# Business Case: City Management

Sub-task: Managing Renewable Energy in City

SDG Goal: 11

Target: 11.3, 11.6, 11.7

Indicator: 11.3

## City Infrastructure Plan: Vatsalya Nagar

### 1. Renewable Energy Solutions

To establish efficient and sustainable renewable energy infrastructure in Vatsalya Nagar, we will analyze and optimize the energy network using algorithms like Dijkstra's, Kruskal's, and Prim's for resource allocation and critical connection optimization.

Below is the detailed plan for renewable energy stations required for Vatsalya Nagar:

| Station ID | Type | Distance Between Stations (km) | Capacity per Station (MW) |
|---|---|---|---|
| SP1 | Solar | 15 | 50 |
| SP2 | Wind | 20 | 60 |
| SP3 | Hybrid | 25 | 80 |

#### Algorithm Details:

##### *A. Dijkstra's Algorithm (Optimal Energy Path Calculation)*

- **Input**: Graph with nodes, edges, and weights (distance or capacity).
- **Output**: Shortest path from the source energy station to all other nodes.

**Algorithm Steps**:

1. Initialize distances to all nodes as infinity (∞) except the source node (set to 0).
2. Use a priority queue to select the node with the smallest distance.
3. For the current node, update the distances to its neighbors if a shorter path is found.
4. Repeat until all nodes are processed.

---

##### *1. Renewable Energy Management:*

- **Nodes**: Represent energy stations (e.g., A, B, C).
- **Edges**: Represent connections with weights for capacity or distance.
- **Paths**: Optimized using Dijkstra's algorithm.

### *2. Grid Network Design:*

- **Nodes**: Represent distribution hubs and storage facilities.
- **Edges**: Represent power lines with capacities.
- **Network**: Spanning tree derived from Kruskal's algorithm.

| Zone | Node ID | Connection | Distance (km) | Capacity (MW) |
|---|---|---|---|---|
| Solar Zone | A | B, C | 3, 5 | 50, 60 |
| Wind Zone | B | A, D | 3, 7 | 50, 70 |
| Hybrid Zone | C | A, D, E | 5, 4, 6 | 80, 60, 90 |
| Distribution Hub | D | B, C, F | 7, 4, 8 | 70, 60, 100 |
| Storage Facility | E | C, F | 6, 3 | 90, 110 |
| Backup Grid | F | D, E | 8, 3 | 100, 110 |

### B. Kruskal's Algorithm

Kruskal's Algorithm is primarily used for finding the Minimum Spanning Tree (MST) of a graph, which minimizes the total edge weights (costs) while ensuring all nodes are connected. In the context of renewable energy, this algorithm helps in efficiently connecting energy stations to minimize overall infrastructure costs.

An optimal network design for the city was developed using **Kruskal's Algorithm** to construct a **minimum spanning tree (MST)**. This approach minimizes the total cost of connecting all critical points in the city's infrastructure, such as transport hubs, residential zones, and commercial centers.

The **code implementation** is available **HERE**. The code leverages the **Union-Find data structure** (Disjoint Set Union) to efficiently manage edge selection and cycle detection.

The efficiency of the code is **O(E log E)**, where **E** represents the number of edges. Kruskal's Algorithm employs the **greedy technique**, sorting all edges by weight and progressively adding the smallest edge to the MST, provided it does not form a cycle.

This case demonstrates how **Kruskal's Algorithm** can be utilized in real-world applications such as designing cost-efficient utility networks, constructing green transportation systems, or optimizing resource allocation in urban planning.

### C. Prim's Algorithm

Prim's Algorithm is similar to Kruskal's but starts from a single node and grows the MST incrementally. In renewable energy systems, Prim's Algorithm is applied for optimizing connections within a local area renewable energy system, ensuring minimal costs while maintaining robust connectivity.

An optimal infrastructure plan for the city was designed using **Prim's Algorithm** to construct a minimum spanning tree (MST) of the city's road network. This approach

ensures that all key locations are connected with minimal total cost while maintaining efficiency.

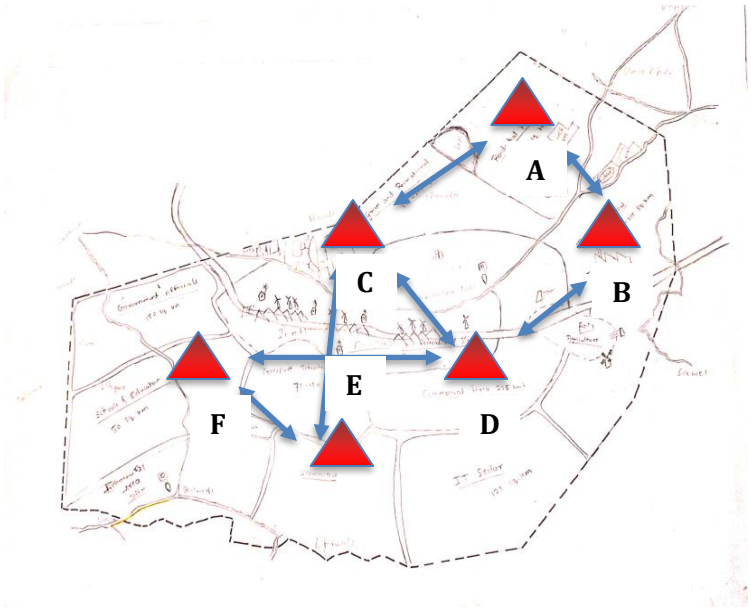
the **code implementation** can be accessed **HERE**. The code utilizes the **Heap data structure** for efficient edge selection and connection management.

The efficiency of the code is **O(E log V)**, where **E** represents the number of edges and **V** represents the number of nodes. The algorithm employs the **greedy technique**, iteratively selecting the smallest edge that connects a new node to the growing MST while avoiding cycles.

This case highlights how **Prim's Algorithm** can be applied in real-time scenarios, such as optimizing road networks, designing cost-efficient utility grids, or planning eco-friendly transportation systems for urban development.

### D. Dijkstra's Algorithm

Dijkstra's Algorithm finds the shortest path in a weighted graph. In renewable energy applications, it is utilized for optimizing energy delivery paths to minimize power losses, thereby enhancing the efficiency of energy distribution across the network.




An optimal tour is prepared for the city major using Dijkstras algorithm. The code can be viewed HERE. The code is implemented using Heap data Structure. The efficiency of the code is Ellog V and the algorithm uses the greedy technique. This case made me realize on how Dijkstras algorithm can be used in real time scenarios.

## 2. Smart Waste Management

To address waste management challenges in Vatsalya Nagar, we will optimize the waste collection, transportation, and processing network using advanced algorithms such as Dijkstra's Algorithm, Kruskal's Algorithm, Prim's Algorithm, BFS/DFS for critical traversal tasks, and Decision Trees for waste classification. This plan will ensure efficient waste disposal and sustainable resource management.

Below is the detailed plan for smart waste management stations required for Vatsalya Nagar:

| Station ID | Type | Distance Between Stations (km) | Capacity per Station (tons/day) |
|---|---|---|---|
| WM1 | Collection Point | 2 | 5 |
| WM2 | Sorting Hub | 5 | 20 |
| WM3 | Recycling Plant | 15 | 50 |

### Algorithm Details:

#### *1. Dijkstra's Algorithm (Shortest Path Calculation for Waste Collection Routes):*

- **Input:** Graph with nodes (waste collection stations), edges (roads), and weights (distance or congestion level).
- **Output:** Shortest path from a source node (e.g., residential area) to all other waste stations for efficient collection.
- **Steps:**
    1. Initialize distances to all waste stations as infinity (∞), except for the source node (set to 0).
    2. Use a priority queue to select the node with the smallest distance.
    3. For the selected node, update distances to its neighboring stations if a shorter path is found.
    4. Repeat the process until all waste stations are processed and the shortest paths are found.

An efficient waste transportation plan for the city's **Smart Waste Management Stations** was created using **Dijkstra's Algorithm**. This algorithm ensures the shortest and most cost-effective paths are used for waste collection and transfer between stations.

The **code implementation** is available HERE. The code is implemented using the **Heap data structure**, enabling efficient selection of nodes and edge relaxation.

The efficiency of the code is **O(E log V)**, where **E** represents the number of edges and **V** the number of nodes. The algorithm uses the **greedy technique**, iteratively selecting the shortest path to connect waste collection and processing points while updating paths to neighboring stations if a shorter route is found.

This case illustrates how **Dijkstra's Algorithm** can be applied in real-world scenarios, such as planning optimal routes for waste collection vehicles, reducing transportation time and costs, and ensuring sustainable operations in waste management systems.

### *2. Kruskal's Algorithm / Prim's Algorithm (Designing Efficient Waste Collection Networks):*

- **Purpose:** Design a minimum spanning tree for the waste transfer network to minimize transportation costs.
- **Kruskal's Algorithm:** Sort all the edges by weight (cost or distance), and keep adding edges to the network without creating cycles until all nodes are connected.
- **Prim's Algorithm:** Start from a random node and iteratively add the shortest edge to the existing network until all nodes are connected.

An optimal network for **Smart Waste Management Stations** was designed using **Kruskal's Algorithm** to construct a **minimum spanning tree (MST)**. This ensures all waste collection and processing points are connected with minimal transportation costs.

The **code implementation** can be accessed HERE. The code uses the **Union-Find data structure** (Disjoint Set Union) for efficient edge selection and cycle detection.

The efficiency of the code is **O(E log E)**, where **E** represents the number of edges. The algorithm uses the **greedy technique**, sorting all edges by weight (cost or distance) and progressively adding the smallest edge to the MST, provided it does not form a cycle.

This case highlights how **Kruskal's Algorithm** can be applied in real-world scenarios, such as optimizing waste collection routes, reducing transportation costs, and designing sustainable waste management systems.

### *3. BFS/DFS for Critical Traversal Tasks (Route Optimization and Critical Path Identification):*

- **BFS (Breadth-First Search):** Used for finding the shortest path when all edges have equal weight, suitable for route planning in waste collection.

The **code implementation** is available HERE. The algorithm processes nodes in a **queue structure**, ensuring that all stations at the same level (distance) are explored before moving deeper into the network.

The efficiency of the code is **O(V + E)**, where **V** represents the number of stations (nodes) and **E** the number of routes (edges). BFS guarantees that the shortest path in terms of the number of hops (connections) is determined for waste collection.

- **DFS (Depth-First Search):** Used to explore all possible routes and ensure there are no unnecessary cycles in the waste collection routes.

The **code implementation** is available HERE. The algorithm employs a **stack structure** (either explicitly or via recursion) to traverse routes as deeply as possible before backtracking.

The efficiency of the code is **O(V + E)**, where **V** represents the number of stations (nodes) and **E** the number of routes (edges). DFS is particularly effective for identifying disconnected components in the network or analyzing routes to ensure no processing station is isolated.

### *4. Decision Trees (Waste Classification Based on Material Type):*

- **Input:** Raw waste data, such as material type (plastic, glass, organic, etc.).
- **Output:** Waste classification based on predefined material categories.
- **Steps:**
    1. Gather data about waste material types (e.g., plastic, glass, paper).
    2. Use a decision tree model to classify waste based on specific features like weight, size, or material composition.
    3. Ensure efficient separation of waste for recycling or disposal.

The code implementation is available **HERE**. The decision tree operates by recursively splitting the dataset based on the most informative features, such as waste volume, distance between stations, or processing capacity, to arrive at effective routing and processing decisions.

The computational complexity of building the decision tree is O(n × m × log(n)), where nnn is the number of data points, and mmm is the number of features. Once constructed, predictions using the tree are efficient, with a complexity of O(log(n)).

---

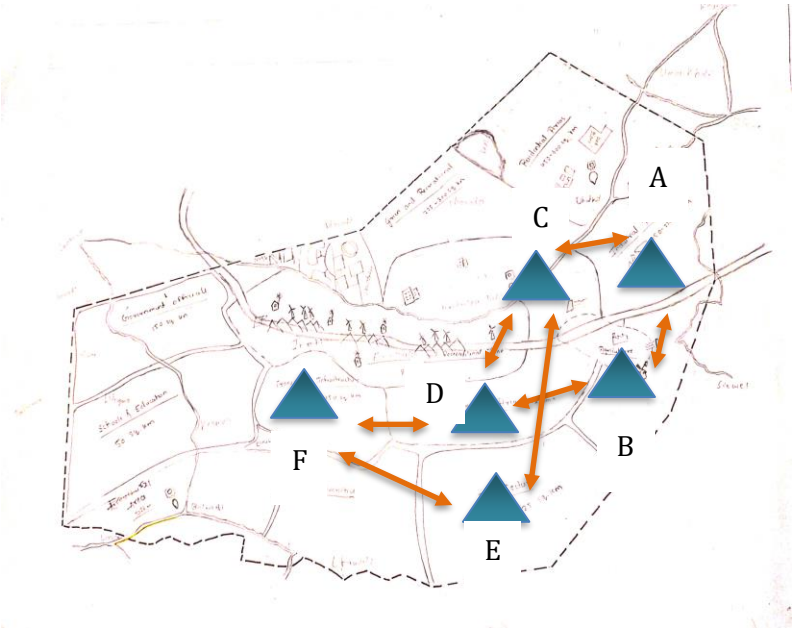
| Zone | Node ID | Connection | Distance (km) | Congestion Level (Weight) |
|---|---|---|---|---|
| Residential Zone | A | B, C | 3, 5 | 2,3 |
| Commercial Hub | B | A, D | 3, 7 | 2,5 |
| Industrial Area | C | A, D, E | 5, 4, 6 | 3,4,6 |
| Transport Hub | D | B, C, F | 7, 4, 8 | 5,4,7 |
| Recreational Park | E | C, F | 6, 3 | 6,2 |
| Smart Waste Station | F | D, E | 8, 3 | 7,2 |

### Waste Network Design:

- **Waste Transfer Network:** Nodes represent waste processing stations or hubs. The edges represent waste transfer routes with costs (e.g., transportation costs, distance). A minimum spanning tree will be derived using Kruskal's or Prim's Algorithm to optimize the transportation of waste.
- **Waste Classification System:** Decision Trees will be used to classify waste into categories like recyclable, organic, hazardous, and general waste based on material type for effective sorting and disposal.

### Example of Network Optimization using Kruskal's Algorithm:

- We would create a graph where nodes represent waste management stations and edges represent roads for waste transportation.
- By applying Kruskal's Algorithm, we will identify the minimum cost pathways to connect all stations, ensuring efficient waste transfer and minimizing transportation costs.




## 3. Green Urban Mobility

To address traffic management and environmental sustainability issues in Vatsalya Nagar, we will analyze and optimize the transportation network by incorporating Green Urban Mobility strategies. The plan will utilize algorithms like Dijkstra's Algorithm, Kruskal's Algorithm, BFS/DFS, Graph Matching Algorithms, and Optimization Algorithms to create an efficient and eco-friendly urban transport system. Additionally, our approach will focus on reducing $CO_2$ emissions and matching transport demand with available resources.

Below is the detailed plan for green urban mobility facilities required for Vatsalya Nagar:

| Facility ID | Type | Distance Between Stations (km) | Efficiency (Passengers/hour) |
|---|---|---|---|
| GM1 | Electric Bus Hub | 10 | 500 |
| GM2 | Bike Sharing | 3 | 50 |
| GM3 | Charging Points | 5 | 10 EVs/hour |

## Algorithm Details:

### *1. BFS/DFS for Critical Traversal Tasks (Optimizing Public Transit and Private Routes):*

- **BFS (Breadth-First Search):** Used for finding the shortest paths with equal edge weights, which can be helpful for routing public transport like buses or shared mobility services (e.g., e-scooters).

The the **code implementation** is available HERE. The algorithm processes nodes in a **queue structure**, ensuring that all stations at the same level (distance) are explored before moving deeper into the network.

The efficiency of the code is **O(V + E)**, where **V** represents the number of stations (nodes) and **E** the number of routes (edges). BFS guarantees that the shortest path in terms of the number of hops (connections) is determined for waste collection.

- **DFS (Depth-First Search):** Used for route exploration and ensuring no unnecessary cycles or inefficient routes in the transport system.

The **code implementation** is available HERE. The algorithm employs a **stack structure** (either explicitly or via recursion) to traverse routes as deeply as possible before backtracking.

The efficiency of the code is **O(V + E)**, where **V** represents the number of stations (nodes) and **E** the number of routes (edges). DFS is particularly effective for identifying disconnected components in the network or analyzing routes to ensure no processing station is isolated.

### *2. Graph Matching Algorithms (Matching Transport Demand with Resources):*

- **Input:** Data on transport demand (e.g., number of passengers, peak times) and available resources (e.g., buses, electric vehicles, bike-sharing).
- **Output:** Efficient allocation of available transport resources to meet the demand in a way that optimizes utilization and reduces waiting time.
- **Steps:**
    1. Represent transport demand and resources as a bipartite graph.
    2. Use matching algorithms (e.g., Hungarian Algorithm, Hopcroft-Karp Algorithm) to match available vehicles to areas with high demand for transport.
    3. Optimize the matching process to ensure that vehicles are used effectively, reducing congestion and improving service delivery.

A network optimization framework for the city's Green Urban Mobility Network was developed using Graph Matching Algorithms. These algorithms are pivotal for pairing

demand and supply nodes, such as matching ride requests with electric vehicle drivers or connecting bike-sharing stations.

The code implementation is available **HERE**. Graph Matching Algorithms aim to identify an optimal set of connections between nodes in a bipartite or general graph, maximizing efficiency while minimizing costs such as travel time, energy consumption, or distance.

### *3. Optimization Algorithms (Minimizing $CO_2$ Emissions in Public and Private Transport):*

- **Objective:** Minimize the overall $CO_2$ emissions from both private and public transport networks.
- **Approach:** Use optimization techniques like Linear Programming or Genetic Algorithms to:
    - Maximize the use of low-emission or electric vehicles.
    - Plan public transport schedules to reduce fuel consumption and $CO_2$ emissions.
    - Optimize routes and timing for ride-sharing services.
- **Steps:**
    1. Define emission cost functions based on vehicle types, distance, and travel patterns.
    2. Use optimization algorithms to find solutions that minimize $CO_2$ emissions while still meeting transportation needs.
    3. Focus on integrating green technologies like electric vehicles (EVs) and hybrid vehicles in the public and private sectors.

A comprehensive strategy for enhancing the efficiency of the city's Green Urban Mobility Network was developed using Optimization Algorithms. These algorithms focus on maximizing or minimizing key objectives, such as reducing energy consumption, travel time, or emissions, while ensuring balanced and sustainable operations.

The code implementation is available **HERE**. Optimization Algorithms operate by iteratively searching for the best solution to the defined problem within given constraints, such as vehicle capacity, station availability, or traffic conditions.

### Green Transport Network Design:

- **Traffic Management Network:** Nodes represent transportation hubs and residential/commercial areas. Edges represent green transport routes like electric bus lines, bike-sharing stations, and carpooling lanes. Dijkstra's Algorithm will be used to find the shortest eco-friendly routes between these zones, considering $CO_2$ emissions and road congestion.
- **Public Transport Network:** The goal is to create a low-emission, efficient public transport network using a combination of electric buses, trains, and shared mobility services. Kruskal's or Prim's Algorithm will be used to design cost-efficient routes and connections between the transport hubs, ensuring optimal use of resources.

- **Private Transport Optimization:** Graph Matching Algorithms will be used to match passenger demand with available green vehicles (e.g., electric cars, shared bikes, etc.). This approach ensures resources are allocated efficiently to minimize waiting time and reduce overall emissions.
- **$CO_2$ Optimization:** Optimization Algorithms will minimize $CO_2$ emissions by promoting electric vehicle use, optimizing transport schedules, and maximizing shared rides or public transport trips to reduce the number of vehicles on the road.

---

### Example of Green Urban Mobility Optimization:

1. **Dijkstra's Algorithm:** Used to find the shortest eco-friendly routes between zones, considering both distance and congestion, allowing electric vehicles (EVs) to avoid congested areas for faster travel.
2. **Kruskal's/Prim's Algorithm:** Will create an optimal network for green buses and electric vehicles, minimizing the total cost of transport and ensuring all major hubs are connected in an environmentally friendly way.
3. **Graph Matching Algorithm:** Efficiently matches the demand for electric taxis or ride-sharing with available vehicles, ensuring no resources are wasted and waiting times are minimized.
4. **Optimization Algorithms:** These algorithms will minimize $CO_2$ emissions by optimizing the use of green transport resources, including optimizing routes for electric buses and planning schedules to maximize the utilization of low-emission vehicles.

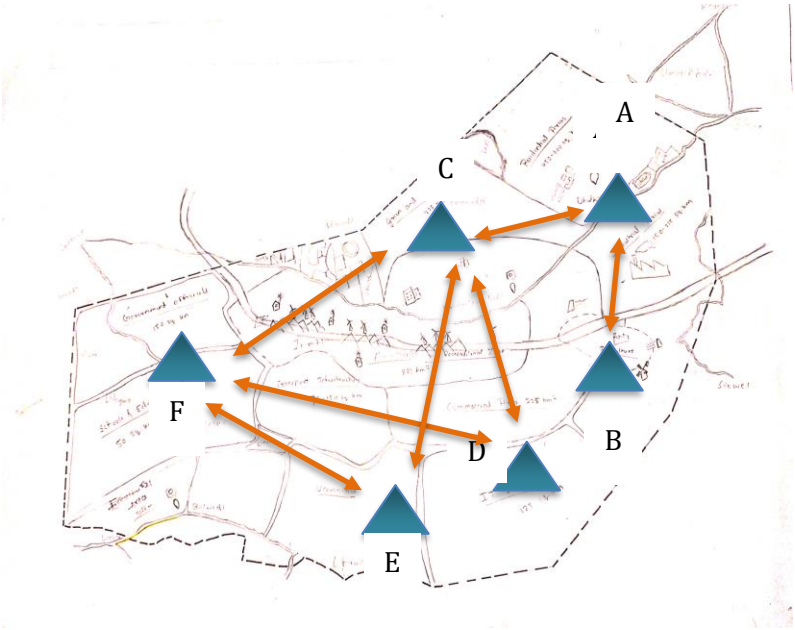
### Metro Network Design for Green Mobility:

- **Nodes:** Major transport hubs, such as metro stations, electric bus depots, and shared mobility centers.
- **Edges:** Connections representing public transportation lines, such as metro rail lines, electric bus routes, and bike lanes.
- **Spanning Tree:** A minimum spanning tree derived from Kruskal's or Prim's Algorithm will help optimize the transport network, ensuring cost-efficiency and reducing emissions.

By using these algorithms, we will be able to build an efficient, sustainable, and green transport network for Vatsalya Nagar, improving mobility while minimizing environmental impact.

Below is the detailed plan for green urban mobility facilities required for Vatsalya Nagar:

| Zone | Node ID | Connection | Distance (km) | Congestion Level (Weight) |
|---|---|---|---|---|
| Residential Zone | A | B, C | 3, 5 | 2,3 |
| Commercial Hub | B | A, D | 3, 7 | 2,5 |
| Industrial Area | C | A, D, E | 5, 4, 6 | 3,4,6 |
| Transport Hub | D | B, C, F | 7, 4, 8 | 5,4,7 |
| Recreational Park | E | C, F | 6, 3 | 6,2 |
| Smart Waste Station | F | D, E | 8, 3 | 7,2 |


A
C
F
D
B
E