



# SAFERANGERS



Presented by :

- |                                       |        |
|---------------------------------------|--------|
| 1) Norsyahirah Bt Sapuan              | 211283 |
| 2) Nur A'in A'liya Bt Ahmad Alauddin  | 211278 |
| 3) Nurul Natasha Nabilah Bt Mohd Dino | 211909 |





# SCENARIO AND GOAL

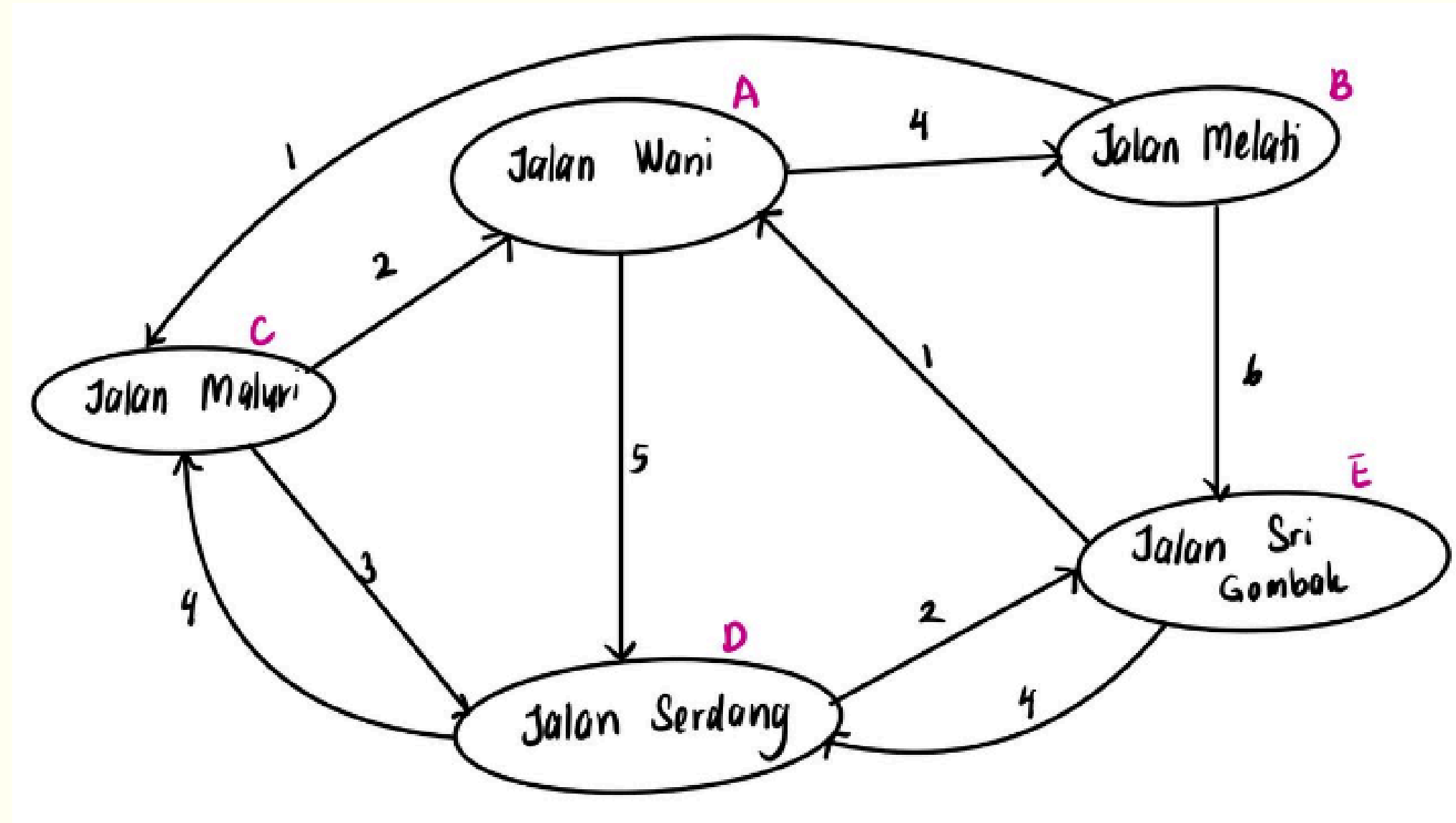
## **Scenario :**



**Kuala Lumpur is hit by a powerful earthquake, leading to widespread devastation across the city. The immediate priority is to rescue trapped individuals, provide medical assistance to the injured, and restore essential services. Emergency services, including the Fire and Rescue Department, Civil Defence Force, and medical teams, are mobilized. However, the damaged roads, collapsed buildings, and ongoing aftershocks create significant challenges for the responders. Coordinating these efforts requires an efficient, real-time decision-making system that can handle the complexity and urgency of the situation.**



**Based on the illustration below there is the road's name (a string), its value represents distance in kilometers (an integer). Then, implement an algorithm to assist emergency services in finding the shortest path from Jalan Serdang as the evacuation site.**





**Goal :**

**To develop an algorithm that helps emergency services in Malaysia to find the shortest and safest paths to reach affected areas. Compare each route from each place that states in the illustration, to choose which is the shortest path. The starting place emergency service should be is the Jalan Serdang.**



# ALGORITHM

**Algorithm Paradigm :** The paradigm used for Dijkstra's algorithm is greedy algorithm. The greedy approach builds up a solution one by one, always choosing the next one that offers the most immediate benefit. In the context of Dijkstra's algorithm, the "immediate benefit" is the shortest path.



## PSEUDOCODE :

```
function Dijkstra(Graph, startVertex):
    distances = new array(size of Graph)
    visited = new array(size of Graph)
    for i from 0 to size of Graph:
        distances[i] = INFINITY
    distances[startVertex] = 0
    for i from 0 to size of Graph:
        u = findMinDistanceVertex(distances, visited)
        if u == -1:
            break
        visited[u] = true
        for v from 0 to size of Graph:
            if not visited[v] and Graph.adjMatrix[u][v] != 0
and distances[u] != INFINITY:
                alt = distances[u] + Graph.adjMatrix[u][v]
                if alt < distances[v]:
                    distances[v] = alt
    return distances
```

```
function findVertexIndex(Graph, data):
    for i from 0 to size of Graph.vertexData:
        if Graph.vertexData[i] equals data:
            return i
    return -1

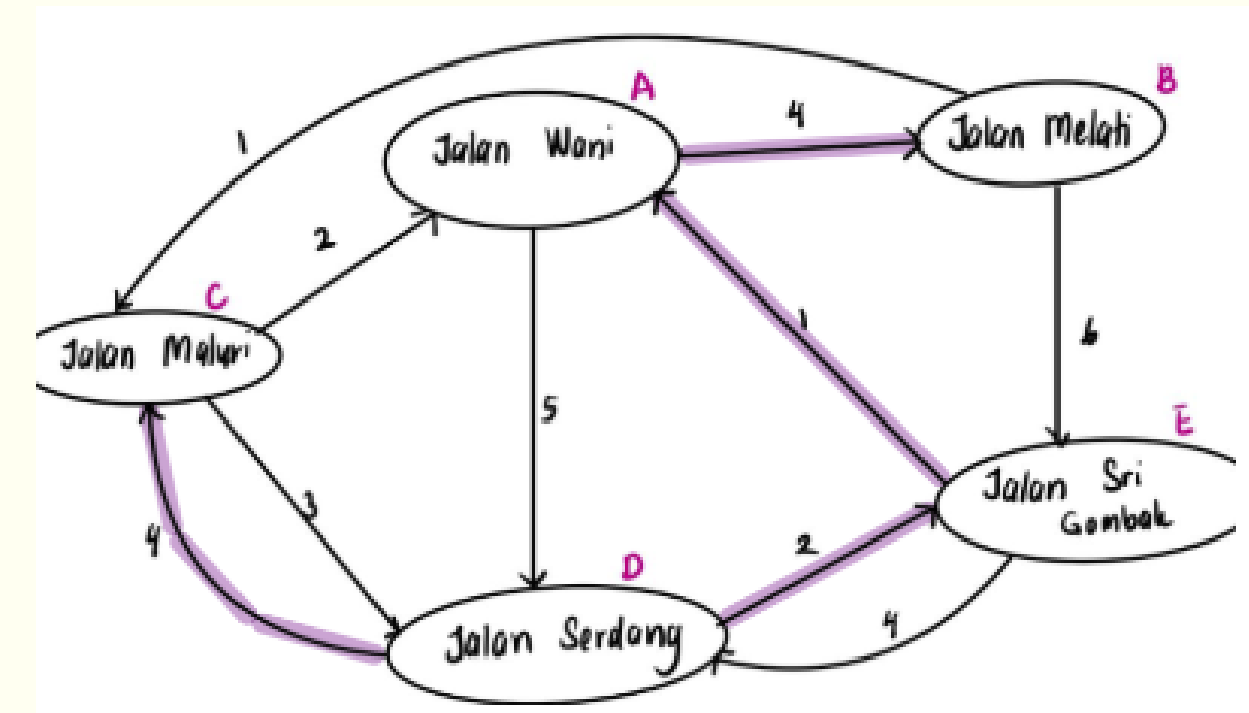
function findMinDistanceVertex(distances, visited):
    minDistance = INFINITY
    minIndex = -1
    for i from 0 to size of distances:
        if not visited[i] and distances[i] < minDistance:
            minDistance = distances[i]
            minIndex = i
    return minIndex
```

# OUTPUT

```
Shortest distance from Jalan Serdang (Evacuation Site) to Jalan Wani: 3
Shortest distance from Jalan Serdang (Evacuation Site) to Jalan Melati: 7
Shortest distance from Jalan Serdang (Evacuation Site) to Jalan Maluri: 4
Shortest distance from Jalan Serdang (Evacuation Site) to Jalan Serdang: 0
Shortest distance from Jalan Serdang (Evacuation Site) to Jalan Sri Gombak: 2
```



The output of the algorithm shows the shortest path between Jalan Serdang, which is the evacuation site, to all other places (vertices). For example, the shortest path from Jalan Serdang to Jalan Wani is 5 and from Jalan Serdang to Jalan Melati is 7. Therefore, to find the total of the shortest path for this scenario is by highlighting the path based on the coding output and summing all the values of the paths which is also known as weighted edges. The highlighted path, as determined by the algorithm, includes the following routes: Jalan Serdang to Jalan Sri Gombak, Jalan Sri Gombak to Jalan Wani, Jalan Wani to Jalan Melati, and Jalan Serdang to Jalan Maluri. Therefore, the total of these paths is  $2 + 1 + 4 + 4 = 11$  and this is also the value of the shortest path by using Dijkstra Algorithm.





# ALGORITHM ANALYSIS



The parts **needing recurrence**, and the optimization function :

- **Initialize** : Create arrays to track distances from the start vertex to all other vertices (distances) and to track whether a vertex has been visited (visited). And also set the distance to the start vertex to 0 and all other distances to infinity.
- **Relaxation** : For the selected vertex  $u$ , the algorithm examines each of its unvisited neighbors  $v$ . If a shorter path to  $v$  through  $u$  is found, the distance to  $v$  is updated. This is called "relaxation" and ensures that the shortest known distance to each vertex is always accurate.

The **key optimization** in Dijkstra's algorithm is the use of a priority queue to efficiently select the unvisited vertex with the smallest known distance. In the algorithm, `findMinDistanceVertex` performs this task linearly, but using a priority queue reduces the time complexity significantly.

**Correctness:** Dijkstra's algorithm is correct because it ensures that once a vertex's shortest path is finalized, its shortest path cannot be altered by later updates from vertices with longer paths. This is guaranteed because the algorithm always selects the shortest available path from the set of vertices that haven't been visited yet.

## CONT...

### **Worst-Case Analysis: $O(V^2 \log V)$**

- Dijkstra's algorithm on a dense graph ( $E \approx V^2$ ).
- High computational effort due to many edges.
- Slower in densely populated urban centers.
- Guarantees finding the shortest path despite longer computation times.

### **Average-Case Analysis: $O((V + E) \log V)$**

- Moderate density graphs, typical of suburban/semi-urban areas.
- Balanced mix of vertices and edges.
- Efficient and timely shortest-path calculations.
- Suitable for typical urban environments.

### **Best-Case Analysis: $O((V + E) \log V)$**

- Sparse graphs with fewer edges compared to vertices.
- Dominated by  $V \log V$  term, leading to efficient performance.
- Quick calculations in sparsely populated or less connected areas.
- Ensures rapid response times.



THANK  
YOU

