thấy kết quả;
               stop = 1;
            }
          else
           {
             Xác định vị trí ô trống;
             Sinh ra các trạng thái kế tiếp;
             if (trạng thái kế đã có trong OPEN hoặc CLOSE)
             {
                 Xét xem hàm g của nó có nhỏ hơn hàm g đã có
              hay chưa.
                 Nếu nhỏ hơn thì cập nhật lại f, g, h, dad.
             }
             if(trạng thái kế chưa có cả trong OPEN và CLOSE)
               {
           Tính toán f, g, h, dad của nó và đẩy nó vào OPEN.
               }
         Tìm trạng thái tốt nhất trong OPEN.
      }
    }
```

Lần ngược thuộc tính cha trong tập CLOSE và xây dựng lại đường đi cần tìm.

Các hàm cơ bản được sử dụng :

- Hàm PosZero dùng để xác định vị trí của ô trống,từ đó có thể suy ra các trạng thái có thể có bằng viêc di chuyển ô trống.

```
void PosZero(state a, int &zx, int &zy)
{
    int i,j;
    for (i=0;i<pSIZE;i++)
        for (j=0;j<pSIZE;j++)
            if (a.s[i][j]==0)
```

```
                {
                    zx = j;
                    zy = i;
                    break;
                }
    }
```

❖ Hàm hSub để tính khoảng cách từ vị trí hiện tại của những ô bị sai đến vị trí đúng

```
int hSub(int a, int row, int col)
{
    int i,j;
    int result=0;
    for (i=0;i<pSIZE;i++)
        for (j=0;j<pSIZE;j++)
            if (gs.s[i][j]==a)
            {
                result = abs(i-row) + abs(j-col);
                break;
            }
    return result;
}
```

❖ hàm h(state a) để tính tổng các hi

```
int h(state a)
{
    int i,j;
    int sum=0;
    putchar('\n');
    for (i=0;i<pSIZE;i++)
        for (j=0;j<pSIZE;j++)
            if (a.s[i][j] != gs.s[i][j])
            {
                sum += hSub(a.s[i][j],i,j);
                printf("\t h[%d]=%d", a.s[i][j], hSub(a.s[i][j],
i, j) );
            }
    return sum;
}
```

❖ hàm Backtrack() để đệ quy lần ngược tìm đường đi trong tập CLOSE.

```
void Backtrack(state a, state b[MAX], int size)
{
    int i;
    char bf[5];
    if (a.dad == 0)
    {
        printf("");
    }
    else
    {
        for (i=1; i<=size; i++)
            if (b[i].index == a.dad)
            {
                display(b[i],"\t:-------");
                Backtrack(b[i],b,size);
            }
    }
}
```

❖ hàm CreateNextStates() để sinh các trạng thái kế tiếp

```
void CreateNextStates()
{
    g++;
    int zx,zy;
    PosZero(bs, zx, zy);
    //printf("\n zx=%d zy=%d", zx,zy);

    if (zx - 1 >= 0) // left
    {
        cs[++id] = bs;
        cs[id].index = id;
        swap(cs[id].s[zy][zx] , cs[id].s[zy][zx-1]);
        process(cs[id]);
    }
    if (zx + 1 < pSIZE) //right
    {
        cs[++id] = bs;
```

```
        cs[id].index = id;
        swap(cs[id].s[zy][zx] , cs[id].s[zy][zx+1]);
        process(cs[id]);
    }
    if (zy - 1 >= 0) // up
    {
        cs[++id] = bs;
        cs[id].index = id;
        swap(cs[id].s[zy][zx] , cs[id].s[zy-1][zx]);
        process(cs[id]);
    }
    if (zy + 1 < pSIZE) //down
    {
        cs[++id] = bs;
        cs[id].index = id;
        swap(cs[id].s[zy][zx] , cs[id].s[zy+1][zx]);
        process(cs[id]);
    }}
```

3)Mã nguồn :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define MAX 100
#define pSIZE 3

int begin[pSIZE][pSIZE];
int end[pSIZE][pSIZE]  ;


typedef struct
{
      int index;
      int s[pSIZE][pSIZE];
```

```
      int f,g,h;
      int dad;
} state;

state fs;
state gs;
state bs;
state cs[MAX];
state open[MAX],close[MAX];
int openSize,closeSize;
int zx,zy;
int id=0,g=0;
     void docin(int begin[pSIZE][pSIZE],int &n)
  {  int i,j;
       FILE *f;
       f=fopen("INPUT.TXT","r");
            if((f==NULL))  printf("\n KO MO DC");
          else  {  fscanf(f,"%d",&n);
                for(i=0;i<n+1;i++)
                for(j=0;j<n+1;j++)
               fscanf(f,"%d",&begin[i][j]);
            }
       fclose(f);
       //printf("doc xong\n");
  }
     void docout(int end[pSIZE][pSIZE],int &n)
  {  int i,j;
       FILE *f;
       f=fopen("OUTPUT.TXT","r");
            if((f==NULL))  printf("\n KO MO DC");
          else  {  fscanf(f,"%d",&n);
                for(i=0;i<n+1;i++)
                for(j=0;j<n+1;j++)
               fscanf(f,"%d",&end[i][j]);
            }
       fclose(f);
       printf("doc xong\n");
```

```
  }
  void xuatb(int begin[pSIZE][pSIZE],int n)
  {   int i,j;
      for(i=0;i<n+1;i++)
     {  for(j=0;j<n+1;j++)
           printf("\t%d",begin[i][j]);
        printf("\n\n");
     }
  }
   void xuate(int end[pSIZE][pSIZE],int n)
  {   int i,j;
      for(i=0;i<n+1;i++)
     {  for(j=0;j<n+1;j++)
           printf("\t%d",end[i][j]);
        printf("\n\n");
     }
  }
void Line()
{
      printf("\n ------------------------------------------------
-\n");
}

void display(state a, char s[100])
{
      printf("\n %s: \n", s);
      int i,j;
      for (i=0;i<pSIZE;i++)
      {
            printf("\n\t");
            for (j=0;j<pSIZE;j++)
            {
                  printf(a.s[i][j]==0 ? "   " : " %d
",a.s[i][j]);
            }
      }
      printf("\t (id=%d f=%d g=%d h=%d dad=%d)\n", a.index, a.f,
```

```
a.g, a.h, a.dad);
}

void show(state a[MAX], int size)
{
	int i;
	printf(" [SIZE=%d]", size);
	for (i=1;i<=size;i++)
		display(a[i],"\tTrang thai con");
}

void init()
{
	int i,j;
	for (i=0;i<pSIZE;i++)
		for (j=0;j<pSIZE;j++)
		{
			fs.s[i][j]=begin[i][j];
			gs.s[i][j]=end[i][j];
		}
	fs.index = 0;
	fs.dad = 0;
	display(fs,"Trang thai ban dau");
	display(gs,"Trang thai dich");
}

int Equals(state a, state b)
{
	int i,j;
    for (i=0;i<pSIZE;i++)
		for (j=0;j<pSIZE;j++)
		if (a.s[i][j]!=b.s[i][j])
			{
				return 0;
				break;
			}
	return 1;
```

```
}

state &add(state a, state b[MAX], int &size)
{
      size++;
      int i,j;
      b[size] = a;

}

state &remove(state a, state b[MAX], int &size)
{
      int i,j;
      for (i=0;i<=size;i++)
            if (Equals(a,b[i]))
                  for (j=i+1;j<size;j++)
                        b[j-1]=b[j];
      size--;
}

void PosZero(state a, int &zx, int &zy)
{
    int i,j;
      for (i=0;i<pSIZE;i++)
            for (j=0;j<pSIZE;j++)
                  if (a.s[i][j]==0)
            {
                        zx = j;
                        zy = i;
                break;
            }
}

int hSub(int a, int row, int col)
{
    int i,j;
    int result=0;
```

```
    for (i=0;i<pSIZE;i++)
        for (j=0;j<pSIZE;j++)
            if (gs.s[i][j]==a)
            {
                result = abs(i-row) + abs(j-col);
                break;
            }
    return result;
}

int h(state a)
{
    int i,j;
    int sum=0;
    putchar('\n');
    for (i=0;i<pSIZE;i++)
        for (j=0;j<pSIZE;j++)
            if (a.s[i][j] != gs.s[i][j])
            {
                sum += hSub(a.s[i][j],i,j);
                printf("\t h[%d]=%d", a.s[i][j], hSub(a.s[i][j],
i, j) );
            }
    return sum;
}

void swap(int &a, int &b)
{
    int t;
    t = a;
    a = b;
    b = t;
}

int isIn(state a, state b[MAX], int size)
{
    int i;
```

```
    for (i=0; i<=size; i++)
        if (Equals(b[i], a))
        {
                    return 1;
            break;
        }
      return 0;
}

state process(state &a)
{
    if (isIn(a,open,openSize))
    {
        a.g = g;
        a.h = h(a);
        if (a.g + a.h < a.f)
        {
            remove(a,open,openSize);
            a.f = a.g + a.h;
            a.dad = bs.index;
            add(a,open,openSize);
        }
    }
    if (isIn(a,close,closeSize))
    {
        a.g = g;
        a.h = h(a);
        if (a.g + a.h < a.f)
        {
            remove(a,close,closeSize);
            a.f = a.g + a.h;
            a.dad = bs.index;
            add(a,close,closeSize);
        }
    }
    if (!isIn(a,open,openSize) && !isIn(a,close,closeSize))
    {
```

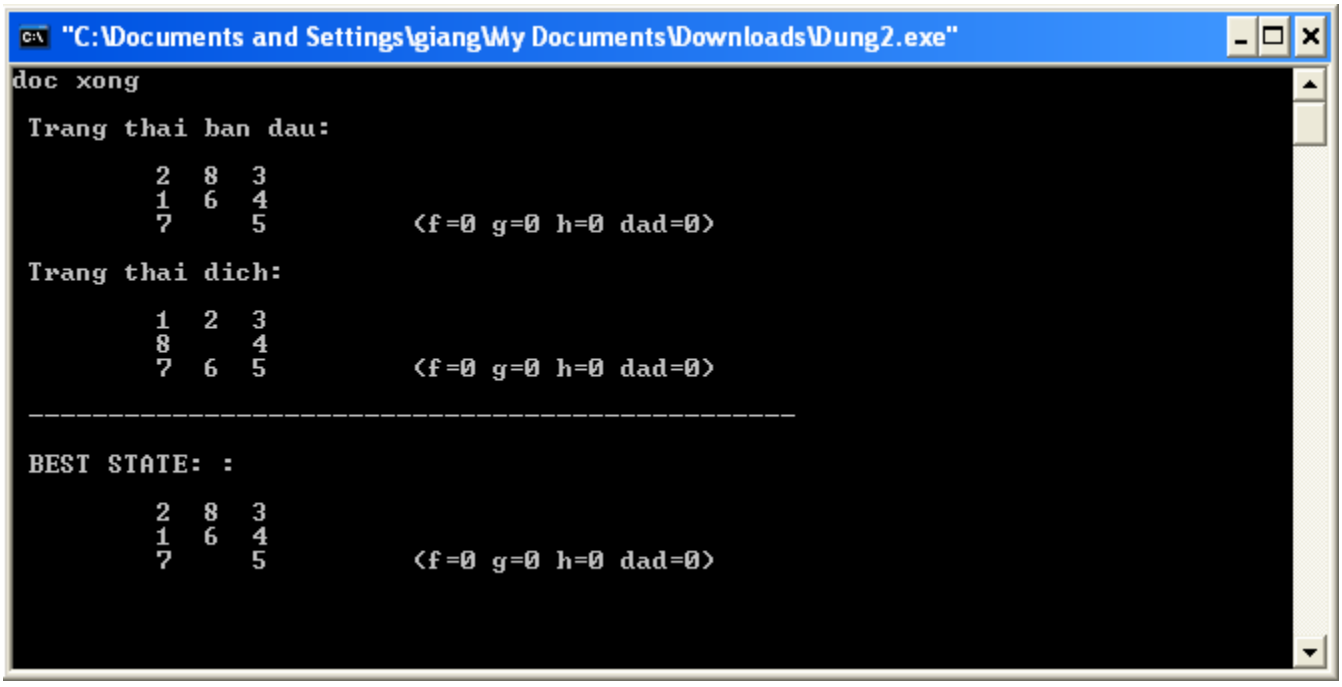
```
            a.g = g;
            a.h = h(a);
            a.f = a.g + a.h;
            a.dad = bs.index;
            add(a,open,openSize);

        }
    }

    void CreateNextStates()
    {
        g++;
        int zx,zy;
        PosZero(bs, zx, zy);
        //printf("\n zx=%d zy=%d", zx,zy);

        if (zx - 1 >= 0) // left
        {
            cs[++id] = bs;
            cs[id].index = id;
            swap(cs[id].s[zy][zx] , cs[id].s[zy][zx-1]);
            process(cs[id]);
        }
        if (zx + 1 < pSIZE) //right
        {
            cs[++id] = bs;
            cs[id].index = id;
            swap(cs[id].s[zy][zx] , cs[id].s[zy][zx+1]);
            process(cs[id]);
        }
        if (zy - 1 >= 0) // up
        {
            cs[++id] = bs;
            cs[id].index = id;
            swap(cs[id].s[zy][zx] , cs[id].s[zy-1][zx]);
            process(cs[id]);
        }
```

```
    if (zy + 1 < pSIZE) //down
    {
        cs[++id] = bs;
        cs[id].index = id;
        swap(cs[id].s[zy][zx] , cs[id].s[zy+1][zx]);
        process(cs[id]);
    }
}

state BestState(state a[MAX],int size)
{
    int i;
    int pos;
    int min=10000;
    for (i=1;i<=size;i++)
        if (a[i].f<min && a[i].f>0)
        {
            min = a[i].f;
            pos = i;
        }
    return a[pos];
}

void Backtrack(state a, state b[MAX], int size)
{
    int i;
    char bf[5];
    if (a.dad == 0)
    {
        printf("");
    }
    else
    {
        for (i=1; i<=size; i++)
            if (b[i].index == a.dad)
            {
                display(b[i],"\t:-------");
```

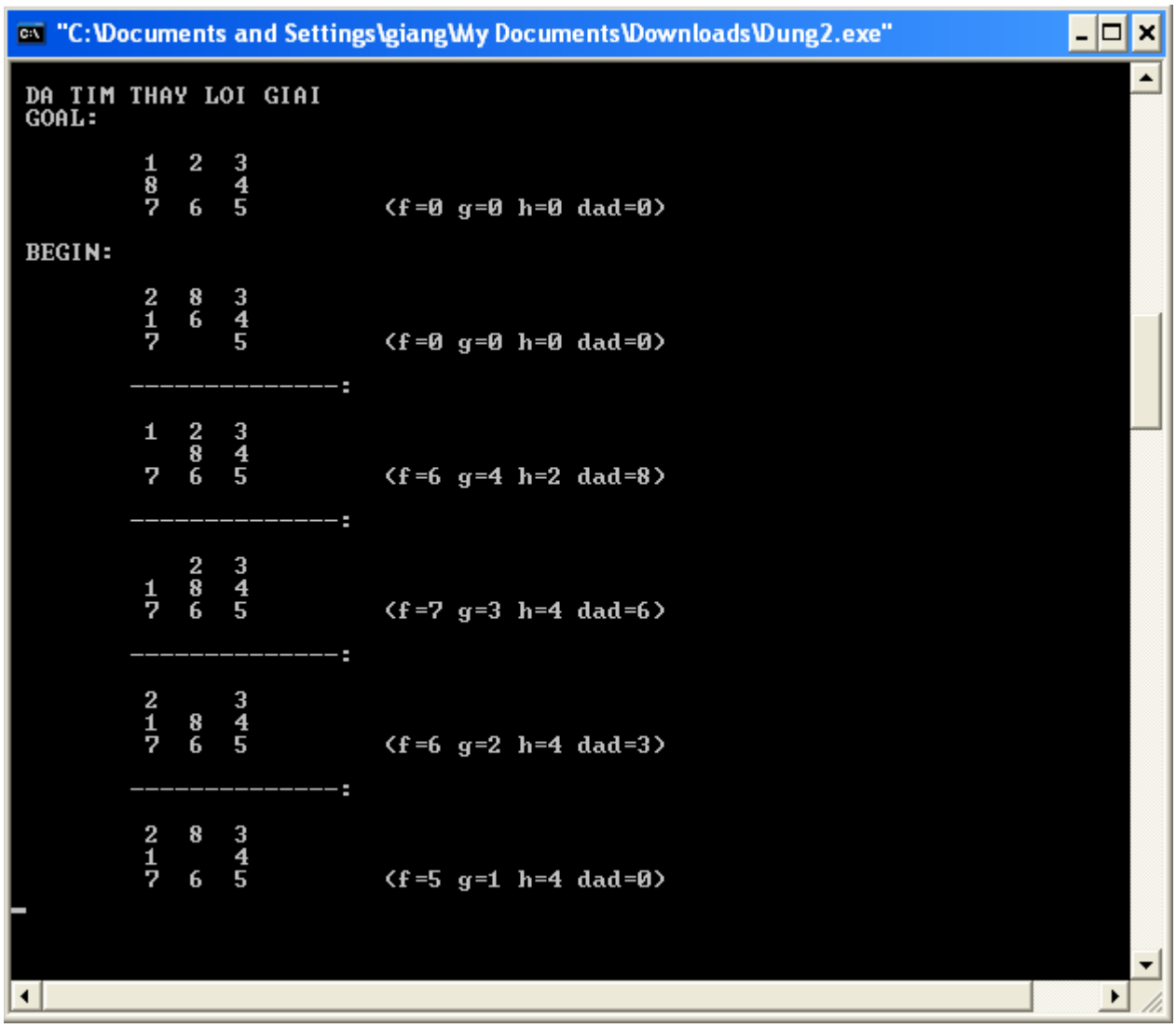
```
                Backtrack(b[i],b,size);
            }
        }
    }

    void solve()
    {
        bs = fs;
        openSize = closeSize = 0;
        add(bs,open,openSize);
          int stop=0;

        while(!stop)
        {
            display(bs,"BEST STATE: ");
            remove(bs,open,openSize);
            add(bs,close,closeSize);
            if (Equals(bs,gs))
            {
                printf("\n DA TIM THAY LOI GIAI");
                stop = 0;
                        break;
            }
            else
            {
                CreateNextStates();
            }
            if (openSize>0)
                bs = BestState(open,openSize);
                   Line();
            getch();
        }
        display(fs,"BEGIN");
        display(gs,"GOAL");
        Backtrack(close[closeSize],close,closeSize);

    }
```

```
int main()
{    int n;
 docin(begin,n);
 docout(end,n);
      //xuatb(begin,n);
      //xuate(end,n);
    init();
    Line();
    solve();
      getch();

    return 0;
}
```

3) Demo:



Kết quả :


"C:\Documents and Settings\giang\My Documents\Downloads\Dung2.exe"
DA TIM THAY LOI GIAI
GOAL:
1 2 3
8 4
7 6 5
(f=0 g=0 h=0 dad=0)
BEGIN:
2 8 3
1 6 4
7 5
(f=0 g=0 h=0 dad=0)
----------:
1 2 3
8 4
7 6 5
(f=6 g=4 h=2 dad=8)
----------:
2 3
1 8 4
7 6 5
(f=7 g=3 h=4 dad=6)
----------:
2 3
1 8 4
7 6 5
(f=6 g=2 h=4 dad=3)
----------:
2 8 3
1 4
7 6 5
(f=5 g=1 h=4 dad=0)