Pyramid Transition Matrix

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 2:

```
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, 8]`.
2. `allowed` will have length in range `[0, 200]`.
3. Letters in all strings will be chosen from the set `{'A', 'B', 'C', 'D', 'E', 'F', 'G'}`.

Solution 1

For the following example:
"ABCD"
["BCE","BCF","ABA","CDA","AEG","FAG","GGG"]
Should we output false (if I am interpreting the problem correctly)?
But the standard code outputs true. Could anyone explain this?

written by figurative original link here

Solution 2

The standard DP solution cannot pass the most upvoted counter example. Here is my DFS/backtracking solution with a couple optimizations. The run time is ~1000 ms (now 6 ms possibly due to large test cases removed). The big O run time would be exponential.

Theoretically, BFS should work. We can use unordered_set<string> to keep all possible strings for each level. Then move up and generate possible combinations for next level. When reaching the top, if the set is non-empty, return true. However, it will result in TLE. The advantage of DFS is early termination. Once we find a valid solution, we don't have to traverse the whole tree, although the worse case runtime is the same.

A couple optimization:

1. memoization: unordered_set<string> invalid, to keep all the strings that won't work.
2. 2D vector<char> to keep edges, such as "ABE".
3. early termination if a string cannot generate a string of next level. For example, "ABFE" while there is no edge of "BF#".

```
class Solution {
public:
    bool pyramidTransition(string bottom, vector<string>& allowed) {
        unordered_set<string> invalid;
        // 7 characters from A to G, size 49 is sufficient.
        vector<vector<char>> edges(49);
        for (string& s: allowed) {
            int key = (s[0]-'A')*7+s[1]-'A';
            edges[key].push_back(s[2]);
        }
        return helper(invalid, bottom, edges);
    }
private:
    bool helper(unordered_set<string>& invalid, string& bottom, vector<vector<char>
>& edges) {
        if (bottom.size() <= 1) return true;
        if (invalid.count(bottom)) return false;
        int n = bottom.size();
        // early termination if next level string is impossible
        for (int i = 0; i < n-1; i++) {
            int key = (bottom[i]-'A')*7+bottom[i+1]-'A';
            if (edges[key].empty()) {
                invalid.insert(bottom);
                return false;
            }
        }
        // try all possible strings (from backtracking) of next level
        string path(n-1, 'A');
        if (dfs(invalid, bottom, edges, path, 0)) return true;
        invalid.insert(bottom);
        return false;
    }
    bool dfs(unordered_set<string>& invalid, string& s, vector<vector<char>>& edges,
string& path, int idx) {
        // find a possible string of next level
        if (idx+1 == s.size()) return helper(invalid, path, edges);
        int key = (s[idx]-'A')*7+s[idx+1]-'A';
        for (char c: edges[key]) {
            path[idx] = c;
            if (dfs(invalid, s, edges, path, idx+1)) return true;
        }
        return false;
    }
};
```

written by zestypanda original link here

Solution 3

Please refer to the discussion here https://leetcode.com/articles/pyramid-transition-matrix/
I can easily find a few submitted solutions are DP-based, which must fail on this test case.
"AAAA"
["AAB","AAC","BCD","BBE","DEF"]
The correct answer should be "false".
Unfortunately, the 1st, 2nd and 3rd winners in this contest are ALL WRONG for this problem.

written by guan.huifeng original link here