Split Assembled Strings

Given a list of strings, you could assemble these strings together into a loop. Among all the possible loops, you need to find the lexicographically biggest string after cutting and making one breakpoint of the loop, which will make a looped string into a regular one.

So, to find the lexicographically biggest string, you need to experience two phases:

1. Assemble all the strings into a loop, where you can reverse some strings or not and connect them in the same order as given.
2. Cut and make one breakpoint in any place of the loop, which will make a looped string into a regular string starting from the character at the cutting point.

And your job is to find the lexicographically biggest one among all the regular strings.

Example:

```
Input: "abc", "xyz"
Output: "zyxcba"
Explanation: You can get the looped string "-abcxyz-", "-abczyx-", "-cbaxyz-", "-cbaz
yx-",
where '-' represents the looped status.
The answer string came from the fourth looped one,
where you could cut from the middle and get "zyxcba".
```

Note:

1. The input strings will only contain lowercase letters.
2. The total length of all the strings will not over 1000.

Solution 1

To solve this problem, we must keep in heart the following points:

1. We know the cut point must come from the one string, assumed it is called c-string.
2. Then except the c-string, **all the other string** must become its lexicographically biggest status, assumed it is called b-status. Since only in this situation, we could get the lexicographically biggest string after cutting.
3. To reach the point 2, we need to first let all the string reach its b-status for the convenience of traversing all the strings afterward.
4. Then, for each string's traversal procedure, we need to decide whether it should be reversed or not since we don't know which might generate the final answer, and then we enumerated all the characters in this string.

```cpp
class Solution {
public:
    int n;
    string ans = "";
    void solve(vector<string>& strs, int i, bool flag) {
        string temp = strs[i];
        if (flag) reverse(temp.begin(), temp.end()); // reverse the string
        int size = (int)temp.size();
        for (int k=0; k<size; ++k) {
            string str1 = "";
            string str2 = "";

            for (int j=i+1; j<n; ++j) str1 += strs[j]; // all the string behind the strs[i]
            for (int j=0; j<i; ++j) str2 += strs[j]; // all the string before the strs[i]

            string newOne = temp.substr(k) + str1 + str2 + temp.substr(0, k); // cut the looped string at the place indexed k
            if (ans == "") ans = newOne;
            else if (ans < newOne) ans = newOne;
        }
    }
    string splitLoopedString(vector<string>& strs) {
        n = (int)strs.size();
        if (n == 0) return "";
        // Reach the b-status for all the strings.
        for (int i=0; i<n; ++i) {
            string temp = strs[i];
            reverse(temp.begin(), temp.end());
            strs[i] = strs[i] > temp ? strs[i] : temp;
        }
        for (int i=0; i<n; ++i) {
            solve(strs, i, true); // reverse the string situation
            solve(strs, i, false); // not reverse the string situation
        }
        return ans;
    }
};
```

written by love_Fawn original link here

Solution 2

For every starting direction and letter, let's determine the best string we can make. For subsequent fragments we encounter, we always want them flipped in the orientation that makes them largest.

Thus, for every token, for every starting direction, for every starting letter in the token, we can compute the candidate string directly. We take the maximum of these.

```
def splitLoopedString(self, A):
    B = [max(x, x[::-1]) for x in A]
    ans = None
    for i, token in enumerate(B):
        for start in (token, token[::-1]):
            for j in xrange(len(start) + 1):
                ans = max(ans, start[j:] + "".join(B[i+1:] + B[:i]) + start[:j])
    return ans
```

written by awice original link here

Solution 3

The idea is similar to this post by @love_FDU_llp; just optimized for brevity and performance (6 ms vs. 100 ms).
Optimization 1: do not check the cutting point if the first letter is smaller than the first letter of the current best result.
Optimization 2: if a sub-string best result is the same as the current best result, then there is a lop and we are done.
I also tried few more ideas, but it did not improve OJ runtime but increased the number of lines.

```
string splitLoopedString(vector<string>& strs) {
    string s = "", res = "a";
    for (auto i = 0; i < strs.size(); ++i) {
        auto r = strs[i];
        reverse(r.begin(), r.end());
        s += max(r, strs[i]);
    }
    for (auto i = 0, st = 0; i < strs.size(); st += strs[i++].size()) {
        string p1 = strs[i], p2 = strs[i], c_res = string(1, res[0]), body = s.subs
tr(st + p1.size()) + s.substr(0, st);
        reverse(p2.begin(), p2.end());
        for (auto j = 0; j < strs[i].size(); ++j) {
            if (p1[j] >= c_res[0]) c_res = max(c_res, p1.substr(j) + body + p1.subs
tr(0, j));
            if (p2[j] >= c_res[0]) c_res = max(c_res, p2.substr(j) + body + p2.subs
tr(0, j));
        }
        if (res == c_res) return res;
        res = max(res, c_res);
    }
    return res;
}
```

written by votrubac original link here