# Copyright

The content of this book comes from [geeksforgeeks.org](http://geeksforgeeks.org) and it's licensed under [Creative Commons Attribution-NonCommercial-NoDerivs 2.5 India](http://creativecommons.org/licenses/by-nc-nd/2.5/in/deed.en_US)

Made by [Jing](http://www.jing-zhou.me/). 2015.

Updated on August 13, 2015

Head over to [this github repo](https://github.com/gnijuohz/geeksforgeeks-as-books) to report issues or contribute.

# Advantages of BST over Hash Table

[Hash Table](http://geeksquiz.com/hashing-set-1-introduction/) supports following operations in Θ(1) time.  
 1) Search  
 2) Insert  
 3) Delete

The time complexity of above operations in a self-balancing [Binary Search Tree (BST)](http://geeksquiz.com/binary-search-tree-set-1-search-and-insertion/) (like [Red-Black Tree](http://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/), [AVL Tree](http://www.geeksforgeeks.org/avl-tree-set-1-insertion/), [Splay Tree](http://www.geeksforgeeks.org/splay-tree-set-1-insert/), etc) is O(Logn).

So Hash Table seems to beating BST in all common operations. When should we prefer BST over Hash Tables, what are advantages. Following are some important points in favor of BSTs.

1. We can get all keys in sorted order by just doing Inorder Traversal of BST. This is not a natural operation in Hash Tables and requires extra efforts.
2. Doing [order statistics](http://www.geeksforgeeks.org/find-k-th-smallest-element-in-bst-order-statistics-in-bst/), [finding closest lower and greater elements](http://www.geeksforgeeks.org/floor-and-ceil-from-a-bst/), [doing range queries](http://www.geeksforgeeks.org/print-bst-keys-in-the-given-range/) are easy to do with BSTs. Like sorting, these operations are not a natural operation with Hash Tables.
3. BSTs are easy to implement compared to hashing, we can easily implement our own customized BST. To implement Hashing, we generally rely on libraries provided by programming languages.
4. With BSTs, all operations are guaranteed to work in O(Logn) time. But with Hashing, Θ(1) is average time and some particular operations may be costly, especially when table resizing happens.

This article is contributed by **Himanshu Gupta**. 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/advantages-of-bst-over-hash-table/>

Category: [Trees](http://www.geeksforgeeks.org/category/tree/) Tags: [Hashing](http://www.geeksforgeeks.org/tag/hashing/)

# 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](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Hashing](http://www.geeksforgeeks.org/tag/hashing/)

# Clone a Binary Tree with Random Pointers

Given a Binary Tree where every node has following structure.

struct node {   
 int key;   
 struct node \*left,\*right,\*random;  
}

The random pointer points to any random node of the binary tree and can even point to NULL, clone the given binary tree.

**Method 1 (Use Hashing)**  
 The idea is to store mapping from given tree nodes to clone tre node in hashtable. Following are detailed steps.

1) Recursively traverse the given Binary and copy key value, left pointer and right pointer to clone tree. While copying, store the mapping from given tree node to clone tree node in a hashtable. In the following pseudo code, ‘cloneNode’ is currently visited node of clone tree and ‘treeNode’ is currently visited node of given tree.

cloneNode->key = treeNode->key  
 cloneNode->left = treeNode->left  
 cloneNode->right = treeNode->right  
 map[treeNode] = cloneNode

2) Recursively traverse both trees and set random pointers using entries from hash table.

cloneNode->random = map[treeNode->random]

Following is C++ implementation of above idea. The following implementation uses [map](http://www.cplusplus.com/reference/map/map/)from C++ STL. Note that map doesn’t implement hash table, it actually is based on self-balancing binary search tree.

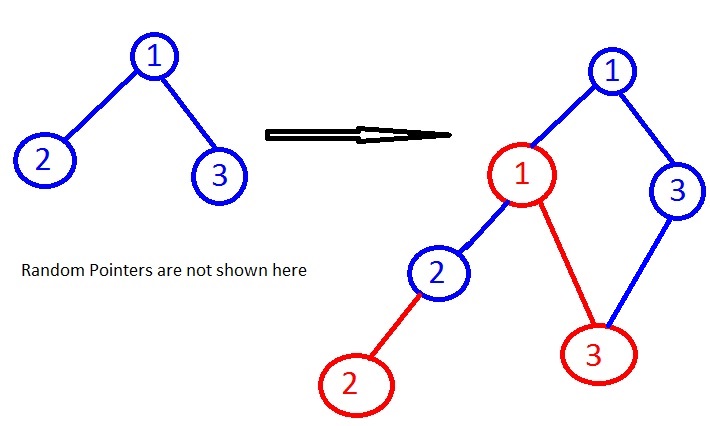
// A hashmap based C++ program to clone a binary tree with random pointers  
#include<iostream>  
#include<map>  
using namespace std;  
  
/\* A binary tree node has data, pointer to left child, a pointer to right  
 child and a pointer to random node\*/  
struct Node  
{  
 int key;  
 struct Node\* left, \*right, \*random;  
};  
  
/\* Helper function that allocates a new Node with the  
 given data and NULL left, right and random pointers. \*/  
Node\* newNode(int key)  
{  
 Node\* temp = new Node;  
 temp->key = key;  
 temp->random = temp->right = temp->left = NULL;  
 return (temp);  
}  
  
/\* Given a binary tree, print its Nodes in inorder\*/  
void printInorder(Node\* node)  
{  
 if (node == NULL)  
 return;  
  
 /\* First recur on left sutree \*/  
 printInorder(node->left);  
  
 /\* then print data of Node and its random \*/  
 cout << "[" << node->key << " ";  
 if (node->random == NULL)  
 cout << "NULL], ";  
 else  
 cout << node->random->key << "], ";  
  
 /\* now recur on right subtree \*/  
 printInorder(node->right);  
}  
  
// This function creates clone by copying key and left and right pointers  
// This function also stores mapping from given tree node to clone.  
Node\* copyLeftRightNode(Node\* treeNode, map<Node \*, Node \*> \*mymap)  
{  
 if (treeNode == NULL)  
 return NULL;  
 Node\* cloneNode = newNode(treeNode->key);  
 (\*mymap)[treeNode] = cloneNode;  
 cloneNode->left = copyLeftRightNode(treeNode->left, mymap);  
 cloneNode->right = copyLeftRightNode(treeNode->right, mymap);  
 return cloneNode;  
}  
  
// This function copies random node by using the hashmap built by  
// copyLeftRightNode()  
void copyRandom(Node\* treeNode, Node\* cloneNode, map<Node \*, Node \*> \*mymap)  
{  
 if (cloneNode == NULL)  
 return;  
 cloneNode->random = (\*mymap)[treeNode->random];  
 copyRandom(treeNode->left, cloneNode->left, mymap);  
 copyRandom(treeNode->right, cloneNode->right, mymap);  
}  
  
// This function makes the clone of given tree. It mainly uses  
// copyLeftRightNode() and copyRandom()  
Node\* cloneTree(Node\* tree)  
{  
 if (tree == NULL)  
 return NULL;  
 map<Node \*, Node \*> \*mymap = new map<Node \*, Node \*>;  
 Node\* newTree = copyLeftRightNode(tree, mymap);  
 copyRandom(tree, newTree, mymap);  
 return newTree;  
}  
  
/\* Driver program to test above functions\*/  
int main()  
{  
 //Test No 1  
 Node \*tree = newNode(1);  
 tree->left = newNode(2);  
 tree->right = newNode(3);  
 tree->left->left = newNode(4);  
 tree->left->right = newNode(5);  
 tree->random = tree->left->right;  
 tree->left->left->random = tree;  
 tree->left->right->random = tree->right;  
  
 // Test No 2  
 // tree = NULL;  
  
 // Test No 3  
 // tree = newNode(1);  
  
 // Test No 4  
 /\* tree = newNode(1);  
 tree->left = newNode(2);  
 tree->right = newNode(3);  
 tree->random = tree->right;  
 tree->left->random = tree;  
 \*/  
  
 cout << "Inorder traversal of original binary tree is: \n";  
 printInorder(tree);  
  
 Node \*clone = cloneTree(tree);  
  
 cout << "\n\nInorder traversal of cloned binary tree is: \n";  
 printInorder(clone);  
  
 return 0;  
}

Output:

Inorder traversal of original binary tree is:  
[4 1], [2 NULL], [5 3], [1 5], [3 NULL],  
  
Inorder traversal of cloned binary tree is:  
[4 1], [2 NULL], [5 3], [1 5], [3 NULL],

**Method 2 (Temporarily Modify the Given Binary Tree)**

**1.** Create new nodes in cloned tree and insert each new node in original tree between the left pointer edge of corresponding node in the original tree (See the below image).  
 i.e. if current node is A and it’s left child is B ( A — >> B ), then new cloned node with key A wil be created (say cA) and it will be put as A — >> cA — >> B (B can be a NULL or a non-NULL left child). Right child pointer will be set correctly i.e. if for current node A, right child is C in original tree (A — >> C) then corresponding cloned nodes cA and cC will like cA —- >> cC

[](http://d2dskowxfbo68o.cloudfront.net/wp-content/uploads/Binary_Tree1.jpg)

**2.** Set random pointer in cloned tree as per original tree  
 i.e. if node A’s random pointer points to node B, then in cloned tree, cA will point to cB (cA and cB are new node in cloned tree corresponding to node A and B in original tree)

**3.** Restore left pointers correctly in both original and cloned tree

Following is C++ implementation of above algorithm.

#include <iostream>  
using namespace std;  
  
/\* A binary tree node has data, pointer to left child, a pointer to right  
 child and a pointer to random node\*/  
struct Node  
{  
 int key;  
 struct Node\* left, \*right, \*random;  
};  
  
/\* Helper function that allocates a new Node with the  
 given data and NULL left, right and random pointers. \*/  
Node\* newNode(int key)  
{  
 Node\* temp = new Node;  
 temp->key = key;  
 temp->random = temp->right = temp->left = NULL;  
 return (temp);  
}  
  
/\* Given a binary tree, print its Nodes in inorder\*/  
void printInorder(Node\* node)  
{  
 if (node == NULL)  
 return;  
  
 /\* First recur on left sutree \*/  
 printInorder(node->left);  
  
 /\* then print data of Node and its random \*/  
 cout << "[" << node->key << " ";  
 if (node->random == NULL)  
 cout << "NULL], ";  
 else  
 cout << node->random->key << "], ";  
  
 /\* now recur on right subtree \*/  
 printInorder(node->right);  
}  
  
// This function creates new nodes cloned tree and puts new cloned node  
// in between current node and it's left child  
// i.e. if current node is A and it's left child is B ( A --- >> B ),  
// then new cloned node with key A wil be created (say cA) and  
// it will be put as  
// A --- >> cA --- >> B  
// Here B can be a NULL or a non-NULL left child  
// Right child pointer will be set correctly  
// i.e. if for current node A, right child is C in original tree  
// (A --- >> C) then corresponding cloned nodes cA and cC will like  
// cA ---- >> cC  
Node\* copyLeftRightNode(Node\* treeNode)  
{  
 if (treeNode == NULL)  
 return NULL;  
  
 Node\* left = treeNode->left;  
 treeNode->left = newNode(treeNode->key);  
 treeNode->left->left = left;  
 if(left != NULL)  
 left->left = copyLeftRightNode(left);  
  
 treeNode->left->right = copyLeftRightNode(treeNode->right);  
 return treeNode->left;  
}  
  
// This function sets random pointer in cloned tree as per original tree  
// i.e. if node A's random pointer points to node B, then  
// in cloned tree, cA wil point to cB (cA and cB are new node in cloned  
// tree corresponding to node A and B in original tree)  
void copyRandomNode(Node\* treeNode, Node\* cloneNode)  
{  
 if (treeNode == NULL)  
 return;  
 if(treeNode->random != NULL)  
 cloneNode->random = treeNode->random->left;  
 else  
 cloneNode->random = NULL;  
  
 if(treeNode->left != NULL && cloneNode->left != NULL)  
 copyRandomNode(treeNode->left->left, cloneNode->left->left);  
 copyRandomNode(treeNode->right, cloneNode->right);  
}  
  
// This function will restore left pointers correctly in  
// both original and cloned tree  
void restoreTreeLeftNode(Node\* treeNode, Node\* cloneNode)  
{  
 if (treeNode == NULL)  
 return;  
 if (cloneNode->left != NULL)  
 {  
 Node\* cloneLeft = cloneNode->left->left;  
 treeNode->left = treeNode->left->left;  
 cloneNode->left = cloneLeft;  
 }  
 else  
 treeNode->left = NULL;  
  
 restoreTreeLeftNode(treeNode->left, cloneNode->left);  
 restoreTreeLeftNode(treeNode->right, cloneNode->right);  
}  
  
//This function makes the clone of given tree  
Node\* cloneTree(Node\* treeNode)  
{  
 if (treeNode == NULL)  
 return NULL;  
 Node\* cloneNode = copyLeftRightNode(treeNode);  
 copyRandomNode(treeNode, cloneNode);  
 restoreTreeLeftNode(treeNode, cloneNode);  
 return cloneNode;  
}  
  
  
/\* Driver program to test above functions\*/  
int main()  
{  
/\* //Test No 1  
 Node \*tree = newNode(1);  
 tree->left = newNode(2);  
 tree->right = newNode(3);  
 tree->left->left = newNode(4);  
 tree->left->right = newNode(5);  
 tree->random = tree->left->right;  
 tree->left->left->random = tree;  
 tree->left->right->random = tree->right;  
  
// Test No 2  
// Node \*tree = NULL;  
/\*  
// Test No 3  
 Node \*tree = newNode(1);  
  
// Test No 4  
 Node \*tree = newNode(1);  
 tree->left = newNode(2);  
 tree->right = newNode(3);  
 tree->random = tree->right;  
 tree->left->random = tree;  
  
 Test No 5  
 Node \*tree = newNode(1);  
 tree->left = newNode(2);  
 tree->right = newNode(3);  
 tree->left->left = newNode(4);  
 tree->left->right = newNode(5);  
 tree->right->left = newNode(6);  
 tree->right->right = newNode(7);  
 tree->random = tree->left;  
\*/  
// Test No 6  
 Node \*tree = newNode(10);  
 Node \*n2 = newNode(6);  
 Node \*n3 = newNode(12);  
 Node \*n4 = newNode(5);  
 Node \*n5 = newNode(8);  
 Node \*n6 = newNode(11);  
 Node \*n7 = newNode(13);  
 Node \*n8 = newNode(7);  
 Node \*n9 = newNode(9);  
 tree->left = n2;  
 tree->right = n3;  
 tree->random = n2;  
 n2->left = n4;  
 n2->right = n5;  
 n2->random = n8;  
 n3->left = n6;  
 n3->right = n7;  
 n3->random = n5;  
 n4->random = n9;  
 n5->left = n8;  
 n5->right = n9;  
 n5->random = tree;  
 n6->random = n9;  
 n9->random = n8;  
  
/\* Test No 7  
 Node \*tree = newNode(1);  
 tree->left = newNode(2);  
 tree->right = newNode(3);  
 tree->left->random = tree;  
 tree->right->random = tree->left;  
\*/  
 cout << "Inorder traversal of original binary tree is: \n";  
 printInorder(tree);  
  
 Node \*clone = cloneTree(tree);  
  
 cout << "\n\nInorder traversal of cloned binary tree is: \n";  
 printInorder(clone);  
  
 return 0;  
}

Output:

Inorder traversal of original binary tree is:  
[5 9], [6 7], [7 NULL], [8 10], [9 7], [10 6], [11 9], [12 8], [13 NULL],  
  
Inorder traversal of cloned binary tree is:  
[5 9], [6 7], [7 NULL], [8 10], [9 7], [10 6], [11 9], [12 8], [13 NULL],

This article is contributed by **Anurag Singh**. 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/clone-binary-tree-random-pointers/>

Category: [Trees](http://www.geeksforgeeks.org/category/tree/) Tags: [Hashing](http://www.geeksforgeeks.org/tag/hashing/)

Post navigation

[← MAQ Software Interview Experience | Set 2](http://www.geeksforgeeks.org/maq-software-interview-experience-set-2/) [Oracle Interview | Set 8 →](http://www.geeksforgeeks.org/oracle-interview-set-8/)

# Count distinct elements in every window of size k

Given an array of size n and an integer k, return the of count of distinct numbers in all windows of size k.

Example:

Input: arr[] = {1, 2, 1, 3, 4, 2, 3};  
 k = 4  
Output:  
3  
4  
4  
3  
  
Explanation:  
First window is {1, 2, 1, 3}, count of distinct numbers is 3  
Second window is {2, 1, 3, 4} count of distinct numbers is 4  
Third window is {1, 3, 4, 2} count of distinct numbers is 4  
Fourth window is {3, 4, 2, 3} count of distinct numbers is 3

**We strongly recommend you to minimize your browser and try this yourself first.**

Source: [Microsoft Interview Question](http://qa.geeksforgeeks.org/1137/counting-factorial-numbers-in-c)

A **Simple Solution** is to traverse the given array, consider every window in it and count distinct elements in the window. Below is C++ implementation of simple solution.

// Simple C++ program to count distinct elements in every  
// window of size k  
#include <iostream>  
using namespace std;  
  
// Counts distinct elements in window of size k  
int countWindowDistinct(int win[], int k)  
{  
 int dist\_count = 0;  
  
 // Traverse the  
 for (int i=0; i<k; i++)  
 {  
 // Check if element arr[i] exists in arr[0..i-1]  
 int j;  
 for (j=0; j<i; j++)  
 if (win[i] == win[j])  
 break;  
 if (j==i)  
 dist\_count++;  
 }  
 return dist\_count;  
}  
  
// Counts distinct elements in all windows of size k  
void countDistinct(int arr[], int n, int k)  
{  
 // Traverse through every window  
 for (int i=0; i<=n-k; i++)  
 cout << countWindowDistinct(arr+i, k) << endl;  
}  
  
// Driver program  
int main()  
{  
 int arr[] = {1, 2, 1, 3, 4, 2, 3}, k = 4;  
 int n = sizeof(arr)/sizeof(arr[0]);  
 countDistinct(arr, n, k);  
 return 0;  
}

Output:

3  
4  
4  
3

Time complexity of the above solution is O(nk2). We can improve time complexity to O(nkLok) by modifying countWindowDistinct() to use sorting. The function can further be optimized to use [hashing to find distinct elements](http://geeksquiz.com/print-distinct-elements-given-integer-array/) in a window. With hashing the time complexity becomes O(nk). Below is a different approach that works in O(n) time.

An **Efficient Solution** is to use the count of previous window, while sliding the window. The idea is to create a hash map that stores elements of current widow. When we slide the window, we remove an element from hash and add an element. We also keep track of distinct elements. Below is algorithm.

1) Create an empty hash map. Let hash map be hM

2) Initialize distinct element count ‘dist\_count’ as 0.

3) Traverse through first window and insert elements of first window to hM. The elements are used as key and their counts as value in hM. Also keep updating ‘dist\_count’

4) Print ‘dist\_count’ for first window.

3) Traverse through remaining array (or other windows).  
 ….a) Remove the first element of previous window.  
 …….If the removed element appeared only once  
 …………..remove it from hM and do “dist\_count–”  
 …….Else (appeared multiple times in hM)  
 …………..decrement its count in hM

….a) Add the current element (last element of new window)  
 …….If the added element is not present in hM  
 …………..add it to hM and do “dist\_count++”  
 …….Else (the added element appeared multiple times)  
 …………..increment its count in hM

Below is Java implementation of above approach.

// An efficient Java program to count distinct elements in  
// every window of size k  
import java.util.HashMap;  
  
class CountDistinctWindow  
{  
 static void countDistinct(int arr[], int k)  
 {  
 // Creates an empty hashMap hM  
 HashMap<Integer, Integer> hM =  
 new HashMap<Integer, Integer>();  
  
 // initialize distinct element count for  
 // current window  
 int dist\_count = 0;  
  
 // Traverse the first window and store count  
 // of every element in hash map  
 for (int i = 0; i < k; i++)  
 {  
 if (hM.get(arr[i]) == null)  
 {  
 hM.put(arr[i], 1);  
 dist\_count++;  
 }  
 else  
 {  
 int count = hM.get(arr[i]);  
 hM.put(arr[i], count+1);  
 }  
 }  
  
 // Print count of first window  
 System.out.println(dist\_count);  
  
 // Traverse through the remaining array  
 for (int i = k; i < arr.length; i++)  
 {  
  
 // Remove first element of previous window  
 // If there was only one occurrence, then  
 // reduce distinct count.  
 if (hM.get(arr[i-k]) == 1)  
 {  
 hM.remove(arr[i-k]);  
 dist\_count--;  
 }  
 else // reduce count of the removed element  
 {  
 int count = hM.get(arr[i-k]);  
 hM.put(arr[i-k], count-1);  
 }  
  
 // Add new element of current window  
 // If this element appears first time,  
 // increment distinct element count  
 if (hM.get(arr[i]) == null)  
 {  
 hM.put(arr[i], 1);  
 dist\_count++;  
 }  
 else // Increment distinct element count  
 {  
 int count = hM.get(arr[i]);  
 hM.put(arr[i], count+1);  
 }  
  
 // Print count of current window  
 System.out.println(dist\_count);  
 }  
 }  
  
 // Driver method  
 public static void main(String arg[])  
 {  
 int arr[] = {1, 2, 1, 3, 4, 2, 3};  
 int k = 4;  
 countDistinct(arr, k);  
 }  
}

Output:

3  
4  
4  
3

Time complexity of the above solution is O(n).

This article is contributed by **Piyush**. 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/count-distinct-elements-in-every-window-of-size-k/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Hashing](http://www.geeksforgeeks.org/tag/hashing/)

Post navigation

[← Amazon Interview Experience | Set 208 (On-Campus for Internship)](http://www.geeksforgeeks.org/amazon-interview-experience-set-208-on-campus-for-internship/) [Design Patterns | Set 1 (Introduction) →](http://www.geeksforgeeks.org/design-patterns-set-1-introduction/)

# Design a data structure that supports insert, delete, search and getRandom in constant time

Design a data structure that supports following operations in Θ(1) time.

insert(x): Inserts an item x to the data structure if not already present.

remove(x): Removes an item x from the data structure if present.

search(x): Searches an item x in the data structure.

getRandom(): Returns a random element from current set of elements

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

We can use [hashing](http://www.geeksforgeeks.org/tag/hashing/)to support first 3 operations in Θ(1) time. How to do the 4th operation? The idea is to use a resizable array (ArrayList in Java, vector in C) together with hashing. [Resizable arrays support insert in Θ(1) amortized time complexity](http://www.geeksforgeeks.org/analysis-algorithm-set-5-amortized-analysis-introduction/). To implement getRandom(), we can simply pick a random number from 0 to size-1 (size is number of current elements) and return the element at that index. The hash map stores array values as keys and array indexes as values.

Following are detailed operations.

***insert(x)***  
 1) Check if x is already present by doing a hash map lookup.  
 2) If not present, then insert it at the end of the array.  
 3) Add in hash table also, x is added as key and last array index as index.

***remove(x)***  
 1) Check if x is present by doing a hash map lookup.  
 2) If present, then find its index and remove it from hash map.  
 3) Swap the last element with this element in array and remove the last element.  
 Swapping is done because the last element can be removed in O(1) time.  
 4) Update index of last element in hash map.

***getRandom()***  
 1) Generate a random number from 0 to last index.  
 2) Return the array element at the randomly generated index.

***search(x)***  
 Do a lookup for x in hash map.

Below is Java implementation of the data structure.

/\* Java program to design a data structure that support folloiwng operations  
 in Theta(n) time  
 a) Insert  
 b) Delete  
 c) Search  
 d) getRandom \*/  
import java.util.\*;  
  
// class to represent the required data structure  
class MyDS  
{  
 ArrayList<Integer> arr; // A resizable array  
  
 // A hash where keys are array elements and vlaues are  
 // indexes in arr[]  
 HashMap<Integer, Integer> hash;  
  
 // Constructor (creates arr[] and hash)  
 public MyDS()  
 {  
 arr = new ArrayList<Integer>();  
 hash = new HashMap<Integer, Integer>();  
 }  
  
 // A Theta(1) function to add an element to MyDS  
 // data structure  
 void add(int x)  
 {  
 // If ekement is already present, then noting to do  
 if (hash.get(x) != null)  
 return;  
  
 // Else put element at the end of arr[]  
 int s = arr.size();  
 arr.add(x);  
  
 // And put in hash also  
 hash.put(x, s);  
 }  
  
 // A Theta(1) function to remove an element from MyDS  
 // data structure  
 void remove(int x)  
 {  
 // Check if element is present  
 Integer index = hash.get(x);  
 if (index == null)  
 return;  
  
 // If present, then remove element from hash  
 hash.remove(x);  
  
 // Swap element with last element so that remove from  
 // arr[] can be done in O(1) time  
 int size = arr.size();  
 Integer last = arr.get(size-1);  
 Collections.swap(arr, index, size-1);  
  
 // Remove last element (This is O(1))  
 arr.remove(size-1);  
  
 // Update hash table for new index of last element  
 hash.put(last, index);  
 }  
  
 // Returns a random element from MyDS  
 int getRandom()  
 {  
 // Find a random index from 0 to size - 1  
 Random rand = new Random(); // Choose a different seed  
 int index = rand.nextInt(arr.size());  
  
 // Return element at randomly picked index  
 return arr.get(index);  
 }  
  
 // Returns index of element if element is present, otherwise null  
 Integer search(int x)  
 {  
 return hash.get(x);  
 }  
}  
  
// Driver class  
class Main  
{  
 public static void main (String[] args)  
 {  
 MyDS ds = new MyDS();  
 ds.add(10);  
 ds.add(20);  
 ds.add(30);  
 ds.add(40);  
 System.out.println(ds.search(30));  
 ds.remove(20);  
 ds.add(50);  
 System.out.println(ds.search(50));  
 System.out.println(ds.getRandom());  
 }  
}

Output:

2  
3  
40

This article is contributed by **Manish Gupta**. 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/design-a-data-structure-that-supports-insert-delete-search-and-getrandom-in-constant-time/>

# Find the first repeating element in an array of integers

Given an array of integers, find the first repeating element in it. We need to find the element that occurs more than once and whose index of first occurrence is smallest.

Examples:

Input: arr[] = {10, 5, 3, 4, 3, 5, 6}  
Output: 5 [5 is the first element that repeats]  
  
Input: arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10}  
Output: 6 [6 is the first element that repeats]

A **Simple Solution** is to use two nested loops. The outer loop picks an element one by one, the inner loop checks whether the element is repeated or not. Once we find an element that repeats, we break the loops and print the element. Time Complexity of this solution is O(n2)

We can **Use Sorting** to solve the problem in O(nLogn) time. Following are detailed steps.  
 1) Copy the given array to an auxiliary array temp[].  
 2) Sort the temp array using a O(nLogn) time sorting algorithm.  
 3) Scan the input array from left to right. For every element, [count its occurrences in temp[] using binary search](http://www.geeksforgeeks.org/count-number-of-occurrences-in-a-sorted-array/). As soon as we find an element that occurs more than once, we return the element. This step can be done in O(nLogn) time.

We can **Use** [**Hashing**](http://geeksquiz.com/hashing-set-1-introduction/) to solve this in O(n) time on average. The idea is to traverse the given array from right to left and update the minimum index whenever we find an element that has been visited on right side. Thanks to Mohammad Shahid for suggesting this solution.

Following is Java implementation of this idea.

/\* Java program to find first repeating element in arr[] \*/  
import java.util.\*;  
  
class Main  
{  
 // This function prints the first repeating element in arr[]  
 static void printFirstRepeating(int arr[])  
 {  
 // Initialize index of first repeating element  
 int min = -1;  
  
 // Creates an empty hashset  
 HashSet<Integer> set = new HashSet<>();  
  
 // Traverse the input array from right to left  
 for (int i=arr.length-1; i>=0; i--)  
 {  
 // If element is already in hash set, update min  
 if (set.contains(arr[i]))  
 min = i;  
  
 else // Else add element to hash set  
 set.add(arr[i]);  
 }  
  
 // Print the result  
 if (min != -1)  
 System.out.println("The first repeating element is " + arr[min]);  
 else  
 System.out.println("There are no repeating elements");  
 }  
  
 // Driver method to test above method  
 public static void main (String[] args) throws java.lang.Exception  
 {  
 int arr[] = {10, 5, 3, 4, 3, 5, 6};  
 printFirstRepeating(arr);  
 }  
}

Output:

The first repeating element is 5

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-first-repeating-element-array-integers/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Hashing](http://www.geeksforgeeks.org/tag/hashing/)

# Find if there is a subarray with 0 sum

Given an array of positive and negative numbers, find if there is a subarray with 0 sum.

Examples:

Input: {4, 2, -3, 1, 6}  
Output: true   
There is a subarray with zero sum from index 1 to 3.  
  
Input: {4, 2, 0, 1, 6}  
Output: true   
There is a subarray with zero sum from index 2 to 2.  
  
Input: {-3, 2, 3, 1, 6}  
Output: false  
There is no subarray with zero sum.

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

A **simple solution** is to consider all subarrays one by one and check the sum of every subarray. We can run two loops: the outer loop picks a starting point i and the inner loop tries all subarrays starting from i (See [this](http://www.geeksforgeeks.org/find-subarray-with-given-sum/)for implementation). Time complexity of this method is O(n2).

We can also **use hashing**. The idea is to iterate through the array and for every element arr[i], calculate sum of elements form 0 to i (this can simply be done as sum += arr[i]). If the current sum has been seen before, then there is a zero sum array. Hashing is used to store the sum values, so that we can quickly store sum and find out whether the current sum is seen before or not.

Following is Java implementation of the above approach.

// A Java program to find if there is a zero sum subarray  
import java.util.HashMap;  
   
class ZeroSumSubarray {  
   
 // Returns true if arr[] has a subarray with sero sum  
 static Boolean printZeroSumSubarray(int arr[])  
 {  
 // Creates an empty hashMap hM  
 HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();  
   
 // Initialize sum of elements  
 int sum = 0;   
   
 // Traverse through the given array  
 for (int i = 0; i < arr.length; i++)  
 {   
 // Add current element to sum  
 sum += arr[i];  
   
 // Return true in following cases  
 // a) Current element is 0  
 // b) sum of elements from 0 to i is 0  
 // c) sum is already present in hash map  
 if (arr[i] == 0 || sum == 0 || hM.get(sum) != null)   
 return true;  
   
 // Add sum to hash map  
 hM.put(sum, i);  
 }   
   
 // We reach here only when there is no subarray with 0 sum  
 return false;  
 }   
   
 public static void main(String arg[])  
 {  
 int arr[] = {4, 2, -3, 1, 6};  
 if (printZeroSumSubarray(arr))  
 System.out.println("Found a subarray with 0 sum");  
 else  
 System.out.println("No Subarray with 0 sum");   
 }   
}

Output:

Found a subarray with 0 sum

Time Complexity of this 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.

**Exercise:**  
 Extend the above program to print starting and ending indexes of all subarrays with 0 sum.

This article is contributed by **Chirag Gupta**. 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-if-there-is-a-subarray-with-0-sum/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Hashing](http://www.geeksforgeeks.org/tag/hashing/)

# 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](http://www.geeksforgeeks.org/topological-sorting/) of the graph. Time complexity of this solution is O(n).

We can also use [hashing](http://geeksquiz.com/hashing-set-1-introduction/)to 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](http://www.geeksforgeeks.org/category/c-programs/) Tags: [Hashing](http://www.geeksforgeeks.org/tag/hashing/)

# 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](http://www.geeksforgeeks.org/category/c-programs/) Tags: [Hashing](http://www.geeksforgeeks.org/tag/hashing/)

Post navigation

[← Snapdeal Interview Experience | Set 13 (On-Campus for SDET)](http://www.geeksforgeeks.org/snapdeal-interview-experience-set-13-on-campus-for-sdet/) [Amazon Interview Experience | Set 188 (For SDE1) →](http://www.geeksforgeeks.org/amazon-interview-experience-set-188-for-sde1/)

# Group multiple occurrence of array elements ordered by first occurrence

Given an unsorted array with repetitions, the task is to group multiple occurrence of individual elements. The grouping should happen in a way that the order of first occurrences of all elements is maintained.

Examples:

Input: arr[] = {5, 3, 5, 1, 3, 3}  
Output: {5, 5, 3, 3, 3, 1}  
  
Input: arr[] = {4, 6, 9, 2, 3, 4, 9, 6, 10, 4}  
Output: {4, 4, 4, 6, 6, 9, 9, 2, 3, 10}

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

**Simple Solution** is to use nested loops. The outer loop traverses array elements one by one. The inner loop checks if this is first occurrence, if yes, then the inner loop prints it and all other occurrences.

// A simple C++ program to group multiple occurrences of individual  
// array elements  
#include<iostream>  
using namespace std;  
  
// A simple method to group all occurrences of individual elements  
void groupElements(int arr[], int n)  
{  
 // Initialize all elements as not visited  
 bool \*visited = new bool[n];  
 for (int i=0; i<n; i++)  
 visited[i] = false;  
  
 // Traverse all elements  
 for (int i=0; i<n; i++)  
 {  
 // Check if this is first occurrence  
 if (!visited[i])  
 {  
 // If yes, print it and all subsequent occurrences  
 cout << arr[i] << " ";  
 for (int j=i+1; j<n; j++)  
 {  
 if (arr[i] == arr[j])  
 {  
 cout << arr[i] << " ";  
 visited[j] = true;  
 }  
 }  
 }  
 }  
  
 delete [] visited;   
}  
  
/\* Driver program to test above function \*/  
int main()  
{  
 int arr[] = {4, 6, 9, 2, 3, 4, 9, 6, 10, 4};  
 int n = sizeof(arr)/sizeof(arr[0]);  
 groupElements(arr, n);  
 return 0;  
}

Output:

4 4 4 6 6 9 9 2 3 10

Time complexity of the above method is O(n2).

**Binary Search Tree based Method:** The time complexity can be improved to O(nLogn) using self-balancing binary search tree like [Red-Black Tree](http://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/) or [AVL tree](http://www.geeksforgeeks.org/avl-tree-set-1-insertion/). Following is complete algorithm.  
 1) Create an empty Binary Search Tree (BST). Every BST node is going to contain an array element and its count.  
 2) Traverse the input array and do following for every element.  
 ……..a) If element is not present in BST, then insert it with count as 0.  
 ……..b) If element is present, then increment count in corresponding BST node.  
 3) Traverse the array again and do following for every element.  
 …….. If element is present in BST, then do following  
 ……….a) Get its count and print the element ‘count’ times.  
 ……….b) Delete the element from BST.

Time Complexity of the above solution is O(nLogn).

**Hashing based Method:** We can also use hashing. The idea is to replace Binary Search Tree with a Hash Map in above algorithm.

Below is Java Implementation of hashing based solution.

/\* Java program to group multiple occurrences of individual array elements \*/  
import java.util.HashMap;  
  
class Main  
{  
 // A hashing based method to group all occurrences of individual elements  
 static void orderedGroup(int arr[])  
 {  
 // Creates an empty hashmap  
 HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();  
  
 // Traverse the array elements, and store count for every element  
 // in HashMap  
 for (int i=0; i<arr.length; i++)  
 {  
 // Check if element is already in HashMap  
 Integer prevCount = hM.get(arr[i]);  
 if (prevCount == null)   
 prevCount = 0;  
   
 // Increment count of element element in HashMap   
 hM.put(arr[i], prevCount + 1);  
 }  
  
 // Traverse array again   
 for (int i=0; i<arr.length; i++)  
 {   
 // Check if this is first occurrence  
 Integer count = hM.get(arr[i]);   
 if (count != null)  
 {  
 // If yes, then print the element 'count' times  
 for (int j=0; j<count; j++)  
 System.out.print(arr[i] + " ");  
   
 // And remove the element from HashMap.  
 hM.remove(arr[i]);  
 }  
 }  
 }  
  
 // Driver method to test above method  
 public static void main (String[] args)  
 {  
 int arr[] = {10, 5, 3, 10, 10, 4, 1, 3};  
 orderedGroup(arr);  
 }  
}

Output:

10 10 10 5 3 3 4 1

Time Complexity of the above hashing based solution is Θ(n) under the assumption that insert, search and delete operations on HashMap take O(1) time.

This article is contributed by **Himanshu Gupta**. 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/group-multiple-occurrence-of-array-elements-ordered-by-first-occurrence/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Hashing](http://www.geeksforgeeks.org/tag/hashing/)

Post navigation

[← Given a sorted and rotated array, find if there is a pair with a given sum](http://www.geeksforgeeks.org/given-a-sorted-and-rotated-array-find-if-there-is-a-pair-with-a-given-sum/) [SAP Labs Interview Experience | Set 6 (Experienced) →](http://www.geeksforgeeks.org/sap-labs-interview-experience-set-6-experienced/)

# Length of the largest subarray with contiguous elements | Set 2

Given an array of integers, find length of the longest subarray which contains numbers that can be arranged in a continuous sequence.

In the[previous post](http://www.geeksforgeeks.org/length-largest-subarray-contiguous-elements-set-1/), we have discussed a solution that assumes that elements in given array are distinct. Here we discuss a solution that works even if the input array has duplicates.

Examples:

Input: arr[] = cc  
Output: Length of the longest contiguous subarray is 4  
  
Input: arr[] = {10, 12, 12, 10, 10, 11, 10};  
Output: Length of the longest contiguous subarray is 2

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

The idea is similar to previous post. In the previous post, we checked whether maximum value minus minimum value is equal to ending index minus starting index or not. Since duplicate elements are allowed, we also need to check if the subarray contains duplicate elements or not. For example, the array {12, 14, 12} follows the first property, but numbers in it are not contiguous elements.  
 To check duplicate elements in a subarray, we create a hash set for every subarray and if we find an element already in hash, we don’t consider the current subarray.

Following is Java implementation of the above idea.

/\* Java program to find length of the largest subarray which has  
 all contiguous elements \*/  
import java.util.\*;  
  
class Main  
{  
 // This function prints all distinct elements  
 static int findLength(int arr[])  
 {  
 int n = arr.length;  
 int max\_len = 1; // Inialize result  
  
 // One by one fix the starting points  
 for (int i=0; i<n-1; i++)  
 {  
 // Create an empty hash set and add i'th element  
 // to it.  
 HashSet<Integer> set = new HashSet<>();  
 set.add(arr[i]);  
  
 // Initialize max and min in current subarray  
 int mn = arr[i], mx = arr[i];  
  
 // One by one fix ending points  
 for (int j=i+1; j<n; j++)  
 {  
 // If current element is already in hash set, then  
 // this subarray cannot contain contiguous elements  
 if (set.contains(arr[j]))  
 break;  
  
 // Else add curremt element to hash set and update  
 // min, max if required.  
 set.add(arr[j]);  
 mn = Math.min(mn, arr[j]);  
 mx = Math.max(mx, arr[j]);  
  
 // We have already cheched for duplicates, now check  
 // for other property and update max\_len if needed  
 if (mx-mn == j-i)  
 max\_len = Math.max(max\_len, mx-mn+1);  
 }  
 }  
 return max\_len; // Return result  
 }  
  
 // Driver method to test above method  
 public static void main (String[] args)  
 {  
 int arr[] = {10, 12, 12, 10, 10, 11, 10};  
 System.out.println("Length of the longest contiguous subarray is " +  
 findLength(arr));  
 }  
}

Output:

Length of the longest contiguous subarray is 2

Time complexity of the above solution is O(n2) under the assumption that hash set operations like add() and contains() work in O(1) time.

This article is contributed by **Arjun**. 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/length-largest-subarray-contiguous-elements-set-2/>

Category: [Interview Experiences](http://www.geeksforgeeks.org/category/interview-experiences/) Tags: [Hashing](http://www.geeksforgeeks.org/tag/hashing/)

# Given an array A[] and a number x, check for pair in A[] with sum as x

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) in worst case. If we use Quick Sort then O(n^2) in worst case.  
  
Auxiliary Space : Again, depends on sorting algorithm. For example auxiliary space is O(n) for merge sort and O(1) for Heap Sort.  
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 extended for this though.  
Implementation:  
   
# 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);  
 }  
}

**METHOD 2 (Use Hash Map)**  
 Thanks to Bindu for suggesting this method and thanks to [Shekhu](http://geeksforgeeks.org/forum/topic/array)for 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:

#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 = 6;  
  
 printPairs(A, arr\_size, n);  
  
 getchar();  
 return 0;  
}

Time Complexity: O(n)  
 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-whether-or-not-there-exist-two-elements-in-s-whose-sum-is-exactly-x/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Hashing](http://www.geeksforgeeks.org/tag/hashing/)

Post navigation

[← Next Power of 2](http://www.geeksforgeeks.org/next-power-of-2/) [Majority Element →](http://www.geeksforgeeks.org/majority-element/)