Given a sequence of words, print all anagrams together | Set 1

Difficulty Level : Medium • Last Updated : 06 Jul, 2022

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”.

Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.

A **simple method** is to create a Hash Table. Calculate the hash value of each word in such a way that all anagrams have the same hash value. Populate the Hash Table with these hash values. Finally, print those words together with the same hash values. A simple hashing mechanism can be modulo sum of all characters. With modulo sum, two non-anagram words may have the same hash value. This can be handled by matching individual characters.

Following is **another method** to print all anagrams together. Take two auxiliary arrays, index array, and word array. Populate the word array

indices. After sorting, all the anagrams cluster together. Use the index array to print the strings from the original array of strings.

Let us understand the steps with the following input Sequence of Words:

```
"cat", "dog", "tac", "god", "act"
```

1) Create two auxiliary arrays index[] and words[]. Copy all given words to words[] and store the original indexes in index[]

```
index[]:  0   1   2   3   4
words[]: cat dog tac god act
```

2) Sort individual words in words[]. Index array doesn't change.

```
index[]:   0    1    2    3    4
words[]:  act  dgo  act  dgo  act
```

3) Sort the words array. Compare individual words using strcmp() to sort

```
index:     0    2    4    1    3
words[]:  act  act  act  dgo  dgo
```

4) All anagrams come together. But words are changed in the words array. To print the original words, take the index from the index array and use it in the original array. We get

```
"cat tac act dog god"
```

Following are the implementations of the above algorithm. In the

C++

```
// A C++ program to print all anagrams together
#include <bits/stdc++.h>
#include <string.h>
using namespace std;

// structure for each word of duplicate array
class Word {
public:
    char* str; // to store word itself
    int index; // index of the word in the original array
};

// structure to represent duplicate array.
class DupArray {
public:
    Word* array; // Array of words
    int size; // Size of array
};

// Create a DupArray object that contains an array of Words
DupArray* createDupArray(char* str[], int size)
{
    // Allocate memory for dupArray and all members of it
    DupArray* dupArray = new DupArray();
    dupArray->size = size;
    dupArray->array
        = new Word[(dupArray->size * sizeof(Word))];

    // One by one copy words from the given wordArray to
    // dupArray
    int i;
    for (i = 0; i < size; ++i) {
        dupArray->array[i].index = i;
        dupArray->array[i].str
            = new char[(strlen(str[i]) + 1)];
        strcpy(dupArray->array[i].str, str[i]);
    }
```

```
// sorting an array of characters (Word)
int compChar(const void* a, const void* b)
{
    return *(char*)a - *(char*)b;
}

// Compare two words. Used in qsort()
// for sorting an array of words
int compStr(const void* a, const void* b)
{
    Word* a1 = (Word*)a;
    Word* b1 = (Word*)b;
    return strcmp(a1->str, b1->str);
}

// Given a list of words in wordArr[],
void printAnagramsTogether(char* wordArr[], int size)
{
    // Step 1: Create a copy of all words present in given
    // wordArr. The copy will also have original indexes of
    // words
    DupArray* dupArray = createDupArray(wordArr, size);

    // Step 2: Iterate through all words in dupArray and
    // sort individual words.
    int i;
    for (i = 0; i < size; ++i)
        qsort(dupArray->array[i].str,
              strlen(dupArray->array[i].str), sizeof(char),
              compChar);

    // Step 3: Now sort the array of words in dupArray
    qsort(dupArray->array, size, sizeof(dupArray->array[0]),
          compStr);

    // Step 4: Now all words in dupArray are together, but
    // these words are changed. Use the index member of word
    // struct to get the corresponding original word
    for (i = 0; i < size; ++i)
        cout << wordArr[dupArray->array[i].index] << " ";
}
```

```
    char* wordArr[] = { "cat", "dog", "tac", "god", "act" };
    int size = sizeof(wordArr) / sizeof(wordArr[0]);
    printAnagramsTogether(wordArr, size);
    return 0;
}

// This is code is contributed by rathbhupendra
```

C

```
// A C program to print all anagrams together
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// structure for each word of duplicate array
struct Word {
    char* str; // to store word itself
    int index; // index of the word in the original array
};

// structure to represent duplicate array.
struct DupArray {
    struct Word* array; // Array of words
    int size; // Size of array
};

// Create a DupArray object that contains an array of Words
struct DupArray* createDupArray(char* str[], int size)
{
    // Allocate memory for dupArray and all members of it
    struct DupArray* dupArray
        = (struct DupArray*)malloc(sizeof(struct DupArray));
    dupArray->size = size;
    dupArray->array = (struct Word*)malloc(
        dupArray->size * sizeof(struct Word));

    // One by one copy words from the given wordArray to
    // dupArray
    int i;
```

```
        strcpy(dupArray->array[i].str, str[i]);
    }

    return dupArray;
}

// Compare two characters. Used in qsort() for sorting an
// array of characters (Word)
int compChar(const void* a, const void* b)
{
    return *(char*)a - *(char*)b;
}

// Compare two words. Used in qsort() for sorting an array
// of words
int compStr(const void* a, const void* b)
{
    struct Word* a1 = (struct Word*)a;
    struct Word* b1 = (struct Word*)b;
    return strcmp(a1->str, b1->str);
}

// Given a list of words in wordArr[],
void printAnagramsTogether(char* wordArr[], int size)
{
    // Step 1: Create a copy of all words present in given
    // wordArr. The copy will also have original indexes of
    // words
    struct DupArray* dupArray
        = createDupArray(wordArr, size);

    // Step 2: Iterate through all words in dupArray and
    // sort individual words.
    int i;
    for (i = 0; i < size; ++i)
        qsort(dupArray->array[i].str,
              strlen(dupArray->array[i].str), sizeof(char),
              compChar);

    // Step 3: Now sort the array of words in dupArray
    qsort(dupArray->array, size, sizeof(dupArray->array[0]),
```

```
    // struct to get the corresponding original word
    for (i = 0; i < size; ++i)
        printf("%s ", wordArr[dupArray->array[i].index]);
}

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

Java

```
// A Java program to print all anagrams together
import java.util.Arrays;
import java.util.Comparator;
public class GFG {
    // class for each word of duplicate array
    static class Word {
        String str; // to store word itself
        int index; // index of the word in the
        // original array

        // constructor
        Word(String str, int index)
        {
            this.str = str;
            this.index = index;
        }
    }

    // class to represent duplicate array.
    static class DupArray {
        Word[] array; // Array of words
        int size; // Size of array

        // constructor
```

```
            // One by one copy words from the
            // given wordArray to dupArray
            int i;
            for (i = 0; i < size; ++i) {
                // create a word Object with the
                // str[i] as str and index as i
                array[i] = new Word(str[i], i);
            }
        }
    }

    // Compare two words. Used in Arrays.sort() for
    // sorting an array of words
    static class compStr implements Comparator<Word> {
        public int compare(Word a, Word b)
        {
            return a.str.compareTo(b.str);
        }
    }

    // Given a list of words in wordArr[],
    static void printAnagramsTogether(String wordArr[],
                                      int size)
    {
        // Step 1: Create a copy of all words present
        // in given wordArr. The copy will also have
        // original indexes of words
        DupArray dupArray = new DupArray(wordArr, size);

        // Step 2: Iterate through all words in
        // dupArray and sort individual words.
        int i;
        for (i = 0; i < size; ++i) {
            char[] char_arr
                = dupArray.array[i].str.toCharArray();
            Arrays.sort(char_arr);
            dupArray.array[i].str = new String(char_arr);
        }

        // Step 3: Now sort the array of words in
```

```
        // but these words are changed. Use the index
        // member of word struct to get the corresponding
        // original word
        for (i = 0; i < size; ++i)
            System.out.print(
                wordArr[dupArray.array[i].index] + " ");
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
        String wordArr[]
            = { "cat", "dog", "tac", "god", "act" };
        int size = wordArr.length;
        printAnagramsTogether(wordArr, size);
    }
}
// This code is contributed by Sumit Ghosh
```

Python

```
# A Python program to print all anagrams together

# structure for each word of duplicate array


class Word(object):
    def __init__(self, string, index):
        self.string = string
        self.index = index

# Create a DupArray object that contains an array
# of Words


def createDupArray(string, size):
    dupArray = []

    # One by one copy words from the given wordArray
    # to dupArray
```

```
# Given a list of words in wordArr[]


def printAnagramsTogether(wordArr, size):
    # Step 1: Create a copy of all words present in
    # given wordArr.
    # The copy will also have original indexes of words
    dupArray = createDupArray(wordArr, size)

    # Step 2: Iterate through all words in dupArray and sort
    # individual words.
    for i in xrange(size):
        dupArray[i].string = ''.join(sorted(dupArray[i].string))

    # Step 3: Now sort the array of words in dupArray
    dupArray = sorted(dupArray, key=lambda k: k.string)

    # Step 4: Now all words in dupArray are together, but
    # these words are changed. Use the index member of word
    # struct to get the corresponding original word
    for word in dupArray:
        print wordArr[word.index],


# Driver program
wordArr = ["cat", "dog", "tac", "god", "act"]
size = len(wordArr)
printAnagramsTogether(wordArr, size)

# This code is contributed by BHAVYA JAIN
```

Output:

```
cat tac act dog god
```

Time Complexity: Let there be N-words and each word has a maximum of M characters. The upper bound is O(NMLogM + MNLogN).

comparison may take maximum O(M) time. So time to sort an array of words will be O(MNLogN).

Using vector of pair :

The problem can easily be solved with the use of a vector of pairs. The pair will be of string and int. The string will require to store the input string and int will require to store their respective indexes.

here is the implementation of the above approach:

C++

```
#include <bits/stdc++.h>
#include <strings.h>
using namespace std;

void createDuplicateArray(
    vector<pair<string, int> >& dupArray,
    vector<string>& wordAr, int size)
{
    for (int i = 0; i < size; i++) {
        dupArray.push_back(make_pair(wordAr[i], i));
        // pair.first contains the input words and
        // pair.second contains its index
    }
}

void printAnagramsTogether(vector<string>& wordArr,
                           int size)
{

    vector<pair<string, int> >
        dupArray; // dupArray to store the word-index pair
    createDuplicateArray(
        dupArray, wordArr,
        size); // making copy of all the words and their
```

```
    int i;
    for (i = 0; i < size; ++i) {
        sort(dupArray[i].first.begin(),
             dupArray[i].first.end());
    }

    // now sort the whole vector to get the identical words
    // together
    sort(dupArray.begin(), dupArray.end());

    // now all the indentical words are together but we have
    // lost the original form of string
    // so through index stored in the word-index pair fetch
    // the original word from main input
    for (i = 0; i < size; ++i)
        cout << wordArr[dupArray[i].second] << " ";
}

int main()
{
    vector<string> wordArr
        = { "cat", "dog", "tac", "god", "act" };
    printAnagramsTogether(wordArr, wordArr.size());
    return 0;
}
```

Output

```
cat tac act dog god
```

Time complexity:

Let there be N-words and each word has a maximum of M characters.

O(NMLogM + MNLogN).

Using hashmap

Here, we first sort each word, use the sorted word as a key and then put

values will be greater than 1.

C++

```
// C++ program to print anagrams
// together using dictionary
#include <bits/stdc++.h>
using namespace std;

void printAnagrams(string arr[], int size)
{
    unordered_map<string, vector<string> > map;

    // Loop over all words
    for (int i = 0; i < size; i++) {

        // Convert to char array, sort and
        // then re-convert to string
        string word = arr[i];
        char letters[word.size() + 1];
        strcpy(letters, word.c_str());
        sort(letters, letters + word.size() + 1);
        string newWord = "";
        for (int i = 0; i < word.size() + 1; i++) {
            newWord += letters[i];
        }

        // Calculate hashcode of string
        // after sorting
        if (map.find(newWord) != map.end()) {
            map[newWord].push_back(word);
        }
        else {

            // This is the first time we are
            // adding a word for a specific
            // hashcode
            vector<string> words;
            words.push_back(word);
```

```
    // If you want to print non-anagrams,
    // just print the values having size = 1
    unordered_map<string, vector<string> >::iterator it;
    for (it = map.begin(); it != map.end(); it++) {
        vector<string> values = map[it->first];
        if (values.size() > 1) {
            cout << "[";
            for (int i = 0; i < values.size() - 1; i++) {
                cout << values[i] << ", ";
            }
            cout << values[values.size() - 1];
            cout << "]";
        }
    }
}

// Driver code
int main()
{
    string arr[] = { "cat", "dog", "tac", "god", "act" };
    int size = sizeof(arr) / sizeof(arr[0]);

    printAnagrams(arr, size);

    return 0;
}

// This code is contributed by Ankit Garg
```

Java

```
// Java program to print anagrams
// together using dictionary
import java.util.*;

public class FindAnagrams {

    private static void printAnagrams(String arr[])
    {
```

```
        for (int i = 0; i < arr.length; i++) {

            // convert to char array, sort and
            // then re-convert to string
            String word = arr[i];
            char[] letters = word.toCharArray();
            Arrays.sort(letters);
            String newWord = new String(letters);

            // calculate hashcode of string
            // after sorting
            if (map.containsKey(newWord)) {

                map.get(newWord).add(word);
            }
            else {

                // This is the first time we are
                // adding a word for a specific
                // hashcode
                List<String> words = new ArrayList<>();
                words.add(word);
                map.put(newWord, words);
            }
        }

        // print all the values where size is > 1
        // If you want to print non-anagrams,
        // just print the values having size = 1
        for (String s : map.keySet()) {
            List<String> values = map.get(s);
            if (values.size() > 1) {
                System.out.print(values);
            }
        }
    }

    public static void main(String[] args)
    {

        // Driver program
```

```
}
```

Python3

```python
from collections import defaultdict


def printAnagramsTogether(words):
    groupedWords = defaultdict(list)

    # Put all anagram words together in a dictionary
    # where key is sorted word
    for word in words:
        groupedWords["".join(sorted(word))].append(word)

    # Print all anagrams together
    for group in groupedWords.values():
        print(" ".join(group))


if __name__ == "__main__":
    arr = ["cat", "dog", "tac", "god", "act"]
    printAnagramsTogether(arr)
```

C#

```csharp
// C# program to print anagrams
// together using dictionary
using System;
using System.Collections.Generic;

class GFG {
    private static void printAnagrams(String[] arr)
    {
        Dictionary<String, List<String> > map
            = new Dictionary<String, List<String> >();

        // loop over all words
```

```
        char[] letters = word.ToCharArray();
        Array.Sort(letters);
        String newWord = new String(letters);

        // calculate hashcode of string
        // after sorting
        if (map.ContainsKey(newWord)) {
            map[newWord].Add(word);
        }
        else {

            // This is the first time we are
            // adding a word for a specific
            // hashcode
            List<String> words = new List<String>();
            words.Add(word);
            map.Add(newWord, words);
        }
    }

    // print all the values where size is > 1
    // If you want to print non-anagrams,
    // just print the values having size = 1
    List<String> value = new List<String>();
    foreach(KeyValuePair<String, List<String> > entry in
                map)
    {
        value.Add(entry.Key);
    }
    int k = 0;
    foreach(KeyValuePair<String, List<String> > entry in
                map)
    {
        List<String> values = map[value[k++]];
        if (values.Count > 1) {
            Console.Write("[");
            int len = 1;
            foreach(String s in values)
            {
                Console.Write(s);
                if (len++ < values.Count)
```

```
        }
    }

    // Driver Code
    public static void Main(String[] args)
    {
        String[] arr
            = { "cat", "dog", "tac", "god", "act" };
        printAnagrams(arr);
    }
}

// This code is contributed by Princi Singh
```

Output

```
[dog, god][cat, tac, act]
```

Time Complexity : O(M+N).

Auxiliary space: O(M x N).

Output :

```
[cat, tac, act][dog, god]
```

HashMap with O(NM) Solution

In the previous approach, we were sorting every string in order to maintain a similar key, but that cost extra time in this approach will take the advantage of another hashmap to maintain the frequency of the characters which will generate the same hash function for different string having same frequency of characters.

Here, we will take HashMap<HashMap, ArrayList>, the inner hashmap will count the frequency of the characters of each string and the outer HashMap will check whether that hashmap is present or not if present

C++

```
// C++ code to print all anagrams together
#include <bits/stdc++.h>
using namespace std;

void solver(vector<string> my_list)
{

    // Inner hashmap counts frequency
    // of characters in a string.
    // Outer hashmap for if same
    // frequency characters are present in
    // in a string then it will add it to
    // the vector.
    map<map<char, int>, vector<string>> my_map;

    // Loop over all words
    for(string str : my_list)
    {

        // Counting the frequency of the
        // characters present in a string
        map<char, int> temp_map;
        vector<string> temp_my_list;
        for(int i = 0; i < str.length(); ++i)
        {
            ++temp_map[str[i]];
        }

        // If the same frequency of characters
        // are already present then add that
        // string into that arraylist otherwise
        // created a new arraylist and add that
        // string
        auto it = my_map.find(temp_map);
        if (it != my_map.end())
        {
            it->second.push_back(str);
        }
```

```
        }
    }

    // Stores the result in a vector
    vector<vector<string>> result;

    for(auto it = my_map.begin();
             it != my_map.end(); ++it)
    {
        result.push_back(it->second);
    }

    for(int i = 0; i < result.size(); ++i)
    {
          cout << "[";
        for(int j = 0; j < result[i].size(); ++j)
        {
            cout << result[i][j] << ", ";
        }
          cout << "]";
    }
}

// Driver code
int main()
{
    vector<string> my_list = { "cat", "dog", "ogd",
                               "god", "atc" };
    solver(my_list);
    return 0;
}

// This code is contributed by
// Apurba Kumar Gorai(coolapurba05)
```

Java

```
// Java code tp print all anagrams together
import java.util.ArrayList;
```

```
private static ArrayList<ArrayList<String> >
solver(
    ArrayList<String> list)
{

    // Inner hashmap counts frequency
    // of characters in a string.
    // Outer hashmap for if same
    // frequency characters are present in
    // in a string then it will add it to
    // the arraylist.
    HashMap<HashMap<Character, Integer>,
            ArrayList<String> >
        map = new HashMap<HashMap<Character, Integer>,
                          ArrayList<String> >();
    for (String str : list) {
        HashMap<Character, Integer>
            tempMap = new HashMap<Character, Integer>();

        // Counting the frequency of the
        // characters present in a string
        for (int i = 0; i < str.length(); i++) {
            if (tempMap.containsKey(str.charAt(i))) {
                int x = tempMap.get(str.charAt(i));
                tempMap.put(str.charAt(i), ++x);
            }
            else {
                tempMap.put(str.charAt(i), 1);
            }
        }

        // If the same frequency of characters
        // are already present then add that
        // string into that arraylist otherwise
        // created a new arraylist and add that string
        if (map.containsKey(tempMap))
            map.get(tempMap).add(str);
        else {
            ArrayList<String>
                tempList = new ArrayList<String>();
            tempList.add(str);
```

```
        // Stores the result in a arraylist
        ArrayList<ArrayList<String> >
            result = new ArrayList<>();
        for (HashMap<Character, Integer>
                 temp : map.keySet())
            result.add(map.get(temp));
        return result;
    }

    // Drivers Method
    public static void main(String[] args)
    {
        ArrayList<String> list = new ArrayList<>();
        list.add("cat");
        list.add("dog");
        list.add("ogd");
        list.add("god");
        list.add("atc");

        System.out.println(solver(list));
    }
}

// This code is contributed by Arijit Basu(ArijitXfx)
```

Python3

```
# Python code to print all anagrams together
from collections import Counter, defaultdict
user_input = ["cat", "dog", "tac", "edoc", "god", "tacact",
              "act", "code", "deno", "node", "ocde", "done", "catcat"]


def solve(words: list) -> list:
    # defaultdict will create a new list if the key is not found in the d
    m = defaultdict(list)

    # loop over all the words
    for word in words:
        # Counter('cat') :
```

```
        #    frozenset takes an iterable object as input and makes them i
        #    So that hash(frozenset(Counter('cat'))) is equal to
        #   hash of other 'cat' anagrams
        #    >>> frozenset({('c', 1), ('a', 1), ('t', 1)})
        m[frozenset(dict(Counter(word)).items())].append(word)
    return [v for k, v in m.items()]


print(solve(user_input))

# This code is contributed by
# Rohan Kumar(@r0hnx)
```

Output

```
[cat, atc, ][dog, ogd, god, ]
```

Time Complexity: Let there be N-words and each word has a maximum of M characters. The upper bound is O(NM).
Space Complexity: Let there be N-words and each word has maximum M characters, therefore max. storage space for each word with at max. M characters will be O(M), therefore for max N-words, it will be O(N*M). Therefore, the upper bound is O(NM).

This article is contributed by **Aarti_Rathi** and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to **review-team@geeksforgeeks.org**. See your article appearing on the GeeksforGeeks main page and help other Geeks.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Given a sequence of words, print all anagrams together | Set 2
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Previous

Word Break Problem | (Trie solution)

Next

Word Wrap problem (Space optimized solution)

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Given a sequence of words, print all anagrams together | Set 2**
25, Oct 12

02 **Given a sequence of words, print all anagrams together using STL**
12, Jul 17

03 **Print anagrams together in Python using List and Dictionary**
07, Nov 17

04 **Print all pairs of anagrams in a given array of strings**
01, Aug 14

05 **Largest number from the longest set of anagrams possible from all perfect squares of length K**
03, Sep 20

06 **Check if the given string of words can be formed from words present in the dictionary**
04, Jul 18

07 **Print all valid words from given dictionary that are possible using Characters of given Array**
21, Dec 21

08 **Find all substrings that are anagrams of another substring of the string S**
13, Apr 21

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : subharaj01, rituraj_jain, rathbhupendra, Akanksha_Rai, princi singh, Arijit Basu 1, coolapurba05, ankit19146, gabaa406, r0hnx, sooda367, simmytarika5, arorakashish0911, RishMeh19, surinderdawra388, rupeshsk30, sumitgumber28, akashish__, _shinchancode

Article Tags : Amazon, anagram, Morgan Stanley, Snapdeal, Hash, Strings

Practice Tags : Morgan Stanley, Amazon, Snapdeal, Hash, Strings, anagram

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

NodeJS