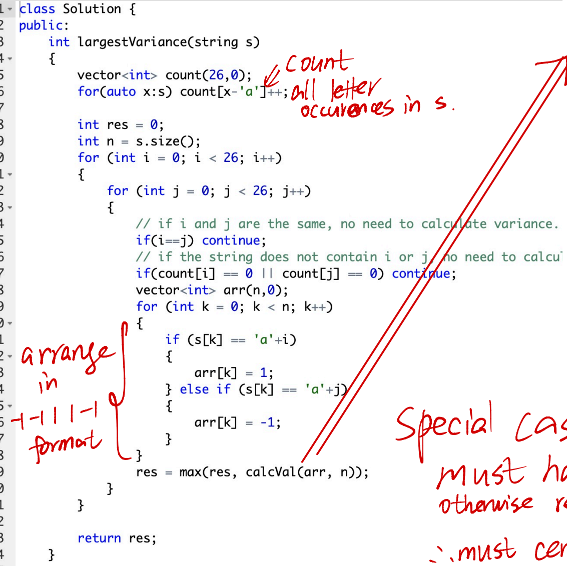
Intuition:

Kadane's algorithm:

find maximum sum of subarrays ending with i;

To solve this, need to loop over all letter pairs:
for 2 letters of interest, assign 1 and -1, and sum up subarrays to find maximum ⇒ Kadane's algorithm

First attempt:



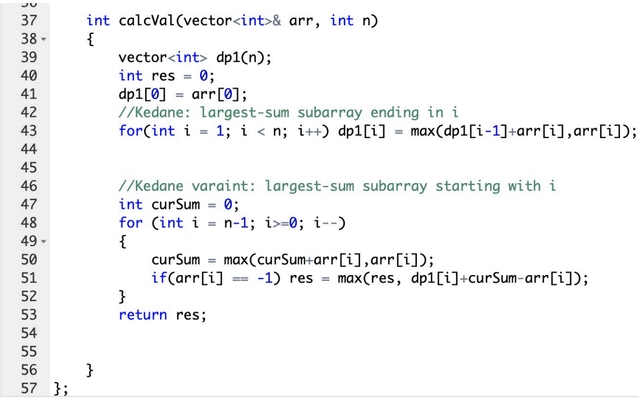
```
class Solution {
public:
    int largestVariance(string s)
    {
        vector<int> count(26,0);
        for(auto x:s) count[x-'a']++;

        int res = 0;
        int n = s.size();
        for (int i = 0; i < 26; i++)
        {
            for (int j = 0; j < 26; j++)
            {
                // if i and j are the same, no need to calculate variance.
                if(i==j) continue;
                // if the string does not contain i or j, no need to calcu
                if(count[i] == 0 || count[j] == 0) continue;
                vector<int> arr(n,0);
                for (int k = 0; k < n; k++)
                {
                    if (s[k] == 'a'+i)
                    {
                        arr[k] = 1;
                    } else if (s[k] == 'a'+j)
                    {
                        arr[k] = -1;
                    }
                }
                res = max(res, calcVal(arr, n));
            }
        }

        return res;
    }
```

count all letter occurrences in s.

arrange in -1-1 1 1-1 format



```
    int calcVal(vector<int>& arr, int n)
    {
        vector<int> dp1(n);
        int res = 0;
        dp1[0] = arr[0];
        //Kedane: largest-sum subarray ending in i
        for(int i = 1; i < n; i++) dp1[i] = max(dp1[i-1]+arr[i],arr[i]);


        //Kedane varaint: largest-sum subarray starting with i
        int curSum = 0;
        for (int i = n-1; i>=0; i--)
        {
            curSum = max(curSum+arr[i],arr[i]);
            if(arr[i] == -1) res = max(res, dp1[i]+curSum-arr[i]);
        }
        return res;


    }
};
```

Special case:

must have -1 in subarray.
otherwise result is wrong

∴ must center around -1

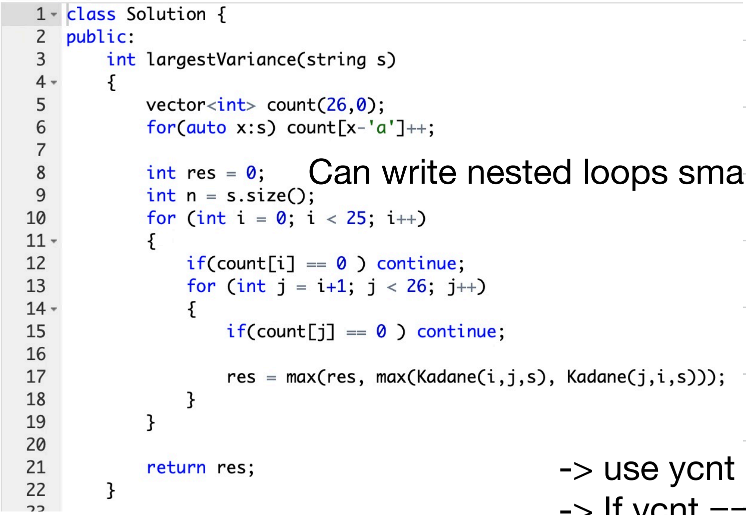
find max subarray ending in this -1

& find max subarray starting with this -1

then result = max1 + max2 - (-1)

but can improve a lot

better solution:



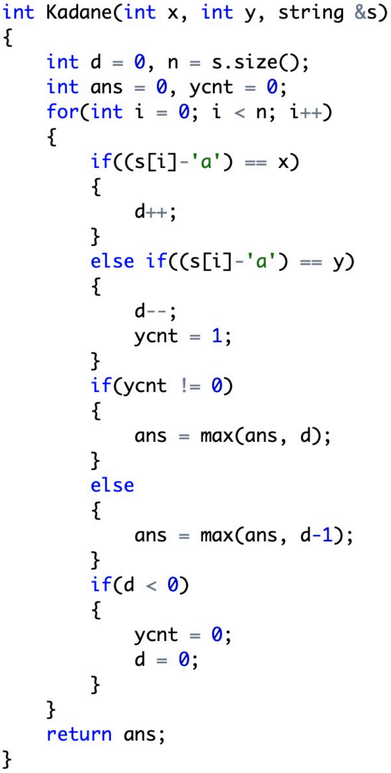
```cpp
class Solution {
public:
    int largestVariance(string s)
    {
        vector<int> count(26,0);
        for(auto x:s) count[x-'a']++;

        int res = 0;
        int n = s.size();
        for (int i = 0; i < 25; i++)
        {
            if(count[i] == 0 ) continue;
            for (int j = i+1; j < 26; j++)
            {
                if(count[j] == 0 ) continue;

                res = max(res, max(Kadane(i,j,s), Kadane(j,i,s)));
            }
        }

        return res;
    }
```

Can write nested loops smarter such that i != j



```cpp
int Kadane(int x, int y, string &s)
{
    int d = 0, n = s.size();
    int ans = 0, ycnt = 0;
    for(int i = 0; i < n; i++)
    {
        if((s[i]-'a') == x)
        {
            d++;
        }
        else if((s[i]-'a') == y)
        {
            d--;
            ycnt = 1;
        }
        if(ycnt != 0)
        {
            ans = max(ans, d);
        }
        else
        {
            ans = max(ans, d-1);
        }
        if(d < 0)
        {
            ycnt = 0;
            d = 0;
        }
    }
    return ans;
}
```

-> use ycnt as a flag to restart.
-> If ycnt == 0, that means when have not encountered a y value yet, so we need to do d-1 to make sure we don't just count
-> If d < 0, that means # of y occurrence is larger than # of x occurrence. In this case, we need to restart.
-> Kadane function counts for all subarrays in s, how many more occurrences does x have compared to y. This value is either positive or 0.
-> Two Kadane calls ensures for each unique (x,y) pair available in s, we count all largest occurrence difference.