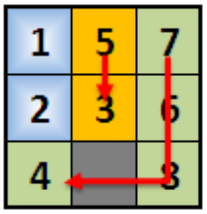
“

YinYang's Programing Blog

"Education is not a preparation for life; education is life itself"

Algorithm – Phân tích và giải bài toán n-puzzle



(https://yinyangit.files.wordpress.com/2010/12/8-puzzle2.png)Trong bài trước (https://yinyangit.wordpress.com/2010/12/13/c-source-y2-n-pluzzle-solver-v1-0-ch%C6%B0%C6%A1ng-trinh-minh-h%E1%BB%8Da-gi%E1%BA%A3i-bai-toan-n-puzzle/) tôi đã giới thiệu chương trình minh họa giải bài toán n-puzzle kèm theo mã nguồn (VC# 2008). Bài viết này sẽ dùng mã nguồn đó để phân tích và giải thích nên tôi chỉ trích những đoạn code quan trọng lên đây.

Download mã nguồn (http://www.mediafire.com/?ncv8xt2ze5o69kf) (VC# 2008 – 229KB)

I. Giới thiệu:

Vui lòng đọc tại đây (https://yinyangit.wordpress.com/2010/12/11/algorithm-tim-hi%E1%BB%83u-v%E1%BB%81-bai-toan-n-puzzle-updated/)

II. Thuật toán tìm kiếm A*

A* là một giải thuật tìm kiếm thường được sử dụng trong các vấn đề liên quan đến đồ thị và tìm đường đi. Nó được chọn không chỉ vì tính hiệu quả mà còn vì rất dễ dàng để hiểu và cài đặt. Bạn cần nắm rõ thuật toán này trước khi tiếp tục. Tôi giải sử bạn đã biết về các lý thuyết này, tuy nhiên để tiện lợi cho việc tham khảo bạn có thể đọc ở hai link bên dưới:
– Giải thuật tìm kiếm A* (http://vi.wikipedia.org/wiki/Gi%E1%BA%A3i_thu%E1%BA%ADt_t%C3%ACm_ki%E1%BA%BFm_A*)

– A* search algorithm (http://en.wikipedia.org/wiki/A*_search_algorithm)

III. Phân tích bài toán

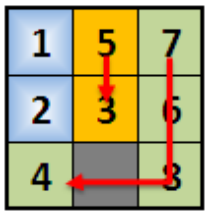
– Như đã giới thiệu trong bài trước, có những trạng thái của bảng số không thể chuyển về trạng thái đích, ta gọi là **cấu hình hợp lệ** và không hợp lệ. Tỉ lệ giữa chúng là ½, điều này có thể nhận ra dễ dàng từ phương pháp tính xem bài toán có thể đưa về trạng thái đích hay không.

– Rất dễ thấy là 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! = 362880 (m = 3) và 15-puzzle là 16! = 20922789888000 (m =4). Bạn có thể khi m tăng lên 1 đơn vị thì không gian trạng thái tăng lên rất nhanh, điều này khiến cho việc giải quyết các phiên bản m>3 ít khi được áp dụng.

– Để áp dụng thuật toán A* giải bài toán này, bạn cần một hàm heuristic **h** để ước lượng giá trị của mỗi trạng thái của bảng số. Có một số cách bạn có thể đã biết tới như tính dựa vào khoảng cách sai lệch của các ô số với vị trí đúng, hoặc đơn giản là đếm xem có bao nhiêu ô sai vị trí,… Ở đây tôi chọn theo cách thứ nhất, tức là tính tổng số ô sai lệch của các ô số so với vị trí đúng của nó. Đây là cách tính thường được sử dụng và nó có tên gọi là **Manhattan**.

> **Nếu bạn tìm kiếm trên mạng về **Manhattan** bạn sẽ thấy rằng đây là tên một quận của thành phố New York, nơi mà các con đường chạy ngang dọc như một bàn cờ lớn, nhờ thế việc tìm được cũng như di chuyển rất thuận lợi.*
>
> *Tham khảo: http://en.wikipedia.org/wiki/Taxicab_geometry (http://en.wikipedia.org/wiki/Taxicab_geometry)*

-Ví dụ:



(https://yinyangit.files.wordpress.com/2010/12/8-puzzle2.png)
Tính khoảng cách Manhattan

Trong bảng số 3×3 trên, để di chuyển ô số 5 vào đúng vị trí ta cần di chuyển nó 1 lần, để di chuyển ô số 7 về đúng vị trí ta cần cần 4 lần (qua 4 ô khác). Để có được kết quả này ta làm phép tính đơn giản: lấy tổng khoảng cách của dòng và cột giữa hai vị trí (ví dụ với ô số 7):

– Lấy tọa độ của ô số 7 ta có row1 = 0 và column1 = 2

– Lấy tọa độ của ô số 7 khi ở vị trí đúng, ta có ow2 = 2 và column2 = 0

– Vậy khoảng cách Manhattan của hai ô này là:

$$|row1 - row2| + |column1 - column2| = |0 - 2| + |2 - 0| = 4$$

Theo đó, ta tính được **h = 0+1+4+2+2+0+1+1+1 = 12**

Như vậy ở trạng thái đích thì bảng số sẽ có giá trị thấp nhất là 0.

Từ giá trị của một ô số ta tính vị trí dòng và cột của nó như sau:

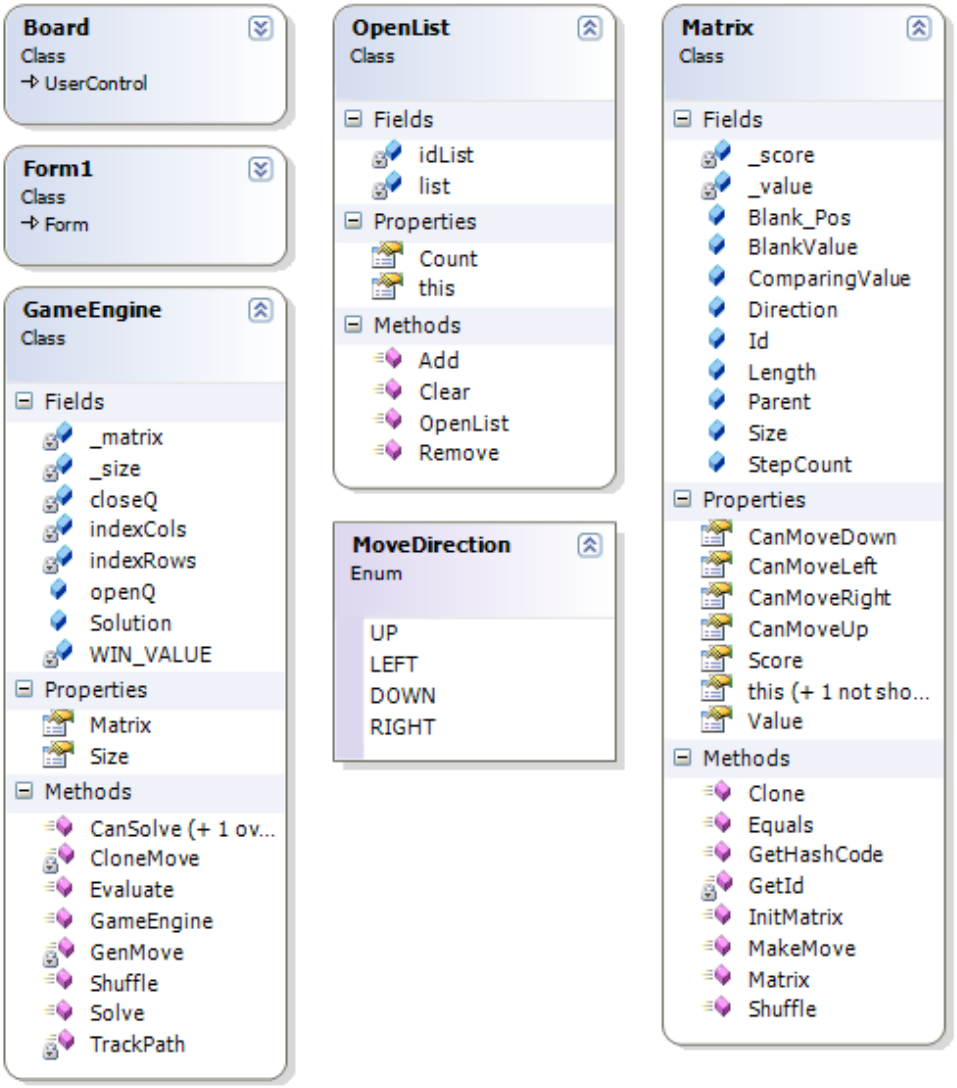
Ví dụ ô số 7 có thứ tự trong bảng là 6 (tính từ 0 với m là cạnh) ta có row = 6 / 3 = 2, col = 6 % 3 = 0. Vậy tổng quát lại ta có:

RowIndex = Index / m

ColIndex = Index % m

IV. Triển khai thuật toán A*

Trước khi tiếp tục bạn nên có một cái nhìn tổng quát về các lớp chính được tôi sử dụng trong project. Sau đó tôi sẽ giải thích một vài đoạn code chính để bạn dễ hiểu.



(https://yinyangit.files.wordpress.com/2010/12/classdiagram.png)

Class **Form** và **Board** thuộc phần giao diện, tôi dùng lớp Board để hiển thị bảng số lên trên form. Bạn có thể dễ dàng sửa lại lớp này để thay đổi cách hiển thị mà không làm ảnh hưởng đến hoạt động của chương trình.

– Lớp **Matrix**: Đại diện cho một bảng số, có thể coi là một node trong cây tìm kiếm khi bạn cài đặt giải thuật.

- Value: mảng lưu bảng số
- Score: lưu trữ giá trị từ hàm heuristic h của bảng số
- ComparingValue: là giá trị dùng để so sánh các phần tử Matrix trong OpenList, là tổng của Score và StepCount (số nước đi từ trạng thái đầu tiên đến trạng thái hiện tại)
- Size: Độ lớn cạnh của bảng số.
- Length: Tổng số phần tử của bảng số
- Parent: Lưu đối tượng cha, là trạng thái trước của trạng thái hiện tại
- Direction: đối tượng kiểu MoveDirection, lưu hướng di chuyển để từ trạng thái trước đó tới trạng thái hiện tại
- Clone(): phương thức này tạo ra một bản sao của đối tượng. Vì Matrix là một lớp có kiểu tham chiếu nên để tạo bản sao bạn không thể gán các biến như với các kiểu giá trị int, double,…
- GetId(): Tạo ra id cho đối tượng, id dựa vào thứ tự sắp xếp các số trong mảng, dĩ nhiên với 2 mảng khác nhau thì id cũng khác nhau. Việc dùng Id sẽ giúp ta kiểm tra và tìm kiếm đối tượng dễ dàng hơn khi chúng ở trong một collection. (Bạn nên cẩn thận khi cho kích thước bảng quá lớn sẽ vượt ngoài phạm vi kiểu int của biến Id).
- MakeMove(MoveDirection): thực hiện "di chuyển" (hoán vị) bảng số dựa vào hướng di chuyển được truyền vào
- Shuffle(): Xáo trộn bảng số

– **OpenList**: Chứa các đối tượng Matrix (node) đã được duyệt tới, khi thêm một node vào danh sách. Ta sẽ chèn nó vào đúng vị trí sao cho OpenList luôn được sắp xếp từ nhỏ đến lớn.

- idList: Danh sách cài đặt bằng HashSet chứa id của các phần tử được thêm vào, dùng để kiểm tra trước khi thêm xem trong OpenList có phần tử đó chưa. Việc lưu trữ dùng mã "băm" sẽ giúp việc tìm kiếm nhanh hơn so với các dạng collection khác.

– **GameEngine**: đối tượng quản lý chung các hoạt động của thuật toán, chứa danh sách OPEN và CLOSE. Danh sách "solution" để lưu lại đường đi tìm được từ trạng thái đầu tiên tới đích.

- Solve(): phương thức chính để giải bài toán.
- Evaluate(): hàm lượng giá Heuristic tính giá trị của một bảng số.
- GenMove(Matrix): Sử dụng phương thức CloneMove(Matrix, MoveDirection) sinh ra các nước đi tiếp theo từ node được truyền vào. Nếu node mới tạo ra đã tồn tại trong CLOSE thì kiểm tra và cập nhật nước đi ngắn hơn cho node.
- TrackPath(): Tạo danh sách các nước đi từ trạng thái đầu tiên đến đích và lưu vào đối tượng solution.

1. Lớp Matrix:

Trong lớp này thay vì sử dụng mảng hai chiều để lưu bảng số, tôi sử dụng mảng 1 chiều để so sánh. Tuy điều này có lợi về lưu trữ và giúp cho một vài đoạn code viết dễ dàng hơn nhưng nó cũng làm một số khác lại trở nên tốn chi phí hơn. Chính vì thế mà hiệu suất của chương trình với việc dùng mảng một chiều có thể coi như tương đương với mảng hai chiều. Tuy nhiên trong phần cuối, tôi sẽ chỉ ra một cách để khắc phục điểm này của mảng một chiều.

```
internal void GetId()
{
    this.Id = 0;
    int n = 1;
    for (int i = 0; i < Length - 1; i++)
    {
        if (_value[i] == BlankValue)
            Blank_Pos = i;
        this.Id += _value[i] * n;
        n *= 10;
    }
}
```

Phương thức này gọi để tự tạo ra Id cho đối tượng. Nó chuyển mảng một chiều thành 1 số có Length-1 chữ số (chữ số cuối cùng không cần xét đến) theo thứ tự ngược lại bảng số. Bạn không cần quan tâm đến thứ tự xếp của Id ngược hay xuôi với bảng số vì chúng không ảnh hưởng gì cả.

```
public bool CanMoveUp
{
    get { return Blank_Pos > Size - 1; }
}
public bool CanMoveDown
{
    get { return Blank_Pos < Length - Size; }
}
public bool CanMoveLeft
{
    get { return GameEngine.IndexCols[Blank_Pos] > 0; }
}
public bool CanMoveRight
{
    get { return GameEngine.IndexCols[Blank_Pos] < Size - 1; }
}
```

Các property trên sẽ được viết rất dễ dàng nếu như bạn dùng mảng hai chiều. Trong bảng số **Size*Size**, để ô trống (**Blank_Pos**) có thể di chuyển lên trên (hay xuống dưới), tức là nó không nằm ở dòng đầu tiên (hoặc dòng cuối nếu như di chuyển xuống) của bảng, và phép kiểm tra rất đơn giản như bạn thấy ở trên.

Tương tự với phép kiểm tra di chuyển trái/phải, ta lấy tọa độ ô trống chia dư cho Size để lấy được tọa độ cột, cuối cùng là so sánh.

```
public override bool Equals(object obj)
{
    Matrix m = obj as Matrix;
    if (m == null)
        return false;
    return this.Id.Equals(m.Id);
}
```

Phương thức này cần được override nếu bạn muốn các collection tìm kiếm đúng đối tượng Matrix mà mình muốn.

2. Lớp GameEngine

Các đoạn mã được tôi đưa phần giải thích vào dưới dạng chú thích.

Phương thức kiểm tra bài toán có cấu hình hợp lệ không (có thể đưa về dạng đích)

```
/// <summary>
/// Kiểm tra xem puzzle có thể chuyển về dạng đích ko
/// Xem thêm tại https://yinyangit.wordpress.com/2010/12/11/algorithm-tim-hi%
/// </summary>
/// <param name="array"></param>
/// <returns></returns>
public bool CanSolve(Matrix matrix)
{
    int value = 0;
    for (int i = 0; i < matrix.Length; i++)
    {
        int t = matrix[i];
        if (t > 1 && t < matrix.BlankValue)
        {
            for (int m = i + 1; m < matrix.Length; m++)
                if (matrix[m] < t)
                    value++;

        }
    }

    if (Size % 2 == 1)
    {
        return value % 2 == 0;
    }
    else
    {

        // Vị trí dòng tính từ 1
        int row = IndexRows[_matrix.Blank_Pos] + 1;
        return value % 2 == row % 2;
    }
}
```

Thuật toán này tôi đã giới thiệu trong bài trước, cách cài đặt cũng đơn giản. Ở phần so sánh cuối bạn có viết như sau, chúng trả về kết quả tương tự:

int row = _matrix.Blank_Pos / Size ;

return value % 2 != row % 2;

Phương thức chính Solve() để giải bài toán rất đơn giản:

```
public void Solve()
{
    // Làm rỗng các collection
    closeQ.Clear();
    openQ.Clear();
    Solution.Clear();

    // Thêm phần tử hiện tại vào OPEN
    this._matrix.Parent = null;
    this._matrix.Score = Evaluate(this._matrix);
    openQ.Add(this._matrix);

    while (openQ.Count > 0)
    {
        // Lấy node có giá trị (ComparingValue) nhỏ nhất
        Matrix m = openQ[0];
        // Kiểm tra xem có phải trạng thái đích
        if (m.Score == WIN_VALUE)
        {
            // Tạo solution
            TrackPath(m);
            return;
        }

        // Xóa node đầu tiên của OPEN
        openQ.Remove(m);
        // Sinh các node tiếp theo của node m
        GenMove(m);
    }
}
```

Và phương thức GenMove():

```
/// <summary>
/// Sinh nước đi
/// </summary>
/// <param name="matrix"></param>
private void GenMove(Matrix matrix)
{
    Matrix m1;
    // nếu node này đã từng xét qua
    if (closeQ.ContainsKey(matrix.Id))
    {
        m1 = closeQ[matrix.Id];
        // Kiểm tra và cập nhật nếu có số nước đi ít hơn node trong CLOSE
        if (matrix.StepCount < m1.StepCount)
            m1 = matrix;
    }
    else
        closeQ.Add(matrix.Id, matrix);

    // Sinh ra các node con
    if (matrix.Direction != MoveDirection.LEFT && matrix.CanMoveRight)
    {
        CloneMove(matrix, MoveDirection.RIGHT);
    }
    if (matrix.Direction != MoveDirection.UP && matrix.CanMoveDown)
    {
        CloneMove(matrix, MoveDirection.DOWN);
    }
    if (matrix.Direction != MoveDirection.RIGHT && matrix.CanMoveLeft)
    {
        CloneMove(matrix, MoveDirection.LEFT);
    }

    if (matrix.Direction != MoveDirection.DOWN && matrix.CanMoveUp)
    {
        CloneMove(matrix, MoveDirection.UP);
    }
}
```

Trong đoạn mã sau:

if (matrix.Direction != MoveDirection.LEFT && matrix.CanMoveRight)
{

CloneMove(matrix, MoveDirection.RIGHT);

}

Lý do tôi kiểm tra hướng của node hiện tại vì nếu node cha của nó ở bên phải thì việc sinh nước đi bên phải là không cần thiết. Ở đây direction chính là hướng đi để một node cha biến trở thành node con.

Phương thức lượng giá heuristic:

```
1  public int Evaluate(Matrix matrix)
2  {
3      // Ô nằm sai vị trí bị cộng điểm bằng khoảng cách ô đó đến vị trí đúng
4      int score = 0;
5
6      for (int i = 0; i < matrix.Length; i++)
7      {
8          int value = matrix[i] - 1;
9          score += Math.Abs(IndexRows[i] - IndexRows[value]) + Math.Abs(IndexCo
10     }
11     return score;
12 }
```

Với mã nguồn tham khảo trên, bạn có thể tự cài đặt một project để giải các bài toán tương tự. Tuy nhiên tốc độ giải của chương trình chưa thật sự làm tôi hài lòng. Vì thế trong khi viết bài này tôi đã thực hiện một vài ý tưởng nhỏ để cải tiến tốc độ, và sẽ cập nhật vào mã nguồn được đính kèm bên dưới. Bạn có thể bỏ qua phần tôi sắp trình bày nếu thấy không cần thiết.

V. Một vài hướng cải thiện tốc độ chương trình

– **Hạn chế các tính toán lặp lại nhiều lần:** Trong chương trình này, ta khó có thể cải tiến chương trình để vừa làm tăng tốc độ, vừa làm giảm bớt bộ nhớ sử dụng. Ở đây ta nhận thấy, với một chương trình nhỏ dạng này, việc sử dụng bộ nhớ thêm một chút cũng không ảnh hưởng gì, và cái ta cần là tốc độ tính toán.

Ví dụ, thử xem lại phương thức Evaluate() trong lớp GameEngine, ta phải thực hiện tính toán để đổi từ một giá trị sang hai giá trị dòng và cột. Các giá trị này chỉ nằm trong khoảng từ 0 đến độ dài của mảng lưu trữ. Mỗi lần tạo ra một node mới ta lại tính lại một phép tính tương tự. Vậy để cải tiến, ta chỉ việc lưu trữ sẵn những giá trị này trong mảng và sử dụng nó thông qua giá trị đó.

Tạo hai mảng int toàn cục lớp trong GameEngine:

public static int[] IndexRows;

public static int[] IndexCols;

Tôi sử dụng public và static để có thể dùng được trong các lớp khác, nếu bạn cảm thấy nó làm chương trình thêm rắc rối thì chỉ cần để private.

Trong property Size của lớp GameEngine, ta sẽ khởi tạo giá trị cho hai mảng trên:

```
public int Size
{
    get { return _size; }
    set
    {
        _size = value;
        _matrix = new Matrix(_size);

        int m = _size * _size;
        IndexRows = new int[m];
        IndexCols = new int[m];
        for (int i = 0; i < m; i++)
        {
            IndexRows[i] = i / _size;
            IndexCols[i] = i % _size;
        }

    }
}
```

Bạn có thể ta tính sẵn giá trị dòng và cột của mọi phần tử trong bảng số. Các giá trị này chỉ được tính một lần duy nhất mỗi khi Size được gán nên bạn không phải lo lắng về sự lãng phí hay thừa thãi của nó. Mọi phần tử của hai mảng này đều được dùng đến. Ta sửa lại phương thức Evaluate() như sau:

```
public int Evaluate(Matrix matrix)
{
    int score = 0;

    for (int i = 0; i < matrix.Length; i++)
    {
        int value = matrix[i] - 1;

        score += Math.Abs(IndexRows[i] - IndexRows[value]) + Math.Abs(IndexCo
    }
    return score;
}
```

Bạn có thể kiểm tra và nhận thấy chúng không có sự khác biệt về tốc độ lắm so với phiên bản cũ. Tuy nhiên khi bảng số có kích thước lớn, sự cải thiện này sẽ thể hiện ưu điểm của nó rõ ràng hơn. Cụ thể khi tính toán trong phiên bản 15-puzzle, phương thức Evaluate() này chạy nhanh hơn 4 lần so với cách cũ.

– **Dùng mã lệnh ưu tiên tốc độ**: Bạn có thể tham khảo thêm bài viết (https://yinyangit.wordpress.com/2009/09/17/ctang-hi%e1%bb%87u-su%e1%ba%a5t-ch%c6%b0%c6%a1ng-trinh-c-p1/) của tôi về cải thiện hiệu suất chương trình C#. Ở đây tôi nói "ưu tiên" tức là ta phải chịu thiệt một chút gì đó để bù lại tốc độ, đó thường là làm mã nguồn khó hiểu hơn, thiếu tính OOP. Ví dụ như trong chương trình này tôi sử dụng properties rất ít, nguyên nhân là vì nếu truy xuất trực tiếp biến thì sẽ nhanh hơn.

– **Chấp nhận lời giải tương đối**: Dĩ nhiên nếu bạn chỉ quan tâm đến việc có tìm được lời giải hay không, còn chất lượng lời giải không quan trọng thì bạn nên suy nghĩ đến hướng này. Bạn có thể đi tìm như những thuật toán khác, chẳng hạn như các thuật toán tìm kiếm theo chiều sâu Depth-first search, Iterative deepening search. Chẳng hạn trong project này nếu tôi vẫn sử dụng A* và bỏ đi vấn đề tìm đường đi ngắn nhất (tức là chỉ so sánh dựa vào giá trị của hàm heuristic) thì kết quả tìm đường đi của phiên bản 8-puzzle gần như ngay lập tức.

VI. Lời kết

Kết thúc bài viết này, bạn có thể mở rộng bài toán và giải được những bảng số có dạng mxn, cách làm cũng tương tự tuy nhiên bạn cần kiểm tra lại cách thức để kiểm tra xem trạng thái của bảng số có hợp lệ không. Chương trình này có thể tạm chấp nhận với mức 8-puzzle, tuy nhiên nếu muốn giải được bài toán với n tương đối lớn thì ta cần tìm một giải pháp khác.

https://yinyangit.wordpress.com (https://yinyangit.wordpress.com/)

Bài này đã được đăng trong Algorithms và được gắn thẻ AI, N-puzzle. Đánh dấu đường dẫn tĩnh.

thoughts on “Algorithm – Phân tích và giải bài toán n-puzzle”

Trinh nói: *Thứ Sáu, Tháng Mười Hai 17, 2010 lúc 2:48 sáng*

0

0

i
Rate This

Tung oi, sao ko tao cong cu “tao bản in” ha T? tớ mún in ra để đọc…..:(

Yin Yang nói: *Thứ Hai, Tháng Mười Hai 20, 2010 lúc 3:42 chiều*

1

0

i
Rate This

Đã thêm chức năng in trang (1 Jan 2011)

Nguyễn Duy Đức nói: *Thứ Năm, Tháng Tư 28, 2011 lúc 8:39 chiều*

0

0

i

Rate This

Bạn ơi,bạn tìm ra hướng giải quyết của n-puzzle chưa???(n>8)
Giúp mình với nhé.Cám ơn bạn nhiều

Yin Yang nói: *Thứ Bảy, Tháng Tư 30, 2011 lúc 1:40 chiều*

0

0

i
Rate This

Chương trình n-puzzle solver này có thể được thay đổi để chạy tốt với 15-puzzle với cùng thuật toán. Tuy nhiên vì thiên về mục đích học tập nên mình coi trọng mục đích dễ hiểu hơn là tối ưu. Với n bất kì, có thể sử dụng giải thuật khác như: xếp các cạnh bên trước, xếp từng hàng,… Các giải thuật này không đảm bảo ra lời giải tốt nhưng sẽ cho ra kết quả nhanh chóng.

Hiện tại mình đang bận nên chưa tiếp tục nghiên cứu vấn đề này, khi nào rảnh sẽ tiếp tục đưa ra phiên bản n-puzzle cao hơn.

Thân!

linh nói: *Thứ Tư, Tháng Tám 3, 2011 lúc 3:25 chiều*

0

0

i
Rate This

bạn ơi cho mình hỏi ở cái hàm
public int Evaluate(Matrix matrix)
thì int value = matrix[i] – 1;
mình thấy gia trị của bảng số là mảng value[] cơ mà tại sao matrix[i] được nhỉ lại thế nhỉ(mình mới học ko biết mong bạn chỉ giao^^)

Yin Yang nói: *Thứ Tư, Tháng Tám 3, 2011 lúc 4:37 chiều*

0

0

i
Rate This

Bạn có thể thấy trong lớp Matrix mình tạo một indexer với tham số truyền vào là index:

```
public int this[int index]
{
get { return _value[index]; }
set { _value[index] = value; }
}
```

dựa vào indexer này ta sẽ lấy được giá trị tương ứng từ mảng value mà không cần truy xuất đến mảng này.
Thân!

trần hiền nói: *Thứ Năm, Tháng Chín 15, 2011 lúc 10:03 sáng*

0

0

i
Rate This

bạn có thể giải thích giùm mình hàm này được không?
public override bool Equals(object obj)
{
Matrix m = obj as Matrix;
if (m == null)
return false;
return this.Id.Equals(m.Id);
}
mình đọc mà vẫn không hiểu

Yin Yang nói: *Thứ Năm, Tháng Chín 15, 2011 lúc 3:56 chiều*

1

0

i
Rate This

Phương thức Equals() dùng để so sánh hai đối tượng với nhau xem có bằng nhau không (dựa trên một hay nhiều property của chúng). Phương thức này được gọi bởi một đối tượng (ví dụ Matrix) và yêu cầu tham số là một đối tượng khác để so sánh với chính nó (trong đoạn mã trên là obj).

Đầu tiên ta sẽ chuyển kiểu của obj sang Matrix, nếu như obj không phải kiểu Matrix thì biến m sẽ là null. Vì vậy ta kiểm tra nếu m = null thì sẽ return false. Tất nhiên khi hai đối tượng không cùng kiểu thì chúng sẽ khác nhau.

Ngược lại thì ta sẽ so sánh dựa vào Id của đối tượng hiện tại với đối tượng truyền vào trong tham số

bool value = this.Id.Equals(m.Id);
return value;

trần hiền nói: *Thứ Bảy, Tháng Chín 17, 2011 lúc 8:17 sáng*

0

0

i
Rate This

closeQ = new SortedList();
bạn có thể nói cho mình kểu khai báo trên là khai báo cái gì ko?

Yin Yang nói: *Thứ Bảy, Tháng Chín 17, 2011 lúc 2:12 chiều*

0

0

i
Rate This

SortedList là một collection tự động sắp xếp các phần tử của nó khi được thêm vàobạn có thể coi tại link sau:
http://msdn.microsoft.com/en-us/library/system.collections.sortedlist.aspx

trần hiền nói: *Thứ Bảy, Tháng Chín 17, 2011 lúc 8:41 sáng*

0

0

i
Rate This

đoạn code này là gì nhỉ????
```
if (Size % 2 == 1)
{
return value % 2 == 0;
}
else
{
// Vị trí dòng tính từ 1
int row = IndexRows[_matrix.Blank_Pos] + 1;
return value % 2 == row % 2;
}
}
```

Yin Yang nói: *Thứ Bảy, Tháng Chín 17, 2011 lúc 2:14 chiều*

0

0

i
Rate This

Bạn vui lòng coi tại đây:
https://yinyangit.wordpress.com/2010/12/11/algorithm-tim-hi%e1%bb%83u-v%e1%bb%81-bai-toan-n-puzzle-updated/

trần hiền nói: *Chủ Nhật, Tháng Chín 18, 2011 lúc 2:27 sáng*

0

0

i
Rate This

cảm ơn bạn rất nhiều

TuanClick nói: *Chủ Nhật, Tháng Chín 18, 2011 lúc 9:54 chiều*

0

0

i
Rate This

Mình hỏi 1 chút về hàm này
internal void GetId()
{
this.Id = 0;
int n = 1;
for (int i = 0; i < Length – 1; i++)
{
if (_value[i] == BlankValue)
Blank_Pos = i;
this.Id += _value[i] * n;
n *= 10;
}
}
Khi debug chương trình của bạn thì với puzzle 15 và 24 thì máy treo. Có phải do biến n trong đoạn trên vượt quá giá trị của kiểu int .Theo mình biết thì kiểu int trong C# giá trị có thể nhận được là 2*10^9.Đoạn trên có thể bỏ dòng n*=10 dc ko? thay vào đó mình khởi tạo giá trị n=2 dc ko?.

Yin Yang nói: *Thứ Hai, Tháng Chín 19, 2011 lúc 12:01 chiều*

0

0

i
Rate This

Việc tạo id này dựa trên thứ tự của các ô trong bảng. Nếu bạn có thể sử dụng cách tạo id khác hoặc không cần tạo id thì nên thử. Dự định của mình là viết lại ví dụ này bằng thuật toán tìm kiếm sâu dần nhưng chưa có thời gian. Nếu có thể thì bạn nên thử bằng thuật toán này xem sao.

Hoài Nam nói: *Thứ Năm, Tháng Mười Hai 1, 2011 lúc 8:14 chiều*

0

0

i

Rate This

Anh ơi cho em hỏi trong đoạn code dưới đây mục đích của việc cập nhật trong tập Close là để làm gì vậy? hình như em thấy việc cập nhật này không có mục đích gì cả? thay vào đó mình phải cập nhật trong Open mới đúng phải không anh?

```
private void CloneMove(Matrix parent, MoveDirection direction)
{
// Tạo và cập nhật các giá trị cho node con
Matrix m = parent.Clone();
m.MakeMove(direction);
m.Direction = direction;
m.StepCount++;

// Nếu node đã có trong CLOSE
if (closeQ.ContainsKey(m.Id))
{
Matrix m1 = closeQ[m.Id];
if (m.StepCount < m1.StepCount)
m1 = m;
}
else
{
// Đặt các thuộc tính và thêm vào OPEN
m.Parent = parent;
m.Score = Evaluate(m);
openQ.Add(m);
}
}

private void GenMove(Matrix matrix)
{
Matrix m1;
// nếu node này đã từng xét qua
if (closeQ.ContainsKey(matrix.Id))
{
m1 = closeQ[matrix.Id];
// Kiểm tra và cập nhật nếu có số nước đi ít hơn node trong CLOSE
if (matrix.StepCount < m1.StepCount)
m1 = matrix;
}
else
closeQ.Add(matrix.Id, matrix);

// Sinh ra các node con
if (matrix.Direction != MoveDirection.LEFT && matrix.CanMoveRight)
{
CloneMove(matrix, MoveDirection.RIGHT);
}
if (matrix.Direction != MoveDirection.UP && matrix.CanMoveDown)
{
CloneMove(matrix, MoveDirection.DOWN);
}
if (matrix.Direction != MoveDirection.RIGHT && matrix.CanMoveLeft)
```

```
{
CloneMove(matrix, MoveDirection.LEFT);
}

if (matrix.Direction != MoveDirection.DOWN && matrix.CanMoveUp)
{
CloneMove(matrix, MoveDirection.UP);
}
}
```

Yin Yang nói: *Thứ Sáu, Tháng Mười Hai 2, 2011 lúc 12:17 sáng*

0

0

i
Rate This

Trong comment của code trên mình đã ghi rồi đó. Tập Close có mục đích lưu trữ các nước đã đi rồi. Nó có 2 tác dụng:

1. Ko duyệt lại các nước đã đi
2. Nếu nước đi hiện tại có số bước nhiều hơn 1 nước đi tương ứng trong Close thì sẽ cập nhật lại nước đi đó.

Hoài Nam nói: *Thứ Sáu, Tháng Mười Hai 2, 2011 lúc 6:49 sáng*

0

0

i
Rate This

ah, em hiểu rùi cảm ơn anh!! chúc anh luôn vui vẻ và hạnh phúc!

anymous nói: *Thứ Bảy, Tháng Năm 12, 2012 lúc 1:52 chiều*

0

0

i
Rate This

cho mình hỏi chức năng của tập Close nha, có gì khác biệt giữa tập Open với tập Close, mình đọc hoài mà ko hiếu, mong bạn giải thích giùm mình.

Yin Yang nói: *Thứ Bảy, Tháng Năm 12, 2012 lúc 4:00 chiều*

0

0

i
Rate This

Rất đơn giản.
Open: lưu các trạng thái sắp được duyệt.
Close: các trạng thái đã được duyệt.

Hoài Nam nói: *Chủ Nhật, Tháng Năm 13, 2012 lúc 11:28 sáng*

0

0

i
Rate This

Đây là chương trình 8-puzzle của anh. em viết lại bằng mảng 2 chiều. và được đã được kiểm tra trên trang 8puzzle.com
Ảnh: http://upanh.ssc.vn/images/3138puzzle.png
exe: http://www.mediafire.com/?97ssgu06cmoaj8j (.Netframe work 4.0)

Yin Yang nói: *Chủ Nhật, Tháng Năm 13, 2012 lúc 11:52 sáng*

0

0

i
Rate This

Kinh nghiệm của mình sau khi làm ví dụ n-puzzle này là sử dụng thuật toán tìm kiếm sâu dần (iterative deepening search algorithm) như 8puzzle.com. Số nước tìm kiếm và tốc độ tìm kiếm có thể nhanh hơn A*.

Nếu bạn có những ý kiến hoặc kinh nghiệm gì khi thực hiện demo này có thể cung cấp và mình có thể mạn phép post lên để giới thiệu.

Hoài Nam nói: *Chủ Nhật, Tháng Năm 13, 2012 lúc 10:14 chiều*

0

0

i
Rate This

Demo ở trên em vẫn sử dụng A* của anh tuy nhiên em có chỉnh sửa 1 chút trong lớp PQ, và thật toán A*, vì khi debug thử chương trình của anh, em thấy nó có bỏ qua 1 số trạng thái dẫn đến trạng thái đích nhanh hơn. Còn vấn đề với IDS, em cung đã có làm thử nhưng em thấy nó hơi chậm (em sử dụng hàng đợi với bước lặp tăng dần-theo slide bài giảng em tìm trên mạng.)
Đây là code chương trình em đã làm dựa trên nền tảng của anh.với các thuật toán DFS với độ sâu giới hạn, BrFS, IDS.
http://www.mediafire.com/?s3cgald4cg8ph3c

Hoài Nam nói: *Chủ Nhật, Tháng Năm 13, 2012 lúc 10:15 chiều*

0

0

i
Rate This

Chúc anh ngày càng có nhiều sản phẩm hay hơn nữa.

> **Yin Yang** nói: *Thứ Hai, Tháng Năm 14, 2012 lúc 3:51 chiều*
>
> 0
>
> 0
>
> i
> Rate This
>
> Cảm ơn bạn đã chia sẻ kinh nghiệm và code, chúc bạn luôn thành công trong cuộc sống.

hùng nói: *Thứ Ba, Tháng Năm 22, 2012 lúc 11:13 chiều*

0

0

i
Rate This

anh ơi, giải thích giúp e hàm shuffle() với, e chưa hiểu lắm, cái quy luật trộn ô ý a

> **Yin Yang** nói: *Thứ Tư, Tháng Năm 23, 2012 lúc 2:43 chiều*
>
> 0
>
> 0
>
> i
> Rate This
>
> Chỉ đơn giản là lặp qua mảng với biến lưu chỉ số i, trong thân vòng lặp ta lấy 1 vị trí j random và hoán vị phần tử tại i với phần tử tại j.

Nhàn Trịnh nói: *Chủ Nhật, Tháng Mười 16, 2016 lúc 8:55 chiều*

0

0

i
Rate This

Anh ơi, cho em hỏi ý nghĩa của đoan code này là gì ko ạ?
Em cảm ơn!

```
public void MakeMove(MoveDirection direction)
{
int position;
if (direction == MoveDirection.UP)
position = Blank_Pos – Size;
else if (direction == MoveDirection.DOWN)
position = Blank_Pos + Size;
else if (direction == MoveDirection.LEFT)
position = Blank_Pos – 1;
else
position = Blank_Pos + 1;

_value[Blank_Pos] = _value[position];
_value[position] = this.BlankValue;

Blank_Pos = position;
GetId();
}
```

Đã đóng bình luận.