BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO DỰ ÁN**

**Học phần: Trí tuệ nhân tạo**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn:** | **Nguyễn Đình Cường** |
| **Sinh viên thực hiện:** | **Nguyễn Sanh Quốc Huy** |
| **Lớp:** | **62 CNTT-3** |
| **MSSV:** | **62130757** |

**Khánh Hòa – Tháng 7/2023**

**LỜI MỞ ĐẦU**

Trong thế giới đang phát triển với sự hội nhập và cạnh tranh không ngừng, cuộc cách mạng khoa học và kỹ thuật đang đưa nhân loại vào thời kỳ đầy hứa hẹn. Công nghệ thông tin là một lĩnh vực đang trải qua tiến bộ đáng kinh ngạc, mang đến những ứng dụng đột phá như máy tính, điện thoại thông minh và robot. Từ kinh doanh, giáo dục đến y tế, mọi lĩnh vực đều được thúc đẩy bởi sự tiến bộ trong Công nghệ Thông tin.Trí tuệ nhân tạo đóng vai trò trung tâm trong sự phát triển này và đang trở thành một ngành khoa học nóng bỏng trong những năm gần đây. Thuật ngữ trí tuệ nhân tạo (Aritifical Intellegence) này được đưa ra lần đầu bởi John McCarthy vào năm 1956, và đối với nhiều người, trí tuệ nhân tạo chỉ là khả năng của máy móc hoặc máy tính. Nhưng thực tế, nó đã tiến bộ đáng kể, có khả năng "nhận thức" và "học tập" giống như con người.

Bài báo cáo này tập trung chia sẻ kiến thức về trí tuệ nhân tạo, từ quá trình học tập, nghiên cứu, đến các lý thuyết và thuật toán cơ bản, cũng như ứng dụng thực tiễn.

**MỤC LỤC**

[Bài 1: Trò chơi Tic Tac Toe 4](#_Toc142562675)

[1. Giới thiệu trò chơi 4](#_Toc142562676)

[2. Thuật toán Minimax 4](#_Toc142562677)

[3. Cài đặt chương trình 6](#_Toc142562678)

[Bài 2: Bài toán 8-Puzzle, 15-Puzzle 9](#_Toc142562679)

[1. Giới thiệu bài toán 8-puzzle, 15-puzzle 9](#_Toc142562680)

[2. Thuật toán A\* 9](#_Toc142562681)

[3. Cài đặt chương trình 13](#_Toc142562682)

[Bài 3: Bài toán TSP sử dụng thuật toán di truyền GA 24](#_Toc142562683)

[1. Giới thiệu bài toán TSP (Travelling Salesman Problem) 24](#_Toc142562684)

[2. Cơ sở lý thuyết 25](#_Toc142562685)

[3. Cài đặt chương trình 28](#_Toc142562686)

[Bài 4: Cài đặt thuật giải Quinlan trong máy học 36](#_Toc142562687)

[1. Lý thuyết 36](#_Toc142562688)

[2. Thuật toán 36](#_Toc142562689)

[3. Cài đặt chương trình 39](#_Toc142562690)

[Bài 5: Tìm đường đi ngắn nhất trong mê cung bằng thuật toán A\* 46](#_Toc142562691)

[1. Giới thiệu 46](#_Toc142562692)

[2. Thuật toán A\* (đã trình bày ở bài 2) 47](#_Toc142562693)

[3. Cài đặt chương trình 47](#_Toc142562694)

[Bài 6: Bài toán người nông dân qua sông. 55](#_Toc142562695)

[1. Giới thiệu 55](#_Toc142562696)

[2. Thuật toán tìm kiếm theo chiều rộng (BFS) 56](#_Toc142562697)

[3. Cài đặt chương trình 58](#_Toc142562698)

[KẾT LUẬN 63](#_Toc142562699)

[1. Kết quả đạt được 63](#_Toc142562700)

[2. Đề xuất 63](#_Toc142562701)

# Bài 1: Trò chơi Tic Tac Toe

## Giới thiệu trò chơi

* 1. Trò chơi Tic Tac Toe

Tic Tac Toe là một trò chơi trên bàn cờ đơn giản, giúp chúng ta giải trí sau những giờ học căng thăng. Trò chơi được chơi trên một bảng gồm 9 ô vuông, được chia thành 3 hàng và 3 cột.

Hai người chơi sẽ lần lượt đánh dấu vào những ô trống bằng ký hiệu của mình, một người đánh "X" và người còn lại đánh "O", với mục đích đưa ba ký hiệu của mình liên tiếp theo hàng ngang, hàng dọc hoặc chéo để thắng cuộc. Nếu không có người chơi nào đạt được điều kiện thắng sau khi tất cả các ô đã được đánh đầy, trò chơi sẽ kết thúc với kết quả là hòa. Tic Tac Toe là trò chơi giải trí phổ biến và thường được sử dụng để giúp trẻ em phát triển tư duy và kỹ năng tương tác xã hội.

Tính thân thiện của trò chơi khiến trò chơi này đã trở thành một trò chơi dạy học trong các trường học để dạy về tinh thần thể thao và trí tuệ. Người Việt thường chơi trò tương tự, gọi là Cờ ca-rô, bàn cờ không giới hạn trong 9 ô, có thể vẽ thêm ô, để kéo rộng ra cho đến khi người nào đạt được một dãy 5 thì thắng cuộc. Bạn hãy thoả thích chơi game cùng với máy tính hoặc bạn bè, và cùng trổ tài của mình để có thể giành chiến thắng trong game vui này.

* 1. Yêu cầu chức năng
* Giao diện trò chơi đơn giản, dễ dàng thao tác với người sử dụng.
* Các chức năng được thực hiện đúng, chính xác.
  1. Phân tích thiết kế trò chơi

Trong trò chơi này chỉ có chức năng đấu với máy (không có đấu với người). Người chơi và máy sẽ thay phiên nhau đánh vào từng ô, người đánh X, máy đánh O cho tới khi nào 1 trong 2 hoàn thành được 3 ô liên tiếp (theo hàng dọc, ngang hay chéo) giống nhau thì sẽ thắng. Trong bài này chúng em sẽ trình bày cách làm một trò chơi như thế này.

## Thuật toán Minimax

Minimax là một thuật toán đệ quy được sử dụng trong lý thuyết trò chơi để tối ưu hóa quyết định giúp người chơi tối đa hóa lợi ích của mình và tối thiểu hóa lợi ích của đối thủ.. Trong trò chơi đối kháng như tic-tac-toe, cờ vua, cờ tướng, cờ caro, người chơi MAX và MIN thay phiên đi nước đi của mình.

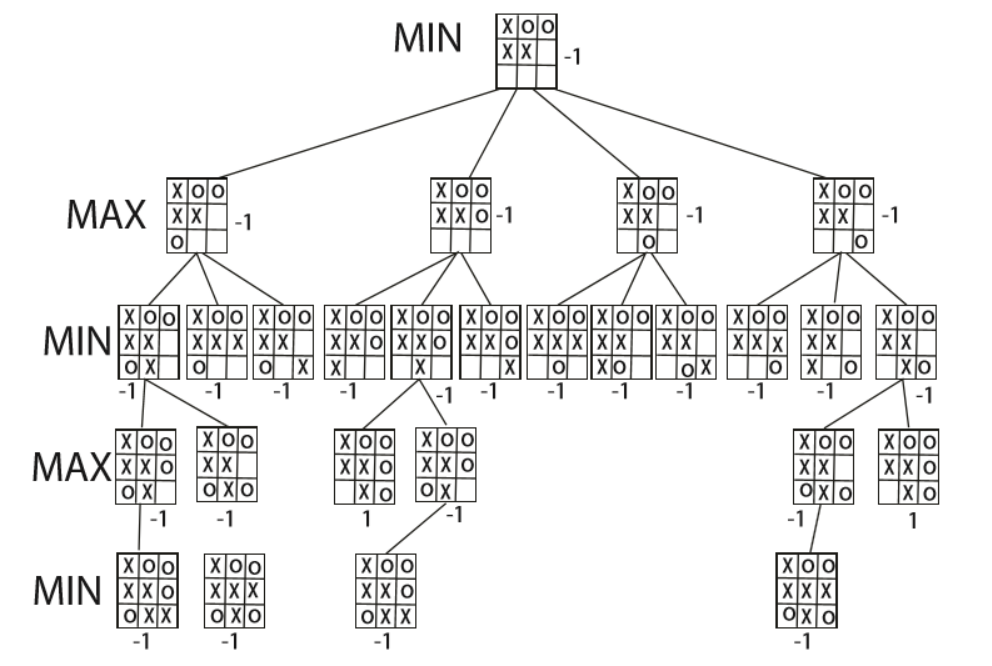
Mục tiêu: MAX cố gắng tối đa hóa lợi ích của mình (thắng lợi), trong khi MIN cố gắng tối thiểu hóa lợi ích của MAX và cố gắng làm cho điểm số của mình càng âm càng tốt (thắng MIN).

Cây trò chơi: Minimax xây dựng một cây trò chơi, trong đó mỗi Node biểu diễn một trạng thái của trò chơi và các cạnh nối các Node biểu thị các bước di chuyển có thể được thực hiện trong trò chơi. Các Node chia thành hai loại: MAX Node và MIN Node.

Đánh giá giá trị Node: Thuật toán Minimax đánh giá giá trị của các Node trong cây bằng cách đệ quy từ Node gốc xuống các Node lá và quay lại Node gốc. Khi đánh giá, MAX Node chọn giá trị lớn nhất của các Node con, trong khi MIN Node chọn giá trị nhỏ nhất của các Node con.

Quyết định tối ưu: Sau khi tính toán giá trị cho toàn bộ cây trò chơi, giá trị của Node gốc sẽ là giá trị tối ưu cho lượt của người chơi hiện tại. Người chơi sẽ chọn bước đi dẫn đến Node con có giá trị tối ưu như vậy.

Tối ưu hóa và cắt tỉa: Do cây trò chơi có thể rất lớn, việc xây dựng toàn bộ cây và đánh giá giá trị của mỗi Node có thể tốn rất nhiều thời gian và bộ nhớ. Vì vậy, Minimax thường được kết hợp với phương pháp cắt tỉa (pruning) như Alpha-Beta Pruning để giảm số lượng Node cần tính toán, làm cho thuật toán hiệu quả hơn.

****

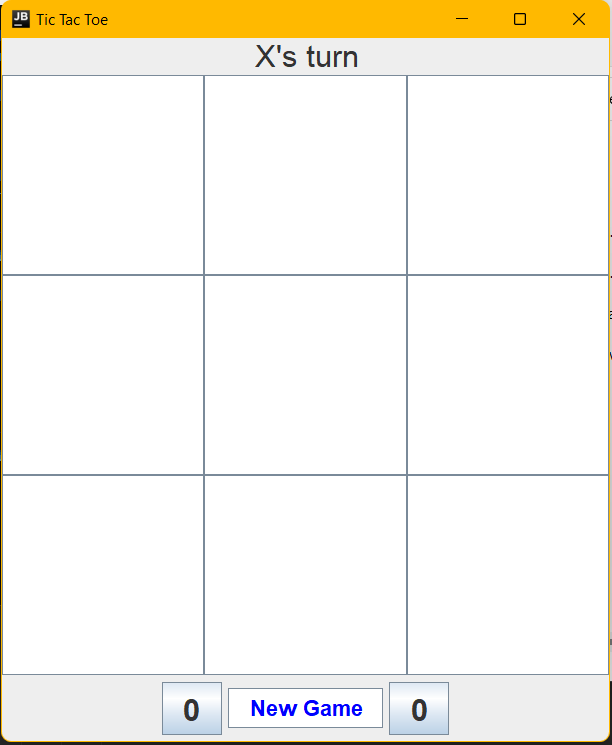
## Cài đặt chương trình

Sử dụng ngôn ngữ lập trình java kết hợp với thư viện đồ họa swing.

Phương thức minimax:

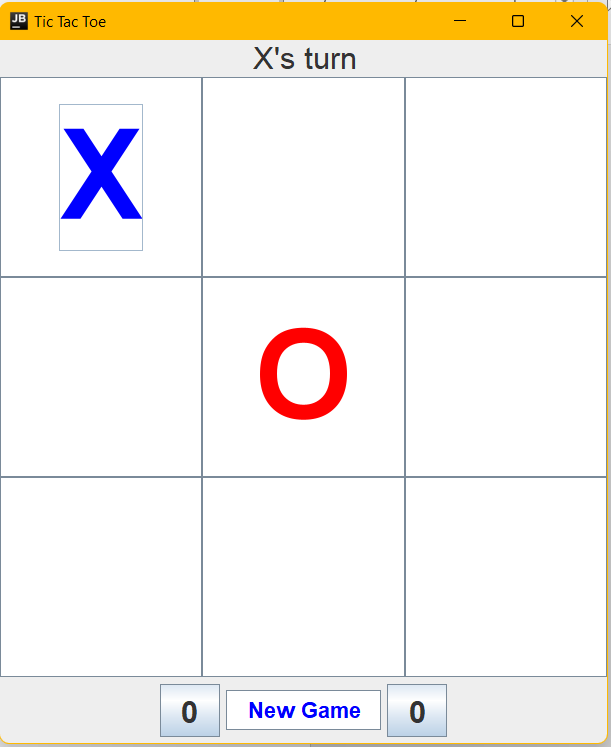
|  |
| --- |
| private int **minimax**(String[][] board, int depth, boolean isMaximizing) {  if (checkWin("X")) {  return -1; // Nếu X thắng trả về -1  } else if (checkWin("O")) {  return 1; // Nếu O thắng trả về 1  } else if (checkDraw()) {  return 0; // Nếu hòa trả về 0  }  // Nếu đang là lượt của người chơi muốn tối ưu hóa (Maximizer)  if (isMaximizing) {  int bestScore = Integer.MIN\_VALUE;  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  if (board[i][j].isEmpty()) {  board[i][j] = "O";  // Gọi đệ quy tới độ sâu tiếp theo và đánh giá nước đi của Minimizer (đã chuyển lượt)  int score = minimax(board, depth + 1, false);  // Hoàn tác nước đi tạm thời  board[i][j] = "";  // Lấy giá trị lớn nhất giữa bestScore và score  bestScore = Math.max(score, bestScore);  }  }  }  // Trả về điểm số tốt nhất cho Maximizer  return bestScore;  } else {  int bestScore = Integer.MAX\_VALUE;  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  if (board[i][j].isEmpty()) {  board[i][j] = "X";  // Gọi đệ quy tới độ sâu tiếp theo và đánh giá nước đi của Maximizer (đã chuyển lượt)  int score = minimax(board, depth + 1, true);  board[i][j] = "";  // Lấy giá trị nhỏ nhất giữa bestScore và score  bestScore = Math.min(score, bestScore);  }  }  }  // Trả về điểm số tốt nhất cho Minimizer  return bestScore;  }  } |

Giao diện trò chơi:



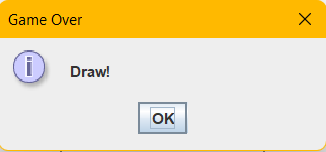
Hình 1. Giao diện khi chạy chương trình

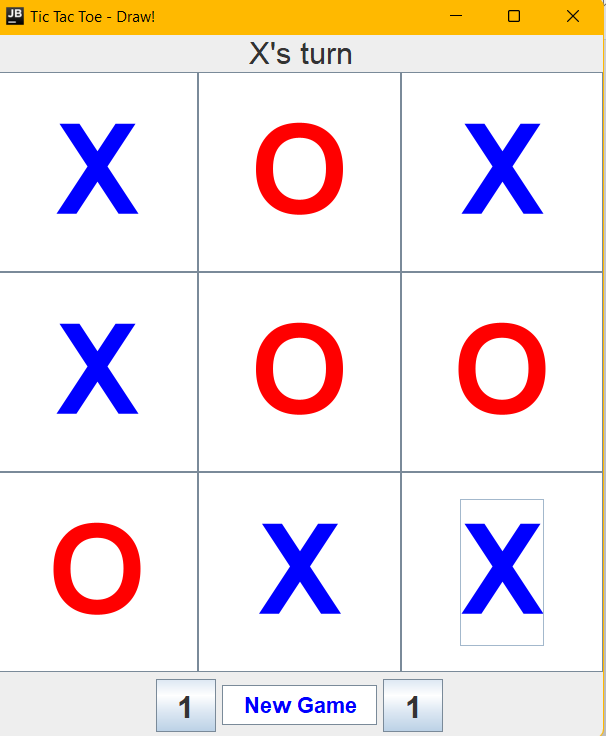
Trò chơi sẽ chọn ngẫu nhiên người hay máy đi trước, người đánh X, máy đánh O  
Ở trên hiển thị lượt của người chơi, ở dưới hiển thị điểm số của người chơi và máy được cộng dồn sau mỗi ván chơi. Một nút “New Game” để khởi động lại trò chơi. Người chơi chỉ cần click chuột trái vào ô muốn đánh, sau đó máy tính sẽ tự động đánh với người chơi.



Hình 2. Người chơi và máy lần lượt đánh với nhau

Khi kết thúc trò chơi sẽ hiển thị hộp thoại thông báo trạng thái của trò chơi: người chơi nào thắng hay hòa và tự động cập nhật điểm số. Điểm số được tính như sau: +10 cho người chơi nào thắng, +1 nếu cả hai kết thúc hòa.





Hình 3. Kết thúc trò chơi với kết quả hòa

# Bài 2: Bài toán 8-Puzzle, 15-Puzzle

## Giới thiệu bài toán 8-puzzle, 15-puzzle

Bài toán 8-puzzle (hay còn gọi là 8 số) là một bài toán quen thuộc với những người bắt đầu tiếp cận với môn Trí tuệ nhân tạo. Bài toán có nhiều phiên bản khác nhau dựa theo số ô, như 8-puzzle, 15-puzzle... Bài toán 8-puzzle gồm một bảng ô vuông kích thước 3x3, có 8 ô được đánh số từ 1 tới 8 và một ô trống. Tương tự như vậy với bài toán 15-puzzle gồm 1 bảng ô vuông có kích thước 4x4, có 15 ô được đánh số từ 1 đến 15 và 1 ô trống. Trạng thái ban đầu, các ô được sắp xếp một cách ngẫu nhiên, nhiệm vụ của người chơi là tìm cách đưa chúng về đúng thứ tự. Trong quá trình giải bài toán, tại mỗi bước, ta giả định chỉ có ô trống là di chuyển, như vậy, tối đa ô trống có thể có 4 khả năng di chuyển (lên trên, xuống dưới, sang trái, sang phải).

## Thuật toán A\*

* 1. Giới thiệu

Thuật toán A\* là một thuật toán tìm đường thông qua các trạng thái từ trạng thái ban đầu (n0) đến một trạng thái đích (n). 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.

* 1. Mô tả thuật toán

Để xác định chi phí dự kiến để đi từ một trạng thái hiện tại n tới trạng thái đích, ta sử dụng hàm đánh giá f(n) = g(n) + h(n).

Trong đó:

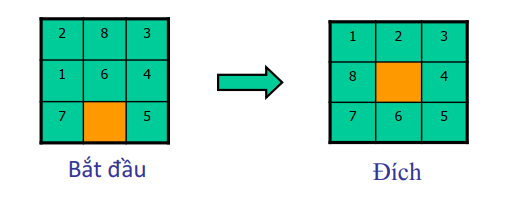
g(n) là chi phí từ trạng thái ban đầu n0 tới trạng thái hiện tại n. Đây là chi phí thực tế đã đi qua để đạt được trạng thái hiện tại.

h(n) là một ước lượng chi phí từ trạng thái hiện tại n tới trạng thái đích. Đây là một hàm heuristic (ước lượng) được sử dụng để xác định khoảng cách dự kiến từ n tới đích. Hàm heuristic này phải thoả mãn tính chất: 0 ≤ h(n) ≤ h\*(n), trong đó h\*(n) là chi phí thực tế để đi từ n tới trạng thái đích.

Hàm heuristic h(n) được sử dụng để định hướng thuật toán A\* theo một hướng ưu tiên dựa trên ước lượng khoảng cách tới đích. Nhờ vậy, A\* có xu hướng tìm các đường đi tiềm năng ngắn nhất đầu tiên, giúp giảm số lượng trạng thái phải duyệt và làm cho thuật toán hiệu quả hơn so với việc duyệt theo thứ tự tất cả các trạng thái.

Như vậy, thuật toán A\* kết hợp cả thông tin thực tế (g(n)) và thông tin heuristic (h(n)) để điều hướng tìm kiếm một cách tối ưu từ trạng thái ban đầu tới trạng thái đích, và đảm bảo sẽ tìm ra đường đi ngắn nhất nếu có tồn tại một đường đi từ trạng thái ban đầu đến trạng thái đích.

Xét bài toán 8-puzzle với hàm heuristic:



Heuristic 1: Tổng số miếng sai vị trí.

Heuristic 2: Tổng khoảng cách sai vị trí của từng miếng.

Việc chọn lựa hàm Heuristic là khó khăn và có ý nghĩa quyết định đối với tốc độ của giải thuật.

Xét ví dụ là bài toán 8 puzzle với:

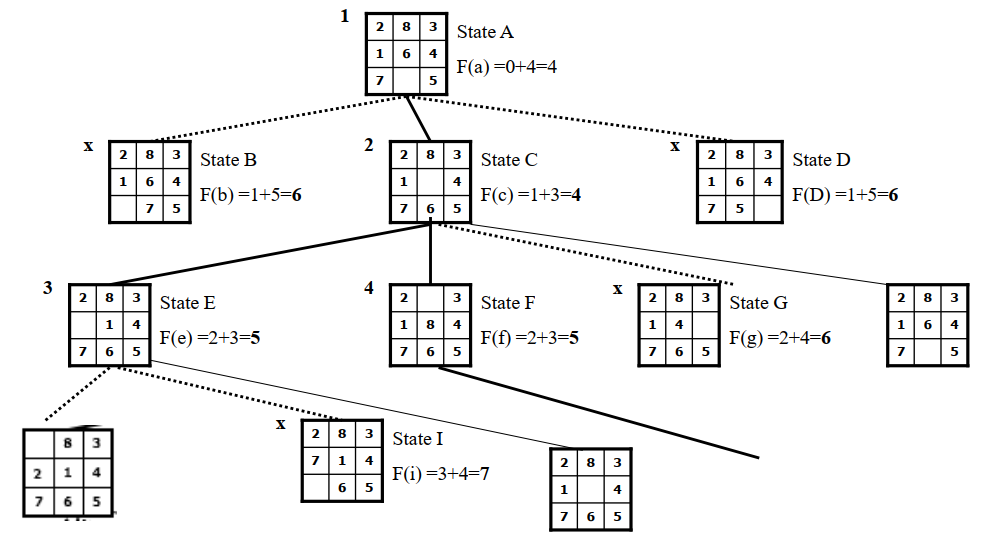
Hàm lượng giá: F(n) = G(n) + H(n).

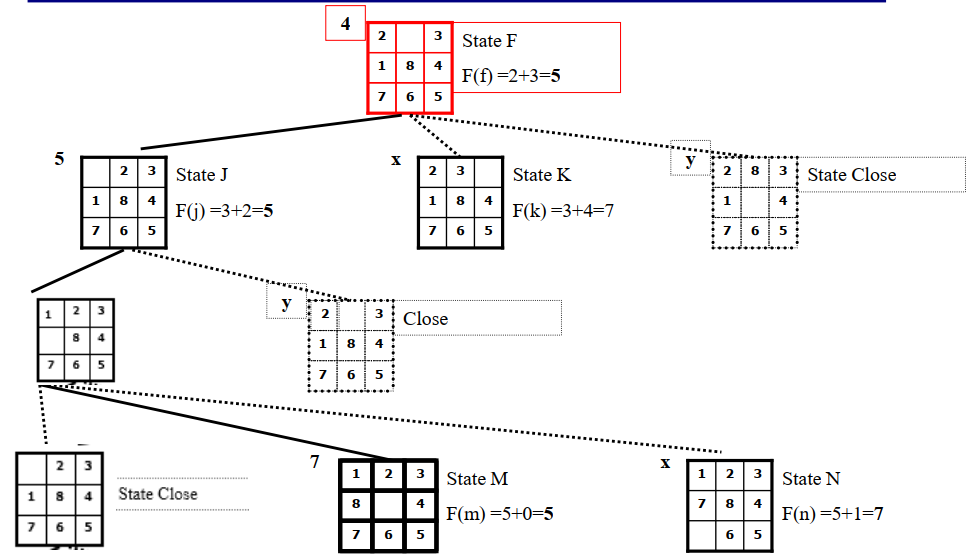
Với G(n): số lần chuyển vị trí đã thực hiện.

H(n): Số miếng nằm sai vị trí.

Nút X có giá trị heuristic tốt hơn nút Y nếu F(x) < F(y).

Ta có hoạt động của giải thuật Best First search trên như hình sau:





* 1. Mô phỏng thuật toán

Ban đầu ta có Open là tập chứa các trạng thái chưa được xét (sắp xếp theo f tăng dần), Close là tập các trạng thái đã được xét. Ban đầu Open chỉ chứa trạng thái ban đầu, tập Close rỗng.

Begin

Open:={Start}; Close:=∅;

While (Open <> ∅) do

Begin

X=Retrieve(Open); {Chọn X sao cho f(X) đạt là nhỏ nhất}

If (X=Goal) then return True

Else

Begin

Sinh ra các trạng thái con của X;

For mỗi nút con Y của X do

If (Y ∉ Open) và (Y ∉ Close)

Begin

Tính f(Y);

Open = Open ∪ {Y};

End;

If (Y ∈ Open)

If (g(Y) < g(Y’)) cập nhật lại giá trị f(Y’), đặt cha của Y’ là X;

If (Y ∈ Close)

If (g(Y) < g(Y’)) cập nhật lại giá trị f(Y’) , đặt cha của Y’ là X, cập nhật lại giá trị f và g của tất cả các con của Y đã có trong Open và Close.

End;

Close = Close ∪ {X};

End;

Return False;

End.

## Cài đặt chương trình

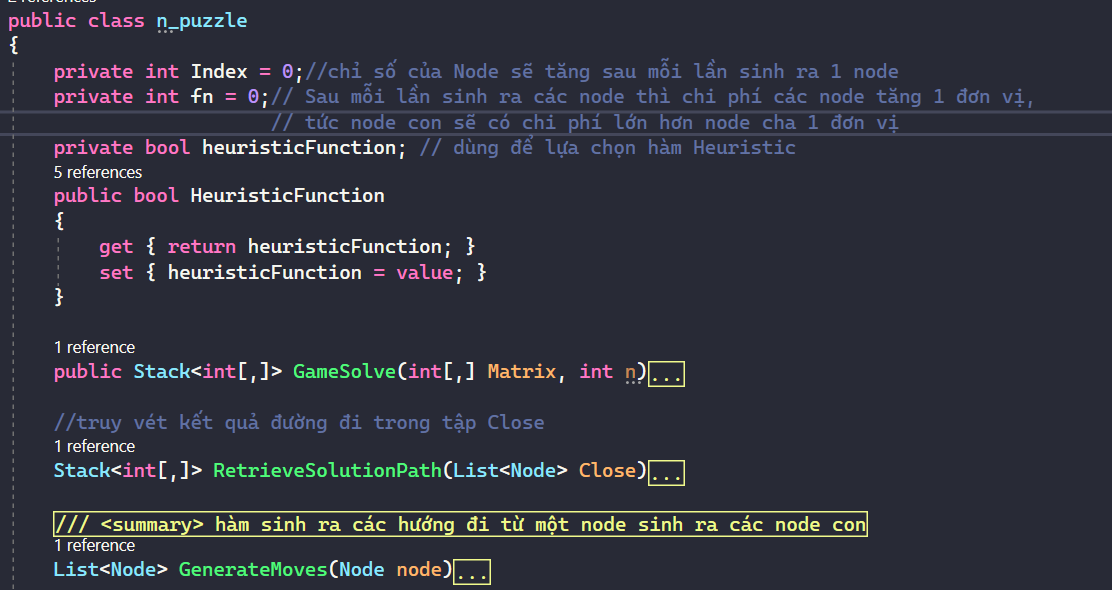
Sử dụng ngôn ngữ lập trình C# kết hợp đồ họa Window form trên Visual Studio

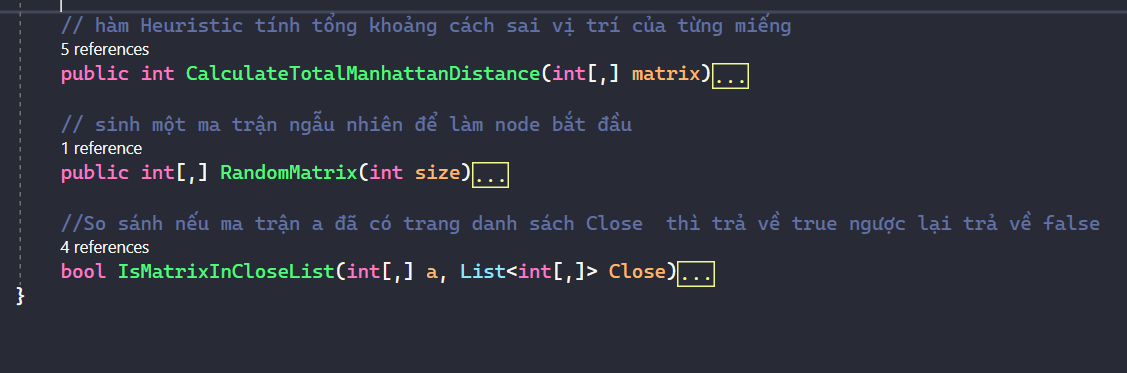
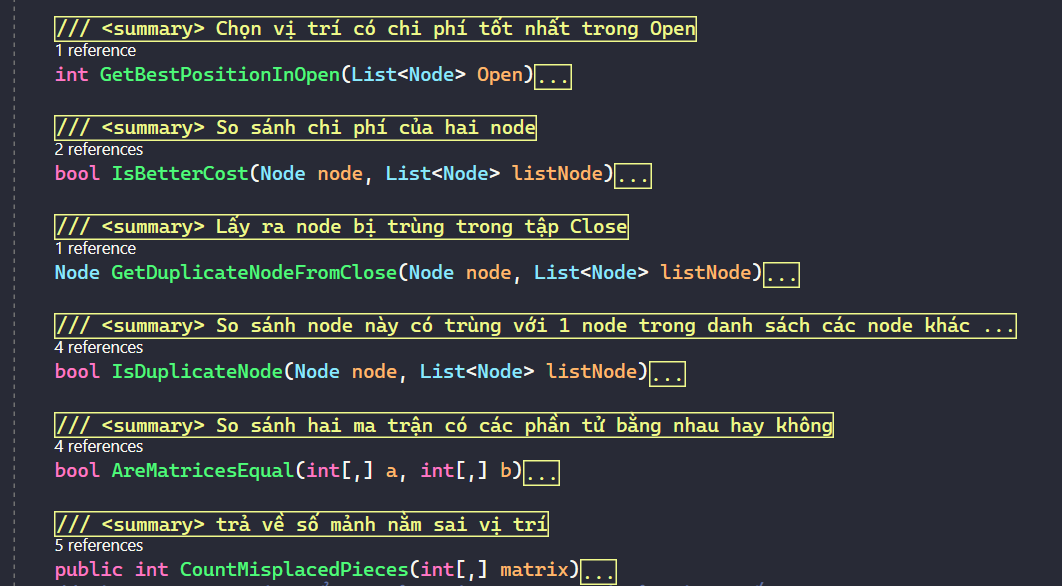
* Đầu vào: trạng thái hiện tại, trạng thái đích
* Đầu ra: tập các trạng thái từ trạng thái hiện tại đến trạng thái đích

Class Node:

|  |
| --- |
| public class Node  {  public int[,] puzzleMatrix; // mảng 2 chiều lưu trữ ma trận  public int heuristicCost; // //ước lượng chi phí đến đích  public int index; // chỉ số của node  public int parent; // cha của node, dùng để truy vết kết quả  public int fn; // chi phí đi đến node đó    } |

Class n\_puzzle:





Phương thức GameSolve:

|  |
| --- |
| public Stack<int[,]> **GameSolve**(int[,] Matrix, int n)  {  Stack<int[,]> stkResult = new Stack<int[,]>();  List<Node> Close = new List<Node>();  List<Node> Open = new List<Node>();  //khai báo và khởi tạo cho node đầu tiên  Node Node = new Node();  Node.puzzleMatrix = Matrix;    Node.heuristicCost = heuristicFunction==true ?  CountMisplacedPieces(Matrix) : CalculateTotalManhattanDistance(Matrix);  Node.index = 0;  Node.parent = -1;  Node.fn = 0;  //cho trạng thái đầu tiên vào Open;  Open.Add(Node);  int t = 0;  while (Open.Count != 0)  {  #region chọn node tốt nhất trong tập Open và chuyển nó sang Close  Node = new Node();  Node = Open[t];  Open.Remove(Node);  Close.Add(Node);  #endregion  //nếu node có chi phí ước lượng là 0, tức là đích thì thoát  if (Node.heuristicCost == 0) break;  else  {  //sinh hướng đi của node hiện tại  List<Node> lstHuongDi = new List<Node>();  lstHuongDi = GenerateMoves(Node);  for (int i = 0; i < lstHuongDi.Count; i++)  {  //hướng đi không thuộc Open và Close  if (!IsDuplicateNode(lstHuongDi[i], Open) &&  !IsDuplicateNode(lstHuongDi[i], Close))  {  Open.Add(lstHuongDi[i]);  }  else  { //nếu hướng đi thuộc Open  if (IsDuplicateNode(lstHuongDi[i], Open))  {  /\*nếu hướng đi đó tốt hơn thì sẽ được cập nhật lại,  ngược lại thì sẽ không cập nhật\*/  IsBetterCost(lstHuongDi[i], Open);  }  else  {  //nếu hướng đi thuộc Close  if (IsDuplicateNode(lstHuongDi[i], Close))  {  /\*nếu hướng đi đó tốt hơn thì sẽ được cập nhật lại,  ngược lại thì sẽ không cập nhật và chuyển từ  Close sang Open\*/  if (IsBetterCost(lstHuongDi[i], Close))  {  Node temp = new Node();  temp =  GetDuplicateNodeFromClose(lstHuongDi[i], Close);  Close.Remove(temp);  Open.Add(temp);  }  }  }  }  }  //chọn vị trí có phí tốt nhất trong Open  t = GetBestPositionInOpen(Open);  }  }  //truy vết kết quả trong tập Close  stkResult = RetrieveSolutionPath(Close);  return stkResult;  } |

Phương thức GenerateMoves:

|  |
| --- |
| List<Node> **GenerateMoves**(Node node)  {  int n = node.puzzleMatrix.GetLength(0);//lấy số hàng của ma trận  List<Node> listMoves = new List<Node>();  #region Xác định vị trí mảnh trống, có giá trị là 0  int h = 0;  int c = 0;  bool ok = false;  for (h = 0; h < n; h++)  {  for (c = 0; c < n; c++)  if (node.puzzleMatrix[h, c] == 0)  {  ok = true;  break;  }  if (ok) break;  }  #endregion  Node Temp = new Node();  Temp.puzzleMatrix = new int[n, n];  //Copy mảng Ma trận sang mảng ma trận tạm  Array.Copy(node.puzzleMatrix, Temp.puzzleMatrix,  node.puzzleMatrix.Length);  fn++;// tăng chi phí của node con lên 1 đơn vị  #region Xét các hướng đi theo 4 hướng: trên, dưới, phải, trái  //xét hàng ngang bắt đầu từ hàng thứ 2 trở đi  if (h > 0 && h <= n - 1)  {  // thay đổi hướng đi của ma trận  Temp.puzzleMatrix[h, c] = Temp.puzzleMatrix[h - 1, c];  Temp.puzzleMatrix[h - 1, c] = 0;  //cập nhật lại thông số của node  Temp.heuristicCost = heuristicFunction==true ?  CountMisplacedPieces(Temp.puzzleMatrix) :  CalculateTotalManhattanDistance(Temp.puzzleMatrix);  Index++;  Temp.index = Index;  Temp.parent = node.index;  Temp.fn = fn + Temp.heuristicCost;  listMoves.Add(Temp);  //sau khi thay đổi ma trận thì copy lại ma trận cha cho MaTran để  xét trường hợp tiếp theo  Temp = new Node();  Temp.puzzleMatrix = new int[n, n];  Array.Copy(node.puzzleMatrix, Temp.puzzleMatrix,  node.puzzleMatrix.Length);  }  //xét hàng ngang bắt đầu từ hàng thứ cuối cùng - 1 trở xuống  if (h < n - 1 && h >= 0)  {  // thay đổi hướng đi của ma trận  Temp.puzzleMatrix[h, c] = Temp.puzzleMatrix[h + 1, c];  Temp.puzzleMatrix[h + 1, c] = 0;  //cập nhật lại thông số của node  Temp.heuristicCost = heuristicFunction == true ?  CountMisplacedPieces(Temp.puzzleMatrix) :  CalculateTotalManhattanDistance(Temp.puzzleMatrix);  Index++;  Temp.index = Index;  Temp.parent = node.index;  Temp.fn = fn + Temp.heuristicCost;  listMoves.Add(Temp);  //sau khi thay đổi ma trận thì copy lại ma trận cha cho MaTran để  xét trường hợp tiếp theo  Temp = new Node();  Temp.puzzleMatrix = new int[n, n];  Array.Copy(node.puzzleMatrix, Temp.puzzleMatrix,  node.puzzleMatrix.Length);  }  //Xét cột dọc bắt đầu từ cột thứ 2 trở đi  if (c > 0 && c <= n - 1)  {  // thay đổi hướng đi của ma trận  Temp.puzzleMatrix[h, c] = Temp.puzzleMatrix[h, c - 1];  Temp.puzzleMatrix[h, c - 1] = 0;  //cập nhật lại thông số của node  Temp.heuristicCost = heuristicFunction == true ?  CountMisplacedPieces(Temp.puzzleMatrix) :  CalculateTotalManhattanDistance(Temp.puzzleMatrix);  Index++;  Temp.index = Index;  Temp.parent = node.index;  Temp.fn = fn + Temp.heuristicCost;  listMoves.Add(Temp);  //sau khi thay đổi ma trận thì copy lại ma trận cha cho MaTran để  xét trường hợp tiếp theo  Temp = new Node();  Temp.puzzleMatrix = new int[n, n];  Array.Copy(node.puzzleMatrix, Temp.puzzleMatrix,  node.puzzleMatrix.Length);  }  //Xét cột dọc bắt đầu từ cột cuối cùng -1 trở xuống  if (c < n - 1 && c >= 0)  {  // thay đổi hướng đi của ma trận  Temp.puzzleMatrix[h, c] = Temp.puzzleMatrix[h, c + 1];  Temp.puzzleMatrix[h, c + 1] = 0;  //cập nhật lại thông số của node  Temp.heuristicCost = heuristicFunction == true ?  CountMisplacedPieces(Temp.puzzleMatrix) :  CalculateTotalManhattanDistance(Temp.puzzleMatrix);  Index++;  Temp.index = Index;  Temp.parent = node.index;  Temp.fn = fn + Temp.heuristicCost;  listMoves.Add(Temp);  //đến đây đã xét hết hướng đi nên không cần copy lại ma trận  }  #endregion  return listMoves;  } |

Phương thức RetrieveSolutionPath:

|  |
| --- |
| Stack<int[,]> **RetrieveSolutionPath**(List<Node> Close)  {  Stack<int[,]> solutionPath = new Stack<int[,]>();  int t = Close[Close.Count - 1].parent;  Node temp = new Node();  solutionPath.Push(Close[Close.Count - 1].puzzleMatrix);  while (t != -1)  {  for (int i = 0; i < Close.Count; i++)  {  if (t == Close[i].index)  {  temp = Close[i];  break;  }  }  solutionPath.Push(temp.puzzleMatrix);  t = temp.parent;  }  return solutionPath;  } |

Phương thức CountMisplacedPieces:

|  |
| --- |
| public int **CountMisplacedPieces**(int[,] matrix)  {  int misplacedPieces = 0;  int[,] targetMatrix = new int[,]  {  {1,2,3},  {4,5,6},  {7,8,0},  };  int n = matrix.GetLength(0);  if (n == 4)  {  targetMatrix = new int[,]  {  {1,2,3,4},  {5,6,7,8},  {9,10,11,12},  {13,14,15,0},  };  }  for (int i = 0; i < n; i++)  {  for (int j = 0; j < n; j++)  {  if (targetMatrix[i, j] != matrix[i, j] && matrix[i, j] != 0)  {  misplacedPieces++;  }  }  }  return misplacedPieces;  } |

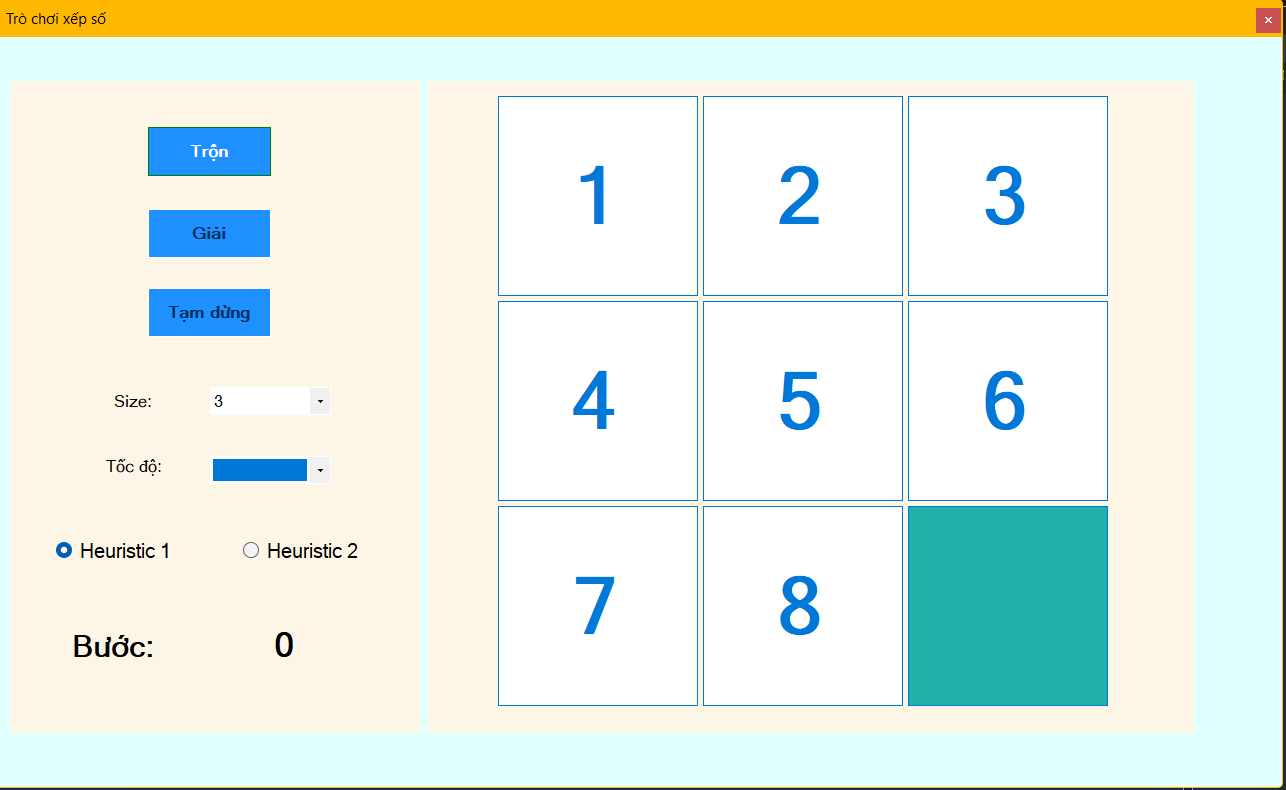
Phương thức CalculateTotalManhattanDistance:

|  |
| --- |
| public int **CalculateTotalManhattanDistance**(int[,] matrix)  {  int n = matrix.GetLength(0);  int totalDistance = 0;  for (int i = 0; i < n; i++)  {  for (int j = 0; j < n; j++)  {  int value = matrix[i, j];  if (value != 0)  {  // Tính toán vị trí đích của miếng  int targetRow = (value - 1) / n;  int targetCol = (value - 1) % n;  // Tính khoảng cách sai vị trí  totalDistance += Math.Abs(i - targetRow) + Math.Abs(j –  targetCol);  }  }  }  return totalDistance;  } |

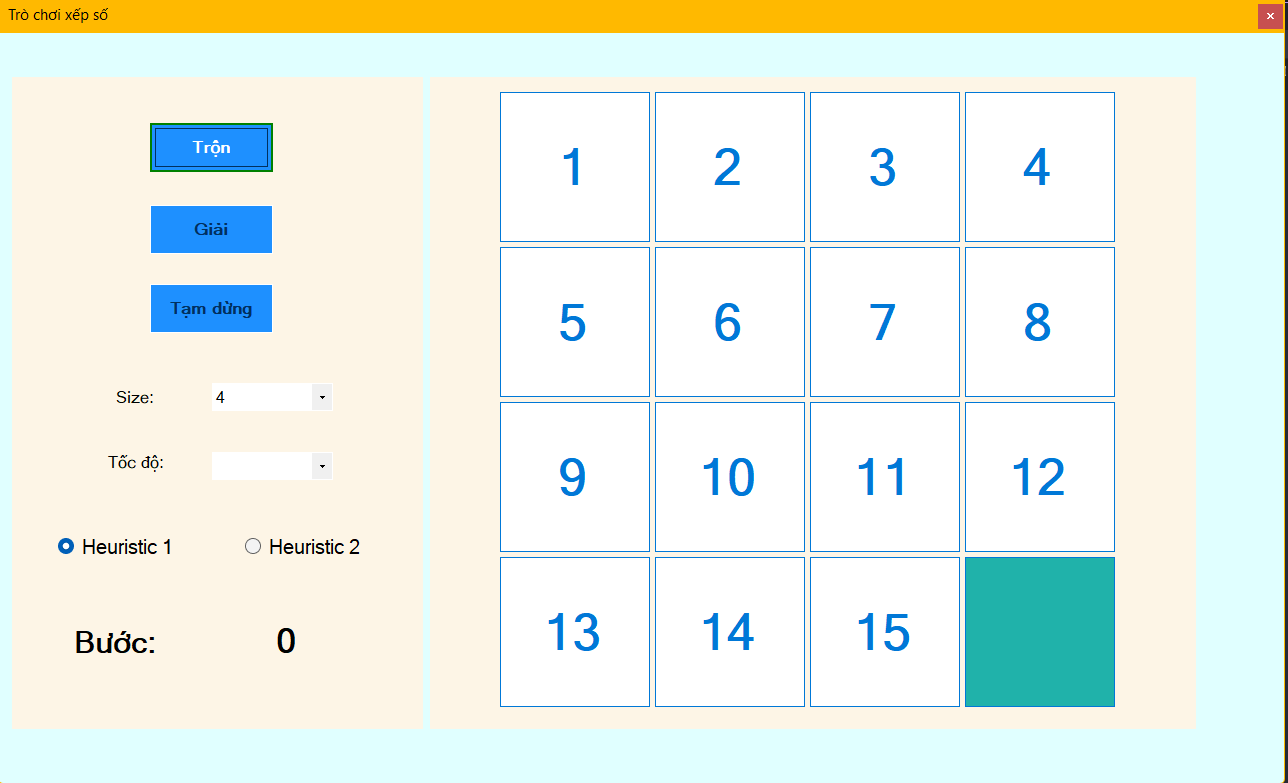
Phương thức GetBestPositionInOpen:

|  |
| --- |
| int **GetBestPositionInOpen**(List<Node> Open)  {  if (Open.Count != 0)  {  Node min = new Node();  min = Open[0];  int vt = 0;  for (int i = 1; i < Open.Count; i++)  if (min.heuristicCost > Open[i].heuristicCost)  {  min = Open[i];  vt = i;  }  else  {  if (min.heuristicCost == Open[i].heuristicCost)  {  if (min.fn > Open[i].fn)  {  min = Open[i];  vt = i;  }  }  }  return vt;  }  return 0;  } |

Giao diện chương trình:



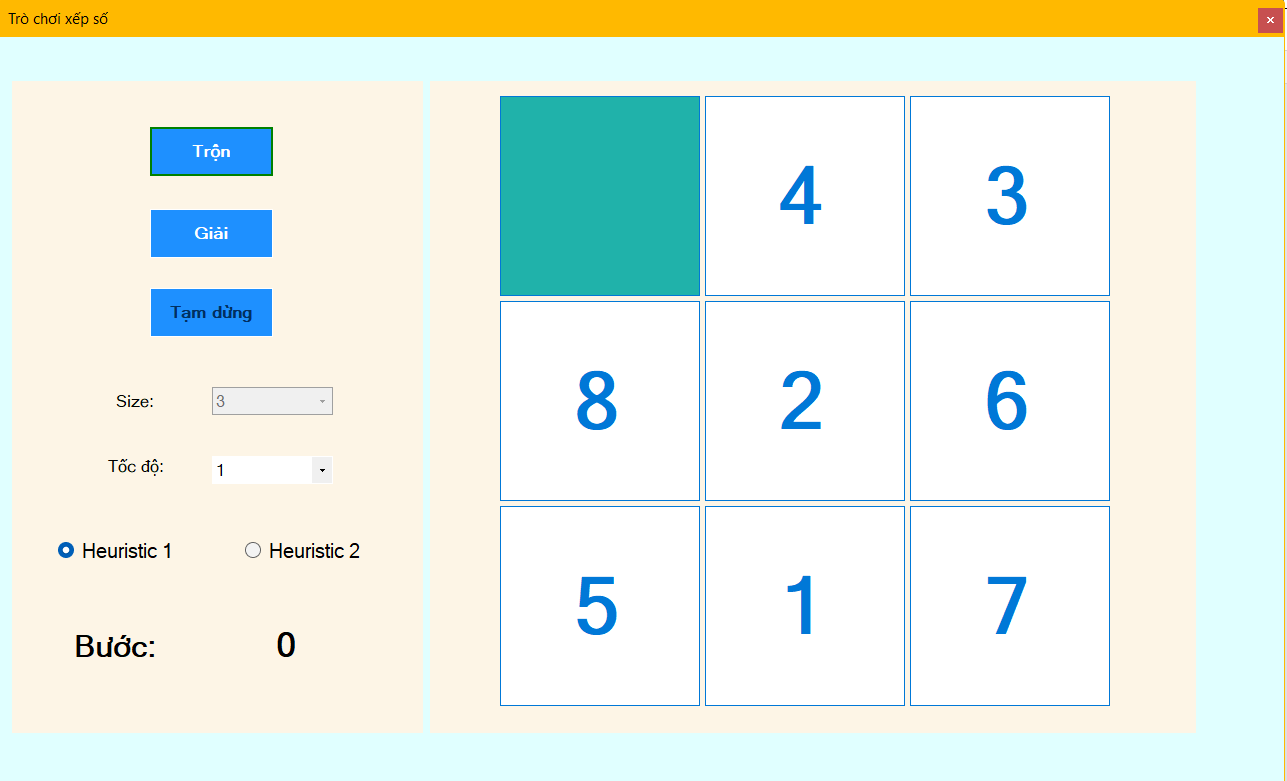
Hình 4. Giao diện ban đầu của chương trình với size=3



Hình 5. Giao diện ban đầu của chương trình với size=4

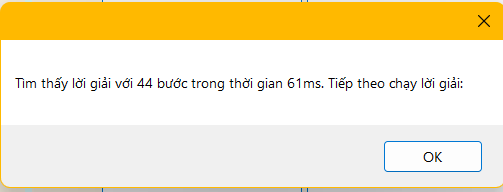
Giao diện gồm có 2 phần:

* Phần 1: phía bên trái, có 3 nút lần lượt là “Trộn”, “Giải” và “Tạm dừng”. “Size” và “Tốc độ” là 2 combobox, size dùng để chọn kích thước của ma trận là 9 ô hay 16 ô còn tốc độ dùng để điều chỉnh tốc độ sắp xếp của chương trình. Hai radioButton là “Heuristic 1” và “Heuristic 2” dùng để lựa chọn hàm Heuristic để thực hiện giải thuật. Một nhãn hiển thị số bước đã di chuyển.
* Phần 2: phía bên phải, hiển thị trạng thái của ma trận. Khi mới chạy, chương trình hiển thị trạng thái đích.



Hình 6. Giao diện chương trình sau khi nhấn nút "Trộn" với size=3

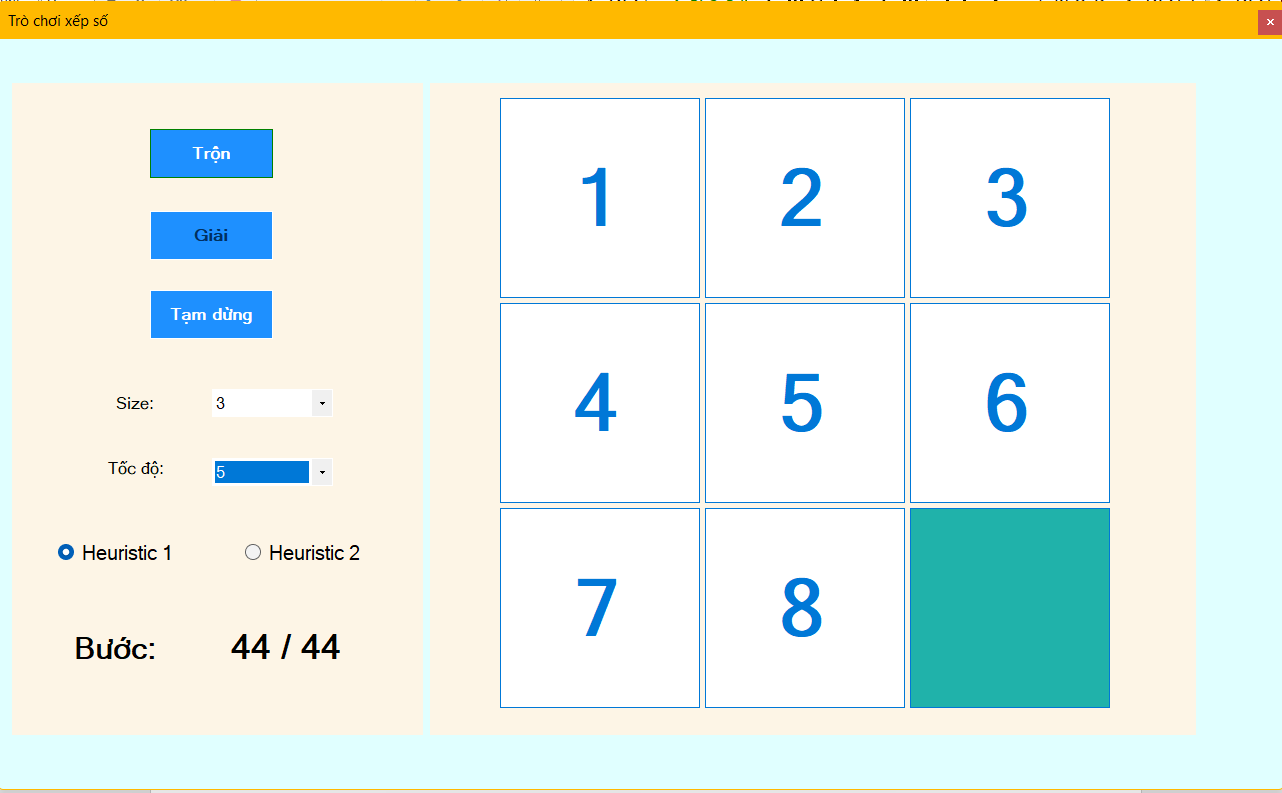
Khi nhấn nút “Trộn” thì sẽ tạo ngẫu nhiên 1 ma trận làm đầu vào cho chương trình. Khi nhấn “Giải” thì chương trình thực hiện giải thuật để tìm kiếm lời giải. Khi tìm thấy lời giải thì sẽ hiển thị thông báo tìm thấy lời giải trong thời gian bao nhiêu cùng với số bước.



Hình 7. Thông báo tìm thấy lời giải

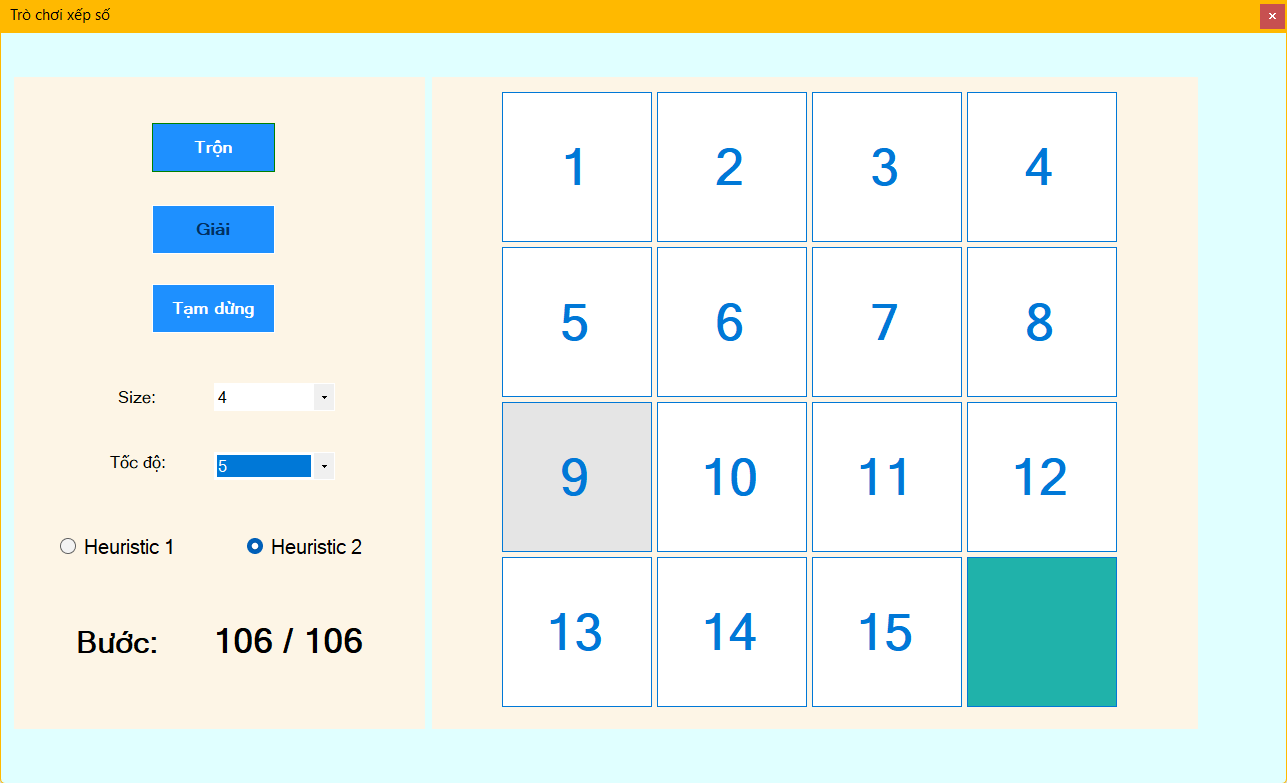
Sau đó nhấn “OK” thì chương trình sẽ tự động chạy lời giải để đưa các ô về trạng thái đích. Người dùng có thể tam dừng, hay điều chỉnh tốc độ chạy lời giải của chương trình.

Kết quả sau khi chạy với size=3 và Heuristic 1



Hình 8. Kết quả sau khi chạy với size=3 và Heuristic 1

Kết quả sau khi chạy tương tự với size=4



Hình 9. Kết quả sau khi chạy với size=4 và Heuristic 2

Có một số trường hợp với size=4 do không gian lớn nên thời gian tìm kiếm lời giải có thể lâu hơn (lớn hơn 20 giây) thì sẽ dừng tìm kiếm và hiển thị thông báo: “Thời gian tìm kiếm quá lâu > 20000ms”.

# Bài 3: Bài toán TSP sử dụng thuật toán di truyền GA

## Giới thiệu bài toán TSP (Travelling Salesman Problem)

Bài toán TSP là một bài toán tối ưu hóa trong lĩnh vực quy hoạch tuyến tính, được xem là một trong những bài toán cơ bản và phổ biến nhất trong lý thuyết đồ thị và tối ưu hóa.Bài toán người bán hàng là một bài toán NP-khó thuộc thể loại tối ưu rời rạc hay tổ hợp được nghiên cứu trong vận trù học hoặc lý thuyết khoa học máy tính. Bài toán đặt ra câu hỏi: "Một người bán hàng cần thăm n thành phố khác nhau và quay về thành phố xuất phát. Hãy tìm một hành trình ngắn nhất mà người bán hàng có thể thăm tất cả các thành phố một lần và quay về thành phố xuất phát. Hiện nay, giải thuật tối ưu nhất để giải quyết bài toán này là kiểu đệ quy quay lui hoặc duyệt toàn bộ trạng thái, nhưng với số lượng thành phố lớn, việc duyệt tất cả các trạng thái trở nên không khả thi.

## Cơ sở lý thuyết

* 1. Tiến hóa tự nhiên

Quá trình tiến hóa tự nhiên là quá trình mà thế hệ sau thường có xu hướng tốt hơn thế hệ trước đó. Điều này xảy ra do các cá thể có đặc điểm phù hợp với môi trường sẽ sinh sống và sinh sản tốt hơn, từ đó di truyền những đặc điểm tích cực cho thế hệ tiếp theo. Quá trình tiến hóa này dẫn đến sự tiến bộ và thích nghi của các loài trong tự nhiên.

Tiến hóa tự nhiên được duy trì nhờ hai quá trình cơ bản:

1. Sinh sản: Đây là quá trình sinh sản và tái sản xuất của các cá thể trong quần thể. Các cá thể sinh sản tạo ra thế hệ tiếp theo, và thông qua quá trình di truyền gen, các đặc tính di truyền từ cha mẹ được chuyển đạt cho con cái. Quá trình sinh sản này đảm bảo sự tiếp tục của loài và cung cấp nguồn gen đa dạng để tạo ra các biến thể.

2. Chọn lọc tự nhiên: Đây là quá trình tự nhiên chọn lọc những cá thể có đặc điểm phù hợp với môi trường để sống sót và sinh sản. Những cá thể có đặc điểm vượt trội, giúp chúng tồn tại và sinh sản tốt hơn, có khả năng di truyền những đặc điểm tích cực cho thế hệ tiếp theo. Ngược lại, các cá thể không phù hợp sẽ có cơ hội sống sót và sinh sản thấp hơn.

Sự thay đổi môi trường là động lực thúc đẩy quá trình tiến hóa. Khi môi trường thay đổi, các cá thể phải thích nghi để tồn tại. Những cá thể có đặc tính phù hợp với môi trường mới sẽ có lợi thế sinh tồn và di truyền đặc tính này cho thế hệ sau. Điều này giúp loài tiến hóa và phát triển theo thời gian.

Các cá thể sinh ra trong quá trình tiến hóa nhờ sự lai ghép ở thế hệ bố mẹ. Một cá thể mới có thể mang những đặc tính của thế hệ bố mẹ thông qua quá trình di truyền gen. Tuy nhiên, cũng có thể xảy ra đột biến, khi một cá thể mới có thể mang những đặc tính hoàn toàn mới, không xuất phát từ thế hệ bố mẹ. Đột biến xảy ra với xác suất nhỏ hơn nhiều so với quá trình di truyền gen, nhưng nó là nguồn cung cấp đa dạng gen cho quần thể, giúp các loài thích nghi với môi trường mới và tiến hóa theo hướng mới.

Các thuật toán di truyền dựa trên các quá trình cơ bản như lai ghép, đột biến, sinh sản và chọn lọc tự nhiên. Các quá trình này được mô phỏng dưới dạng các phép toán trong giải thuật di truyền GA, giúp tìm kiếm các giải pháp tối ưu cho các bài toán tối ưu hóa và trí tuệ nhân tạo.

* 1. Thuật toán di truyền GA (Genetic Algorithm)

Giải thuật di truyền GA là kỹ thuật phỏng theo quá trình thích nghi tiến hóa của các quần thể sinh học dựa trên học thuyết Darwin. GA là phương pháp tìm kiếm tối ưu ngẫu nhiên bằng cách mô phỏng theo sự tiến hóa của con người hay của sinh vật. Tư tưởng của thuật toán di truyền là mô phỏng các hiện tượng tự nhiên, là kế thừa và đấu tranh sinh tồn.

Cấu trúc dữ liệu biểu diễn không gian lời giải của bài toán: Trong giải thuật di truyền, không gian lời giải thường được biểu diễn bằng chuỗi các gen. Mỗi gen thể hiện một thành phần của lời giải và có thể là một số, ký tự, hoặc bất kỳ kiểu dữ liệu nào phù hợp với bài toán cụ thể. Ví dụ, trong bài toán TSP, mỗi gen có thể là chỉ số đại diện cho một thành phố trong hành trình du lịch. Các chuỗi gen này tạo nên một cá thể trong quần thể.

Phương pháp khởi tạo quần thể ban đầu P(0): Quần thể ban đầu P(0) được tạo ra bằng cách ngẫu nhiên tạo ra một số lượng cá thể, mỗi cá thể biểu diễn một lời giải tiềm năng cho bài toán. Quá trình khởi tạo này có thể được thực hiện bằng cách tạo ngẫu nhiên các chuỗi gen hoặc áp dụng các phương pháp tạo lời giải khởi đầu thông minh hơn dựa trên kiến thức về bài toán.

Hàm xác định độ thích nghi Eval(): Hàm xác định độ thích nghi (hoặc hàm mục tiêu - objective function) có nhiệm vụ đánh giá mức độ tốt (hoặc xấu) của mỗi cá thể trong quần thể. Hàm này sẽ tính toán giá trị số đại diện cho chất lượng của lời giải, dựa trên yêu cầu của bài toán. Độ thích nghi càng cao nghĩa là lời giải càng tốt và ngược lại. Mục tiêu của GA là tối ưu hóa hàm đánh giá này để tìm kiếm lời giải tối ưu hoặc gần tối ưu nhất.

Các phép toán di truyền (lai ghép, đột biến, sinh sản và chọn lọc tự nhiên):

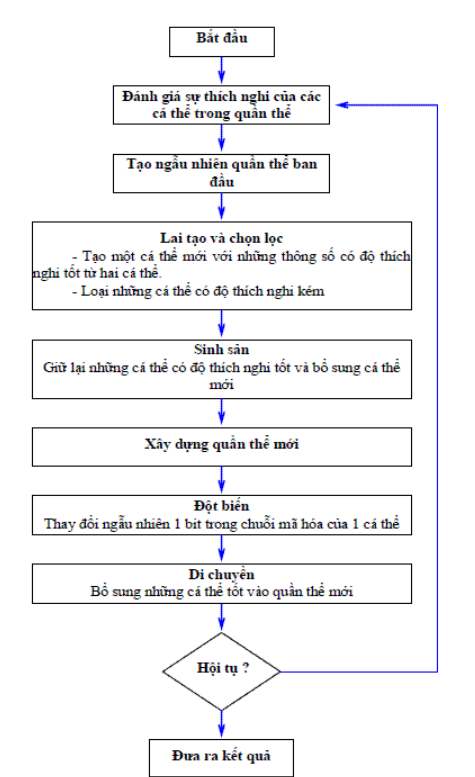
* Lai ghép (Crossover): Quá trình lai ghép kết hợp các chuỗi gen của hai cá thể cha mẹ để tạo ra cá thể con mới. Các phương pháp lai ghép khác nhau được áp dụng để tạo sự đa dạng và khám phá không gian giải pháp.
* Đột biến (Mutation): Quá trình đột biến làm thay đổi một số gen trong cá thể con mới một cách ngẫu nhiên. Đột biến giúp giữ lại tính ngẫu nhiên và giải quyết vấn đề rơi vào cực tiểu cục bộ.
* Sinh sản (Reproduction): Quá trình sinh sản tạo ra các cá thể con mới từ các cá thể cha mẹ thông qua lai ghép và đột biến.
* Chọn lọc tự nhiên (Natural selection): Quá trình chọn lọc xác định cá thể nào sẽ được giữ lại trong quần thể cho thế hệ tiếp theo dựa trên độ thích nghi của từng cá thể. Các cá thể có độ thích nghi cao sẽ có cơ hội cao hơn để được chọn lọc và tiếp tục tham gia vào quá trình tiến hóa.

Các tham số cần sử dụng trong thuật giải (kích thước quần thể, xác suất lai, xác suất đột biến, ...): Các tham số cần sử dụng trong GA ảnh hưởng đến hiệu suất và kết quả của thuật toán. Các tham số quan trọng bao gồm:

* Kích thước quần thể (population size): Số lượng cá thể trong mỗi thế hệ.
* Xác suất lai ghép (crossover rate): Xác suất một cặp cá thể cha mẹ thực hiện quá trình lai ghép.
* Xác suất đột biến (mutation rate): Xác suất một cá thể con mới bị đột biến sau quá trình lai ghép.
* Tiêu chuẩn dừng (termination criteria): Điều kiện để dừng quá trình tiến hóa, như số thế hệ tối đa hoặc đạt được giá trị mục tiêu đủ tốt.

Việc tinh chỉnh các tham số này đòi hỏi kiến thức và kinh nghiệm để đạt hiệu suất tốt nhất cho giải thuật di truyền trong từng bài toán cụ thể.

Sơ đồ thuật toán của GA:

****

Thuật giải GA đã và đang được ứng dụng để giải quyết các bài toán trong rất nhiều lĩnh vực của cuộc sống cũng như trong kỹ thuật.

## Cài đặt chương trình

Sử dụng ngôn ngữ lập trình C++ để thực hiện.

Đầu vào: số đỉnh thành phố, ma trận khoảng cách, kích thước quần thể, số thế hệ.

Đầu ra: Tìm được đường đi (chuỗi gen) tối ưu.

Code chương trình:

Cấu trúc chung biểu diễn 1 cá thể trong quần thể:

|  |
| --- |
| // Cau truc chung bieu dien 1 ca the trong quan the  struct **Individual** {  std::vector<int> chromosome; // Chuoi chua cac dinh (gen)  int fitness; // Gia tri fitness (chi phi) cua ca the  **Individual**() {  chromosome.resize(N);  for (int i = 0; i < N; ++i)  chromosome[i] = i;  std::random\_shuffle(chromosome.begin(), chromosome.end()); // Hoan vi ngau nhien cac gen trong chuoi  fitness = calculateFitness(); // Tinh finess cua ca the ngay tu khi khoi tao  }  // Tinh Fitness cua ca the bang cach tinh tong khoang cach giua cac dinh trong chromosome  int **calculateFitness**() {  int sum = 0;  for (int i = 0; i < N - 1; ++i)  sum += dist[chromosome[i]][chromosome[i + 1]];  // Cong them khoang cach tu diem cuoi cung den diem dau tien  sum += dist[chromosome[N - 1]][chromosome[0]];  return sum;  }  }; |

Hàm lai ghép giữa 2 cá thể để tạo ra cá thể con mới:

|  |
| --- |
| Individual **crossover**(const Individual &parent1, const Individual &parent2) {  int len = parent1.chromosome.size(); // Do dai chuoi gen cua ca the cha me  int cutPoint1 = rand() % len;// Diem cat 1 - ngau nhien chon 1 vi tri cat trong chuoi gen  int cutPoint2 = rand() % len;// Diem cat 2 - ngau nhien chon 1 vi tri cat khac trong chuoi gen  // neu diem cat 2 < diem cat 1 thi dao vi tri lai  if (cutPoint2 < cutPoint1)  std::swap(cutPoint1, cutPoint2);  std::vector<bool> used(len, false); // chua cac gen da su dung trong qua trinh lai ghep  Individual child;  for (int i = cutPoint1; i <= cutPoint2; ++i) {  // sao chep cac gen trong khoang [cutPoint1, cutPoint2] cua ca the cha vao ca the con  child.chromosome[i] = parent1.chromosome[i];  used[parent1.chromosome[i]] = true; // danh dau cac gen da duoc su dung tu ca the cha  }  int pos = 0;  for (int i = 0; i < len; ++i) {  if (!used[parent2.chromosome[i]]) {  while (used[child.chromosome[pos]])  ++pos; // tim vi tri trong cua ca the con  // sao chep cac gen tu ca the me vao vi tri trong cua ca the con  child.chromosome[pos] = parent2.chromosome[i];  used[parent2.chromosome[i]] = true; // danh dau cac gen da duoc su dung tu ca the me  }  }  // Tinh gia tri Fitness cua ca the con sau khi lai ghep  child.fitness = child.calculateFitness();  return child;  } |

Hàm đột biến cho 1 cá thể:

|  |
| --- |
| void **mutate**(Individual &individual) {  const double mutationRate = 0.2; // ty le dot bien  if ((double)rand() / RAND\_MAX < mutationRate) {  int len = individual.chromosome.size(); // do dai cua chuoi gen (chromosome) trong ca the  int pos1 = rand() % len; // chon ngau nhien vi tri thu 1 de dot bien  int pos2 = rand() % len; // chon ngau nhien vi tri thu 2 de dot bien  // hoan doi gia tri cua 2 gen tai vi tri 1 va 2 trong chuoi gen, thuc hien qua trinh dot bien  std::swap(individual.chromosome[pos1], individual.chromosome[pos2]);  // cap nhat lai gia tri Fitness bang cach goi ham calculateFitness  individual.fitness = individual.calculateFitness();  }  } |

Hàm giải thuật GA để giải bài toán TSP:

|  |
| --- |
| void **geneticAlgorithm**(int populationSize, int numGenerations) {  std::vector<Individual> population(populationSize); // tao 1 quan the ban co kich thuoc populationSize  const double crossoverRate = 0.8; // ty le lai ghep  // Khoi tao quan the ban dau  #pragma omp parallel for  for (int i = 0; i < populationSize; ++i)  population[i] = Individual();    // Tien hanh tien hoa qua cac the he  for (int generation = 1; generation <= numGenerations; ++generation) {  // Sap xep quan the theo thu tu tang dan cua Fitness  std::sort(population.begin(), population.end(), cmp);  int eliteSize = populationSize / 5; // kich thuoc cua nhom ca the (dai dien cho nhung ca the tot nhat)    // Chon cac ca the tu nhom elite de tao lai quan the moi (loai bo nhung ca the yeu)  #pragma omp parallel for  for (int i = eliteSize; i < populationSize; ++i)  population[i] = population[rand() % eliteSize];  // Lai ghep cac ca the cha me de tao ra cac ca the con moi, cho den so luong ca the dat du populationSize  while (population.size() < populationSize) {  int parent1Index = rand() % eliteSize;  int parent2Index = rand() % eliteSize;  if ((double)rand() / RAND\_MAX < crossoverRate)  population.push\_back(crossover(population[parent1Index], population[parent2Index]));  }  // Tien hanh dot bien ngau nhien cho tung ca the trong quan the  #pragma omp parallel for  for (int i = 0; i < populationSize; ++i)  mutate(population[i]);  // In thong tin ve the he hien tai va ca the tot nhat trong quan the  printGeneration(generation, population);  }  // ket thuc thuat toan, sap xep la quan the theo thu tu tang dan cua gia tri Fitness  std::sort(population.begin(), population.end(), cmp);  std::cout << "Ket qua sau di truyen: " << population[0].fitness << "\n";  std::cout << "Chuoi dinh (gen) toi uu: ";  for (int i = 0; i < N; ++i) {  std::cout << population[0].chromosome[i] << " -> ";  }  std::cout << population[0].chromosome[0];  std::cout << "\n";  } |

Hàm in thông tin về thế hệ hiện tại:

|  |
| --- |
| void **printGeneration**(int generation, const std::vector<Individual> &population) {  std::cout << "The he " << generation << ":\n";  const Individual &bestIndividual = \*std::min\_element(population.begin(), population.end(), cmp);  std::cout << "Chuoi gen tot nhat: ";  for (int j = 0; j < N; ++j) {  std::cout << bestIndividual.chromosome[j];  if (j != N - 1)  std::cout << " -> ";  }  std::cout << " -> " << bestIndividual.chromosome[0];  std::cout << ", Fitness: " << bestIndividual.fitness << "\n";  std::cout << "-----------------------------\n";  } |

Hàm main:

|  |
| --- |
| int **main**() {  std::srand(std::time(0));  int populationSize; // Kich thuoc quan the  int numGenerations; // So the he  std::cout << "Chuong trinh giai bai toan TSP bang thuat toan di truyen GA:";  std::cout << "\n Ban muon doc du lieu tu file nao";  std::cout << "\n 1. input.txt";  std::cout << "\n 2. input2.txt";  int chose;  std:: string fileName="";  do {  std::cout << "\nNhap lua chon cua ban (1 hoac 2): ";  std::cin >> chose;  if (chose!=1 && chose!=2) std::cout << "Lua chon khong hop le. Vui long nhap lai:";  } while (chose!=1 && chose!=2);  // Kiem tra nguoi dung chon doc du lieu tu file nao  if (chose == 1) {  fileName = "input.txt";  }  else if (chose == 2) {  fileName = "input2.txt";  }  // Doc du lieu tu file  std::ifstream inputFile(fileName.c\_str());  if (!inputFile) {  std::cerr << "Khong the mo file.\n";  return 1;  }  inputFile >> N;  if (N <= 0 ) {  std::cerr << "So dinh không hop le.\n";  return 1;  }  // Doc kich thuoc quan the (so luong ca the) va so the he  inputFile >> populationSize;  inputFile >> numGenerations;  for (int i = 0; i < N; ++i)  for (int j = 0; j < N; ++j)  inputFile >> dist[i][j];  inputFile.close();  geneticAlgorithm(populationSize, numGenerations );  return 0;  } |

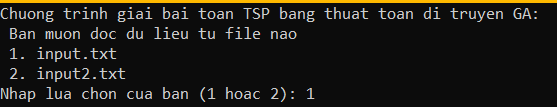
Chương trình đọc dữ liệu từ file excel input.txt hoặc file input2.txt:

File input.txt:

|  |
| --- |
| 20  200  300  0 10 15 20 30 40 35 25 30 20 15 25 30 45 50 55 60 65 70 75  10 0 35 25 40 50 45 35 40 30 25 35 40 55 60 65 70 75 80 85  15 35 0 30 50 60 55 45 50 40 35 45 50 65 70 75 80 85 90 95  20 25 30 0 45 55 50 40 45 35 30 40 45 60 65 70 75 80 85 90  30 40 50 45 0 10 25 35 40 50 55 45 40 55 60 65 70 75 80 85  40 50 60 55 10 0 35 45 50 60 65 55 50 65 70 75 80 85 90 95  35 45 55 50 25 35 0 10 15 25 30 20 15 30 35 40 45 50 55 60  25 35 45 40 35 45 10 0 20 30 35 25 20 35 40 45 50 55 60 65  30 40 50 45 40 50 15 20 0 10 15 5 10 25 30 35 40 45 50 55  20 30 40 35 50 60 25 30 10 0 5 15 20 35 40 45 50 55 60 65  15 25 35 30 55 65 30 35 15 5 0 10 15 30 35 40 45 50 55 60  25 35 45 40 45 55 20 25 5 15 10 0 5 20 25 30 35 40 45 50  30 40 50 45 40 50 15 20 10 20 15 5 0 15 20 25 30 35 40 45  45 55 65 60 55 65 30 35 25 35 30 20 15 0 5 10 15 20 25 30  50 60 70 65 60 70 35 40 30 40 35 25 20 5 0 5 10 15 20 25  55 65 75 70 65 75 40 45 35 45 40 30 25 10 5 0 5 10 15 20  60 70 80 75 70 80 45 50 40 50 45 35 30 15 10 5 0 5 10 15  65 75 85 80 75 85 50 55 45 55 50 40 35 20 15 10 5 0 5 10  70 80 90 85 80 90 55 60 50 60 55 45 40 25 20 15 10 5 0 5  75 85 95 90 85 95 60 65 55 65 60 50 45 30 25 20 15 10 5 0 |

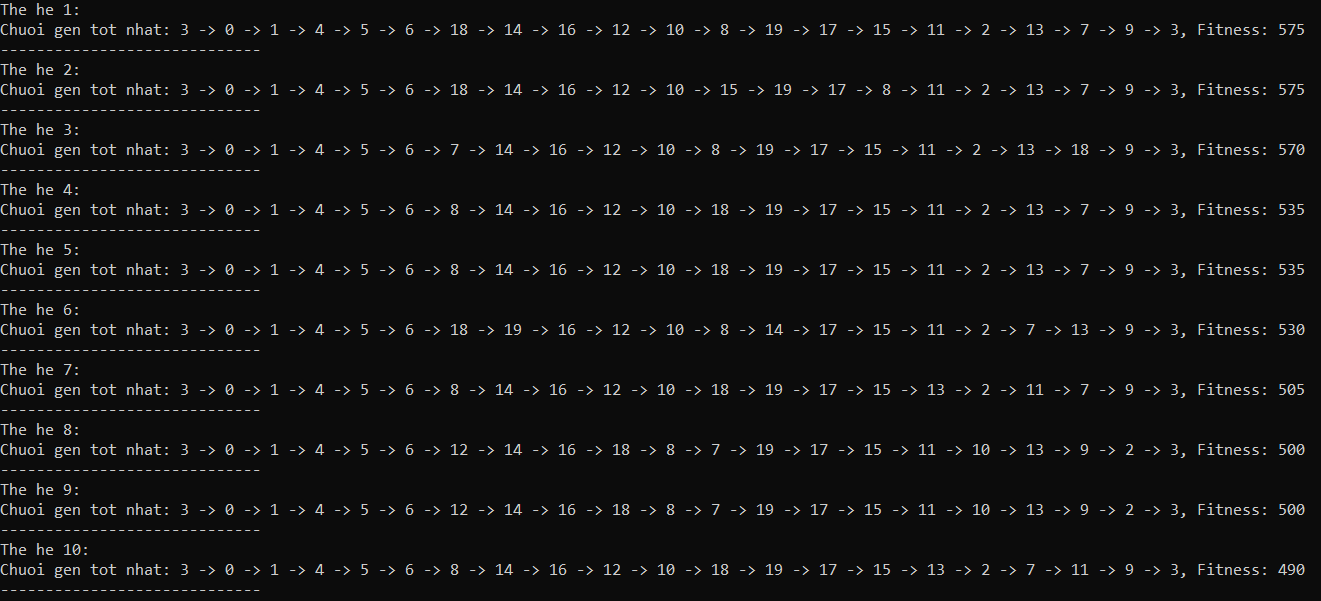
Dòng đầu tiên là số đỉnh của đồ thị, dòng thứ 2 là kích thước của quẩn thể, dòng thứ 3 là số thế hệ, cuối cùng là ma trân khoảng cách.

Giao diện chương trình:



Hình 10. Lựa chọn file muốn đọc

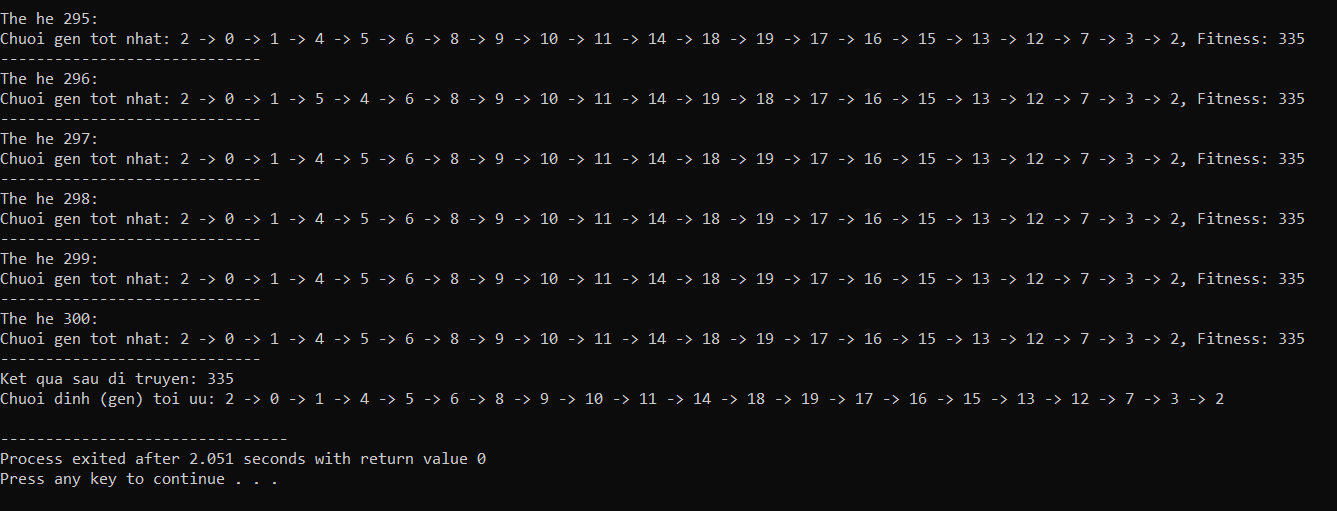
Chọn file muốn đọc input.txt hoặc input2.txt



Hình 11. Một đoạn kết quả khi chạy chương trình

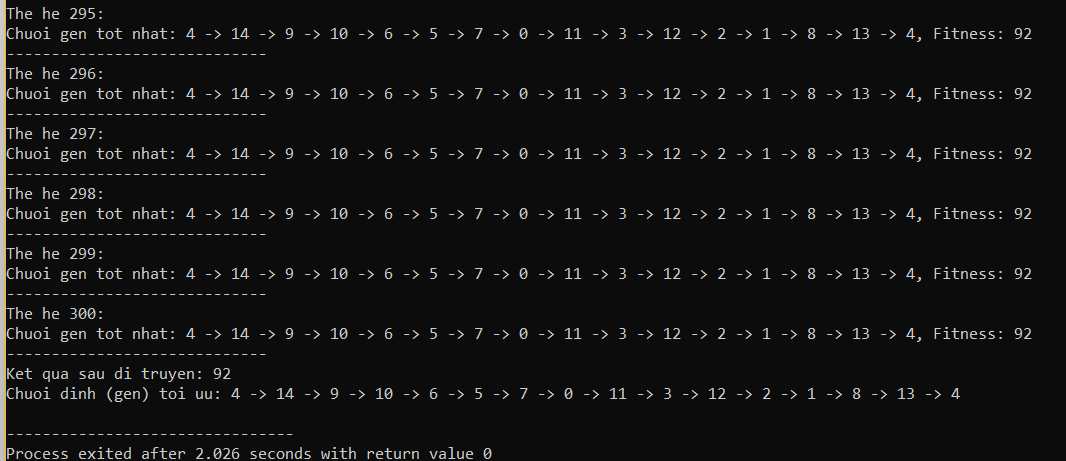
Đây chỉ là 1 đoạn kết quả khi chạy chương trình với file input.txt. Kết quả in ra màn hình gồm có thế hệ và chuỗi gen tốt nhất kèm theo giá trị Fitness.

Kết quả sau khi chạy xong chương trình với file input.txt:



Hình 12. Kết quả sau khi chạy xong chương trình với file input.txt

Kết quả sau khi chạy xong chương trình với file input2.txt:



Hình 13. Kết quả sau khi chạy xong chương trình với file input2.txt

Tìm ra được đường đi (chuỗi gen) tối ưu cho bài toán

# Bài 4: Cài đặt thuật giải Quinlan trong máy học

## Lý thuyết

Là thuật toán học theo quy nạp dùng luật, đa mục tiêu. Do Quinlan đưa ra năm 1979.

Ý tưởng: Chọn thuộc tính quan trọng nhất để tạo cây quyết định.

Thuộc tính quan trọng nhất là thuộc tính phân loại bảng quan sát thành các bảng con sao cho từ mỗi bảng con này dễ phân tích để tìm quy luật chung.

## Thuật toán

B1: Phân loại thuộc tính dẫn xuất và thuộc tính mục tiêu

Thuộc tính mục tiêu: là thuộc tính quan tâm.

Thuộc tính dẫn xuất: là thuộc tính quan quan sát.

B2: Tính độ bất định Entropy cho mỗi thuộc tính và chọn thuộc tính có Entropy nhỏ nhất theo công thức:

B3: Tính vector đặt trưng của thuộc tính vừa chọn

VA(j) = (T(j , r1), T(j , r2) , …, T(j , rn))

T(j, ri) = (tổng số phần tử trong phân hoạch có giá trị thuộc tính dẫn xuất A là j và có giá trị thuộc tính mục tiêu là ri ) / ( tổng số phần tử trong phân hoạch có giá trị thuộc tính dẫn xuất A là j )

\* r1, r2, …, rn là các giá trị của thuộc tính mục tiêu

B4: Loại bỏ thuộc tính đã được phân hoạch

Nếu vẫn còn thuộc tính đẫn xuất quay lại bước 2 để tính độ bất định Etropy cho các thuộc tính dẫn xuất.

Ngược lại, kết thúc thuật toán.

Ví dụ minh họa:

Bảng khảo sát:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Age | Competition | Type | Profit |
| 1 | Old | No | Software | Down |
| 2 | Midle | Yes | Software | Down |
| 3 | Midle | No | Hardware | Up |
| 4 | Old | No | Hardware | Down |
| 5 | New | No | Hardware | Up |
| 6 | New | No | Software | Up |
| 7 | Midle | No | Software | Up |
| 8 | New | Yes | Software | Up |
| 9 | Midle | Yes | Hardware | Down |
| 10 | Old | Yes | Hardware | Down |

Tính độ bất định cho mỗi thuộc tính và chọn thuộc tính có độ bất định nhỏ nhất

**Text, letter

Description automatically generated**

Tương tự:

E(C/Age) = 0.4

E(C/Type) = 0.97024

Age có độ bất định nhỏ nhất (có nhiều thông tin nhất)

Tính vector đặt trưng của thuộc tính Age

VAge (Old) = (0, 1)

VAge (New) = (1, 0)

VAge (Middle) = (2/4, 2/4)

Loại bỏ thuộc tính đã phân hoạch

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Competition | Type | Profit |
| 1 | Yes | Software | Down |
| 2 | No | Hardware | Up |
| 3 | No | Software | Up |
| 4 | Yes | Hardware | Down |

**VCompetition (Yes) = (0, 1)**

**VCompetition (No) = (1, 0)**

**A picture containing chart

Description automatically generated**

## Cài đặt chương trình

Sử dụng ngôn ngữ lập trình java

Đầu vào: dữ liệu được biểu diễn bằng các cột thuộc tính và các hàng là các mẫu dữ liệu. Nói gọn lại là "Tập dữ liệu" (Dataset).

Đầu ra: Xây dựng cây quyết định, xuất ra các luật dựa trên cây quyết định.

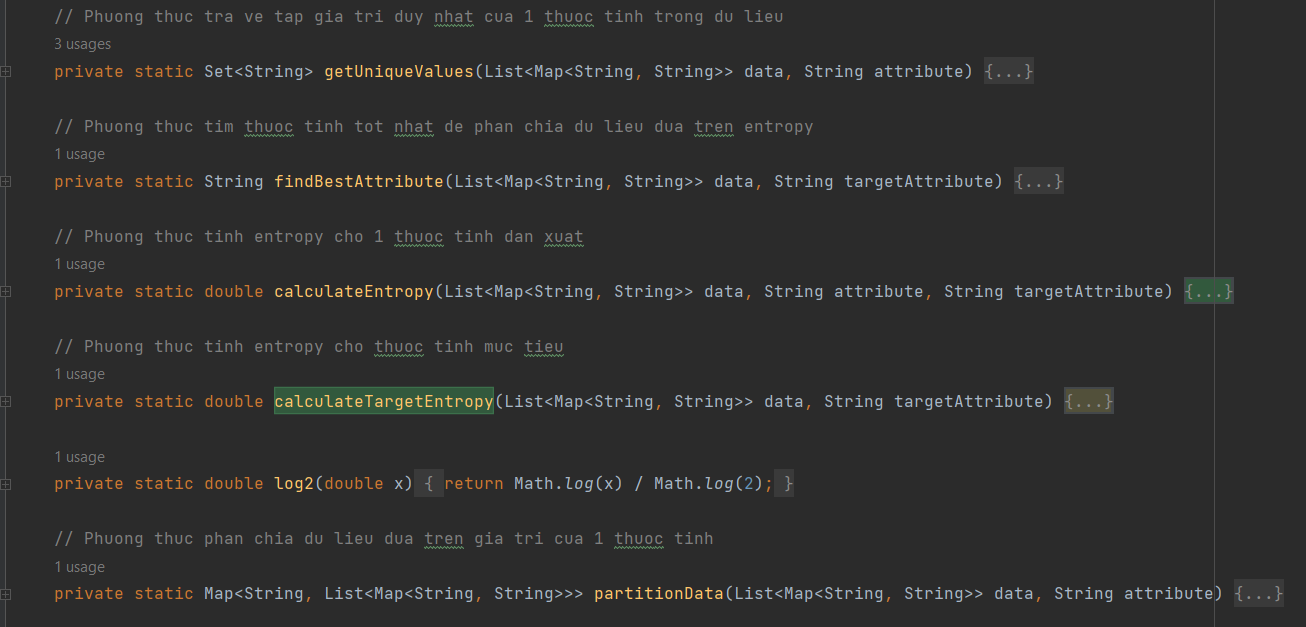
Code chương trình

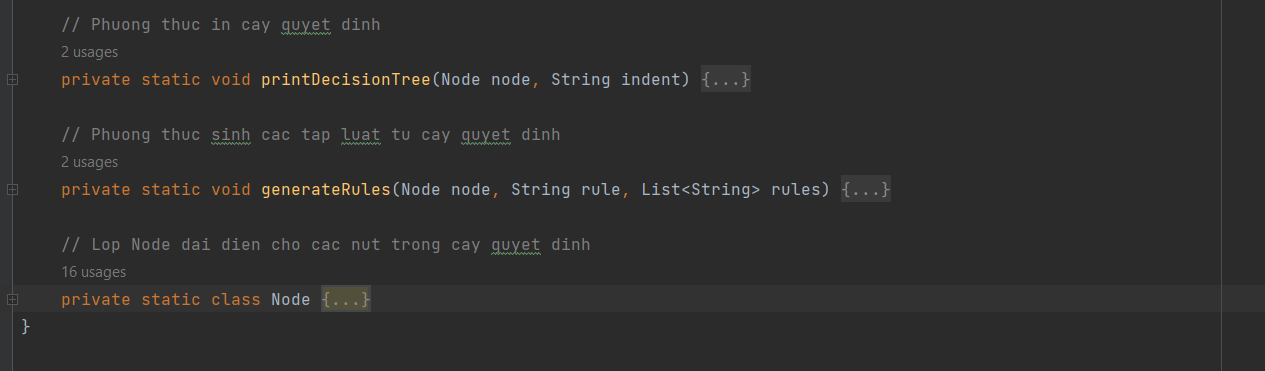
Class Node:

|  |
| --- |
| // Lop Node dai dien cho cac nut trong cay quyet dinh  private static class **Node** {  private String attribute;  private String targetValue; // chi cho cac nut la  private Map<String, Node> children;  public Node(String attribute) {  this.attribute = attribute;  this.children = new HashMap<>();  }  public String getAttribute() {  return attribute;  }  public String getTargetValue() {  return targetValue;  }  public void setTargetValue(String targetValue) {  this.targetValue = targetValue;  }  public Map<String, Node> getChildren() {  return children;  }  public void addChild(String attributeValue, Node childNode) {  children.put(attributeValue, childNode);  }  public boolean isLeaf() {  return targetValue != null;  }  } |

Class QuinlanAlgorithm:







Phương thức xây dựng cây quyết định

|  |
| --- |
| private static Node **buildDecisionTree**(List<Map<String, String>> data, String targetAttribute) {  Set<String> targetValues = getUniqueValues(data, targetAttribute);  // Neu tat ca cac mau deu co cung 1 gia tri thuoc tinh muc tieu, tra ve nut la voi gia tri thuoc tinh muc tieu do  if (targetValues.size() == 1) {  String targetValue = targetValues.iterator().next();  Node leafNode = new Node(null);  leafNode.setTargetValue(targetValue);  return leafNode;  }  // Tim thuoc tinh dan xuat tot nhat đe phan chia du lieu  String bestAttribute = findBestAttribute(data, targetAttribute);  Node root = new Node("" + bestAttribute);  // Phan chia du lieu và xay dung cây con cho tung nhanh  Map<String, List<Map<String, String>>> partitions = partitionData(data, bestAttribute);  for (Map.Entry<String, List<Map<String, String>>> entry : partitions.entrySet()) {  String attributeValue = entry.getKey();  List<Map<String, String>> partitionedData = entry.getValue();  Node subTree = buildDecisionTree(partitionedData, targetAttribute);  root.addChild(attributeValue, subTree);  }  return root;  } |

Phương thức tính entropy cho 1 thuộc tính dẫn xuất:

|  |
| --- |
| private static double **calculateEntropy**(List<Map<String, String>> data, String attribute, String targetAttribute) {  Set<String> uniqueValues = getUniqueValues(data, attribute);  double entropy = 0.0;  for (String value : uniqueValues) {  List<Map<String, String>> partitionedData = data.stream()  .filter(record -> record.get(attribute).equals(value))  .collect(Collectors.toList());  double probability = (double) partitionedData.size() / data.size();  double targetEntropy = calculateTargetEntropy(partitionedData, targetAttribute);  entropy += probability \* targetEntropy;  }  return entropy;  } |

Phương thức tìm thuộc tính tốt nhất để phân chia dữ liệu dựa trên entropy:

|  |
| --- |
| private static String **findBestAttribute**(List<Map<String, String>> data, String targetAttribute) {  double minEntropy = Double.MAX\_VALUE;  String bestAttribute = null;  // Duyet qua cac thuoc tinh dan xuat de tim thuoc tinh tot nhat  for (String attribute : data.get(0).keySet()) {  if (attribute.equals(targetAttribute)) continue; // Bo qua thuoc tinh muc tieu  double entropy = calculateEntropy(data, attribute, targetAttribute);  if (entropy < minEntropy) {  minEntropy = entropy;  bestAttribute = attribute;  }  }  return bestAttribute;  } |

Phương thức phân chia dữ liệu dựa trên giá trị của 1 thuộc tính:

|  |
| --- |
| Private static Map<String, List<Map<String, String>>>**partitionData**(List<Map<String, String>> data, String attribute) {  Map<String, List<Map<String, String>>> partitions = new HashMap<>();  for (Map<String, String> record : data) {  String attributeValue = record.get(attribute);  if (!partitions.containsKey(attributeValue)) {  partitions.put(attributeValue, new ArrayList<>());  }  partitions.get(attributeValue).add(record);  }  return partitions;  } |

Hàm main:

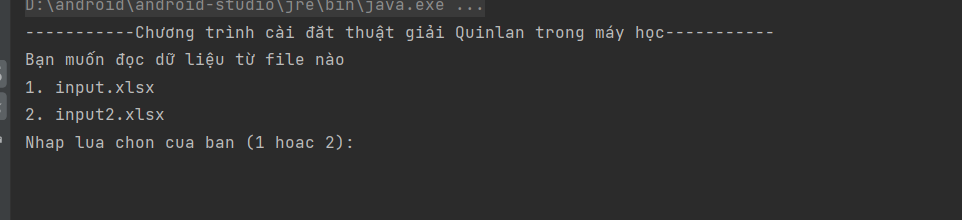
|  |
| --- |
| public static void **main**(String[] args) {  System.out.println("-----------Chương trình cài đăt thuật giải Quinlan trong máy học-----------");  System.out.println("Bạn muốn đọc dữ liệu từ file nào");  System.out.println("1. input.xlsx");  System.out.println("2. input2.xlsx");  int chose;  String fileName = "";  Scanner scanner = new Scanner(System.in);  do {  System.out.print("Nhập lựa chọn của bạn (1 hoặc 2): ");  chose = scanner.nextInt();  if (chose != 1 && chose != 2) {  System.out.println("Lựa chọn không hợp lệ. Vui lòng nhập lại:");  }  } while (chose != 1 && chose != 2);  // Kiem tra nguoi dung chon doc du lieu tu file nao  if (chose == 1) {  fileName = "input.xlsx";  } else if (chose == 2) {  fileName = "input2.xlsx";  }  // Buoc 1: Doc du lieu tu file  List<Map<String, String>> data = readDataFromExcel(fileName);  // In bang du lieu da doc duoc tu file  System.out.println("Bảng dữ liệu:");  printData(data);  // Buoc 2: Xay dung cay quyet dinh  String targetAttribute = getTargetAtrribute();  Node decisionTree = buildDecisionTree(data, targetAttribute);  // Buoc 3: Ve cay quyet dinh  System.out.println("\nCây quyết định:");  printDecisionTree(decisionTree, "");  // Buoc 4: In ra cac tap luat  System.out.println("\nCác tập luật:");  List<String> rules = new ArrayList<>();  generateRules(decisionTree, "", rules);  for (String rule : rules) {  System.out.println(rule);  }  } |

Chương trình đọc dữ liệu từ file excel input.xlsx hoặc file input2.xlsx:

File input.xlsx

|  |  |  |  |
| --- | --- | --- | --- |
| **Age** | **Competition** | **Type** | **Profit** |
| Old | No | Software | Down |
| Midle | Yes | Software | Down |
| Midle | No | Hardware | Up |
| Old | No | Hardware | Down |
| New | No | Hardware | Up |
| New | No | Software | Up |
| Midle | No | Software | Up |
| New | Yes | Software | Up |
| Midle | Yes | Hardware | Down |
| Old | Yes | Hardware | Down |

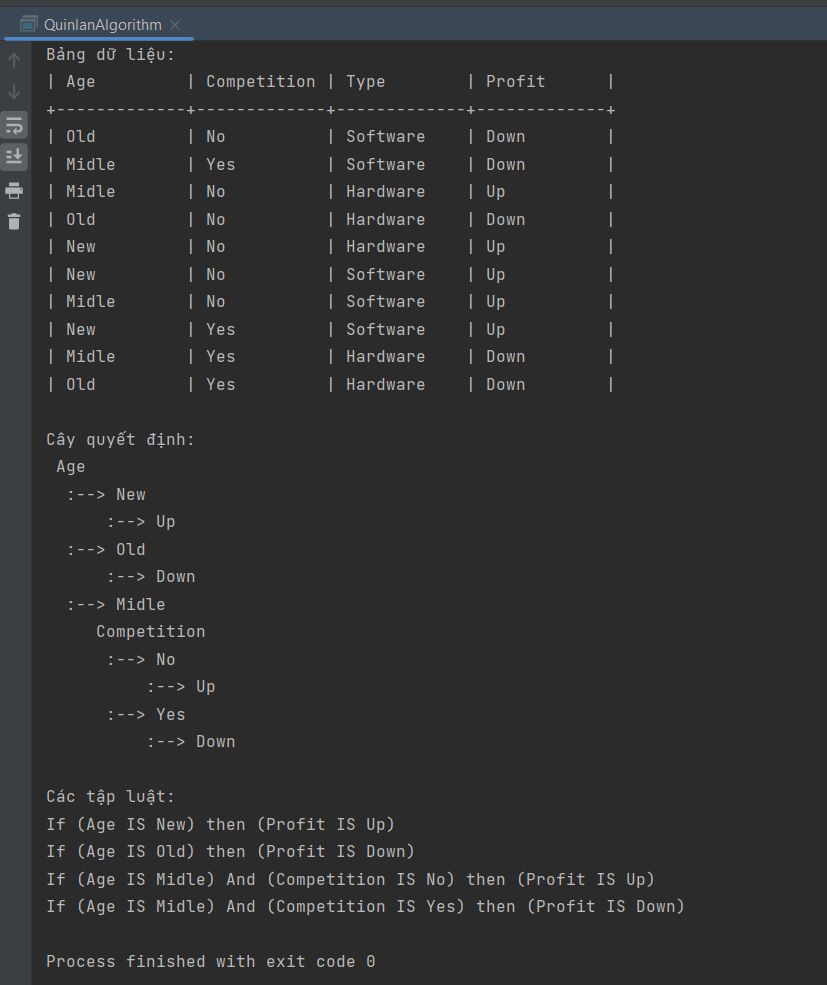
Giao diện chương trình:



Hình 14. Lựa chọn file excel muốn đọc

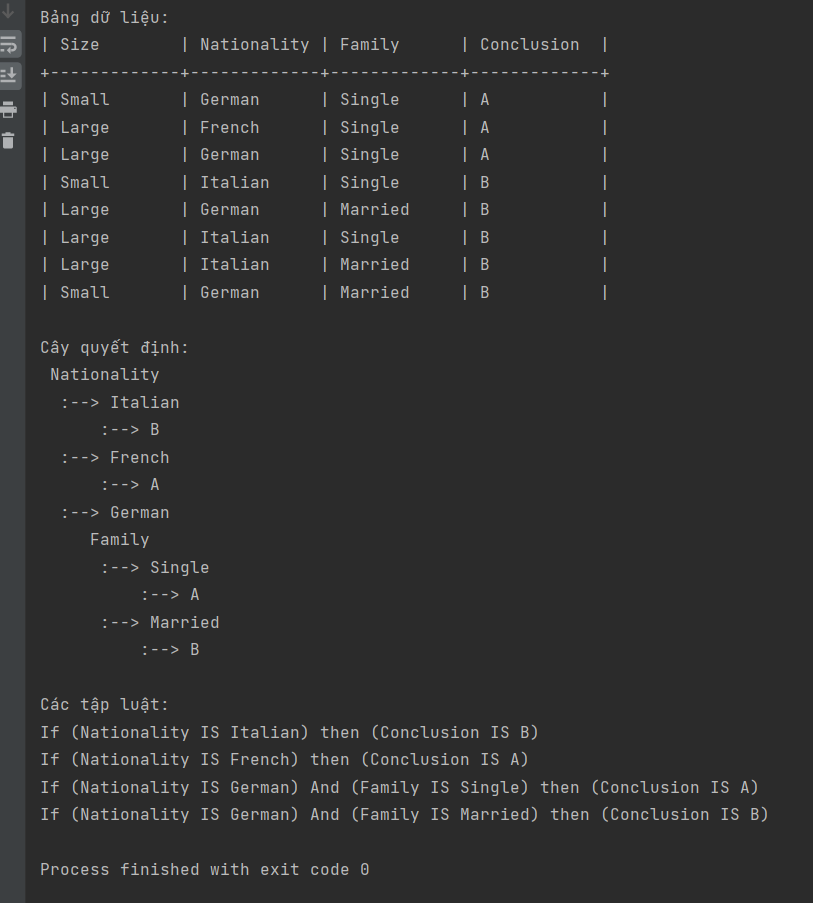
Chọn file dữ liệu muốn đọc.

Đây là kết quả khi chạy chương trình với đầu vào là file input.xlsx:



Hình 15. Kết quả khi chạy chương trình với đầu vào là file input.xlsx

Đây là kết quả khi chạy file input2.xlsx:



Hình 16. Kết quả khi chạy chương trình với đầu vào là file input2.xlsx

Kết quả gồm có:

* Một bảng dữ liệu đã đọc được từ file excel.
* Vẽ cây quyết định.
* Xây dựng các tập luật dựa trên cây quyết định.

# Bài 5: Tìm đường đi ngắn nhất trong mê cung bằng thuật toán A\*

## Giới thiệu

Bài toán tìm đường đi ngắn nhất trong mê cung là một bài toán quen thuộc trong lĩnh vực trí tuệ nhân tạo và thuật toán. Bài toán đặt ra câu hỏi: "Làm thế nào để tìm một đường đi từ một vị trí bắt đầu đến một vị trí kết thúc trong một mê cung sao cho độ dài của đường đi là ngắn nhất?"

Mô tả bài toán: cho trước một mê cung thường được biểu diễn dưới dạng một ma trận hai chiều, trong đó mỗi ô đại diện cho một vị trí trong mê cung.

Các giá trị trong ma trận có thể là:

* 0: Ô trống, có thể đi qua.
* 1: Tường, không thể đi qua.
* 2: Vị trí đích mà chúng ta muốn đến.

Mê cung có một điểm xuất phát và một điểm đích.

Nhiệm vụ là tìm một đường đi từ điểm xuất phát đến điểm đích, sao cho độ dài của đường đi là ngắn nhất. Bài toán tìm đường đi ngắn nhất trong mê cung thường được giải quyết bằng các thuật toán tìm kiếm đồ thị như thuật toán Dijkstra, thuật toán A\* và thuật toán BFS (Breath-First Search).

Bài toán tìm đường đi ngắn nhất trong mê cung có ứng dụng rất rộng trong thế giới thực như trong ngành robot hướng dẫn, điều hướng trong trò chơi, hệ thống GPS, và nhiều lĩnh vực khác đòi hỏi xác định đường đi tối ưu.

## Thuật toán A\* (đã trình bày ở bài 2)

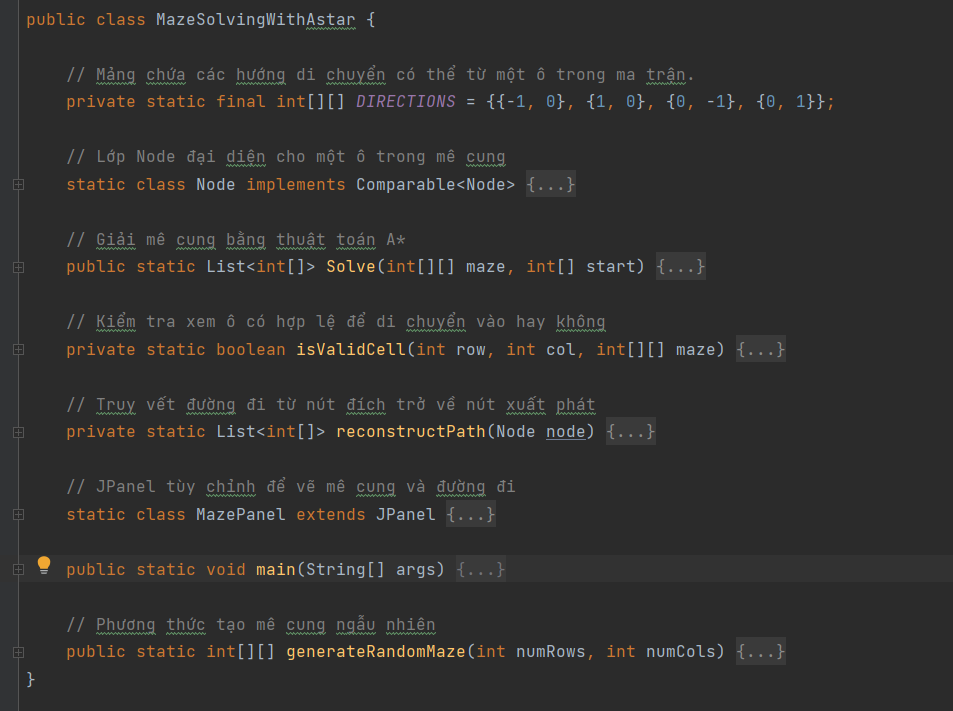
## Cài đặt chương trình

Sử dụng ngôn ngữ lập trình java kết hợp với thư viện đồ họa swing

Đầu vào: một ma trận biểu diễn mê cung, trong đó: 0 đại diện cho ô trống có thể di chuyển qua, 1 đại diện cho tường không thể di chuyển qua và 2 đại diện cho ô đích. Một điểm xuất phát.

Đầu ra: đường đi từ điểm bắt đầu đến điểm đích trong mê cung

Class MazeSolvingWithAstar:



Class Node:

|  |
| --- |
| static class **Node** implements Comparable<Node> {  private final int row; // Chỉ số hàng của ô  private final int col; // Chỉ số cột của ô  private final int cost; // Chi phí để đi đến ô này từ điểm bắt đầu  private final int heuristic; // Giá trị heuristic (khoảng cách Manhattan đến điểm đích)  private final Node parent; // Node cha  private final int[][] maze; // Mê cung  public **Node**(int row, int col, int cost, Node parent, int[][] maze) {  this.row = row;  this.col = col;  this.cost = cost;  this.parent = parent;  this.maze = maze;  this.heuristic = calculateManhattanDistance();  }  public int getCost() {  return cost;  }  public int getHeuristic() {  return heuristic;  }  public Node getParent() {  return parent;  }  // Tính toán giá trị heuristic theo khoảng cách Manhattan  private int **calculateManhattanDistance**() {  int[] goal = findGoalCell(); // Lấy tọa độ ô đích  // Tính khoảng cách Manhattan bằng cách tính tổng khoảng cách giữa các chiều hàng và cột  return Math.abs(row - goal[0]) + Math.abs(col - goal[1]);  }  // Tìm vị trí ô đích trong mê cung  private int[] **findGoalCell**() {  int[] goal = new int[2];  // Duyệt qua tất cả các ô trong mê cung để tìm ô đích (có giá trị 2)  for (int i = 0; i < maze.length; i++) {  for (int j = 0; j < maze[0].length; j++) {  if (maze[i][j] == 2) {  goal[0] = i;  goal[1] = j;  return goal;  }  }  }  return goal;  }  // So sánh các node để xác định ưu tiên trong hàng đợi ưu tiên  @Override  public int **compareTo** (Node other) {  int totalCost = cost + heuristic;  int otherTotalCost = other.getCost() + other.getHeuristic();  if (totalCost < otherTotalCost) {  return -1;  } else if (totalCost > otherTotalCost) {  return 1;  }  return 0;  }  } |

Phương thức Solve:

|  |
| --- |
| public static List<int[]> **Solve**(int[][] maze, int[] start) {  PriorityQueue<Node> openList = new PriorityQueue<>(); // Hàng đợi ưu tiên chứa các Node cần kiểm tra  Set<String> closedList = new HashSet<>(); // Tập hợp chứa các Node đã kiểm tra  // Tạo Node ban đầu từ ô bắt đầu  Node initialNode = new Node(start[0], start[1], 0, null, maze);  openList.add(initialNode);  // Tiến hành tìm đường đi trong mê cung  while (!openList.isEmpty()) {  Node currentNode = openList.poll(); // Lấy Node đầu tiên trong hàng đợi ưu tiên  // Nếu Node hiện tại là điểm đích, tìm thấy đường đi và trả về đường đi đã xây dựng  if (maze[currentNode.row][currentNode.col] == 2) {  return reconstructPath(currentNode);  }  closedList.add(currentNode.row + "," + currentNode.col); // Đánh dấu Node hiện tại là đã kiểm tra  for (int[] direction : DIRECTIONS) {  int newRow = currentNode.row + direction[0];  int newCol = currentNode.col + direction[1];  // Kiểm tra xem ô mới có hợp lệ để di chuyển vào không  if (isValidCell(newRow, newCol, maze)) {  int newCost = currentNode.getCost() + 1;  Node newNode = new Node(newRow, newCol, newCost, currentNode, maze);  // Nếu ô mới chưa được kiểm tra, thêm vào hàng đợi ưu tiên  if (!closedList.contains(newRow + "," + newCol)) {  openList.add(newNode);  }  }  }  }  return null; // không tim thấy đường đi  } |

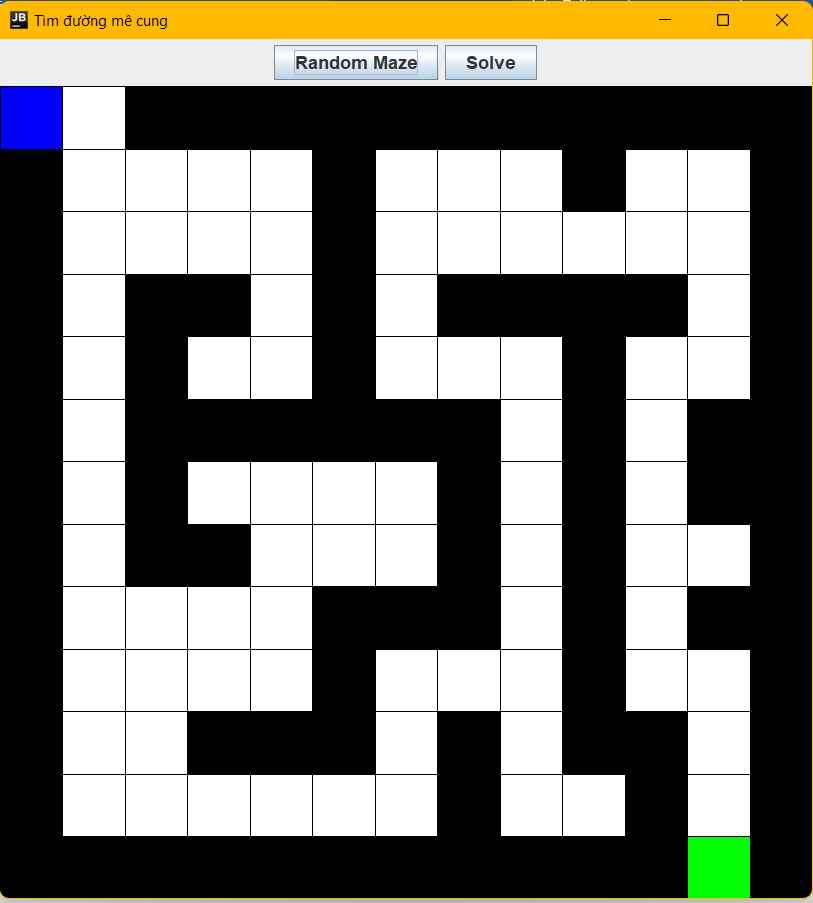
Phương thức reconstructPath:

|  |
| --- |
| private static List<int[]> **reconstructPath**(Node node) {  List<int[]> path = new ArrayList<>();  // Lặp qua các nút từ nút hiện tại đến nút gốc và thêm tọa độ của từng ô vào danh sách path  while (node != null) {  path.add(new int[]{node.row, node.col});  node = node.getParent();  }  Collections.reverse(path); // Đảo ngược danh sách để có thứ tự từ nút gốc đến đích  return path; // Trả về danh sách các tọa độ trên đường đi từ nút đích đến nút gốc.  } |

Phương thức main:

|  |
| --- |
| public static void **main**(String[] args) {  // Khởi tạo mê cung ban đầu và điểm bắt đầu  int[][] maze = {  {0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},  {1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1},  {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1},  {1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1},  {1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1},  {1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1},  {1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1},  {1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1},  {1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1},  {1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1},  {1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1},  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1},  {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1},  };  int[] start = {0, 0};  // Tạo một frame GUI để hiển thị mê cung và đường đi  SwingUtilities.invokeLater(() -> {  JFrame frame = new JFrame("Tìm đường mê cung");  frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  List<int[]> path = Solve(maze, start);  MazePanel mazePanel = new MazePanel(maze, path);  // Tạo nút Random Maze để tạo mê cung ngẫu nhiên  JButton randomMazeButton = new JButton("Random Maze");  randomMazeButton.setFont(new Font("Arial", Font.BOLD, 15));  randomMazeButton.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  int numRows = maze.length;  int numCols = maze[0].length;  int[][] newMaze = generateRandomMaze(numRows, numCols);  List<int[]> newPath = Solve(newMaze, start);  mazePanel.setMaze(newMaze);  mazePanel.setPath(newPath);  frame.revalidate();  frame.repaint();  }  });  // Tạo nút Solve  JButton solveButton = new JButton("Solve");  solveButton.setFont(new Font("Arial", Font.BOLD, 15));  solveButton.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  if (mazePanel.getPath() != null) {  JOptionPane.showMessageDialog(frame, "Tìm thấy đường đi với  "+(mazePanel.getPath().size() - 1) + " bước.");  randomMazeButton.setEnabled(false);  Timer timer = new Timer(300, new ActionListener() {  private int step = 0;  @Override  public void actionPerformed(ActionEvent e) {  if (step < mazePanel.getPath().size()) {  int[] cell = mazePanel.getPath().get(step);  mazePanel.colorCell(cell[0], cell[1], 3);  step++;  } else {  JOptionPane.showMessageDialog(frame, "Đã đến đích!");  ((Timer) e.getSource()).stop();  randomMazeButton.setEnabled(true);  }  }  });  timer.start();  }  else {  JOptionPane.showMessageDialog(frame, "Không tim thấy đường đi đến đích.");  randomMazeButton.setEnabled(true);  }  }  });  frame.setLayout(new BorderLayout());  JPanel buttonPanel = new JPanel();  buttonPanel.add(randomMazeButton);  buttonPanel.add(solveButton);  frame.add(buttonPanel, BorderLayout.NORTH);  frame.add(mazePanel, BorderLayout.CENTER);  frame.pack();  frame.setLocationRelativeTo(null);  frame.setVisible(true);  });  } |

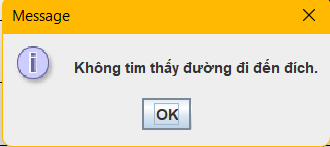
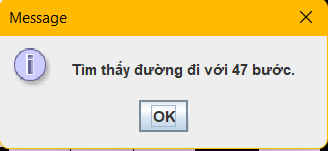
Giao diện chương trình:



Hình 17. Giao diện chương trình khi mới chạy

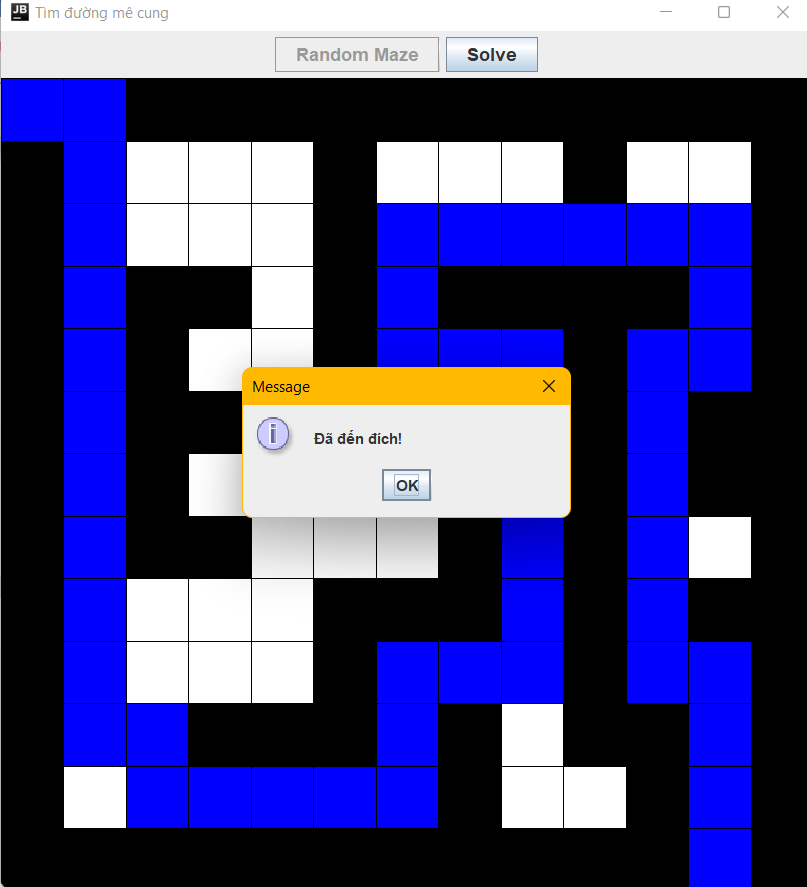
Chương trình gồm có 2 nút “Random Maze” để tạo ngẫu nhiên mê cung nếu cần, “Solve” và một mê cung. Các ô màu trắng đại diện cho đường đi, ô màu đen đại diện cho tường (không thể đi qua), ô màu xanh lá đại diện cho điểm đích và ô màu xanh dương đại diện cho điểm xuất phát.

Khi nhấn “Solve” thì chương trình sẽ thực hiện thuật toán tìm đường đi đến đích của mê cung, hiển thị thông báo tìm thấy cùng với số bước hoặc không tìm thấy đường đi đến đích.



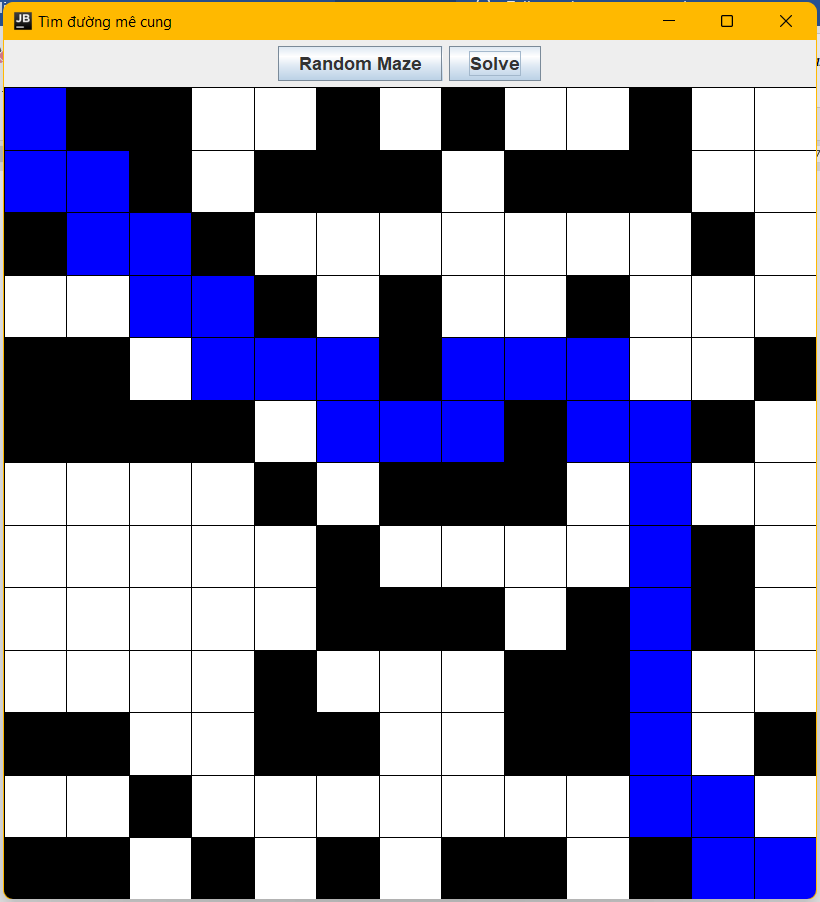
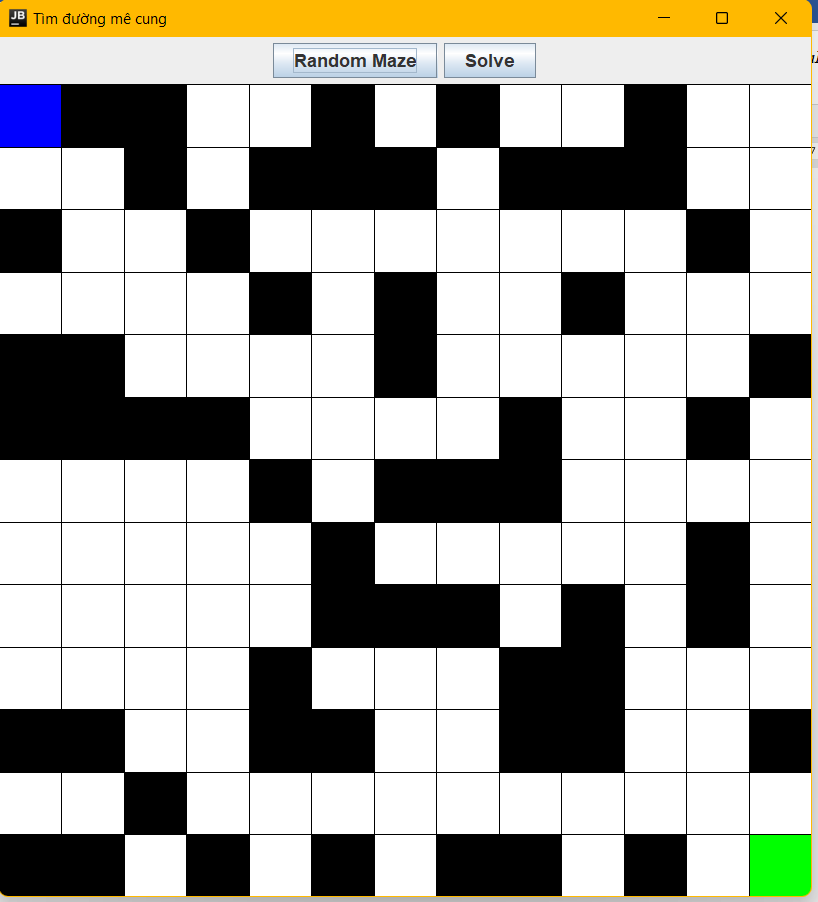
Hình 18. Thông báo cho người dùng

Sau đó chương trình tô màu xanh dương đường đi đến đích và thông báo “Đã đến đích!”. Hiệu ứng chuyển động giống như con rắn đang đi săn mồi.



Hình 19. Giao diện chương trình khi đi đến đích

Một trường hợp ma trận ngẫu nhiên khác:



Hình 20. Mê cung ngẫu nhiên và lời giải đi từ điểm xuất phát đến điểm đích

Kết quả tìm được đường đi ngắn nhất từ điểm xuất phát đến điểm đích trong mê cung.

# Bài 6: Bài toán người nông dân qua sông.

## Giới thiệu

Bài toán người nông dân qua sông, còn được gọi là bài toán "River Crossing Puzzle" hoặc "Fox, Goose, and Bag of Beans", là một bài toán trí tuệ nhân tạo thú vị và cổ điển. Bài toán đề cập đến việc di chuyển một nhóm các đối tượng từ một bờ sông sang bờ sông khác bằng một thuyền nhỏ, với một số quy tắc cần tuân theo để đảm bảo an toàn cho tất cả các đối tượng.

Câu chuyện: một người nông dân đứng bên bờ sông, còn lại ba đối tượng: một con cừu, một bó cải, và một con sói. Người nông dân muốn chở tất cả ba đối tượng qua bờ sông bằng một chiếc thuyền nhỏ. Tuy nhiên, thuyền chỉ có thể chở một đối tượng duy nhất mỗi lần và người nông dân phải ở trong thuyền để lái nó.

Quy tắc:

* Người nông dân có thể di chuyển thuyền và một đối tượng từ bờ này sang bờ khác.
* Nếu người nông dân không ở đó, con sói sẽ ăn thịt cừu.
* Nếu cừu không có sự chăm sóc của người nông dân, nó sẽ ăn bó cải.

Mục tiêu của bài toán là tìm cách chuyển tất cả các đối tượng từ bờ sông này sang bờ sông khác mà vẫn tuân theo các quy tắc trên mà không có bất kỳ sự vi phạm nào. Bài toán thử thách người giải quyết trong việc tìm ra một dãy các bước di chuyển hợp lý để đảm bảo an toàn cho tất cả các đối tượng.

Bài toán này thường được sử dụng để minh họa các khái niệm trong lĩnh vực trí tuệ nhân tạo như tìm kiếm, quyết định và giải quyết vấn đề. Ngoài ra, nó cũng có thể được dùng để giải thích các khái niệm về quản lý tài nguyên, điều kiện ràng buộc và chiến lược trong việc giải quyết các vấn đề phức tạp.

## Thuật toán tìm kiếm theo chiều rộng (BFS)

* 1. Giới thiệu

Thuật toán tìm kiếm theo chiều rộng (Breadth-First Search - BFS) là một trong những thuật toán tìm kiếm cơ bản và quan trọng trong lĩnh vực trí tuệ nhân tạo và khoa học máy tính. BFS được sử dụng để tìm kiếm và khám phá đồ thị hay cấu trúc dữ liệu có cấu trúc tương tự.

* 1. Nguyên tắc hoạt động:

Thuật toán BFS hoạt động bằng cách duyệt qua các đỉnh (nodes) của đồ thị hay cấu trúc dữ liệu từ đỉnh gốc (root) và tiến hành tìm kiếm theo chiều rộng, tức là trước tiên duyệt qua các đỉnh kề với đỉnh gốc, sau đó duyệt qua các đỉnh kề với các đỉnh đã được duyệt trước đó, và tiếp tục quá trình này cho đến khi tìm ra đỉnh mục tiêu hoặc duyệt hết toàn bộ đỉnh.

Cách thức hoạt động:

Bắt đầu từ đỉnh gốc (root), đánh dấu nó là đã được thăm.

Đưa đỉnh gốc vào hàng đợi (queue).

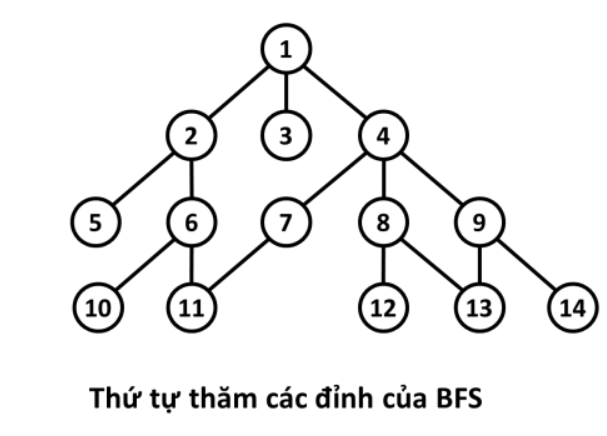
Lặp cho đến khi hàng đợi trống:

a. Lấy đỉnh đầu tiên ra khỏi hàng đợi.

b. Duyệt qua tất cả các đỉnh kề với đỉnh đang xét mà chưa được thăm:

Đánh dấu các đỉnh kề là đã thăm và đưa chúng vào hàng đợi.

c. Lặp lại bước a và b cho đến khi tất cả các đỉnh đã được duyệt qua.



* 1. Ưu điểm, nhược điểm và ứng dụng

*Ưu điểm:*

* BFS luôn tìm được đường đi ngắn nhất từ đỉnh gốc đến các đỉnh còn lại.
* Được sử dụng trong trường hợp tìm kiếm đường đi ngắn nhất.
* Khám phá toàn bộ cấu trúc dữ liệu.

*Nhược điểm:*

Sử dụng nhiều bộ nhớ để lưu trữ hàng đợi, vì vậy không phù hợp với các cấu trúc dữ liệu lớn.

*Ứng dụng:*

* Tìm kiếm đường đi ngắn nhất trong các bài toán tìm đường, ví dụ như trong bài toán tìm đường đi giữa hai địa điểm trên bản đồ.
* Kiểm tra tính kết nối của đồ thị.
* Tìm kiếm trong các bài toán trạng thái (state space search) như giải bài toán người nông dân qua sông.

Thuật toán BFS là một trong những khái niệm cơ bản trong khoa học máy tính và được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau.

## Cài đặt chương trình

Sử dụng ngôn ngữ lập trình java

Đầu vào: trạng thái ban đầu của bài toán là người nông dân, sói, cừu và bắp cải ở bên bờ trái, bờ phải trống.

Đầu ra: lời giải cụ thể từ trạng thái ban đầu đến trạng thái kết thúc, bao gồm các bước di chuyển và trạng thái tương ứng tại mỗi bước. Ở trạng thái kết thúc thì người nông dân, sói, cừu, bắp cải qua được bờ bên phải mà vẫn tuân thủ các quy định đã được đặt ra của bài toán.

Code chương trình:

Class RiverCrossingBFS:



Class State:

|  |
| --- |
| private static class **State** {  Set<String> leftBank; // Tập hợp chứa các đối tượng ở bờ trái.  Set<String> rightBank; // Tập hợp chứa các đối tượng ở bờ phải.  boolean farmerOnLeft; // Biểu thị vị trí của nông dân, true nếu nông dân ở bờ trái và false nếu nông dân ở bờ phải  State parent; // Trạng thái cha của trạng thái hiện tại  public State(Set<String> leftBank, Set<String> rightBank, boolean farmerOnLeft, State parent) {  this.leftBank = new HashSet<>(leftBank);  this.rightBank = new HashSet<>(rightBank);  this.farmerOnLeft = farmerOnLeft;  this.parent = parent;  }  // Kiểm tra xem trạng thái có hợp lệ không, tức là không có sự kết hợp nguy hiểm nào (sói ăn cừu, cừu ăn bắp cải khi không có nông dân).  public boolean **isValid**() {  return !(hasDangerousCombination(leftBank) || hasDangerousCombination(rightBank));  }  // Kiểm tra xem tập hợp các đối tượng có sự kết hợp nguy hiểm nào không.  private boolean **hasDangerousCombination**(Set<String> bank) {  return (bank.contains("W") && bank.contains("S") && !bank.contains("F"))  || (bank.contains("S") && bank.contains("C") && !bank.contains("F"));  }  // Kiểm tra xem trạng thái hiện tại có phải trạng thái kết thúc (tất cả đối tượng ở bờ phải, không có gì ở bờ trái) hay không.  public boolean **isFinalState**() {  return leftBank.isEmpty() && rightBank.containsAll(Arrays.asList(OCCUPANTS));  }  // Phương thức tạo trạng thái mới sau khi thực hiện bước di chuyển  public State **makeMove**(String move) {  Set<String> newLeftBank = new HashSet<>(leftBank);  Set<String> newRightBank = new HashSet<>(rightBank);  // Đảo chiều vị trí của người nông dân sau mỗi bước di chuyển  boolean newFarmerOnLeft = !farmerOnLeft;  // Duyệt qua từng ký tự trong chuỗi bước di chuyển  for (char c : move.toCharArray()) {  String occupant = String.valueOf(c);  // Nếu người nông dân đang ở bờ trái  if (farmerOnLeft) {  newLeftBank.remove(occupant); // Di chuyển người và vật thể ra khỏi bờ trái  newRightBank.add(occupant); // Đặt người và vật thể vào bờ phải  } else {  newRightBank.remove(occupant);  newLeftBank.add(occupant);  }  }  return new State(newLeftBank, newRightBank, newFarmerOnLeft, this);  }  @Override  public String **toString**() {  StringBuilder left = new StringBuilder();  for (var o : leftBank) {  left.append(occupantToString(o));  }  StringBuilder right = new StringBuilder();  for (var o : rightBank) {  right.append(occupantToString(o));  }  // Trả về chuỗi mô tả trạng thái của cả hai bờ  return "Left: " + left + "\nRight: " + right ;  }  } |

Phương thức solveRiverCrossing :

|  |
| --- |
| public static void **solveRiverCrossing**() {  // Tạo trạng thái ban đầu với danh sách vật thể ở bờ trái và bờ phải  Set<String> initialLeftBank = new HashSet<>(Arrays.asList(OCCUPANTS));  Set<String> initialRightBank = new HashSet<>();  State initialState = new State(initialLeftBank, initialRightBank, true, null);  // Tạo hàng đợi để thực hiện tìm kiếm theo chiều rộng  Queue<State> queue = new LinkedList<>();  queue.offer(initialState);  // Thực hiện tìm kiếm theo chiều rộng  while (!queue.isEmpty()) {  State currentState = queue.poll(); // Lấy trạng thái hiện tại ra khỏi hàng đợi để kiểm tra  // Kiểm tra nếu trạng thái hiện tại là trạng thái cuối cùng  if (currentState.isFinalState()) {  // In ra lời giải  printSolution(currentState);  break;  }  // Duyệt qua các bước di chuyển và tạo các trạng thái kế tiếp  for (String move : MOVES) {  State nextState = currentState.makeMove(move);  // Kiểm tra xem trạng thái kế tiếp có hợp lệ không  if (nextState.isValid()) {  // Thêm trạng thái kế tiếp vào hàng đợi để kiểm tra sau  queue.offer(nextState);  }  }  }  } |

Phương thức printSolution:

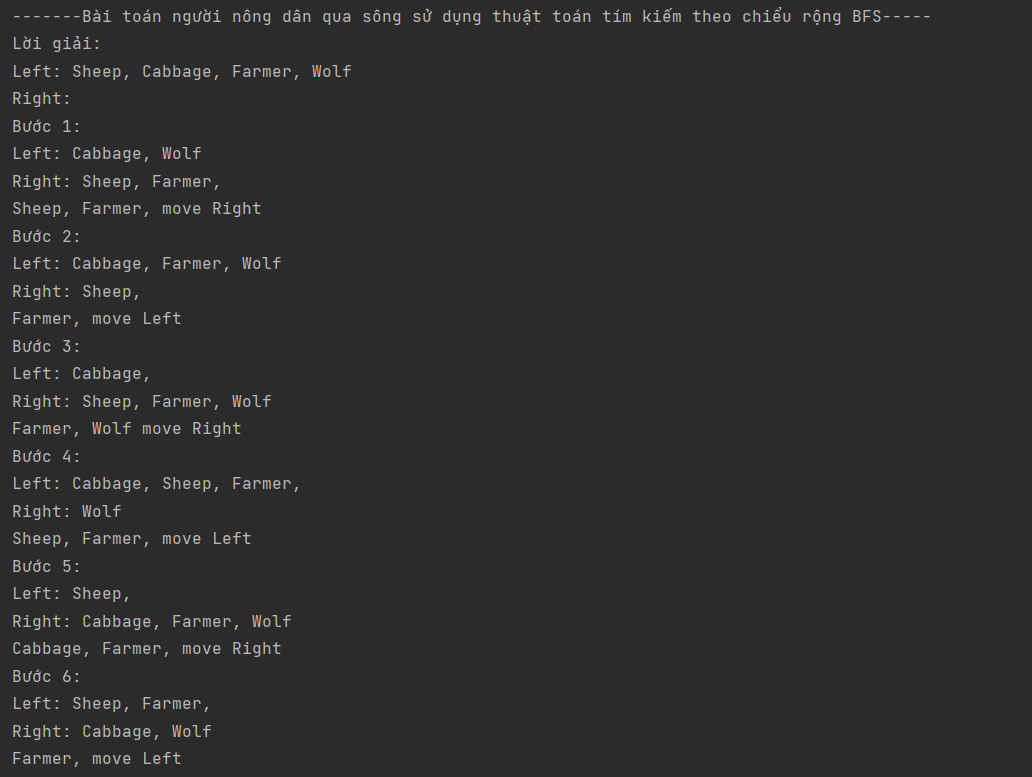
|  |
| --- |
| public static void **printSolution**(State state) {  // Tạo danh sách các trạng thái trong chuỗi lời giải  List<State> solutionPath = new ArrayList<>();  while (state != null) {  solutionPath.add(state);  state = state.parent;  }  Collections.reverse(solutionPath); // Đảo ngược danh sách để bắt đầu từ trạng thái ban đầu  System.out.println("Lời giải:");  for (int i = 0; i < solutionPath.size(); i++) {  State step = solutionPath.get(i);  if (i == 0) {  System.out.println(step);  } else {  System.out.println("Bước " + i + ":");  System.out.println(step); // In ra trạng thái hiện tại  String move = findMove(solutionPath.get(i - 1),  solutionPath.get(i));  System.out.println(move); // In ra bước di chuyển  }  }  } |

Phương thức occupantToString:

|  |
| --- |
| public static String **occupantToString**(String occupant) {  switch (occupant) {  case "S":  return "Sheep, ";  case "C":  return "Cabbage, ";  case "F":  return "Farmer, ";  case "W":  return "Wolf ";  default:  return "";  }  } |

Giao diện chương trình:

Kết quả khi chạy:





Hình 21. Kết quả khi chạy chương trình

Chương trình hiển thị từng bước và mỗi bước gồm có:

* Trạng thái của bờ trái.
* Trạng thái của bờ phải.
* Bước di chuyển trong trạng thái đó.

Cuối cùng đưa được tất cả các đối tượng: người nông dân, sói, cừu và bắp cải qua được bờ bên kia.

# KẾT LUẬN

## Kết quả đạt được

Báo cáo này là kết quả của quá trình nghiên cứu và học tập về trí tuệ nhân tạo, thông qua những buổi học trên lớp và thời gian tự học tại nhà. Trong bài báo cáo này, em đã đạt được một số kết quả quan trọng:

* Hiểu rõ và tiếp thu được những kiến thức cơ bản về trí tuệ nhân tạo.
* Nắm vững các thuật toán và phương pháp quan trọng trong lĩnh vực trí tuệ nhân tạo.
* Giải quyết được một số bài toán thực tế liên quan đến trí tuệ nhân tạo.

Tuy nhiên, do hạn chế về thời gian và khả năng, bài báo cáo vẫn còn một số khuyết điểm cần được cải thiện. Các bài toán đã được giải quyết và thiết kế vẫn còn một số hạn chế, cần được cải tiến một cách hiệu quả hơn.

## Đề xuất

Trong quá trình hoàn thành bài báo cáo với thời gian và năng lực có hạn nên bài báo cáo không tránh khỏi những sai sót, em rất mong nhận được sự góp ý, nhận xét, đánh giá của Thầy. Những ý kiến này sẽ giúp em tiến xa hơn trong việc tìm hiểu, phân tích và giải quyết các bài toán liên quan đến trí tuệ nhân tạo, từ đó hoàn thiện kỹ năng và kiến thức của mình. Em sẽ tiếp tục học hỏi thêm nhiều thuật toán, thuật giải về trí tuệ nhân tạo. Vận dụng và giải quyết các bài toán thực tế dựa trên những kiến thức đã học.

Em xin chân thành cảm ơn.