Journal of Software Engineering, Information and Communication Technology (SEICT)

Journal homepage: https://ejournal.upi.edu/index.php/SEICT

# Searching for the Fastest Route to Tourist Attractions with the Kruskal Algorithm in the C++ Programming Language

*Billdan Satriana Roseandree[1], Amanda Jayanti Mulyana [2], Raymico Fuji[3], Aldini Hegle Pratama[4] dan Purnama[5]*

[1-5]Software Engineering Study Program, Universitas Pendidikan Indonesia, Bandungm, Indonesia
billdansatriana@upi.edu[1*)], amanda1601@upi.edu[2)], raymico.fuji49@upi.edu[3)], aldini7@upi.edu[4)], purnama31@upi.edu[5)]

## ABSTRACT

*In the tourism industry, finding the fastest way to various attractions is important. In this research, efficient and accurate algorithms facilitate the travel planning process. One algorithm that has proven effective in solving this problem is Kruskal's algorithm. This research aims to implement Kruskal's algorithm in C++ programming language to find the fastest route between tourist destinations. This research uses the C++ programming language to implement the Kruskal algorithm. Information about tourist attractions and distances between tourist destinations are presented in a graph. Kruskal's algorithm is used to find the shortest path using the concept of MST (minimum spanning tree). This research results in a C++ program that can use Kruskal's algorithm to find the fastest route between tourist destinations based on the shortest distance. The program leads to some tourist destinations that must be visited for the fastest route. Using Kruskal's algorithm, the program finds the fastest route between tourist destinations, considering the shortest distance. Thus, this research provides an efficient and accurate solution to the problem of determining the fastest route in the tourism industry. The resulting program can be a useful guide for tourists when planning their trips and optimizing the time and effort to visit various tourist attractions.*



## ARTICLE INFO

***Article History:***
*Submitted/Received 00 xxx 2021*
*First Revised 00 xxx 2021*
*Accepted 00 xxx 2021*
*First Available online 00 xxx 2021*
*Publication Date 00 xxx 2021*

***Keyword:***
*C++ Programming Language;*
*Fastest Route;*
*Kruskal Algorithm;*
*Minimum Spanning Tree;*
*Tourism Industry;*

## 1. INTRODUCTION

Tourism is a sector that continues to grow in the global industry, with the emergence of various tourist destinations that attract the attention of tourists from various corners of the world. When traveling, one of the important things for tourists is finding the fastest route to visit various tourist attractions.

Kruskal's algorithm is one of the algorithms used in graph theory to find a weighted graph's minimum spanning tree (MST) [1]. This algorithm is especially useful in finding the fastest route between tourist attractions connected via certain routes. Regarding this, the research aims to implement the Kruskal algorithm in the C++ programming language to find the fastest route between tourist attractions. Using the Kruskal algorithm, it is hoped that the best routes connecting these tourist attractions can be found with minimum travel time. The C++ programming language was chosen as the main language to implement this algorithm because C++ is a programming language commonly used in software development and performs well. Implementing the Kruskal algorithm in the C++ programming language also provides code development and maintenance flexibility.

This research is hoped to provide a better understanding of implementing the Kruskal algorithm and its application in finding the fastest route. Apart from that, this research will also include experiments and evaluation results of the performance of the Kruskal algorithm in finding the fastest routes to tourist attractions. Thus, this research has an important objective of increasing efficiency in tourist travel through the use of the Kruskal algorithm in the C++ programming language.

## 2. METHODS

### 2.1. Extreme Programming (XP)

This research uses the Extreme Programming (XP) method as a reference. XP is an agile software development method emphasizing coding activities as the main activity at each stage of the development cycle [2] [3]. Figure 1 displays the stages contained in XP.

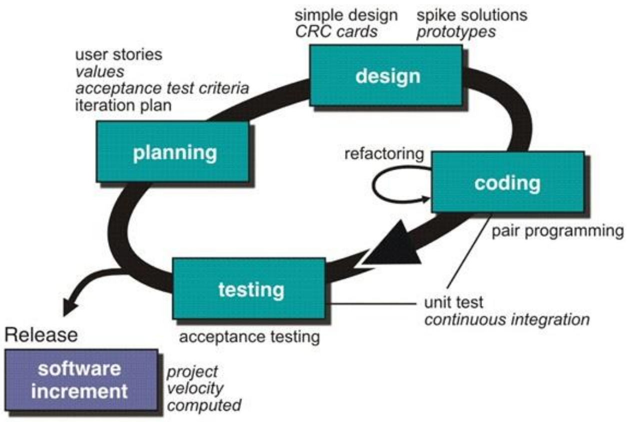


**Figure 1.** Stages in Extreme Programming (XP)

From Figure 1 above, the stages in the software development method with XP are as follows:

1. Planning: The initial step to start research is to define the requirements needed, the results that will be produced, the services that will be developed in the application, and the features and functionality of the application that will be built [1].
2. Design: This step is part of planning an application that suits its use needs [1].

DOI: https://doi.org/10.17509/seict.v4i2.59506
p-ISSN 2774-1656 e- ISSN 2774-1699

3. Coding: Steps in preparing code for software that will be applied in application development so that it can be a solution to existing problems [1].
4. Testing: Testing services or application features and functionality built as the final stage of the testing process. At this stage, conclusions can be drawn from the tests' results [1].

**2.2. XP Steps**

1. Planning:

Planning involves planning the features needed to find the shortest route. The features needed are 1) Determining the distance between locations and determining the type of graph, 2) Representing location neighbors using a graph, 3) Running the minimum spanning tree algorithm, and 4) Displaying MST results.

2. Design:

Design in XP is done by drawing an undirected and weighted graph, then determining its representation using an adjacency matrix, and finally designing the Kruskal algorithm as a problem-solving algorithm.

3. Coding

The programming language used for coding is C++ by implementing the Kruskal Algorithm.

a. Graph

A graph is a discrete structure consisting of vertices and edges (vertices and edges) that connect the vertices in the graph. Many types of graphs depend on whether the graph has directed and/or weighted edges.

b. Adjacency Matrix

The adjacency matrix is one way to represent edges in a graph to express the connection between vertices in the graph. Adjacency matrix representation becomes more effective if there are a large number of edges in a graph compared to an adjacency list. Let G = (V, E) be a simple graph where |V| = n. So, the order of the adjacency matrix AG is n×n with a value of 1 for the (i, j)th vertex, which is neighboring and 0 for the (i, j)th vertex, which is not neighboring for an undirected and unweighted graph. Still, if the graph is weighted, then a weight value will be given to the (i, j) neighboring nodes.

c. Kruskal's Algorithm

The Kruskal algorithm is a Greedy algorithm for finding the Minimum Spanning Tree (MST) in graphs, especially undirected and weighted graphs. The following is the pseudocode of Kruskal's algorithm [5].

In molestie ipsum lorem. Aenean id mi arcu. Phasellus semper efficitur eros eu laoreet. Vivamus vitae malesuada turpis. Morbi interdum orci iaculis tempor facilisis. Suspendisse euismod commodo nulla. Nullam eget congue justo. Phasellus vestibulum quis risus ut pharetra.

**ALGORITHM 2 Kruskal's Algorithm.**

```
procedure Kruskal(G: weighted connected undirected graph with n vertices)
T := empty graph
for i := 1 to n − 1
    e := any edge in G with smallest weight that does not form a simple circuit
        when added to T
    T := T with e added
return T {T is a minimum spanning tree of G}
```

**Figure 2.** Kruskal's Algorithmn

DOI: https://doi.org/10.17509/seict.v4i2.59506
p-ISSN 2774-1656 e- ISSN 2774-1699

*d. Testing*

After all the code is integrated, unit testing will be carried out on each program functionality to ensure the program runs well [6].

## 3. RESULTS AND DISCUSSION

The Kruskal algorithm was implemented to find the shortest route to a location using the C++ programming language, and MST results were found.

A. Device Information

The program was built with the help of the Visual Studio Code IDE (Integrated Development Environment) application and the g++ compiler version 9.2.0 on a laptop with the following specifications.

- Windows 11, 64-bit.
- AMD Ryzen 7 5700U with Radeon Graphics @ 1.80 GHz
- 8 GB DDR4 RAM.
- 512 GB SSD.

B. Application of the Kruskal Algorithm

The Kruskal algorithm is generally applied to undirected and weighted graphs. In this case study, 40 locations are depicted vertically, and edges depict paths connecting between locations with weights stating the distance between locations in meters. Tourist locations are expressed in node form starting from the $0^{th}$ index.

**Table 1.** Tourist attractions nodes

| **Node** | **Location of Tourist Attractions** |
|---|---|
| 0. | UPI Kampus Cibiru |
| 1. | Nimo Highland |
| 2. | Kawah Papandayan |
| 3. | Darajat Pass |
| 4. | Pantai Pangandaran |
| 5. | Kebun Binatang Bandung |
| 6. | Museum Sribaduga |
| 7. | Kiara Artha Park |
| 8. | Taman Langit |
| 9. | Situ Patengan |
| 10. | Museum Gedung Sate |
| 11. | Grey Art Gallery |
| 12. | Kawah Putih |
| 13. | Kebun Raya Cibodas |
| 14. | Orchid Forest Cikol |
| 15. | Floating Market Lembang |
| 16. | Dago Dreampark |
| 17. | Kampung Gajah Wonderland |
| 18. | Taman Bunga Begonia |
| 19. | De Ranch Lembang |
| 20. | Curug Tilu Leuwi Opat |
| 21. | Taman Wisata Maribaya |
| 22 | Trans Studio Bandung |
| 23. | Taman Hutan Raya Juanda |

DOI: https://doi.org/10.17509/seict.v4i2.59506
p-ISSN 2774-1656 e- ISSN 2774-1699

| | |
|---|---|
| 24. | Gunung Tangkuban Perahu |
| 25. | Taman Wisata Alam Ciwidey |
| 26. | Curug Dago |
| 27. | Trans Studio Mini Bandung |
| 28. | Kampung Daun Culture Gallery |
| 29. | Jendela Alam |
| 30. | Farmhouse Susu Lembang |
| 31. | Kampung Gajah Wonderland |
| 32. | Curug Cimahi |
| 33. | Taman Superhero |
| 34. | De Ranch Lembang |
| 35. | Museum Geologi Bandung |
| 36. | Kampung Cai Ranca Upas |
| 37. | Amazing Art World |
| 38. | Taman Wisata Grafika Cikole |
| 39. | Kampung Wisata Situ Cileunca |

In order to be computable, graphs need to be converted into graph representations that can be processed easily. Because the graph in this case study has many sides, the graph representation that is suitable to use is the adjacency matrix. Below is the neighborhood matrix.

**Tabel 2.** Adjacency Matrix

| V | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 10894 | 25393 | 16573 | 1900 | 3421 | 29009 | 25389 | 22718 | 20558 |
| 1 | 10894 | 0 | 4548 | 6097 | 33591 | 15310 | 28157 | 6075 | 4753 | 24646 |
| 2 | 25393 | 4548 | 0 | 17512 | 19762 | 7617 | 21472 | 11712 | 17139 | 28982 |
| 3 | 16573 | 6097 | 17512 | 0 | 2655 | 23813 | 24622 | 3600 | 13423 | 1481 |
| 4 | 1900 | 33591 | 19762 | 2655 | 0 | 10514 | 19538 | 3510 | 17279 | 5144 |
| 5 | 3421 | 15310 | 7617 | 23813 | 10514 | 0 | 13292 | 22003 | 26996 | 24196 |
| 6 | 29009 | 28157 | 21472 | 24622 | 19538 | 13292 | 0 | 27869 | 17687 | 21222 |
| 7 | 25389 | 6075 | 11712 | 3600 | 3510 | 22003 | 27869 | 0 | 13529 | 8129 |
| 8 | 22718 | 4753 | 17139 | 13423 | 17279 | 26996 | 17687 | 13529 | 0 | 3161 |
| 9 | 20558 | 24646 | 28982 | 1481 | 5144 | 24196 | 21222 | 8129 | 3161 | 0 |

| | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 0 | 31833 | 8164 | 21851 | 2626 | 23848 | 28611 | 13181 | 22221 | 16944 |
| 11 | 31833 | 0 | 6109 | 19662 | 10357 | 5675 | 31877 | 29503 | 23663 | 3865 |
| 12 | 8164 | 6109 | 0 | 14829 | 9526 | 13938 | 21424 | 4829 | 6706 | 19540 |
| 13 | 21851 | 19662 | 14829 | 0 | 14357 | 3223 | 33678 | 24775 | 3363 | 24245 |
| 14 | 2626 | 10357 | 9526 | 14357 | 0 | 23142 | 2752 | 21608 | 10010 | 26508 |
| 15 | 23848 | 5675 | 13938 | 3223 | 23142 | 0 | 19443 | 30292 | 23171 | 29318 |
| 16 | 28611 | 31877 | 21424 | 33678 | 2752 | 19443 | 0 | 6997 | 28489 | 28870 |
| 17 | 13181 | 29503 | 4829 | 24775 | 21608 | 30292 | 6997 | 0 | 19240 | 10601 |
| 18 | 22221 | 23663 | 6706 | 3363 | 10010 | 23171 | 28489 | 19240 | 0 | 29323 |
| 19 | 16944 | 3865 | 19540 | 24245 | 26508 | 29318 | 28870 | 10601 | 29323 | 0 |

| | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|

DOI: https://doi.org/10.17509/seict.v4i2.59506
p-ISSN 2774-1656 e- ISSN 2774-1699

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 0 | 7705 | 27303 | 10781 | 9380 | 5693 | 32907 | 27633 | 30560 | 17336 |
| 21 | 7705 | 0 | 15256 | 27504 | 30815 | 5686 | 25307 | 13158 | 16147 | 19337 |
| 22 | 27303 | 15256 | 0 | 13392 | 26951 | 31080 | 24481 | 13022 | 26162 | 2278 |
| 23 | 10781 | 27504 | 13392 | 0 | 7541 | 11116 | 2012 | 2146 | 2655 | 13393 |
| 24 | 9380 | 30815 | 26951 | 7541 | 0 | 13249 | 22136 | 19392 | 21675 | 8636 |
| 25 | 5693 | 5686 | 31080 | 11116 | 13249 | 0 | 27630 | 14037 | 1792 | 31714 |
| 26 | 32907 | 25307 | 24481 | 2012 | 22136 | 27630 | 0 | 4925 | 23361 | 29164 |
| 27 | 27633 | 13158 | 13022 | 2146 | 19392 | 14037 | 4925 | 0 | 32754 | 32591 |
| 28 | 30560 | 16147 | 26162 | 2655 | 21675 | 1792 | 23361 | 32754 | 0 | 20949 |
| 29 | 17336 | 19337 | 2278 | 13393 | 8636 | 31714 | 29164 | 32591 | 20949 | 0 |

| | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
|---|---|---|---|---|---|---|---|---|---|---|
| 30 | 0 | 1754 | 23018 | 3111 | 6651 | 30492 | 26697 | 7582 | 21136 | 12243 |
| 31 | 1754 | 0 | 15776 | 24650 | 3258 | 13433 | 31877 | 29503 | 23663 | 3865 |
| 32 | 23018 | 15776 | 0 | 29149 | 8213 | 24840 | 18780 | 12159 | 10198 | 8514 |
| 33 | 3111 | 24650 | 29149 | 0 | 10860 | 7766 | 23578 | 21418 | 28032 | 11168 |
| 34 | 6651 | 3258 | 8213 | 10860 | 0 | 2735 | 13931 | 27489 | 15387 | 6055 |
| 35 | 30492 | 13433 | 24840 | 7766 | 2735 | 0 | 33544 | 1159 | 33584 | 12191 |
| 36 | 26697 | 5919 | 18780 | 23578 | 13931 | 33544 | 0 | 4449 | 8240 | 6973 |
| 37 | 7582 | 29503 | 12159 | 21418 | 27489 | 1159 | 4449 | 0 | 4517 | 9922 |
| 38 | 21136 | 23663 | 10198 | 28032 | 15387 | 33584 | 8240 | 4517 | 0 | 7748 |
| 39 | 12243 | 3865 | 8514 | 11168 | 6055 | 12191 | 6973 | 9922 | 7748 | 0 |

After the graph is represented in a matrix as in Table 2, the Kruskal algorithm is run with the steps starting from sorting the edges from smallest to largest (ascending), then starting from the edges with the smallest weight that do not form a cycle/circuit are put into MST.

C. Source Code

The program code begins by creating Side and Graph structs. The graph representation uses an adjacency matrix with a two-dimensional vector array. Then, the graph will be initialized with 40 vertices or nodes representing the number of locations. After that, the sides will be added randomly as dummy data, then the MST will be searched using the Kruskal algorithm and the results will be displayed.

DOI: https://doi.org/10.17509/seict.v4i2.59506
p-ISSN 2774-1656 e- ISSN 2774-1699

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;

struct Sisi
{
    int asal, tujuan, bobot;
};

struct Graf
{
    int V, E;
    vector < vector < int > > matriksKetetanggaan;
};

Graf buatGraf(int simpul)
{
    Graf graf;

    graf.V = simpul;
    graf.E = 0;

    graf.matriksKetetanggaan.resize(simpul, vector < int > (simpul,
0));
    return graf;
}

void tambahSisi(Graf &graf, int asal, int tujuan, int bobot)
{
    if (asal == tujuan)
    {
        return;
    }

    graf.matriksKetetanggaan[asal][tujuan] = bobot;
    graf.matriksKetetanggaan[tujuan][asal] = bobot;
    graf.E++;
}
```

DOI: https://doi.org/10.17509/seict.v4i2.59506
p-ISSN 2774-1656 e- ISSN 2774-1699

```
// KRUSKAL MULAI
bool bandingSisi(const Sisi &s1, const Sisi &s2)
{
    return s1.bobot < s2.bobot;
}

queue < Sisi > kruskalMST(const Graf &graf)
{
    // Membaca semua sisi dari matriks ketetanggaan
    vector < Sisi > edges;

    for (int i = 0; i < graf.V; i++)
    {
        for (int j = i; j < graf.V; j++)
        {
            if (graf.matriksKetetanggaan[i][j] != 0)
            {
                edges.push_back({i, j, graf.matriksKetetanggaan[i][j]});
            }
        }
    }

    // Mengurutkan sisi-sisi berdasarkan bobotnya
    sort(edges.begin(), edges.end(), bandingSisi);

    // Minimum Spanning Tree
    queue < Sisi > mst;
    vector < int > parent(graf.V, -1);

        // Menjalankan algoritma Kruskal
    for (const auto &sisi : edges)
    {
        // Mencari akar dari set yang berisi asal dan tujuan
        int rootAsal = sisi.asal;
        while (parent[rootAsal] != -1)
        {
            rootAsal = parent[rootAsal];
        }

        int rootTujuan = sisi.tujuan;
        while (parent[rootTujuan] != -1)
        {
            rootTujuan = parent[rootTujuan];
        }

        if (rootAsal != rootTujuan)
        {
            mst.push(sisi);
            parent[rootTujuan] = rootAsal;  // Menggabungkan dua himpunan
        }
    }

    return mst
}
// KRUSKAL SELESAI
```

**Figure 5.** main() Function

Based on Figure 3, the graph is initialized in the createGraf() function to create an instance of a graph object with the number of vertices as a parameter, and there is also an addEdge() function to add edges to the graph. Then, the Kruskal Algorithm in Figure 4 is implemented with the kruskalMST() function, which returns a queue to store MST edges. Then, in the main() function in Figure 5, all sides are filled with dummy data or distances between artificial locations, and the implementation results are displayed.

D. Output

*The r esulting output is as follows.*

DOI: https://doi.org/10.17509/seict.v4i2.59506
p-ISSN 2774-1656 e- ISSN 2774-1699

Matriks Ketetanggaan

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 10894 | 25393 | 16573 | 1900 | 3421 | 29009 | 25389 | 22718 | 20558 |
| 1 | 10894 | 0 | 4548 | 6097 | 33591 | 15310 | 28157 | 6075 | 4753 | 24646 |
| 2 | 25393 | 4548 | 0 | 17512 | 19762 | 7617 | 21472 | 11712 | 17139 | 28982 |
| 3 | 16573 | 6097 | 17512 | 0 | 2655 | 23813 | 24622 | 3600 | 13423 | 1481 |
| 4 | 1900 | 33591 | 19762 | 2655 | 0 | 10514 | 19538 | 3510 | 17279 | 5144 |
| 5 | 3421 | 15310 | 7617 | 23813 | 10514 | 0 | 13292 | 22003 | 26996 | 24196 |
| 6 | 29009 | 28157 | 21472 | 24622 | 19538 | 13292 | 0 | 27869 | 17687 | 21222 |
| 7 | 25389 | 6075 | 11712 | 3600 | 3510 | 22003 | 27869 | 0 | 13529 | 8129 |
| 8 | 22718 | 4753 | 17139 | 13423 | 17279 | 26996 | 17687 | 13529 | 0 | 3161 |
| 9 | 20558 | 24646 | 28982 | 1481 | 5144 | 24196 | 21222 | 8129 | 3161 | 0 |

| | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 0 | 31833 | 8164 | 21851 | 2626 | 23848 | 28611 | 13181 | 22221 | 16944 |
| 11 | 31833 | 0 | 6109 | 19662 | 10357 | 5675 | 31877 | 29503 | 23663 | 3865 |
| 12 | 8164 | 6109 | 0 | 14829 | 9526 | 13938 | 21424 | 4829 | 6706 | 19540 |
| 13 | 21851 | 19662 | 14829 | 0 | 14357 | 3223 | 33678 | 24775 | 3363 | 24245 |
| 14 | 2626 | 10357 | 9526 | 14357 | 0 | 23142 | 2752 | 21608 | 10010 | 26508 |
| 15 | 23848 | 5675 | 13938 | 3223 | 23142 | 0 | 19443 | 30292 | 23171 | 29318 |
| 16 | 28611 | 31877 | 21424 | 33678 | 2752 | 19443 | 0 | 6997 | 28489 | 28870 |
| 17 | 13181 | 29503 | 4829 | 24775 | 21608 | 30292 | 6997 | 0 | 19240 | 10601 |
| 18 | 22221 | 23663 | 6706 | 3363 | 10010 | 23171 | 28489 | 19240 | 0 | 29323 |
| 19 | 16944 | 3865 | 19540 | 24245 | 26508 | 29318 | 28870 | 10601 | 29323 | 0 |

| | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 0 | 7705 | 27303 | 10781 | 9380 | 5693 | 32907 | 27633 | 30560 | 17336 |
| 21 | 7705 | 0 | 15256 | 27504 | 30815 | 5686 | 25307 | 13158 | 16147 | 19337 |
| 22 | 27303 | 15256 | 0 | 13392 | 26951 | 31080 | 24481 | 13022 | 26162 | 2278 |
| 23 | 10781 | 27504 | 13392 | 0 | 7541 | 11116 | 2012 | 2146 | 2655 | 13393 |
| 24 | 9380 | 30815 | 26951 | 7541 | 0 | 13249 | 22136 | 19392 | 21675 | 8636 |
| 25 | 5693 | 5686 | 31080 | 11116 | 13249 | 0 | 27630 | 14037 | 1792 | 31714 |
| 26 | 32907 | 25307 | 24481 | 2012 | 22136 | 27630 | 0 | 4925 | 23361 | 29164 |
| 27 | 27633 | 13158 | 13022 | 2146 | 19392 | 14037 | 4925 | 0 | 32754 | 32591 |
| 28 | 30560 | 16147 | 26162 | 2655 | 21675 | 1792 | 23361 | 32754 | 0 | 20949 |
| 29 | 17336 | 19337 | 2278 | 13393 | 8636 | 31714 | 29164 | 32591 | 20949 | 0 |

| | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
|---|---|---|---|---|---|---|---|---|---|---|
| 30 | 0 | 1754 | 23018 | 3111 | 6651 | 30492 | 26697 | 7582 | 21136 | 12243 |
| 31 | 1754 | 0 | 15776 | 24650 | 3258 | 13433 | 5919 | 14578 | 5680 | 30929 |
| 32 | 23018 | 15776 | 0 | 29149 | 8213 | 24840 | 18780 | 12159 | 10198 | 8514 |
| 33 | 3111 | 24650 | 29149 | 0 | 10860 | 7766 | 23578 | 21418 | 28032 | 11168 |
| 34 | 6651 | 3258 | 8213 | 10860 | 0 | 2735 | 13931 | 27489 | 15387 | 6055 |
| 35 | 30492 | 13433 | 24840 | 7766 | 2735 | 0 | 33544 | 1159 | 33584 | 12191 |
| 36 | 26697 | 5919 | 18780 | 23578 | 13931 | 33544 | 0 | 4449 | 8240 | 6973 |
| 37 | 7582 | 14578 | 12159 | 21418 | 27489 | 1159 | 4449 | 0 | 4517 | 9922 |
| 38 | 21136 | 5680 | 10198 | 28032 | 15387 | 33584 | 8240 | 4517 | 0 | 7748 |
| 39 | 12243 | 30929 | 8514 | 11168 | 6055 | 12191 | 6973 | 9922 | 7748 | 0 |

**Figure 6**. Output matriks ketetanggaan

DOI: https://doi.org/10.17509/seict.v4i2.59506
p-ISSN 2774-1656 e- ISSN 2774-1699

Rute Minimum

| Langkah | Sisi | Bobot |
|---|---|---|
| 1 | Nimo Highland ==> Taman Wisata Alam Ciwidey | 1028 |
| 2 | Kawah Papandayan ==> Gunung Tangkuban Perahu | 1058 |
| 3 | Taman Hutan Raya Juanda ==> Amazing Art World | 1073 |
| 4 | Museum Geologi Bandung ==> Amazing Art World | 1159 |
| 5 | Taman Bunga Begonia ==> Kampung Cai Ranca Upas | 1189 |
| 6 | Kampung Gajah Wonderland ==> Kampung Daun Culture Gallery | 1229 |
| 7 | Situ Patengan ==> Grey Art Gallery | 1235 |
| 8 | Nimo Highland ==> Kampung Wisata Situ Cileunca | 1241 |
| 9 | Darajat Pass ==> Dago Dreampark | 1300 |
| 10 | Kebun Raya Cibodas ==> Jendela Alam | 1330 |
| 11 | Kebun Binatang Bandung ==> Kampung Gajah Wonderland | 1335 |
| 12 | Taman Bunga Begonia ==> Kampung Gajah Wonderland | 1474 |
| 13 | UPI Kampus Cibiru ==> Jendela Alam | 1475 |
| 14 | Darajat Pass ==> Situ Patengan | 1481 |
| 15 | Situ Patengan ==> Taman Bunga Begonia | 1503 |
| 16 | Nimo Highland ==> Kampung Daun Culture Gallery | 1540 |
| 17 | Museum Sribaduga ==> Curug Tilu Leuwi Opat | 1608 |
| 18 | Dago Dreampark ==> Kampung Daun Culture Gallery | 1658 |
| 19 | Kebun Raya Cibodas ==> Curug Tilu Leuwi Opat | 1690 |
| 20 | Grey Art Gallery ==> De Ranch Lembang | 1702 |
| 21 | Taman Bunga Begonia ==> Curug Tilu Leuwi Opat | 1748 |
| 22 | Farmhouse Susu Lembang ==> Kampung Gajah Wonderland | 1754 |
| 23 | UPI Kampus Cibiru ==> Pantai Pangandaran | 1900 |
| 24 | Kiara Artha Park ==> Floating Market Lembang | 1911 |
| 25 | Orchid Forest Cikole ==> Farmhouse Susu Lembang | 1943 |
| 26 | Situ Patengan ==> Curug Cimahi | 1958 |
| 27 | Taman Hutan Raya Juanda ==> Curug Dago | 2012 |
| 28 | Taman Langit ==> Trans Studio Mini Bandung | 2052 |
| 29 | Taman Langit ==> Taman Wisata Alam Ciwidey | 2071 |
| 30 | Taman Hutan Raya Juanda ==> Trans Studio Mini Bandung | 2146 |
| 31 | Trans Studio Bandung ==> Jendela Alam | 2278 |
| 32 | Gunung Tangkuban Perahu ==> Museum Geologi Bandung | 2386 |
| 33 | Kiara Artha Park ==> Amazing Art World | 2560 |
| 34 | Museum Gedung Sate ==> Orchid Forest Cikole | 2626 |
| 35 | Kebun Binatang Bandung ==> De Ranch Lembang | 2769 |
| 36 | Nimo Highland ==> Kawah Putih | 2926 |
| 37 | Farmhouse Susu Lembang ==> Taman Superhero | 3111 |
| 38 | Taman Wisata Maribaya ==> De Ranch Lembang | 3327 |
| 39 | Jendela Alam ==> Taman Wisata Grafika Cikole | 3599 |

Minimum Spanning Tree Cost: 72385

**Figure 7.** MST Output

Figure 6 displays the adjacency matrix where the rows represent the origin node, and the columns represent the destination node. Then, in Figure 7, the MST results show 72,385 meters for the total cost of travel connecting all tourist locations.

**Table 3.** Kruskal Algorithm Steps

| LANGKAH | EDGE | | BOBOT |
|---|---|---|---|
| | I | J | |
| 0 | | | |
| 1 | 1 | 25 | 1028 |
| 2 | 2 | 24 | 1058 |
| 3 | 23 | 37 | 1073 |
| 4 | 35 | 37 | 1159 |
| 5 | 18 | 36 | 1189 |
| 6 | 17 | 28 | 1229 |
| 7 | 9 | 11 | 1235 |
| 8 | 1 | 39 | 1241 |
| 9 | 3 | 16 | 1300 |
| 10 | 13 | 29 | 1330 |
| 11 | 5 | 17 | 1335 |
| 26 | 9 | 32 | 1958 |
| 27 | 23 | 26 | 2012 |

| | | | |
|---|---|---|---|
| 12 | 18 | 31 | 1474 |
| 13 | 0 | 29 | 1475 |
| 14 | 3 | 9 | 1058 |
| 15 | 9 | 18 | 1481 |
| 16 | 1 | 28 | 1540 |
| 17 | 6 | 20 | 1608 |
| 18 | 16 | 28 | 1658 |
| 19 | 13 | 20 | 1690 |
| 20 | 11 | 34 | 1702 |
| 21 | 18 | 20 | 1748 |
| 22 | 30 | 31 | 1754 |
| 23 | 0 | 4 | 1900 |
| 24 | 7 | 15 | 1911 |
| 25 | 14 | 30 | 1943 |
| 33 | 7 | 37 | 2560 |
| 34 | 10 | 14 | 2626 |

DOI: https://doi.org/10.17509/seict.v4i2.59506
p-ISSN 2774-1656 e- ISSN 2774-1699

| | | | |
|---|---|---|---|
| 28 | 8 | 27 | 2052 |
| 29 | 8 | 25 | 2071 |
| 30 | 23 | 27 | 2146 |
| 31 | 22 | 29 | 2278 |
| 32 | 24 | 35 | 2386 |
| 35 | 5 | 19 | 2769 |
| 36 | 1 | 12 | 2926 |
| 37 | 30 | 33 | 3111 |
| 38 | 21 | 34 | 3327 |
| 39 | 29 | 38 | 3599 |

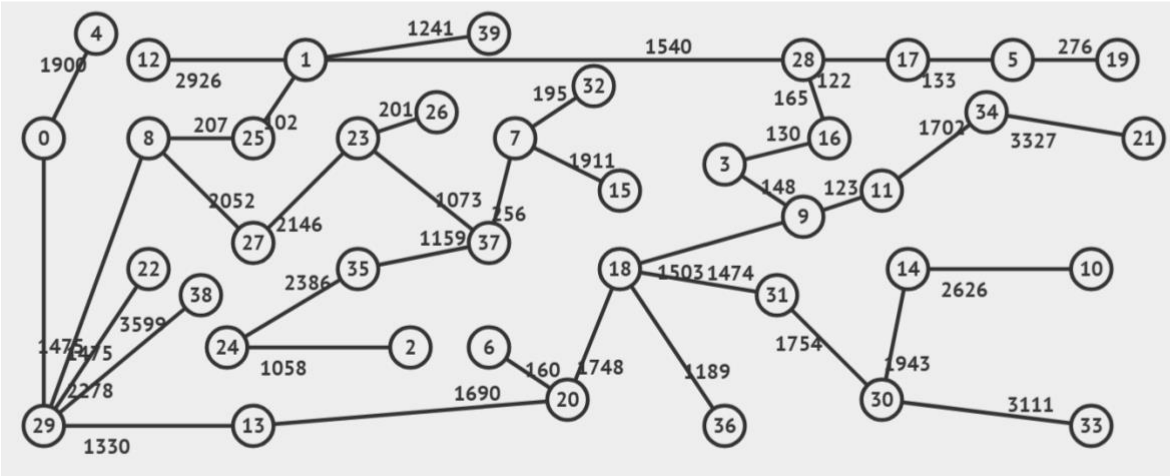


**Figure 8.** MST Graph

## 4. CONCLUSION

This research aims to find the fastest route between certain tourist attractions using the Kruskal algorithm in the C++ programming language. Kruskal's algorithm is one of the fastest route-finding algorithms popular in graph processing. This method focuses on finding the shortest path that involves the minimum path from one tourist spot to another. Dummy data is used in this study to simulate tourist attractions and the distance between them. The Kruskal algorithm is then applied to the dummy data to find the fastest route. In the C++ programming language, the Kruskal algorithm is implemented using data structures such as graphs or matrices. Using C++ provides flexibility and efficiency in data processing and algorithm use. This research is important to help tourists or travelers efficiently plan their trips. By finding the fastest routes between tourist attractions, the time and effort required to travel can be reduced.

However, remember that these conclusions are based on the use of dummy data and do not include actual results from actual research. Further research and testing with real data is needed to confirm the effectiveness of Kruskal's algorithm in finding the fastest route between tourist attractions.

## 5. REFERENCES

[1] Yasin, M., & Afandi, B. (2014). Simulasi Minimum Spanning Tree Graf Berbobot Menggunakan Algoritma Prim dan Algoritma Kruskal. *Jurnal Educazione: Jurnal Pendidikan, Pembelajaran dan Bimbingan dan konseling*, *2*(2).

[2] Borman, R. I., Priandika, A. T., & Edison, A. R. (2020). Implementasi Metode Pengembangan Sistem Extreme Programming (XP) pada Aplikasi Investasi Peternakan. *JUSTIN (Jurnal Sistem Dan Teknologi Informasi)*, *8*(3), 272-277.

DOI: https://doi.org/10.17509/seict.v4i2.59506
p-ISSN 2774-1656 e- ISSN 2774-1699

[3] Suryantara, I. G. N., Kom, S., & Kom, M. (2017). *Merancang Applikasi dengan Metodologi Extreme Programming*. Elex Media Komputindo.

[4] K. Rosen, *Discrete Maths and Its Applications Global Edition 7e*. McGraw Hill, 2012.

[5] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 2010.

DOI: https://doi.org/10.17509/seict.v4i2.59506
p-ISSN 2774-1656 e- ISSN 2774-1699