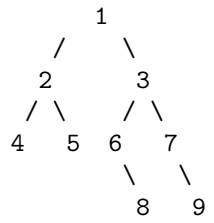
Contents

Chapter 1

Print a Binary Tree in Vertical Order | Set 2 (Hashmap based Method)

Given a binary tree, print it vertically. The following example illustrates vertical order traversal.




```
The output of print this tree vertically will be:
4
2
1 5 6
3 8
7
9
```

We strongly recommend to minimize the browser and try this yourself first.

We have discussed a $O(n^2)$ solution in the previous post. In this post, an efficient solution based on hash map is discussed. We need to check the Horizontal Distances from root for all nodes. If two nodes have the same Horizontal Distance (HD), then they are on same vertical line. The idea of HD is simple. HD for root is 0, a right edge (edge connecting to right subtree) is considered as +1 horizontal distance and a left edge is considered as -1 horizontal distance. For example, in the above tree, HD for Node 4 is at -2, HD for Node 2 is -1, HD for 5 and 6 is 0 and HD for node 7 is +2.
We can do inorder traversal of the given Binary Tree. While traversing the tree, we can recursively calculate HDs. We initially pass the horizontal distance as 0 for root. For left subtree, we pass the Horizontal Distance as Horizontal distance of root minus 1. For right subtree, we pass the Horizontal Distance as Horizontal Distance of root plus 1. For every HD value, we maintain a list of nodes in a hasp map. Whenever we see a node in traversal, we go to the hash map entry and add the node to the hash map using HD as a key in map.

Following is C++ implementation of the above method. Thanks to Chirag for providing the below C++ implementation.

```
// C++ program for printing vertical order of a given binary tree
#include <iostream>
#include <vector>
#include <map>
using namespace std;

// Structure for a binary tree node
struct Node
{
    int key;
    Node *left, *right;
};

// A utility function to create a new node
struct Node* newNode(int key)
{
    struct Node* node = new Node;
    node->key = key;
    node->left = node->right = NULL;
    return node;
}

// Utility function to store vertical order in map 'm'
// 'hd' is horigontal distance of current node from root.
// 'hd' is initally passed as 0
void getVerticalOrder(Node* root, int hd, map<int, vector<int>> &m)
{
    // Base case
    if (root == NULL)
        return;

    // Store current node in map 'm'
    m[hd].push_back(root->key);

    // Store nodes in left subtree
    getVerticalOrder(root->left, hd-1, m);

    // Store nodes in right subtree
    getVerticalOrder(root->right, hd+1, m);
}

// The main function to print vertical oder of a binary tree
// with given root
void printVerticalOrder(Node* root)
{
    // Create a map and store vertical oder in map using
    // function getVerticalOrder()
    map < int,vector<int> > m;
    int hd = 0;
    getVerticalOrder(root, hd,m);

    // Traverse the map and print nodes at every horigontal
    // distance (hd)
```

```
    map< int,vector<int> > :: iterator it;
    for (it=m.begin(); it!=m.end(); it++)
    {
        for (int i=0; i<it->second.size(); ++i)
            cout << it->second[i] << " ";
        cout << endl;
    }
}

// Driver program to test above functions
int main()
{
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);
    root->right->right->right = newNode(9);
    cout << "Vertical order traversal is \n";
    printVerticalOrder(root);
    return 0;
}
```

Output:

```
 Vertical order traversal is
4
2
1 5 6
3 8
7
9
```

Time Complexity of hashing based solution can be considered as O(n) under the assumption that we have good hashing function that allows insertion and retrieval operations in O(1) time. In the above C++ implementation, map of STL is used. map in STL is typically implemented using a Self-Balancing Binary Search Tree where all operations take O(Logn) time. Therefore time complexity of above implementation is O(nLogn).

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

http://www.geeksforgeeks.org/print-binary-tree-vertical-order-set-2/

Category: Trees

Chapter 2

Find whether an array is subset of another array | Added Method 3

Given two arrays: arr1[0..m-1] and arr2[0..n-1]. Find whether arr2[] is a subset of arr1[] or not. Both the arrays are not in sorted order. It may be assumed that elements in both array are distinct.

Examples:
Input: arr1[] = {11, 1, 13, 21, 3, 7}, arr2[] = {11, 3, 7, 1}
Output: arr2[] is a subset of arr1[]

Input: arr1[] = {1, 2, 3, 4, 5, 6}, arr2[] = {1, 2, 4}
Output: arr2[] is a subset of arr1[]

Input: arr1[] = {10, 5, 2, 23, 19}, arr2[] = {19, 5, 3}
Output: arr2[] is not a subset of arr1[]

Method 1 (Simple)
Use two loops: The outer loop picks all the elements of arr2[] one by one. The inner loop linearly searches for the element picked by outer loop. If all elements are found then return 1, else return 0.

```
#include<stdio.h>

/* Return 1 if arr2[] is a subset of arr1[] */
bool isSubset(int arr1[], int arr2[], int m, int n)
{
    int i = 0;
    int j = 0;
    for (i=0; i<n; i++)
    {
        for (j = 0; j<m; j++)
        {
           if(arr2[i] == arr1[j])
              break;
        }

        /* If the above inner loop was not broken at all then
           arr2[i] is not present in arr1[] */
        if (j == m)
           return 0;
```

```
    }

    /* If we reach here then all elements of arr2[]
      are present in arr1[] */
    return 1;
}

int main()
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);

    if(isSubset(arr1, arr2, m, n))
      printf("arr2[] is subset of arr1[] ");
    else
      printf("arr2[] is not a subset of arr1[]");

    getchar();
    return 0;
}
```

Time Complexity: O(m*n)

Method 2 (Use Sorting and Binary Search)

```
1) Sort arr1[] O(mLogm)
2) For each element of arr2[], do binary search for it in sorted arr1[].
         a) If the element is not found then return 0.
3) If all elements are present then return 1.
```

```
#include<stdio.h>

/* Fucntion prototypes */
void quickSort(int *arr, int si, int ei);
int binarySearch(int arr[], int low, int high, int x);

/* Return 1 if arr2[] is a subset of arr1[] */
bool isSubset(int arr1[], int arr2[], int m, int n)
{
    int i = 0;

    quickSort(arr1, 0, m-1);
    for (i=0; i<n; i++)
    {
        if (binarySearch(arr1, 0, m-1, arr2[i]) == -1)
           return 0;
    }
```

```
    /* If we reach here then all elements of arr2[]
      are present in arr1[] */
    return 1;
}

/* FOLLOWING FUNCTIONS ARE ONLY FOR SEARCHING AND SORTING PURPOSE */
/* Standard Binary Search function*/
int binarySearch(int arr[], int low, int high, int x)
{
  if(high >= low)
  {
    int mid = (low + high)/2;  /*low + (high - low)/2;*/

    /* Check if arr[mid] is the first occurrence of x.
        arr[mid] is first occurrence if x is one of the following
        is true:
        (i)  mid == 0 and arr[mid] == x
        (ii) arr[mid-1] < x and arr[mid] == x
     */
    if(( mid == 0 || x > arr[mid-1]) && (arr[mid] == x))
      return mid;
    else if(x > arr[mid])
      return binarySearch(arr, (mid + 1), high, x);
    else
      return binarySearch(arr, low, (mid -1), x);
  }
 return -1;
}

void exchange(int *a, int *b)
{
    int temp;
    temp = *a;
    *a   = *b;
    *b   = temp;
}

int partition(int A[], int si, int ei)
{
    int x = A[ei];
    int i = (si - 1);
    int j;

    for (j = si; j <= ei - 1; j++)
    {
        if(A[j] <= x)
        {
            i++;
            exchange(&A[i], &A[j]);
        }
    }
    exchange (&A[i + 1], &A[ei]);
    return (i + 1);
}
```

```
/* Implementation of Quick Sort
A[] --> Array to be sorted
si  --> Starting index
ei  --> Ending index
*/
void quickSort(int A[], int si, int ei)
{
    int pi;    /* Partitioning index */
    if(si < ei)
    {
        pi = partition(A, si, ei);
        quickSort(A, si, pi - 1);
        quickSort(A, pi + 1, ei);
    }
}

/*Driver program to test above functions */
int main()
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);

    if(isSubset(arr1, arr2, m, n))
      printf("arr2[] is subset of arr1[] ");
    else
      printf("arr2[] is not a subset of arr1[] ");

    getchar();
    return 0;
}
```

Time Complexity: O(mLogm + nLogm). Please note that this will be the complexity if an mLogm algorithm is used for sorting which is not the case in above code. In above code Quick Sort is sued and worst case time complexity of Quick Sort is O(m^2)

Method 3 (Use Sorting and Merging)
1) Sort both arrays: arr1[] and arr2[] O(mLogm + nLogn)
2) Use Merge type of process to see if all elements of sorted arr2[] are present in sorted arr1[].

Thanks to Parthsarthifor suggesting this method.

```
/* Return 1 if arr2[] is a subset of arr1[] */
bool isSubset(int arr1[], int arr2[], int m, int n)
{
    int i = 0, j = 0;

    if(m < n)
       return 0;

    quickSort(arr1, 0, m-1);
```

```
    quickSort(arr2, 0, n-1);
    while( i < n && j < m )
    {
        if( arr1[j] <arr2[i] )
            j++;
        else if( arr1[j] == arr2[i] )
        {
            j++;
            i++;
        }
        else if( arr1[j] > arr2[i] )
            return 0;
    }

    if( i < n )
        return 0;
    else
        return 1;
}
```

Time Complexity: O(mLogm + nLogn) which is better than method 2. Please note that this will be the complexity if an nLogn algorithm is used for sorting both arrays which is not the case in above code. In above code Quick Sort is sued and worst case time complexity of Quick Sort is O(n^2)

Method 4 (Use Hashing)
1) Create a Hash Table for all the elements of arr1[].
2) Traverse arr2[] and search for each element of arr2[] in the Hash Table. If element is not found then return 0.
3) If all elements are found then return 1.

Note that method 1, method 2 and method 4 don't handle the cases when we have duplicates in arr2[]. For example, {1, 4, 4, 2} is not a subset of {1, 4, 2}, but these methods will print it as a subset.

Source: http://geeksforgeeks.org/forum/topic/if-an-array-is-subset-of-another

Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.

Source

http://www.geeksforgeeks.org/find-whether-an-array-is-subset-of-another-array-set-1/

Category: Arrays

Post navigation

← A Boolean Array Puzzle Dynamic Programming | Set 4 (Longest Common Subsequence) →

Chapter 3

Union and Intersection of two Linked Lists

Given two Linked Lists, create union and intersection lists that contain union and intersection of the elements present in the given lists. Order of elments in output lists doesn't matter.

Example:

```
Input:
   List1: 10->15->4->20
   lsit2:  8->4->2->10
Output:
   Intersection List: 4->10
   Union List: 2->8->20->4->15->10
```

Method 1 (Simple)
Following are simple algorithms to get union and intersection lists respectively.

Intersection (list1, list2)
Initialize result list as NULL. Traverse list1 and look for its each element in list2, if the element is present in list2, then add the element to result.

Union *(list1, list2):*
Initialize result list as NULL. Traverse list1 and add all of its elements to the result.
Traverse list2. If an element of list2 is already present in result then do not insert it to result, otherwise insert.

This method assumes that there are no duplicates in the given lists.

Thanks to Shekhufor suggesting this method. Following is C implementation of this method.

```
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
```

```
    int data;
    struct node* next;
};

/* A utility function to insert a node at the begining of a linked list*/
void push (struct node** head_ref, int new_data);

/* A utilty function to chec if given data is present in a list */
bool isPresent (struct node *head, int data);

/* Function to get union of two linked lists head1 and head2 */
struct node *getUnion (struct node *head1, struct node *head2)
{
    struct node *result = NULL;
    struct node *t1 = head1, *t2 = head2;

    // Insert all elements of list1 to the result list
    while (t1 != NULL)
    {
        push(&result, t1->data);
        t1 = t1->next;
    }

    // Insert those elements of list2 which are not present in result list
    while (t2 != NULL)
    {
        if (!isPresent(result, t2->data))
            push(&result, t2->data);
        t2 = t2->next;
    }

    return result;
}

/* Function to get intersection of two linked lists head1 and head2 */
struct node *getIntersection (struct node *head1, struct node *head2)
{
    struct node *result = NULL;
    struct node *t1 = head1;

    // Traverse list1 and search each element of it in list2. If the element
    // is present in list 2, then insert the element to result
    while (t1 != NULL)
    {
        if (isPresent(head2, t1->data))
            push (&result, t1->data);
        t1 = t1->next;
    }

    return result;
}

/* A utility function to insert a node at the begining of a linked list*/
void push (struct node** head_ref, int new_data)
```

```
{
    /* allocate node */
    struct node* new_node =
        (struct node*) malloc(sizeof(struct node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* A utility function to print a linked list*/
void printList (struct node *node)
{
    while (node != NULL)
    {
        printf ("%d ", node->data);
        node = node->next;
    }
}

/* A utilty function that returns true if data is present in linked list
else return false */
bool isPresent (struct node *head, int data)
{
    struct node *t = head;
    while (t != NULL)
    {
        if (t->data == data)
            return 1;
        t = t->next;
    }
    return 0;
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct node* head1 = NULL;
    struct node* head2 = NULL;
    struct node* intersecn = NULL;
    struct node* unin = NULL;

    /*create a linked lits 10->15->5->20 */
    push (&head1, 20);
    push (&head1, 4);
    push (&head1, 15);
    push (&head1, 10);
```

```
    /*create a linked lits 8->4->2->10 */
    push (&head2, 10);
    push (&head2, 2);
    push (&head2, 4);
    push (&head2, 8);

    intersecn = getIntersection (head1, head2);
    unin = getUnion (head1, head2);

    printf ("\n First list is \n");
    printList (head1);

    printf ("\n Second list is \n");
    printList (head2);

    printf ("\n Intersection list is \n");
    printList (intersecn);

    printf ("\n Union list is \n");
    printList (unin);

    return 0;
}
```

Output:

```
 First list is
10 15 4 20
 Second list is
8 4 2 10
 Intersection list is
4 10
 Union list is
2 8 20 4 15 10
```

Time Complexity: O(mn) for both union and intersection operations. Here m is the number of elements in first list and n is the number of elements in second list.

Method 2 (Use Merge Sort)
In this method, algorithms for Union and Intersection are very similar. First we sort the given lists, then we traverse the sorted lists to get union and intersection.
Following are the steps to be followed to get union and intersection lists.

1) Sort the first Linked List using merge sort. This step takes O(mLogm) time. Refer this post for details of this step.
2) Sort the second Linked List using merge sort. This step takes O(nLogn) time. Refer this post for details of this step.
3) Linearly scan both sorted lists to get the union and intersection. This step takes O(m + n) time. This step can be implemented using the same algorithm as sorted arrays algorithm discussed here.

Time complexity of this method is O(mLogm + nLogn) which is better than method 1's time complexity.

Method 3 (Use Hashing)
Union (list1, list2)
Initialize the result list as NULL and create an empty hash table. Traverse both lists one by one, for each element being visited, look the element in hash table. If the element is not present, then insert the element to result list. If the element is present, then ignore it.

Intersection (list1, list2)
Initialize the result list as NULL and create an empty hash table. Traverse list1. For each element being visited in list1, insert the element in hash table. Traverse list2, for each element being visited in list2, look the element in hash table. If the element is present, then insert the element to result list. If the element is not present, then ignore it.

Both of the above methods assume that there are no duplicates.

Time complexity of this method depends on the hashing technique used and the distribution of elements in input lists. In practical, this approach may turn out to be better than above 2 methods.

Source: http://geeksforgeeks.org/forum/topic/union-intersection-of-unsorted-lists

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

http://www.geeksforgeeks.org/union-and-intersection-of-two-linked-lists/

Chapter 4

Find a pair with given sum

Write a C program that, given an array A[] of n numbers and another number x, determines whether or not there exist two elements in S whose sum is exactly x.

METHOD 1 (Use Sorting)

Algorithm:

```
hasArrayTwoCandidates (A[], ar_size, sum)
1) Sort the array in non-decreasing order.
2) Initialize two index variables to find the candidate
   elements in the sorted array.
       (a) Initialize first to the leftmost index: l = 0
       (b) Initialize second  the rightmost index:  r = ar_size-1
3) Loop while l
Time Complexity: Depends on what sorting algorithm we use.  If we use Merge Sort or Heap Sort then (-)(nlogn)

Auxiliary Space : Again, depends on sorting algorithm. For example auxiliary space is O(n) for merge sort and
Example:

Let Array be  {1, 4, 45, 6, 10, -8} and sum to find be 16
Sort the array

A = {-8, 1, 4, 6, 10, 45}
Initialize l = 0,  r = 5

A[l] + A[r] ( -8 + 45)  >  16    => decrement r.  Now r = 10

A[l] + A[r] ( -8 + 10)    increment l.  Now l = 1

A[l] + A[r] ( 1 + 10)     increment l.  Now l = 2

A[l] + A[r] ( 4 + 10)     increment l.  Now l = 3

A[l] + A[r] ( 6 + 10)  == 16    =>  Found candidates (return 1)
Note: If there are more than one pair having the given sum then this algorithm reports only one. Can be easily
Implementation:
```

C

```
# include <stdio.h>
# define bool int

void quickSort(int *, int, int);

bool hasArrayTwoCandidates(int A[], int arr_size, int sum)
{
    int l, r;

    /* Sort the elements */
    quickSort(A, 0, arr_size-1);

    /* Now look for the two candidates in the sorted
       array*/
    l = 0;
    r = arr_size-1;
    while (l < r)
    {
         if(A[l] + A[r] == sum)
              return 1;
         else if(A[l] + A[r] < sum)
              l++;
         else // A[i] + A[j] > sum
              r--;
    }
    return 0;
}

/* Driver program to test above function */
int main()
{
    int A[] = {1, 4, 45, 6, 10, -8};
    int n = 16;
    int arr_size = 6;

    if( hasArrayTwoCandidates(A, arr_size, n))
        printf("Array has two elements with sum 16");
    else
        printf("Array doesn't have two elements with sum 16 ");

    getchar();
    return 0;
}

/* FOLLOWING FUNCTIONS ARE ONLY FOR SORTING
    PURPOSE */
void exchange(int *a, int *b)
{
    int temp;
    temp = *a;
```

```
    *a   = *b;
    *b   = temp;
}

int partition(int A[], int si, int ei)
{
    int x = A[ei];
    int i = (si - 1);
    int j;

    for (j = si; j <= ei - 1; j++)
    {
        if(A[j] <= x)
        {
            i++;
            exchange(&A[i], &A[j]);
        }
    }
    exchange (&A[i + 1], &A[ei]);
    return (i + 1);
}

/* Implementation of Quick Sort
A[] --> Array to be sorted
si  --> Starting index
ei  --> Ending index
*/
void quickSort(int A[], int si, int ei)
{
    int pi;    /* Partitioning index */
    if(si < ei)
    {
        pi = partition(A, si, ei);
        quickSort(A, si, pi - 1);
        quickSort(A, pi + 1, ei);
    }
}
```

Python

```
# Python program to check for the sum condition to be satisified
def hasArrayTwoCandidates(A,arr_size,sum):

    # sort the array
    quickSort(A,0,arr_size-1)
    l = 0
    r = arr_size-1

    # traverse the array for the two elements
    while l<r:
        if (A[l] + A[r] == sum):
            return 1
```

```
        elif (A[l] + A[r] < sum):
            l += 1
        else:
            r -= 1
    return 0

# Implementation of Quick Sort
# A[] --> Array to be sorted
# si  --> Starting index
# ei  --> Ending index
def quickSort(A, si, ei):
    if si < ei:
        pi=partition(A,si,ei)
        quickSort(A,si,pi-1)
        quickSort(A,pi+1,ei)

# Utility function for partitioning the array(used in quick sort)
def partition(A, si, ei):
    x = A[ei]
    i = (si-1)
    for j in range(si,ei):
        if A[j] <= x:
            i += 1

            # This operation is used to swap two variables is python
            A[i], A[j] = A[j], A[i]

        A[i+1], A[ei] = A[ei], A[i+1]

    return i+1


# Driver program to test the functions
A = [1,4,45,6,10,-8]
n = 16
if (hasArrayTwoCandidates(A, len(A), n)):
    print("Array has two elements with the given sum")
else:
    print("Array doesn't have two elements with the given sum")

## This code is contributed by __Devesh Agrawal__
```

Output:

```
Array has two elements with the given sum
```

METHOD 2 (Use Hash Map)
Thanks to Bindu for suggesting this method and thanks to Shekhufor providing code.
This method works in O(n) time if range of numbers is known.
Let sum be the given sum and A[] be the array in which we need to find pair.

```
1) Initialize Binary Hash Map M[] = {0, 0, …}
2) Do following for each element A[i] in A[]
   (a)  If M[x  - A[i]] is set then print the pair (A[i], x - A[i])
   (b)  Set M[A[i]]
```

Implementation:

C/C++

```
#include <stdio.h>
#define MAX 100000

void printPairs(int arr[], int arr_size, int sum)
{
  int i, temp;
  bool binMap[MAX] = {0}; /*initialize hash map as 0*/

  for (i = 0; i < arr_size; i++)
  {
      temp = sum - arr[i];
      if (temp >= 0 && binMap[temp] == 1)
         printf("Pair with given sum %d is (%d, %d) \n",
                 sum, arr[i], temp);
      binMap[arr[i]] = 1;
  }
}

/* Driver program to test above function */
int main()
{
    int A[] = {1, 4, 45, 6, 10, 8};
    int n = 16;
    int arr_size = sizeof(A)/sizeof(A[0]);

    printPairs(A, arr_size, n);

    getchar();
    return 0;
}
```

Java

```
// Java implementation using Hashing
import java.io.*;

class PairSum
{
    private static final int MAX = 100000; // Max size of Hashmap
```

```
    static void printpairs(int arr[],int sum)
    {
        //Declares and initializes the whole array as false
        boolean[] binmap = new boolean[MAX];

        for (int i=0; i<arr.length; ++i)
        {
            int temp = sum-arr[i];

            //checking for condition
            if (temp>=0 && binmap[temp])
            {
                System.out.println("Pair with given sum " +
                                   sum + " is (" + arr[i] +
                                   ", "+temp+")");
            }
            binmap[arr[i]] = true;
        }
    }

    // Main to test the above function
    public static void main (String[] args)
    {
        int A[] = {1, 4, 45, 6, 10, 8};
        int n = 16;
        printpairs(A,  n);
    }
}

// This article is contributed by Aakash Hasija
```

Python

```
# Python program to find if there are two elements wtih given sum
CONST_MAX = 100000

# function to check for the given sum in the array
def printPairs(arr, arr_size, sum):

    # initialize hash map as 0
    binmap = [0]*CONST_MAX

    for i in range(0,arr_size):
        temp = sum-arr[i]
        if (temp>=0 and binmap[temp]==1):
            print "Pair with the given sum is", arr[i], "and", temp
        binmap[arr[i]]=1

# driver program to check the above function
A = [1,4,45,6,10,-8]
n = 16
printPairs(A, len(A), n)
```

```
# This code is contributed by __Devesh Agrawal__
```

Time Complexity: O(n)
Output:

```
Pair with given sum 16 is (10, 6)
```

Auxiliary Space: O(R) where R is range of integers.

If range of numbers include negative numbers then also it works. All we have to do for negative numbers is to make everything positive by adding the absolute value of smallest negative integer to all numbers.

Please write comments if you find any of the above codes/algorithms incorrect, or find other ways to solve the same problem.

Source

http://www.geeksforgeeks.org/write-a-c-program-that-given-a-set-a-of-n-numbers-and-another-number-x-determines-whetl

Category: Arrays Tags: Hashing

Post navigation

← Next Power of 2 Majority Element →

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

Chapter 5

Check if a given array contains duplicate elements within k distance from each other

Given an unsorted array that may contain duplicates. Also given a number k which is smaller than size of array. Write a function that returns true if array contains duplicates within k distance.

Examples:

```
Input: k = 3, arr[] = {1, 2, 3, 4, 1, 2, 3, 4}
Output: false
All duplicates are more than k distance away.

Input: k = 3, arr[] = {1, 2, 3, 1, 4, 5}
Output: true
1 is repeated at distance 3.

Input: k = 3, arr[] = {1, 2, 3, 4, 5}
Output: false

Input: k = 3, arr[] = {1, 2, 3, 4, 4}
Output: true
```

A **Simple Solution** is to run two loops. The outer loop picks every element 'arr[i]' as a starting element, the inner loop compares all elements which are within k distance of 'arr[i]'. The time complexity of this solution is O(kn).

We can solve this problem **in Θ(n) time using Hashing.** The idea is to one by add elements to hash. We also remove elements which are at more than k distance from current element. Following is detailed algorithm.

1) Create an empty hashtable.
2) Traverse all elements from left from right. Let the current element be 'arr[i]'
….a) If current element 'arr[i]' is present in hashtable, then return true.
….b) Else add arr[i] to hash and remove arr[i-k] from hash if i is greater than or equal to k

```
/* Java program to Check if a given array contains duplicate
   elements within k distance from each other */
import java.util.*;

class Main
{
    static boolean checkDuplicatesWithinK(int arr[], int k)
    {
        // Creates an empty hashset
        HashSet<Integer> set = new HashSet<>();

        // Traverse the input array
        for (int i=0; i<arr.length; i++)
        {
            // If already present n hash, then we found
            // a duplicate within k distance
            if (set.contains(arr[i]))
               return true;

            // Add this item to hashset
            set.add(arr[i]);

            // Remove the k+1 distant item
            if (i >= k)
              set.remove(arr[i-k]);
        }
        return false;
    }

    // Driver method to test above method
    public static void main (String[] args)
    {
        int arr[] = {10, 5, 3, 4, 3, 5, 6};
        if (checkDuplicatesWithinK(arr, 3))
           System.out.println("Yes");
        else
           System.out.println("No");
    }
}
```

Output:

```
Yes
```

This article is contributed by **Anuj**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

http://www.geeksforgeeks.org/check-given-array-contains-duplicate-elements-within-k-distance/

Category: Arrays Tags: Hashing

Post navigation

← Zoho Interview | Set 4 Amazon Interview Experience | Set 163 (For SDE II) →

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

Chapter 6

Find Itinerary from a given list of tickets

Given a list of tickets, find itinerary in order using the given list.

Example:

```
Input:
"Chennai" -> "Banglore"
"Bombay" -> "Delhi"
"Goa"    -> "Chennai"
"Delhi"  -> "Goa"

Output:
Bombay->Delhi, Delhi->Goa, Goa->Chennai, Chennai->Banglore,
```

It may be assumed that the input list of tickets is not cyclic and there is one ticket from every city except final destination.

One Solution is to build a graph and do Topological Sorting of the graph. Time complexity of this solution is O(n).

We can also use hashingto avoid building a graph. The idea is to first find the starting point. A starting point would never be on 'to' side of a ticket. Once we find the starting point, we can simply traverse the given map to print itinerary in order. Following are steps.

```
1) Create a HashMap of given pair of tickets.  Let the created
   HashMap be 'dataset'. Every entry of 'dataset' is of the form
   "from->to" like "Chennai" -> "Banglore"

2) Find the starting point of itinerary.
     a) Create a reverse HashMap.  Let the reverse be 'reverseMap'
        Entries of 'reverseMap' are of the form "to->form".
        Following is 'reverseMap' for above example.
        "Banglore"-> "Chennai"
        "Delhi"   -> "Bombay"
        "Chennai" -> "Goa"
```

```
        "Goa"     ->  "Delhi"

     b) Traverse 'dataset'.  For every key of dataset, check if it
        is there in 'reverseMap'.  If a key is not present, then we
        found the starting point. In the above example, "Bombay" is
        starting point.

3) Start from above found starting point and traverse the 'dataset'
   to print itinerary.
```

All of the above steps require O(n) time so overall time complexity is O(n).

Below is Java implementation of above idea.

```
// Java program to print itinerary in order
import java.util.HashMap;
import java.util.Map;

public class printItinerary
{
    // Driver function
    public static void main(String[] args)
    {
        Map<String, String> dataSet = new HashMap<String, String>();
        dataSet.put("Chennai", "Banglore");
        dataSet.put("Bombay", "Delhi");
        dataSet.put("Goa", "Chennai");
        dataSet.put("Delhi", "Goa");

        printResult(dataSet);
    }

    // This function populates 'result' for given input 'dataset'
    private static void printResult(Map<String, String> dataSet)
    {
        // To store reverse of given map
        Map<String, String> reverseMap = new HashMap<String, String>();

        // To fill reverse map, iterate through the given map
        for (Map.Entry<String,String> entry: dataSet.entrySet())
            reverseMap.put(entry.getValue(), entry.getKey());

        // Find the starting point of itinerary
        String start = null;
        for (Map.Entry<String,String> entry: dataSet.entrySet())
        {
              if (!reverseMap.containsKey(entry.getKey()))
              {
                   start = entry.getKey();
                   break;
              }
        }
```

```
        // If we could not find a starting point, then something wrong
        // with input
        if (start == null)
        {
           System.out.println("Invalid Input");
           return;
        }

        // Once we have starting point, we simple need to go next, next
        // of next using given hash map
        String to = dataSet.get(start);
        while (to != null)
        {
            System.out.print(start +  "->" + to + ", ");
            start = to;
            to = dataSet.get(to);
        }
    }
}
```

Output:

```
Bombay->Delhi, Delhi->Goa, Goa->Chennai, Chennai->Banglore,
```

This article is compiled by **Rahul Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

http://www.geeksforgeeks.org/find-itinerary-from-a-given-list-of-tickets/

Category: Misc Tags: Hashing

Post navigation

← Load Balancing on Servers (Random Algorithm) Flipkart Interview Experience | Set 26 →

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

Chapter 7

Find number of Employees Under every Employee

Given a dictionary that contains mapping of employee and his manager as a number of (employee, manager) pairs like below.

```
{ "A", "C" },
{ "B", "C" },
{ "C", "F" },
{ "D", "E" },
{ "E", "F" },
{ "F", "F" }

In this example C is manager of A,
C is also manager of B, F is manager
of C and so on.
```

Write a function to get no of employees under each manager in the hierarchy not just their direct reports. It may be assumed that an employee directly reports to only one manager. In the above dictionary the root node/ceo is listed as reporting to himself.

Output should be a Dictionary that contains following.

```
A - 0
B - 0
C - 2
D - 0
E - 1
F - 5
```

Source: Microsoft Interview

This question might be solved differently but i followed this and found interesting, so sharing:

```
 1. Create a reverse map with Manager->DirectReportingEmployee
    combination. Off-course employee will be multiple so Value
    in Map is List of Strings.
        "C" --> "A", "B",
        "E" --> "D"
        "F" --> "C", "E", "F"


2. Now use the given employee-manager map to iterate  and at
   the same time use newly reverse map to find the count of
   employees under manager.

   Let the map created in step 2 be 'mngrEmpMap'
   Do following for every employee 'emp'.
     a) If 'emp' is not present in 'mngrEmpMap'
          Count under 'emp' is 0 [Nobody reports to 'emp']
     b) If 'emp' is present in 'mngrEmpMap'
          Use the list of direct reports from map 'mngrEmpMap'
          and recursively calculate number of total employees
          under 'emp'.
```

A trick in step 2.b is to use memorization(Dynamic programming) while finding number of employees under a manager so that we don't need to find number of employees again for any of the employees. In the below code populateResultUtil() is the recursive function that uses memoization to avoid re-computation of same results.

Below is Java implementation of above ides

```
// Java program to find number of persons under every employee
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class NumberEmployeeUnderManager
{
    // A hashmap to store result. It stores count of employees
    // under every employee, the count may by 0 also
    static Map<String,Integer> result =
                            new HashMap<String, Integer>();

    // Driver function
    public static void main(String[] args)
    {
        Map<String, String> dataSet = new HashMap<String, String>();
        dataSet.put("A", "C");
        dataSet.put("B", "C");
        dataSet.put("C", "F");
        dataSet.put("D", "E");
        dataSet.put("E", "F");
        dataSet.put("F", "F");

        populateResult(dataSet);
```

```
    System.out.println("result = " + result);
}

// This function populates 'result' for given input 'dataset'
private static void populateResult(Map<String, String> dataSet)
{
    // To store reverse of original map, each key will have 0
    // to multiple values
    Map<String, List<String>> mngrEmpMap =
                              new HashMap<String, List<String>>();

    // To fill mngrEmpMap, iterate through the given map
    for (Map.Entry<String,String> entry: dataSet.entrySet())
    {
        String emp = entry.getKey();
        String mngr = entry.getValue();
        if (!emp.equals(mngr)) // excluding emp-emp entry
        {
            // Get the previous list of direct reports under
            // current 'mgr' and add the current 'emp' to the list
            List<String> directReportList = mngrEmpMap.get(mngr);

            // If 'emp' is the first employee under 'mgr'
            if (directReportList == null)
                directReportList = new ArrayList<String>();

            directReportList.add(emp);

            // Replace old value for 'mgr' with new
            // directReportList
            mngrEmpMap.put(mngr, directReportList);
        }
    }

    // Now use manager-Emp map built above to populate result
    // with use of populateResultUtil()

    // note- we are iterating over original emp-manager map and
    // will use mngr-emp map in helper to get the count
    for (String mngr: dataSet.keySet())
        populateResultUtil(mngr, mngrEmpMap);
}

// This is a recursive function to fill count for 'mgr' using
// mngrEmpMap.  This function uses memoization to avoid re-
// computations of subproblems.
private static int populateResultUtil(String mngr,
                           Map<String, List<String>> mngrEmpMap)
{
    int count = 0;

    // means employee is not a manager of any other employee
    if (!mngrEmpMap.containsKey(mngr))
    {
```

```
            result.put(mngr, 0);
            return 0;
        }

        // this employee count has already been done by this
        // method, so avoid re-computation
        else if (result.containsKey(mngr))
            count = result.get(mngr);

        else
        {
            List<String> directReportEmpList = mngrEmpMap.get(mngr);
            count = directReportEmpList.size();
            for (String directReportEmp: directReportEmpList)
               count +=  populateResultUtil(directReportEmp, mngrEmpMap);

            result.put(mngr, count);
        }
        return count;
    }
}
```

Output:

```
result = {D=0, E=1, F=5, A=0, B=0, C=2}
```

This article is contributed by **Chandan Prakash**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

http://www.geeksforgeeks.org/find-number-of-employees-under-every-manager/

Category: Misc Tags: Hashing

Post navigation

← Snapdeal Interview Experience | Set 13 (On-Campus for SDET) Amazon Interview Experience | Set 188 (For SDE1) →

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