**Báo cáo môn Lý thuyết đồ thị**

Đề tài : Game line

Sinh viên : Lê Nguyên Dũng

Email : [dungcoi@virusvn.com](mailto:dungcoi@virusvn.com)

Giới thiệu đề tài :

Đề tài mô phỏng lại hoàn chỉnh các thao tác của một game thông dụng là Line.

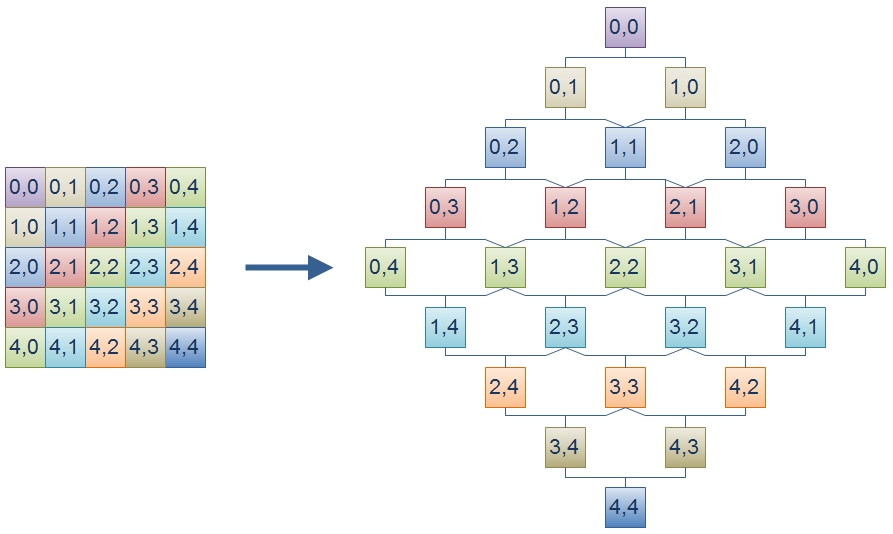
Vấn đề đồ thị trong game :

1. Chuyển từ ma trận bàn cờ qua đồ thị
   1. Ý tưởng giải thuật
   2. Số ô nhớ tốn kém để lưu
   3. Các phương thức lưu trữ
      1. Đề xuất phương pháp tối ưu
      2. Phương pháp được sử dụng
2. Bài toán tìm đường đi ngắn nhất
   1. Thuật toán loang
      1. Ý tưởng
      2. Mô phỏng thuật toán
      3. Demo bước loang
      4. Thực tiễn code
   2. Thuật toán heuristic
      1. Ý tưởng
      2. Mô phỏng thuật toán
      3. Demo bước giải
      4. Vấn đề hình học và hàm heuristic
      5. Thực tiễn code

**1. Chuyển từ ma trận bàn cờ qua đồ thị :**

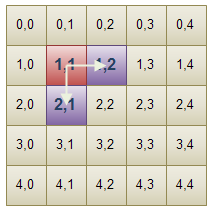
1.a. Ý tưởng giải thuật :

Đưa ma trận bàn cờ từ dạng ma trận qua danh sách các điểm kề theo hình sau :



Ý tưởng để thiết lập danh sách kề là :

* Lặp xét hết từng điểm trên ma trận
* Tại mỗi điểm, chỉ xét có tạo cạnh liên thông với 2 điểm lân cận tiến.



*Chúng ta chỉ xét 2 điểm do thuật toán này mang tính lặp, nên các điểm lân cận trên sẽ cũng được xét và tạo cạnh với điểm đang được xét.*

1.b. Số ô nhớ tốn để lưu trữ

Do số điểm của bàn cờ là cố định và có một mức bậc cao nhất mà điểm có thể mang. Nên ở đây chúng ta có thể tìm ra số ô nhớ tối ưu cần lưu trữ để tiết kiệm tới mức tối đa số bộ nhớ phải lưu trữ.

Cho **N** là **độ dài cạnh bàn cờ vuông**, số cạnh tối đa (Cũng chính là số ô nhớ mà danh sách kề phải dùng) là : **2x(N-1)2 + 2xN - 1**

*(Chúng ta có thể dễ dàng tìm ra công thức trên dựa vào hình vẽ)*

Cụ thể ở đây bàn cờ có kích thước là 10x10, vậy số ô nhớ cần khai báo để lưu là :

**2x92 + 2x10 – 1 = 181**

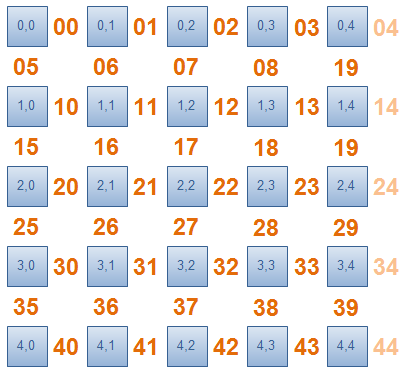
1.c. Các phương thức lưu trữ

**Cách thức lưu trữ ma trận kề đơn giản, nhưng không được chọn do không tốt do tốn quá nhiều bộ nhớ**

1.c.i. Đề xuất phương pháp tối ưu

Đây là cách thức em đã thực hiện tuy nhiên chưa hoàn thành do tốn quá nhiều thời gian. Ở đây em chỉ xin ghi lại.

Ý tưởng chính của cách thức lưu danh sách cạnh ở thay vì lưu cả danh sách thì sẽ gây ra sự lãng phí rất lớn cho những thao tác lặp thì ở đây vẫn sẽ lưu danh sách nhưng sẽ theo một cấu trúc tĩnh. Dựa vào đó ở mỗi thao tác tìm yếu tố cạnh kề thì có thể kết hợp với yếu tố tọa độ của điểm để tránh việc phải lặp quá nhiều lần.



Thầy có xem hàm tìm kiếm ít tốn kém chi phí hơn tại hàm **FindPath3** và hàm khởi tạo **initWalk3** em vẫn còn giữ trong source.

Ví dụ :

**Code :**

**For iCount = 0 To 3**

If (pos2canhKe(i, iCount, iNext) >= 0) Then

'Tìm điểm kề điểm hiện tại

If pathVisited(iNext) = 0 Then

pathVisited(iNext) = tim + 1

End If

If (iNext = iTo) Then

isFounded = True

End If

End If

Next iCount

Private Function pos2canhKe(ByVal iPoint As Long, ByVal lVal As Long, ByRef iRetPoint As Long) As Long

'Hàm được gọi để trả về các cạnh tương ứng có thể có của 1 điểm

'lVal : Vị trí cạnh

' 1 : Trên

' 0 : Trước Point 2 : Sau

' 3 : Dưới

Dim pos As Point

pos = index2pos(iPoint)

Dim pNext As Point

Dim lRes As Long

If lVal = 2 Then

lRes = pos.y \* 10 \* 2 + pos.x

pNext.x = pos.x + 1

pNext.y = pos.y

ElseIf lVal = 3 Then

lRes = pos.y \* 10 \* 2 + 10 + pos.x

pNext.x = pos.x

pNext.y = pos.y + 1

ElseIf lVal = 0 Then

lRes = pos.y \* 10 \* 2 + pos.x - 1

pNext.x = pos.x - 1

pNext.y = pos.y

ElseIf lVal = 1 Then

lRes = (pos.y - 1) \* 10 \* 2 + 10 + pos.x

pNext.x = pos.x

pNext.y = pos.y - 1

Else

Return -1

End If

iRetPoint = pos2index(pNext)

If ((lRes >= 0) And (lRes <= 189)) And (iRetPoint >= 0) And (iRetPoint <= 99) Then

Return lRes

Else

Return -1

End If

End Function

Thay cho đoạn sau trong thuật toán ban đầu :

**Code :**

For iCount = 0 To wWalk.lCount

If (wWalk.lWalk(iCount).pos1 = i) Or (wWalk.lWalk(iCount).pos2 = i) Then

'Tìm điểm kề điểm hiện tại

If wWalk.lWalk(iCount).pos1 = i Then

iNext = wWalk.lWalk(iCount).pos2

Else

iNext = wWalk.lWalk(iCount).pos1

End If

If pathVisited(iNext) = 0 Then

pathVisited(iNext) = tim + 1

End If

If (iNext = iTo) Then

isFounded = True

End If

End If

Next iCount

1.c.ii. Phương pháp được sử dụng

Tương tự như cách xây dựng 1 mảng thông thường nhưng nếu dạng mảng hay danh sách liên kết sẽ tốn nhiều thời gian tái cấu trúc mỗi khi thêm bớt.

Ở đây em sử dụng một công cụ có sẵn rất lợi thế của các ngôn ngữ lập trình hỗ trợ GUI xây dựng sẵn là ListBox.

**Code :**

Private Sub initWalk()

'Thiết lập danh sách kề

lstGraph1.Items.Clear()

lstGraph2.Items.Clear()

Dim iX As Integer

Dim iY As Integer

For iX = 0 To 189

wWalk(iX) = False

Next

Dim iCount As Integer

Dim curPos As Point

Dim pNext As Point

For iX = 0 To 9

For iY = 0 To 9

If iPixel(iX, iY) < 0 Then

'Nếu quân cờ đang xét hiện nay là rỗng

curPos.x = iX

curPos.y = iY

For iCount = 0 To 1

pNext.x = curPos.x + unitStep(iCount).x

pNext.y = curPos.y + unitStep(iCount).y

If (pNext.x >= 0) And (pNext.x <= 9) And (pNext.y >= 0) And (pNext.y <= 9) Then

'Kiểm tra tính hợp lệ của điểm kề (tiến) được xét

If iPixel(pNext.x, pNext.y) < 0 Then

'Nếu là rỗng

lstGraph1.Items.Add(pos2index(curPos))

lstGraph2.Items.Add(pos2index(pNext))

End If

End If

Next iCount

End If

Next

Next

End Sub

Do tại bất kỳ thay đổi nào của tọa độ thì đều gọi hàm **setPixel** nên chỉ cần hiệu chỉnh thêm yếu tố hiệu chỉnh ListBox để không phải tạo lại danh sách liên tục.

**Code :**

Private Sub setPixel(ByVal pos As Point, ByVal iValue As Integer)

Dim iIndex As Integer

Dim pNext As Point

Dim iCount As Integer

iIndex = pos2index(pos)

If iValue <= -1 Then

pPixel(iIndex).Image = Nothing

iPixel(pos.x, pos.y) = -1

'Khởi tạo thêm các giá trị điểm kề của điểm bắt đầu

For iCount = 0 To 3

pNext.x = pos.x + unitStep(iCount).x

pNext.y = pos.y + unitStep(iCount).y

If (pNext.x >= 0) And (pNext.x <= 9) And (pNext.y >= 0) And (pNext.y <= 9) Then

If (iPixel(pNext.x, pNext.y) < 0) Then

With wWalk

lstGraph1.Items.Add(pos2index(pos))

lstGraph2.Items.Add(pos2index(pNext))

End With

End If

End If

Next iCount

Else

pPixel(iIndex).Image = imgList.Images.Item(iValue)

iPixel(pos.x, pos.y) = iValue

Dim iX As Integer

Dim iTmp As Integer

'Xóa cạnh trong list

iTmp = pos2index(pos)

For iX = lstGraph1.Items.Count - 1 To 0 Step -1

If (lstGraph1.Items(iX) = iTmp) Or (lstGraph2.Items(iX) = iTmp) Then

lstGraph1.Items.RemoveAt(iX)

lstGraph2.Items.RemoveAt(iX)

End If

Next

End If

End Sub

**2. Bài toán tìm đường đi ngắn nhất**

2.a. Thuật toán loang :

2.a.i. Ý tưởng :

Duyệt dần các điểm trên đồ thị này bằng cách sau khi đi tới 1 điểm, đánh dấu điểm đó đã đi và tiếp tục duyệt các điểm lân cận của điểm đó. Để duyệt những điểm kề của một điểm chúng ta tiến hành lặp để tìm kiếm có điểm hiện tại trong dach sách kề hay không.

Trong mỗi bước loang ta có sự đánh dấu hệ số , ở các bước loang sau hệ số này sẽ được tăng dần cho đến khi gặp điểm đích.

2.a.ii. Mô phỏng thuật toán:

*(Ban đầu em tính viết mã giả, nhưng do chưa quen viết nên tạm thời em chỉ viết được mô phỏng thuật toán theo ý em)*

**Thực hiện loang trên danh sách cạnh kề**

tim := 1

pathVisited(s) := tim

while (isFound=False)

{

*Lặp danh sách kề ma trận tìm x sao*

if (pathVisited(x) = tim)

{

*Lặp tìm các lân cận của x*

pNext = lân cận x

pathVisited(pNext) := tim+1

if (pathVisited(pNext) = Điểm đến)

isFound = True

}

dec(tim)

if (isFound=False)

{

if (sum = lastsum)

{

*2 điểm không liên thông*

exit function

}

else

{

lastsum = sum

}

}

}

**Thực hiện tìm đường trên đồ thị**

if (isFound=True)

{

curPos = Điểm đến

trace(tim) := curPos

while (pathVisited(curPos)) > 1

{

*Lặp tìm điểm x lân cận của curPos thỏa bằng tim-1*

if (pathVisited(x) = tim-1)

{

*trace(tim-1) := x*

*curPos := x*

*tim := tim-1*

}

}

}

**Kết quả trả về :**

Độ dài đường đi : *tim*

Đường đi : *Mảng trace*

2.a.iii. Demo các bước loang :

|  |  |  |
| --- | --- | --- |
| **1.** | **2.** | **3.** |
| **4.** | **5.** | **6.** |
| **7.** | **8.** | **9-10.** |

Kết quả :



Minh họa một trường hợp có chướng ngại vật :



2.a.iv. Thực tiễn code :

**Code :** Tiến hành loang

Private Sub findPath(ByVal pFrom As Point, ByVal pTo As Point)

'Tìm kiếm loang sử dụng ListBox

Dim pNext As Point

Dim iCount As Integer

'Khởi tạo thêm các giá trị điểm kề của điểm bắt đầu

For iCount = 0 To 3

pNext.x = pFrom.x + unitStep(iCount).x

pNext.y = pFrom.y + unitStep(iCount).y

If (pNext.x >= 0) And (pNext.x <= 9) And (pNext.y >= 0) And (pNext.y <= 9) Then

If (iPixel(pNext.x, pNext.y) < 0) Then

With wWalk

lstGraph1.Items.Add(pos2index(pFrom))

lstGraph2.Items.Add(pos2index(pNext))

End With

End If

End If

Next iCount

'Bắt đầu tìm kiếm từ iFrom

Dim iFrom As Integer

'Điểm đích iTo

Dim iTo As Integer

iTo = pos2index(pTo)

Dim tim As Long

tim = 1

Dim pathVisited(99) As Integer

Dim i As Integer

For i = 0 To 99

pathVisited(i) = 0

Next

iFrom = pos2index(pFrom)

Dim isFounded As Boolean

isFounded = False

Dim sum As Long

Dim lastsum As Long

pathVisited(iFrom) = 1

Dim iNext As Integer

Do While isFounded = False

For i = 0 To 99

If pathVisited(i) = tim Then

For iCount = 0 To lstGraph1.Items.Count - 1

If (lstGraph1.Items(iCount) = i) Or (lstGraph2.Items(iCount) = i) Then

'Tìm điểm kề điểm hiện tại

If lstGraph1.Items(iCount) = i Then

iNext = Val(lstGraph2.Items(iCount))

Else

iNext = Val(lstGraph1.Items(iCount))

End If

If pathVisited(iNext) = 0 Then

pathVisited(iNext) = tim + 1

End If

If (iNext = iTo) Then

isFounded = True

End If

End If

Next iCount

End If

Next

If isFounded = False Then

sum = 0

Dim i1 As Long

For i1 = 0 To 99

sum = sum + pathVisited(i1)

Next i1

If sum = lastsum Then

'MsgBox("thu@")

Exit Do

Else

lastsum = sum

End If

End If

tim = tim + 1

Loop

**Code :** Tìm đường đi ngắn nhất vào kết quả loang thu được

If isFounded = True Then

'Nếu tìm được đường đi

trace.iLen = tim - 1

ReDim trace.pos(trace.iLen)

Dim curIndex As Integer

curIndex = iTo

Do While pathVisited(curIndex) > 1

For iCount = 0 To lstGraph1.Items.Count - 1

If (lstGraph1.Items(iCount) = curIndex) Or (lstGraph2.Items(iCount) = curIndex) Then

'Tìm điểm kề điểm hiện tại

If lstGraph1.Items(iCount) = curIndex Then

iNext = Val(lstGraph2.Items(iCount))

Else

iNext = Val(lstGraph1.Items(iCount))

End If

If pathVisited(iNext) = tim - 1 Then

trace.pos(tim - 1) = index2pos(iNext)

curIndex = iNext

tim = tim - 1

Exit For

End If

End If

Next iCount

Loop

'Tiến hành vẽ cờ di chuyển

Dim iIndex As Integer

For iCount = 1 To trace.iLen

iIndex = pos2index(trace.pos(iCount))

pPixel(iIndex).Image = imgGo.Images.Item(0)

Next

Else

trace.iLen = 0

Dim iX As Integer

Dim iTmp As Integer

'Xóa cạnh trong list

iTmp = pos2index(pFrom)

For iX = lstGraph1.Items.Count - 1 To 0 Step -1

If (lstGraph1.Items(iX) = iTmp) Or (lstGraph2.Items(iX) = iTmp) Then

lstGraph1.Items.RemoveAt(iX)

lstGraph2.Items.RemoveAt(iX)

End If

Next

End If

End Sub

2.b. Thuật toán heuristic :

2.b.i. Ý tưởng :

Tận dụng yếu tố tọa độ, chúng ta đi tìm một giải thuật “thông minh” thuật toán đã sử dụng (Loang) nhầm tối ưu hóa về số thao tác tính toán.

Về cơ bản ở đây, thuật giải heuristic là sự loang có định hướng dựa vào hàm heuristic.

Hàm heuristic là dựa vào các chỉ số tọa độ hiện tại của điểm hiện tại với điểm đích với mong muốn với bước đi như thế nào đó sẽ là đúng và trùng với bước đi ngắn nhất.

Tuy nhiên không phải mọi trường hợp heuristic cũng sử dụng số bước tính toán là ít nhất, thậm chí đôi lúc đường đi mà thuật heuristic tìm ra còn dài hơn các thuật toán vét cạn khác (Ở trên chúng ta đã sử dụng loang).

2.b.i. Mô phỏng thuật toán :

**Thực hiện tìm kiếm heuristic trên danh sách**

{Điểm s là điểm bắt đầu, điểm f là điểm kết thúc}

<Khởi tạo stack rỗng>

push s

pathVisited(s) := 1

isFound := false

while (<Nếu stack chưa rỗng>)

{

pop x

if (x == f)

isFound := true

<Hàm heristic chọn các điểm ưu tiên, cho một biến đại diện chung là **r**>

<push r>

r := pathVisited(x)+1

}

**Thực hiện tìm đường trên đồ thị**

if (isFound=True)

{

tim := pathVisited(f)

curPos = f

trace(tim) := curPos

while (pathVisited(curPos)) > 1

{

*Lặp tìm điểm x lân cận của curPos thỏa bằng tim-1*

if (pathVisited(x) = tim-1)

{

*trace(tim-1) := x*

*curPos := x*

*tim := tim-1*

}

}

}

**Kết quả trả về :**

Độ dài đường đi : *tim*

Đường đi : *Mảng trace*

2.a.iii. Demo các thực hiện:

|  |  |  |
| --- | --- | --- |
| **1.** | **2.** | **3.** |
| **4.** | **5.** | **6.** |
| **7.** | **8.** | **9-10-11-12-13** |

Kết quả :



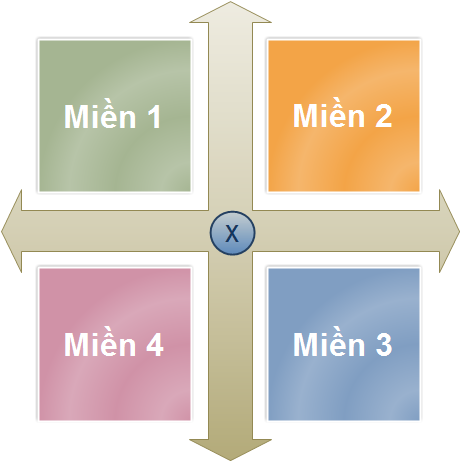
Minh họa một trường hợp có chướng ngại vật :



2.a.iv. Vấn đề hình học và hàm heuristic

Tại một điểm trên bàn cờ, ta chia xung quanh nó ra thành 4 miền.

Mục tiêu khi chia nhỏ các miền là nhằm dễ dàng chia trường hợp dễ xử lý ước lượng.



Các ước lượng khá đơn giản theo một lý luận như sau :

Điểm **hiện tại** là A, điểm đích là B

Lặp liên tục để chọn 1 điểm **hiện tại** mới

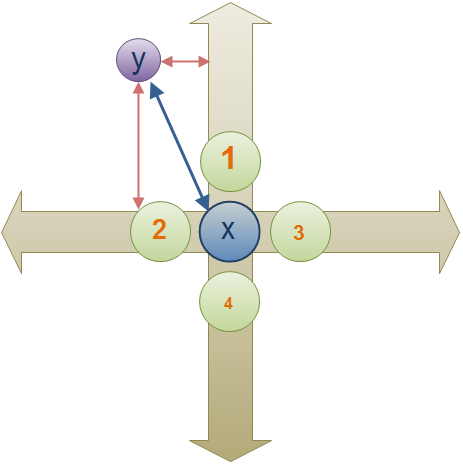
Ưu tiên chọn điểm C thỏa sao cho **C gần B hơn A**

Chúng ta xét ví dụ bên dưới

Cho **X,Y** là 2 điểm trên bàn cờ như trong hình.

d(X.x, Y.x) = max(X.x – Y.x, Y.x – X.x)

d(X.y, Y.y) = max(X.y – Y.y, Y.y – X.y)



Ở đây điểm **Y** nằm ở **miền 1** so với điểm **X**

Ta có :

d(X.y, Y.y) > d(X.x, Y.x)

Như vậy các điểm ưu tiên chọn trong bước đi kế tiếp sẽ theo thứ tự : **1, 2, 3, 4** như hình vẽ.

Hàm heuristic chính là việc ước lượng như bên trên.

2.a.v. Thực tiễn code

Ở trong code này, em có thực hiện một số thao tác như sử dụng mảng trung gian **arrUnit** nhằm tối ưu hóa về mặt số lượng dòng code, có thể hiểu là thay vì việc phải sắp xếp và tự cộng trừ thì chúng ta đảo các trị trong mảng này tương ứng sắp xếp lại mảng **unitStep** để có các thứ tự ưu tiên.

Theo cách em hiểu, ở đây **arrUnit** gần như là một *“mặt nạ”* để tùy trường hợp thì truy vấn đến mảng **unitStep** cho hợp lý.

unitStep(0).x = 0

unitStep(0).y = 1

unitStep(1).x = 1

unitStep(1).y = 0

unitStep(2).x = 0

unitStep(2).y = -1

unitStep(3).x = -1

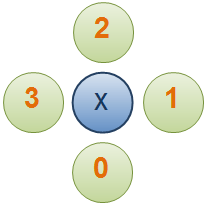
unitStep(3).y = 0

Với công thức tác động chung như sau với điểm X

X.x = X.x + unitStep(**i**).x

X.y = X.y + unitStep(**i**).y

Sau đây là các vị trí **i** tương ứng sau khi tác động :



Bằng cách tra cứu mảng **unitStep** như sau :

X.x = X.x + unitStep(**arrUnit(i)**).x

X.y = X.y + unitStep(**arrUnit(i)**).y

Như vậy việc thay đổi các giá trị của **arrUnit(i)** tương ứng với việc sắp xếp lại mảng **unitStep**.

Ngoài ra để dễ dàng tìm kiếm lại đường đi thì em có sử dụng thêm yếu tố đánh dấu bậc tương tự như loang.

Trong code sẽ có chia nhỏ rõ ràng hơn các trường hợp nhỏ cần xét và hình cho từng trường hợp.

**Code :** Các hàm xử lý Stack cơ bản

'---Stack---

Dim stack(99) As Long

Dim curPosStack As Long

Private Sub initStack()

curPosStack = -1

End Sub

Private Function isEmptyStack() As Boolean

If curPosStack = -1 Then

Return True

End If

Return False

End Function

Private Sub pushStack(ByVal lVal As Long)

curPosStack = curPosStack + 1

stack(curPosStack) = lVal

End Sub

Private Function popStack() As Long

If isEmptyStack() = False Then

Dim lRes As Long

lRes = stack(curPosStack)

curPosStack = curPosStack - 1

Return lRes

Else

Return -1

End If

End Function

'-----------

**Code :** Tìm đường đi ngắn nhất dựa vào bảng ma trận kết quả loang

Private Sub findPath(ByVal pFrom As Point, ByVal pTo As Point)

'Tìm kiếm sử dụng heuristic

Dim pNext As Point

Dim iCount As Integer

'Khởi tạo thêm các giá trị điểm kề của điểm bắt đầu

For iCount = 0 To 3

pNext.x = pFrom.x + unitStep(iCount).x

pNext.y = pFrom.y + unitStep(iCount).y

If (pNext.x >= 0) And (pNext.x <= 9) And (pNext.y >= 0) And (pNext.y <= 9) Then

If (iPixel(pNext.x, pNext.y) < 0) Then

With wWalk

lstGraph1.Items.Add(pos2index(pFrom))

lstGraph2.Items.Add(pos2index(pNext))

End With

End If

End If

Next iCount

'Bắt đầu tìm kiếm từ iFrom

Dim iFrom As Integer

'Điểm đích iTo

Dim iTo As Integer

iTo = pos2index(pTo)

Dim tim As Long

tim = 1

Dim pathVisited(99) As Integer

Dim i As Integer

For i = 0 To 99

pathVisited(i) = 0

Next

iFrom = pos2index(pFrom)

Dim isFounded As Boolean

isFounded = False

Dim iNext As Integer

'Khởi tạo stack

initStack()

'Điểm thêm điểm khởi đầu vào stack

pushStack(iFrom)

Dim curIndex As Integer

Dim curPos As Point

'Mảng lưu ưu tiên hệ số

Dim arrUnit(4) As Integer

pathVisited(iFrom) = 1

Do While isEmptyStack() = False

curIndex = popStack()

If curIndex = iTo Then

isFounded = True

Exit Do

End If

'---Hàm heuristic---

curPos = index2pos(curIndex)

'Tính toán vị trí pTo nằm trong phần gốc nào so với curPos

' |

'---O--------------> X

' | 1 | 2

' |-----curPos----

' | 4 | 3

' |

' Y

If (curPos.x >= pTo.x) And (curPos.y >= pTo.y) Then

'Góc 1

If (curPos.x - pTo.x) >= (curPos.y - pTo.y) Then

' |x |2

' |1----x----4

' | |3

arrUnit(4) = 1

arrUnit(3) = 0

arrUnit(2) = 2

arrUnit(1) = 3

Else

' | x|1

' |2----x----3

' | |4

arrUnit(4) = 0

arrUnit(3) = 1

arrUnit(2) = 3

arrUnit(1) = 2

End If

ElseIf (curPos.x < pTo.x) And (curPos.y > pTo.y) Then

'Góc 2

If (pTo.x - curPos.x) >= (curPos.y - pTo.y) Then

' | |2 x

' |4----x----1

' | |3

arrUnit(4) = 3

arrUnit(3) = 0

arrUnit(2) = 2

arrUnit(1) = 1

Else

' | |1x

' |3----x----2

' | |4

arrUnit(4) = 0

arrUnit(3) = 3

arrUnit(2) = 1

arrUnit(1) = 2

End If

ElseIf (curPos.x <= pTo.x) And (curPos.y <= pTo.y) Then

'Góc 3

If (pTo.x - curPos.x) >= (pTo.y - curPos.y) Then

' | |3

' |4----x----1

' | |2 x

arrUnit(4) = 3

arrUnit(3) = 2

arrUnit(2) = 0

arrUnit(1) = 1

Else

' | |4

' |3----x----2

' | |1x

arrUnit(4) = 2

arrUnit(3) = 3

arrUnit(2) = 1

arrUnit(1) = 0

End If

ElseIf (curPos.x > pTo.x) And (curPos.y < pTo.y) Then

'Góc 4

If (curPos.x - pTo.x) >= (pTo.y - curPos.y) Then

' | |3

' |1----x----4

' |x |2

arrUnit(4) = 1

arrUnit(3) = 2

arrUnit(2) = 0

arrUnit(1) = 3

Else

' | |4

' |2----x----3

' | x|1

arrUnit(4) = 2

arrUnit(3) = 1

arrUnit(2) = 3

arrUnit(1) = 0

End If

End If

For iCount = 4 To 1 Step -1

pNext.x = curPos.x + unitStep(arrUnit(iCount)).x

pNext.y = curPos.y + unitStep(arrUnit(iCount)).y

If isHaveCanh(curIndex, pos2index(pNext)) = True Then

If pathVisited(pos2index(pNext)) = 0 Then

pushStack(pos2index(pNext))

pathVisited(pos2index(pNext)) = pathVisited(pos2index(curPos)) + 1

End If

End If

Next

'---Hết hàm Heuristic---

Loop

tim = pathVisited(pos2index(pTo))

If isFounded = True Then

'Nếu tìm được đường đi

trace.iLen = tim

ReDim trace.pos(trace.iLen)

curIndex = iTo

Do While pathVisited(curIndex) > 1

For iCount = 0 To lstGraph1.Items.Count - 1

If (lstGraph1.Items(iCount) = curIndex) Or (lstGraph2.Items(iCount) = curIndex) Then

'Tìm điểm kề điểm hiện tại

If lstGraph1.Items(iCount) = curIndex Then

iNext = Val(lstGraph2.Items(iCount))

Else

iNext = Val(lstGraph1.Items(iCount))

End If

If pathVisited(iNext) = tim - 1 Then

trace.pos(tim - 1) = index2pos(iNext)

curIndex = iNext

tim = tim - 1

Exit For

End If

End If

Next iCount

Loop

'Tiến hành vẽ cờ di chuyển

Dim iIndex As Integer

For iCount = 1 To trace.iLen - 1

iIndex = pos2index(trace.pos(iCount))

pPixel(iIndex).Image = imgGo.Images.Item(0)

Next

Else

trace.iLen = 0

Dim iX As Integer

Dim iTmp As Integer

'Xóa cạnh trong list

iTmp = pos2index(pFrom)

For iX = lstGraph1.Items.Count - 1 To 0 Step -1

If (lstGraph1.Items(iX) = iTmp) Or (lstGraph2.Items(iX) = iTmp) Then

lstGraph1.Items.RemoveAt(iX)

lstGraph2.Items.RemoveAt(iX)

End If

Next

End If

End Sub