2307. Check for Contradictions in Equations

Leetcode Link

Problem Description

You are given a 2D array of strings `equations` and an array of real numbers `values`, where `equations[i] = [Ai, Bi]` and `values[i]` means that `Ai / Bi = values[i]`. Your task is to determine if there exists a contradiction in the equations. Return `true` if there is a contradiction, or `false` otherwise.

Example

Consider the following input:

- equations = [["a", "b"], ["b", "c"]]
- values = [2.0, 3.0]

For this input, there is no contradiction and the output should be `false`.

Approach

We can use the concept of a directed weighted graph to represent equations that reflect their divisions. The graph vertices are the unique element strings in the input `equations` array, and the edges are the given fractions or their reciprocals (reverse fractions). As we traverse the graph, we need to track the accumulated products of the weights.

We can use a depth-first search (DFS) algorithm to traverse the graph, keeping track of the visited nodes and their corresponding accumulated products. A contradiction will occur if we visit a previously visited node and the ratio of the current accumulated value to the previously recorded value is significantly different.

The input string values are converted to integer indices to improve performance.

Algorithm

1. Create a hash map to map strings to integer indices.
2. Build a directed weighted graph from the input `equations` array.
3. Initialize a `seen` array to track the accumulated value of visited nodes.
4. Iterate through the graph and perform a DFS from each node.
5. If there is a contradiction during DFS traversal, return `true`, else return `false`.

DFS Traversal Example

Input:

- equations = [["a", "b"], ["b", "c"]]
- values = [2.0, 3.0]

Step by step process:

1. Convert string to int:

   strToInt = {"a": 0, "b": 1, "c": 2}

2. Build directed weighted graph:

   graph = [[(1, 2.0)], # ("a", "b") with value 2.0 [(0, 0.5), (2, 3.0)], # ("b", "a") with value 0.5, ("b", "c") with value 3.0 [(1, 1/3)] # ("c", "b") with value 1/3]

3. Initialize seen array:

   seen = [0, 0, 0]

4. Iterate through graph and perform DFS:
   - For node 0, DFS traversal: `0 -> 1 -> 2`
     - Update `seen` array: `[1, 2, 6]`
   - All nodes are visited, no contradiction found.
5. Return `false` as no contradiction found.

C++ Solution

```cpp
cpp
#include <cmath>
#include <unordered_map>
#include <vector>
using namespace std;

class Solution {
 public:
  bool checkContradictions(vector<vector<string>>& equations,
                           vector<double>& values) {
    // Convert string to int for better performance
    unordered_map<string, int> strToInt;

    for (const vector<string>& equation : equations) {
      const string& u = equation[0];
      const string& v = equation[1];
      if (!strToInt.count(u))
        strToInt[u] = strToInt.size();
      if (!strToInt.count(v))
        strToInt[v] = strToInt.size();
    }

    vector<vector<pair<int, double>>> graph(strToInt.size());
    vector<double> seen(graph.size());

    for (int i = 0; i < equations.size(); ++i) {
      const int u = strToInt.at(equations[i][0]);
      const int v = strToInt.at(equations[i][1]);
      graph[u].emplace_back(v, values[i]);
      graph[v].emplace_back(u, 1 / values[i]);
    }

    for (int i = 0; i < graph.size(); ++i)
      if (!seen[i] && dfs(graph, i, seen, 1.0))
        return true;

    return false;
  }

 private:
  bool dfs(const vector<vector<pair<int, double>>>& graph, int u,
           vector<double>& seen, double val) {
    if (seen[u])
      return abs(val / seen[u] - 1) > 1e-5;

    seen[u] = val;
    for (const auto& [v, w] : graph[u])
      if (dfs(graph, v, seen, val / w))
        return true;

    return false;
  }
};
```

Note: Make sure to include the necessary header files when using the C++ solution.## Python Solution

```python
python
from typing import List, Tuple

class Solution:
    def checkContradictions(self, equations: List[List[str]], values: List[float]) -> bool:
        # Convert string to int for better performance
        str_to_int = {}

        for equation in equations:
            u, v = equation
            if u not in str_to_int:
                str_to_int[u] = len(str_to_int)
            if v not in str_to_int:
                str_to_int[v] = len(str_to_int)

        graph = [[] for _ in range(len(str_to_int))]
        seen = [0] * len(graph)

        for i, (u, v) in enumerate(equations):
            u_idx = str_to_int[u]
            v_idx = str_to_int[v]
            graph[u_idx].append((v_idx, values[i]))
            graph[v_idx].append((u_idx, 1 / values[i]))

        def dfs(u: int, val: float) -> bool:
            nonlocal graph, seen

            if seen[u]:
                return abs(val / seen[u] - 1) > 1e-5

            seen[u] = val
            for v, w in graph[u]:
                if dfs(v, val / w):
                    return True

            return False

        for i in range(len(graph)):
            if not seen[i] and dfs(i, 1.0):
                return True

        return False
```

JavaScript Solution

```
javascript
class Solution {
    checkContradictions(equations, values) {
        // Convert string to int for better performance
        const strToInt = new Map();

        for (let equation of equations) {
            const [u, v] = equation;
            if (!strToInt.has(u))
                strToInt.set(u, strToInt.size);
            if (!strToInt.has(v))
                strToInt.set(v, strToInt.size);
        }

        const graph = new Array(strToInt.size).fill(null).map(() => []);
        const seen = new Array(graph.length).fill(0);

        for (let i = 0; i < equations.length; ++i) {
            const [u, v] = equations[i];
            const uIdx = strToInt.get(u);
            const vIdx = strToInt.get(v);
            graph[uIdx].push([vIdx, values[i]]);
            graph[vIdx].push([uIdx, 1 / values[i]]);
        }

        const dfs = (graph, u, seen, val) => {
            if (seen[u])
                return Math.abs(val / seen[u] - 1) > 1e-5;

            seen[u] = val;
            for (let [v, w] of graph[u]) {
                if (dfs(graph, v, seen, val / w))
                    return true;
            }

            return false;
        }

        for (let i = 0; i < graph.length; ++i) {
            if (!seen[i] && dfs(graph, i, seen, 1.0))
                return true;
        }

        return false;
    }
}
```

Java Solution

```
java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

class Solution {
    public boolean checkContradictions(List<List<String>> equations, double[] values) {
        // Convert string to int for better performance
        Map<String, Integer> strToInt = new HashMap<>();

        for (List<String> equation : equations) {
            String u = equation.get(0);
            String v = equation.get(1);
            strToInt.putIfAbsent(u, strToInt.size());
            strToInt.putIfAbsent(v, strToInt.size());
        }

        List<List<Pair<Integer, Double>>> graph = new ArrayList<>(strToInt.size());
        for (int i = 0; i < strToInt.size(); ++i)
            graph.add(new ArrayList<>());
        double[] seen = new double[graph.size()];

        for (int i = 0; i < equations.size(); ++i) {
            List<String> equation = equations.get(i);
            String u = equation.get(0);
            String v = equation.get(1);
            int uIdx = strToInt.get(u);
            int vIdx = strToInt.get(v);
            graph.get(uIdx).add(new Pair<>(vIdx, values[i]));
            graph.get(vIdx).add(new Pair<>(uIdx, 1 / values[i]));
        }

        for (int i = 0; i < graph.size(); ++i) {
            if (seen[i] == 0 && dfs(graph, i, seen, 1.0))
                return true;
        }

        return false;
    }

    private boolean dfs(List<List<Pair<Integer, Double>>> graph, int u, double[] seen, double val) {
        if (seen[u] != 0)
            return Math.abs(val / seen[u] - 1) > 1e-5;

        seen[u] = val;
        for (Pair<Integer, Double> entry : graph.get(u)) {
            int v = entry.getKey();
            double w = entry.getValue();
            if (dfs(graph, v, seen, val / w))
                return true;
        }

        return false;
    }

    static class Pair<K, V> {
        private K key;
        private V value;

        public Pair(K key, V value) {
            this.key = key;
            this.value = value;
        }

        public K getKey() {
            return key;
        }

        public V getValue() {
            return value;
        }
    }
}
```


Got a question? Ask the Teaching Assistant anything you don't understand.