



# Fastest Route for Public Health Center in Bandung with Dijkstra Algorithm and FP-Growth Recommendation in C++ Programming Language

Andika Eka Kurnia<sup>1</sup>, Gregorius Christian Sunaryo<sup>2</sup>, Muhammad Rafi Zamzami<sup>3</sup>, Naila Melany<sup>4</sup>, Rafi Nazhmi Nugraha<sup>5</sup>, Rahma Dina Ariyanti<sup>6</sup>

<sup>1,2,3,4,5,6</sup>Software Engineering Study Program, Universitas Pendidikan Indonesia, Indonesia

Correspondence: E-mail: [andika.eka.kurnia@upi.edu](mailto:andika.eka.kurnia@upi.edu)

## ABSTRACT

This research addresses the need for efficient public health center access in Bandung. In this case study, the solution to that specific problem is implemented using the C++ programming language. Consequently, extreme programming is chosen as the research methodology. We implemented Dijkstra's algorithm to find the shortest routes and the FP-Growth algorithm to recommend frequently visited health centers based on previous history. By leveraging C++ for its performance advantages, the system maps urban villages and calculates optimal paths. Additional features manage village data, create routes, and display health center information. The combined use of these algorithms enhances navigation and healthcare access in urban settings, though future work should consider real-time traffic conditions.

© 2021 Kantor Jurnal dan Publikasi UPI

## ARTICLE INFO

### Article History:

Submitted/Received 04 Mar 2024

First Revised 01 Apr 2024

Accepted 29 Apr 2024

First Available online 07 May 2024

Publication Date 05 June 2024

### Keyword:

Graph,

Dijkstra Algorithm,

FP-Growth Algorithm,

C++ Programming Language,

Public Health Center in Bandung.

## 1. INTRODUCTION

Access to fast and efficient basic healthcare services is an urgent need for the community, especially in large cities like Bandung. The city faces complex traffic challenges, where long travel times and traffic uncertainties can hinder access to healthcare services (Syed, et al, 2014), which in this case is the Community Health Centers (Puskesmas). This situation becomes even more critical in emergencies, where delays can be fatal (Czeisler, et al. 2020). Therefore, it is crucial to find a solution that ensures the community can reach the nearest Puskesmas quickly and efficiently.

The specific problem at hand is determining the fastest route to Puskesmas in each district of Bandung, considering the complexity of the city's road network. Additionally, there is a need for a system that can recommend relevant destination healthcare services based on previous usage patterns.

To address these challenges, this research proposes the use of two algorithms: Dijkstra and FP-Growth, implemented in the C++ programming language. The Dijkstra algorithm is chosen for its effectiveness in finding the shortest path in a graph network (Sipayung, et al. 2023), making it highly suitable for determining the fastest route in a city with numerous nodes and paths like Bandung. Meanwhile, the FP-Growth algorithm will be used to provide recommendations based on previous fastest route history (Anwar, et al. 2023), ensuring that users receive the most visited public health center .

The Dijkstra algorithm is used to find the shortest path from a single source node to all other nodes in a weighted graph with non-negative weights. Designed by Edsger W. Dijkstra, this algorithm is highly effective in solving shortest path problems. The use of Dijkstra's algorithm is extensive in various programming applications (Goodrich et al., 2011). For instance, this algorithm helps determine the shortest route for data transmission from one node to another. In GPS navigation, Dijkstra's algorithm is used to find the fastest route from one location to another. Additionally, this algorithm is beneficial in games and simulations for calculating the shortest path on a map, as well as in transportation systems to optimize travel routes and public transport schedules (Taufikurrachman 2020).

Implementing Dijkstra's algorithm in the C++ programming language offers high performance and strong control over memory usage and other resources, allowing the algorithm to run faster and more efficiently. C++ also provides flexibility in code development and maintenance, as well as integration with various libraries and development tools (Goodrich et al., 2011). Therefore, using Dijkstra's algorithm in C++ can optimize paths and routes in various practical applications.

## 2. METHODS

### 2.1. Research Method

The research method used in this research is Extreme Programming (XP), Extreme Programming is a method for developing software that focuses on coding activities in each software development cycle performed (Roseandree et al., 2023). The extreme programming method not only focuses on coding but also aspects of software development as a whole. According to Septiani and Habibie (2022), the XP method has several stages including:

1. Planning

At this stage the activity carried out is planning, such as identifying and analyzing problems and also the needs of both users and systems.

2. Design

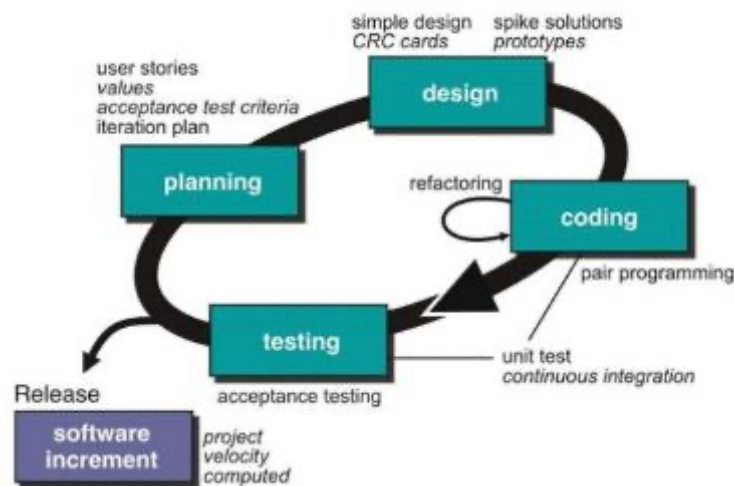
After analyzing and identifying problems and needs, the next step is to create a system design using system modeling and architectural modeling with diagrams.

3. Coding

At this stage, implementing the system design that has been made into the programming language.

4. Testing

At this stage, testing is carried out on the functionality of the system to see if it is in accordance with expectations and to see if there are bugs or errors when the system is running.



**Figure 1.** Extreme Programming Method (XP)

## 2.2 Research Step

By using the extreme programming (XP) method, there are several stages carried out in this study as shown in **Figure 1**, where the stages are in accordance with the extreme programming (XP) method.

1. Planning

At this step, researchers analyze and identify problems related to the distance to health institutions (public health centre). How to get through the path with the shortest distance. Next, make a plan to build a system that consists of several features, such as adding villages and health centres, viewing a list of health centers, adding routes, finding the fastest path, and viewing route search history.

2. Design

The design is made by drawing a graph on a map of the North Bandung Regency section, then searching for data to obtain the name of the village, health center as vertex, and distance as weight or edge, then designing the fastest path search using the dijkstra algorithm.

3. Coding

The programming language used is C++ by using two algorithms for the main features. first using the dijkstra algorithm to determine the fastest path and the fp-growth algorithm to provide recommendations for the destination of the health center to be visited.

4. Testing

After the system has been built, unit testing is carried out for each functionality to ensure that all features or programmes run properly and as expected and to check whether there are bugs or errors when running the programme.

### 3. RESULTS AND DISCUSSION

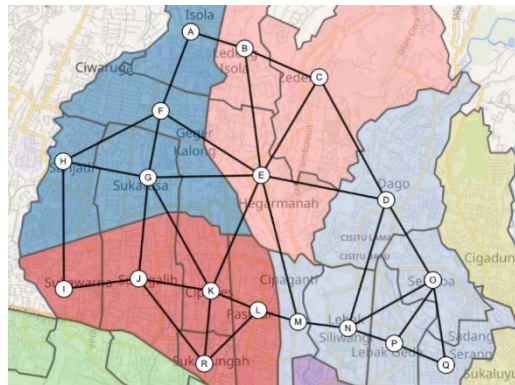
#### 3.1. Data Scope

In this study, several locations were used as samples, as shown in **Table 1**. These locations include four districts and eighteen urban villages in Bandung, Indonesia. These sample locations were chosen to represent areas with a Public Health Centre, allowing us to determine the shortest route to reach them.

**Table 1** Data Scope Location Detail

No	District	Urban Village	Public Health Center	Code
1	Sukasari	Isola	Puskesmas Ledeng	A
		Gegerkalong	UPT Puskesmas Sukarasa	F
		Sukarasa	None	G
		Sarijadi	Puskesmas Sarijadi	H
2	Cidadap	Ledeng	Puskesmas Cipaku	B
		Ciumbuleuit	Puskesmas Dadali Ciumbuleuit	C
		Hegarmanah	Puskesmas Ciumbuleuit	E
3	Sukajadi	Sukawarna	Puskesmas Sukawarna	I
		Sukagalih	UPT Puskesmas Sukagalih	J
		Cipedes	UPTD Puskesmas Sukajadi	K
		Sukabungah	None	R
		Pasteur	None	L
4	Coblong	Dago	Puskesmas Dago	D
		Cipaganti	None	M
		Lebak Siliwangi	None	N
		Lebak Gede	Puskesmas Sekeloa	P
		Sekeloa	None	O
		Sadang Serang	UPTD Puskesmas Puter	Q

After some research regarding those locations, the following map **Figure 2** represents the existing urban villages and, if applicable, their corresponding Public Health Centers, represented by their codes as shown previously in **Table 1**. The center points are directly at the public health center in those urban villages. If there is no public health center at those urban villages, the center point in Google Maps was chosen. The distance is measured from Google Map shortest route by walking or motorcycle mode between those points.



**Figure 2.** Locations of Urban Villages and the Connected Edges

### 3.2. Graph Representation

Graph is one of the hierarchical data structures used to describe the relationship between pairs of objects or vertices. In a graph, there are vertices and edges. In the program to determine the fastest route to the health center, researchers use a weighted and directed graph. A directed and weight graph is a graph that has direction and value on each (edge). In this case, the vertex is the name of the urban village with the corresponding public health center and district while the edge is the distance in meters (m). **Figure 3** below shows the pseudocode for initializing a structure to create a single vertex and edge instance.

```

Define Structure Vertex
String district
String urbanVillage
String publicHealthCenter

Function Constructor(district, urbanVillage, publicHealthCenter)
Set this.district to district
Set this.urbanVillage to urbanVillage
Set this.publicHealthCenter to publicHealthCenter
End Function
End Structure

Define Structure Edge
Vertex firstVertex
Vertex secondVertex
Long Integer distance

Function Constructor(firstVertex, secondVertex, edgeDistance)
Set this.firstVertex to firstVertex
Set this.secondVertex to secondVertex
Set this.distance to edgeDistance
End Function
End Structure
    
```

**Figure 3.** Pseudocode for Vertex and Edge Structures

**Table 2** Graph Representation

Data Type	Properties	Example Value
vector<Vertex *>	Vertices	{ {Sukasari, Puskesmas Ledeng, Isola}, ... }
vector<Edge *>	Edges	{ {Gegerkalong, Isola, 800} ... }

The graph is represented as a set of vertices and edges with table-like structures, as shown in **Table 2** and **Figure 4**. This graph representation is easy to read due to its declarative nature when creating vertices and edges, which differs from the usual matrix representation. Not only that, this representation makes it easy to implemented related algorithms like Dijkstra and FP-Growth, which are covered in the next section.

```

Define Structure Graph
  List of Vertex pointers called vertices
  List of Edge pointers called edges
End Structure

```

**Figure 4.** Pseudocode for Graph

### 3.3. Dijkstra's Algorithm Implementation

The dijkstra algorithm is implemented in C++ using map and pair structure to represent each vertex and the corresponding weight or distance as shown in **Figure 5**. The first step in this algorithm is to calculate the shortest distance between the starting vertex and destination vertex. The second step is backward to get all the vertices in its map and pair structure.

```

Function findShortestPath(startVertex, endVertex):
  Initialize shortestDistance as a map from Vertex to (Vertex, Integer)
  Initialize visitedVertices as an empty list
  For each vertex in this.vertices:
    Set shortestDistance[vertex] to (null, INFINITY)

  Set currentVertex to startVertex
  Set shortestDistance[currentVertex] to (currentVertex, 0)
  While currentVertex is not endVertex:
    Append currentVertex to visitedVertices
    Set currentEdges to this.getEdges(currentVertex)
    For each edge in currentEdges:
      Set destinationVertex to edge.secondVertex
      Set distance to shortestDistance[currentVertex].second + edge.distance

      If distance < shortestDistance[destinationVertex].second:
        Set shortestDistance[destinationVertex] to (currentVertex, distance)

  Set minDistance to INFINITY
  For each vertex in this.vertices:
    If shortestDistance[vertex].second < minDistance and
      shortestDistance[vertex].first is not vertex and
      vertex is not in visitedVertices:
      Set minDistance to shortestDistance[vertex].second

```

```
Set currentVertex to vertex

If minDistance is INFINITY:
  Initialize emptyGraph as a new Graph
  Return (emptyGraph, INFINITY)

Set totalDistance to shortestDistance[endVertex].second
Initialize shortestPath as a new Graph

While currentVertex is not startVertex:
  shortestPath.addVertex(currentVertex)
  Set previousVertex to shortestDistance[currentVertex].first
  For each edge in this.edges:
    If (edge.firstVertex is currentVertex and edge.secondVertex is previousVertex) or
      (edge.firstVertex is previousVertex and edge.secondVertex is currentVertex):
      shortestPath.addEdge(currentVertex, previousVertex, edge.distance)
      Break

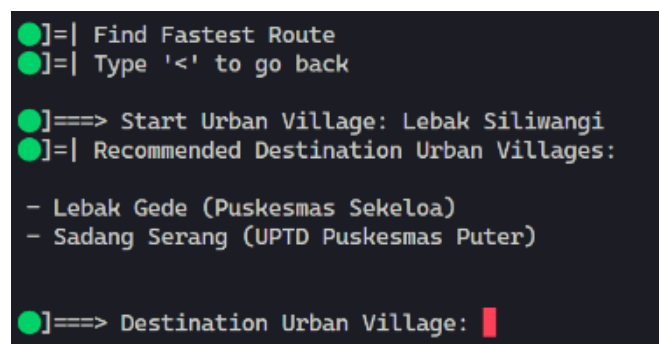
  Set currentVertex to previousVertex

shortestPath.addVertex(startVertex)
shortestPath.reverseEdges()
Return (shortestPath, totalDistance)
End Function
```

Figure 5. Dijkstra's Algorithm Pseudocode

### 3.4. FP-growth Implementation

The FP-Growth recommendation is implemented in the fastest route feature when users input the start urban village. The transaction data is calculated from the fastest route history in the history.json file every time users enter the start urban village. After the frequent pattern generated, the system selects all items with the highest frequent pattern and recommends them to the users as shown in **Figure 6**. If none are selected, skip the recommendation and continue to the destination urban village input. The complete source code for this implementation could be found on this github <https://github.com/dikdns/fastest-rute-public-health-center>.



```
●]=| Find Fastest Route
●]=| Type '<' to go back

●]===> Start Urban Village: Lebak Siliwangi
●]=| Recommended Destination Urban Villages:

- Lebak Gede (Puskesmas Sekeloa)
- Sadang Serang (UPTD Puskesmas Puter)

●]===> Destination Urban Village: █
```

Figure 6. Recommendation Terminal Screen

### 3.5. Extended Features

#### 3.5.1. Add Urban Villages

The Add Urban Villages feature is a process for managing the data of urban villages in the system. This feature helps to ensure that the data is up-to-date, structured, and permanently stored, thus supporting various operations related to neighborhoods in the system.

This feature consists of several main steps:

- Requesting User Input: The user is asked to enter the information of the new neighborhood, including the name of the sub district and neighborhood, as well as the name of the public health center (if applicable).
- Creating a New Vertex Object: Based on the information entered by the user, a new Vertex object is created to represent the new neighborhood.
- Add to Main Graph: The new Vertex is added to the main Graph called mainRoute. The Graph stores information about the structure and relationships between neighborhoods in the system.
- Saving Data to JSON File: The new neighborhood data is saved to a JSON file called districts.json for permanent storage.

This feature also has several benefits:

- Updating Urban Village Data: This feature allows system administrators to add new neighborhoods and update existing neighborhood information, ensuring that the neighborhood data in the system is always up-to-date.
- Maintaining System Structure: Adding new neighborhoods to the main Graph maintains the structure and relationships between neighborhoods in the system, allowing for various operations and data analysis involving those neighborhoods.
- Permanent Data Storage: Saving neighborhood data to the JSON file districts.json ensures that neighborhood data is permanently stored and can be accessed by the system in the future.

### 3.5.2. Create Route

The Create Route feature allows users to easily build a network of connections between health centers. First, the system collects the name of the starting health center from the user and then retrieves the name of the destination health center. The user is asked to enter the distance between the two health centers in meters. Next, a new Edge object is created to connect the two newly created Vertex objects, and the distance entered by the user is assigned to the Edge object. This new Edge object, representing the connection between the health centers, is inserted into the main routes. Finally, all route data, including health center names, distances, and associated Edge objects, are stored in a JSON file named "routes.json," which serves as a repository for the network of connections between health centers.

### 3.6. Demo Scenarios

```

●]=| Fastest Route Public Health Center
●]=| Menu:
●]=| 1. Add Public Health Center
●]=| 2. View Public Health Center
●]=| 3. Add Route
●]=| 4. Find Fastest Route
●]=| 5. View Fastest Route History
●]=| 6. Exit
●]==> Choose menu: █

```

**Figure 7.** Main Menu

In the scenario where the user starts in the Pasteur urban village, the program initiates by prompting the user to select the "Find Fastest Route" option, as illustrated in **Figure 7**. Upon selection, the user is prompted to input the starting urban village, designated as "Pasteur".



The program then utilizes FP-Growth algorithms to recommend a destination, as depicted in **Figure 8**. In this instance, the suggested destination is "Cipedes (UPTD Puskesmas Sukajadi)", based on predefined common routes stored in the history.json file.

```
●]=| Find Fastest Route
●]=| Type '<' to go back

●]==> Start Urban Village: Pasteur
●]=| Recommended Destination Urban Village:
Cipedes (UPTD Puskesmas Sukajadi)

●]==> Destination Urban Village: █
```

**Figure 8.** Demo Scenario: Recommendation Based on User Location in Pasteur

The user has the option to either accept this recommendation or enter a different destination. After entering the destination, as shown in **Figure 9**, the program utilizes Dijkstra's algorithm to determine the shortest path between Pasteur and the selected destination. In this particular case, the algorithm computes that the shortest distance between Pasteur and Cipedes is approximately 750 meters.

```
●]=| Find Fastest Route
●]=| Type '<' to go back

●]==> Start Urban Village: Pasteur
●]=| Recommended Destination Urban Village:
Cipedes (UPTD Puskesmas Sukajadi)

●]==> Destination Urban Village: Cipedes

+Pasteur -> +UPTD Puskesmas Sukajadi (Cipedes)

Total Distance: 750 meter

●]=| Fastest route has been saved to history

●]==> Press enter to continue... █
```

**Figure 9.** Demo Scenario Results for User Location in Pasteur

In another scenario where the user is located on Tubagus Ismail Street within the Sekeloa Urban Village, similar to the previous scenario, the user inputs the starting urban village as "Sekeloa". The program then recommends "Lebak Gede (Puskesmas Sekeloa)" as the destination. After the user enters the destination, the program calculates the shortest distance, which is 800 meters, between Sekeloa and Lebak Gede.

```
●]=| Find Fastest Route
●]=| Type '<' to go back

●]==> Start Urban Village: Sekeloa
●]=| Recommended Destination Urban Village:
Lebak Gede (Puskesmas Sekeloa)

●]==> Destination Urban Village: Lebak Gede

+Sekeloa -> +Puskesmas Sekeloa (Lebak Gede)

Total Distance: 800 meter

●]=| Fastest route has been saved to history

●]==> Press enter to continue... █
```

**Figure 10.** Demo Scenario Results for User Location in Sekeloa

#### 4. CONCLUSION

This research shows that the combination of the two algorithms, namely Dijkstra and FP-Growth, is very effective in creating a system that is not only optimal in terms of determining the fastest route to the health center but also intelligent in providing recommendations for health center destinations based on the previous history. The results of this research can be implemented to improve health services and navigation efficiency in an urban context. However, it should be underlined that there are several other factors besides distance that affect the fastest path, these other factors can be road conditions and situations.

#### 5. AUTHORS' NOTE

The authors declare that there is no conflict of interest regarding the publication of this article. Authors confirmed that the paper was free of plagiarism.

#### 6. REFERENCES

- Anwar, B., Ambiyar, and Fadhliah. (2023). Application of the FP-Growth method to determine drug sales patterns. *Sinkron: Jurnal dan Penelitian Teknik Informatika*, 7(1).
- Czeisler, M. É., Marynak, K., Clarke, K. E., et al. (2020). Delay or avoidance of medical care because of COVID-19–related concerns — United States, June 2020. *MMWR Morbidity and Mortality Weekly Report*, 69(1250–1257).
- Goodrich, M. T., Tamassia, R., and Mount, D. M. (2011). *Data structures and algorithms in C++*. John Wiley & Sons.
- Roseandree, B. S., Mulyana, A. J., Fuji, R., Pratama, A. H., and Purnama, P. Searching for the Fastest Route to Tourist Attractions with the Kruskal Algorithm in the C++ Programming Language. *Journal of Software Engineering, Information and Communication Technology (SEICT)*, 4(2), 139-150.
- Septiani, N. A., and Habibie, F. Y. (2022). Penggunaan Metode Extreme Programming Pada Perancangan Sistem Informasi Pelayanan Publik. *Jurnal Sistem Komputer dan Informatika*, 3(3), 341-349.
- Sipayung, L. Y., Sinaga, C. R., and Sagala, A. C. (2023). Application of Dijkstra's algorithm to determine the shortest route from city center to Medan city tourist attractions. *Journal of Computer Networks, Architecture and High Performance Computing*, 5(2).
- Srikant, R., and Agrawal, R. (1995). Mining generalized association rules.
- Syed, S. T., Gerber, B. S., and Sharp, L. K. (2013). Traveling towards disease: transportation barriers to health care access. *Journal of community health*, 38(5), 976–993.
- Taufikurrachman, H., Huda, M. N., Satrio P. P, M. R., and M, M. H. (2020, April 30). Konsep Lintasan Terpendek algoritma dijkstra. YouTube.