Evaluate Division

Equations are given in the format `A / B = k`, where `A` and `B` are variables represented as strings, and `k` is a real number (floating point number). Given some queries, return the answers. If the answer does not exist, return `-1.0`.

Example:
Given `a / b = 2.0, b / c = 3.0.`
queries are: `a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ? .`
return `[6.0, 0.5, -1.0, 1.0, -1.0 ].`

The input is: `vector<pair<string, string>> equations, vector<double>& values, vector<pair<string, string>> queries` , where `equations.size() == values.size()`, and the values are positive. This represents the equations. Return `vector<double>`.

According to the example above:

```
equations = [ ["a", "b"], ["b", "c"] ],
values = [2.0, 3.0],
queries = [ ["a", "c"], ["b", "a"], ["a", "e"], ["a", "a"], ["x", "x"] ].
```

The input is always valid. You may assume that evaluating the queries will result in no division by zero and there is no contradiction.

Subscribe to see which companies asked this question

Solution 1

Image a/b = k as a link between node a and c, the weight from a to c is k, the reverse link is 1/k. Query is to find a path between two node.

```
    public double[] calcEquation(String[][] equations, double[] values, String[][
] queries) {
        HashMap<String, ArrayList<String>> pairs = new HashMap<String, ArrayList<
String>>();
        HashMap<String, ArrayList<Double>> valuesPair = new HashMap<String, Array
List<Double>>();
        for (int i = 0; i < equations.length; i++) {
            String[] equation = equations[i];
            if (!pairs.containsKey(equation[0])) {
                pairs.put(equation[0], new ArrayList<String>());
                valuesPair.put(equation[0], new ArrayList<Double>());
            }
            if (!pairs.containsKey(equation[1])) {
                pairs.put(equation[1], new ArrayList<String>());
                valuesPair.put(equation[1], new ArrayList<Double>());
            }
            pairs.get(equation[0]).add(equation[1]);
            pairs.get(equation[1]).add(equation[0]);
            valuesPair.get(equation[0]).add(values[i]);
            valuesPair.get(equation[1]).add(1/values[i]);
        }

        double[] result = new double[queries.length];
        for (int i = 0; i < queries.length; i++) {
            String[] query = queries[i];
            result[i] = dfs(query[0], query[1], pairs, valuesPair, new HashSet<St
ring>(), 1.0);
            if (result[i] == 0.0) result[i] = -1.0;
        }
        return result;
    }

    private double dfs(String start, String end, HashMap<String, ArrayList<String
>> pairs, HashMap<String, ArrayList<Double>> values, HashSet<String> set, double
value) {
        if (set.contains(start)) return 0.0;
        if (!pairs.containsKey(start)) return 0.0;
        if (start.equals(end)) return value;
        set.add(start);

        ArrayList<String> strList = pairs.get(start);
        ArrayList<Double> valueList = values.get(start);
        double tmp = 0.0;
        for (int i = 0; i < strList.size(); i++) {
            tmp = dfs(strList.get(i), end, pairs, values, set, value*valueList.ge
t(i));
            if (tmp != 0.0) {
                break;
            }
        }
        set.remove(start);
        return tmp;
    }
```

written by helloc93 original link here

Solution 2

A variation of **Floyd–Warshall**, computing quotients instead of shortest paths. An equation `A/B=k` is like a graph edge `A->B`, and `(A/B)*(B/C)*(C/D)` is like the path `A->B->C->D`. Submitted once, accepted in 35 ms.

```
def calcEquation(self, equations, values, queries):
    quot = collections.defaultdict(dict)
    for (num, den), val in zip(equations, values):
        quot[num][num] = quot[den][den] = 1.0
        quot[num][den] = val
        quot[den][num] = 1 / val
    for k, i, j in itertools.permutations(quot, 3):
        if k in quot[i] and j in quot[k]:
            quot[i][j] = quot[i][k] * quot[k][j]
    return [quot[num].get(den, -1.0) for num, den in queries]
```

Variation without the `if` (submitted twice, accepted in 68 and 39 ms):

```
def calcEquation(self, equations, values, queries):
    quot = collections.defaultdict(dict)
    for (num, den), val in zip(equations, values):
        quot[num][num] = quot[den][den] = 1.0
        quot[num][den] = val
        quot[den][num] = 1 / val
    for k in quot:
        for i in quot[k]:
            for j in quot[k]:
                quot[i][j] = quot[i][k] * quot[k][j]
    return [quot[num].get(den, -1.0) for num, den in queries]
```

Could save a line with `for i, j in itertools.permutations(quot[k], 2)` but it's longer and I don't like it as much here.

written by StefanPochmann original link here

Solution 3

```
class Solution
{
    // date: 2016-09-12     location: Santa Clara City Library
public:
    vector<double> calcEquation(vector<pair<string, string>> equations, vector<do
uble>& values, vector<pair<string, string>> queries)
    {
        unordered_map<string, Node*> map;
        vector<double> res;
        for (int i = 0; i < equations.size(); i ++)
        {
            string s1 = equations[i].first, s2 = equations[i].second;
            if (map.count(s1) == 0 && map.count(s2) == 0)
            {
                map[s1] = new Node();
                map[s2] = new Node();
                map[s1] -> value = values[i];
                map[s2] -> value = 1;
                map[s1] -> parent = map[s2];
            }
            else if (map.count(s1) == 0)
            {
                map[s1] = new Node();
                map[s1] -> value = map[s2] -> value * values[i];
                map[s1] -> parent = map[s2];
            }
            else if (map.count(s2) == 0)
            {
                map[s2] = new Node();
                map[s2] -> value = map[s1] -> value / values[i];
                map[s2] -> parent = map[s1];
            }
            else
                unionNodes(map[s1], map[s2], values[i], map);
        }

        for (auto query : queries)
        {
            if (map.count(query.first) == 0 || map.count(query.second) == 0 || fi
ndParent(map[query.first]) != findParent(map[query.second]))
                res.push_back(-1);
            else
                res.push_back(map[query.first] -> value / map[query.second] -> va
lue);
        }
        return res;
    }

private:
    struct Node
    {
        Node* parent;
        double value = 0.0;
        Node()  {parent = this;}
    };

    void unionNodes(Node* node1, Node* node2, double num, unordered_map<string, N
ode*>& map)
    {
        Node* parent1 = findParent(node1), *parent2 = findParent(node2);
        double ratio = node2 -> value * num / node1 -> value;
        for (auto it = map.begin(); it != map.end(); it ++)
            if (findParent(it -> second) == parent1)
                it -> second -> value *= ratio;
        parent1 -> parent = parent2;
    }

    Node* findParent(Node* node)
    {
        if (node -> parent == node)
            return node;
        node -> parent = findParent(node -> parent);
        return node -> parent;
    }
};
```

written by Mad_air original link here