**Design Document**

**1. Overview**

The project seeks to identify optimal locations for dam construction along a river that passes through multiple states and junctions to prevent flooding. The river system is modelled as a directed graph where nodes represent junctions or locations, edges represent river flows, and weights indicate the volume of water. The problem requires developing an efficient algorithm based on Breadth-First Search (BFS) to perform the identification.

**2. Objectives**

• Identify junctions where multiple branches split to control downstream flow.

• Select locations with high flow capacities for significant water storage potential.

• Ensure that the outgoing water flow at identified nodes meets the specified threshold.

**3. Graph Representation**

• Nodes (Vertices): Junctions or locations along the river.

• Edges (Arrows): River flows, directed downstream.

• Weights (Values): Volume of water flowing along the edge.

**4. Criteria for Dam Construction**

* Multiple Branches Split: Construct dams at nodes where the river splits into multiple branches to control the flow.
* High Flow Capacities: Prioritize nodes with high flow capacities for water storage.

**5. Algorithm Design**

1. Initialize Variables:

* Create a new list to maintain the nodes visited.
* Create an empty list dam\_locations to store potential locations of dams.
* Create an empty list bfs\_order to store the order in which BFS is done.

2. Traverse Each Node:

For each node in the graph:

* If the node is not visited:
  + Create a queue and add the node.
  + Mark the node as visited.

3. BFS:

While the queue is not empty:

* Remove a node (current\_node).
* Retrieve the edges of the current\_node.
* Calculate the total outgoing flow from the current\_node.
* If the current\_node has multiple branches and the outgoing flow exceeds a given threshold:
  + Add the current\_node and its flow to dam\_locations.
* Remove the unvisited edges and mark them as visited.

4. Return Results:

Return the list of dam locations and the BFS traversal order.

**5.1.1. Design Justification**

**Breadth-First Search (BFS):**

BFS is selected based on its ability to perform a level-order traversal so that all nodes at the present level are traversed before any at the next level. This allows for systematic determination of the number of junctions with multiple branches.

**Queue Data Structure:**

A queue is used to control nodes during BFS traversal. Nodes are put into the queue when they are found and taken out when they are visited, ensuring the order of BFS.

**Flow Threshold:**

A flow threshold is applied to filter nodes with high water storage capacity. Only junctions with flows going out of them greater than the threshold are considered for the construction of dams.

**5.1.2. Time Complexity Analysis**

Initialization:

The time complexity of initializing the visited list and dam locations list is O(V)O(V), where VV is the number of vertices.

Node Traversal:

The time complexity of traversing each node is O(V)O(V) since every node is visited once.

BFS Traversal:

The time complexity of processing each node and its edges during BFS is O(V+E) O (V + E), where VV is the number of vertices and EE is the number of edges.

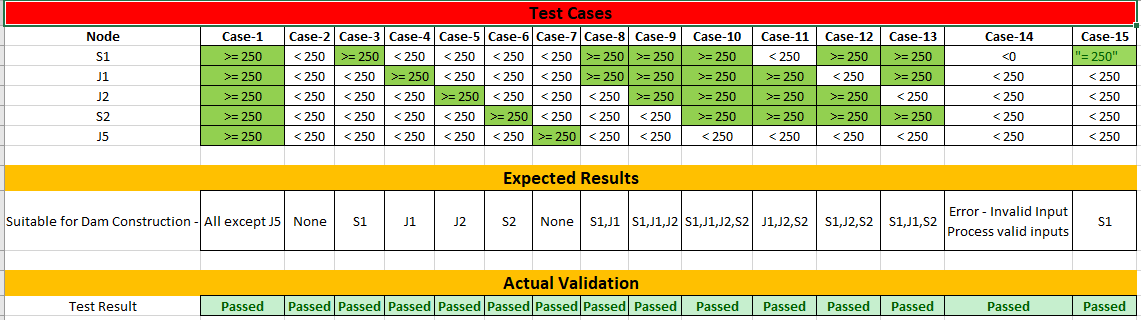
Total Time Complexity:

The algorithm has a time complexity of O(V+E) O (V + E) hence it is quite efficient in locating potential dam locations in the river system.

|  |  |  |
| --- | --- | --- |
| **Operation** | **Description** | **Time Complexity** |
| Graph Initialization | Initialize graph and adjacency list | O(V) |
| Add Edge | Add an edge to the graph | O(1)O(1) per edge |
| Read Graph | Read edges from file, build node mapping, and add edges | O(E+V) |
| BFS Traversal | Perform BFS, calculate outgoing flow, and identify dam locations | O(V+E) |
| Write Output | Write BFS traversal and dam locations to output file | O(V+E) |
| **Overall Time Complexity** | Combined time complexity of all steps | **O(V+E)** |

**5.1.3. Implementation and Testing**

Python code written to address this problem statement and thoroughly tested using Jupiter notebook. Below are the test cases validated against this code.



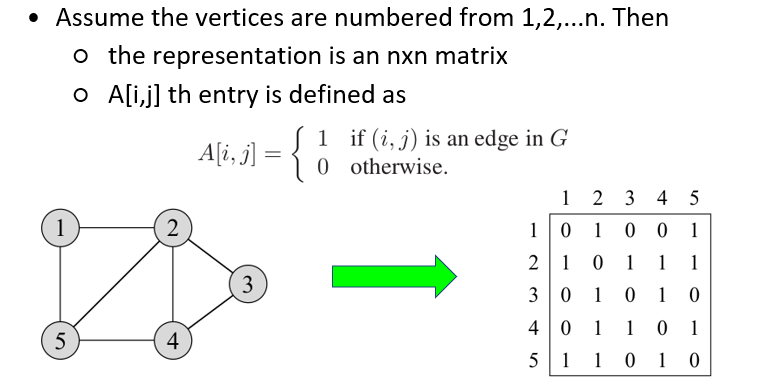
**5.1.4. Summary**

The algorithm proposed identifies possible dam locations in a river system represented as a directed graph using BFS. The choices made during the design phase, such as BFS traversal, queue management, and flow threshold filtering, would result in potential junctions to be eventually selected for a dam. For this reason, the time complexity of O(V+E) O (V + E) of the algorithm provides evidence for efficiency and suitability for large river systems with several junctions and flows.

**6. Alternate way of modelling the problem with cost implication/ Other approaches**

This problem of identifying potential dam construction sites can be solved using either an **adjacency matrix** representation of the graph.

An adjacency matrix is a 2D array where each cell at position (i, j) indicates the presence of an edge between vertex i and vertex j. If there is no edge, the cell can contain a zero or infinity, depending on whether weights are used.



**Cost implication & other comparison –**

* **Adjacency Matrix**:
  + **Space Complexity**: O(V^2)
  + **Time Complexity**: Edge lookups are O(1)*O*(1), but iterating through edges can be O(V^2).
  + **Operational Costs**: Higher for sparse graphs due to wasted space and inefficient traversal.
* **BFS Approach/ Adjacency list**:
  + **Space Complexity**: O(V+E)
  + **Time Complexity**: O(V+E)
  + **Operational Costs**: Generally lower due to efficient traversal and direct identification of potential dam sites.

Overall, while both methods can be used to solve the problem, BFS is typically more efficient and cost-effective in scenarios involving sparse graphs commonly encountered in river systems. The adjacency matrix may be more suitable for dense graphs but comes with higher costs in terms of memory and potentially slower traversal times.