Computing Lab Assignments

Practice Problems

ISI, Kolkata

October 7, 2025

Outline

Outline

Detecting Cycles in an Undirected Graph Using DSU

- **Goal:** Determine if a simple undirected graph contains a cycle.
- **Input:** A graph with V vertices and E edges.
- **Output:** `True` if the graph contains a cycle; otherwise, `False`.
- **Assumption:** The graph does not contain self-loops.

Example: The graph with vertices $\{0, 1, 2\}$ and edges $\{(0, 1), (1, 2), (2, 0)\}$ contains a cycle.

Cycle Detection Algorithm Using DSU

- **Initialize:** Create a parent array where each vertex is its own parent.
- **Process Edges:** For each edge (u, v):
  - `Find(u)` and `Find(v)`:
    - If `Find(u) == Find(v)`, a cycle is detected.
    - Otherwise, perform `Union(u, v)` to merge the sets.
- **Union Operation:** Connect two subsets into a single subset.
- **Find Operation:** Determine the representative (root) of the set containing an element.

Note: This approach assumes the graph does not contain self-loops.

Outline

Problem Outline

- Given a **connected, undirected, weighted graph** $G(V, E)$
- Find a **spanning tree** that connects all vertices with the **minimum total edge weight**
- **Input:** list of edges with weights
- **Output:** set of edges forming the Minimum Spanning Tree (MST)
- **Goal:** minimize total cost while keeping graph connected and acyclic

Example: Network design (roads, cables, or pipelines) with minimal cost.

Solution Outline — Kruskal's Algorithm

Data Structure: Disjoint Set

1. Sort all edges in **non-decreasing order** of their weight.
2. Initialize each vertex as a separate **set**
3. For each edge in sorted order:
   - If the edge **does not form a cycle**, include it in the MST.
   - Otherwise, skip it.
4. Stop when $V - 1$ edges are included.

Complexity: $O(E \log E)$ due to sorting.

Outline

Problem: Earliest Moment When Everyone Become Friends

- **Goal:** Determine the earliest time at which all individuals in a group become friends.
- **Input:**
  - N: Number of people (labeled from 0 to $N-1$).
  - M: Number of friendship events.
  - arr[M][3]: Each event is represented as {U, V, time}, indicating that person U and person V become friends at the given time.
- **Output:** The earliest time when everyone is friends, or -1 if it's not possible.

Example:

- Input: $N = 4$, arr $= \{\{0, 1, 2\}, \{1, 2, 3\}, \{2, 3, 4\}\}$
- Output: 4

Solution Using Disjoint Set Union (DSU)

- **Approach:** Utilize DSU to efficiently manage the merging of groups.
- **Steps:**
  1. Initialize a DSU structure for N people.
  2. Sort the events in ascending order of time.
  3. Process each event:
     - If the two people are already in the same group, skip the event.
     - Otherwise, merge their groups.
     - If all people are in the same group, return the current time.
  4. If all events are processed and not all are connected, return -1.
- **Time Complexity:** $O(MlgM)$.

Note: Sorting the events ensures that we process them in chronological order.

Outline

Problem: Minimum Changes to Make Two Arrays Identical

- **Goal:** Make two arrays A and B identical by performing the fewest operations.
- **Operation:** Choose two integers x and y, and replace all occurrences of x in both arrays with y.
- **Input:** Two arrays A and B of size N.
- **Output:** Minimum number of operations required to make A and B identical.

Example:

- Input: $A = [2, 1, 1, 3, 5]$, $B = [1, 2, 2, 4, 5]$
- Output: 2 operations

Solution Using Disjoint Set Union (DSU)

- **Approach:** Use DSU to group elements that can be made identical.
- **Steps:**
  1. Initialize a DSU structure for each element.
  2. For each index i, if $A[i] \neq B[i]$, perform a union operation on $A[i]$ and $B[i]$.
  3. After processing all indices, count the number of distinct sets formed.
  4. The minimum number of operations is the sum of the (size-1) of all distinct sets.

Note: This approach ensures that we minimize the number of operations by efficiently grouping elements.

Outline

Problem: Minimum Cost to Provide Water

- **Goal:** Ensure every village has water at minimum cost.
- **Input:**
  - N: Number of villages (1 to N).
  - wells[i]: Cost to build a well in village i.
  - pipes[X, Y, C]: Cost (C) to connect village X to village Y with a pipe.
- **Output:** Minimum cost to provide water to all villages.

Example:

- Input: $N = 3$, wells $= [1, 2, 2]$, pipes $= [[1, 2, 1], [2, 3, 1]]$
- Output: 3

Solution Using Minimum Spanning Tree (MST)

- **Approach:** Model the problem as a graph and find the MST.
- **Steps:**
  1. Create a graph with villages as vertices.
  2. Add edges from the `pipes` array between villages.
  3. Add a pseudo vertex (0) connected to each village with edges weighted by `wells[i]`.
  4. Apply Kruskal's or Prim's algorithm to find the MST.
- **Time Complexity:** $O(E \log E)$, where E is the number of edges.

Example Walkthrough

- **Input:** $N = 4$, wells $= [1, 1, 1, 1]$,
  pipes $= [[1, 2, 100], [2, 3, 100], [2, 4, 50]]$
- **Graph Construction:**
  - Vertices: 0, 1, 2, 3, 4
  - Edges: {(1, 2, 100), (2, 3, 100), (2, 4, 50)}
  - Add pseudo vertex 0 with edges: {(0, 1, 1), (0, 2, 1), (0, 3, 1), (0, 4, 1)}
- **MST Calculation:** Apply Kruskal's algorithm to find the MST.
- **Output:** 4

Outline

Sources

https://leetcode.com/problems/number-of-operations-to-make-network-connected/

https://www.geeksforgeeks.org/dsa/disjoint-set-data-structures/

https://www.spoj.com/problems/CLFLARR/