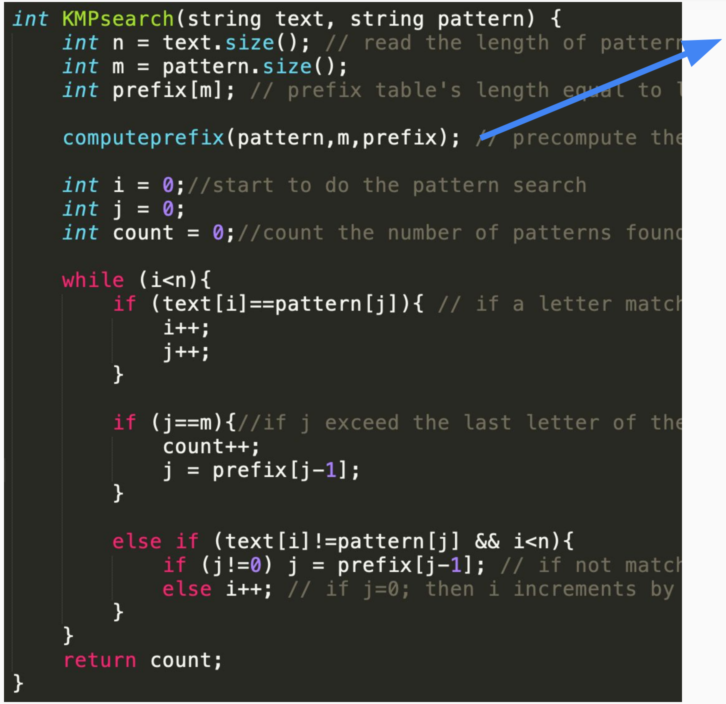
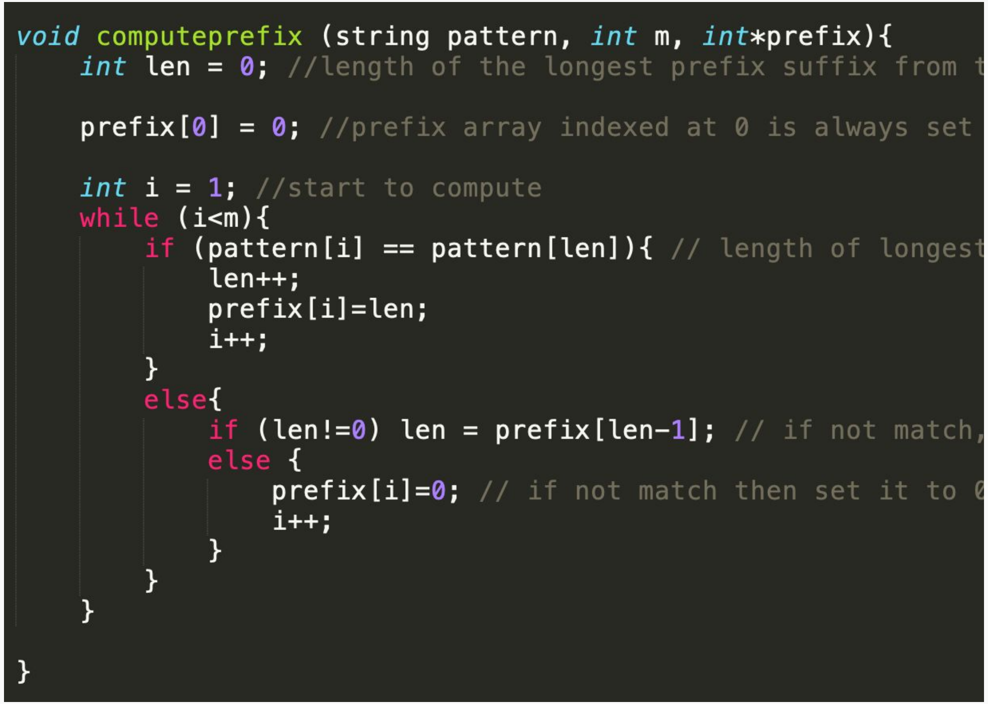
EC504 Project

Tao Zhang, Zehao Hui, Junyi Tang

KMP Searching Algorithm



```
int KMPsearch(string text, string pattern) {
    int n = text.size(); // read the length of pattern
    int m = pattern.size();
    int prefix[m]; // prefix table's length equal to l

    computeprefix(pattern,m,prefix); // precompute the

    int i = 0;//start to do the pattern search
    int j = 0;
    int count = 0;//count the number of patterns found

    while (i<n){
        if (text[i]==pattern[j]){ // if a letter match
            i++;
            j++;
        }

        if (j==m){//if j exceed the last letter of the
            count++;
            j = prefix[j-1];
        }

        else if (text[i]!=pattern[j] && i<n){
            if (j!=0) j = prefix[j-1]; // if not match
            else i++; // if j=0; then i increments by
        }
    }
    return count;
}
```



```
void computeprefix (string pattern, int m, int*prefix){
    int len = 0; //length of the longest prefix suffix from t

    prefix[0] = 0; //prefix array indexed at 0 is always set

    int i = 1; //start to compute
    while (i<m){
        if (pattern[i] == pattern[len]){ // length of longest
            len++;
            prefix[i]=len;
            i++;
        }
        else{
            if (len!=0) len = prefix[len-1]; // if not match,
            else {
                prefix[i]=0; // if not match then set it to 0
                i++;
            }
        }
    }

}
```

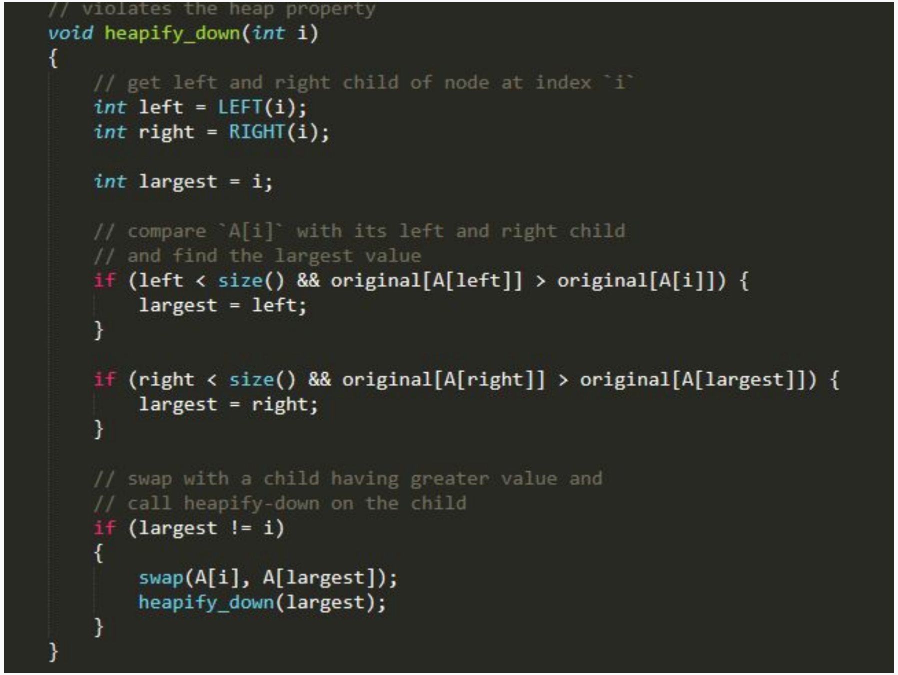
KMP Searching Algorithm

- Time Complexity: O(n+m) -> O(n)
    - #n is the total number of characters in tweets
    - #m is the number of characters in query
    - As m is much smaller than n if n is large enough, the time complexity will approach O(n)

KMP with Input Logic

- Time Complexity: O(k*n)
  - #n is the total number of characters in tweets
  - #k is the number of words in the query
  - Since we run a KMP algorithm on each word in the query, KMP algorithm will induce O(k*n) for time complexity.

Max Heap for Ranking



```
// violates the heap property
void heapify_down(int i)
{
    // get left and right child of node at index `i`
    int left = LEFT(i);
    int right = RIGHT(i);

    int largest = i;

    // compare `A[i]` with its left and right child
    // and find the largest value
    if (left < size() && original[A[left]] > original[A[i]]) {
        largest = left;
    }

    if (right < size() && original[A[right]] > original[A[largest]]) {
        largest = right;
    }

    // swap with a child having greater value and
    // call heapify-down on the child
    if (largest != i)
    {
        swap(A[i], A[largest]);
        heapify_down(largest);
    }
}
```

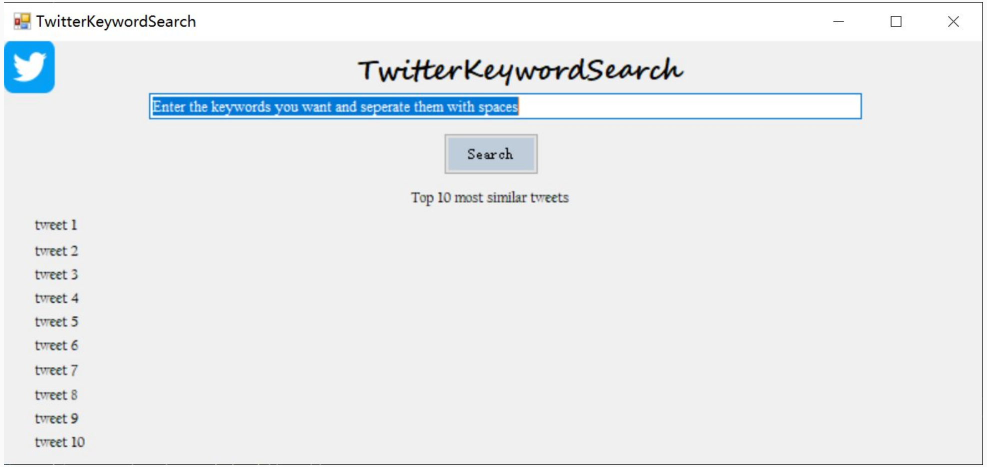
Total time complexity is O(n+logn)

Normally we compare the number in our heap

In this project, we compare store index and compare the A[index]

GUI - CLR in Visual Studio 2019

- Time Complexity: O(1)



Enter the keywords you want and seperate them with spaces

tweet 1
tweet 2
tweet 3
tweet 4
tweet 5
tweet 6
tweet 7
tweet 8
tweet 9
tweet 10

Thanks!
Here comes the demo: