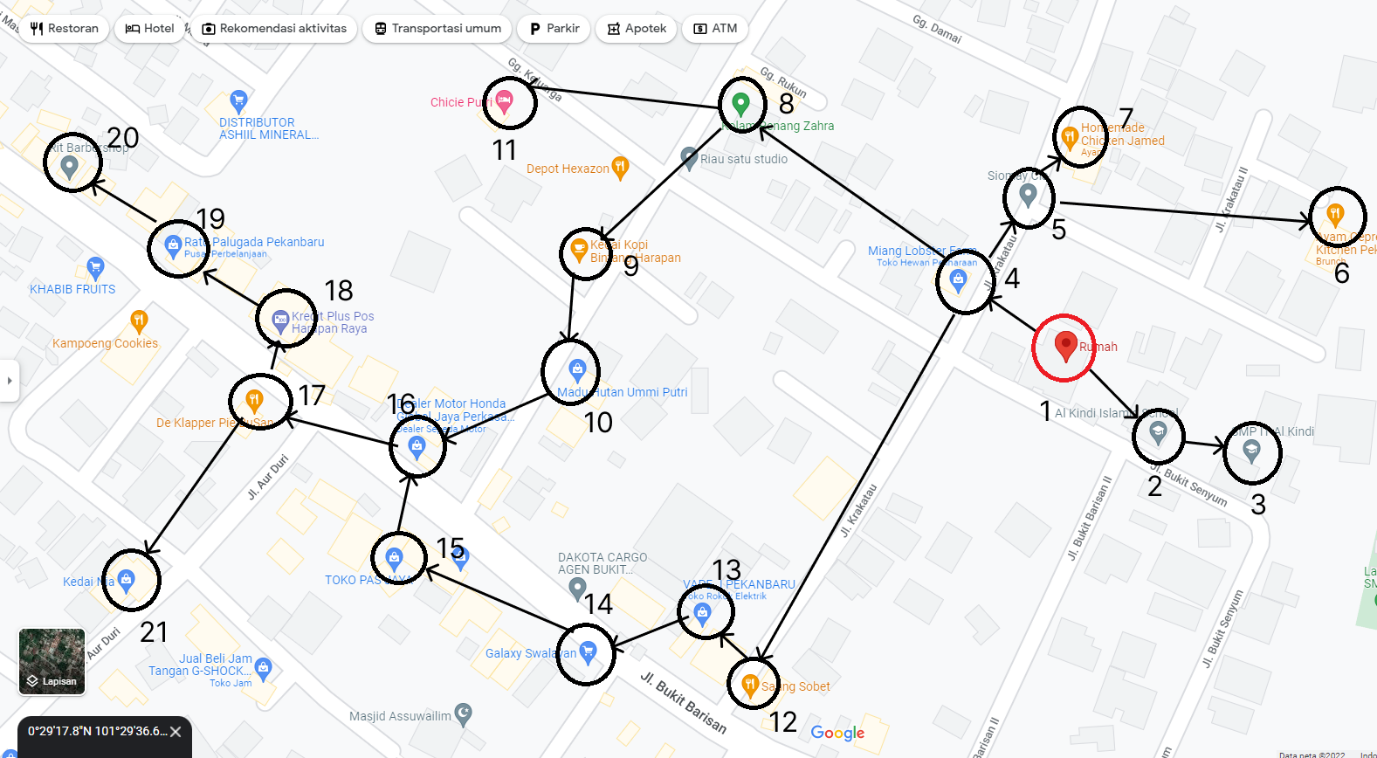
**Persiapan Final Project – Graph Traversal**

**Wahyu Andhika Rizaldi – 5027211003**

**Struktur Data B**

**Github : https://github.com/wahyuandhikarizaldi/Struktur-Data-B/tree/main/Persiapan%20FP%20%E2%80%93%20Graph%20Traversal**

1. Graf Berarah dari rumah



2 dan 3 :

* Adjacency List BFS

#include <iostream>

#include <list>

#include <iterator>

#include <queue>

using namespace std;

class Graph

{

protected:

    int vertex;

    list<int> adj[100];

public:

    Graph(int v)

    {

        vertex = v;

    }

    void addVertex(int v1, int v2)

    {

        adj[v1].push\_back(v2);

    }

    void showVertex()

    {

        for (int i = 1; i <= vertex; i++)

        {

            cout << i << " --> ";

            list<int>::iterator it; // iterator for list

            for (it = adj[i].begin(); it != adj[i].end(); it++)

            {

                cout << \*it << " "; // \*it is the value of the node/ngeakses pointer nya

            }

            cout << endl;

        }

    }

    void BFS(int start)

    {

        bool visited[100];

        // initialize all nodes as not visited

        for (int i = 1; i <= vertex; i++)

        {

            visited[i] = false;

        }

        queue<int> q;

        q.push(start);

        visited[start] = true;

        while (!q.empty())

        {

            int front = q.front(); // get the front element

            list<int>::iterator it;

            for (it = adj[q.front()].begin(); it != adj[q.front()].end(); it++) // iterate through the list

            {

                if (!visited[\*it]) // if the node is not visited

                {

                    q.push(\*it);         // push the node to the queue

                    visited[\*it] = true; // mark the node as visited

                }

                q.pop(); // remove the front element

            }

        }

    }

};

int main()

{

    Graph graf(21);

    graf.addVertex(1, 2);

    graf.addVertex(2, 3);

    graf.addVertex(1, 4);

    graf.addVertex(4, 5);

    graf.addVertex(5, 6);

    graf.addVertex(5, 7);

    graf.addVertex(4, 8);

    graf.addVertex(8, 11);

    graf.addVertex(8, 9);

    graf.addVertex(9, 10);

    graf.addVertex(10, 16);

    graf.addVertex(4, 12);

    graf.addVertex(12, 13);

    graf.addVertex(13, 14);

    graf.addVertex(14, 15);

    graf.addVertex(15, 16);

    graf.addVertex(16, 17);

    graf.addVertex(17, 21);

    graf.addVertex(17, 18);

    graf.addVertex(18, 19);

    graf.addVertex(19, 20);

    graf.showVertex();

    graf.BFS(1);

}

Output :

1 --> 2 4

2 --> 3

3 -->

4 --> 5 8 12

5 --> 6 7

6 -->

7 -->

8 --> 11 9

9 --> 10

10 --> 16

11 -->

12 --> 13

13 --> 14

14 --> 15

15 --> 16

16 --> 17

17 --> 21 18

18 --> 19

19 --> 20

20 -->

21 -->

* Adjacency Matrix BFS

#include <iostream>

#include <list>

using namespace std;

class Graph

{

  int numVertices;

  list<int> \*adjMatrix;

  bool \*visited;

  public:

    Graph(int vertices);

    void addEdge(int src, int dest);

    void BFS(int startVertex);

    void showMatrix();

};

Graph::Graph(int vertices)

{

  numVertices = vertices;

  adjMatrix = new list<int>[vertices];

}

// Add edges to the graph

void Graph::addEdge(int src, int dest)

{

  adjMatrix[src].push\_back(dest);

  adjMatrix[src].sort();

}

// BFS algorithm

void Graph::BFS(int startVertex)

{

  visited = new bool[numVertices];

  for (int i = 0; i < numVertices; i++)

    visited[i] = false;

  list<int> queue;

  visited[startVertex] = true;

  queue.push\_back(startVertex);

  list<int>::iterator i;

  while (!queue.empty())

  {

    int currVertex = queue.front();

    // Fungsi memberhentikan BFS

    // if (currVertex == endVertex)

    // {

    //   break;

    // }

    //

    cout << "(V" << currVertex << ")";

    queue.pop\_front();

    for (i = adjMatrix[currVertex].begin(); i != adjMatrix[currVertex].end(); ++i)

    {

      int adjVertex = \*i;

      if (!visited[adjVertex])

      {

        visited[adjVertex] = true;

        queue.push\_back(adjVertex);

      }

    }

  }

}

void Graph::showMatrix()

{

  for (int i = 0; i < numVertices; i++)

  {

    list<int>::iterator it;

    int track = 0;

    int through = 0;

    for (it = adjMatrix[i].begin(); it != adjMatrix[i].end(); ++it)

    {

      // int track = 0;

      int temp = \*it;

      if (through == 0)

      {

        while (track < temp)

        {

          cout << "0 ";

          ++track;

          through++;

        }

      }

      else

      {

        while ((track + 1) < temp)

        {

          cout << "0 ";

          ++track;

          through++;

        }

      }

      cout << "1 ";

      through++;

    }

    while ((numVertices - through) > 0)

    {

      through++;

      cout << "0 ";

    }

    cout << endl;

  }

}

int main()

{

Graph graf(22);

graf.addEdge(1, 2);

graf.addEdge(2, 3);

graf.addEdge(1, 4);

graf.addEdge(4, 5);

graf.addEdge(5, 6);

graf.addEdge(5, 7);

graf.addEdge(4, 8);

graf.addEdge(8, 11);

graf.addEdge(8, 9);

graf.addEdge(9, 10);

graf.addEdge(10, 16);

graf.addEdge(4, 12);

graf.addEdge(12, 13);

graf.addEdge(13, 14);

graf.addEdge(14, 15);

graf.addEdge(15, 16);

graf.addEdge(16, 17);

graf.addEdge(17, 21);

graf.addEdge(17, 18);

graf.addEdge(18, 19);

graf.addEdge(19, 20);

  graf.BFS(1);

  cout << endl;

  graf.showMatrix();

  return 0;

}

Output :

(V1)(V2)(V4)(V3)(V5)(V8)(V12)(V6)(V7)(V9)(V11)(V13)(V10)(V14)(V16)(V15)(V17)(V18)(V21)(V19)(V20)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0