1168. Optimize Water Distribution in a Village

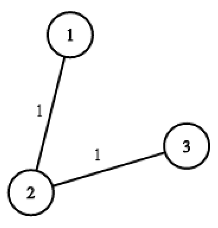
Hard 666 26 Add to List Share

There are `n` houses in a village. We want to supply water for all the houses by building wells and laying pipes.

For each house `i`, we can either build a well inside it directly with cost `wells[i - 1]` (note the `-1` due to **0-indexing**), or pipe in water from another well to it. The costs to lay pipes between houses are given by the array `pipes`, where each `pipes[j] = [house1_j, house2_j, cost_j]` represents the cost to connect `house1_j` and `house2_j` together using a pipe. Connections are bidirectional.

Return *the minimum total cost to supply water to all houses.*

Example 1:




```
Input: n = 3, wells = [1,2,2], pipes = [[1,2,1],[2,3,1]]
Output: 3
Explanation:
The image shows the costs of connecting houses using pipes.
The best strategy is to build a well in the first house with cost 1 and connect
the other houses to it with cost 2 so the total cost is 3.
```

Example 2:

```
Input: n = 2, wells = [1,1], pipes = [[1,2,1]]
Output: 2
```

Constraints:

- $2 <= n <= 10^4$
- `wells.length == n`
- $0 <= wells[i] <= 10^5$
- $1 <= pipes.length <= 10^4$
- `pipes[j].length == 3`
- $1 <= house1_j, house2_j <= n$
- $0 <= cost_j <= 10^5$
- $house1_j != house2_j$

Accepted 22,620 | Submissions 36,055

Seen this question in a real interview before? Yes No

Companies

0 ~ 6 months | 6 months ~ 1 year | 1 year ~ 2 years

Facebook | 2

Related Topics

Union Find | Graph | Minimum Spanning Tree

Hide Hint 1

What if we model this problem as a graph problem?

Hide Hint 2

A house is a node and a pipe is a weighted edge.

Hide Hint 3

How to represent building wells in the graph model?

Hide Hint 4

Add a virtual node, connect it to houses with edges weighted by the costs to build wells in these houses.

Hide Hint 5

The problem is now reduced to a Minimum Spanning Tree problem.

Java | Autocomplete

```java
class Solution {
    public int minCostToSupplyWater(int n, int[] wells, int[][] pipes) {
        // min heap to maintain the order of edges to be visited.
        PriorityQueue<Pair<Integer, Integer>> edgesHeap =
                new PriorityQueue<>(n, (a, b) -> (a.getKey() - b.getKey()));

        // representation of graph in adjacency list
        List<List<Pair<Integer, Integer>>> graph = new ArrayList<>(n + 1);
        for (int i = 0; i < n + 1; ++i) {
            graph.add(new ArrayList<Pair<Integer, Integer>>());
        }

        // add a virtual vertex indexed with 0,
        //   then add an edge to each of the house weighted by the cost
        for (int i = 0; i < wells.length; ++i) {
            Pair<Integer, Integer> virtualEdge = new Pair<>(wells[i], i + 1);
            graph.get(0).add(virtualEdge);
            // initialize the heap with the edges from the virtual vertex.
            edgesHeap.add(virtualEdge);
        }

        // add the bidirectional edges to the graph
        for (int i = 0; i < pipes.length; ++i) {
            int house1 = pipes[i][0];
            int house2 = pipes[i][1];
            int cost = pipes[i][2];
            graph.get(house1).add(new Pair<Integer, Integer>(cost, house2));
            graph.get(house2).add(new Pair<Integer, Integer>(cost, house1));
        }

        // kick off the exploration from the virtual vertex 0
        Set<Integer> mstSet = new HashSet<>();
        mstSet.add(0);

        int totalCost = 0;
        while (mstSet.size() < n + 1) {
            Pair<Integer, Integer> edge = edgesHeap.poll();
            int cost = edge.getKey();
            int nextHouse = edge.getValue();
            if (mstSet.contains(nextHouse)) {
                continue;
            }

            // adding the new vertex into the set
            mstSet.add(nextHouse);
            totalCost += cost;

            // expanding the candidates of edge to choose from in the next round
            for (Pair<Integer, Integer> neighborEdge : graph.get(nextHouse)) {
                if (!mstSet.contains(neighborEdge.getValue())) {
                    edgesHeap.add(neighborEdge);
                }
            }
        }

        return totalCost;
    }
}
```