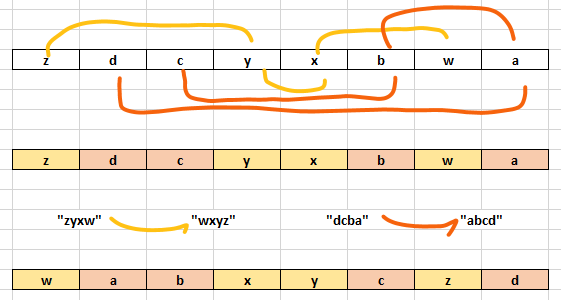
Intuition

If a group of characters is interconnected by swap pairs, you can freely rearrange characters within that group.

In this example, we have two group of interconnected characters, so we can make rearrangements to achieve the smallest string as shown in the picture below.

"zdcyxbwa"

[[0,3],[4,6],[3,4],[1,7],[2,5],[5,7]]



Solution

Identify groups using union-find. For each group, collect all its characters in a string.

Note that in the disjoined set ds, the find operation returns the parent index (p). We use that index to identify the group, and put all group characters into m[p].

Sort the string, then put the rearranged characters back to their respective positions in the group.

int find(vector<int>& ds, int i) {

return ds[i] < 0 ? i : ds[i] = find(ds, ds[i]);

}

string smallestStringWithSwaps(string s, vector<vector<int>>& pairs) {

vector<int> ds(s.size(), -1);

vector<vector<int>> m(s.size());

for (auto& p : pairs) {

auto i = find(ds, p[0]), j = find(ds, p[1]);

if (i != j)

ds[j] = i;

}

for (auto i = 0; i < s.size(); ++i)

m[find(ds, i)].push\_back(i);

for (auto &ids : m) {

string ss = "";

for (auto id : ids)

ss += s[id];

sort(begin(ss), end(ss));

for (auto i = 0; i < ids.size(); ++i)

s[ids[i]] = ss[i];

}

return s;

}

Disjoint set optimizations

Note that, in the find function, we are using path compression. In other words, we "flatten" our hierarchy every time we search, so that elements references the parent index directly: ds[i] = find(ds, ds[i]);. Without path compression, you will get TLE on the last test case.

In addition, we can store the count of elements (negated) in the parent of the group. Then, when merging two groups, we add elements from the smaller group into the larger one. That way, we reduce the number of recursive calls for the find function to "flatten" newly added elements.

Below is the optimized solution (the changes are for ```if (i != j)```` condition). Note that, for this problem, this optimization is not that important (though the runtime went down from 140 to 128 ms for me) since we only have 26 elements. Though, it may help you quite a bit when you deal with larger groups.

int find(vector<int>& ds, int i) {

return ds[i] < 0 ? i : ds[i] = find(ds, ds[i]);

}

string smallestStringWithSwaps(string s, vector<vector<int>>& pairs) {

vector<int> ds(s.size(), -1);

vector<vector<int>> m(s.size());

for (auto& p : pairs) {

auto i = find(ds, p[0]), j = find(ds, p[1]);

if (i != j) {

if (-ds[i] < -ds[j])

swap(i, j);

ds[i] += ds[j];

ds[j] = i;

}

}

for (auto i = 0; i < s.size(); ++i)

m[find(ds, i)].push\_back(i);

for (auto &ids : m) {

string ss = "";

for (auto id : ids)

ss += s[id];

sort(begin(ss), end(ss));

for (auto i = 0; i < ids.size(); ++i)

s[ids[i]] = ss[i];

}

return s;

}

Complexity Analysis

* Runtime: O(n log n)
* Memory: O(n)