function

algorithm - Generalize the Boyer–Moore majority vote algorithm

Mgen

Oct 6, 2016

The Boyer–Moore majority vote algorithm finds a majority element from a given list of numbers, that is, the element that appears more than half of the time. In this post, we will try to generalize this algorithm and find the element that appears more than `n / k` times, where `n` is the length of the list.

In the original algorithm, only one counter is used, in this generalized algorithm, `k` counters are maintained, each of them stores the number of occurrences of an element, this can be easily implemented via a hash table.

For each element in our list, if the number of counters is less than `k` or current element exists in the hash table, increase the corresponding counter by 1, if current element does not exist in the hash table, decrease all counters by 1 and remove all 0 values.

After the first pass, there will be `k` candidates in the hash table, during the second pass, each candidate will be further verified, if a candidate does not appear more than `n / k` times, it would be rejected.

So our generalized algorithm will run in `O(n * k)` , can we optimize it further? Yes, we can maintain another hash table to count the occurrences of each element, during the second pass, just look up the number of occurrences in hash table, since looking up in a hash table takes `O(1)` , so the overall time complexity is `O(n + k)` or `O(n)` for simplicity.

C++ implementation:

```cpp
// Finds all elements that appears more than "nums.size() / k" times
vector<int> findRepeatedElements(vector<int>& nums, int k) {
    unordered_map<int, int> counters;
    unordered_map<int, int> occurrences;
    for (auto n: nums) {
        occurrences[n]++;

        // maintains k counters at most
        if (counters.size() < k - 1 || counters.count(n)) {
            counters[n]++;
        } else {
            // decrease all counters by 1
            // remove 0 values
            auto it = counters.begin();
            while (it != counters.end()) {
                if (it->second == 1) {
                    it = counters.erase(it);
                } else {
                    counters[it->first]--;
                    it++;
                }
            }
```

Some examples:

```cpp
vector<int> nums1 = {1, 2, 3, 4, 1, 1, 3, 3};
vector<int> nums2 = {1, 2, 3, 4, 5, 6};

// 3 and 1 are majority elements
auto result = findRepeatedElements(nums1, 4);
for (auto n: result) {
    cout << n << " ";
}
cout << endl;

// No majority element in nums2
result = findRepeatedElements(nums2, 3);
for (auto n: result) {
    cout << n << " ";
}
cout << endl;
```

Output:

```
3 1
```