**TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG**

**A blue and white logo with a book and a square cap

Description automatically generatedKHOA CÔNG NGHỆ THÔNG TIN**

**TIỂU LUẬN MÔN HỌC**

**CẤU TRÚC RỜI RẠC**

**Đồ án**

GVHD: Nguyễn Thị Thùy Trang

Thực hiện : Vũ Đình Ân–2001220273

Trần Anh Vũ–2001225912

Nguyễn Vũ Duy–2033220716

Nguyễn Thanh Bình–2033210602

Lê Phước Trường Duy–2033210030

**TP.Hồ Chí Minh – 11/2023**

**Mục Lục**

[1. Giới thiệu thuật toán 3](#_Toc151931713)

[1.1 Thuật toán Dijkstra 3](#_Toc151931714)

[1.2 Thuật toán Floyd 4](#_Toc151931715)

[2.Giới thiệu code về thuật toán Dijkstra và Floyd 4](#_Toc151931716)

[2.1. Khai Báo và Namespace 4](#_Toc151931717)

[2.2. Khởi Tạo và Biến Quan Trọng 5](#_Toc151931718)

[2.3 Phương Thức Khởi Tạo và Khởi Tạo Ma Trận Trọng Số 6](#_Toc151931719)

[2.4 Cập Nhật Ma Trận Trọng Số Từ ListView 7](#_Toc151931720)

[2.5 Hiển Thị Ma Trận Trọng Số Lên ListView 7](#_Toc151931721)

[2.6 Kiểm Tra Chỉ Số Hợp Lệ và Cập Nhật Ma Trận Trọng Số 7](#_Toc151931722)

[2.7 Cập Nhật ComboBox và Vẽ Điểm, Đường 11](#_Toc151931723)

[2.8 Xử Lý Làm Mới và Lưu Mở Tệp Tin 12](#_Toc151931724)

[2.9 Thuật Toán Dijkstra 14](#_Toc151931725)

[2.10 Các Phương Thức Hỗ Trợ Thuật Toán Dijkstra 14](#_Toc151931726)

[2.11 Các Phương Thức Vẽ Lại và Tìm Điểm Trên PictureBox 15](#_Toc151931727)

[2.12. Phương Thức Làm Mới Ma Trận Trọng Số 17](#_Toc151931728)

[2.13 Xử Lý Sự Kiện Mở và Lưu Tệp Tin 17](#_Toc151931729)

[2.14 Xử Lý Sự Kiện Đóng Ứng Dụng và Hiển Thị Thông Tin Nhóm 18](#_Toc151931730)

[2.15 Xử Lý Sự Kiện Thuật Toán Dijkstra 18](#_Toc151931731)

[2.16 Các Phương Thức Hỗ Trợ Thuật Toán Dijkstra 19](#_Toc151931732)

[2.17 Xử Lý Sự Kiện Tìm Đường Ngắn Nhất - Dijkstra 20](#_Toc151931733)

[2.18 Các Phương Thức Hỗ Trợ Thuật Toán Dijkstra 22](#_Toc151931734)

[2.19 Sự kiện btnF\_Click:\*\* 28](#_Toc151931735)

[2.20 Phương thức FloydWarshall:\*\* 29](#_Toc151931736)

[2.21 Phương thức DisplayFloydWarshallResults:\*\* 30](#_Toc151931737)

[2.22 Trực quan hóa Đồ thị:\*\* 32](#_Toc151931738)

[2.23 Demo thuật toán trên c# 34](#_Toc151931739)

[2.23.1 Demo Thuật toán Dijkstra c# 34](#_Toc151931740)

[2.23.2 Demo thuật toán floy: 35](#_Toc151931741)

# 1. Giới thiệu thuật toán

## 1.1 Thuật toán Dijkstra

Thuật toán Dijkstra là một thuật toán tìm đường ngắn nhất từ ​​​​một sản phẩm xuất ra đến tất cả các hạng còn lại trong sơ đồ có số không âm. Được đặt tên theo nhà toán học và nhà máy tính Hà Lan Edsger Dijkstra, thuật toán này rất phổ biến và được sử dụng trong nhiều ứng dụng thực tế, ý tưởng như mạng định tuyến tuyến tính.

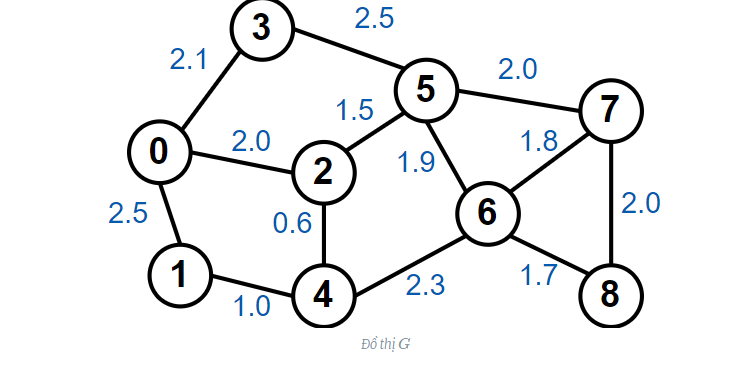
Thuật toán hoạt động bằng cách duy trì một tập hợp các đỉnh trong đó ta đã biết chắc chắn đường đi ngắn nhất. Mỗi bước, thuật toán sẽ chọn ra một đỉnh u mà chắc chắn sẽ không thể tối ưu hơn nữa, sau đó tiến hành tối ưu các đỉnh v khác dựa trên các cạnh (u,v) đi ra từ đỉnh u. Sau N bước, tất cả các đỉnh đều sẽ được chọn, và mọi đường đi tìm được sẽ là ngắn nhất.

Cụ thể hơn, thuật toán sẽ duy trì đường đi ngắn nhất đến tất cả các đỉnh. Ở mỗi bước, chọn đường đi S→u có tổng trọng số nhỏ nhất trong tất cả các đường đi đang được duy trì. Sau đó tiến hành tối ưu các đường đi S→v bằng cách thử kéo dài thành S→u→v như đã mô tả trên.

**Ví Dụ**

Để dễ dàng hiểu ý tưởng của thuật toán. Chúng ta cùng xem ví dụ với đồ thị vô hướng *G*. Thuật toán Dijkstra sẽ tìm khoảng cách từ đỉnh gốc 0 tới tất cả các đỉnh còn lại trong đồ thị *G*.

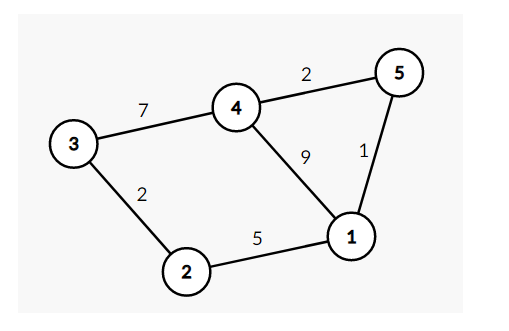
A table with numbers and symbols

Description automatically generated

## 1.2 Thuật toán Floyd

[Thuật toán](https://dothanhspyb.com/khai-niem-mo-ta-thuat-toan-do-phuc-tap-cua-thuat-toan/) Floyd-Warshall là một [thuật toán](https://dothanhspyb.com/khai-niem-mo-ta-thuat-toan-do-phuc-tap-cua-thuat-toan/) giải quyết bài toán tìm đường đi ngắn nhất (shortest path) giữa tất cả các cặp đỉnh trong đồ thị có hướng hoặc vô hướng, có thể có trọng số âm nhưng không chứa chu trình âm.

Ý tưởng của [thuật toán](https://dothanhspyb.com/khai-niem-mo-ta-thuat-toan-do-phuc-tap-cua-thuat-toan/) là sử dụng phương pháp cập nhật giá trị của ma trận trọng số sao cho tại mỗi bước, ta sẽ xem xét một đỉnh mới và cập nhật giá trị ngắn nhất của đường đi giữa các cặp đỉnh thông qua đỉnh này. Cụ thể, ta sử dụng một ma trận kích thước VxV (V là số đỉnh của đồ thị) để lưu trữ giá trị ngắn nhất hiện tại giữa mỗi cặp đỉnh. [Thuật toán](https://dothanhspyb.com/khai-niem-mo-ta-thuat-toan-do-phuc-tap-cua-thuat-toan/) sẽ thực hiện V lần lặp, mỗi lần lặp đều sẽ xem xét một đỉnh mới, và cập nhật giá trị ngắn nhất cho tất cả các cặp đỉnh thông qua đỉnh này.

Ví dụ Cho đồ thị vô hướng *G*, tìm đường đi ngắn nhất giữa các cặp đỉnh trong đồ thị trên.

# 2.Giới thiệu code về thuật toán Dijkstra và Floyd

## 2.1. Khai Báo và Namespace

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Nhom8\_CTRR

{

//...

}

\*\*using:\*\* Khai báo các thư viện và namespace sử dụng trong ứng dụng.

## 2.2. Khởi Tạo và Biến Quan Trọng

private int[,] weightMatrix;

private List<Point> points = new List<Point>();

private Dictionary<(int, int), int> weights = new Dictionary<(int, int), int>();

private string mode = "";

private int selected = -1;

\*\*weightMatrix:\*\* Lưu trữ ma trận trọng số của các đỉnh.

\*\*points:\*\* Danh sách các điểm trên hình vẽ.

\*\*weights:\*\* Lưu trữ trọng số giữa các cặp đỉnh.

\*\*mode:\*\* Chế độ hiện tại của ứng dụng (Thêm điểm, Xóa, Vẽ đường, Di chuyển).

\*\*selected:\*\* Chỉ số của điểm được chọn hoặc -1 nếu không có điểm nào được chọn.

## 2.3 Phương Thức Khởi Tạo và Khởi Tạo Ma Trận Trọng Số

public frmMain()

{

InitializeComponent();

InitializeWeightMatrix();

}

private void InitializeWeightMatrix()

{

const int defaultRowCount = 3;

const int defaultColumnCount = 3;

weightMatrix = new int[defaultRowCount, defaultColumnCount];

}

\*\*frmMain():\*\* Phương thức khởi tạo của lớp chính, khởi tạo các thành phần và ma trận trọng số.

\*\*InitializeWeightMatrix():\*\* Khởi tạo ma trận trọng số với số hàng và cột mặc định.

## 2.4 Cập Nhật Ma Trận Trọng Số Từ ListView

private void lvMaTran\_SelectedIndexChanged(object sender, EventArgs e)

{

// Xử lý sự kiện khi một mục trong ListView được chọn.

// Cập nhật ma trận trọng số từ giá trị trong ListView.

}

\*\*lvMaTran\_SelectedIndexChanged():\*\* Xử lý sự kiện khi một mục trong ListView (lvMaTran) được chọn. Cập nhật ma trận trọng số từ giá trị trong ListView.

## 2.5 Hiển Thị Ma Trận Trọng Số Lên ListView

private void DisplayWeightMatrix()

{

// Hiển thị ma trận trọng số lên ListView.

}

\*\*DisplayWeightMatrix():\*\* Hiển thị ma trận trọng số lên ListView.

## 2.6 Kiểm Tra Chỉ Số Hợp Lệ và Cập Nhật Ma Trận Trọng Số

private bool IsValidIndex(int index)

{

return index >= 0 && index < weightMatrix.GetLength(0);

}

private void UpdateWeightMatrix(Point p1, Point p2, int weight)

{

//...

}

private void ResizeWeightMatrix(int newSize)

{

//...

}

\*\*IsValidIndex():\*\* Kiểm tra xem một chỉ số có hợp lệ trong mảng hay không.

\*\*UpdateWeightMatrix():\*\* Cập nhật ma trận trọng số khi thêm đỉnh hoặc vẽ đường.

\*\*ResizeWeightMatrix():\*\* Thay đổi kích thước ma trận trọng số nếu cần.

**2.7.1Thêm Điểm, Xóa Điểm, Vẽ Đường và Di Chuyển**

private void btnPoint\_Click(object sender, EventArgs e)

{

mode = "Point";

}

private void btnDelete\_Click(object sender, EventArgs e)

{

mode = "Delete";

}

private void btnLine\_Click(object sender, EventArgs e)

{

mode = "Line";

selected = -1;

}

private void btnMove\_Click(object sender, EventArgs e)

{

mode = "Move";

}

private void pictureBox1\_MouseClick(object sender, MouseEventArgs e)

{

// Xử lý sự kiện khi click vào pictureBox.

// Thêm điểm, xóa điểm, vẽ đường hoặc di chuyển điểm tùy thuộc vào chế độ hiện tại.

}

private void pictureBox1\_MouseMove(object sender, MouseEventArgs e)

{

// Xử lý sự kiện khi di chuyển chuột trên pictureBox.

// Di chuyển điểm nếu đang ở chế độ di chuyển.

}

private void pictureBox1\_MouseUp(object sender, MouseEventArgs e)

{

// Xử lý sự kiện khi nhả chuột trên pictureBox.

// Kết thúc chế độ di chuyển.

}

private void pictureBox1\_MouseDown(object sender, MouseEventArgs e)

{

// Xử lý sự kiện khi nhấn chuột trên pictureBox.

// Bắt đầu di chuyển nếu đang ở chế độ di chuyển.

}

Các phương thức và sự kiện liên quan đến thêm điểm, xóa điểm, vẽ đường và di chuyển điểm trên pictureBox.

## 2.7 .2Cập Nhật ComboBox và Vẽ Điểm, Đường

private void UpdateComboBoxes()

{

// Cập nhật ComboBox chứa danh sách các điểm.

}

private void DrawPoint(Point p)

{

// Vẽ điểm trên pictureBox.

}

private void DrawLine(Point p1, Point p2, int weight)

{

// Vẽ đường và hiển thị trọng số giữa hai điểm trên pictureBox.

}

private void EraseBorder(Point p)

{

// Xóa viền xung quanh điểm.

}

private void DrawBorder(Point p, Color color)

{

// Vẽ viền xung quanh điểm để đánh dấu được chọn.

}

\*\*UpdateComboBoxes():\*\* Cập nhật ComboBox với danh sách các điểm.

\*\*DrawPoint():\*\* Vẽ điểm trên pictureBox.

\*\*DrawLine():\*\* Vẽ đường và hiển thị trọng số giữa hai điểm trên pictureBox.

\*\*EraseBorder():\*\* Xóa viền xung quanh điểm để bỏ đánh dấu đã chọn.

\*\*DrawBorder():\*\* Vẽ viền xung quanh điểm để đánh dấu đã chọn.

## 2.8 Xử Lý Làm Mới và Lưu Mở Tệp Tin

private void btnReset\_Click(object sender, EventArgs e)

{

// Xử lý sự kiện làm mới ứng dụng.

}

private void openTSM\_Click(object sender, EventArgs e)

{

// Xử lý sự kiện mở tệp tin.

}

private void saveTSM\_Click(object sender, EventArgs e)

{

// Xử lý sự kiện lưu tệp tin.

}

private void closeTSM\_Click(object sender, EventArgs e)

{

// Xử lý sự kiện đóng ứng dụng.

}

private void InfoTSM\_Click(object sender, EventArgs e)

{

// Xử lý sự kiện hiển thị thông tin nhóm.

}

\*\*btnReset\_Click():\*\* Xử lý sự kiện làm mới ứng dụng.

\*\*openTSM\_Click():\*\* Xử lý sự kiện mở tệp tin.

\*\*saveTSM\_Click():\*\* Xử lý sự kiện lưu tệp tin.

\*\*closeTSM\_Click():\*\* Xử lý sự kiện đóng ứng dụng.

\*\*InfoTSM\_Click():\*\* Xử lý sự kiện hiển thị thông tin nhóm.

## 2.9 Thuật Toán Dijkstra

private void btnD\_Click(object sender, EventArgs e)

{

// Xử lý sự kiện tìm đường đi ngắn nhất bằng thuật toán Dijkstra.

}

\*\*btnD\_Click():\*\* Xử lý sự kiện tìm đường đi ngắn nhất bằng thuật toán Dijkstra.

## 2.10 Các Phương Thức Hỗ Trợ Thuật Toán Dijkstra

private bool AreVerticesConnected(int startVertex, int endVertex)

{

// Kiểm tra xem hai đỉnh có được nối với nhau hay không.

}

private int Dijkstra(int start, int end, out Dictionary<int, int> previousVertices)

{

// Thuật toán Dijkstra để tìm đường đi ngắn nhất.

}

private string ConstructPath(Dictionary<int, int> previousVertices, int startVertex, int endVertex)

{

// Xây dựng đường đi từ đỉnh xuất phát đến đỉnh kết thúc.

}

private void DisplayDijkstraPath(Dictionary<int, int> previousVertices, int startVertex, int endVertex, int totalDistance)

{

// Hiển thị đường đi và độ dài trong TextBox.

}

Các phương thức hỗ trợ thuật toán Dijkstra như kiểm tra kết nối giữa các đỉnh, thực hiện thuật toán Dijkstra, xây dựng đường đi, và hiển thị đường đi và độ dài.

## 2.11 Các Phương Thức Vẽ Lại và Tìm Điểm Trên PictureBox

private void RedrawPictureBox()

{

// Vẽ lại toàn bộ PictureBox sau khi có thay đổi trong danh sách điểm và trọng số.

}

private int FindPoint(int x, int y)

{

// Tìm chỉ số của điểm dựa trên tọa độ x, y.

}

private void DrawBorder(Point p, Color color)

{

// Vẽ viền xung quanh điểm để đánh dấu đã chọn.

}

private void RemoveWeights(int index)

{

// Xóa các trọng số liên quan đến điểm khi điểm bị xóa.

}

\*\*RedrawPictureBox():\*\* Vẽ lại toàn bộ PictureBox sau khi có thay đổi trong danh sách điểm và trọng số.

\*\*FindPoint():\*\* Tìm chỉ số của điểm dựa trên tọa độ x, y.

\*\*DrawBorder():\*\* Vẽ viền xung quanh điểm để đánh dấu đã chọn.

\*\*RemoveWeights():\*\* Xóa các trọng số liên quan đến điểm khi điểm bị xóa.

## 2.12. Phương Thức Làm Mới Ma Trận Trọng Số

private void ClearWeightMatrix()

{

// Làm mới ma trận trọng số khi có thay đổi trong danh sách điểm.

}

\*\*ClearWeightMatrix():\*\* Làm mới ma trận trọng số khi có thay đổi trong danh sách điểm.

## 2.13 Xử Lý Sự Kiện Mở và Lưu Tệp Tin

private void openTSM\_Click(object sender, EventArgs e)

{

// Xử lý sự kiện mở tệp tin.

}

private void saveTSM\_Click(object sender, EventArgs e)

{

// Xử lý sự kiện lưu tệp tin.

}

\*\*openTSM\_Click():\*\* Xử lý sự kiện mở tệp tin.

\*\*saveTSM\_Click():\*\* Xử lý sự kiện lưu tệp tin.

## 2.14 Xử Lý Sự Kiện Đóng Ứng Dụng và Hiển Thị Thông Tin Nhóm

private void closeTSM\_Click(object sender, EventArgs e)

{

// Xử lý sự kiện đóng ứng dụng.

}

private void InfoTSM\_Click(object sender, EventArgs e)

{

// Xử lý sự kiện hiển thị thông tin nhóm.

}

\*\*closeTSM\_Click():\*\* Xử lý sự kiện đóng ứng dụng.

\*\*InfoTSM\_Click():\*\* Xử lý sự kiện hiển thị thông tin nhóm.

## 2.15 Xử Lý Sự Kiện Thuật Toán Dijkstra

private void btnD\_Click(object sender, EventArgs e)

{

// Xử lý sự kiện tìm đường đi ngắn nhất bằng thuật toán Dijkstra.

}

\*\*btnD\_Click():\*\* Xử lý sự kiện tìm đường đi ngắn nhất bằng thuật toán Dijkstra.

## 2.16 Các Phương Thức Hỗ Trợ Thuật Toán Dijkstra

private bool AreVerticesConnected(int startVertex, int endVertex)

{

// Kiểm tra xem hai đỉnh có được nối với nhau hay không.

}

private int Dijkstra(int start, int end, out Dictionary<int, int> previousVertices)

{

// Thuật toán Dijkstra để tìm đường đi ngắn nhất.

}

private string ConstructPath(Dictionary<int, int> previousVertices, int startVertex, int endVertex)

{

// Xây dựng đường đi từ đỉnh xuất phát đến đỉnh kết thúc.

}

private void DisplayDijkstraPath(Dictionary<int, int> previousVertices, int startVertex, int endVertex, int totalDistance)

{

// Hiển thị đường đi và độ dài trong TextBox.

}

Các phương thức hỗ trợ thuật toán Dijkstra như kiểm tra kết nối giữa các đỉnh, thực hiện thuật toán Dijkstra, xây dựng đường đi, và hiển thị đường đi và độ dài.

## 2.17 Xử Lý Sự Kiện Tìm Đường Ngắn Nhất - Dijkstra

private void btnD\_Click(object sender, EventArgs e)

{

// Lấy chỉ số của điểm xuất phát và điểm kết thúc từ ComboBoxes.

int startIndex = cbBD.SelectedIndex;

int endIndex = cbKT.SelectedIndex;

// Kiểm tra xem đã chọn đúng điểm xuất phát và kết thúc chưa.

if (startIndex == -1 || endIndex == -1)

{

MessageBox.Show("Vui lòng chọn cả điểm xuất phát và điểm kết thúc.", "Lỗi", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

// Kiểm tra xem có đường đi giữa hai điểm hay không.

if (!AreVerticesConnected(startIndex, endIndex))

{

MessageBox.Show("Hai điểm không có đường đi giữa chúng.", "Lỗi", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

// Thực hiện thuật toán Dijkstra để tìm đường đi ngắn nhất.

Dictionary<int, int> previousVertices;

int totalDistance = Dijkstra(startIndex, endIndex, out previousVertices);

// Hiển thị đường đi và độ dài trong TextBox.

DisplayDijkstraPath(previousVertices, startIndex, endIndex, totalDistance);

}

\*\*btnD\_Click():\*\* Xử lý sự kiện khi người dùng nhấn nút để tìm đường đi ngắn nhất bằng thuật toán Dijkstra.

## 2.18 Các Phương Thức Hỗ Trợ Thuật Toán Dijkstra

private bool AreVerticesConnected(int startVertex, int endVertex)

{

// Kiểm tra xem có đường đi giữa hai đỉnh không.

HashSet<int> visited = new HashSet<int>();

Queue<int> queue = new Queue<int>();

queue.Enqueue(startVertex);

while (queue.Count > 0)

{

int currentVertex = queue.Dequeue();

visited.Add(currentVertex);

for (int i = 0; i < weightMatrix.GetLength(0); i++)

{

if (!visited.Contains(i) && weightMatrix[currentVertex, i] > 0)

{

queue.Enqueue(i);

// Nếu đỉnh hiện tại là đỉnh kết thúc, có đường đi giữa startVertex và endVertex.

if (i == endVertex)

{

return true;

}

}

}

}

return false;

}

private int Dijkstra(int start, int end, out Dictionary<int, int> previousVertices)

{

// Thuật toán Dijkstra để tìm đường đi ngắn nhất.

Dictionary<int, int> distances = new Dictionary<int, int>();

previousVertices = new Dictionary<int, int>();

HashSet<int> visited = new HashSet<int>();

for (int i = 0; i < weightMatrix.GetLength(0); i++)

{

distances[i] = int.MaxValue;

}

distances[start] = 0;

while (visited.Count < weightMatrix.GetLength(0))

{

int current = -1;

int minDistance = int.MaxValue;

for (int i = 0; i < weightMatrix.GetLength(0); i++)

{

if (!visited.Contains(i) && distances[i] < minDistance)

{

current = i;

minDistance = distances[i];

}

}

if (current == -1)

{

break;

}

visited.Add(current);

for (int i = 0; i < weightMatrix.GetLength(0); i++)

{

if (!visited.Contains(i) && weightMatrix[current, i] > 0)

{

int newDistance = distances[current] + weightMatrix[current, i];

if (newDistance < distances[i])

{

distances[i] = newDistance;

previousVertices[i] = current; // Lưu đỉnh trước đó

}

}

}

}

return distances[end];

}

private string ConstructPath(Dictionary<int, int> previousVertices, int startVertex, int endVertex)

{

// Xây dựng đường đi từ đỉnh xuất phát đến đỉnh kết thúc.

List<int> path = new List<int>();

int currentVertex = endVertex;

while (currentVertex != startVertex)

{

path.Add(currentVertex + 1); // Chuyển từ chỉ số bắt đầu từ 1 sang chỉ số bắt đầu từ 0

if (previousVertices.ContainsKey(currentVertex))

{

currentVertex = previousVertices[currentVertex];

}

else

{

// Không tìm thấy đỉnh trước đó, thoát vòng lặp để tránh lỗi

break;

}

}

path.Add(startVertex + 1); // Chuyển từ chỉ số bắt đầu từ 1 sang chỉ số bắt đầu từ 0

path.Reverse();

// Kiểm tra chiều dài của path

if (path.Count < 2)

{

// Không đủ đỉnh để tạo đường đi, không tính tổng độ dài

return $"{string.Join(" -> ", path)} độ dài: 0";

}

// Tính độ dài tổng của đường đi

int totalDistance = 0;

for (int i = 0; i < path.Count - 1; i++)

{

int vertex1 = path[i] - 1; // Chuyển từ chỉ số bắt đầu từ 1 sang chỉ số bắt đầu từ 0

int vertex2 = path[i + 1] - 1; // Chuyển từ chỉ số bắt đầu từ 1 sang chỉ số bắt đầu từ 0

// Kiểm tra giá trị của path[i] và path[i + 1] có nằm trong phạm vi của mảng không

if (vertex1 >= 0 && vertex1 < weightMatrix.GetLength(0) &&

vertex2 >= 0 && vertex2 < weightMatrix.GetLength(1))

{

totalDistance += weightMatrix[vertex1, vertex2];

}

else

{

// Xử lý ngoại lệ hoặc thông báo lỗi tùy thuộc vào yêu cầu của bạn

Console.WriteLine("Lỗi: Đỉnh nằm ngoài phạm vi của ma trận trọng số.");

}

}

return $"{string.Join(" -> ", path)} độ dài: {totalDistance}";

}

private void DisplayDijkstraPath(Dictionary<int, int> previousVertices, int startVertex, int endVertex, int totalDistance)

{

// Hiển thị đường đi và độ dài trong TextBox.

txtLog.AppendText($"Đường đi từ {startVertex + 1} đến {endVertex + 1}: {ConstructPath(previousVertices, startVertex, endVertex)}\n");

}

\*\*AreVerticesConnected():\*\* Kiểm tra xem có đường đi giữa hai đỉnh hay không.

\*\*Dijkstra():\*\* Thuật toán Dijkstra để tìm đường đi ngắn nhất.

\*\*ConstructPath():\*\* Xây dựng đường đi từ đỉnh xuất phát đến đỉnh kết thúc.

\*\*DisplayDijkstraPath():\*\* Hiển thị đường đi và độ dài trong TextBox.

Dưới đây là giải thích chi tiết từng phần của mã, kèm theo đoạn mã C# tương ứng:

## 2.19 Sự kiện btnF\_Click:\*\*

- Khi người dùng nhấn nút "F", thuật toán Floyd-Warshall được kích hoạt và kết quả được hiển thị.

private void btnF\_Click(object sender, EventArgs e)

{

int[,] distanceMatrix = FloydWarshall();

DisplayFloydWarshallResults(distanceMatrix);

}

## 2.20 Phương thức FloydWarshall:\*\*

- `FloydWarshall` tạo một ma trận khoảng cách và sau đó thực hiện thuật toán Floyd-Warshall để cập nhật ma trận này.

- Sử dụng ba vòng lặp để kiểm tra xem có đường đi ngắn hơn nào không, và nếu có, cập nhật ma trận khoảng cách.

private int[,] FloydWarshall()

{

int size = weightMatrix.GetLength(0);

int[,] distanceMatrix = new int[size, size];

// Khởi tạo ma trận khoảng cách với trọng số ban đầu

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

distanceMatrix[i, j] = weightMatrix[i, j];

}

}

// Áp dụng thuật toán Floyd-Warshall

for (int k = 0; k < size; k++)

{

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

if (distanceMatrix[i, k] != int.MaxValue && distanceMatrix[k, j] != int.MaxValue

&& distanceMatrix[i, k] + distanceMatrix[k, j] < distanceMatrix[i, j])

{

distanceMatrix[i, j] = distanceMatrix[i, k] + distanceMatrix[k, j];

}

}

}

}

return distanceMatrix;

}

## 2.21 Phương thức DisplayFloydWarshallResults:\*\*

- Hiển thị ma trận khoảng cách sau khi thuật toán Floyd-Warshall được thực hiện. Dữ liệu này thường được đưa vào một TextBox để dễ đọc.

private void DisplayFloydWarshallResults(int[,] distanceMatrix)

{

txtLog.AppendText("Đường Đi Ngắn Nhất Theo Thuật Toán Floyd-Warshall:\n");

for (int i = 0; i < distanceMatrix.GetLength(0); i++)

{

for (int j = 0; j < distanceMatrix.GetLength(1); j++)

{

txtLog.AppendText($"{distanceMatrix[i, j]} ");

// Thêm dấu cách giữa các số

if (j < distanceMatrix.GetLength(1) - 1)

{

txtLog.AppendText(" ");

}

}

// Xuống dòng sau mỗi hàng

txtLog.AppendText("\r\n");

}

// Xuống dòng sau bảng

txtLog.AppendText("\r\n");

}

## 2.22 Trực quan hóa Đồ thị:\*\*

- `HighlightShortestPath` tô màu đường đi ngắn nhất trên biểu đồ sử dụng màu đỏ.

- `ClearAllPathsColors` xóa màu của các đường đi trước đó để chuẩn bị cho việc tô màu mới.

- `ConstructShortestPath` xây dựng lại đường đi ngắn nhất từ ma trận đỉnh trước đó.

private void HighlightShortestPath(List<int> path)

{

ClearAllPathsColors();

Graphics g = pictureBox1.CreateGraphics();

Pen pen = new Pen(Color.Red, 2);

for (int i = 0; i < path.Count - 1; i++)

{

Point p1 = points[path[i] - 1];

Point p2 = points[path[i + 1] - 1];

Color segmentColor = pen.Color;

pen.Color = segmentColor;

g.DrawLine(pen, p1, p2);

}

pen.Dispose();

g.Dispose();

}

private void ClearAllPathsColors()

{

RedrawPictureBox();

}

private List<int> ConstructShortestPath(Dictionary<int, int> previousVertices, int startVertex, int endVertex)

{

// ...

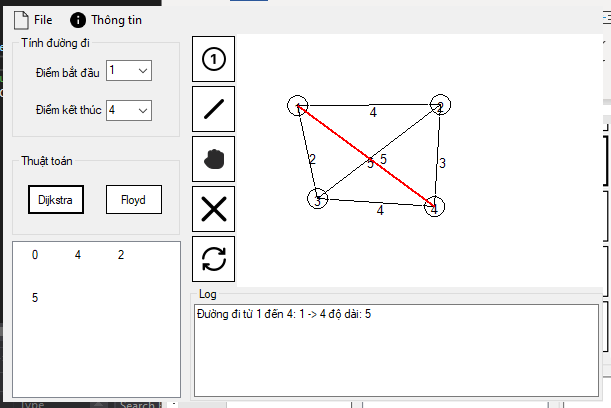
}

## 2.23 Demo thuật toán trên c#

### 2.23.1 Demo Thuật toán Dijkstra c#

Cho 4 đỉnh có độ dài như hình A screenshot of a computer

Description automatically generated

Chọn điểm bắt đầu và điểm kết thúc

2.23.2 Demo thuật toán floy: 