ULAB
UNIVERSITY OF LIBERAL ARTS BANGLADESH

Project Name: Complex Engineering Project Final Report

Group Name: RMDM-Group

Group Members: -

1.Dewan Rezwan Ahmed (202014016)
2.Mahfuja Akter Shifa (201014001)
3.Rabeya Bosri (201014028)
4.Md. Mizanur Rahman (201014061)

Submission Date: 21-05-2022

Table of Content

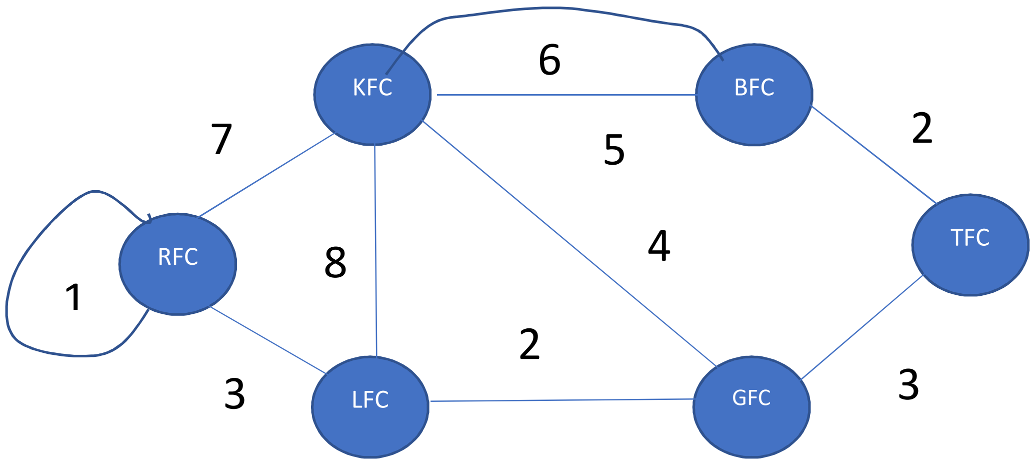
Individual Contributions Breakdown:

Date	Name	Topic
21-05-2022	Md.Mizanur Rahman Rabeya Bosri Mahfuja Akter Shifa Dewan Rezwan Ahmed	Individual Contributions Breakdown
20-05-2022	Mahfuja Akter Shifa	System Requirements
21-05-2022	Dewan Rezwan Ahmed Rabeya Bosri	Functional Requirements Specification
21-05-2022	Md.Mizanur Rahman	Design of tests
20-05-2022	Md.Mizanur Rahman Rabeya Bosri Mahfuja Akter Shifa Dewan Rezwan Ahmed	Code Editing
21-05-2022	Rabeya Bosri Mahfuja Akter Shifa	Organization
21-05-2022	Dewan Rezwan Ahmed	Current Status and Future work

1. Statement of Requirements

1.1 Problem Statement:

On our way to Officers Club, Baily Road from the ULAB Research Building, we plan to visit all bakery shops and we are short on time. Inside the program, we will use Kruskal Algorithms to find the minimum spanning tree. Among multiple paths in the graph, our program will let the user find the optimal and shortest path to get the minimum time needed to move from one node to another.



The graph shows the different bakeries which are situated From Baily Road to the ULAB Research Building. From KFC to BFC there is 5 mins time needed to go. From BFC to TFC 2 mins needed, from TFC to BFC 3 mins needed, from GFC to LFC 2 mins needed, from LFC to RFC 3 mins needed, from KFC to LFC 8 min needed and from KFC to GFC 4 mins needed to go. There is a loop in RFC that means there is another path that takes 1 min to go to the similar bakery and from KFC to BFC there is another path which take 6 mins to go.

2. Glossary of Terms:

Applications where Kruskal's algorithm is generally used:

1. Landing cables
2. TV Network
3. Tour Operations
4. LAN Networks
5. A network of pipes for drinking water or natural gas.
6. An electric grid
7. Single-link Cluster

Minimum Spanning Tree: A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected. By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.

General Properties of Spanning Tree:

We now understand that one graph can have more than one spanning tree. Following are a few properties of the spanning tree connected to graph G –

- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e., the spanning tree is **minimally connected**.
- Adding one edge to the spanning tree will create a circuit or loop, i.e., the spanning tree is **maximally acyclic**.

Developer/Programmer: A developer is the key individual behind all software applications. Generally, developers are well versed in at least one programming language and proficient in the art of structuring and developing software code for software or a program. Depending on job role and type of software developed, a developer may be classified as a software developer, application developer, mobile developer, Web developer, etc.

Database: Database is a program where the application's information is stored and retrieved.

The bandwidth of Path: Bandwidth: The maximum amount of data transmitted over an internet connection in each amount of time.

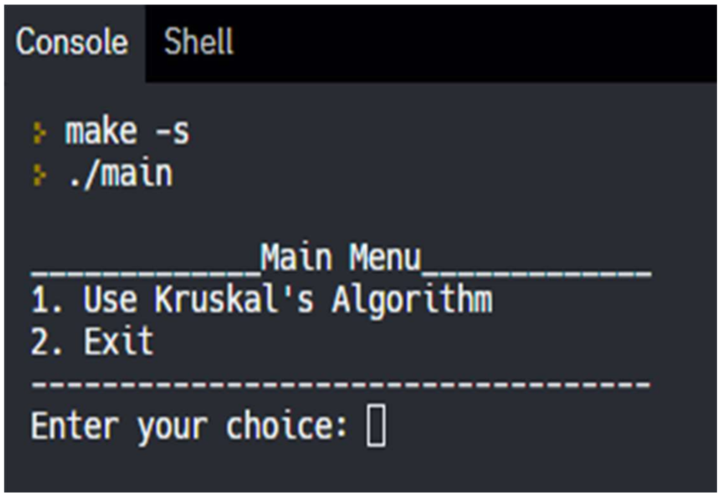
3. System Requirements

3.1 Enumerated Functional Requirements:

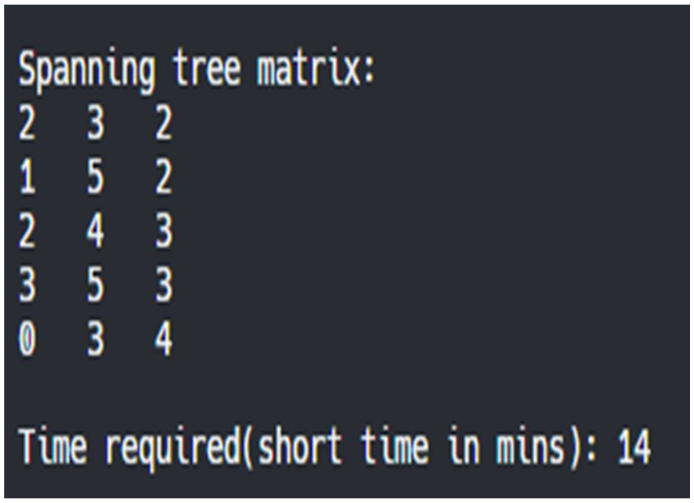
Priority 5 is more vital than Priority 1

Identifier	Priority	Requirements
REQ-1	5	This program will use Kruskal's Algorithm to sort the shortest distance to travel all the bakeries from Baily Road to the ULAB Research Building.
REQ-2	5	This program will allow to visit all bakeries.
REQ-3	2	This program will allow to users to take source bakery and destination bakery.
REQ-4	2	This program will allow to take the time needed to go to destination bakery from source bakery.
REQ-5	4	This program will allow to show the spanning matrix.
REQ-6	4	This program will allow to show the minimum time needed after visiting all the bakeries.
REQ-7	5	This program will show the final output that means the minimum time with spanning matrix
REQ-8	5	This program will exit after completing the process
REQ-9	5	This program will not allow loop.
REQ-10	5	This program will not be applicable for directed graph.

3.2 On-Screen Appearance Requirements:



```
> make -s
> ./main

_____________Main Menu____________
1. Use Kruskal's Algorithm
2. Exit
-----------------------------------
Enter your choice: 
```



```
Spanning tree matrix:
2   3   2
1   5   2
2   4   3
3   5   3
0   3   4

Time required(short time in mins): 14
```

4. Functional Requirements Specification

4.1 Actors and Goals

Actor	Rule	Goal
User	The user will be able to choose either Kruskal algorithm or exit from the main menu, input number of vertices, and then input all the adjacency edge values corresponding to node values i.e. [0][1] = 6. The user will also have to state the source and destination node name.	The program will produce a spanning tree matrix to show which nodes and edge values are taken to calculate the minimum cost to travel from source to destination.

5. Design of Tests

5.1 Test Cases

Test Case Identifier: TC-1

Use Case Tested: UC-2

Pass/Fail Criteria: The test will pass if we use inputs to get expected results in an undirected graph.

Input Data:

[0][0] = 0 [0][1] = 5 [0] [2] = 8 [0][3] = 4 [0][4] =7 [0][5] = 0

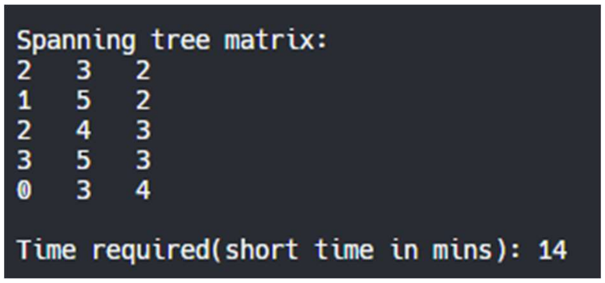
[1][0] = 5 [1][1] = 0 [1][2] = 0 [1][3] = 0 [1][4] = 0 [1][5] = 0

[2][0] = 8 [2][1] = 0 [2][2] = 0 [2][3] = 2 [2][4] = 3 [2][5] = 0

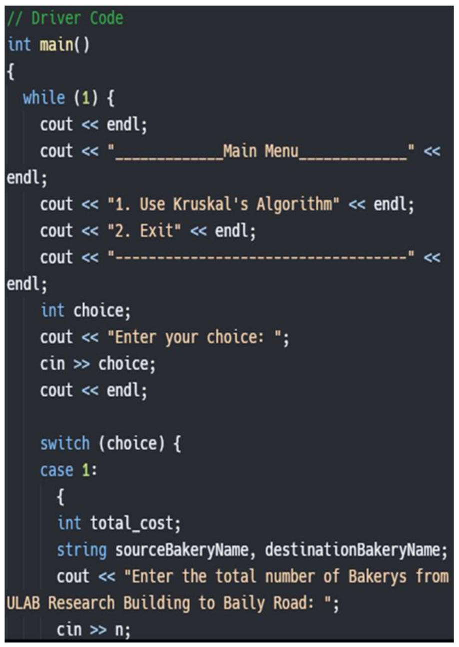
[3][0] = 4 [3][1] = 0 [3][2] = 2 [3][3] = 0 [3][4] = 0 [3][5] = 3

[4][0] = 7 [4][1] =0 [4][2] = 3 [4][3] = 0 [4][4] = 0 [4][5] = 0

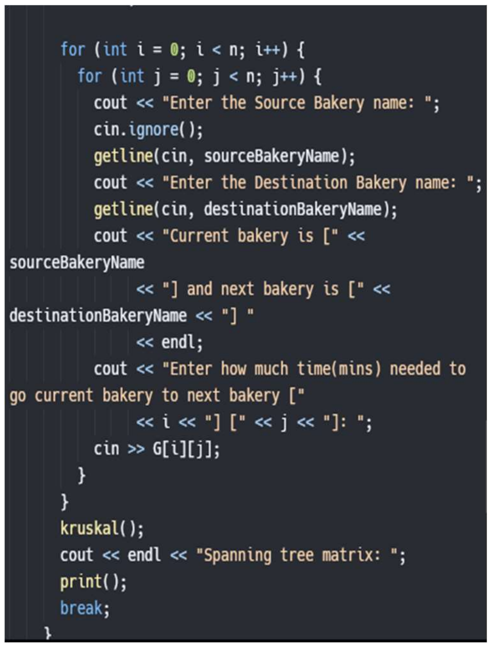
[5][0] = 0 [5][1] = 2 [5][2] = 0 [5][3] = 3 [5][4] = 0 [5][5] = 0

Test Procedure:	**Expected Result:**
Step 1: Select a starting vertex **Step 2:** Repeat Steps 3 and 4 until there are fringe vertices **Step 3:** Select an edge 'e' connecting the tree vertex and fringe vertex that has minimum weight Step 4: Add the selected edge and the vertex to the minimum spanning tree T [END OF LOOP] **Step 5:** At last, we see that, the minimum cost is 13 **Step 6:** EXIT	Minimum spanning tree of the connected graph. Gives the actual result with minimum time.  Spanning tree matrix: 2 3 2 1 5 2 2 4 3 3 5 3 0 3 4 Time required(short time in mins): 14

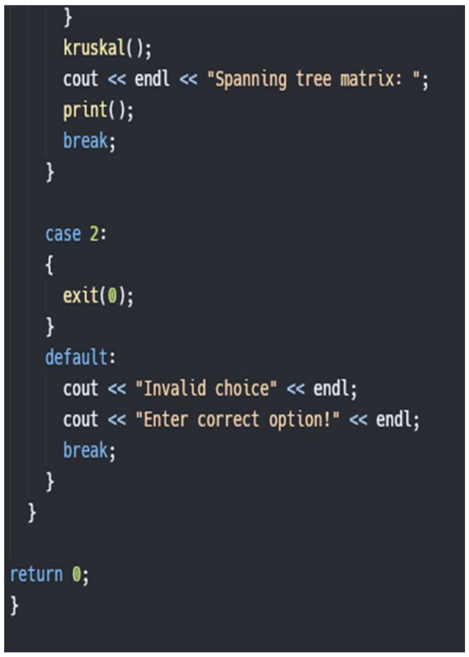
Code analysis: We have used some functions to develop the program properly. First of all we use a main function and Kruskal (), sort () and print () functions are called from the main function.



```
// Driver Code
int main()
{
  while (1) {
    cout << endl;
    cout << "____________Main Menu____________" <<
endl;
    cout << "1. Use Kruskal's Algorithm" << endl;
    cout << "2. Exit" << endl;
    cout << "----------------------------------" <<
endl;
    int choice;
    cout << "Enter your choice: ";
    cin >> choice;
    cout << endl;

    switch (choice) {
    case 1:
      {
      int total_cost;
      string sourceBakeryName, destinationBakeryName;
      cout << "Enter the total number of Bakerys from
ULAB Research Building to Baily Road: ";
      cin >> n;
```

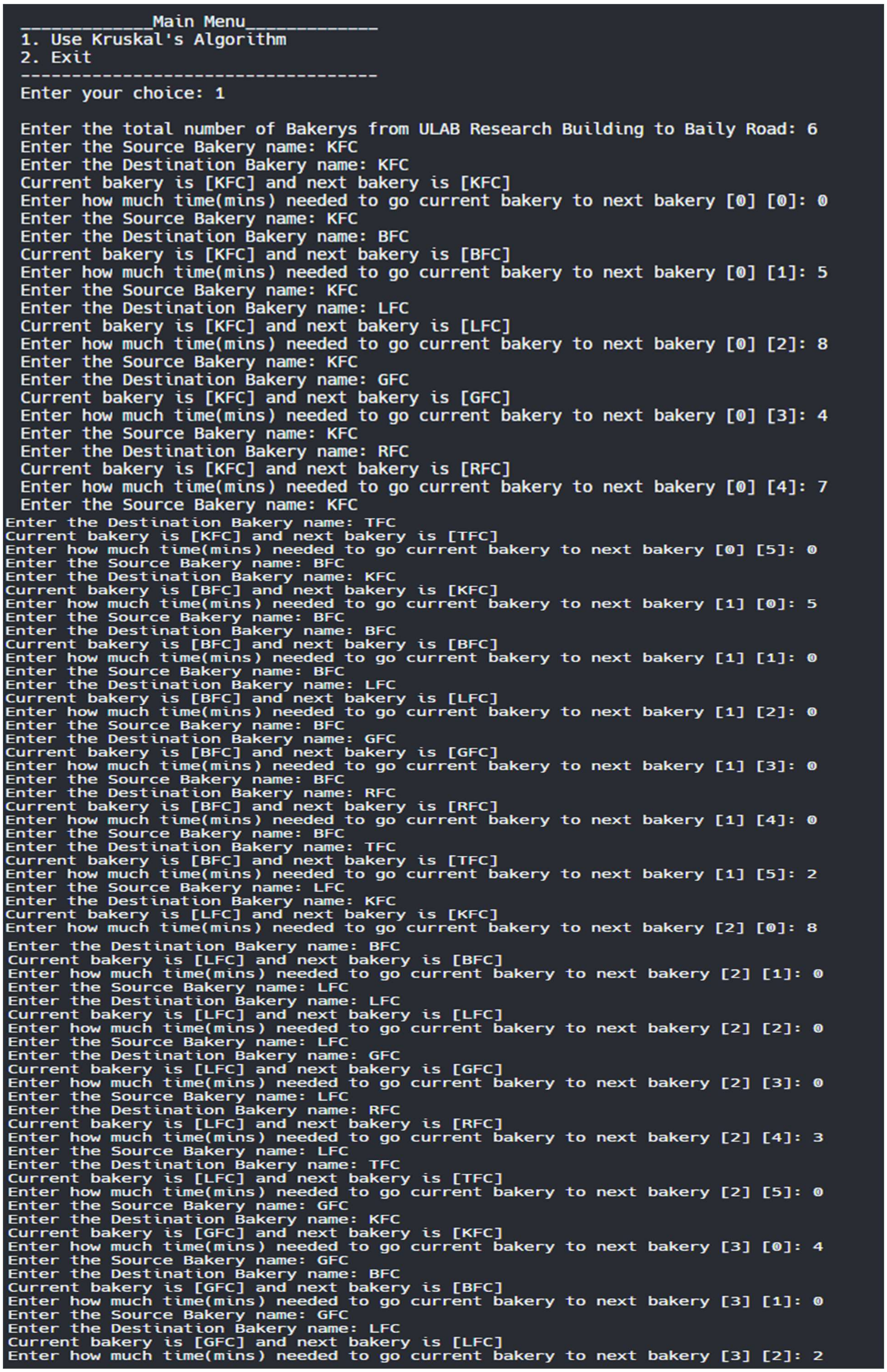


```
      for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
          cout << "Enter the Source Bakery name: ";
          cin.ignore();
          getline(cin, sourceBakeryName);
          cout << "Enter the Destination Bakery name: ";
          getline(cin, destinationBakeryName);
          cout << "Current bakery is [" <<
sourceBakeryName
              << "] and next bakery is [" <<
destinationBakeryName << "] "
              << endl;
          cout << "Enter how much time(mins) needed to
go current bakery to next bakery ["
              << i << "] [" << j << "]: ";
          cin >> G[i][j];
        }
      }
      kruskal();
      cout << endl << "Spanning tree matrix: ";
      print();
      break;
    }
```

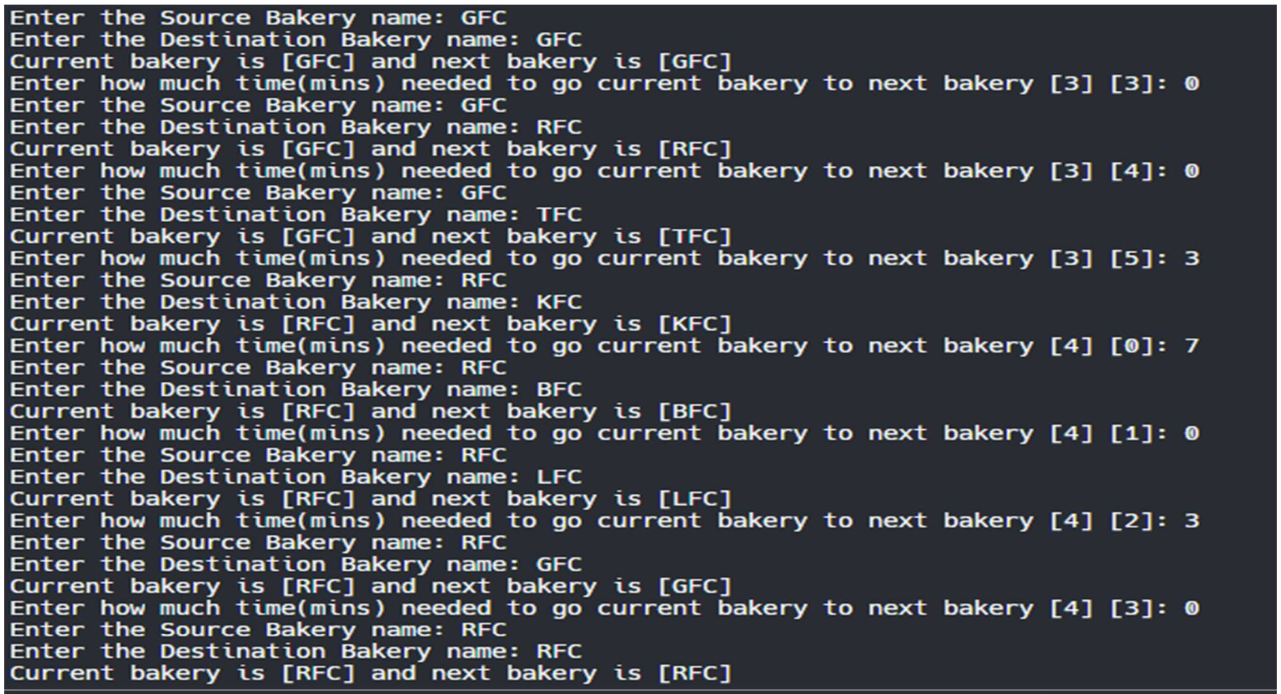


```
      }
      kruskal();
      cout << endl << "Spanning tree matrix: ";
      print();
      break;
    }

    case 2:
    {
      exit(0);
    }
    default:
      cout << "Invalid choice" << endl;
      cout << "Enter correct option!" << endl;
      break;
    }
  }

return 0;
}
```

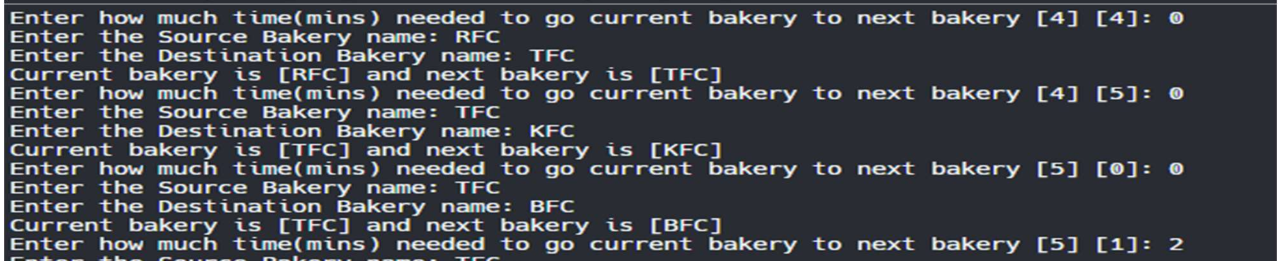
Here is switch case. User can choice their options. If user choice option 1 the main implementation part of Kruskal’s algorithm will be executed. Then the program will ask for how many bakeries are their form Baily Road to The ULAB Research Building. After entering the total number of bakeries, it will ask the source and destination bakery each time.



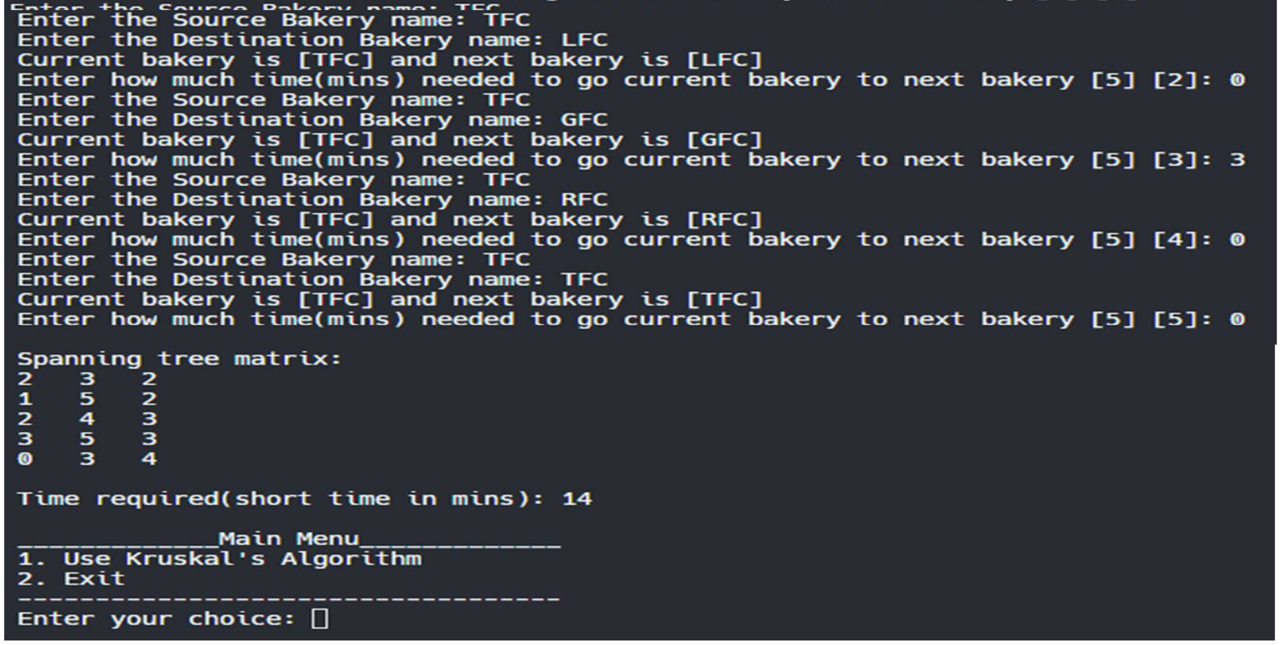
```
____________Main Menu____________
1. Use Kruskal's Algorithm
2. Exit
-----------------------------------
Enter your choice: 1

Enter the total number of Bakerys from ULAB Research Building to Baily Road: 6
Enter the Source Bakery name: KFC
Enter the Destination Bakery name: KFC
Current bakery is [KFC] and next bakery is [KFC]
Enter how much time(mins) needed to go current bakery to next bakery [0] [0]: 0
Enter the Source Bakery name: KFC
Enter the Destination Bakery name: BFC
Current bakery is [KFC] and next bakery is [BFC]
Enter how much time(mins) needed to go current bakery to next bakery [0] [1]: 5
Enter the Source Bakery name: KFC
Enter the Destination Bakery name: LFC
Current bakery is [KFC] and next bakery is [LFC]
Enter how much time(mins) needed to go current bakery to next bakery [0] [2]: 8
Enter the Source Bakery name: KFC
Enter the Destination Bakery name: GFC
Current bakery is [KFC] and next bakery is [GFC]
Enter how much time(mins) needed to go current bakery to next bakery [0] [3]: 4
Enter the Source Bakery name: KFC
Enter the Destination Bakery name: RFC
Current bakery is [KFC] and next bakery is [RFC]
Enter how much time(mins) needed to go current bakery to next bakery [0] [4]: 7
Enter the Source Bakery name: KFC
Enter the Destination Bakery name: TFC
Current bakery is [KFC] and next bakery is [TFC]
Enter how much time(mins) needed to go current bakery to next bakery [0] [5]: 0
Enter the Source Bakery name: BFC
Enter the Destination Bakery name: KFC
Current bakery is [BFC] and next bakery is [KFC]
Enter how much time(mins) needed to go current bakery to next bakery [1] [0]: 5
Enter the Source Bakery name: BFC
Enter the Destination Bakery name: BFC
Current bakery is [BFC] and next bakery is [BFC]
Enter how much time(mins) needed to go current bakery to next bakery [1] [1]: 0
Enter the Source Bakery name: BFC
Enter the Destination Bakery name: LFC
Current bakery is [BFC] and next bakery is [LFC]
Enter how much time(mins) needed to go current bakery to next bakery [1] [2]: 0
Enter the Source Bakery name: BFC
Enter the Destination Bakery name: GFC
Current bakery is [BFC] and next bakery is [GFC]
Enter how much time(mins) needed to go current bakery to next bakery [1] [3]: 0
Enter the Source Bakery name: BFC
Enter the Destination Bakery name: RFC
Current bakery is [BFC] and next bakery is [RFC]
Enter how much time(mins) needed to go current bakery to next bakery [1] [4]: 0
Enter the Source Bakery name: BFC
Enter the Destination Bakery name: TFC
Current bakery is [BFC] and next bakery is [TFC]
Enter how much time(mins) needed to go current bakery to next bakery [1] [5]: 2
Enter the Source Bakery name: LFC
Enter the Destination Bakery name: KFC
Current bakery is [LFC] and next bakery is [KFC]
Enter how much time(mins) needed to go current bakery to next bakery [2] [0]: 8
Enter the Destination Bakery name: BFC
Current bakery is [LFC] and next bakery is [BFC]
Enter how much time(mins) needed to go current bakery to next bakery [2] [1]: 0
Enter the Source Bakery name: LFC
Enter the Destination Bakery name: LFC
Current bakery is [LFC] and next bakery is [LFC]
Enter how much time(mins) needed to go current bakery to next bakery [2] [2]: 0
Enter the Source Bakery name: LFC
Enter the Destination Bakery name: GFC
Current bakery is [LFC] and next bakery is [GFC]
Enter how much time(mins) needed to go current bakery to next bakery [2] [3]: 0
Enter the Source Bakery name: LFC
Enter the Destination Bakery name: RFC
Current bakery is [LFC] and next bakery is [RFC]
Enter how much time(mins) needed to go current bakery to next bakery [2] [4]: 3
Enter the Source Bakery name: LFC
Enter the Destination Bakery name: TFC
Current bakery is [LFC] and next bakery is [TFC]
Enter how much time(mins) needed to go current bakery to next bakery [2] [5]: 0
Enter the Source Bakery name: GFC
Enter the Destination Bakery name: KFC
Current bakery is [GFC] and next bakery is [KFC]
Enter how much time(mins) needed to go current bakery to next bakery [3] [0]: 4
Enter the Source Bakery name: GFC
Enter the Destination Bakery name: BFC
Current bakery is [GFC] and next bakery is [BFC]
Enter how much time(mins) needed to go current bakery to next bakery [3] [1]: 0
Enter the Source Bakery name: GFC
Enter the Destination Bakery name: LFC
Current bakery is [GFC] and next bakery is [LFC]
Enter how much time(mins) needed to go current bakery to next bakery [3] [2]: 2
```



```
Enter the Source Bakery name: GFC
Enter the Destination Bakery name: GFC
Current bakery is [GFC] and next bakery is [GFC]
Enter how much time(mins) needed to go current bakery to next bakery [3] [3]: 0
Enter the Source Bakery name: GFC
Enter the Destination Bakery name: RFC
Current bakery is [GFC] and next bakery is [RFC]
Enter how much time(mins) needed to go current bakery to next bakery [3] [4]: 0
Enter the Source Bakery name: GFC
Enter the Destination Bakery name: TFC
Current bakery is [GFC] and next bakery is [TFC]
Enter how much time(mins) needed to go current bakery to next bakery [3] [5]: 3
Enter the Source Bakery name: RFC
Enter the Destination Bakery name: KFC
Current bakery is [RFC] and next bakery is [KFC]
Enter how much time(mins) needed to go current bakery to next bakery [4] [0]: 7
Enter the Source Bakery name: RFC
Enter the Destination Bakery name: BFC
Current bakery is [RFC] and next bakery is [BFC]
Enter how much time(mins) needed to go current bakery to next bakery [4] [1]: 0
Enter the Source Bakery name: RFC
Enter the Destination Bakery name: LFC
Current bakery is [RFC] and next bakery is [LFC]
Enter how much time(mins) needed to go current bakery to next bakery [4] [2]: 3
Enter the Source Bakery name: RFC
Enter the Destination Bakery name: GFC
Current bakery is [RFC] and next bakery is [GFC]
Enter how much time(mins) needed to go current bakery to next bakery [4] [3]: 0
Enter the Source Bakery name: RFC
Enter the Destination Bakery name: RFC
Current bakery is [RFC] and next bakery is [RFC]
```

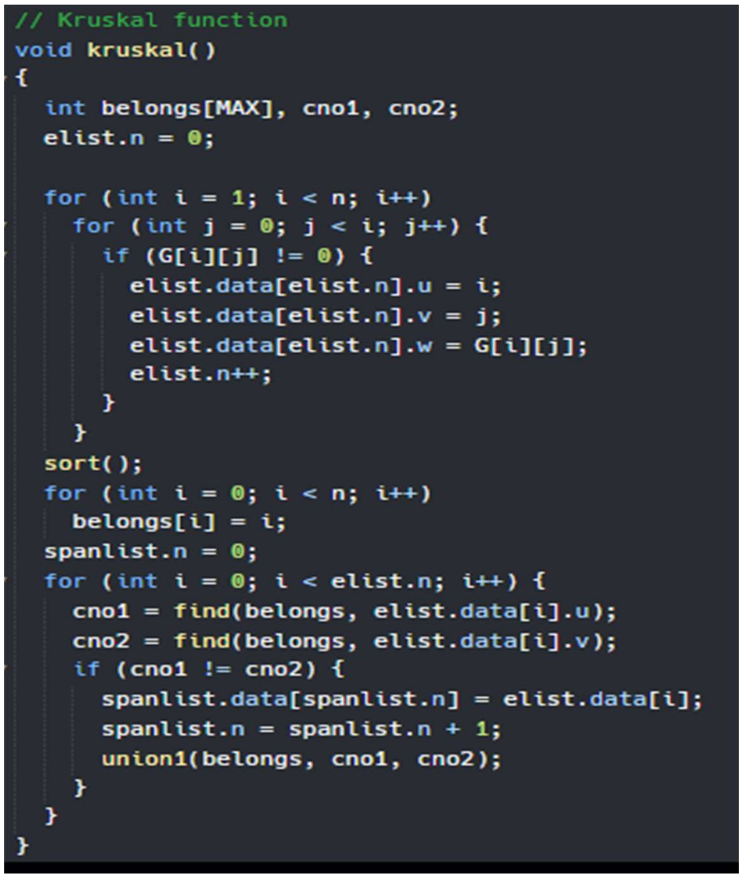


```
Enter how much time(mins) needed to go current bakery to next bakery [4] [4]: 0
Enter the Source Bakery name: RFC
Enter the Destination Bakery name: TFC
Current bakery is [RFC] and next bakery is [TFC]
Enter how much time(mins) needed to go current bakery to next bakery [4] [5]: 0
Enter the Source Bakery name: TFC
Enter the Destination Bakery name: KFC
Current bakery is [TFC] and next bakery is [KFC]
Enter how much time(mins) needed to go current bakery to next bakery [5] [0]: 0
Enter the Source Bakery name: TFC
Enter the Destination Bakery name: BFC
Current bakery is [TFC] and next bakery is [BFC]
Enter how much time(mins) needed to go current bakery to next bakery [5] [1]: 2
```

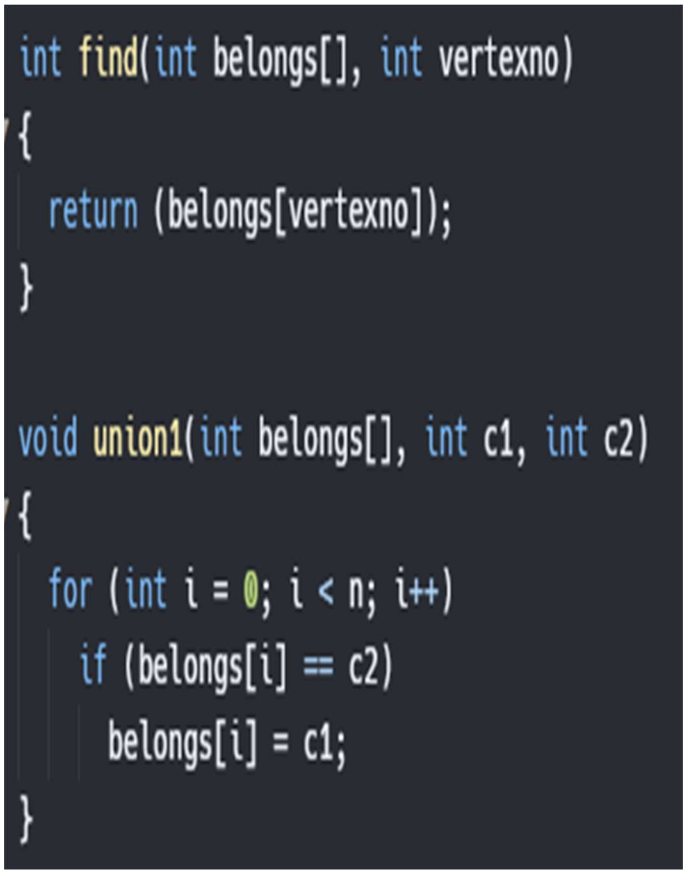


```
Enter the Source Bakery name: TFC
Enter the Destination Bakery name: LFC
Current bakery is [TFC] and next bakery is [LFC]
Enter how much time(mins) needed to go current bakery to next bakery [5] [2]: 0
Enter the Source Bakery name: TFC
Enter the Destination Bakery name: GFC
Current bakery is [TFC] and next bakery is [GFC]
Enter how much time(mins) needed to go current bakery to next bakery [5] [3]: 3
Enter the Source Bakery name: TFC
Enter the Destination Bakery name: RFC
Current bakery is [TFC] and next bakery is [RFC]
Enter how much time(mins) needed to go current bakery to next bakery [5] [4]: 0
Enter the Source Bakery name: TFC
Enter the Destination Bakery name: TFC
Current bakery is [TFC] and next bakery is [TFC]
Enter how much time(mins) needed to go current bakery to next bakery [5] [5]: 0

Spanning tree matrix:
2   3   2
1   5   2
2   4   3
3   5   3
0   3   4

Time required(short time in mins): 14

_____________Main Menu_____________
1. Use Kruskal's Algorithm
2. Exit
-----------------------------------
Enter your choice: 
```

Then the main function will take all the iputes and sort the weights of the bakeries means how many time needed to go from source bakery to destination bakery in ascending order from the funciton named Kruskal's. After sorting find and union function will be execuded. Find funciton will be execude to fine the parent of any vertex and union function will be executed to find a new parent for the edge which add to MST.

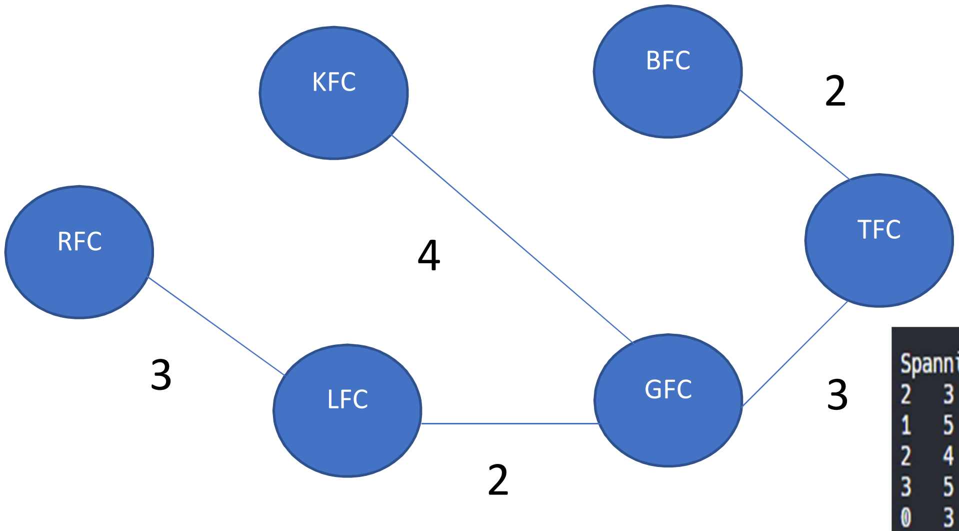


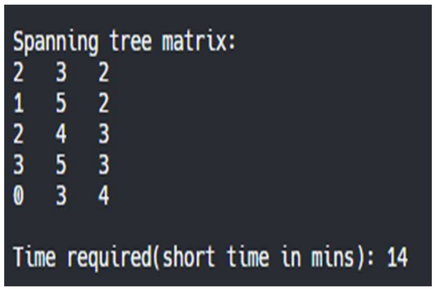
```
// Kruskal function
void kruskal()
{
  int belongs[MAX], cno1, cno2;
  elist.n = 0;

  for (int i = 1; i < n; i++)
    for (int j = 0; j < i; j++) {
      if (G[i][j] != 0) {
        elist.data[elist.n].u = i;
        elist.data[elist.n].v = j;
        elist.data[elist.n].w = G[i][j];
        elist.n++;
      }
    }
  sort();
  for (int i = 0; i < n; i++)
    belongs[i] = i;
  spanlist.n = 0;
  for (int i = 0; i < elist.n; i++) {
    cno1 = find(belongs, elist.data[i].u);
    cno2 = find(belongs, elist.data[i].v);
    if (cno1 != cno2) {
      spanlist.data[spanlist.n] = elist.data[i];
      spanlist.n = spanlist.n + 1;
      union1(belongs, cno1, cno2);
    }
  }
}
```



```
int find(int belongs[], int vertexno)
{
  return (belongs[vertexno]);
}

void union1(int belongs[], int c1, int c2)
{
  for (int i = 0; i < n; i++)
    if (belongs[i] == c2)
      belongs[i] = c1;
}
```

Output:





```
Spanning tree matrix:
2   3   2
1   5   2
2   4   3
3   5   3
0   3   4

Time required(short time in mins): 14
```

Justification:

The Kruskal's algorithm makes no compromise on producing a minimum-spanning tree. On any given set of locations, the Kruskal's algorithm will always ensure a resultant least weight graph in terms of cost or distance or any parameter in question. By applying the Kruskal's algorithm, the distance traveled by Labels drivers when making deliveries will be greatly reduced, time will be utilized effectively and costs associated, mostly in terms of fuel, may highly be minimized.

Addressing complex engineering problems:

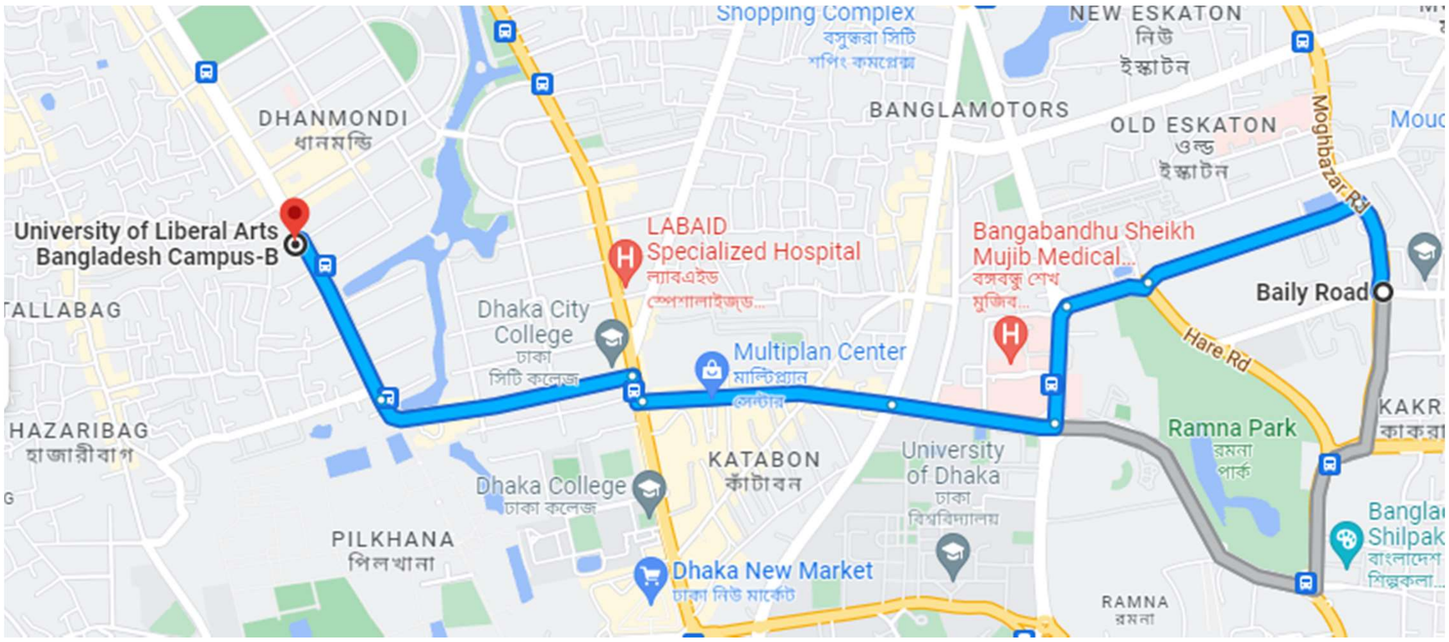
In this problem, you must visit all the bakeries on the way from Baily Road to ULAB Research Building, but you are short of time. It is not so an easy task to visit all the bakeries in short time. So, we have decided to implement Kruskal's algorithm to solve the problem because Kruskal's algorithm can give the optimal solution. It just ignores loops and relax the minimum time from multiple paths, sort the times (mins) in ascending order then create spanning matrix and sum up how much time is needed.

Current Status:

We have designed a simple program to learn how to use the Kruskal algorithm. Building this program has enabled us to look forward to the real-life applications where these algorithms could be used to solve real world problems. This program can give the optimal solution of the given problem and finds the shortest path after visiting all the vertices.

Future Scope:

The program we created already has many applications out there in the world i.e., Land Networks, Landing cables. We hope to implement Kruskal algorithm in bigger projects like developing a tour app, where users will be able to experience the tour in minimum cost possible or we will build a game where we might need to create a map for the character to explore the destinations in minimum time.


Shopping Complex
NEW ESKATON
BANGLAMOTORS
OLD ESKATON
DHANMONDI
Moghbazar Rd
University of Liberal Arts Bangladesh Campus-B
LABAID Specialized Hospital
Bangabandhu Sheikh Mujib Medical...
Baily Road
Dhaka City College
Multiplan Center
Hare Rd
HAZARIBAG
Ramna Park
KATABON
University of Dhaka
Dhaka College
PILKHANA
Dhaka New Market
RAMNA
Shilpak