LAPORAN TUGAS BESAR
SISTEM NAVIGASI KAMPUS

Laporan ini Disusun Untuk Memenuhi Tugas Besar Mata Kuliah Struktur Data

Semester Ganjil/Tahun 2024

Dosen Pengampu: Wahyu Andi Saputra

Disusun Oleh :

Arzario Irsyad Al Fatih	2211104032
Satria Ariq Adelard Dompas	2211104033
Ade Fatkhul Anam	2211104051

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

A. Deskripsi Topik Tugas Besar: Sistem Navigasi Kampus

Study Case

Telkom University kampus Bandung memiliki beberapa gedung dan jalan penghubung di antara gedung-gedung tersebut. Setiap gedung adalah node, dan jalan penghubung antar gedung adalah edge. Anda bertugas membangun sistem navigasi untuk membantu pengguna menemukan rute terpendek di antara gedung-gedung di kampus.

Telkom University memiliki banyak gedung yang terhubung oleh jalan-jalan sebagai penghubung antar gedung. Untuk mempermudah mahasiswa, dosen, dan pengunjung dalam menjelajahi kampus, diperlukan sebuah sistem navigasi yang dapat membantu mereka menemukan rute terpendek dari satu gedung ke gedung lainnya. Sistem ini akan mengandalkan prinsip-prinsip struktur data graf untuk merepresentasikan hubungan antar gedung dan jalan.

Deskripsi Masalah

- Node (Vertex)

  Merepresentasikan gedung-gedung di kampus. Setiap gedung memiliki atribut seperti nama atau kode Gedung.

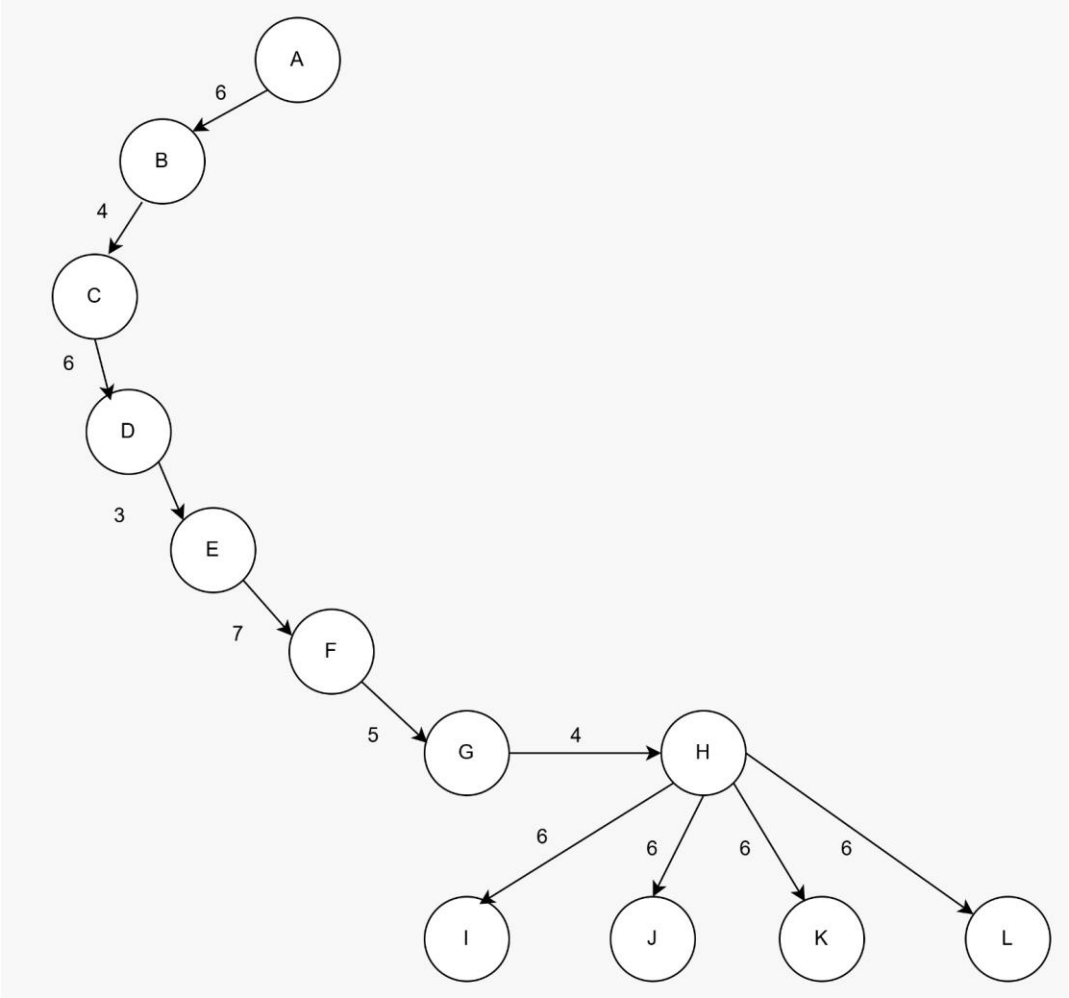
  Keterangan Gedung

  - A : Fakultas Ilmu Terapan
  - B : Fakultas Ekonomi Bisnis
  - C : Fakultas Komunikasi Bisnis
  - D : Fakultas Industri Kreatif
  - E : Tel-U Comvention Hall
  - F : Gedung Kuliah Umum
  - G : Gedung Kemahasiswaan
  - H : Gedung Rektorat/Gedung Marketing
  - I : Fakultas Teknik Elektro/Gedung Keuangan
  - J : Gedung Serba Guna
  - K : Fakultas Teknik Informatika
  - L : Fakultas Rekayasa Industri

- Edge

  Merepresentasikan jalan penghubung antar gedung. Setiap edge memiliki bobot (weight) yang bisa berupa jarak fisik, waktu tempuh, atau tingkat kesulitan jalan.
- Graf

  Direpresentasikan sebagai graf berbobot (weighted graph). Jenis graf dapat berupa graf berarah atau tidak berarah, tergantung pada kondisi jalan di kampus.



B. Output Running Script

- Cari Gedung Tersibuk

```
========== Sistem Navigasi Kampus ==========
1. Cari gedung tersibuk
2. Cari rute tercepat antar gedung
3. Tampilkan semua jarak antar gedung
0. Keluar
Pilihan: 1

Gedung tersibuk (paling sering dilewati):
- L: 1 kali
- E: 1 kali
- K: 1 kali
- J: 1 kali
- D: 1 kali
- C: 1 kali
- I: 1 kali
- F: 1 kali
- G: 1 kali
- B: 1 kali
- H: 1 kali
```

- Cari Rute Tercepat

```
========== Sistem Navigasi Kampus ==========
1. Cari gedung tersibuk
2. Cari rute tercepat antar gedung
3. Tampilkan semua jarak antar gedung
0. Keluar
Pilihan: 2

Masukkan gedung awal: B
Masukkan gedung tujuan: H

Rute terpendek dari B ke H adalah:
B -> C -> D -> E -> F -> G -> H
Jarak: 29 meter
```

- Tampilkan Semua Jarak antar Gedung

```
========== Sistem Navigasi Kampus ==========
1. Cari gedung tersibuk
2. Cari rute tercepat antar gedung
3. Tampilkan semua jarak antar gedung
0. Keluar
Pilihan: 3

Koneksi antar gedung:
- Gedung G terhubung dengan:
  - Gedung H dengan jarak 4 meter

- Gedung F terhubung dengan:
  - Gedung G dengan jarak 5 meter

- Gedung E terhubung dengan:
  - Gedung F dengan jarak 7 meter

- Gedung H terhubung dengan:
  - Gedung I dengan jarak 6 meter
  - Gedung J dengan jarak 6 meter
  - Gedung K dengan jarak 6 meter
  - Gedung L dengan jarak 6 meter

- Gedung B terhubung dengan:
  - Gedung C dengan jarak 4 meter

- Gedung D terhubung dengan:
  - Gedung E dengan jarak 3 meter

- Gedung C terhubung dengan:
  - Gedung D dengan jarak 6 meter

- Gedung A terhubung dengan:
  - Gedung B dengan jarak 5 meter
```

C. Script

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <string>
#include <queue>
#include <limits>
#include <algorithm> // Header untuk fungsi reverse

using namespace std;

class Graph {
private:
    unordered_map<string, vector<pair<string, int>>> adjList;

public:
    void addEdge(const string &u, const string &v, int weight) {
```

```
        adjList[u].emplace_back(v, weight);
    }

    void busiestNodes() {
        unordered_map<string, int> nodeFrequency;

        for (const auto &node : adjList) {
            for (const auto &neighbor : node.second) {
                nodeFrequency[neighbor.first]++;
            }
        }

        cout << "\nGedung tersibuk (paling sering dilewati):\n";
        for (const auto &entry : nodeFrequency) {
            cout << "- " << entry.first << ": " << entry.second << "
kali" << endl;
        }
    }

    void shortestPath(const string &start, const string &end) {
        unordered_map<string, int> distances;
        unordered_map<string, string> previous;
        for (const auto &node : adjList) {
            distances[node.first] = numeric_limits<int>::max();
        }
        distances[start] = 0;

        auto compare = [](pair<int, string> a, pair<int, string> b)
{ return a.first > b.first; };
        priority_queue<pair<int, string>, vector<pair<int, string>>,
decltype(compare)> pq(compare);

        pq.emplace(0, start);

        while (!pq.empty()) {
            auto [currentDist, currentNode] = pq.top();
            pq.pop();

            if (currentDist > distances[currentNode]) continue;

            for (const auto &neighbor : adjList[currentNode]) {
                int newDist = currentDist + neighbor.second;
                if (newDist < distances[neighbor.first]) {
                    distances[neighbor.first] = newDist;
                    previous[neighbor.first] = currentNode;
                    pq.emplace(newDist, neighbor.first);
                }
            }
```

```
        }

        if (distances[end] == numeric_limits<int>::max()) {
            cout << "\nTidak ada rute dari " << start << " ke " <<
end << "." << endl;
        } else {
            cout << "\nRute terpendek dari " << start << " ke " <<
end << " adalah: " << endl;
            string current = end;
            vector<string> path;
            while (current != start) {
                path.push_back(current);
                current = previous[current];
            }
            path.push_back(start);
            reverse(path.begin(), path.end());

            for (size_t i = 0; i < path.size(); ++i) {
                cout << path[i];
                if (i != path.size() - 1) cout << " -> ";
            }
            cout << "\nJarak: " << distances[end] << " meter" <<
endl;
        }
    }

    void displayConnections() {
        cout << "\nKoneksi antar gedung:\n";
        for (const auto &start : adjList) {
            cout << "- Gedung " << start.first << " terhubung
dengan:\n";
            for (const auto &neighbor : start.second) {
                cout << "  - Gedung " << neighbor.first << " dengan
jarak " << neighbor.second << " meter\n";
            }
            cout << endl;
        }
    }
};

int main() {
    Graph graph;

    // Menambahkan edge sesuai deskripsi hubungan gedung
    graph.addEdge("A", "B", 5);
    graph.addEdge("B", "C", 4);
    graph.addEdge("C", "D", 6);
    graph.addEdge("D", "E", 3);
```

```
    graph.addEdge("E", "F", 7);
    graph.addEdge("F", "G", 5);
    graph.addEdge("G", "H", 4);
    graph.addEdge("H", "I", 6);
    graph.addEdge("H", "J", 6);
    graph.addEdge("H", "K", 6);
    graph.addEdge("H", "L", 6);

    int choice;
    do {
        cout << "\n========== Sistem Navigasi Kampus ==========" <<
endl;
        cout << "1. Cari gedung tersibuk" << endl;
        cout << "2. Cari rute tercepat antar gedung" << endl;
        cout << "3. Tampilkan semua jarak antar gedung" << endl;
        cout << "0. Keluar" << endl;
        cout << "Pilihan: ";
        cin >> choice;

        switch (choice) {
        case 1:
            graph.busiestNodes();
            break;
        case 2: {
            string start, end;
            cout << "\nMasukkan gedung awal: ";
            cin >> start;
            cout << "Masukkan gedung tujuan: ";
            cin >> end;
            graph.shortestPath(start, end);
            break;
        }
        case 3:
            graph.displayConnections();
            break;
        case 0:
            cout << "Keluar dari sistem." << endl;
            break;
        default:
            cout << "Pilihan tidak valid." << endl;
        }
    } while (choice != 0);

    return 0;
}
```

Deskripsi

Class: Graph

1. void addEdge(const string &u, const string &v, int weight)
   - Tujuan: Menambahkan edge ke graf dengan merepresentasikan hubungan antara dua gedung.
   - Parameter:
     - u : Nama gedung asal.
     - v : Nama gedung tujuan.
     - weight : Bobot edge, berupa jarak antara gedung u dan v.
   - Cara Kerja:
     1. Menyimpan pasangan (v, weight) dalam daftar tetangga (adjacency list) milik u.

2. void busiestNodes()
   - Tujuan: Menghitung frekuensi keterhubungan setiap gedung (berapa kali sebuah gedung menjadi tujuan dari gedung lain).
   - Cara Kerja:
     1. Iterasi setiap node dalam adjacency list.
     2. Untuk setiap tetangga (neighbor) dari node, tingkatkan nilai frekuensinya dalam map nodeFrequency.
     3. Cetak gedung-gedung dengan jumlah koneksi sebagai tujuan (frekuensi).

3. void shortestPath(const string &start, const string &end)
   - Tujuan: Menemukan rute terpendek antara gedung start dan end menggunakan algoritma Dijkstra.
   - Parameter:
     - start : Gedung awal.
     - end : Gedung tujuan.
   - Cara Kerja:
     1. Inisialisasi semua jarak (distances) ke maksimum (numeric_limits<int>::max()), kecuali gedung awal (start) yang diatur ke 0.
     2. Gunakan priority queue untuk menyimpan pasangan (jarak, node) dengan prioritas minimum jarak.

3. Iterasi selama priority queue tidak kosong:
    - Ambil node dengan jarak terkecil.
    - Periksa semua tetangga node tersebut:
    - Jika jarak baru lebih kecil, perbarui distances dan tambahkan ke priority queue.
4. Setelah selesai:
    - Jika jarak ke end masih maksimum, berarti tidak ada jalur.
    - Jika jalur ditemukan, rekonstruksi jalur dari end ke start menggunakan map previous.
5. Cetak jalur dan total jarak.

4. void displayConnections()
    - Tujuan: Menampilkan semua koneksi antar gedung beserta jarak di antaranya.
    - Cara Kerja:
        1. Iterasi setiap gedung dalam adjacency list.
        2. Untuk setiap gedung, cetak daftar tetangganya dan jarak ke setiap tetangga.

Function: main()

1. Inisialisasi Graf
    - Tujuan: Membuat graf dengan menambahkan edge sesuai deskripsi hubungan gedung.
    - Cara Kerja:
        - Panggil addEdge() untuk setiap pasangan gedung dengan jaraknya.

2. Menu Interaktif
    - Tujuan: Memberikan opsi kepada pengguna untuk menggunakan sistem navigasi.
    - Cara Kerja:
        1. Pilihan 1: Menampilkan gedung tersibuk (paling sering dilewati).
            - Memanggil busiestNodes().
        2. Pilihan 2: Menemukan rute tercepat antar dua gedung.

- Meminta input gedung awal dan tujuan, lalu memanggil shortestPath(start, end).

3. Pilihan 3: Menampilkan semua koneksi dan jarak antar gedung.
   - Memanggil displayConnections().
4. Pilihan 0: Keluar dari program.
5. Default: Menampilkan pesan kesalahan jika pilihan tidak valid.

D. Pembagian Tugas

1. Arzario Irsyad Al Fatih

   Tugas:

   Implementasi Struktur Data dan Pengelolaan Graf

   Deskripsi:

   - Bertanggung jawab untuk mengimplementasikan dan mengelola graf menggunakan struktur data adjacency list.
   - Membangun sistem untuk menambahkan hubungan antar gedung (menggunakan fungsi addEdge).
   - Mengelola data gedung yang ada, serta memastikan hubungan antar gedung ditambahkan dengan benar.

   Fungsi yang Dikerjakan:

   - addEdge(const string &u, const string &v, int weight): Menambahkan edge antar gedung beserta bobot (jarak) ke dalam adjacency list.

2. Satria Ariq Adelard Dompas

   Tugas:

   Menangani Pencarian Gedung Tersibuk

   Deskripsi:

   - Bertanggung jawab untuk menghitung gedung yang paling sering dilewati, atau yang memiliki frekuensi keterhubungan terbanyak.
   - Mengimplementasikan fungsi untuk menghitung dan menampilkan gedung-gedung yang sering dikunjungi berdasarkan koneksi.

   Fungsi yang Dikerjakan:

- busiestNodes(): Menghitung frekuensi keterhubungan setiap gedung (berapa kali gedung tersebut menjadi tujuan atau dilalui), dan menampilkan gedung yang paling sering dilewati.

3. Ade Fatkhul Anam

   Tugas:

   Menangani Pencarian Rute Terpendek

   Deskripsi:

   - Bertanggung jawab untuk mengimplementasikan algoritma Dijkstra untuk mencari rute terpendek antar gedung.
   - Mengatur prioritas queue untuk mencari jalur dengan jarak paling pendek dan mengembalikan rute serta jaraknya.
   - Mengimplementasikan sistem untuk menampilkan rute terpendek dan jarak antar gedung.

   Fungsi yang Dikerjakan:

   - shortestPath(const string &start, const string &end): Menerapkan algoritma Dijkstra untuk mencari rute terpendek antara gedung start dan gedung end.

   Pembagian Tugas Umum untuk Semua Anggota:

   Arzario Irsyad Al Fatih, Satria Ariq Adelard Dompas, dan Ade Fatkhul Anam akan berkolaborasi dalam:

   - Mengintegrasikan Semua Fungsi: Memastikan bahwa setiap fungsi berfungsi dengan baik dan terintegrasi dengan benar.
   - Pengujian Sistem: Menguji sistem untuk memastikan semua fungsi berjalan dengan baik.
   - Dokumentasi: Menulis laporan tugas besar, menyiapkan presentasi, dan memberikan penjelasan tentang sistem kepada dosen dan audiens.