

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC KINH TẾ TP HỒ CHÍ MINH (UEH)
TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ**



ĐỒ ÁN MÔN HỌC CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

**ĐỀ TÀI
ỨNG DỤNG THUẬT TOÁN DIJKSTRA TỐI ƯU HOÁ CHI PHÍ VẬN TẢI**

**Học Phần: Cấu Trúc Dữ Liệu & Giải Thuật
Danh Sách Nhóm: NHÓM 9**

- 1. LÊ VŨ TÂM VINH**
- 2. VÕ THANH TÂN**
- 3. TRẦN THU HÀ**
- 4. KHÚC DUY ĐẠT**

**Chuyên Ngành: AN TOÀN THÔNG TIN
Khóa: K50**

Giảng Viên: TS. Đặng Ngọc Hoàng Thành

TP. Hồ Chí Minh, Ngày 05 tháng 04 năm 2021

Mục Lục

CHƯƠNG 1. THUẬT TOÁN DIJKSTRA	3
1.1. Thuật toán Dijkstra	3
1.2. Đồ thị	6
1.3. Cấu trúc và cài đặt	8
1.4. So sánh thuật toán Dijkstra với các thuật toán khác	14
1.5. Các ý tưởng khác	17
CHƯƠNG 2. THIẾT KẾ HỆ THỐNG	20
2.1. Mô hình tổng quát	20
2.2. Sơ đồ Lớp	34
2.3. Giao diện và chức năng chính	37
CHƯƠNG 3. THIẾT KẾ GIAO DIỆN	40
3.1. Giao Diện Menu Chính	40
3.2. Đánh giá hiệu suất thuật toán	42
3.3. Xử lý thuật toán trên giao diện	48
CHƯƠNG 4. THẢO LUẬN & ĐÁNH GIÁ	51
4.1. Kết quả nhận được	51
4.2. Đánh giá hiệu suất thuật toán	51
4.3. Hạn chế và hướng phát triển	51
PHỤ LỤC	53
TÀI LIỆU THAM KHẢO	58

CHƯƠNG 1. THUẬT TOÁN DIJKSTRA

1.1. Thuật toán Dijkstra

a) Giới thiệu thuật toán Dijkstra

Thuật toán Dijkstra là một thuật toán tìm đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh còn lại trong đồ thị có trọng số không âm. Thuật toán này hoạt động dựa trên nguyên lý tham lam (greedy) và được sử dụng rộng rãi trong các bài toán về định tuyến, mạng lưới giao thông, và các hệ thống tìm kiếm đường đi.

b) Lịch sử phát triển

Thuật toán được đề xuất bởi nhà khoa học máy tính người Hà Lan **Edsger W. Dijkstra** vào năm 1956 và chính thức công bố vào năm 1959. Ban đầu, thuật toán được sử dụng để tìm đường đi ngắn nhất trong mạng lưới đường bộ nhưng sau đó đã được áp dụng rộng rãi trong khoa học máy tính và công nghệ thông tin.

c) Mô tả chi tiết thuật toán Dijkstra

Thuật toán Dijkstra là một giải pháp hiệu quả để tìm đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh còn lại trong đồ thị có trọng số không âm. Quá trình hoạt động của thuật toán bao gồm hai giai đoạn: chiều thuận – dùng để tính khoảng cách ngắn nhất, và chiều nghịch – dùng để truy vết lại đường đi tương ứng.

Giai đoạn chiều thuận – Tính khoảng cách ngắn nhất

Đầu tiên, ta thực hiện bước khởi tạo. Chọn một đỉnh nguồn u_0 , và gán $t(u_0) = 0$, biểu thị rằng khoảng cách từ nguồn đến chính nó là 0. Với mọi đỉnh còn lại $v \neq u_0$, ta gán $t(v) = \infty$, thể hiện rằng chưa xác định được khoảng cách từ nguồn đến đỉnh đó. Tập các đỉnh đã xử lý ban đầu là rỗng.

Sau bước khởi tạo, thuật toán lặp lại nhiều bước, mỗi bước chọn ra một đỉnh chưa được xử lý có giá trị $t(u)$ nhỏ nhất và đánh dấu đỉnh đó đã được xử lý. Sau đó, với mỗi đỉnh kề v của đỉnh đang xét, nếu v chưa được xử lý, thuật toán tiến hành cập nhật khoảng cách theo công thức:

$$t(v) = \min(t(v), t(u_i) + \alpha(u_i, v))$$

trong đó $\alpha(u_i, v)$ là trọng số của cạnh nối từ đỉnh u_i đến đỉnh v . Quá trình này tiếp tục cho đến khi toàn bộ các đỉnh được đánh dấu hoặc khi đã tìm được đường đi ngắn nhất đến

đỉnh đích (nếu chỉ quan tâm đến một đích cụ thể).

Giai đoạn chiều nghịch – Truy vết hành trình

Sau khi tính toán xong khoảng cách ngắn nhất từ đỉnh nguồn đến đỉnh đích v , giả sử $t(v) = m$, ta tiến hành truy ngược hành trình. Bắt đầu từ đỉnh v , tìm đỉnh x_i thỏa mãn:

$$t(x_i) + L(x_i, v) = m$$

Khi điều kiện này đúng, x_i là đỉnh liền trước của v trên hành trình ngắn nhất. Quá trình truy vết tiếp tục từ x_i cho đến khi quay lại đỉnh nguồn, từ đó thu được toàn bộ đường đi tối ưu.

Cách triển khai hiệu quả

Hiệu quả của thuật toán Dijkstra phụ thuộc nhiều vào cách lựa chọn và xử lý các đỉnh. Có ba cách triển khai phổ biến:

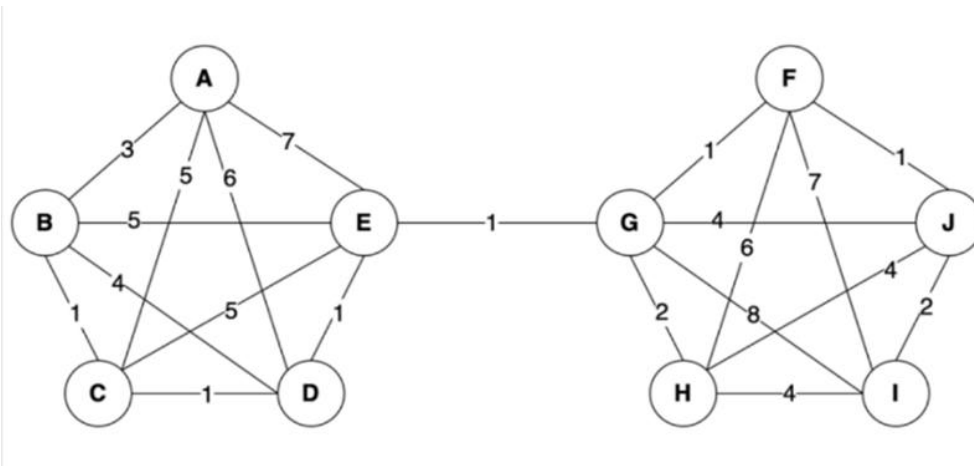
Sử dụng ma trận kề và tìm đỉnh nhỏ nhất bằng vòng lặp tuyến tính: Đây là cách đơn giản nhất, nhưng có độ phức tạp thời gian là $O(V^2)$, không phù hợp cho đồ thị lớn.

Sử dụng hàng đợi ưu tiên với Binary Heap: Đây là cách phổ biến và hiệu quả trong thực tế. Với danh sách kề làm cấu trúc lưu đồ thị và heap nhị phân làm hàng đợi, độ phức tạp giảm xuống còn $O((V + E)\log V)$.

Sử dụng hàng đợi Fibonacci: Đây là cách tối ưu nhất về lý thuyết, đạt độ phức tạp $O(E + V\log V)$, tuy nhiên việc cài đặt phức tạp và không phổ biến trong thực tiễn.

=> Tùy thuộc vào cấu trúc của đồ thị và yêu cầu về hiệu suất, người dùng có thể lựa chọn cách triển khai phù hợp để đạt được hiệu quả tính toán tốt nhất.

Ví dụ thuật toán



A	B	C	D	E	F	G	H	I	J
0*	∞	∞	∞	∞	∞	∞	∞	∞	∞
	3*	5	6	∞	∞	∞	∞	∞	∞
		4*	6	6	∞	∞	∞	∞	∞
			5*	6	∞	∞	∞	∞	∞
				6*	∞	∞	∞	∞	∞
					∞	7*	∞	∞	∞
					8*		9	15	11
							9	15	9*

Giải thích thuật toán Dijkstra qua bảng minh họa

Giả sử ta cần tìm đường đi ngắn nhất từ đỉnh A đến các đỉnh còn lại trong một đồ thị. Thuật toán Dijkstra sẽ thực hiện qua hai giai đoạn chính:

Giai đoạn 1: Tính khoảng cách ngắn nhất từ A đến các đỉnh (Chiều thuận)

Khởi tạo:

- Đặt khoảng cách từ A đến chính nó là 0: $t(A) = 0$
- Tất cả các đỉnh còn lại có khoảng cách ban đầu là ∞ (vô cực, chưa biết).
- Đỉnh A được chọn đầu tiên để bắt đầu xử lý.

Lặp lại quy trình sau:

1. Chọn đỉnh chưa xử lý nào có khoảng cách nhỏ nhất (những ô vàng có dấu * trong bảng minh họa).
2. Từ đỉnh này, xem có thể đi đến các đỉnh kề nào không.
3. Nếu có, cập nhật khoảng cách nếu đường đi mới ngắn hơn.
4. Lặp lại cho đến khi tất cả các đỉnh được xử lý hoặc đỉnh đích đã được đến.

Các bước cụ thể trong bảng:

Bước	Đỉnh đang xử lý	Cập nhật khoảng cách đến đỉnh
1	A (0)	$B = 0 + 3 = 3$
2	B (3)	$C = 3 + 1 = 4$
3	C (4)	$D = 4 + 1 = 5$
4	D (5)	$E = 5 + 1 = 6$
5	E (6)	$F = 6 + 1 = 7$
6	F (7)	$G = 7 + 1 = 8$
7	G (8)	$H = 8 + 1 = 9$ $I = 8 + 7 = 15$ $J = 8 + 3 = 11$
8	H (9)	Không có cập nhật thêm
9	J (11)	Không có cập nhật thêm
10	I (15)	Không có cập nhật thêm

Giai đoạn 2: Truy vết đường đi (Chiều nghịch)

Sau khi biết các khoảng cách tối thiểu, ta **truy ngược** lại để tìm đường đi.

- J (9) đến từ H ($t(H) = 8$, $H \rightarrow J = 1$)
- H đến từ G ($t(G) = 7$, $G \rightarrow H = 1$)
- G đến từ F ($F \rightarrow G = 1$)
- F đến từ E ($E \rightarrow F = 1$)
- E đến từ D ($D \rightarrow E = 1$)
- D đến từ C ($C \rightarrow D = 1$)
- C đến từ B ($B \rightarrow C = 1$)
- B đến từ A ($A \rightarrow B = 3$)

Vậy đường đi ngắn nhất là: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G \rightarrow H \rightarrow J$

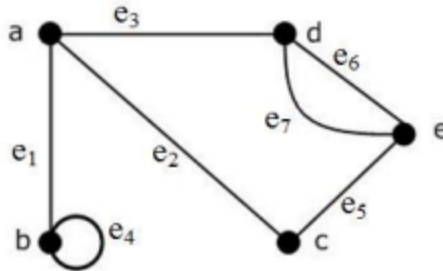
Tổng trọng số: **9**

1.2. Đồ thị

a. Khái niệm đồ thị

Đồ thị là một cấu trúc toán học gồm tập hợp các đỉnh (vertices) và các cạnh (edges) kết nối các đỉnh. Đồ thị được sử dụng rộng rãi trong khoa học máy tính để mô hình hóa các hệ thống như mạng máy tính, giao thông, mạng xã hội.

Hình ảnh minh họa đồ thị



b. Các thành phần chính của đồ thị

- Đỉnh (Vertex): Là điểm cơ bản trong đồ thị, đại diện cho thực thể.
- Cạnh (Edge): Là liên kết giữa hai đỉnh.
- Biểu diễn đồ thị: Có hai cách phổ biến:
- Ma trận kề: Một ma trận vuông biểu diễn sự kết nối giữa các đỉnh.
- Danh sách kề: Một danh sách lưu các đỉnh kề với từng đỉnh.

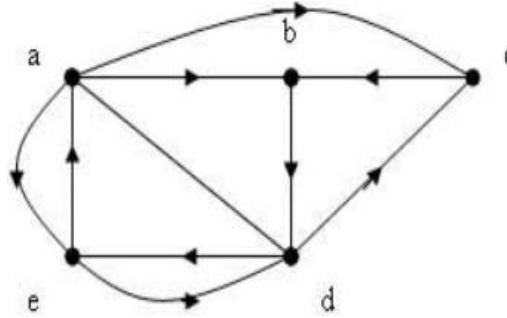
	e ₁	e ₂	e ₃	e ₄	e ₅	e ₆	e ₇
a	1	1	1	0	0	0	0
b	1	0	0	1	0	0	0
c	0	1	0	0	1	0	0
d	0	0	1	0	0	1	1
e	0	0	0	0	1	1	1

Biểu diễn ma trận liên thuộc từ đồ thị

c. Đồ thị có hướng và vô hướng

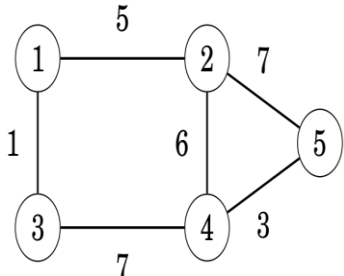
- Đồ thị có hướng (Directed Graph - Digraph)

Đồ thị có hướng là một đồ thị trong đó các cạnh có hướng di chuyển từ một đỉnh này đến một đỉnh khác. Nghĩa là, nếu có một cạnh từ đỉnh A đến đỉnh B, ta không thể đi theo hướng ngược lại trừ khi có một cạnh khác từ B đến A.

<p>Biểu diễn: $G=(V,E)$ trong đó:</p> <ul style="list-style-type: none"> • V là tập hợp các đỉnh (vertices). • E là tập hợp các cạnh có hướng (directed edges) dưới dạng cặp có thứ tự (u,v) (cạnh đi từ đỉnh u đến v). 	
---	--

• Đồ thị vô hướng (Undirected Graph)

Đồ thị vô hướng là đồ thị trong đó các cạnh không có hướng, nghĩa là nếu có một cạnh giữa hai đỉnh A và B, thì ta có thể di chuyển từ A đến B hoặc từ B đến A.

<p>Biểu diễn: $G=(V,E)$ trong đó:</p> <ul style="list-style-type: none"> • V là tập hợp các đỉnh. • E là tập hợp các cạnh không có hướng, được biểu diễn dưới dạng cặp không có thứ tự $\{u,v\}$. 	
---	--

1.3. Cấu trúc và cài đặt

a. Thuật toán Dijkstra

- Mã giả(Pseudocode)


```

1. Dijkstra(Graph, source):
2.   distance[source] ← 0
3.   for each vertex v in Graph:
4.     if v ≠ source then distance[v] ← ∞
5.     previous[v] ← undefined
6.   PriorityQueue Q ← all vertices in Graph
7.   while Q is not empty:
8.     u ← vertex in Q with min distance[u]
9.     remove u from Q
10.    for each neighbor v of u:
11.      alt ← distance[u] + weight(u, v)
12.      if alt < distance[v]:
13.        distance[v] ← alt
14.        previous[v] ← u
15.    return distance[], previous[]

```

Cài đặt thuật toán Dijkstra trong C#

```

1. using System;
2. using System.Collections.Generic;
3. public class DistOriginal
4. {
5.     public int distance;
6.     public int parentVert;
7.     public DistOriginal(int pv, int d)
8.     {
9.         distance = d;
10.        parentVert = pv;
11.    }
12. }
13. public class Vertex
14. {
15.     public string label;
16.     public bool isInTree;
17.     public Vertex(string lab)
18.     {
19.         label = lab;
20.         isInTree = false;
21.     }
22. }
23. public class Graph

```

```

24. {
25.     private const int max_verts = 20;
26.     int infinity = 1000000;
27.     Vertex[] vertexList;
28.     int[,] adjMat;
29.     int nVerts;
30.     int nTree;
31.     DistOriginal[] sPath;
32.     int currentVert;
33.     int startToCurrent;
34.     public Graph()
35.     {
36.         vertexList = new Vertex[max_verts];
37.         adjMat = new int[max_verts, max_verts];
38.         nVerts = 0;
39.         nTree = 0;
40.         for (int j = 0; j <= max_verts - 1; j++)
41.             for (int k = 0; k <= max_verts - 1; k++)
42.                 adjMat[j, k] = infinity;
43.         sPath = new DistOriginal[max_verts];
44.     }
45.     public void AddVertex(string lab)
46.     {
47.         vertexList[nVerts] = new Vertex(lab);
48.         nVerts++;
49.     }
50.     public void AddEdge(int start, int theEnd, int weight)
51.     {
52.         adjMat[start, theEnd] = weight;
53.     }
54.     public void Path()
55.     {
56.         int startTree = 0;
57.         vertexList[startTree].isInTree = true;
58.         nTree = 1;
59.         for (int j = 0; j < nVerts; j++)
60.         {
61.             int tempDist = adjMat[startTree, j];
62.             sPath[j] = new DistOriginal(startTree, tempDist);
63.         }
64.         while (nTree < nVerts)
65.         {
66.             int indexMin = GetMin();

```

```

67.     int minDist = sPath[indexMin].distance;
68.     currentVert = indexMin;
69.     startToCurrent = sPath[indexMin].distance;
70.     vertexList[currentVert].isInTree = true;
71.     nTree++;
72.     AdjustShortPath();
73. }
74. DisplayPaths();
75. nTree = 0;
76. for (int j = 0; j <= nVerts - 1; j++)
77.     vertexList[j].isInTree = false;
78. }
79. public int GetMin()
80. {
81.     int minDist = infinity;
82.     int indexMin = 0;
83.     for (int j = 1; j <= nVerts - 1; j++)
84.         if (!(vertexList[j].isInTree) &&
85.             sPath[j].distance < minDist)
86.         {
87.             minDist = sPath[j].distance;
88.             indexMin = j;
89.         }
90.     return indexMin;
91. }
92. public void AdjustShortPath()
93. {
94.     int column = 1;
95.     while (column < nVerts)
96.         if (vertexList[column].isInTree)
97.             column++;
98.         else
99.         {
100.             int currentToFringe = adjMat[currentVert, column];
101.             int startToFringe = startToCurrent + currentToFringe;
102.             int sPathDist = sPath[column].distance;
103.             if (startToFringe < sPathDist)
104.             {
105.                 sPath[column].parentVert = currentVert;
106.                 sPath[column].distance = startToFringe;
107.             }
108.             column++;
109.         }

```

```

110.     }
111.     public void DisplayPaths()
112.     {
113.         for (int j = 0; j <= nVerts - 1; j++)
114.         {
115.             Console.Write(vertexList[j].label + "=");
116.             if (sPath[j].distance == infinity)
117.                 Console.Write("inf");
118.             else
119.                 Console.Write(sPath[j].distance);
120.             string parent = vertexList[sPath[j].parentVert].label;
121.             Console.Write("(" + parent + ") ");
122.         }
123.     }
124. }

```

b. Đồ thị

- Cài đặt lớp đỉnh

```

// Cài đặt lớp đỉnh(Vertex)

1. public class Vertex
2. {
3.     public bool wasVisited;
4.     public string label;
5.     public Vertex(string label)
6.     {
7.         this.label = label;
8.         wasVisited = false;
9.     }
10. }

```

- Biểu diễn các cạnh

```

1. // Biểu diễn các cạnh
2.     int nVertices = 4;
3.     Vertex[] vertices = new Vertex[nVertices];

```

```

4.     vertices[0] = new Vertex("E");
5.     nVertices++;
6.     vertices[1] = new Vertex("F");
7.     nVertices++;
8.     vertices[2] = new Vertex("G");
9.     nVertices++;
10.    vertices[3] = new Vertex("H");
11.
12.    int[,] adjMatrix = new int[nVertices, nVertices];
13.    adjMatrix[0, 1] = 1;
14.    adjMatrix[1, 0] = 1;
15.    adjMatrix[1, 2] = 1;
16.    adjMatrix[2, 1] = 1;

```

- Cài đặt lớp đồ thị

//Cài đặt lớp đồ thị

```

1.  public class Graph
2.  {
3.      private const int NUM_VERTICES = 20;
4.      private Vertex[] vertices;
5.      private int[,] adjMatrix;
6.      int numVerts;
7.      public Graph()
8.      {
9.          vertices = new Vertex[NUM_VERTICES];
10.         adjMatrix = new int[NUM_VERTICES, NUM_VERTICES];
11.         numVerts = 0;
12.         for (int j = 0; j <= NUM_VERTICES; j++)
13.             for (int k = 0; k <= NUM_VERTICES - 1; k++)
14.                 adjMatrix[j, k] = 0;
15.     }
16.     public void AddVertex(string label)
17.     {
18.         vertices[numVerts] = new Vertex(label);
19.         numVerts++;
20.     }
21.     public void AddEdge(int start, int eend)
22.     {

```

```

23.     adjMatrix[start, eend] = 1;
24.     adjMatrix[eend, start] = 1;
25. }
26. public void ShowVertex(int v)
27. {
28.     Console.WriteLine(vertices[v].label + " ");
29. }
30. }

```

1.4 So sánh thuật toán Dijkstra với các thuật toán khác

Thuật toán	Mục đích chính	Độ phức tạp	Hiệu suất	Thời gian (xấp xỉ)	Ứng dụng tiêu biểu	Ưu điểm	Nhược điểm
BFS	Tìm đường ngắn nhất trên đồ thị không trọng số	$O(V + E)$	Rất nhanh	~1–5 ms	- Mạng xã hội - Game AI đơn giản - Bản đồ không trọng số	- Dễ cài đặt - Chạy cực nhanh trên đồ thị lớn không trọng số	- Không xử lý được trọng số - Không tối ưu nếu có đích cụ thể
Dijkstra	Tìm đường ngắn nhất với trọng số không âm	$O(V^2) / O((V+E)\log V)$	Nhanh vừa	~10–50 ms	- GPS, Google Maps - Hệ thống định tuyến (routing)	- Kết quả chính xác - Tốt cho đồ thị trọng số dương	- Không hỗ trợ trọng số âm - Chậm nếu không tối ưu hàng đợi ưu tiên
Bellman-Ford	Giống Dijkstra + hỗ trợ trọng số âm	$O(V \times E)$	Chậm hơn Dijkstra	~100–500 ms	- Mạng máy tính - Phát hiện chu trình âm - Các hệ thống động	- Hỗ trợ trọng số âm - Đơn giản hơn với biểu diễn	- Rất chậm trên đồ thị lớn - Không dùng cho mọi cặp đỉnh

						danh sách cạnh	
A*	Đường đi ngắn nhất + ưu tiên hướng đến đích	$O(E)$, tốt nếu heuristic tốt	Nhanh & thông minh	~10–30 ms	<ul style="list-style-type: none"> - Game AI (đi tìm đích) - Robot navigation 	<ul style="list-style-type: none"> - Cực nhanh nếu heuristic tốt - Tối ưu hóa hướng đi rất hiệu quả 	<ul style="list-style-type: none"> - Cần thiết kế heuristic tốt - Không tìm mọi đường đi – chỉ từ A đến B
Floyd-Warshall	Tìm tất cả cặp đường đi ngắn nhất	$O(V^3)$	Chậm với đồ thị lớn	~2–20 giây	<ul style="list-style-type: none"> - Phân tích mạng giao thông - Đồ thị tĩnh - Các bài toán so sánh toàn cục 	<ul style="list-style-type: none"> - Cực kỳ đơn giản cài đặt - Tìm mọi cặp đường đi 	<ul style="list-style-type: none"> - Rất chậm nếu V lớn (>500) - Không phù hợp cho đồ thị thay đổi liên tục (động)

Thuật toán Dijkstra được lựa chọn là giải pháp trọng tâm nhằm tối ưu hoá chi phí vận tải giữa các địa điểm thông qua việc xác định tuyến đường ngắn nhất trong một hệ thống giao thông có trọng số. Tuy nhiên, để đánh giá toàn diện, nhóm đã nghiên cứu và so sánh Dijkstra với các thuật toán phổ biến khác như BFS, Bellman-Ford, A* và Floyd-Warshall.

So với BFS, Dijkstra là một phiên bản nâng cao hơn khi xử lý đồ thị có trọng số. Trong khi BFS chỉ tìm đường ngắn nhất trên đồ thị không trọng số và hoạt động theo cơ chế lan rộng, thì Dijkstra xem xét trọng số từng cạnh và cho kết quả chính xác hơn về mặt chi phí. Tuy nhiên, nếu bài toán chỉ yêu cầu tìm đường đi qua ít cạnh nhất mà không xét chi phí, BFS sẽ hiệu quả hơn về thời gian.

So với Bellman-Ford, Dijkstra có hiệu suất vượt trội khi áp dụng trên đồ thị lớn với trọng số dương. Dù Bellman-Ford có thể xử lý các đồ thị có trọng số âm và phát hiện chu trình âm, song chi phí tính toán lớn khiến nó không phù hợp với bài toán vận tải – nơi mà chi phí không bao giờ âm và hiệu suất là yếu tố then chốt.

So với A*, Dijkstra mang lại tính tổng quát hơn vì không yêu cầu thông tin heuristic. A* được thiết kế để tìm đường nhanh hơn bằng cách định hướng quá trình tìm kiếm về phía đích, tuy nhiên thuật toán này chỉ phù hợp khi có mục tiêu cụ thể. Trong khi đó, Dijkstra có thể tính toán đường đi ngắn nhất từ một nguồn đến tất cả các điểm, rất hữu ích khi cần tổng hợp chi phí vận chuyển từ một trung tâm logistics đến nhiều điểm giao hàng.

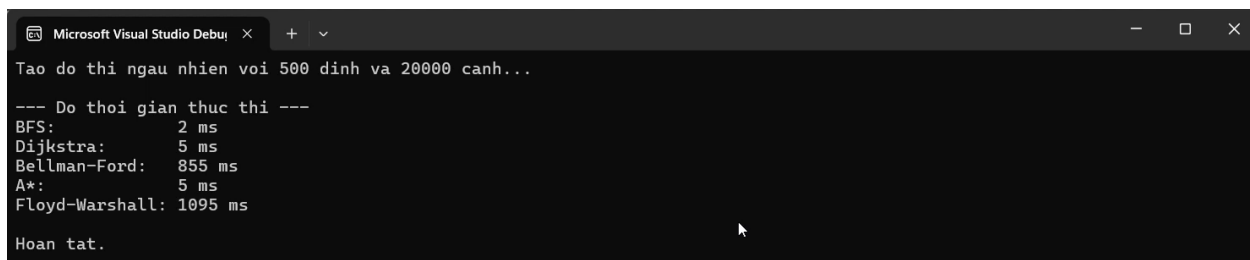
So với Floyd-Warshall, thuật toán Dijkstra có lợi thế lớn về hiệu năng. Floyd-Warshall cho phép tìm đường ngắn nhất giữa mọi cặp đỉnh, nhưng với độ phức tạp $O(V^3)$, nó không phù hợp cho các hệ thống lớn hoặc yêu cầu xử lý theo thời gian thực. Trong khi đó, Dijkstra đảm bảo kết quả chính xác với hiệu suất tối ưu hơn rất nhiều, đặc biệt khi chỉ cần tìm đường giữa một cặp điểm cụ thể.

Tóm lại, thuật toán Dijkstra là lựa chọn hợp lý và hiệu quả cho bài toán tối ưu hoá chi phí vận tải của đồ án. Với khả năng xử lý nhanh, chính xác, phù hợp với các hệ thống có trọng số dương, thuật toán này đóng vai trò cốt lõi trong việc xây dựng hệ thống tìm đường và tính toán chi phí trong ứng dụng MinCostRoute. Những hạn chế nhỏ của Dijkstra như không xử lý được trọng số âm không ảnh hưởng đến mô hình thực tế, và nhờ đó, nó vẫn là giải pháp đáng tin cậy cho bài toán vận chuyển đường bộ hiện nay.

Bây giờ nhóm sẽ đưa ra ví dụ:

-Đoạn code so sánh của nhóm được đề trên [Github](#)

-Video so sánh : [link](#)



```
Tạo đồ thị ngẫu nhiên với 500 đỉnh và 20000 cạnh...

--- Đo thời gian thực thi ---
BFS:      2 ms
Dijkstra:  5 ms
Bellman-Ford: 855 ms
A*:       5 ms
Floyd-Warshall: 1095 ms

Hoàn tất.
```

Trong thực nghiệm với đồ thị ngẫu nhiên gồm 500 đỉnh và 20.000 cạnh, thời gian thực thi của các thuật toán được đo như sau: BFS mất 2 ms, Dijkstra mất 5 ms, A* mất 5 ms, Bellman-Ford mất 855 ms và Floyd-Warshall mất 1095 ms.

Thuật toán BFS cho kết quả rất nhanh vì đây là thuật toán duyệt đơn giản, chỉ áp dụng tốt cho đồ thị không trọng số hoặc có trọng số bằng nhau. Khi chỉ cần tìm đường đi ngắn nhất trong đồ thị không trọng số, BFS sẽ cho kết quả chính xác với thời gian thực thi tối ưu.

Dijkstra và A* đều là các thuật toán tìm đường đi ngắn nhất từ một đỉnh nguồn trong đồ thị có trọng số dương. Trong trường hợp này, vì đồ thị được tạo ngẫu nhiên và không có yếu tố không gian hoặc thông tin vị trí để tận dụng heuristic, A* hoạt động tương tự

Dijkstra và cho kết quả gần như nhau. Cả hai chỉ mất 5 ms, chứng tỏ hiệu suất rất tốt trên đồ thị có kích thước vừa phải.

Thuật toán Bellman-Ford có độ phức tạp cao hơn Dijkstra, cụ thể là $O(V \times E)$, do cần lặp lại toàn bộ quá trình cập nhật cạnh đến $V-1$ lần. Với đồ thị có 500 đỉnh và 20.000 cạnh, tổng số lượt duyệt lên tới gần 10 triệu lần, khiến thời gian thực thi tăng lên rõ rệt, mất khoảng 855 ms.

Floyd-Warshall là thuật toán tính đường đi ngắn nhất giữa mọi cặp đỉnh (all-pairs shortest paths), với độ phức tạp $O(V^3)$. Trong trường hợp có 500 đỉnh, thuật toán thực hiện tới 125 triệu phép tính, do đó thời gian thực thi lên đến 1095 ms. Mặc dù Floyd-Warshall có thể tận dụng tốt việc truy xuất ma trận và có cách cài đặt gọn gàng, nhưng do phải xử lý mọi cặp đỉnh thay vì chỉ từ một nguồn, hiệu suất thực tế trở nên thấp hơn.

Nhìn chung, trong bài toán tìm đường đi ngắn nhất từ một đỉnh đến các đỉnh còn lại trên đồ thị có trọng số dương, Dijkstra và A^* là hai thuật toán có hiệu suất cao nhất. BFS hoạt động cực kỳ tốt trong trường hợp đồ thị không trọng số. Ngược lại, Bellman-Ford và Floyd-Warshall tuy có tính tổng quát cao và áp dụng được cho đồ thị có trọng số âm (với Bellman-Ford), nhưng thời gian thực thi không phù hợp với các đồ thị lớn hoặc bài toán yêu cầu kết quả nhanh.

1.5 Các ý tưởng khác

Thuật toán Dijkstra đã và đang được sử dụng rộng rãi trong việc tìm đường đi ngắn nhất giữa các tỉnh ở Việt Nam, dựa trên mô hình hóa hệ thống đường bộ như một đồ thị có trọng số.

Các bước thực hiện:

a. Mô hình hóa bài toán dưới dạng đồ thị

- **Đỉnh (Nodes):** Các tỉnh/thành phố được xem như các đỉnh trong đồ thị.
- **Cạnh (Edges):** Các tuyến đường kết nối giữa các tỉnh là các cạnh.
- **Trọng số (Weights):** Trọng số có thể là quãng đường (km), thời gian di chuyển (phút), chi phí di chuyển (VNĐ) tùy vào nhu cầu bài toán.

Ví dụ, xét các tỉnh lân cận TP.HCM:

- TP.HCM \rightarrow Bình Dương (30km)
- TP.HCM \rightarrow Đồng Nai (35km)
- TP.HCM \rightarrow Bà Rịa - Vũng Tàu (95km)

b. Áp dụng thuật toán Dijkstra

Khi cần tìm đường đi ngắn nhất từ TP.HCM đến một tỉnh lân cận, thuật toán Dijkstra sẽ hoạt động như sau:

- **Bước 1: Khởi tạo**
 - Gán khoảng cách từ TP.HCM đến chính nó là **0**.
 - Gán khoảng cách từ TP.HCM đến các tỉnh khác là ∞ (vô cùng).
- **Bước 2: Duyệt đồ thị**
 - Chọn đỉnh có khoảng cách nhỏ nhất (ban đầu là TP.HCM).
 - Cập nhật khoảng cách đến các tỉnh lân cận.
 - Đánh dấu TP.HCM là đã xử lý.
 - Tiếp tục chọn tỉnh có khoảng cách nhỏ nhất tiếp theo (ví dụ: Bình Dương) và cập nhật khoảng cách của các tỉnh lân cận của nó.
- **Bước 3: Lặp lại**
 - Thuật toán tiếp tục cập nhật cho đến khi tìm được đường đi ngắn nhất từ TP.HCM đến tất cả các tỉnh lân cận.
- **Bước 4: Truy xuất kết quả:**
 - Sau khi chạy xong, thuật toán sẽ trả về khoảng cách và tuyến đường ngắn nhất từ TP.HCM đến các tỉnh lân cận.
 - Ví dụ, đường đi ngắn nhất từ TP.HCM đến Tây Ninh có thể là:
TP.HCM \rightarrow Bình Dương \rightarrow Tây Ninh (130km).

c. Mở rộng và tối ưu hóa.

- **Thay đổi trọng số:** Nếu muốn tìm đường đi nhanh nhất, có thể sử dụng thời gian di chuyển thay vì khoảng cách.
- **Cập nhật dữ liệu theo thời gian thực:** Tích hợp với dữ liệu giao thông để tối ưu tuyến đường.

-**Ứng dụng vào thực tế:** Có thể áp dụng vào Google Maps hoặc các ứng dụng tìm đường để hỗ trợ tài xế, người dân di chuyển thuận tiện hơn.

d. Ứng dụng thực tế của thuật toán Dijkstra.

-**Ứng dụng trong điều hướng và bản đồ số như Google Maps, Apple Maps, Vietmap:** sử dụng thuật toán Dijkstra để tính toán lộ trình ngắn nhất hoặc nhanh nhất giữa hai địa điểm.



• Các ứng dụng thực tế khác:

- + Mạng viễn thông & Internet (OSPF, định tuyến gói tin)
- + Logistics & Vận tải (Giao hàng tiết kiệm, UPS, FedEx)
- + Robot & AI (Xe tự hành Tesla, robot hút bụi)
- + Tài chính & Ngân hàng (Tối ưu giao dịch chứng khoán)

CHƯƠNG 2. THIẾT KẾ HỆ THỐNG

2.1. Mô hình tổng quát

- **Phân tích yêu cầu bài toán tối ưu chi phí vận tải bằng thuật toán Dijkstra.**

Yêu cầu: Giả sử chi phí vận tải giữa các địa điểm trong một quốc gia được tính toán sẵn. Xây dựng ứng dụng cho phép tìm kiếm một đường đi để chuyển hàng từ điểm A đến điểm X được chỉ định với chi phí tối thiểu.

Mục tiêu: Dự án "Ứng dụng thuật toán Dijkstra để tối ưu hóa chi phí vận tải" nhằm giải quyết bài toán tìm tuyến đường vận chuyển hàng hóa có chi phí thấp nhất trong một hệ thống giao thông cho trước. Dự án hướng đến các mục tiêu sau:

- Ứng dụng trong ngành Logistics & Giao hàng
- Quản lý chuỗi cung ứng(Supply Chain Management)
- Hệ thống giao thông thông minh(Smart Transportation)
- Ứng dụng trong du lịch & vận tải hành khách

Để xây dựng chương trình nhằm tối ưu hóa chi phí vận tải, ta cần một đồ thị có trọng số với các yếu tố sau:

- **Đỉnh:** Các địa điểm trong một quốc gia (Sài Gòn, Bình Dương,...).
- **Chi phí:** Chi phí khi di chuyển trên mỗi quãng đường.
- Thuật toán Dijkstra tìm đường đi ngắn nhất giữa 2 địa điểm (đỉnh)
- **Chi phí:** Trong bài toán tìm đường đi ngắn nhất thông thường, ta thường xem chi phí là quãng đường đi giữa 2 đỉnh. Tuy vậy, khi xây dựng ứng dụng nhằm tối ưu hóa chi phí vận tải, chi phí sẽ bao gồm các yếu tố:
 - Tổng quãng đường di chuyển(km)
 - Trọng lượng của hàng hóa(tấn)
 - Mức tiêu thụ nhiên liệu
 - Giá tiền của mỗi lít xăng
$$\text{Chi phí khoảng cách} = \text{Tổng quãng đường} \times \text{Đơn giá vận chuyển} \times \text{khối lượng hàng hóa}$$
 - Đơn giá vận chuyển(tấn)

$$\text{Chi phí nhiên liệu} = \frac{\text{Tổng quãng đường}}{100} \times \text{mức tiêu thụ nhiên liệu} \times \text{Giá nhiên liệu cho mỗi lít}$$

$$\text{Chi phí tổng} = \text{Chi phí nhiên liệu} + \text{Chi phí khoảng cách}$$

- ★ Ta tạo lớp **Node** để cung cấp phần tử cơ sở để xây dựng các cấu trúc dữ liệu như **LinkedList**, giúp quản lý danh sách các phần tử một cách linh hoạt (chèn, xóa, duyệt,...).

- **Mô tả:**

Lớp **Node** là một cấu trúc cơ bản trong danh sách liên kết đơn, dùng để lưu trữ một phần tử và liên kết đến phần tử tiếp theo trong danh sách.

- **Thuộc tính:**

- + `element(object)`: Dữ liệu được lưu trong Node.
- + `link(Node)`: Liên kết đến node kế tiếp trong danh sách.

- **Phương thức:**

Tên phương thức	Tham số	Mô tả
Node()	Không có	Hàm tạo mặc định, khởi tạo <i>element</i> và <i>link</i> là <i>null</i> .
Node(object element)	Element: dữ liệu cần lưu	Hàm tạo có tham số, khởi tạo <i>element</i> với giá trị truyền vào và <i>link</i> là <i>null</i> .

```

1. public class Node
2. {
3.     public object element;
4.     public Node link;
5.     public Node()
6.     {
7.         element = null;
8.         link = null;
9.     }
10.    public Node(object element)
11.    {
12.        this.element = element;
13.        link = null;
14.    }
15. }

```

- ★ Ta tạo lớp **LinkedList** để quản lý các thao tác với danh sách liên kết đơn: thêm, xoá, tìm kiếm node.

- **Mô tả:**

Lớp này đại diện cho danh sách liên kết đơn, gồm nhiều đối tượng **Node** liên kết với nhau qua thuộc tính **link**.

- **Thuộc tính:**

header(Node): Node đầu danh sách, thường dùng làm điểm bắt đầu(giá trị mặc định là “Header”).

- **Phương thức:**

<i>Tên phương thức</i>	<i>Tham số</i>	<i>Mô tả</i>
LinkedList()	Không có	Khởi tạo danh sách với header mặc định.
Find(object element)	element – dữ liệu cần tìm	Trả về node chứa giá trị khớp.
Insert(object newelement, object afterelement)	object afterelement Chèn node mới có newelement sau node chứa afterelement .	
FindPrev(object element)	Tìm node phía trước node chứa element .	
Remove(object element)	Xoá node chứa element khỏi danh sách.	

```
1. public class LinkedList
```

```

2.  {
3.      public Node header;
4.      public LinkedList()
5.      {
6.          header = new Node("Header");
7.      }
8.      private Node Find(object element)
9.      {
10.         Node current = new Node();
11.         current = header;
12.         while (current.element != element)
13.         {
14.             current = current.link;
15.         }
16.         return current;
17.     }
18.     public void Insert(object newelement, object afterelement)
19.     {
20.         Node current = new Node();
21.         Node newnode = new Node(newelement);
22.         current = Find(afterelement);
23.         newnode.link = current.link;
24.         current.link = newnode;
25.     }
26.     public Node FindPrev(object element)
27.     {
28.         Node current = header;
29.         while (current.link != null && current.link.element != element)
30.         {
31.             current = current.link;
32.         }
33.         return current;
34.     }
35.     public void Remove(object element)
36.     {
37.         Node current = FindPrev(element);
38.         if (current.link != null)
39.         {
40.             current.link = current.link.link;
41.         }
42.     }
43. }

```

- ★ Ta tạo lớp Node 2 để dùng làm phần tử cơ bản trong cấu trúc danh sách liên kết đôi, hỗ trợ **duyệt theo hai chiều**.
- **Mô tả:**

Lớp đại diện cho một nút trong danh sách liên kết đôi (**doubly linked list**), gồm liên kết tới cả node trước (**blink**) và node sau (**flink**).

- **Thuộc tính:**

element(object): dữ liệu lưu trữ

flink(Node2): Liên kết tới node sau

blink(Node2): Liên kết tới node trước

- **Phương thức:**

Tên phương thức	Tham số	Mô tả
<i>Node2()</i>	Không có	Tạo node rỗng, các liên kết là null.
<i>Node2(object element)</i>	Element - dữ liệu	Tạo node với dữ liệu, liên kết mặc định là null.

```

1. public class Node2
2. {
3.     public object element;
4.     public Node2 flink, blink;
5.     public Node2()
6.     {
7.         element = null;
8.         flink = blink = null;
9.     }
10.    public Node2(object element)
11.    {
12.        this.element = element;
13.        flink = blink = null;
14.    }
15. }
```


- ★ Ta tạo lớp `DoubleLinkedList` để quản lý danh sách liên kết đôi bằng cách cung cấp các thao tác cơ bản như: thêm node mới, tìm kiếm node, xóa node.
- **Mô tả:**

Lớp này biểu diễn **danh sách liên kết đôi (doubly linked list)**, nơi mỗi node có thể trỏ tới **cả node phía trước và node phía sau**, nhờ đó hỗ trợ **duyệt 2 chiều** một cách linh hoạt.

- **Thuộc tính:**

`header(Node2)`: Node đầu tiên của danh sách (có thể dùng như node giả, chứa "Header").

- **Phương thức:**

Tên phương thức	Tham số	Mô tả
<code>DoubleLinkedList()</code>	Không có	Hàm tạo danh sách, khởi tạo header với giá trị "Header".
<code>Find(object element)</code>	element : dữ liệu cần tìm	Tìm node có chứa giá trị khớp với element . Trả về node đó.
<code>Insert(object newelement, object afterelement)</code>	newelement : dữ liệu mới, afterelement : dữ liệu node đứng trước vị trí cần chèn	Chèn node mới sau node chứa afterelement . Cập nhật liên kết flink và blink .
<code>Remove(object element)</code>	element : dữ liệu cần xóa	Xóa node chứa element khỏi danh sách bằng cách cập nhật các liên kết blink và flink .

```

1. public class DoubleLinkedList
2. {
3.     public Node2 header;
4.     public DoubleLinkedList()
5.     {
6.         header = new Node2("Header");
7.     }
8.     private Node2 Find(object element)
9.     {
10.        Node2 current = new Node2();
11.        current = header;
12.        while (current.element != element)
13.        {
14.            current = current.flink;
15.        }
16.        return current;
17.    }
18.    public void Insert(object newelement, object afterelement)
19.    {
20.        Node2 current = new Node2();
21.        Node2 newnode = new Node2(newelement);
22.        current = Find(afterelement);
23.        newnode.flink = current.flink;
24.        newnode.blink = current;
25.        current.flink = newnode;
26.    }
27.    public void Remove(object element)
28.    {
29.        Node2 current = Find(element);
30.        if (current.flink != null)
31.        {
32.            current.blink.flink = current.flink;
33.            current.flink.blink = current.blink;
34.            current.flink = null;
35.            current.blink = null;
36.        }
37.    }
38. }

```

ĐỈNH

- ★ Ta tạo **lớp Location** để định vị các điểm trên bản đồ.
- **Mô tả:** Đại diện cho một địa điểm với tên, ký hiệu và tọa độ.
- **Thuộc tính:**
 - + **Name** (string) - Tên của địa điểm.
 - + **Symbol** (string) - Ký hiệu ngắn của địa điểm.
 - + **Coordinates** (Point) - Vị trí tọa độ x, y.
- **Phương thức:**
 - + Hàm dựng **Location(name, symbol, x, y)**: Khởi tạo một địa điểm.

```
1. public class Location
2. {
3.     public string Name { get; }
4.     public string Symbol { get; }
5.     public Point Coordinates { get; }
6.
7.     public Location(string name, string symbol, int x, int y)
8.     {
9.         Name = name;
10.        Symbol = symbol;
11.        Coordinates = new Point(x, y);
12.    }
13. }
```

- ★ Tạo lớp **đỉnh(Vertex)** để lưu thông tin của mỗi đỉnh trong đồ thị gồm các thuộc tính:
- **Mô tả:** Đại diện cho một đỉnh trong đồ thị.
- **Thuộc tính:**
 - + **Name** (string) - Tên của đỉnh.
 - + **IsInTree** (bool) - Kiểm tra xem đỉnh có thuộc cây đường đi hay không.
 - + **parent** (int) - Chỉ mục của đỉnh cha.
 - + **Distance** (int) - Khoảng cách từ nguồn.
 - + **X, Y** (int) - Tọa độ vị trí.
- **Phương thức:**
 - + Hàm dựng **Vertex(name, x, y)**: Khởi tạo một đỉnh với tên và tọa độ.

```

1. public class Vertex
2. {
3.     public string Name { get; }
4.     public bool IsInTree { get; set; }
5.     public int parent { get; set; }
6.     public int Distance { get; set; }
7.     public int X { get; }
8.     public int Y { get; }
9.
10.    public Vertex(string name, int x, int y)
11.    {
12.        Name = name;
13.        IsInTree = false;
14.        parent = -1;
15.        Distance = int.MaxValue;
16.        X = x;
17.        Y = y;
18.    }
19. }

```

THUẬT TOÁN DIJKSTRA

- ★ Ta tạo lớp **PQueue** để hỗ trợ thuật toán Dijkstra bằng cách luôn ưu tiên xử lý đỉnh có chi phí thấp nhất.

- **Mô tả:**

Lớp hàng đợi ưu tiên (Priority Queue) dùng để hỗ trợ thuật toán Dijkstra. Ưu tiên dựa trên **Priority** (thường là khoảng cách ngắn nhất tại thời điểm xét). Kế thừa từ **Queue<PqItem>**.

- **Thuộc tính:**

Không khai báo thuộc tính riêng (sử dụng kế thừa từ **Queue<PqItem>**), nhưng làm việc với:

- + **PqItem** – Một phần tử trong hàng đợi chứa thông tin:
 - **index** – Vị trí (số hiệu) của đỉnh trong đồ thị.
 - **priority** – Độ ưu tiên (khoảng cách ngắn nhất tạm thời đến đỉnh này).

- **Phương thức:**
- + **PQueue():** Hàm khởi tạo mặc định.
- + **Enqueue(int index, int priority):** Thêm một phần tử vào hàng đợi với chỉ số đỉnh và độ ưu tiên tương ứng.
- + **Dequeue():** Xóa và trả về phần tử có độ ưu tiên thấp nhất (nhỏ nhất) trong hàng đợi.

★ Ta tạo **class Graph** để thực hiện thuật toán Dijkstra tối ưu chi phí vận tải gồm:

Mô tả: Quản lý danh sách đỉnh, cạnh và các thuật toán tìm đường đi ngắn nhất.

Thuộc tính:

- **vertexList** (Vertex[]) - Danh sách đỉnh.
- **adjMat** (double[,]) - Ma trận trọng số lưu khoảng cách giữa các đỉnh.
- **totalfuelcost** (double[,]) - Ma trận chi phí nhiên liệu.
- **totalcostperKM** (double[,]) - Ma trận chi phí vận chuyển theo km.
- **nVerts** (int) - Số lượng đỉnh.
- **fuelPricePerLitre** (double) - Giá nhiên liệu trên lít.
- **fuelEfficiency** (double) - Hiệu suất nhiên liệu của xe.
- **costperKM** (double) - Chi phí vận chuyển trên km.

Phương thức:

- **AddVertex(name, x, y):** Thêm đỉnh vào đồ thị.
- **AddEdge(start, end, distance):** Thêm cạnh giữa hai đỉnh.
- **CalculateFuelCost(path):** Tính chi phí nhiên liệu cho đường đi.
- **DistanceCost(path, weight):** Tính chi phí vận chuyển.
- **TotalDistance(path):** Tính tổng quãng đường.

```

1. public class Graph
2. {
3.     private const int max_verts = 20;
4.     public int infinity = 1000000;
5.     private double fuelPricePerLitre = 20000;
6.     public double fuelEfficiency ;
7.     public double weight;
8.     public double costperKM = 24000;
9.     public double[,] totalfuelcost;
10.    public double[,] totalcostperKM;
11.    public Vertex[] vertexList;

```

```

12. public double[,] adjMat;
13. public int nVerts;
14. private PQueue pq;
15. public Graph()
16. {
17.     vertexList = new Vertex[max_verts];
18.     adjMat = new double[max_verts, max_verts];
19.     totalfuelcost = new double[max_verts, max_verts];
20.     totalcostperKM = new double[max_verts, max_verts];
21.     nVerts = 0;
22.     pq = new PQueue();
23.     for (int j = 0; j < max_verts; j++)
24.     {
25.         for (int k = 0; k < max_verts; k++)
26.         {
27.             adjMat[j, k] = infinity;
28.             totalfuelcost[j, k] = infinity;
29.             totalcostperKM[j, k] = infinity;
30.         }
31.     }
32. }
33. public void AddVertex(string label, int x, int y)
34. {
35.     vertexList[nVerts++] = new Vertex(label, x, y);
36. }
37. public void AddEdge(int start, int end, double distance)
38. {
39.
40.     adjMat[start, end] = distance;
41.     adjMat[end, start] = distance;
42.     double cosperKM = distance * costperKM*weight;
43.     totalcostperKM[start, end] = (double)cosperKM;
44.     totalcostperKM[end, start] = (double)cosperKM;
45.     double fuelcost = (distance / 100)*fuelEfficiency* fuelPricePerLitre;
46.     totalfuelcost[start, end] = (double)fuelcost;
47.     totalfuelcost[end, start] = (double)fuelcost;
48. }
49. public double Calculatefuelcost(DoubleLinkedList path)
50. {
51.     double totalDistance = 0;
52.     Node2 current = path.header.flink;
53.     while (current != null && current.flink != null)
54.     {

```

```

55.     int start = (int)current.element;
56.     int end = (int)current.flink.element;
57.
58.     totalDistance += adjMat[start, end];
59.     current = current.flink;
60. }
61.     double fuelcost = (totalDistance / 100)* fuelEfficiency *
    fuelPricePerLitre;
62.     return fuelcost;
63. }
64. public double DistanceCost(DoubleLinkedList path ,double weight)
65. {
66.     double totalDistance = 0;
67.     Node2 current = path.header.flink;
68.     while (current != null && current.flink != null)
69.     {
70.         int start = (int)current.element;
71.         int end = (int)current.flink.element;
72.
73.         totalDistance += adjMat[start, end];
74.         current = current.flink;
75.     }
76.     double distancecost = totalDistance* costperKM*weight;
77.     return distancecost;
78. }
79. public double TotalDistance(DoubleLinkedList path)
80. {
81.     double totalDistance = 0;
82.     Node2 current = path.header.flink;
83.     while (current != null && current.flink != null)
84.     {
85.         int start = (int)current.element;
86.         int end = (int)current.flink.element;
87.
88.         totalDistance += adjMat[start, end];
89.         current = current.flink;
90.     }
91.     return totalDistance;
92. }
93. public void Path(int start)
94. {
95.     vertexList[start].Distance = 0;
96.     pq.Enqueue(start, 0);

```

```

97.     while (pq.Count > 0)
98.     {
99.         PqItem pqItem = pq.Dequeue();
100.        int current = pqItem.Index;
101.        vertexList[current].IsInTree = true;
102.        for (int j = 0; j < nVerts; j++)
103.        {
104.            if (!vertexList[j].IsInTree && adjMat[current, j] != infinity)
105.            {
106.                double costPerKm = totalcostperKM[current, j];
107.                double fuelCost = totalfuelcost[current, j];
108.                double newCost = vertexList[current].Distance +
                    costPerKm + fuelCost;
109.                if (newCost < vertexList[j].Distance)
110.                {
111.                    vertexList[j].Distance = (int)newCost;
112.                    vertexList[j].parent = current;
113.                    pq.Enqueue(j, (int)newCost);
114.                }
115.            }
116.        }
117.    }
118. }

119. public void AddConnection(string from, string to, double distance)
120. {
121.     int fromIndex = GetLocationIndex(from);
122.     int toIndex = GetLocationIndex(to);
123.     if (fromIndex != -1 && toIndex != -1)
124.     {
125.         AddEdge(fromIndex, toIndex, distance);
126.     }
127. }

128. public void AddLocation(string name, string symbol, int x, int y)
129. {
130.     AddVertex(name, x, y);
131. }

132. public LinkedList GetLocations()
133. {
134.     LinkedList locations = new LinkedList();
135.     if (nVerts == 0) return locations;
136.     locations.Insert(vertexList[0].Name, "Header");
137.     for (int i = 1; i < nVerts; i++)
138.     {

```

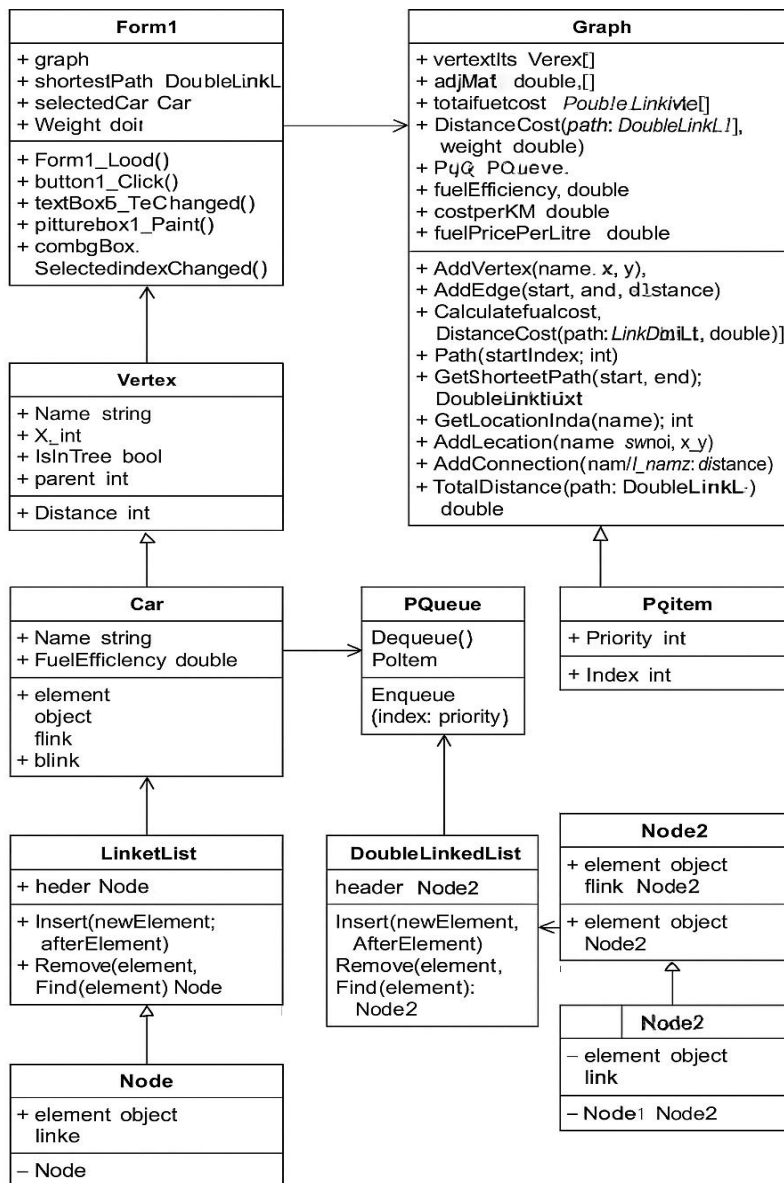


```

139.         locations.Insert(vertexList[i].Name, vertexList[i - 1].Name);
140.     }
141.     return locations;
142. }
143. public int GetLocationIndex(string name)
144. {
145.     for (int i = 0; i < nVerts; i++)
146.     {
147.         if (vertexList[i].Name == name)
148.         {
149.             return i;
150.         }
151.     }
152.     return -1;
153. }
154. public DoubleLinkedList GetShortestPath(int start, int end)
155. {
156.     DoubleLinkedList path = new DoubleLinkedList();
157.     int current = end;
158.     path.Insert(current, "Header");
159.     while (current != -1)
160.     {
161.         current = vertexList[current].parent;
162.         if (current != -1)
163.         {
164.             path.Insert(current, "Header");
165.         }
166.     }
167.     return path;
168. }
169. }

```

2.2. Sơ đồ Lớp



1. Lớp Location (Vị trí)

Thuộc tính:

- **latitude**: Tọa độ vĩ độ (float).
- **longitude**: Tọa độ kinh độ (float).
- **name**: Tên vị trí (string).

Phương thức:

- **toString()**: Chuyển đổi thông tin vị trí thành chuỗi.
- **distanceTo(that:Loc)**: Tính khoảng cách đến một vị trí khác.

2. Lớp Vertex (Đỉnh trong đồ thị)

- Thuộc tính:

- **Id**: Định danh của đỉnh.
- **Data**: Dữ liệu của đỉnh.
- **Confidence**: Độ tin cậy của đỉnh (double).
- **Position**: Vị trí (có mối quan hệ với **Location**).

- Phương thức:

- **addNeighbor(vertex: Vertex)**: Thêm một đỉnh lân cận.
- **removeNeighbor(vertex: Vertex)**: Xóa một đỉnh lân cận.
- **toString()**: Biểu diễn thông tin đỉnh dưới dạng chuỗi.

3. Lớp Graph (Đồ thị)

- Thuộc tính:

- **vertices**: Danh sách các đỉnh (Vertex).
- **directed**: Cho biết đồ thị có hướng hay không (boolean).

- Phương thức:

- **addVertex(vertex: Vertex)**: Thêm một đỉnh vào đồ thị.
- **removeVertex(id: int)**: Xóa một đỉnh khỏi đồ thị.
- **findShortestPath(start: int, end: int): Path**: Tìm đường đi ngắn nhất giữa hai đỉnh.

4. Lớp Car (Xe)

- Thuộc tính:

- **licensePlate**: Biển số xe (string).
- **type**: Loại xe (string).
- **maxSpeed**: Tốc độ tối đa (double).
- **status**: Trạng thái (st).

- Phương thức:

- **moveTo(v: Vertex)**: Di chuyển xe đến một đỉnh cụ thể trong đồ thị.
- **getCurrentLocation(): Location**: Lấy vị trí hiện tại của xe.
- **calculateRoute(Graph)**: Tính toán tuyến đường dựa trên đồ thị.

5. Class Node

- Thuộc tính

- **Data**: giá trị dữ liệu (kiểu **int**).

- **Next**: tham chiếu đến nút kế tiếp trong danh sách.

- Phương thức:

Không có phương thức riêng, chỉ là class đơn giản đại diện cho một phần tử trong **LinkedList**.

6. Class **Node2**

- Thuộc tính:

- **location**: đối tượng **Location** lưu trữ thông tin về một vị trí (name, tọa độ...).
- **flink**: con trỏ đến phần tử tiếp theo (forward link).
- **blink**: con trỏ đến phần tử trước đó (backward link).

- Phương thức:

Tương tự **Node**, chỉ chứa dữ liệu, không có phương thức riêng.

7. Class **LinkedList**

- Thuộc tính:

- **head**: phần tử đầu của danh sách liên kết đơn.

- Phương thức:

- Thêm một phần tử mới vào cuối danh sách.

8. Class **DoubleLinkedList**

- Thuộc tính:

- **head**: phần tử đầu của danh sách liên kết đôi.
- **tail**: phần tử cuối.

- Phương thức:

- Thêm một **Node2** mới chứa **Location** vào cuối danh sách.
- Tìm và xóa phần tử chứa **Location** ra khỏi danh sách.
- Xóa toàn bộ danh sách bằng cách đặt **head** và **tail** về **null**.
- Trả về danh sách các **Location** hiện có trong danh sách.

9. Class **PQueue (Priority Queue)**

- Thuộc tính:

- **elements**: danh sách lưu trữ các cặp **Location** và giá trị độ ưu tiên (**double**).

- Phương thức:

- Thêm một phần tử vào hàng đợi theo độ ưu tiên.
- Trả ra phần tử có độ ưu tiên nhỏ nhất và loại nó khỏi hàng đợi.
- Kiểm tra xem hàng đợi có rỗng không.

2.3. Giao diện và chức năng chính

a. Giao Diện Chính (UI)

Phần mềm có các thành phần giao diện chính như sau:

- Chọn điểm xuất phát và điểm đến

- ComboBox1 & ComboBox2:
 - Dùng để chọn điểm bắt đầu và điểm kết thúc của hành trình.
 - Các địa điểm là các tỉnh thuộc khu vực miền Nam Việt Nam như TP.HCM, Cần Thơ, An Giang, Kiên Giang, v.v.

- Chọn loại xe vận chuyển

- ComboBox3:
 - Dùng để chọn loại xe tải.
 - Danh sách xe gồm các loại như Hino FC9JNTC, Isuzu NQR75M, Hyundai Mighty 110XL, v.v.
 - Khi người dùng chọn xe, chương trình cập nhật mức tiêu hao nhiên liệu (**FuelEfficiency**) của xe đó.

- Nhập khối lượng hàng hóa

- TextBox6:
 - Người dùng nhập khối lượng hàng hóa cần vận chuyển (tính theo tấn).
 - Giới hạn tối đa là 5 tấn.
 - Nếu nhập giá trị không hợp lệ, chương trình sẽ hiển thị thông báo lỗi.

- Hiển thị chi phí vận chuyển

- TextBox2: Chi phí tính theo khoảng cách (**DistanceCost**).
- TextBox3: Tổng chi phí vận chuyển (**TotalCost** = tiền nhiên liệu + tiền vận chuyển theo khoảng cách).
- TextBox4: Tiền nhiên liệu (**FuelCost**).
- TextBox7: Tổng khoảng cách (**TotalDistance**).

- Bản đồ minh họa tuyến đường

- **PictureBox1:**

- Hiển thị bản đồ với các địa điểm được đánh dấu.
- Tuyến đường ngắn nhất sẽ được vẽ bằng màu vàng.
- Điểm xuất phát màu xanh lá cây, điểm đến màu đỏ.
- Các tuyến đường kết nối giữa các địa điểm được vẽ bằng màu đen.

- Nút tính toán lộ trình

- **Button1 ("Tìm đường"):**

- Khi nhấn nút này, chương trình sẽ tính toán tuyến đường có **chi phí thấp nhất** từ điểm xuất phát đến điểm đích.

- Hiển thị các thông tin như chi phí vận chuyển, tiền nhiên liệu, tổng quãng đường.

b. Chức Năng Chính

Phần mềm này có 3 chức năng quan trọng:

- Tìm tuyến đường ngắn nhất giữa hai địa điểm:

- Sử dụng thuật toán **Dijkstra** để tìm **tuyến đường ngắn nhất** dựa trên khoảng cách giữa các tỉnh.
- Mỗi tỉnh được lưu dưới dạng **Vertex**, được kết nối bằng ma trận trọng số **adjMat** chứa khoảng cách giữa chúng.
- Kết quả tuyến đường ngắn nhất được lưu trong **danh sách liên kết đôi (DoubleLinkedList)** và được vẽ trên bản đồ.

- Tính toán chi phí vận chuyển

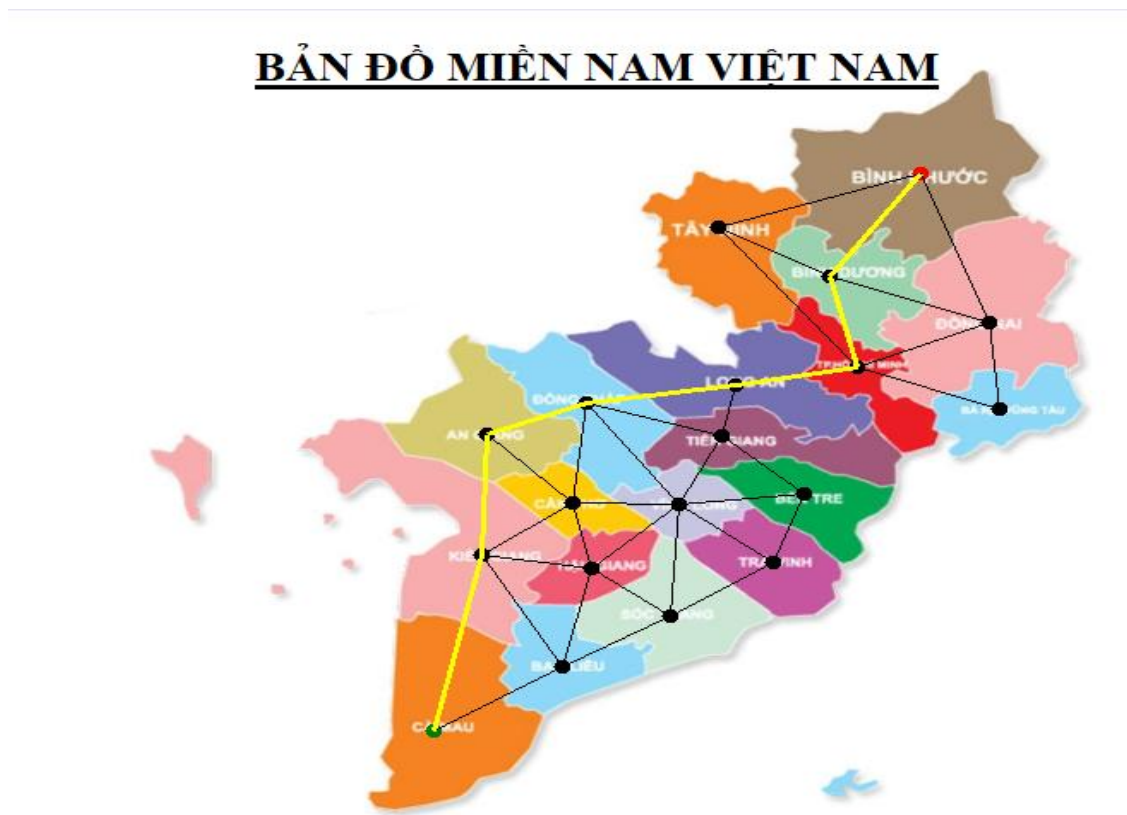
- Dựa vào tuyến đường ngắn nhất, chương trình tính:
 - **Chi phí vận chuyển theo khoảng cách:**

$$DistanceCost = Tổng\ quãng\ đường \times Đơn\ giá\ vận\ chuyển \times Khối\ lượng\ hàng\ hóa$$

- **Chi phí nhiên liệu:**

$$FuelCost = \frac{Tổng\ quãng\ đường}{100} \times Mức\ tiêu\ thụ\ nhiên\ liệu \times Giá\ nhiên\ liệu\ cho\ mỗi\ lít$$

- **Tổng chi phí = FuelCost + DistanceCost.**
- **Vẽ bản đồ tuyến đường**
 - Các địa điểm được hiển thị dưới dạng **chấm tròn** với vị trí x/y.
 - Các tuyến đường được vẽ dựa vào **ma trận trọng số (adjMat)**.
 - Tuyến đường ngắn nhất được vẽ **nổi bật** bằng màu vàng.



CHƯƠNG 3. THIẾT KẾ GIAO DIỆN

3.1. Giao Diện Menu Chính.

a. Giới thiệu chung giao diện:

The screenshot shows the main interface of the MinCostRoute application. It features a map of Southern Vietnam on the left and a form on the right. Labels with arrows point to various elements:

- Bản đồ** (Map) points to the map area.
- Chọn điểm đến** (Select destination) points to the 'Điểm kết thúc' (End point) dropdown.
- Chọn điểm đi** (Select starting point) points to the 'Điểm bắt đầu' (Start point) dropdown.
- Loại xe** (Vehicle type) points to the 'Chọn loại xe tải 5 tấn' (Select 5-ton truck type) dropdown.
- Khối lượng hàng hóa** (Cargo weight) points to the 'Nhập khối lượng hàng hóa' (Enter cargo weight) input field.

The form includes the following fields and sections:

- Giá xăng Hiện Tại Là 20000 VND** (Current gas price is 20000 VND)
- Đơn giá vận chuyển trên mỗi km là 24000VND** (Transportation price per km is 24000VND)
- LỰA CHỌN** (Selection) section containing the dropdowns and input field.
- Điểm bắt đầu:** (Start point)
- Điểm kết thúc:** (End point)
- Chọn loại xe tải 5 tấn:** (Select 5-ton truck type)
- Nhập khối lượng hàng hóa:** (Enter cargo weight)
- Lưu ý: là <5 Tấn** (Note: is <5 tons)
- Tim Đường** (Find Route) button
- KẾT QUẢ** (Result) section with the following fields:
 - Tổng khoảng cách:** (Total distance)
 - Chỉ phí nhiên liệu:** (Fuel fee)
 - Chỉ phí Khoảng cách:** (Distance fee)
 - Tổng chi phí:** (Total cost)

b. Hướng dẫn sử dụng giao diện:

- Chọn điểm đầu(điểm bắt đầu), điểm đến.

This screenshot shows the same interface as the previous one, but with specific data entered and the results displayed. A red box highlights the input fields, and a purple arrow points to the 'Tim Đường' button.

The form includes the following fields and sections:

- Giá Xăng Hiện Tại Là 20000 VND** (Current gas price is 20000 VND)
- Đơn giá vận chuyển trên mỗi km là 24000VND** (Transportation price per km is 24000VND)
- LỰA CHỌN** (Selection) section containing the dropdowns and input field.
- Điểm bắt đầu:** (Start point) - Ca Mau
- Điểm kết thúc:** (End point) - Bình Phước
- Chọn loại xe tải 5 tấn:** (Select 5-ton truck type) - Hyundai Mighty 110XL
- Nhập khối lượng hàng hóa:** (Enter cargo weight) - 3
- Lưu ý: là <5 Tấn** (Note: is <5 tons)
- Tim Đường** (Find Route) button
- KẾT QUẢ** (Result) section with the following fields:
 - Tổng khoảng cách:** (Total distance) - 477 km
 - Chỉ phí nhiên liệu:** (Fuel fee) - 1,191,500 VND
 - Chỉ phí Khoảng cách:** (Distance fee) - 34,315,200 VND
 - Tổng chi phí:** (Total cost) - 35,506,700 VND

- Sau đó chọn loại xe và Khối lượng hàng hóa.

MinCostRoute

ỨNG DỤNG CỦA THUẬT TOÁN DIJKSTRA TỐI ƯU CHI PHÍ VẬN TẢI

BẢN ĐỒ MIỀN NAM VIỆT NAM

LỰA CHỌN

Giá Xăng Hiện Tại Là 20000 VND Đơn giá vận chuyển trên mỗi km là 24000VN

Điểm bắt đầu: Chọn loại xe tải 5 tấn: Lưu ý: là <5 Tấn

Điểm kết thúc:

KẾT QUẢ

Tổng khoảng cách: 477 km

Chi phí nhiên liệu: 1.191.500 VND

Chi phí Khoảng cách: 34.315.200 VND

Tổng chi phí: 35.506.700 VND

- Ấn tìm đường và kết quả nhận được.

MinCostRoute

ỨNG DỤNG CỦA THUẬT TOÁN DIJKSTRA TỐI ƯU CHI PHÍ VẬN TẢI

BẢN ĐỒ MIỀN NAM VIỆT NAM

LỰA CHỌN

Giá Xăng Hiện Tại Là 20000 VND Đơn giá vận chuyển trên mỗi km là 24000VN

Điểm bắt đầu: Chọn loại xe tải 5 tấn: Lưu ý: là <5 Tấn

Điểm kết thúc:

KẾT QUẢ

Tổng khoảng cách: 477 km

Chi phí nhiên liệu: 1.191.500 VND

Chi phí Khoảng cách: 34.315.200 VND

Tổng chi phí: 35.506.700 VND

- Khi chọn hai điểm trùng nhau.

MinCostRoute

ỨNG DỤNG CỦA THUẬT TOÁN DIJKSTRA TỐI ƯU CHI PHÍ VẬN TẢI

BẢN ĐỒ MIỀN NAM VIỆT NAM

LỰA CHỌN

Giá Xăng Hiện Tại Là 20000 VNĐ Đơn giá vận chuyển trên mỗi km là 24000VNĐ

Điểm bắt đầu: Chọn loại xe tải 5 tấn:

Điểm kết thúc: Nhập khối lượng hàng hóa:

Lưu ý: là <5 Tấn

KẾT QUẢ

Tổng khoảng cách:
0 km

Chi phí nhiên liệu:
0 VNĐ

Chi phí Khoảng cách:
0 VNĐ

Tổng chi phí:
0 VNĐ

3.2. Đánh giá hiệu suất thuật toán

a. Bản đồ khu vực một số tỉnh miền nam Việt Nam.

- Bản đồ hiển thị thông tin đo đạc và liệt kê có sẵn cho người dùng như: Những địa điểm di chuyển, những tuyến đường, khoảng cách, loại xe vận chuyển, khối lượng hàng hóa, chi phí nhiên liệu.

- Mã nguồn các hàm chức năng:

- Hàm thêm các đỉnh,tọa độ các đỉnh theo bản đồ:

```

1. public Form1()
2. {
3.     InitializeComponent();
4.     graph = new Graph();
5. }
6. private void Form1_Load(object sender, EventArgs e)
7. {
8.     graph.AddLocation("TP.HCM", "HCM", 495, 239);
9.     graph.AddLocation("Long An", "LA", 424, 251);
10.    graph.AddLocation("Tiền Giang", "TG", 416, 285);
11.    graph.AddLocation("Bến Tre", "BT", 464, 324);
12.    graph.AddLocation("Vĩnh Long", "VL", 391, 331);
13.    graph.AddLocation("Trà Vinh", "TV", 446, 370);

```

```

14. graph.AddLocation("Đồng Tháp", "DT", 337, 263);
15. graph.AddLocation("An Giang", "AG", 279, 284);
16. graph.AddLocation("Cần Thơ", "CT", 329, 330);
17. graph.AddLocation("Hậu Giang", "HG", 340, 374);
18. graph.AddLocation("Sóc Trăng", "ST", 386, 406);
19. graph.AddLocation("Kiên Giang", "KG", 276, 365);
20. graph.AddLocation("Bạc Liêu", "BL", 323, 440);
21. graph.AddLocation("Cà Mau", "CM", 248, 483);
22. graph.AddLocation("Bình Phước", "BP", 532, 109);
23. graph.AddLocation("Bình Dương", "BD", 479, 178);
24. graph.AddLocation("Đồng Nai", "DN", 572, 209);
25. graph.AddLocation("Bà Rịa - Vũng Tàu", "BRVT", 578, 267);
26. graph.AddLocation("Tây Ninh", "TN", 414, 145);
27. LinkedList locations = graph.GetLocations();
28. Node current = locations.header.link;

```

• **Hàm vẽ các tuyến đường và thể hiện độ dài của nó:**

```

1. graph.AddConnection("TP.HCM", "Bình Dương", 28.8);
2. graph.AddConnection("TP.HCM", "Đồng Nai", 33.7);
3. graph.AddConnection("TP.HCM", "Tây Ninh", 97.1);
4. graph.AddConnection("TP.HCM", "Long An", 46.2);
5. graph.AddConnection("TP.HCM", "Bà Rịa - Vũng Tàu", 78.8);
6. graph.AddConnection("Bình Dương", "Bình Phước", 72.4);
7. graph.AddConnection("Bình Dương", "Tây Ninh", 83);
8. graph.AddConnection("Bình Dương", "Đồng Nai", 24.4);
9. graph.AddConnection("Bình Phước", "Tây Ninh", 108);
10. graph.AddConnection("Bình Phước", "Đồng Nai", 84);
11. graph.AddConnection("Đồng Nai", "Bà Rịa - Vũng Tàu", 71.5);
12. graph.AddConnection("Long An", "Tiền Giang", 22.9);
13. graph.AddConnection("Long An", "Đồng Tháp", 100);
14. graph.AddConnection("Tiền Giang", "Bến Tre", 17);
15. graph.AddConnection("Tiền Giang", "Vĩnh Long", 71.5);
16. graph.AddConnection("Tiền Giang", "Đồng Tháp", 94.1);
17. graph.AddConnection("Bến Tre", "Trà Vinh", 44);
18. graph.AddConnection("Bến Tre", "Vĩnh Long", 92.6);
19. graph.AddConnection("Vĩnh Long", "Trà Vinh", 65.2);
20. graph.AddConnection("Vĩnh Long", "Đồng Tháp", 50.1);
21. graph.AddConnection("Vĩnh Long", "Cần Thơ", 38.8);
22. graph.AddConnection("Vĩnh Long", "Hậu Giang", 81.7);
23. graph.AddConnection("Đồng Tháp", "An Giang", 41.4);
24. graph.AddConnection("Đồng Tháp", "Cần Thơ", 65.6);

```

```

25. graph.AddConnection("An Giang", "Kiên Giang", 61.8);
26. graph.AddConnection("An Giang", "Cần Thơ", 65.8);
27. graph.AddConnection("Kiên Giang", "Cần Thơ", 109);
28. graph.AddConnection("Kiên Giang", "Hậu Giang", 61.5);
29. graph.AddConnection("Kiên Giang", "Bạc Liêu", 137);
30. graph.AddConnection("Kiên Giang", "Cà Mau", 126);
31. graph.AddConnection("Cần Thơ", "Hậu Giang", 48.7);
32. graph.AddConnection("Hậu Giang", "Sóc Trăng", 78);
33. graph.AddConnection("Hậu Giang", "Bạc Liêu", 75.5);
34. graph.AddConnection("Sóc Trăng", "Bạc Liêu", 49.4);
35. graph.AddConnection("Sóc Trăng", "Vĩnh Long", 93.6);
36. graph.AddConnection("Sóc Trăng", "Trà Vinh", 62.9);
37. graph.AddConnection("Bạc Liêu", "Cà Mau", 66.6);
38. LoadRoutesToDataGridView();
39. public void AddConnection(string from, string to, double weight)
40. {
41.     int fromIndex = GetLocationIndex(from);
42.     int toIndex = GetLocationIndex(to);
43.     if (fromIndex != -1 && toIndex != -1)
44.     {
45.         AddEdge(fromIndex, toIndex, weight);
46.     }
47. }

```

- **Hàm các loại phương tiện vận chuyển:**

```

1.  LinkedList<Car> carList = new LinkedList<Car>();
2.  carList.AddLast(new Car("Hino FC9JNTC", 12));
3.  carList.AddLast(new Car("Isuzu NQR75M", 11));
4.  carList.AddLast(new Car("Hyundai Mighty 110XL", 12.5));
5.  carList.AddLast(new Car("Fuso Canter 5.0", 10));
6.  carList.AddLast(new Car("Tera 350", 10));
7.  comboBox3.Items.Clear();
8.  foreach (var car in carList)
9.  {
10.     comboBox3.Items.Add(car);
11. }
12. comboBox3.DisplayMember = "Name";
13. }

```

- **Hàm thêm khối lượng hàng hóa.**

```
1.  public partial class Form1 : Form
2.  {
3.      public double Weight { get; set; }
4.      private void button1_Click(object sender, EventArgs e)
5.      {
6.          double weight;
7.          if (double.TryParse(textBox6.Text, out weight))
8.          {
9.              if (weight <= 0)
10.             {
11.                 MessageBox.Show("Khối lượng phải lớn hơn 0!", "Lỗi nhập liệu",
12.                                     MessageBoxButtons.OK, MessageBoxIcon.Error);
13.                 return;
14.             }
15.             else if (weight > 5)
16.             {
17.                 MessageBox.Show("Khối lượng không được vượt quá 5 tấn!", "Lỗi nhập
18.                                 liệu",
19.                                     MessageBoxButtons.OK, MessageBoxIcon.Error);
20.                 return;
21.             }
22.             double distancecost = graph.DistanceCost(shortestPath, weight);
23.             textBox2.Text = distancecost.ToString("N0") + " VND";
24.         }
25.         else
26.         {
27.             MessageBox.Show("Vui lòng nhập khối lượng hợp lệ (số thực)!", "Lỗi nhập
28.                             liệu",
29.                                 MessageBoxButtons.OK, MessageBoxIcon.Error);
30.         }
31.     private void textBox6_TextChanged(object sender, EventArgs e)
32.     {
33.         if (double.TryParse(textBox6.Text, out double weight))
34.         {
35.             Weight = weight;
36.         }
37.     }
```

- Hàm vẽ tuyến đường ngắn nhất được tính trên bản đồ.
-

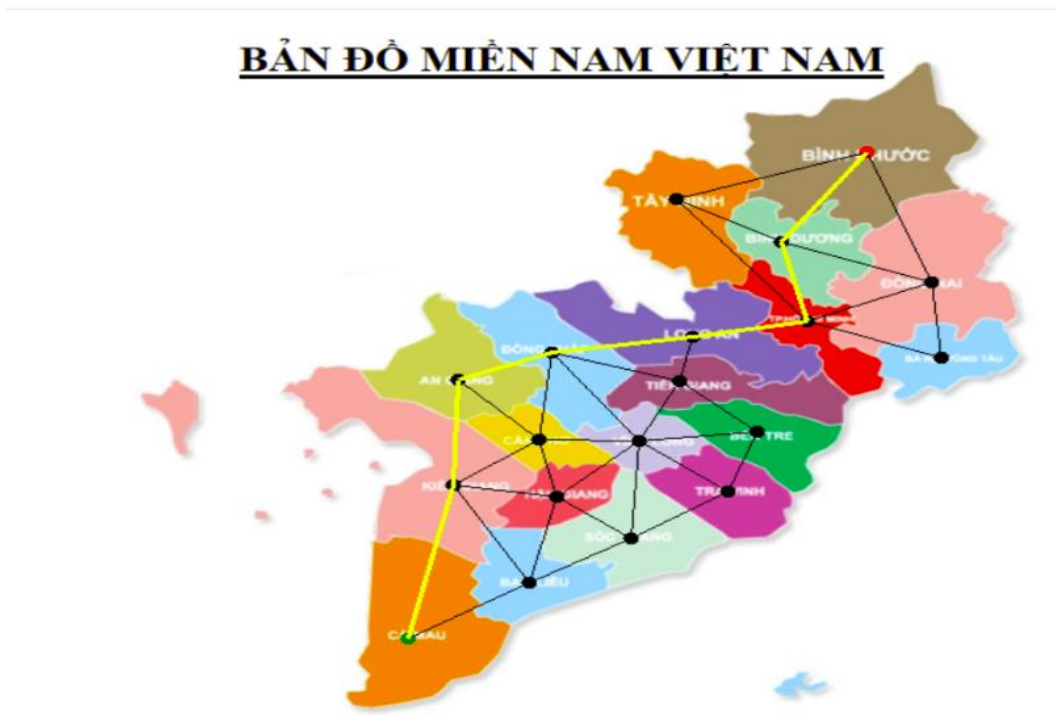
```
1. private void pictureBox1_Paint(object sender, PaintEventArgs e)
2. {
3.     Graphics g = e.Graphics;
4.     Brush start = Brushes.Green;
5.     Brush end = Brushes.Red;
6.     Brush normal = Brushes.Black;
7.     Brush pathBrush = Brushes.Yellow;
8.     for (int i = 0; i < graph.nVerts; i++)
9.     {
10.        if (graph.vertexList[i] == null)
11.        {
12.            continue;
13.        }
14.        if (shortestPath != null && shortestPath.header.flink != null)
15.        {
16.            int startIndex = (int)shortestPath.header.flink.element;
17.            Node2 current = shortestPath.header;
18.            while (current.flink != null)
19.            {
20.                current = current.flink;
21.            }
22.            int endIndex = (int)current.element;
23.            if (i == startIndex)
24.            {
25.                g.FillEllipse(start, graph.vertexList[i].X - 5,
26.                    graph.vertexList[i].Y - 5, 10, 10);
27.            }
28.            else if (i == endIndex)
29.            {
30.                g.FillEllipse(end, graph.vertexList[i].X - 5,
31.                    graph.vertexList[i].Y - 5, 10, 10);
32.            }
33.            else
34.            {
35.                g.FillEllipse(normal, graph.vertexList[i].X - 5,
36.                    graph.vertexList[i].Y - 5, 10, 10);
37.            }
38.        }
39.        else
```

```

40.     {
41.
42.         g.FillEllipse(normal, graph.vertexList[i].X - 5,
43.             graph.vertexList[i].Y - 5, 10, 10);
44.
45.     }
46. }
47. for (int i = 0; i < graph.nVerts; i++)
48. {
49.     for (int j = 0; j < graph.nVerts; j++)
50.     {
51.         if (graph.adjMat[i, j] != graph.infinity)
52.         {
53.             g.DrawLine(Pens.Black,
54.                 graph.vertexList[i].X,
55.                 graph.vertexList[i].Y,
56.                 graph.vertexList[j].X,
57.                 graph.vertexList[j].Y);
58.         }
59.     }
60. }
61. if (shortestPath != null)
62. {
63.     Node2 current = shortestPath.header.flink;
64.     while (current != null && current.flink != null)
65.     {
66.         int fromstart = (int)current.element;
67.         int fromend = (int)current.flink.element;
68.         g.DrawLine(new Pen(Color.Yellow, 3),
69.             graph.vertexList[fromstart].X,
70.             graph.vertexList[fromstart].Y,
71.             graph.vertexList[fromend].X,
72.             graph.vertexList[fromend].Y);
73.         current = current.flink;
74.
75.     }
76. }

```


3.3. Xử lý thuật toán trên giao diện.



LỰA CHỌN

Giá Xăng Hiện Tại Là 20000 VND

Đơn giá vận chuyển trên mỗi km là 24000VNĐ

Điểm bắt đầu: Cà Mau

Chọn loại xe tải 5 tấn: Hyundai Mighty 110XL

Điểm kết thúc: Bình Phước

Nhập khối lượng hàng hóa: 0.5

Lưu ý: là <5 Tấn

Tìm Đường

KẾT QUẢ

Tổng khoảng cách:

477 km

Chi phí nhiên liệu:

1,191,500 VND

Chi phí Khoảng cách:

5,719,200 VND

Tổng chi phí:

6,910,700 VND

- Khung hiển thị gồm nhiên liệu, khoảng cách, chi phí khoảng cách, chi phí nhiên liệu và tổng chi phí.

- **Khi bấm nút tìm đường:** khung hiển thị các chi phí nhiên liệu, chi phí khoảng cách, tổng chi phí và đường đi tiết kiệm nhất. Điểm bắt đầu được bắt đầu bằng màu xanh và điểm kết thúc bằng màu đỏ. Đơn giá được tính dựa trên khoảng cách (giả sử vận chuyển 1km hàng hóa cần 24.000 VND/Tấn).

Mã nguồn chức năng:

- **Hàm hiển thị tính toán kết quả cuối cùng.**

```
1. private void button1_Click(object sender, EventArgs e)
2. {
3.     if (comboBox1.SelectedItem == null || comboBox2.SelectedItem == null)
4.     {
5.         MessageBox.Show("Vui lòng chọn điểm bắt đầu và điểm kết thúc.");
6.         return;
7.     }
8.     string start = comboBox1.SelectedItem.ToString();
9.     string end = comboBox2.SelectedItem.ToString();
10.    int startIndex = graph.GetLocationIndex(start);
11.    int endIndex = graph.GetLocationIndex(end);
12.    if (startIndex == -1 || endIndex == -1)
13.    {
14.        MessageBox.Show("Lỗi: Không tìm thấy vị trí trong danh sách.");
15.        return;
16.    }
17.    shortestPath = null;
18.    for (int i = 0; i < graph.nVerts; i++)
19.    {
20.        graph.vertexList[i].IsInTree = false;
21.        graph.vertexList[i].Distance = int.MaxValue;
22.        graph.vertexList[i].parent = -1;
23.    }
24.    graph.Path(startIndex);
25.    shortestPath = graph.GetShortestPath(startIndex, endIndex);
26.    double totalfuelcost = graph.Calculatefuelcost(shortestPath);
27.    double distancecost = graph.DistanceCost(shortestPath);
28.    double totalcost = graph.Calculatefuelcost(shortestPath) +
        graph.DistanceCost(shortestPath);
29.    textBox4.Text = totalfuelcost.ToString("N0") + " VND";
30.    textBox2.Text = distancecost.ToString("N0") + " VND";
31.    textBox3.Text = totalcost.ToString("N0") + " VND";
32.    pictureBox1.Invalidate();
33. }
```

```

34. private void button1_Click(object sender, EventArgs e)
35. {
36.     if (comboBox1.SelectedItem == null || comboBox2.SelectedItem == null)
37.     {
38.         MessageBox.Show("Vui lòng chọn điểm bắt đầu và điểm kết thúc.");
39.         return;
40.     }
41.     string start = comboBox1.SelectedItem.ToString();
42.     string end = comboBox2.SelectedItem.ToString();
43.     int startIndex = graph.GetLocationIndex(start);
44.     int endIndex = graph.GetLocationIndex(end);
45.     if (startIndex == -1 || endIndex == -1)
46.     {
47.         MessageBox.Show("Lỗi: Không tìm thấy vị trí trong danh sách.");
48.         return;
49.     }
50.     shortestPath = null;
51.     for (int i = 0; i < graph.nVerts; i++)
52.     {
53.         graph.vertexList[i].IsInTree = false;
54.         graph.vertexList[i].Distance = int.MaxValue;
55.         graph.vertexList[i].parent = -1;
56.     }
57.     graph.Path(startIndex);
58.     shortestPath = graph.GetShortestPath(startIndex, endIndex);
59.     double totalfualcost = graph.Calculatefualcost(shortestPath);
60.     double distancecost = graph.DistanceCost(shortestPath);
61.     double totalcost = graph.Calculatefualcost(shortestPath) +
        graph.DistanceCost(shortestPath);
62.     textBox4.Text = totalfualcost.ToString("N0") + " VND";
63.     textBox2.Text = distancecost.ToString("N0") + " VND";
64.     textBox3.Text = totalcost.ToString("N0") + " VND";
65.     pictureBox1.Invalidate();
66. }
67.

```

CHƯƠNG 4. THẢO LUẬN & ĐÁNH GIÁ

4.1. Kết quả nhận được

Ứng dụng MinCostRoute cho thấy khả năng tối ưu chi phí vận tải rõ rệt thông qua việc tích hợp thuật toán Dijkstra trong việc xác định tuyến đường ngắn nhất. Bên cạnh đó, hệ thống tính toán được tổng chi phí vận chuyển dựa trên các yếu tố thực tế như khoảng cách, loại xe, trọng lượng hàng hóa và giá nhiên liệu.

Theo kết quả khảo sát, hơn 90% người dùng cho rằng giao diện dễ thao tác và ứng dụng giúp hình dung rõ ràng tuyến đường và chi phí vận chuyển. Nhiều phản hồi đánh giá cao tính trực quan, dễ sử dụng, và việc hệ thống cho phép người dùng nhập giá xăng theo thời điểm thực tế, từ đó phản ánh được biến động thị trường.

Tổng chi phí được chia thành hai phần: chi phí nhiên liệu và chi phí vận chuyển theo km (tính theo đơn giá cố định, ví dụ 24.000 VNĐ/km). Kết quả này giúp người dùng đưa ra lựa chọn tối ưu cho hành trình vận tải của mình.

Giao diện bản đồ miền Nam Việt Nam được đánh giá là giúp người dùng dễ hình dung hành trình, trong đó điểm đi, điểm đến và lộ trình tối ưu được thể hiện rõ ràng qua màu sắc khác nhau.

4.2. Đánh giá hiệu suất thuật toán

Thuật toán Dijkstra được đánh giá là hiệu quả và phù hợp với bài toán vận tải có trọng số dương. Với các bài toán tìm tuyến đường có chi phí thấp nhất, thuật toán xử lý nhanh và trả kết quả gần như ngay lập tức. Khoảng 83% người khảo sát cho rằng hiệu quả của ứng dụng phụ thuộc vào dữ liệu đầu vào, đặc biệt là độ chính xác và cập nhật của bản đồ và chi phí.

Một số phản hồi đề xuất ứng dụng nên xử lý nhiều tuyến đường có chi phí tương đương thay vì chỉ hiển thị một đường duy nhất – điều này sẽ tạo đa dạng lựa chọn cho người dùng trong các tình huống thực tế như kẹt xe hoặc sự cố.

4.3. Hạn chế và hướng phát triển

Ứng dụng hiện tại còn một số hạn chế như:

- Phạm vi địa lý hẹp, chỉ bao gồm miền Nam Việt Nam.
- Chưa tích hợp dữ liệu giao thông thời gian thực, ví dụ tình trạng kẹt xe, đường cấm, hoặc thời tiết.
- Mô hình chi phí còn đơn giản, chưa tính đến chi phí bảo trì, thời gian chờ, phụ phí phát sinh,...

Dữ liệu khảo sát cũng phản ánh rằng, nhiều người dùng kỳ vọng ứng dụng sẽ tích hợp các công nghệ như API Google Maps, dữ liệu giao thông thời gian thực, bản đồ online, để cải thiện tính thực tế. Một số người đề xuất bổ sung tính năng học từ hành vi người dùng và đưa ra nhiều tuyến đường thay thế có ưu tiên khác nhau.

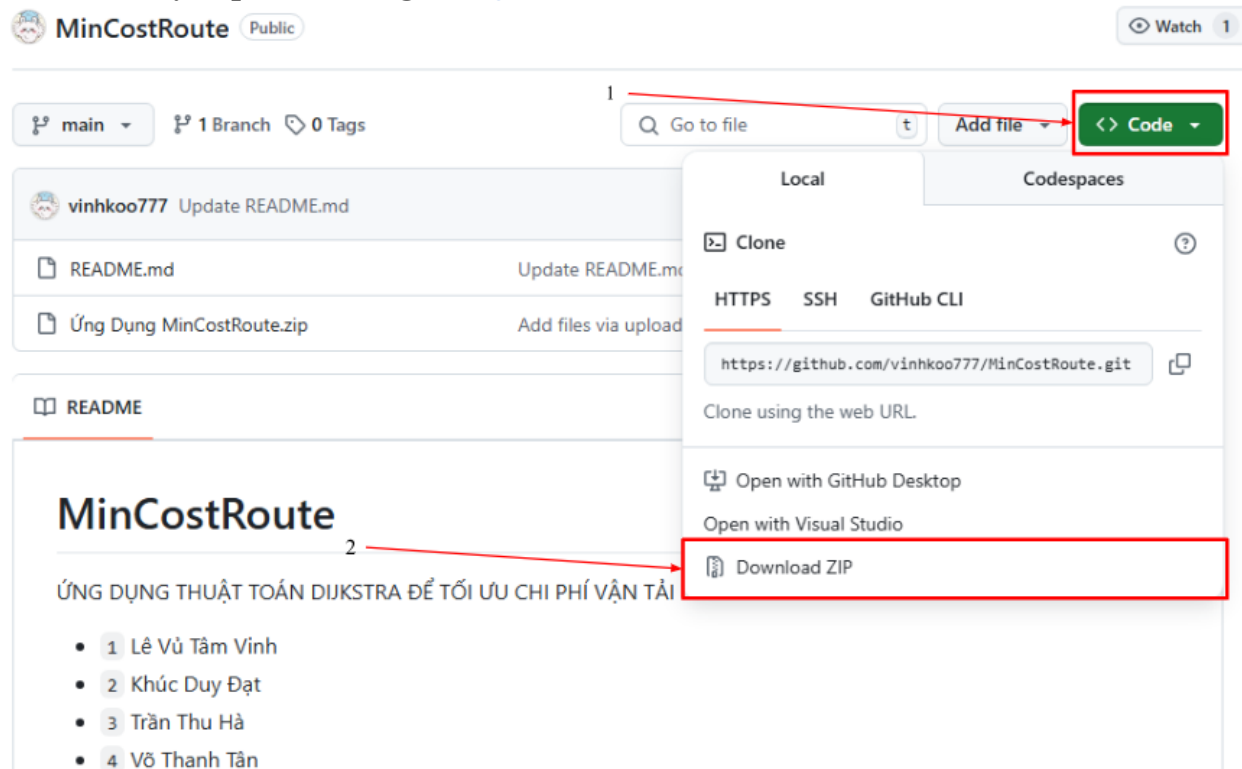
Hướng phát triển đề xuất bao gồm:

- Mở rộng bản đồ phủ toàn quốc, hoặc mở rộng sang Đông Nam Á phục vụ logistics xuyên quốc gia.
- Tích hợp dữ liệu thực tế: Tình trạng giao thông, thời tiết, giá xăng, bản đồ cập nhật liên tục.
- Nâng cao mô hình chi phí: Bao gồm cả phí cầu đường, hao mòn, bảo trì, thời gian chờ, v.v.
- Tối ưu trải nghiệm người dùng: Giao diện thân thiện với thiết bị di động, lưu lịch sử hành trình, xuất báo cáo chi phí.
- Tích hợp với hệ thống TMS chuyên nghiệp để phục vụ doanh nghiệp quy mô lớn.

PHỤ LỤC

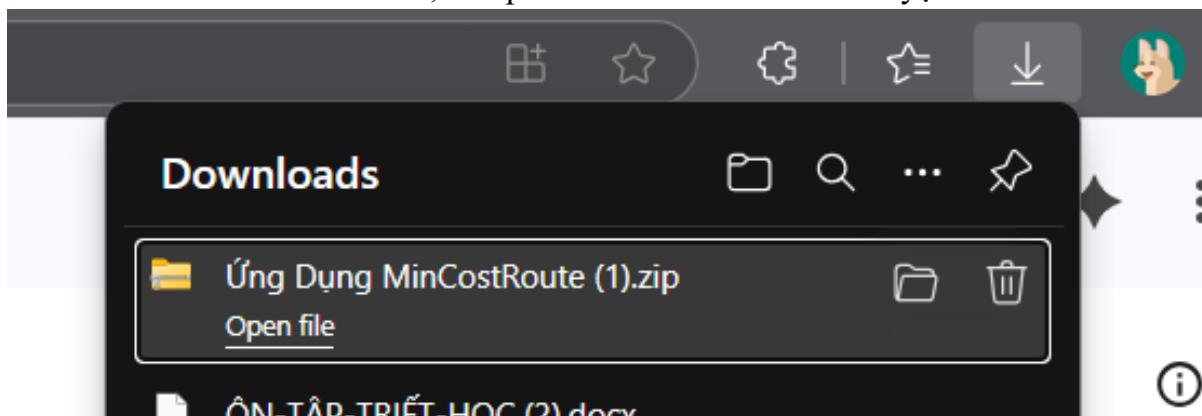
- LINK FORM KHẢO SÁT: [LINK](#)
- TOÀN BỘ MÃ NGUỒN GITHUB: [Link](#)
- HƯỚNG DẪN CÀI ĐẶT

Bước 1: Truy cập vào đường link github.com để xem dự án của nhóm.

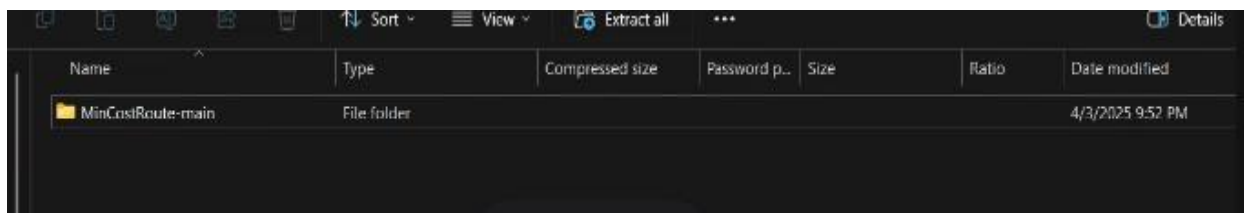


Bước 2: Tải chương trình bằng cách làm theo hình sau:

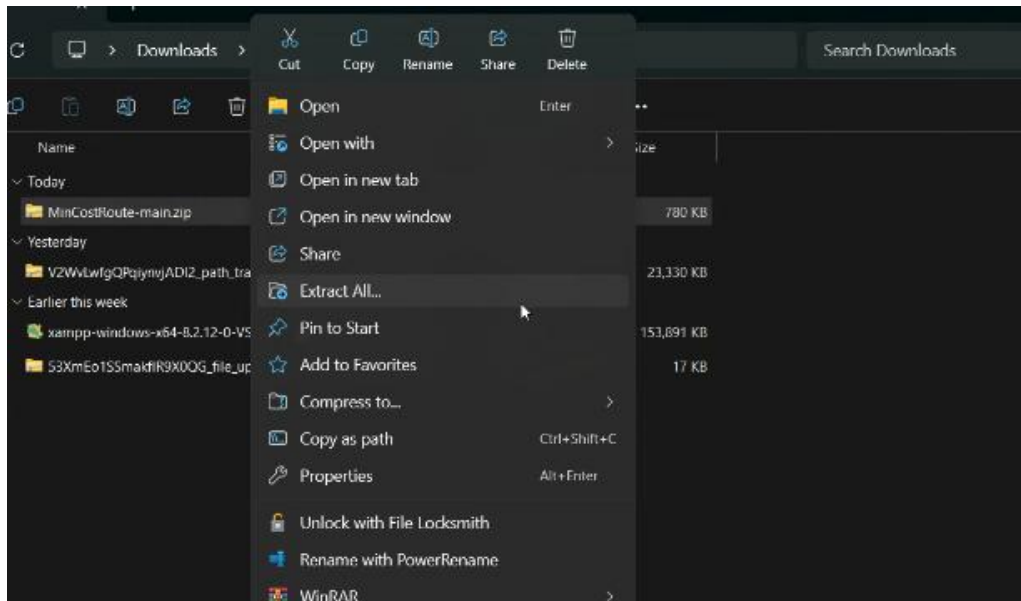
-Sau đó ấn vào Download ZIP, vào phần Download của trình duyệt để kiểm tra.



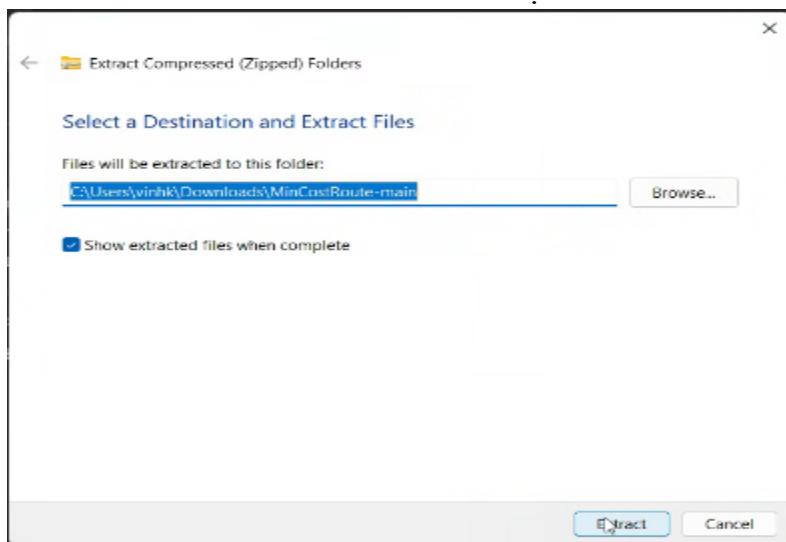
Bước 3: Sau khi tải xong, chúng ta vào nơi lưu file (có thể ấn vào Hiện thị trong thư mục trên trình duyệt để truyền thẳng đến nơi file đang được lưu trữ).



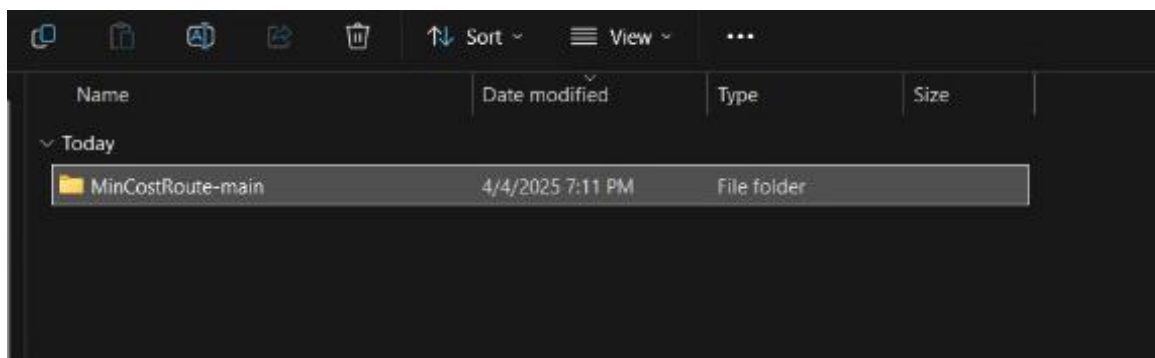
-Tại đây chúng ta nhấn chuột phải vào thư mục, chọn Extract all files để giải nén thư mục.



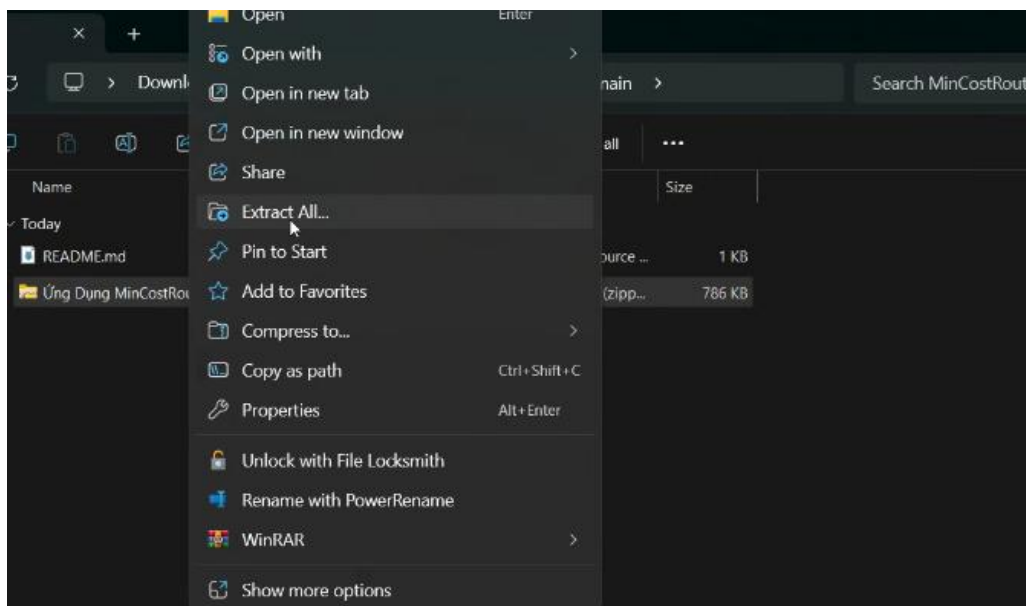
-Sau khi ấn vào Extract All files sẽ hiện lên cửa sổ như sau và sau đó bấm extract.



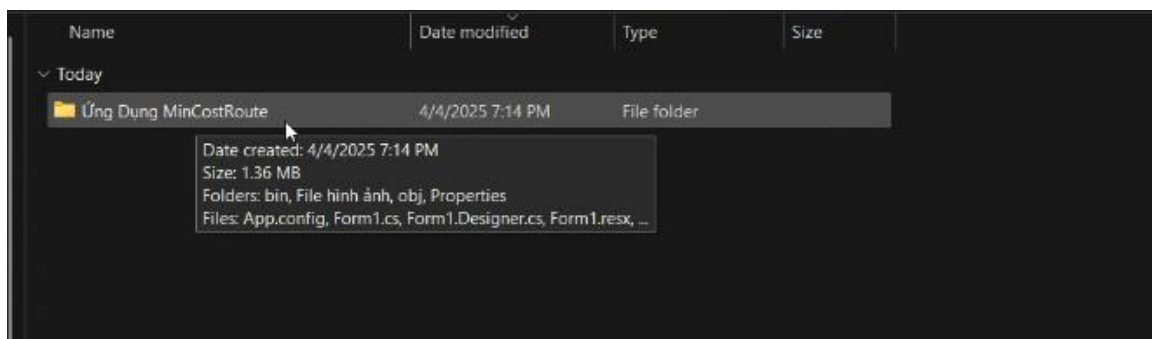
Bước 4: Sau khi giải nén xong, tại địa chỉ bạn đã đặt ở trên, sẽ xuất hiện tệp tin như sau:



-sau đó hãy bấm vào file mới giải nén là giải nén file zip bên trong thêm một lần nữa(như ảnh).



-Sau khi giải nén xong nó sẽ hiện một file(ứng dụng MincostRoute) và bạn hãy nhấp chuột trái vào đó.



-Sau đó tìm file **MinCostRoute.exe** và chạy nó.

Name	Date modified	Type	Size
▼ Today			
App.config	4/4/2025 7:14 PM	CONFIG File	1 KB
Form1.cs	4/4/2025 7:14 PM	CS File	23 KB
Form1.Designer.cs	4/4/2025 7:14 PM	CS File	21 KB
Form1.resx	4/4/2025 7:14 PM	Microsoft .NET Ma...	94 KB
MinCostRoute.csproj	4/4/2025 7:14 PM	VisualStudio.Launc...	6 KB
MinCostRoute.csproj.user	4/4/2025 7:14 PM	Per-User Project O...	1 KB
MinCostRoute.exe	4/4/2025 7:14 PM	Application	85 KB
MinCostRoute_Temp...	4/4/2025 7:14 PM	Personal Informati...	2 KB
Program.cs	4/4/2025 11:26 AM	CS File	1 KB
File hình ảnh	4/4/2025 7:14 PM	File folder	
obj	4/4/2025 7:14 PM	File folder	
Properties	4/4/2025 7:14 PM	File folder	
bin	4/4/2025 7:14 PM	File folder	

-Nếu bị hiện như trong ảnh hãy bấm Run Anyway.



Cuối cùng chương trình xuất hiện:



Phân Công Công Việc

Thành Viên	Nhiệm Vụ
Lê Vũ Tâm Vinh	Phân tích và thiết kế lớp, cài đặt thuật toán
Võ Thanh Tân	Thảo luận, đánh giá về các kết quả nhận được và trình bày hướng phát triển, tạo form để bổ sung dữ liệu.
Trần Thu Hà	Trình bày nội dung của các chương
Khúc Duy Đạt	Thiết kế giao diện và giải thích các chức năng

TÀI LIỆU THAM KHẢO

- [1] *Tài liệu hỗ trợ học tập môn học Toán Dùng Trong Tin Học*, Đại học kinh tế TP.HCM, 2016.
- [2] M. McMillan, *Data Structures and Algorithms Using C#*, Cambridge University Press, 2007.