ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

BÀI TẬP MÔN TRÍ TUỆ NHÂN TẠO

XÂY DỰNG CHƯƠNG TRÌNH GIẢI BÀI TOÁN 8 – PUZZLE

Giảng viên hướng dẫn: PGS.TS Đỗ Văn Nhơn

ThS Nguyễn Thị Ngọc Diễm

Sinh viên thực hiện: Lưu Thanh Sơn - 14520772

Đỗ Phú An – 14520002

Lớp: Trí tuệ nhân tạo (CS106.G21.KHTN)

Thành phố Hồ Chí Minh, ngày 27 tháng 5 năm 2015

Mục lục:

1. Mô tả bài toán

Bài toán Ta-canh đã từng là một trò chơi khá phổ biến, đôi lúc người ta còn gọi đây là bài toán 9-puzzle. Trò chơi bao gồm một hình vuông kích thước 3x3 ô. Có 8 ô có số, mỗi ô có một số từ 1 đến 8. Một ô còn trống. Mỗi lần di chuyển chỉ được di chuyển một ô nằm cạnh ô trống về phía ô trống. Vấn đề là từ một trạng thái ban đầu bất kỳ, làm sao đưa được về trạng thái cuối là trạng thái mà các ô được sắp lần lượt từ 1 đến 8 theo thứ tự từ trái sang phải, từ trên xuống dưới, ô cuối dùng là ô trống.

3	1	4
7		6
8	2	5

Hình 1

1	2	3
4	5	6
7	8	

Hình 2

a. *Cơ sở thuật giải: Thuật giải A**

- Trong khoa học máy tính, A* (đọc là A sao) là một thuật toán tìm kiếm trong đồ thị. 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. Do đó, thuật toán A* là một ví dụ của tìm kiếm theo lựa chọn tốt nhất (best-first search).

```
function A*(điểm_xuất_phát,đích)
    var đóng:= tập rỗng
    var q:=
tạo_hàng_đợi(tạo_đường_đi(điểm_xuất_phát))
    while q không phải tập rỗng
        var p:= lấy_phần_tử_đầu_tiên(q)
        var x:= nút cuối cùng của p
        if x in đóng
            continue
        if x = đích
            return p
        bổ sung x vào tập đóng
        foreach y in các_đường_đi_tiếp_theo(p)
            đưa_vào_hàng_đợi(q, y)
    return failure
```

- Trong đó, `các_đường_đi_tiếp_theo(p)` trả về tập hợp các đường đi tạo bởi việc kéo dài `p` thêm một nút kề cạnh. Giả thiết rằng hàng đợi được sắp xếp tự động bởi giá trị của hàm

- "Tập hợp đóng" (`đóng`) lưu giữ tất cả các *nút cuối cùng của p* (các nút mà các đường đi mới đã được mở rộng tại đó) để tránh việc lặp lại các chu trình (việc này cho ra thuật toán tìm kiếm theo đồ thị). Đôi khi hàng đợi được gọi một cách tương ứng là "tập mở". Tập đóng có thể được bỏ qua (ta thu được thuật toán tìm kiếm theo cây) nếu ta đảm bảo được rằng tồn tại một lời giải hoặc nếu hàm `các_đường_đi_tiếp_theo` được chỉnh để loại bỏ các chu trình.

b. *Ý tưởng chính*

- Sinh ra các trạng thái có thể đi tiếp theo bằng việc di chuyển ô trống. Chọn ra trạng thái tốt nhất trong tập các trạng thái vừa sinh ra, lặp lại thao tác đó cho đến khi tìm được lời giải
  VD: Như hình 1, ta sẽ có 4 trạng thái có thể được sinh ra như sau:

3		4
7	1	6
8	2	5

3	1	4
7	2	6
8		5

3	1	4
	7	6
8	2	5

3	1	4
7	6	
8	2	5

- Chúng ta sẽ coi mỗi một trạng thái sinh ra giống như 1 đường đi. Và trạng thái kết quả (trạng thái đích) là điểm cần đến. Cho nên chúng ta sẽ quy nó về bài toán tìm đường đi. Và thuật toán được áp dụng ở đây là A* (một giải thuật rất linh động cho việc tìm đường đi ngắn nhất)

- Vấn đề ở đây là, giải thuật A* chọn đường đi tiếp theo dựa vào tập các đường đi đã sinh ra từ đường đi trước đó dựa vào 1 hàm đánh giá h. Chúng ta sẽ tính **h** cho mỗi trạng thái bằng đường đi **Manhattan:**

$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|,$$

  Hay

$$|p_1 - q_1| + |p_2 - q_2|.$$

  Với p1, q1 là tọa độ (dòng,cột) của ô đang xét ở trạng thái hiện tại, còn p2, q2 là tọa độ (dòng,cột) của ô đó khi ở trạng thái đích
  Ví Dụ:

3	1	4
	7	6
8	2	5

Xét ô số 8 ở hình trên: vị trí hiện tại: dòng 3, cột 1

Xét ô số 8 ở hình 2: vị trí đích: dòng 3, cột 2

d=$|3-1|+|3-2|=3$

Tương tự, ta tính cho các ô còn lại. Sau đó cộng lại các kết quả vừa tính ta được hàm h. Đây chính là hàm ước lượng cho trạng thái đang xét. A* sẽ dựa vào h để chọn ra trạng thái tối ưu nhất

c. *Thuật giải tổng quát*

```
Function Bai_Toan_Tacanh (trạng thái đầu, trạng thái đích)
{
        OPEN={};
        CLOSE={};
        OPEN ← {Trạng thái đầu};
        Trạng thái t;
        Trong khi t ≠ Trạng thái đích
            {
                Lấy trạng thái trong OPEN đưa vào CLOSE
                Sinh ra tập {trạng thái kế tiếp}
                Với mỗi t trong {trạng thái kế tiếp}
                        - Nếu đã có trong OPEN thì xét: nếu h mới lớn hơn h cũ thì cập nhật lại h
                        - Tương tự với CLOSE
                        - Nếu không có trong cả CLOSE và OPEN thì thêm vào OPEN
            Chọn trạng thái tốt nhất (dựa vào hàm h)
            t ← trạng thái tốt nhất
            }
}
```

2. Đánh giá giải thuật:

Ta có mỗi trạng thái của bảng số là một hoán vị của m x m phần tử (với m là cạnh), như vậy không gian trạng thái của nó là (m x m)!, với 8-puzzle là 9! = (3x3)! = 362880 (m = 3) và 15-puzzle là 16! =(4x4)!= 20922789888000 (m =4). Khi ta tăng lên một đơn vị thì số trạng thái tăng lên rất nhiều !

Ta còn phải xét xem rằng, bài toán với các trạng thái hiện tại có giải được hay không, dựa vào 1 đánh giá sau: Ta lần lượt xét xem sau trạng thái (i,j) có bao nhiêu trạng thái nhỏ hơn nó (trừ ô trống), cộng lần lượt như vậy cho tới trạng thái cuối cùng. Sau đó lấy kết quả vừa thu được chia lấy dư cho 2. Nếu phần dư bằng 0 thì bài toán có thể giải, ngược lại, nếu phần dư khác 0 thì bài toán không giải được

Ví dụ: Có trạng thái sau:

3	1	4
	7	6
8	2	5

Ta có:

3: 1, 2 – h=2

1: không có – h=0

4: 2 – h=1

7: 6, 2, 5 – h=3

6: 2, 5 – h=2

8: 2, 5 – h=2

2: không có – h=0

5: không có: h=0

- ⇨ Tổng h=2+0+1+3+2+2+0+0=10
- ⇨ Vì 10%2=0 nên bài toán có thể giải được

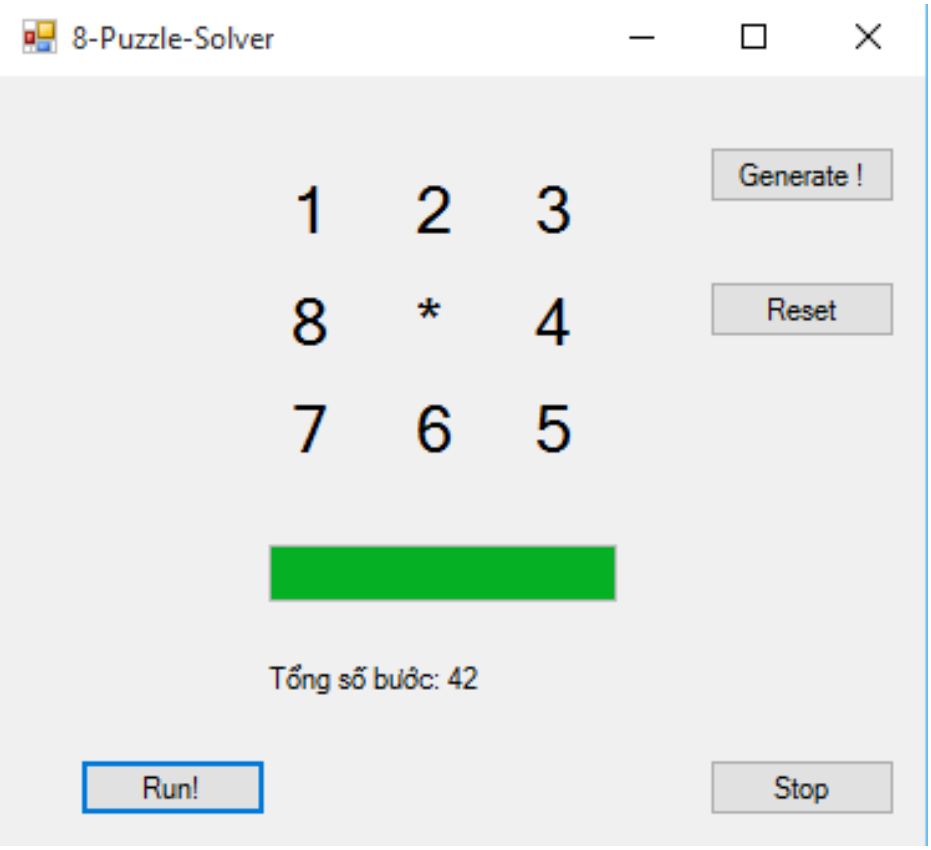
3. Cài đặt thử nghiệm

a. Giao diện chính của chương trình

- Danh sách các nút đều khiển:

Run!	Thực thi bài toán
Reset	Xóa bài toán hiện tại
Stop	Dừng chương trình
Generate	Sinh bài toán một cách ngẫu nhiên

- Giao diện chính:



b. Mã nguồn chương trình:

(Chương trình được viết bằng C#)

- **Lớp Point.cs dùng để lưu trữ tọa độ của 1 ô trong bảng:**

```
class Position
   {
      int row;
      int column;
```

```
        public Position() { row = -1; column = -1; }
-------------------------------------------
        public int getRow()
        {
            return row;
        }
-----------------------------
        public int getColumn()
        {
            return column;
        }
-------------------------------
        public void setRow(int r)
        {
            this.row = r;
        }
-------------------------------------
        public void setColumn(int col)
        {
            this.column = col;
        }
    }
```

- **Lớp Puzzle.cs lưu trữ trạng thái được tạo ra:**

```
class Puzzle
    {
        int[,] Matrix;  //ma trận trạng thái
        int h;   //hàm đánh giá
        int n;
        int[,] dad;  //trạng thái cha sinh ra nó
-------------------------------------------------
        public Puzzle(int n)
        {
            this.n = n;
            Matrix = new int[n, n];
            dad = new int[n, n];
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                {
                    Matrix[i, j] = new int();
                    Matrix[i, j] = 0;
                }
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                {
                    dad[i, j] = new int();
                    dad[i, j] = 0;
                }
            h = 0;
```

```
    }
-----------------------------------------
    public Puzzle() { }

    public int geth()
    {
       return h;
    }
-------------------------------------
    public void seth(int h)
    {
       this.h = h;
    }
---------------------------------------
    public int[,] getMatrix()
    {
       int[,] M = new int[n, n];
       for (int i = 0; i < n; i++)
          for (int j = 0; j < n; j++)
          {
             M[i, j] = new int();
             M[i, j] = this.Matrix[i, j];
          }
       return M;
    }
-----------------------------------------------------
    public void setMatrix(int[,] Matrix)
    {
       for (int i = 0; i < n; i++)
          for (int j = 0; j < n; j++)
             this.Matrix[i, j] = Matrix[i, j];
    }
---------------------------------------
    public int[,] getDad()
    {
       int[,] M = new int[n, n];
       for (int i = 0; i < n; i++)
          for (int j = 0; j < n; j++)
          {
             M[i, j] = new int();
             M[i, j] = this.dad[i, j];
          }
       return M;
    }
-----------------------------------------------
    public void setDad(int[,] p)
    {
       for (int i = 0; i < n; i++)
          for (int j = 0; j < n; j++)
             this.dad[i, j] = p[i, j];
    }
```

```
}
```

- **Lớp Process.cs chứa các hàm chính thực thi:**

```
class Process
   {
       List<Puzzle> Result = new List<Puzzle>();
      bool isDup(List<int>l,int a)
       {
          for (int i = 0; i < l.Count; i++)
              if (a == l[i])
                  return true;
          return false;
       }
```

<u>Các hàm hỗ trợ:</u>

```
------------------Hàm hoán đổi 2 số nguyên----------------------------------------------
      public static void Swap(ref int a, ref int b)
      {
         int temp;
         temp = a;
         a = b;
         b = temp;
      }
-----------------hàm tìm vị trí của 1 ô trong trạng thái hiện tại-----------------------------
      public Position findMatrix(int val, int[,] Matrix, int n)
      {
         Position p = new Position();
         bool isbreak = false;
         for (int i = 0; i < n; i++)
         {
            if (isbreak == true)
                break;
```

```
                else
                    for (int j = 0; j < n; j++)
                    {
                        if (Matrix[i, j] == val)
                        {
                            isbreak = true;
                            p.setColumn(j);
                            p.setRow(i);
                            break;
                        }
                    }
                }
                return p;
            }
        ----------hàm tìm trạng thái trong tập các trạng thái----------
            public int findPuzzle(Puzzle p, List<Puzzle> lP, int n)
            {
                int k = 0;
                for (int i = 0; i < lP.Count; i++)
                    if (duplicateMatrix(p, lP[i], n) == true)
                    {
                        k = i;
                        break;
                    }
                return k;
            }
        ----------hàm kiểm tra xem 2 ma trận có trùng nhau không----------
            public bool duplicateMatrix(int[,] M1, int[,] M2, int n)
            {
                bool l = true;
                for (int i = 0; i < n; i++)
                {
```

```
                if (l == false)
                    break;
                else
                    for (int j = 0; j < n; j++)
                        if (M1[i, j] != M2[i, j])
                        {
                            l = false;
                            break;
                        }
            }
            return l;
        }
----------tìm xem trạng thái hiện tại có trong tập các trạng thái trước đó chưa ?----------
        public bool duplicatePuzzle(Puzzle p, List<Puzzle> lP, int n
        {
            bool k = false;
            for (int i = 0; i < lP.Count; i++)
                if (duplicateMatrix(p, lP[i], n) == true)
                {
                    k = true;
                    break;
                }
            return k;
        }
----------kiểm tra xem 2 trạng thái có trùng nhau không ?----------
        public bool duplicateMatrix(Puzzle p, Puzzle k, int n
        {
            bool l = true;
            int[,] M1 = new int[n, n];
            int[,] M2 = new int[n, n];
            M1 = p.getMatrix();
            M2 = k.getMatrix();
```

```
            for (int i = 0; i < n; i++)
            {
                if (l == false)
                    break;
                else
                    for (int j = 0; j < n; j++)
                        if (M1[i, j] != M2[i, j])
                        {
                            l = false;
                            break;
                        }
            }
            return l;
        }
----------Tiên đoán số trạng thái ----------
int Prediction(int[,]Matrix)
        {
            int temp = 0;
            int first=Matrix[0,0];
            for (int i = 0; i < 3; i++)
                for (int j = 0; j < 3; j++)
                {
                    if (Matrix[i, j] < first && Matrix[i, j] != -1)
                        temp++;
                }
            if (temp % 2 == 0)
                return 1;
            else
                return 0;
        }
```

Các hàm thực thi công việc chính:

```
----------hàm sinh ra ngẫu nhiên ma trận----------
    public void GenerateMatrix(ref int [,]Matrix,int n)
    {
        List<int> h=new List<int>();
        Random r = new Random();
        for(int i=0 ;i<n;i++)
            for(int j=0;j<n;j++)
            {
                if (h.Count <= 0)
                {
                    Matrix[i, j] = r.Next(0, 9);
                    h.Add(Matrix[i, j]);
                }
                else
                {
                    do
                    {
                        Matrix[i, j] = r.Next(0, 9);
                        bool k = isDup(h, Matrix[i, j]);
                    }
                    while (isDup(h, Matrix[i, j]) == true);
                    h.Add(Matrix[i, j]);
                }
            }
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (Matrix[i, j] == 0)
                    Matrix[i, j] = -1;
    }
----------hàm tạo ra trạng thái cuối của Puzzle (trạng thái tốt nhất)----------
public Puzzle LastPuzzle(int [,]Matrix)  //the Best Puzzle (Result for the Game)
```

```
        {
            Puzzle p = new Puzzle(3);
            int[,] M = new int[3, 3] { { 1, 2, 3 }, { 8, -1, 4 }, { 7, 6, 5 } };
            int[,] N = new int[3, 3] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, -1 } };
            if (Prediction(Matrix) == 1)
            {
                p.setMatrix(M);
                p.seth(0);
            }
            else
            {
                p.setMatrix(N);
                p.seth(0);
            }
            return p;
        }
    ----------hàm sinh ra các trạng thái kế tiếp từ trạng thái hiện tại----------
        public List<Puzzle> SwapPuzzle(Puzzle p, int n, Puzzle init)
        {
            List<Puzzle> lP = new List<Puzzle>();
            int[,] Matrix = new int[n, n];
            Matrix = p.getMatrix();
            Position pM = new Position();
            pM = findMatrix(-1, Matrix, n);
            if (pM.getColumn() > 0)    //left
            {
                Matrix = p.getMatrix();
                Puzzle pz = new Puzzle(n);
                Swap(ref Matrix[pM.getRow(), pM.getColumn()], ref Matrix[pM.getRow(),
pM.getColumn() - 1]);
                int h = Measure(Matrix, init, n);
                pz.setMatrix(Matrix);
                pz.seth(h);
```

1	2	3
4	5	6
7	8	-1

Trạng thái 1

1	2	3
8	-1	4
7	6	5

Trạng thái 2

```
                    pz.setDad(p.getMatrix());
                    lP.Add(pz);
                }
                if (pM.getColumn() < n - 1)    //right
                {
                    Matrix = p.getMatrix();
                    Puzzle pz = new Puzzle(n);
                    Swap(ref Matrix[pM.getRow(), pM.getColumn()], ref Matrix[pM.getRow(),
pM.getColumn() + 1]);
                    int h = Measure(Matrix, init, n);
                    pz.setMatrix(Matrix);
                    pz.seth(h);
                    pz.setDad(p.getMatrix());
                    lP.Add(pz);
                }
                if (pM.getRow() > 0)    //up
                {
                    Matrix = p.getMatrix();
                    Puzzle pz = new Puzzle(n);
                    Swap(ref Matrix[pM.getRow(), pM.getColumn()], ref Matrix[pM.getRow() - 1,
pM.getColumn()]);
                    int h = Measure(Matrix, init, n);
                    pz.setMatrix(Matrix);
                    pz.seth(h);
                    pz.setDad(p.getMatrix());
                    lP.Add(pz);
                }
                if (pM.getRow() < n - 1)   //down
                {
                    Matrix = p.getMatrix();
                    Puzzle pz = new Puzzle(n);
                    Swap(ref Matrix[pM.getRow(), pM.getColumn()], ref Matrix[pM.getRow() + 1,
pM.getColumn()]);
```

```
                int h = Measure(Matrix, init, n);
                pz.setMatrix(Matrix);
                pz.seth(h);
                pz.setDad(p.getMatrix());
                lP.Add(pz);
            }
            return lP;
        }
    ----------Chọn ra trạng thái tốt nhất----------
        public Puzzle choosePuzzle(List<Puzzle> lP, int n)
        {
            Puzzle pn = new Puzzle(n);
            int min = 100000;
            int key = 0;
            for (int i = 0; i < lP.Count; i++)
            {
                if (lP[i].geth() < min)
                {
                    min = lP[i].geth();
                    key = i;
                }
            }
            pn.setMatrix(lP[key].getMatrix());
            pn.seth(lP[key].geth());
            pn.setDad(lP[key].getDad());
            return pn;
        }
    ----------hàm đánh giá heuristic trạng thái----------
        public int Measure(int[,] Matrix, Puzzle init, int n)
        {
            int h = 0;
            int[,] MatrixInit = new int[n, n];
```

```
                MatrixInit = init.getMatrix();
                for (int i = 0; i < n; i++)
                    for (int j = 0; j < n; j++)
                    {
                        Position po = new Position();
                        po = findMatrix(Matrix[i, j], MatrixInit, n);
                        h = h + ((Math.Abs(i - po.getRow()) + Math.Abs(j - po.getColumn())));
                    }
                return h;
            }
        ----------hàm tìm đường đi tới trạng thái đích----------
            public List<Puzzle> useAStar(Puzzle p, Puzzle init, int n
            {
                List<Puzzle> OPEN = new List<Puzzle>();
                List<Puzzle> CLOSE = new List<Puzzle>();
                Puzzle pz = new Puzzle(n);
                pz.setMatrix(p.getMatrix());  //pz is the initial state of Puzzle input by user
                pz.seth(p.geth());
                OPEN.Add(p); //add initial state to the OPEN
                bool b = false;
                while (b == false)
                {
                    OPEN.RemoveAt(findPuzzle(pz, OPEN, n));
                    CLOSE.Add(pz);
                    if (duplicateMatrix(pz, init, n) == true)  //if the state is the Result then Exit
                    {
                        b = true;
                    }
                    List<Puzzle> lP = SwapPuzzle(pz, n, init);  //the Generated State from the below state
                    for (int i = 0; i < lP.Count; i++)
                    {
                        if (duplicatePuzzle(lP[i], OPEN, n) == true) //has had in OPEN, then update its value
if its h value is greater than present value
```

```
                        if (lP[i].geth() < OPEN[findPuzzle(lP[i], OPEN, n)].geth())
                        {
                            OPEN[findPuzzle(lP[i], OPEN, n)].seth(lP[i].geth());
                            OPEN[findPuzzle(lP[i], OPEN, n)].setDad(lP[i].getDad());
                        }
                    if (duplicatePuzzle(lP[i], CLOSE, n) == true) //has had in OPEN, then update its value if its h value is greater than present value
                        if (lP[i].geth() < CLOSE[findPuzzle(lP[i], CLOSE, n)].geth())
                        {
                            CLOSE[findPuzzle(lP[i], CLOSE, n)].seth(lP[i].geth());
                            CLOSE[findPuzzle(lP[i], CLOSE, n)].setDad(lP[i].getDad());
                        }
                    if (duplicatePuzzle(lP[i], CLOSE, n) == false && duplicatePuzzle(lP[i], OPEN, n) == false) //add to OPEn if it don't have in both OPEN and CLOSE
                        {
                            OPEN.Add(lP[i]);
                        }
                    }
                    pz = choosePuzzle(OPEN, n);
                }
                return CLOSE;
            }
        ----------Khởi tạo ban đầu ----------
            public void initialGame(ref int [,]Matrix,int n)
            {
                n = 3;
                Matrix = new int[n, n];
                GenerateMatrix(ref Matrix, n);
            }
        ----------truy vết lại trạng thái cha trước đó để hình thành đường đi hoàn chỉnh----------
            List<Puzzle> BackTrack(Puzzle p, List<Puzzle> lP, int n, Puzzle init  )
            {
                List<Puzzle> BPz = new List<Puzzle>();
```

```
            int k = 0;
            BPz.Add(init);
            while (k <= lP.Count)
            {
                for (int i = 1; i < lP.Count; i++)
                    if (duplicateMatrix(p.getDad(), lP[i].getMatrix(), n) == true)
                    {
                        BPz.Add(lP[i]);
                        p.setMatrix(lP[i].getMatrix());
                        p.seth(lP[i].geth());
                        p.setDad(lP[i].getDad());
                    }
                k++;
            }
            return BPz;
        }
    ----------trả về kết quả-------------------------------
        public List<Puzzle> returnResult()
        {
            return this.Result;
        }
    --------thực thi lần lượt các bước--------------
        public void Processing(int [,]Matrix)  //Proccess the Program
        {
            //1. Input
            int n = 3;
            //2. Initial the Best State (Result)
            Puzzle p = new Puzzle(n);
            Puzzle init = new Puzzle(n);
            init = InitPuzzle(n);
            p.setMatrix(Matrix);
            p.seth(Measure(Matrix, init, n));
```

```
            List<Puzzle> lP = new List<Puzzle>();
            //3. Execute it using A-star Algorithm
            lP = useAStar(p, init, n);
            //4. Show the Result
            Result = BackTrack(lP[lP.Count - 1], lP, n, init);
            Result.Reverse(); //reverse the List
            //ShowResult(Result, n);
        }
    }
```

Tài liệu Tham khảo:

1. Hoàng Kiếm, Đinh Nguyễn Anh Dũng – Giáo trình Nhập môn Trí tuệ nhân tạo, Trường Đại học CNTT – ĐHQG.TPHCM, NXB ĐHQG.TPHCM – 2010
2. Lê Hoài Bắc, Tô Hoài Việt – Giáo trình Cơ sở Trí tuệ nhân tạo, Trường Đại học KHTN – ĐHQG.TPHCM, NXB Khoa học và Kỹ Thuật – 2014
3. https://yinyangit.wordpress.com/2010/12/16/algorithm-phan-tich-va-gi%E1%BA%A3i-bai-toan-n-puzzle/
4. http://en.wikipedia.org/wiki/Taxicab_geometry
5. https://**msdn**.microsoft.com/vi-vn
6. http://math.stackexchange.com/questions/293527/how-to-check-if-a-8-puzzle-is-solvable