

Bahria University,

Karachi Campus



COURSE: CSC-221 DATA STRUCTURES AND ALGORITHM
TERM: FALL 2024, CLASS: BSE- 3 (B)

Shortest Path Finder using Dijkstra Algorithm

Engr. Dr Sohaib/ Engr. Saniya Sarim

Signed

Remarks:

Score:

INTRODUCTION & PROBLEM

In the realm of transportation logistics, the efficiency of route planning plays a crucial role. Transport companies often face challenges in optimizing routes to reduce fuel consumption and travel time. The project aims to address these challenges by implementing a Shortest Path Finder using Dijkstra's Algorithm. The focus is on providing a user-friendly solution for route optimization based on latitude and longitude coordinates.

GROUP MEMBERS

- 1) ISRAR AYUB (02-131232-012) [TEAM LEAD]
- 2) MIR HAMZA (02-131232-057)
- 3) ABDUL WAHAB(02-131232-049)
- 4) TAHIR HAMEED(02-131232-106)

PARADIGMS

The development and implementation of the Shortest Path Finder using Dijkstra Algorithm in C# are guided by several key paradigms, each contributing to the overall success and effectiveness of the project.

• Efficiency:

- **Optimal Routing:** The primary objective is to devise an efficient system capable of determining optimal routes between multiple locations, significantly reducing travel time and fuel consumption for transportation companies.
- **Algorithmic Efficiency:** The implementation of Dijkstra's Algorithm ensures a time-efficient solution for finding the shortest paths within a network of locations, facilitating quick decision-making in real-time scenarios.

• User-friendliness:

- **Intuitive User Interfaces:** User interfaces are designed with a focus on simplicity and intuitiveness, allowing even non-technical users within transportation companies to input multiple locations seamlessly.
- **Accessibility:** The system accommodates users with varying levels of technical expertise, promoting widespread adoption and ensuring that route optimization is accessible to all stakeholders.

• Geospatial Awareness:

- **Precision through Coordinates:** Utilizing latitude and longitude coordinates ensures a high degree of accuracy in route planning, reflecting real-world geography and providing reliable distance estimations.
- **Global Applicability:** The system acknowledges the global nature of transportation networks, accommodating a wide range of locations and geographical settings.
- **Adaptability:**
 - **Dynamic Input Handling:** The system is designed to handle dynamic inputs, enabling transportation companies to add, modify, or remove locations on-the-fly, adapting to evolving logistical requirements.
 - **Scalability:** The architecture of the application is scalable, allowing it to accommodate varying scales of transportation networks without compromising performance or accuracy.
- **Integration:**
 - **Compatibility with External Systems:** The project is conceived with an awareness of the need for integration with other systems within transportation companies, fostering seamless data exchange and interoperability.
 - **API Support:** Provides robust Application Programming Interface (API) support, facilitating integration with third-party applications and services that transportation companies may already be utilizing.

These paradigms collectively shape the project, ensuring that it not only solves the immediate problem of route optimization but also aligns with broader principles of efficiency, user-centric design, adaptability, and integration within the complex landscape of transportation logistics.

ALGORITHM & EXPLANATION

Dijkstra's Algorithm:

Dijkstra's Algorithm is a graph search algorithm that finds the shortest path between nodes in a graph, which can represent a network of locations and connections. In our context, each location is a node, and connections between them are represented by weighted edges, where weights are indicative of travel distances.

The algorithm maintains a set of visited and unvisited nodes. It starts from the source node, exploring neighbouring nodes and updating their tentative distances from the source. This process continues until the destination is reached or all possible paths are explored.

ALGORITHM CODE

```

private void CalcCoordinates()
{
    _indexCount = Informations.GlobalPoints.Count();
    Informations.NumberOfElement = Informations.GlobalPoints.Count - 1;
    int[] lst = new int[Informations.NumberOfElement];
    for (int i = 0; i < Informations.NumberOfElement; i++)
    {
        lst[i] = i + 1;
    }
    ShortestAlgorithmByDijkstra(lst, 200);
    for (int i = 0; i < Informations.NumberOfElement; i++) {
        int[] tempList = new int[lst.Length];
        for (int j = 0; j < Informations.NumberOfElement; j++)
        {
            tempList[j] = Convert.ToInt32(_globalArray[i, j]);
        }
        ShortestAlgorithmByDijkstra(tempList, i);
    }
    for (int i = Informations.NumberOfElement; i < (Informations.NumberOfElement * Informations.NumberOfElement); i++)
    {
        int[] tempList = new int[Informations.NumberOfElement];
        for (int j = 0; j < Informations.NumberOfElement; j++)
        {
            tempList[j] = Convert.ToInt32(_globalArray[i, j]);
        }
        ShortestAlgorithmByDijkstra(tempList, i);
    }
    for (int i = 0; i < Informations.Count; i++)
    {
        int[] tempList = new int[Informations.NumberOfElement];
        for (int j = 0; j < Informations.NumberOfElement; j++)
        {
            tempList[j] = Convert.ToInt32(_globalArray[i, j]);
        }
        tempList = AddFunction(tempList, 0);
        for (int a = 0; a < tempList.Length; a++)
        {
            Informations.TotalArray[i, a] = tempList[a].ToString();
        }
    }
    Results();
}

private void PopFunction(int[] list, int[] valueList)
{
    for (int j = 0; j < list.Length; j++)
    {
        int temp = list[list.Length - 1];
        for (int i = list.Length - 1; i > 0; i--)
        {
            list[i] = list[i - 1];
        }
        list[0] = temp;
        for (int i = valueList.Length - 1; i >= 0; i--)
        {
            list = AddFunction(list, valueList[i]);
        }
        for (int i = 0; i < list.Length; i++)
        {
            _globalArray[Informations.Count, i] = list[i].ToString();
        }
    }
}

```

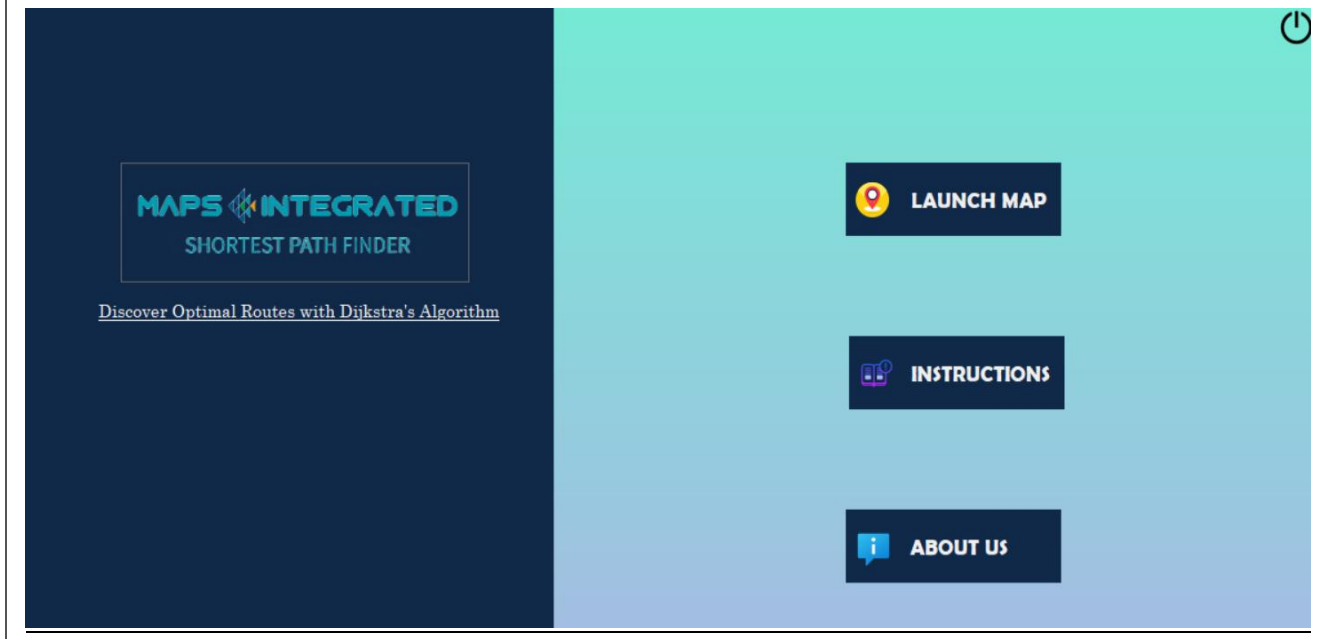
```

    }
    Informations.Count += 1;
    for (int i = 0; i < valueList.Length; i++)
    {
        list = PopElement(list);}} }
private void PopFunctionForZero(int[] list)
{
    for (int j = 0; j < list.Length; j++)
    {
        int temp = list[list.Length - 1];
        for (int i = list.Length - 1; i > 0; i--)
        {
            list[i] = list[i - 1]; }
        list[0] = temp;
        for (int i = 0; i < list.Length; i++)
        {
            _globalArray[j, i] = list[i].ToString(); }
        Informations.Count += 1;}}
private int[] AddFunction(int[] list, int value)
{
    int[] tempList = new int[list.Length + 1];
    int x = 1;
    for (int i = 0; i < list.Length + 1; i++) {
        if (i < x - 1)
            tempList[i] = list[i];
        else if (i == x - 1)
            tempList[i] = value;
        else
            tempList[i] = list[i - 1]; }
    return tempList;}
private int[] PopElement(int[] tempList) {
    tempList = tempList.Where((source, index) => index != 0).ToArray();
    return tempList;}
private void ShortestAlgorithmByDijkstra(int[] list, int id) {
    if (id != 200) {
        if (id < list.Length){
            int[] valueList = new int[1];
            for (int i = 0; i < valueList.Length; i++)
            {
                valueList[i] = list[i];
            }
            for (int i = 0; i < valueList.Length; i++) {
list = PopElement(list);
            }
            PopFunction(list, valueList); }
        else if (list.Length <= id && id < (list.Length * list.Length)) {
            int[] valueList = new int[2];
            for (int i = 0; i < valueList.Length; i++)
            {
                valueList[i] = list[i]; }
            for (int i = 0; i < valueList.Length; i++){
                list = PopElement(list); }
            PopFunction(list, valueList);}} }
        else if (id == 200) {
            PopFunctionForZero(list); } }

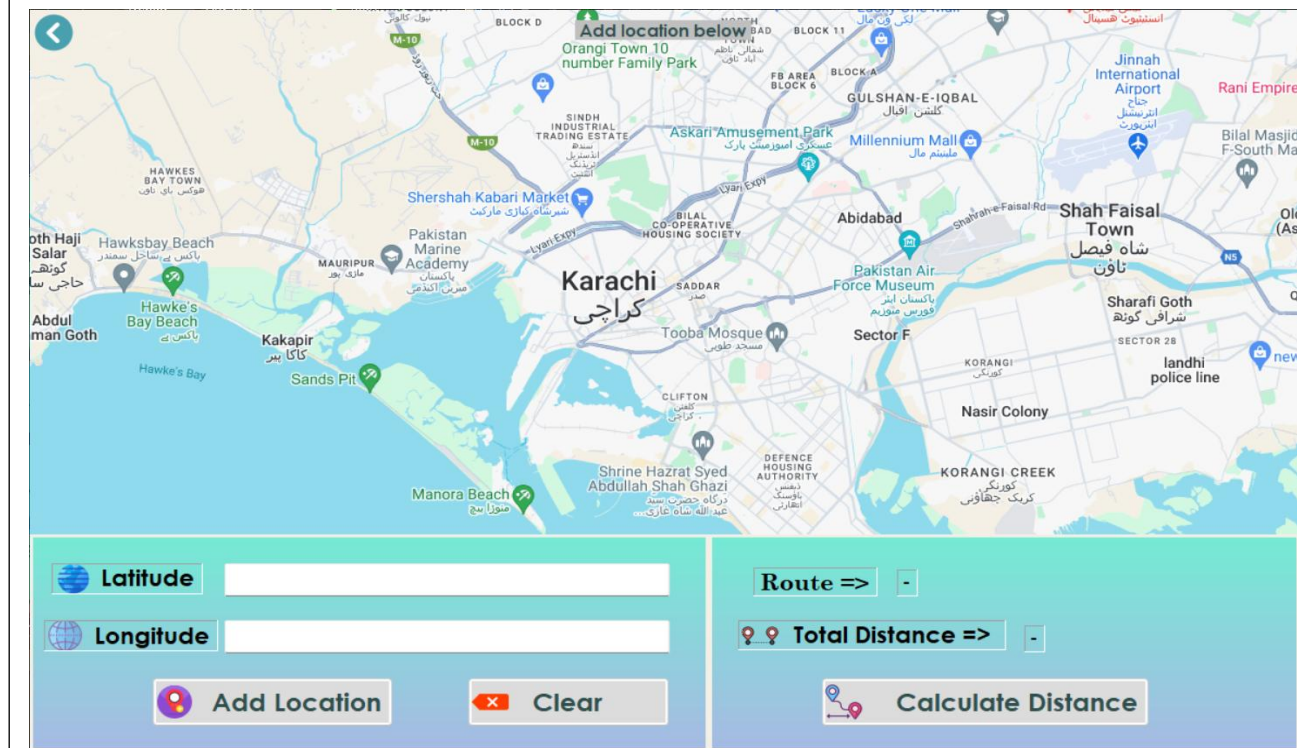
```

INTERFACES


Main Page



MAPS PAGE



INSTRUCTIONS PAGE



INSTRUCTIONS

Add Location:
Enter longitude and latitude for the first location.
Press "Enter Location" to add a pin at the specified coordinates.
Latitude and Longitude cannot be in String

Multiple Locations:
Continue addin as many locations as needed by repeating the process.

Clear Coordinactes:
Use the "Clear" button to remove the longitude and latitude for the current location.

Calcuate Distance:
Click on "Total Distance" to calculate and display the total distance between all added locations.

ABOUT US



ABOUT US

Welcome to our project, a collaborative effort spearheaded by a talented team of individuals committed to innovation and excellence. This project is made by four visinaries : Israr Ayub , Mir Hamza, Abdul Wahab,Tahir hameed.

Module Distribution:

Frontend: Designed with precision and creativity by Abdul wahab , ensuring an intuitive and visually appealing user interface.
Algorithm ImplementationSkillfully executed by : Israr ayub and Mir Hamza , incorporating cutting-edge algorithms for optimal performance.
Map Integration: Meticulously handled by Tahir Hameed , integrating maps seamlessly into the project for enhanced functionality.

CONCLUSION

The Shortest Path Finder using Dijkstra's Algorithm in C# offers a powerful solution for transportation companies to optimize their routes. The utilization of geospatial data ensures accuracy in route planning. The project, by facilitating multiple location inputs, caters to the dynamic needs of transport logistics, contributing to overall efficiency and cost-effectiveness.

This project represents a significant step toward enhancing route optimization capabilities for transportation companies, leveraging the strengths of Dijkstra's Algorithm and providing a user-friendly interface for practical implementation.
