990. Satisfiability of Equality Equations

Medium 758 6 Add to List Share

Given an array `equations` of strings that represent relationships between variables, each string `equations[i]` has length `4` and takes one of two different forms: `"a==b"` or `"a!=b"`. Here, `a` and `b` are lowercase letters (not necessarily different) that represent one-letter variable names.

Return `true` if and only if it is possible to assign integers to variable names so as to satisfy all the given equations.

Example 1:

```
Input: ["a==b","b!=a"]
Output: false
Explanation: If we assign say, a = 1 and b = 1, then the
first equation is satisfied, but not the second.  There is no
way to assign the variables to satisfy both equations.
```

Example 2:

```
Input: ["b==a","a==b"]
Output: true
Explanation: We could assign a = 1 and b = 1 to satisfy both
equations.
```

Example 3:

```
Input: ["a==b","b==c","a==c"]
Output: true
```

Example 4:

```
Input: ["a==b","b!=c","c==a"]
Output: false
```

Example 5:

```
Input: ["c==c","b==d","x!=z"]
Output: true
```

Note:

1. `1 <= equations.length <= 500`
2. `equations[i].length == 4`
3. `equations[i][0]` and `equations[i][3]` are lowercase letters
4. `equations[i][1]` is either `'='` or `'!'`
5. `equations[i][2]` is `'='`

1. "==" : union them.
2. "!=" : determine their not connection.

```cpp
class UF {
public:
    // construct function
    UF(int n) {
        this->_count = n;
        // point parent to self
        // init size as 1
        this->parent = new int[n];
        this->size = new int[n];
        for (int i = 0; i < n; i++) {
            this->parent[i] = i;
            this->size[i] = 1;
        }
    }

    // union p and q
    void _union(int p, int q) {
        int rootP = find(p);
        int rootQ = find(q);
        if (rootP == rootQ) {
            return;
        }

        // connect small tree below big tree
        if (this->size[rootP] < this->size[rootQ]) {
            this->parent[rootP] = rootQ;
        } else {
            this->parent[rootQ] = rootP;
        }

        this->parent[rootP] = rootQ;
        this->_count--;
    }

    // determine whether p an q in the same subset
    bool connected(int p, int q) {
        int rootP = find(p);
        int rootQ = find(q);
        return rootP == rootQ;
    }

    // return the number of subset
    int count() {
        return this->_count;
    }

private:
    int _count;
    int *parent;
    int *size;

    int find(int x) {
        while(this->parent[x] != x) {
            this->parent[x] = this->parent[this->parent[x]];
            x = this->parent[x];
        }
        return x;
    }
};
```

```
60
61  class Solution {
62  public:
63      bool equationsPossible(vector<string>& equations) {
64          // initial uf
65          UF *uf = new UF(26);
66
67          // for "==", union them
68          for (auto equation : equations) {
69              if (equation.at(1) == '=') {
70                  uf->_union(equation.at(0) - 'a', equation.at(3) - 'a');
71              }
72          }
73
74          // for "!=", determine their not connected
75          for (auto equation : equations) {
76              if (equation.at(1) == '!') {
77                  if (uf->connected(equation.at(0) - 'a', equation.at(3) -
    'a')) {
78                      return false;
79                  }
80              }
81          }
82
83          return true;
84      }
85  };
```