757. Pyramid Transition Matrix

DescriptionHintsSubmissionsDiscussSolution

We are stacking blocks to form a pyramid. Each block has a color which is a one letter string, like `'Z'`.

For every block of color `C` we place not in the bottom row, we are placing it on top of a left block of color `A` and right block of color `B`. We are allowed to place the block there only if `(A, B, C)` is an allowed triple.

We start with a bottom row of `bottom`, represented as a single string. We also start with a list of allowed triples `allowed`. Each allowed triple is represented as a string of length 3.

Return true if we can build the pyramid all the way to the top, otherwise false.

Example 1:

```
Input: bottom = "XYZ", allowed = ["XYD", "YZE", "DEA", "FFF"]
Output: true
Explanation:
We can stack the pyramid like this:
    A
   / \
  D   E
 / \ / \
X   Y   Z

This works because ('X', 'Y', 'D'), ('Y', 'Z', 'E'), and ('D', 'E', 'A') are
allowed triples.
```

Example 1:

```
Input: bottom = "XXYX", allowed = ["XXX", "XXY", "XYX", "XYY", "YXZ"]
Output: false
Explanation:
We can't stack the pyramid to the top.
Note that there could be allowed triples (A, B, C) and (A, B, D) with C != D.
```

Note:

1. `bottom` will be a string with length in range `[2, 12]`.
2. `allowed` will have length in range `[0, 343]`.
3. Letters in all strings will be chosen from the set `{'A', 'B', 'C', 'D', 'E', 'F', 'G'}`.

```
//java
class Solution {
    public boolean pyramidTransition(String bottom, List<String> allowed) {
        List<Set<Character>> list = new ArrayList<>();
        for(int i = 0; i < bottom.length(); i++) {
            list.add(new HashSet<>());
            list.get(i).add(bottom.charAt(i));
        }
        Map<String, List<Character>> map = new HashMap<>();
        for(String s : allowed) {
            String key = s.substring(0, 2);
            if(!map.containsKey(key)) map.put(key, new ArrayList<>());
            map.get(key).add(s.charAt(2));
        }
        return helper(list, map);
    }
    public boolean helper(List<Set<Character>> list, Map<String,
List<Character>> map) {
        if(list.size() == 1) return true;
        List<Set<Character>> next = new ArrayList<>();
        for(int i = 1; i < list.size(); i++) {
            next.add(new HashSet<>());
            Set<Character> set1 = list.get(i-1);
            Set<Character> set2 = list.get(i);
            for(Character c1 : set1) {
                for(Character c2 : set2) {
                    if(map.containsKey(c1+""+c2)) next.get(i-
1).addAll(map.get(c1+""+c2));
                }
            }
            if(next.get(i-1).size() == 0) return false;
        }
        return helper(next, map);
    }
}
```

//c++

```
bool helper(vector<unordered_set<char>> list, unordered_map<string,vector<char>>
mymap)

{

      if(list.size()==1) return true;

      vector<unordered_set<char>> listnew(list.size()-1);

      for(int i=1;i<(int)list.size();++i)

      {

            unordered_set<char> set1 = list[i-1];

            unordered_set<char> set2 = list[i];

            for(char c1:set1)

            {

                  for(char c2:set2)

                  {

                        char tmper[20];

                        sprintf(tmper,"%c%c",c1,c2);

                        cout<<string(tmper)<<endl;
                        if(mymap.count(string(tmper)))
                        {
                              vector<char> values = mymap[tmper];
                              for(char cc:values)
                                    listnew[i-1].insert(cc);
                        }
                  }
            }
            if(listnew[i-1].size()==0) return false;
      }
      return helper(listnew,mymap);
}

bool pyramidTransition(string bottom, vector<string>& allowed) {
    vector<unordered_set<char>> list(bottom.size());
    for(int i=0;i<(int)bottom.size();++i)
    {
             list[i].insert(bottom[i]);
       }
       unordered_map<string,vector<char>> mymap;
       for(auto s:allowed)
       {
             string tmp = s.substr(0,2);
             //cout<<tmp<<endl;
             if(!mymap.count(tmp))
             {
                   vector<char> myVec;
                   mymap[tmp]=myVec;
                   mymap[tmp].push_back(s[2]);
             }else{
                   mymap[tmp].push_back(s[2]);
             }
       }
       return helper(list,mymap);
```

```
}
```