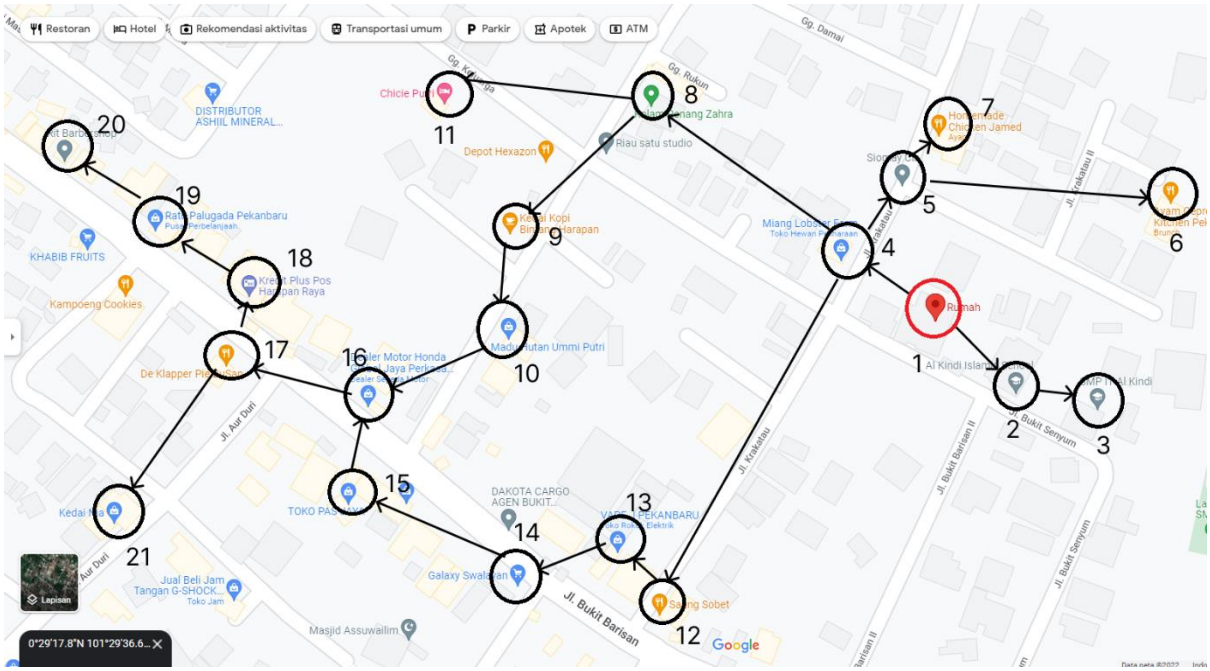
Persiapan Final Project – Graph Traversal

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<bits/stdc++.h>
using namespace std;

class Graph{
private:
    int V;
    vector<list<int>> adj;
public:
    Graph(int V){
        this->V = V;
        adj.resize(V);
    }
```

```
    void addEdge(int v, int w){
        adj[v].push_back(w);
    }

    void showVertex()
{
    for (int i = 1; i <= V; 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 s, int d){
        vector<bool> visited;
        visited.resize(V,false);

        list<int> queue;

        visited[s] = true;
        queue.push_back(s);

        while(!queue.empty()){
            s = queue.front();
            cout << "(V" << s << ")";
            if (s == d) return;
            queue.pop_front();
            for (auto adjecent: adj[s]){
                if (!visited[adjecent]){
                    visited[adjecent] = true;
                    queue.push_back(adjecent);
                }
            }
        }
    }
};



int main()
{
    Graph g(21);
g.addEdge(1, 2);
```

```
g.addEdge(2, 3);
g.addEdge(1, 4);
g.addEdge(4, 5);
g.addEdge(5, 6);
g.addEdge(5, 7);
g.addEdge(4, 8);
g.addEdge(8, 11);
g.addEdge(8, 9);
g.addEdge(9, 10);
g.addEdge(10, 16);
g.addEdge(4, 12);
g.addEdge(12, 13);
g.addEdge(13, 14);
g.addEdge(14, 15);
g.addEdge(15, 16);
g.addEdge(16, 17);
g.addEdge(17, 21);
g.addEdge(17, 18);
g.addEdge(18, 19);
g.addEdge(19, 20);

g.BFS(1, 21);
cout << endl;
g.showVertex();

    return 0;
}
```

Output :

```
(V1)(V2)(V4)(V3)(V5)(V8)(V12)(V6)(V7)(V11)(V9)(V13)(V10)(V14)(V16)(V15)(V17)(V
18)(V19)(V20)
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)(V
18)(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
```