Given a sequence of words, print all anagrams together | Set 2

Given an array of words, print all anagrams together. For example, if the given array is {“cat”, “dog”, “tac”, “god”, “act”}, then output may be “cat tac act dog god”.

We have discussed two different methods in the previous post. In this post, a more efficient solution is discussed.

Trie data structure can be used for a more efficient solution. Insert the sorted order of each word in the trie. Since all the anagrams will end at the same leaf node. We can start a linked list at the leaf nodes where each node represents the index of the original array of words. Finally, traverse the Trie. While traversing the Trie, traverse each linked list one line at a time. Following are the detailed steps.

1) Create an empty Trie
2) One by one take all words of input sequence. Do following for each word
…**a)** Copy the word to a buffer.
…**b)** Sort the buffer
…**c)** Insert the sorted buffer and index of this word to Trie. Each leaf node of Trie is head of a Index list. The Index list stores index of words in original sequence. If sorted buffe is already present, we insert index of this word to the index list.
3) Traverse Trie. While traversing, if you reach a leaf node, traverse the index list. And print all words using the index obtained from Index list.

```
// An efficient program to print all anagrams together
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define NO_OF_CHARS 26

// Structure to represent list node for indexes of words in
// the given sequence. The list nodes are used to connect
// anagrams at leaf nodes of Trie
struct IndexNode
{
    int index;
    struct IndexNode* next;
};

// Structure to represent a Trie Node
struct TrieNode
{
    bool isEnd;  // indicates end of word
    struct TrieNode* child[NO_OF_CHARS]; // 26 slots each for 'a' to 'z'
    struct IndexNode* head; // head of the index list
};


// A utility function to create a new Trie node
struct TrieNode* newTrieNode()
{
    struct TrieNode* temp = new TrieNode;
    temp->isEnd = 0;
    temp->head = NULL;
    for (int i = 0; i < NO_OF_CHARS; ++i)
        temp->child[i] = NULL;
    return temp;
}
```

```
/* Following function is needed for library function qsort(). Refer
   http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
int compare(const void* a, const void* b)
{  return *(char*)a - *(char*)b; }

/* A utility function to create a new linked list node */
struct IndexNode* newIndexNode(int index)
{
    struct IndexNode* temp = new IndexNode;
    temp->index = index;
    temp->next = NULL;
    return temp;
}

// A utility function to insert a word to Trie
void insert(struct TrieNode** root, char* word, int index)
{
    // Base case
    if (*root == NULL)
        *root = newTrieNode();

    if (*word != '\0')
        insert( &( (*root)->child[tolower(*word) - 'a'] ), word+1, index );
    else  // If end of the word reached
    {
        // Insert index of this word to end of index linked list
        if ((*root)->isEnd)
        {
            IndexNode* pCrawl = (*root)->head;
            while( pCrawl->next )
                pCrawl = pCrawl->next;
            pCrawl->next = newIndexNode(index);
        }
        else  // If Index list is empty
        {
            (*root)->isEnd = 1;
            (*root)->head = newIndexNode(index);
        }
    }
}

// This function traverses the built trie. When a leaf node is reached,
// all words connected at that leaf node are anagrams. So it traverses
// the list at leaf node and uses stored index to print original words
void printAnagramsUtil(struct TrieNode* root, char *wordArr[])
{
    if (root == NULL)
        return;

    // If a lead node is reached, print all anagrams using the indexes
    // stored in index linked list
    if (root->isEnd)
    {
        // traverse the list
        IndexNode* pCrawl = root->head;
        while (pCrawl != NULL)
        {
            printf( "%s \n", wordArr[ pCrawl->index ] );
            pCrawl = pCrawl->next;
        }
    }

    for (int i = 0; i < NO_OF_CHARS; ++i)
        printAnagramsUtil(root->child[i], wordArr);
}

// The main function that prints all anagrams together. wordArr[] is input
// sequence of words.
void printAnagramsTogether(char* wordArr[], int size)
{
```

```
    // Create an empty Trie
    struct TrieNode* root = NULL;

    // Iterate through all input words
    for (int i = 0; i < size; ++i)
    {
        // Create a buffer for this word and copy the word to buffer
        int len = strlen(wordArr[i]);
        char *buffer = new char[len+1];
        strcpy(buffer, wordArr[i]);

        // Sort the buffer
        qsort( (void*)buffer, strlen(buffer), sizeof(char), compare );

        // Insert the sorted buffer and its original index to Trie
        insert(&root,  buffer, i);
    }

    // Traverse the built Trie and print all anagrms together
    printAnagramsUtil(root, wordArr);
}


// Driver program to test above functions
int main()
{
    char* wordArr[] = {"cat", "dog", "tac", "god", "act", "gdo"};
    int size = sizeof(wordArr) / sizeof(wordArr[0]);
    printAnagramsTogether(wordArr, size);
    return 0;
}
```

Output:

```
cat
tac
act
dog
god
gdo
```