Java Algorithm Interview Questions

codespaghetti.com/java-algorithms-questions/

Algorithms

Java Algorithm And Data Structure Interview Questions and Programs

Table of Contents:

Top Five Data Structure And Algorithm Interview Questions?

Searching And Sorting Algorithms Interview Questions

Java Quick Sort Interview Questions

What is Quick Sort Algorithm ?

Quick sort partitions an array and then calls itself recursively twice to sort the two resulting subarrays.

This algorithm is quite efficient for large-sized data sets as its average and worst case complexity are of $O(n^2)$, where **n** is the number of items.

ALGORITHM

```
_# choose pivot_
swap a[1,rand(1,n)]

_# 2-way partition_
k = 1
for i = 2:n, if a[i] < a[1], swap a[++k,i]
swap a[1,k]
_→ invariant: a[1..k-1] < a[k] <= a[k+1..n]_

_# recursive sorts_
sort a[1..k-1]
sort a[k+1,n]
```

Full Implementation

```
package codespaghetti.com;

public class MyQuickSort {

    private int array[];
    private int length;

    public void sort(int[] inputArr) {

        if (inputArr == null || inputArr.length == 0) {
            return;
        }
        this.array = inputArr;
        length = inputArr.length;
        quickSort(0, length - 1);
    }

    private void quickSort(int lowerIndex, int higherIndex) {

        int i = lowerIndex;
        int j = higherIndex;
        // calculate pivot number, I am taking pivot as middle index number
        int pivot = array[lowerIndex+(higherIndex-lowerIndex)/2];
        // Divide into two arrays
        while (i <= j) {
            /**
             * In each iteration, we will identify a number from left side which
             * is greater then the pivot value, and also we will identify a number
             * from right side which is less then the pivot value. Once the search
             * is done, then we exchange both numbers.
             */
            while (array[i] < pivot) { i++; } while (array[j] > pivot) {
                j--;
            }
            if (i <= j) {
                exchangeNumbers(i, j);
                //move index to next position on both sides
                i++;
                j--;
            }
        }
        // call quickSort() method recursively
        if (lowerIndex < j)
            quickSort(lowerIndex, j);
        if (i < higherIndex)
            quickSort(i, higherIndex);
    }

    private void exchangeNumbers(int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public static void main(String a[]){

        MyQuickSort sorter = new MyQuickSort();
        int[] input = {24,2,45,20,56,75,2,56,99,53,12};
        sorter.sort(input);
        for(int i:input){
            System.out.print(i);
            System.out.print(" ");
        }
    }
}
```

What are properties of Quick sort ?

- Not stable
- O(lg(n)) extra space
- O(n^2) time, but typically O(n·lg(n)) time
- Not adaptive

When to use Quick sort ?

When carefully implemented, quick sort is robust and has low overhead. When a stable sort is not needed, quick sort is an excellent general-purpose sort – although the 3-way partitioning version should always be used instead.

The 2-way partitioning code shown above is written for clarity rather than optimal performance.

It exhibits poor locality, and, critically, exhibits O(n^2) time when there are few unique keys.

A more efficient and robust 2-way partitioning method is given in Quicksort is Optimal by Robert Sedgewick and Jon Bentley.

The robust partitioning produces balanced recursion when there are many values equal to the pivot, yielding probabilistic guarantees of O(n·lg(n)) time and O(lg(n)) space for all inputs.

With both sub-sorts performed recursively, quick sort requires O(n) extra space for the recursion stack in the worst case when recursion is not balanced.

This is exceedingly unlikely to occur, but it can be avoided by sorting the *smaller* sub-array recursively first; the second sub-array sort is a tail recursive call, which may be done with iteration instead.

With this optimization, the algorithm uses O(lg(n)) extra space in the worst case.

What is performance of Quick sort?

Worst-case performance	O(n^2)
Best-case performance	O(*n* log *n*) (simple partition) or O(*n*) (three-way partition and equal keys)
Average performance	O(*n* log *n*)

Worst-case space complexity	O(*n*) auxiliary (naive) O(log *n*) auxiliary

Java Linear Search Algorithm Interview Questions

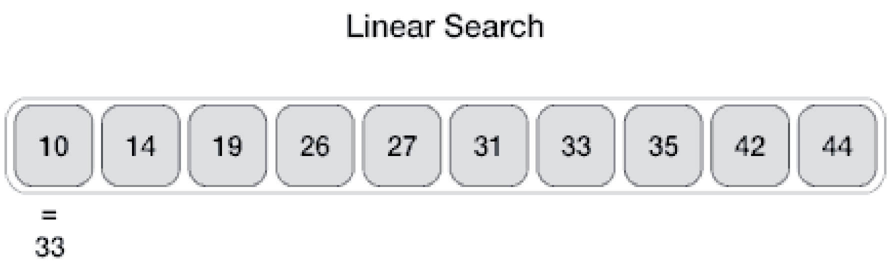
What is Linear or Sequential Search?

linear search or **sequential search** is a method for finding a target value within a list.

It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.

Linear search runs in at worst linear time and makes at most*n* comparisons, where *n* is the length of the list.

How does it work ?



Algorithm

```
Linear Search ( Array A, Value x)

Step 1: Set i to 1
Step 2: if i > n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit
```

Pseudocode

```
procedure linear_search (list, value)

   for each item in the list

      if match item == value

         return the item's location

      end if

   end for

end procedure
```

Full Implementation in Java

```
package codespaghetti.com;

    public class MyLinearSearch {

    public static int linerSearch(int[] arr, int key){

        int size = arr.length;
        for(int i=0;i<size;i++){
            if(arr[i] == key){
                return i;
            }
        }
        return -1;
    }

    public static void main(String a[]){

        int[] arr1= {23,45,21,55,234,1,34,90};
        int searchKey = 34;
        System.out.println("Key "+searchKey+" found at index: "+linerSearch(arr1,
searchKey));
        int[] arr2= {123,445,421,595,2134,41,304,190};
        searchKey = 421;
        System.out.println("Key "+searchKey+" found at index: "+linerSearch(arr2,
searchKey));
    }
}

Output:
Key 34 found at index: 6
Key 421 found at index: 2
```

What is performance of Linear search?

Worst-case performance	$O(n)$
Best-case performance	$O(1)$

Average performance	*O*(*n*)
Worst-case space complexity	*O*(1) iterative

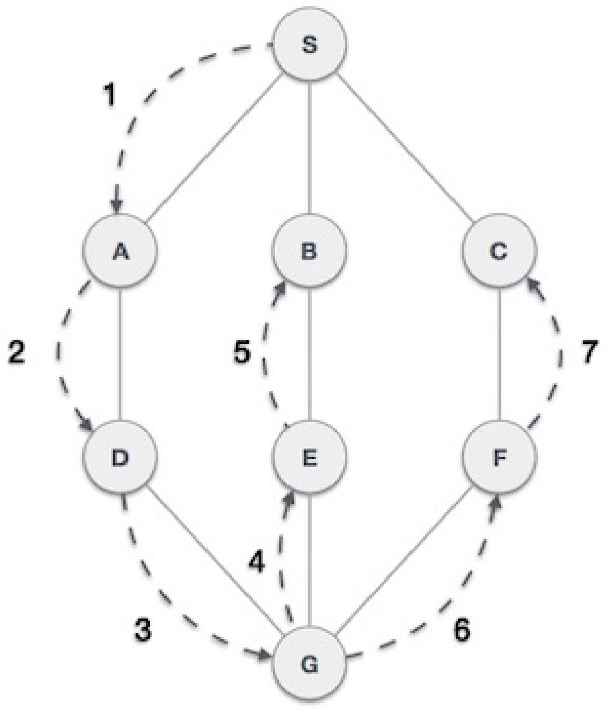
Depth First Interview Questions

What is Depth first search?

Depth First Search algorithm traverses a graph in a depth motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

As in the example given above,Depth First Search algorithm traverses from A to B to C to D first then to E, then to F and lastly to G. It employs the following rules.

- **Rule 1** − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
- **Rule 2** − If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- **Rule 3** − Repeat Rule 1 and Rule 2 until the stack is empty.



Full Implementation in Java

```
import java.io.*;
import java.util.*;

// This class represents a directed graph using adjacency list representation
class Graph
{
 private int V; // No. of vertices

 // Array of lists for Adjacency List Representation
 private LinkedList adj[];

 // Constructor
 Graph(int v)
 {
  V = v;
  adj = new LinkedList[v];
  for (int i=0; i<v; ++i)
   adj[i] = new LinkedList();
 }

 //Function to add an edge into the graph
 void addEdge(int v, int w)
 {
  adj[v].add(w); // Add w to v's list.
 }

 // A function used by DFS
 void DFSUtil(int v,boolean visited[])
 {
  // Mark the current node as visited and print it
  visited[v] = true;
  System.out.print(v+" ");

  // Recur for all the vertices adjacent to this vertex
  Iterator i = adj[v].listIterator();
  while (i.hasNext())
  {
   int n = i.next();
   if (!visited[n])
    DFSUtil(n,visited);
  }
 }

 // The function to do DFS traversal. It uses recursive DFSUtil()
 void DFS()
 {
  // Mark all the vertices as not visited(set as
  // false by default in java)
  boolean visited[] = new boolean[V];

  // Call the recursive helper function to print DFS traversal
  // starting from all vertices one by one
  for (int i=0; i<V; ++i)
   if (visited[i] == false)
    DFSUtil(i, visited);
 }

 public static void main(String args[])
 {
  Graph g = new Graph(4);

  g.addEdge(0, 1);
  g.addEdge(0, 2);
  g.addEdge(1, 2);
  g.addEdge(2, 0);
  g.addEdge(2, 3);
  g.addEdge(3, 3);

  System.out.println("Following is Depth First Traversal");

  g.DFS();
 }
}
Output:
Following is Depth First Traversal
```

```
0 1 2 3
Time Complexity: O(V+E) where V is number of vertices in the graph and E is number
of edges in the graph.
```

What is performance of Depth First search?

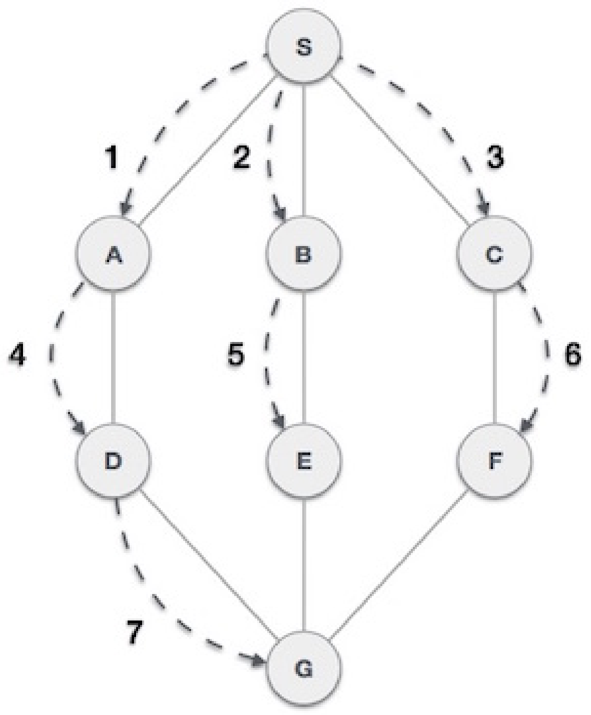
Worst-case performance	{displaystyle O(|V|+|E|)} for explicit graphs traversed without repetition, {displaystyle O(b^{d})} - for implicit graphs with branching factor *b* searched to depth *d*
Worst-case space complexity	{displaystyle O(|V|)} - if entire graph is traversed without repetition, O(longest path length searched) for implicit graphs without elimination of duplicate nodes

Breadth First Interview Questions

What is Breadth first search?

Breadth First Search algorithm traverses a graph in a breadth motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration. As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

- **Rule 1** − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
- **Rule 2** − If no adjacent vertex is found, remove the first vertex from the queue.
- **Rule 3** − Repeat Rule 1 and Rule 2 until the queue is empty.



Full Implementation in Java

```
import java.io.*;
import java.util.*;

// This class represents a directed graph using adjacency list
// representation
class Graph
{
    private int V;   // No. of vertices
    private LinkedList adj[]; //Adjacency Lists

    // Constructor
    Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
            adj[i] = new LinkedList();
    }

    // Function to add an edge into the graph
    void addEdge(int v,int w)
    {
        adj[v].add(w);
    }

    // prints BFS traversal from a given source s
    void BFS(int s)
    {
        // Mark all the vertices as not visited(By default
        // set as false)
        boolean visited[] = new boolean[V];

        // Create a queue for BFS
        LinkedList queue = new LinkedList();

        // Mark the current node as visited and enqueue it
        visited[s]=true;
        queue.add(s);

        while (queue.size() != 0)
        {
            // Dequeue a vertex from queue and print it
            s = queue.poll();
            System.out.print(s+" ");

            // Get all adjacent vertices of the dequeued vertex s
            // If a adjacent has not been visited, then mark it
            // visited and enqueue it
            Iterator i = adj[s].listIterator();
            while (i.hasNext())
            {
                int n = i.next();
                if (!visited[n])
                {
                    visited[n] = true;
                    queue.add(n);
                }
            }
        }
    }

    // Driver method to
    public static void main(String args[])
    {
        Graph g = new Graph(4);

        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        System.out.println("Following is Breadth First Traversal "+
                           "(starting from vertex 2)");

        g.BFS(2);
    }
}

Output:
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
```

What is performance of Breadth First search?

Worst-case performance	[illegible]
Worst-case space complexity	[illegible]

Binary Search Interview Questions

What is Binary search ?

Binary search is a fast search algorithm with run-time complexity of O(log n). This search algorithm works on the principle of divide and conquer.

For this algorithm to work properly, the data collection should be in the sorted form.

Binary search compares the target value to the middle element of the array, if they are unequal, the half in which the target cannot lie is eliminated and the search continues on the remaining half until it is successful or the remaining half is empty.

How does it work?

For a binary search to work, it is mandatory for the target array to be sorted.

Pseudocode

The pseudocode of binary search algorithms should look like this −

```
Procedure binary_search
   A ← sorted array
   n ← size of array
   x ← value to be searched

   Set lowerBound = 1
   Set upperBound = n

   while x not found
      if upperBound < lowerBound
         EXIT: x does not exists.

      set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

      if A[midPoint] < x
         set lowerBound = midPoint + 1

      if A[midPoint] > x
         set upperBound = midPoint - 1

      if A[midPoint] = x
         EXIT: x found at location midPoint

   end while

end procedure
```

Full Implementation in Java

```
package codespaghetti.com;

public class MyBinarySearch {

    public int binarySearch(int[] inputArr, int key) {

        int start = 0;
        int end = inputArr.length - 1;
        while (start <= end) {
            int mid = (start + end) / 2;
            if (key == inputArr[mid]) {
                return mid;
            }
            if (key < inputArr[mid]) {
                end = mid - 1;
            } else {
                start = mid + 1;
            }
        }
        return -1;
    }

    public static void main(String[] args) {

        MyBinarySearch mbs = new MyBinarySearch();
        int[] arr = {2, 4, 6, 8, 10, 12, 14, 16};
        System.out.println("Key 14's position: "+mbs.binarySearch(arr, 14));
        int[] arr1 = {6,34,78,123,432,900};
        System.out.println("Key 432's position: "+mbs.binarySearch(arr1, 432));
    }
}

Output:
Key 14's position: 6
Key 432's position: 4
```

What is performance of Binary search?

Worst-case performance	*O*(log *n*)
Best-case performance	*O*(1)
Average performance	*O*(log *n*)
Worst-case space complexity	*O*(1)

How To Find Minimum Depth of a Binary Tree?

The minimum depth is the number of nodes along the shortest path from root node down to the nearest leaf node.

Analysis:

Traverse the given Binary Tree. For every node, check if it is a leaf node. If yes, then return 1. If not leaf node then if left subtree is NULL, then recur for right subtree.

And if right subtree is NULL, then recur for left subtree. If both left and right subtrees are not NULL, then take the minimum of two heights.

Full Implementation in Java:

```
/* Class containing left and right child of current
node and key value*/
class Node
{
    int data;
    Node left, right;
    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}
public class BinaryTree
{
    //Root of the Binary Tree
    Node root;

    int minimumDepth()
    {
        return minimumDepth(root);
    }

    /* Function to calculate the minimum depth of the tree */
    int minimumDepth(Node root)
    {
        // Corner case. Should never be hit unless the code is
        // called on root = NULL
        if (root == null)
            return 0;

        // Base case : Leaf Node. This accounts for height = 1.
        if (root.left == null && root.right == null)
            return 1;

        // If left subtree is NULL, recur for right subtree
        if (root.left == null)
            return minimumDepth(root.right) + 1;

        // If right subtree is NULL, recur for right subtree
        if (root.right == null)
            return minimumDepth(root.left) + 1;

        return Math.min(minimumDepth(root.left),
                        minimumDepth(root.right)) + 1;
    }
    /* Driver program to test above functions */
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        System.out.println("The minimum depth of "+
          "binary tree is : " + tree.minimumDepth());
    }
}
```

Output:

```
2
```

Performance:

```
Time complexity of the solution is O(n) as it traverses the tree only once.
```

Graphs & Binary Tree Algorithm Questions

Question: Given a undirected graph with corresponding edges. Find the number of possible triangles?

Example

```
0 1
2 1
0 2
4 1
```

Full implementation in Java

```
 class Vertex { List Adjacents; }

        public static int GetNumberOfTriangles(Vertex source)
        {
            int count = 0;

            Queue queue = new Queue();
            HashSet visited = new HashSet();

            queue.Enqueue(source);

            while (!queue.IsEmpty())
            {
                Vertex current = queue.Dequeue();

                // get all non-visited adjacent vertices for current vertex
                List adjacents = current.Adjacents
                                            .Where(v => !visited.Contains(v))
                                            .ToList();

                while (!adjacents.IsEmpty())
                {
                    Vertex curr = adjacents.First();

                    // count the number of common vertices
                    //     adjacents.Contains(c)  => common vertex
                    //     c != curr    => avoid counting itself */
                    //     !visited.Contains(c) => we haven't counted this yet
                    count += curr.Adjacents
                            .Select(c => adjacents.Contains(c)
                                                 && c != curr
                                                 && !visited.Contains(c)
                                     ).Count();

                    // remove the vertex to avoid counting it again in next
iteration
                    adjacents.Remove(curr);

                    queue.Enqueue(curr);
                }

                // Mark the vertex as visited
                visited.Add(current);
            }

            return count;
        }
```

Question: Find Maximum Path Sum in a Binary Tree in Java?

Given a binary tree, find the maximum path sum. The path may start and end at any node in the tree. **Example:**

```
Input: Root of below tree
       1
      /
     2   3
Output: 6

See below diagram for another example.
1+2+3
```

Analysis of the solution:

```
For each node there can be four ways that the max path goes through the node:
 1. Node only
 2. Max path through Left Child + Node
 3. Max path through Right Child + Node
 4. Max path through Left Child + Node + Max path through Right Child

The idea is to keep trace of four paths and pick up the max one in the end. An
important thing to note is, root of every subtree need to return maximum path sum
such that at most one child of root is involved. This is needed for parent function
call. In below code, this sum is stored in ‘max_single’ and returned by the
recursive function.
```

Full Implementation in Java

```
// Java program to find maximum path sum in Binary Tree

/* Class containing left and right child of current
 node and key value*/
class Node {

    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

// An object of Res is passed around so that the
// same value can be used by multiple recursive calls.
class Res {
    public int val;
}

class BinaryTree {

    // Root of the Binary Tree
    Node root;
```

```
// This function returns overall maximum path sum in 'res'
// And returns max path sum going through root.
int findMaxUtil(Node node, Res res)
{

    // Base Case
    if (node == null)
        return 0;

    // l and r store maximum path sum going through left and
    // right child of root respectively
    int l = findMaxUtil(node.left, res);
    int r = findMaxUtil(node.right, res);

    // Max path for parent call of root. This path must
    // include at-most one child of root
    int max_single = Math.max(Math.max(l, r) + node.data,
                              node.data);


    // Max Top represents the sum when the Node under
    // consideration is the root of the maxsum path and no
    // ancestors of root are there in max sum path
    int max_top = Math.max(max_single, l + r + node.data);

    // Store the Maximum Result.
    res.val = Math.max(res.val, max_top);

    return max_single;
}

int findMaxSum() {
    return findMaxSum(root);
}

// Returns maximum path sum in tree with given root
int findMaxSum(Node node) {

    // Initialize result
    // int res2 = Integer.MIN_VALUE;
    Res res = new Res();
    res.val = Integer.MIN_VALUE;

    // Compute and return result
    findMaxUtil(node, res);
    return res.val;
}

/* Driver program to test above functions */
public static void main(String args[]) {
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(10);
    tree.root.left = new Node(2);
    tree.root.right = new Node(10);
    tree.root.left.left = new Node(20);
    tree.root.left.right = new Node(1);
```

```
        tree.root.right.right = new Node(-25);
        tree.root.right.right.left = new Node(3);
        tree.root.right.right.right = new Node(4);
        System.out.println("maximum path sum is : " +
                            tree.findMaxSum());
    }
}
```

```
Output:
 Max path sum is 42
```

Performance

```
Time Complexity: O(n) where n is number of nodes in Binary Tree.
```

Question: A very large binary number

Given a very large binary number which cannot be stored in a variable, determine the remainder of the decimal equivalent of the binary number when divided by 3. Generalize to find the remainder for any number k.

Full Implementation in Java

```
class Main {
  public static void main(String[] args) {
    BigInteger num = new
BigInteger("96206803646456343318453867262626092069820682068202813408108014111117937 59");

    String s = num.toString(2);
    System.out.println(s);

    for (int i = 2; i <= 999; i++) {
      String str1 = num.mod(new BigInteger(Integer.toString(i))).toString();
      String str2 = "" + mod(s, i);
      if (!str1.equals(str2)) {
        System.out.println(i + ":t" + str1 + "t" + str2);
      }
    }
  }

  private static int mod(String num, int k) {
    final long maxValModK = (1L<<;62) % k; int times = 0; int pos = num.length()-1;
int res = 0; while (pos >= 0) {
      long factor = 1;
      long x = 0;
      while (pos >= 0 && factor != 1L<<;62) {
        if (num.charAt(pos--) == '1') {
          x |= factor;
        }
        factor <<= 1;
      }

      x %= k;
      for (int i = 0; i < times; i++) {
        x = (x * maxValModK) % k;
      }

      res += x;
      res %= k;

      times++;
    }
    return res;
  }
}
```

Question: Find the Maximum Number Of Distinct Nodes in a Binary Tree Path.

Full Implementation in java

```
public static int getDisCnt(Tree root){
  Set uniq = new HashSet<>();
  if(root == null){
   return 0;
  }
  return getMaxHelper(root, uniq);
 }

 private static int getMaxHelper(Tree root, Set uniq){
  if(root == null){
   return uniq.size();
  }
  int l = 0;
  int r  = 0;
  if(uniq.add(root.data)){
   l = getMaxHelper(root.left, uniq);
   r = getMaxHelper(root.right, uniq);
   uniq.remove(uniq.size()-1);
  }
  else{
   l = getMaxHelper(root.left, uniq);
   r = getMaxHelper(root.right, uniq);
  }
  return Math.max(l, r);
```

Question: Maximum Path Sum in a Binary Tree.

Given a binary tree, find the maximum path sum. The path may start and end at any node in the tree. **Example:**

```
Input: Root of below tree
       1
      /
     2   3
Output: 6

See below diagram for another example.
1+2+3
```

Analysis of the solution:

```
For each node there can be four ways that the max path goes through the node:
 1. Node only
 2. Max path through Left Child + Node
 3. Max path through Right Child + Node
 4. Max path through Left Child + Node + Max path through Right Child

The idea is to keep trace of four paths and pick up the max one in the end. An
important thing to note is, root of every subtree need to return maximum path sum
such that at most one child of root is involved. This is needed for parent function
call. In below code, this sum is stored in ‘max_single’ and returned by the
recursive function.
```

Full Implementation in Java

```java
// Java program to find maximum path sum in Binary Tree

/* Class containing left and right child of current
 node and key value*/
class Node {

    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

// An object of Res is passed around so that the
// same value can be used by multiple recursive calls.
class Res {
    public int val;
}

class BinaryTree {

    // Root of the Binary Tree
    Node root;

    // This function returns overall maximum path sum in 'res'
    // And returns max path sum going through root.
    int findMaxUtil(Node node, Res res)
    {

        // Base Case
        if (node == null)
            return 0;

        // l and r store maximum path sum going through left and
        // right child of root respectively
        int l = findMaxUtil(node.left, res);
        int r = findMaxUtil(node.right, res);

        // Max path for parent call of root. This path must
        // include at-most one child of root
        int max_single = Math.max(Math.max(l, r) + node.data,
                                  node.data);


        // Max Top represents the sum when the Node under
        // consideration is the root of the maxsum path and no
        // ancestors of root are there in max sum path
        int max_top = Math.max(max_single, l + r + node.data);

        // Store the Maximum Result.
        res.val = Math.max(res.val, max_top);

        return max_single;
    }

    int findMaxSum() {
        return findMaxSum(root);
    }

    // Returns maximum path sum in tree with given root
    int findMaxSum(Node node) {

        // Initialize result
        // int res2 = Integer.MIN_VALUE;
        Res res = new Res();
        res.val = Integer.MIN_VALUE;

        // Compute and return result
        findMaxUtil(node, res);
        return res.val;
    }

    /* Driver program to test above functions */
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(2);
        tree.root.right = new Node(10);
        tree.root.left.left = new Node(20);
        tree.root.left.right = new Node(1);
        tree.root.right.right = new Node(-25);
        tree.root.right.right.left = new Node(3);
        tree.root.right.right.right = new Node(4);
        System.out.println("maximum path sum is : " +
                            tree.findMaxSum());
    }
}
```

```
Output:
 Max path sum is 42
```

Performance

Time Complexity: O(n) where n is number of nodes in Binary Tree.

Question: Print Nodes in Top View of Binary Tree.

Top view of a binary tree is the set of nodes visible when the tree is viewed from the top. Given a binary tree, print the top view of it.

The output nodes can be printed in any order. Expected time complexity is O(n)

A node x is there in output if x is the topmost node at its horizontal distance.

Horizontal distance of left child of a node x is equal to horizontal distance of x minus 1, and that of right child is horizontal distance of x plus 1.

Example

```
    1
    /
   2       3
  /      /
 4    5  6   7
Top view of the above binary tree is
4 2 1 3 7

        1
      /
    2       3

        4

            5

               6
Top view of the above binary tree is
2 1 3 6
```

Analysis of the solution:

```
We need to nodes of same horizontal distance together. We do a level order traversal
so that
the topmost node at a horizontal node is visited before any other node of same
horizontal
distance below it. Hashing is used to check if a node at given horizontal distance
is seen or not.
```

Full Implementation in Java

```
// Java program to print top view of Binary tree
import java.util.*;

// Class for a tree node
class TreeNode
```

```
{
    // Members
    int key;
    TreeNode left, right;

    // Constructor
    public TreeNode(int key)
    {
        this.key = key;
        left = right = null;
    }
}

// A class to represent a queue item. The queue is used to do Level
// order traversal. Every Queue item contains node and horizontal
// distance of node from root
class QItem
{
   TreeNode node;
   int hd;
   public QItem(TreeNode n, int h)
   {
        node = n;
        hd = h;
   }
}

// Class for a Binary Tree
class Tree
{
    TreeNode root;

    // Constructors
    public Tree()  { root = null; }
    public Tree(TreeNode n) { root = n; }

    // This method prints nodes in top view of binary tree
    public void printTopView()
    {
        // base case
        if (root == null) {  return;  }

        // Creates an empty hashset
        HashSet set = new HashSet<>();

        // Create a queue and add root to it
        Queue Q = new LinkedList();
        Q.add(new QItem(root, 0)); // Horizontal distance of root is 0

        // Standard BFS or level order traversal loop
        while (!Q.isEmpty())
        {
            // Remove the front item and get its details
            QItem qi = Q.remove();
            int hd = qi.hd;
            TreeNode n = qi.node;
            // If this is the first node at its horizontal distance,
            // then this node is in top view
            if (!set.contains(hd))
            {
                set.add(hd);
                System.out.print(n.key + " ");
            }

            // Enqueue left and right children of current node
            if (n.left != null)
                Q.add(new QItem(n.left, hd-1));
            if (n.right != null)
                Q.add(new QItem(n.right, hd+1));
        }
    }
}

// Driver class to test above methods
public class Main
{
    public static void main(String[] args)
    {
        /* Create following Binary Tree
             1
           /
          2    3

            4

              5

                6*/
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.right = new TreeNode(4);
        root.left.right.right = new TreeNode(5);
        root.left.right.right.right = new TreeNode(6);
        Tree t = new Tree(root);
        System.out.println("Following are nodes in top view of Binary Tree");
        t.printTopView();
    }
}
```

Output

```
Following are nodes in top view of Binary Tree
1 2 3 6
```

Performance

Time Complexity of the above implementation is O(n) where n is number of nodes in given binary tree. The assumption here is that add() and contains() methods of HashSet work in O(1) time.

Question: You have k lists of sorted integers. Find the smallest range that includes at least one number from each of the k lists.

Example:

List 1: [4, 10, 15, 24, 26] List 2: [0, 9, 12, 20] List 3: [5, 18, 22, 30] The smallest range here would be [20, 24] as it contains 24 from list 1, 20 from list 2, and 22 from list 3.

Full Implementation in Java:

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.PriorityQueue;
import java.util.SortedSet;
import java.util.TreeSet;

public class GoogleProblem {

 public static void main(String[] args) {

  List<List> lists = new ArrayList<List>();
  List list1 = new ArrayList();
  list1.add(4);
  list1.add(10);
  list1.add(15);
  list1.add(24);
  list1.add(26);
  List list2 = new ArrayList();
  list2.add(0);
  list2.add(9);
  list2.add(12);
  list2.add(20);
  List list3 = new ArrayList();
  list3.add(5);
  list3.add(18);
  list3.add(22);
  list3.add(30);

  lists.add(list1);
  lists.add(list2);
  lists.add(list3);

  Result result = findCoveringRange(lists);
  System.out.println(result.startRange + ", " + result.endRange);
 }

 public static Result findCoveringRange(List<List> lists) {
  Result result = null;

  int start = -1, end = -1;
  int rDiff = Integer.MAX_VALUE;
  int k = lists.size();

  PriorityQueue pQueue = new PriorityQueue();
  SortedSet entries = new TreeSet();
  Map<Integer, Data> listNoAndEntry = new HashMap<Integer, Data>();

  for (int i = 0; i < k; i++) pQueue.add(new Data(lists.get(i).remove(0), i)); while
(!pQueue.isEmpty()) { Data minData = pQueue.remove(); if
(lists.get(minData.listNo).size() > 0)
    pQueue.add(new Data(lists.get(minData.listNo).remove(0),
      minData.listNo));

   if (listNoAndEntry.size() == k) {

    Data first = entries.first();
    if ((entries.last().data - first.data) + 1 < rDiff) {
     start = first.data;
     end = entries.last().data;
    }
    entries.remove(first);
    listNoAndEntry.remove(first.listNo);
   }

   if (listNoAndEntry.containsKey(minData.listNo))
    entries.remove(listNoAndEntry.remove(minData.listNo));

   listNoAndEntry.put(minData.listNo, minData);
   entries.add(minData);
  }

  if (listNoAndEntry.size() == k) {

   Data first = entries.first();
   if ((entries.last().data - first.data) + 1 < rDiff) {
    start = first.data;
    end = entries.last().data;
   }
   entries.remove(first);
   listNoAndEntry.remove(first.listNo);
  }

  result = new Result(start, end);
  return result;
 }

}

class Result {
 public final int startRange, endRange;

 public Result(int startRange, int endRange) {
  this.startRange = startRange;
  this.endRange = endRange;
 }
}
```

```
class Data implements Comparable {
 public final int data;
 public final int listNo;

 public Data(int data, int listNo) {
  this.data = data;
  this.listNo = listNo;
 }

 @Override
 public int compareTo(Data o) {
  // TODO Auto-generated method stub
  return data - o.data;
 }
}
```

Java String Algorithm Questions

Question: A k-palindrome is a string which transforms into a palindrome on removing at most k characters.

Example:

```
Given a string S, and an interger K, print "YES" if S is a k-palindrome; otherwise
print "NO".
Constraints:
S has at most 20,000 characters.
0<=k<=30

Sample Test Case#1:
Input - abxa 1
Output - YES
Sample Test Case#2:
Input - abdxa 1
Output - No
```

Full Implementation in Java

```
int ModifiedEditDistance(const string& a, const string& b, int k) {
 int i, j, n = a.size();
 // for simplicity. we should use only a window of size 2*k+1 or
 // dp[2][MAX] and alternate rows. only need row i-1
 int dp[MAX][MAX];
 memset(dp, 0x3f, sizeof dp); // init dp matrix to a value > 1.000.000.000
 for (i = 0 ; i < n; i++)
  dp[i][0] = dp[0][i] = i;

 for (i = 1; i <= n; i++) {
  int from = max(1, i-k), to = min(i+k, n);
  for (j = from; j <= to; j++) {
   if (a[i-1] == b[j-1])   // same character
    dp[i][j] = dp[i-1][j-1];
   // note that we don't allow letter substitutions

   dp[i][j] = min(dp[i][j], 1 + dp[i][j-1]); // delete character j
   dp[i][j] = min(dp[i][j], 1 + dp[i-1][j]); // insert character i
  }
 }
 return dp[n][n];
}
cout << ModifiedEditDistance("abxa", "axba", 1) << endl;  // 2 <= 2*1 - YES
cout << ModifiedEditDistance("abdxa", "axdba", 1) << endl; // 4 > 2*1 - NO
cout << ModifiedEditDistance("abaxbabax", "xababxaba", 2) << endl; // 4 <= 2*2 - YES
```

Question: Find duplicate characters in a String

How to find duplicate characters in a string ?

We will discuss two approaches to count and find the duplicate characters in a string

1.By using HashMap

In this way we will get the character array from String, iterate through that and build a Map with character and their count.

Then iterate through that Map and print characters which have appeared more than once.

So you actually need two loops to do the job, the first loop to build the map and second loop to print characters and counts.

```
package codespaghetti.com;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;
public class FindDuplicateCharacters{

    public static void main(String args[]) {
        printDuplicateCharacters("Programming");
        printDuplicateCharacters("Combination");
        printDuplicateCharacters("Java");
    }

    /*
     * Find all duplicate characters in a String and print each of them.
     */
    public static void printDuplicateCharacters(String word) {
        char[] characters = word.toCharArray();

        // build HashMap with character and number of times they appear in String
        Map<Character, Integer> charMap = new HashMap<Character, Integer>();
        for (Character ch : characters) {
            if (charMap.containsKey(ch)) {
                charMap.put(ch, charMap.get(ch) + 1);
            } else {
                charMap.put(ch, 1);
            }
        }

        // Iterate through HashMap to print all duplicate characters of String
        Set<Map.Entry<Character, Integer>> entrySet = charMap.entrySet();
        System.out.printf("List of duplicate characters in String '%s' %n", word);
        for (Map.Entry<Character, Integer> entry : entrySet) {
            if (entry.getValue() > 1) {
                System.out.printf("%s : %d %n", entry.getKey(), entry.getValue());
            }
        }
    }

}

Output
List of duplicate characters in String 'Programming'
g : 2
r : 2
m : 2
List of duplicate characters in String 'Combination'
n : 2
o : 2
i : 2
List of duplicate characters in String 'Java'
```

2. By using BufferReader

```
package codespaghetti.com;
import java.io.*;
public class CountChar
{

    public static void main(String[] args) throws IOException
    {
      String ch;
      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.print("Enter the Statement:");
      ch=br.readLine();
      int count=0,len=0;
        do
        {
          try
          {
          char name[]=ch.toCharArray();
              len=name.length;
              count=0;
              for(int j=0;j<len;j++) { if((name[0]==name[j])&&((name[0]>=65&&name[0]
<=91)||(name[0]>=97&&name[0]<=123)))
                      count++;
               }
              if(count!=0)
                System.out.println(name[0]+" "+count+" Times");
              ch=ch.replace(""+name[0],"");
          }
          catch(Exception ex){}
        }
        while(len!=1);
   }

}
Output

Enter the Statement:asdf23123sfsdf

a 1 Times

s 3 Times

d 2 Times

f 3 Times
```

Question: Find longest Substring that contains 2 unique characters

---

This is a problem asked by Google

Given a string, find the longest substring that contains only two unique characters. For example, given "abcbbbbcccbdddadacb", the longest substring that contains 2 unique character is "bcbbbbcccb".

1. Longest Substring Which Contains 2 Unique Characters

In this solution, a hashmap is used to track the unique elements in the map. When a third character is added to the map, the left pointer needs to move right.

You can use "abac" to walk through this solution.

```
public int lengthOfLongestSubstringTwoDistinct(String s) {
    int max=0;
    HashMap<Character,Integer> map = new HashMap<Character, Integer>();
    int start=0;

    for(int i=0; i<s.length(); i++){ char c = s.charAt(i); if(map.containsKey(c)){
map.put(c, map.get(c)+1); }else{ map.put(c,1); } if(map.size()>2){
            max = Math.max(max, i-start);

            while(map.size()>2){
                char t = s.charAt(start);
                int count = map.get(t);
                if(count>1){
                    map.put(t, count-1);
                }else{
                    map.remove(t);
                }
                start++;
            }
        }
    }

    max = Math.max(max, s.length()-start);

    return max;
}
```

Now if this question is extended to be "the longest substring that contains k unique characters", what should we do?

2. Solution for K Unique Characters

```
public int lengthOfLongestSubstringKDistinct(String s, int k) {
if(k==0 || s==null || s.length()==0)
return 0;

if(s.length()<k)
return s.length();

HashMap<Character, Integer> map = new HashMap<Character, Integer>();

int maxLen=k;
int left=0;
for(int i=0; i<s.length(); i++){ char c = s.charAt(i); if(map.containsKey(c)){
map.put(c, map.get(c)+1); }else{ map.put(c, 1); } if(map.size()>k){
maxLen=Math.max(maxLen, i-left);

while(map.size()>k){

char fc = s.charAt(left);
if(map.get(fc)==1){
map.remove(fc);
}else{
map.put(fc, map.get(fc)-1);
}

left++;
}
}

}

maxLen = Math.max(maxLen, s.length()-left);

return maxLen;
}
```

Time is O(n).

Question: Find substring with concatenation of all the words in Java

You are given a string, s, and a list of words, words, that are all of the same length. Find all starting indices of substring(s) in s that is a concatenation of each word in words exactly once and without any intervening characters.

For example, given: s="barfoothefoobarman" & words=["foo", "bar"], return [0,9].

Analysis

This problem is similar (almost the same) to Longest Substring Which Contains 2 Unique Characters.

Since each word in the dictionary has the same length, each of them can be treated as a single character.

Java Implementation

```
public List findSubstring(String s, String[] words) {
    ArrayList result = new ArrayList();
    if(s==null||s.length()==0||words==null||words.length==0){
        return result;
    }

    //frequency of words
    HashMap<String, Integer> map = new HashMap<String, Integer>();
    for(String w: words){
        if(map.containsKey(w)){
            map.put(w, map.get(w)+1);
        }else{
            map.put(w, 1);
        }
    }

    int len = words[0].length();

    for(int j=0; j<len; j++){
        HashMap<String, Integer> currentMap = new HashMap<String, Integer>();
        int start = j;//start index of start
        int count = 0;//count totoal qualified words so far

        for(int i=j; i<=s.length()-len; i=i+len){ String sub = s.substring(i,
i+len); if(map.containsKey(sub)){ //set frequency in current map
if(currentMap.containsKey(sub)){ currentMap.put(sub, currentMap.get(sub)+1); }else{
currentMap.put(sub, 1); } count++; while(currentMap.get(sub)>map.get(sub)){
                    String left = s.substring(start, start+len);
                    currentMap.put(left, currentMap.get(left)-1);

                    count--;
                    start = start + len;
                }


                if(count==words.length){
                    result.add(start); //add to result

                    //shift right and reset currentMap, count & start point
                    String left = s.substring(start, start+len);
                    currentMap.put(left, currentMap.get(left)-1);
                    count--;
                    start = start + len;
                }
            }else{
                currentMap.clear();
                start = i+len;
                count = 0;
            }
        }
    }

    return result;
}
```

Question: Find the shortest substring from the alphabet "abc".

Given an input string "aabbccba", find the shortest substring from the alphabet "abc".

```
In the above example, there are these substrings "aabbc", "aabbcc", "ccba" and
"cba". However the shortest substring that contains all the characters in the
alphabet is "cba", so "cba" must be the output.

Output doesnt need to maintain the ordering as in the alphabet.

Other examples:
input = "abbcac", alphabet="abc" Output : shortest substring = "bca".
```

Full Implementation:

```
public class Solution {
    public String minWindow(String s, String t) { // s is the string and t is
alphabet
        int[] map = new int[256];
        int begin=0,end=0; // for substring window
        int head = begin; // for getting the output substring

        for(int i=0;i<t.length();i++) { // fill the map with freq of chars in t
            map[t.charAt(i)]++;
        }

        int count = t.length(); // size of t as we have to have this count check
        int min=Integer.MAX_VALUE;

        while(end<s.length()) { // System.out.println(s.charAt(end) + "t" +
map[s.charAt(end)]); // System.out.println("Step
1t"+count+"t"+begin+"t"+end+"t"+head+"t"+min); if(map[s.charAt(end++)]-->;0) { // if
it is present in map decrease count
                count--;
            }
            // System.out.println("Step
2t"+count+"t"+begin+"t"+end+"t"+head+"t"+min);
            while(count==0) { // t is found in s
                if(end-begin<min) { // update min and head
                    min = end-begin;
                    head = begin;
                }
                if(map[s.charAt(begin++)]++==0) { // shrink the window
                    count++;
                }
            }
            // System.out.println("Step
3t"+count+"t"+begin+"t"+end+"t"+head+"t"+min);
        }

        return min==Integer.MAX_VALUE ? "" : s.substring(head,head+min);
    }
}
```

Question: Given s string, Find max size of a sub-string, in which no duplicate chars present.

Full Implementation

```
public static String longestSubstringUnrepeatedChar(String inputString) {
        String longestSoFar = "";
        String longestSubstringResult = "";
        if (inputString.isEmpty()) {
            return "";
        }
        if (inputString.length() == 1) {
            return inputString;
        }
        Map<Character, Integer> map = new HashMap<Character, Integer>();
        for (int i = 0; i < inputString.length(); i++) { char currentCharacter =
inputString.charAt(i); if (longestSoFar.indexOf(currentCharacter) == -1) { if
(!map.containsKey(currentCharacter)) { map.put(currentCharacter, i); } longestSoFar
= longestSoFar + currentCharacter; } else { longestSoFar =
inputString.substring(map.get(currentCharacter) + 1, i + 1);
map.put(currentCharacter, i); } if (longestSoFar.length() >
longestSubstringResult.length()) {
                longestSubstringResult = longestSoFar;
            }
        }
        return longestSubstringResult;
    }
```

Question: Given s string, Find max size of a sub-string, in which no duplicate chars present.

Full Implementation

```
public static String longestSubstringUnrepeatedChar(String inputString) {
        String longestSoFar = "";
        String longestSubstringResult = "";
        if (inputString.isEmpty()) {
            return "";
        }
        if (inputString.length() == 1) {
            return inputString;
        }
        Map<Character, Integer> map = new HashMap<Character, Integer>();
        for (int i = 0; i < inputString.length(); i++) { char currentCharacter =
inputString.charAt(i); if (longestSoFar.indexOf(currentCharacter) == -1) { if
(!map.containsKey(currentCharacter)) { map.put(currentCharacter, i); } longestSoFar
= longestSoFar + currentCharacter; } else { longestSoFar =
inputString.substring(map.get(currentCharacter) + 1, i + 1);
map.put(currentCharacter, i); } if (longestSoFar.length() >
longestSubstringResult.length()) {
                longestSubstringResult = longestSoFar;
            }
        }
        return longestSubstringResult;
    }
```

Question: How to Reverse Words in a String in Java?

How to reverse words in a string in Java?

String manipulation is an important topic from interview point of view. Chances are you will be asked about a question regarding the string manipulation. Reversing a string is one of the most popular question.

Problem

You are given a string of words and you are required to write a programme which will reverse the string

The input string does not contain leading or trailing spaces and the words are always separated by a single space. **For example** Given string = "This is a test question". And after the reversal it should look like this Reversed string = "question test a is This" Could you do it in-place without allocating extra space?

Analysis of the problem

There are many solutions to reverse the string we will go through all of them one by one

1.Most simple solution to reverse the string?

The easiest way to reverse a string is by using StringBuffer reverse() function in java. But this is not enough. Interviewers will ask you to write your own function to reverse the string.

2. Reverse string by using Recursion ?

```
public static void main(String args[]) throws FileNotFoundException, IOException {

//original string

String str = "This is a test question";

System.out.println("Original String: " + str);

     //recursive method to reverse String in Java

String reversedString = reverseRecursively(str);

System.out.println("Reversed by using Recursion: " +reversedString);

}

public static String reverseRecursively(String str) {

//base case to handle one char string and empty string

if (str.length() < 2) {

return str;

}

return reverseRecursively(str.substring(1)) + str.charAt(0);

}
```

3. Reverse string by using Iteration ?

```
public static void main(String args[]) throws FileNotFoundException, IOException {

//original string

String str = "This is a test question";

System.out.println("Original String: " + str);

     //recursive method to reverse String in Java

String reversedString = reverseRecursively(str);

System.out.println("Reversed by using Recursion: " +reversedString);

}
public static String reverse(String str) {

StringBuilder strBuilder = new StringBuilder();

char[] strChars = str.toCharArray();

for (int i = strChars.length - 1; i >= 0; i--) {

strBuilder.append(strChars[i]);

}

return strBuilder.toString();

}
```

4. Reverse string by using while loop ?

```
package codespaghetti.com;

public void reverseString(char[] inputString) {
    int i=0;
    for(int j=0; j<inputString.length; j++){
        if(inputString[j]==' '){
            reverse(inputString, i, j-1);
            i=j+1;
        }
    }

    reverse(inputString, i, inputString.length-1);

    reverse(inputString, 0, inputString.length-1);
}

public void reverse(char[] inputString, int i, int j){
    while(i<j){
        char temp = inputString[i];
        inputString[i]=inputString[j];
        inputString[j]=temp;
        i++;
        j--;
    }
}
```

Question: If a=1, b=2, c=3,....z=26. Given a string, find all possible codes that string can generate. Give a count as well as print the strings.

Example

Input: "1123". You need to general all valid alphabet codes from this string.Output List aabc //a = 1, a = 1, b = 2, c = 3 kbc // since k is 11, b = 2, c= 3 alc // a = 1, l = 12, c = 3 aaw // a= 1, a =1, w= 23 kw // k = 11, w = 23 Full Implementation: public Set decode(String prefix, String code) { Set set = new HashSet(); if (code.length() == 0) { set.add(prefix); return set; } if (code.charAt(0) == '0') return set; set.addAll(decode(prefix + (char) (code.charAt(0) - '1' + 'a'), code.substring(1))); if (code.length() >= 2 && code.charAt(0) == '1') { set.addAll(decode(prefix + (char) (10 + code.charAt(1) - '1' + 'a'), code.substring(2))); } if (code.length() >= 2 && code.charAt(0) == '2' && code.charAt(1) <= '6') { set.addAll(decode(prefix + (char) (20 + code.charAt(1) - '1' + 'a'), code.substring(2))); } return set; }

Question: Implement strStr() function in Java.

strStr() function

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack. **Java Solution 1 - Naive**

```
public int strStr(String haystack, String
needle) {
    if(haystack==null || needle==null)
        return 0;

    if(needle.length() == 0)
        return 0;

    for(int i=0; i<haystack.length(); i++){
        if(i + needle.length() >
haystack.length())
            return -1;

        int m = i;
        for(int j=0; j<needle.length(); j++){

if(needle.charAt(j)==haystack.charAt(m)){
                if(j==needle.length()-1)
                    return i;
                m++;
            }else{
                break;
            }

        }
    }

    return -1;
}
```

Java Solution 2 - KMP Check out this article to understand KMP algorithm.

```
public int strStr(String haystack, String needle) {
        if(haystack==null || needle==null)
            return 0;

 int h = haystack.length();
 int n = needle.length();

 if (n > h)
  return -1;
 if (n == 0)
  return 0;

 int[] next = getNext(needle);
 int i = 0;

 while (i <= h - n) {
  int success = 1;
  for (int j = 0; j < n; j++) {
   if (needle.charAt(0) != haystack.charAt(i)) {
    success = 0;
    i++;
    break;
   } else if (needle.charAt(j) != haystack.charAt(i + j)) {
    success = 0;
    i = i + j - next[j - 1];
    break;
   }
  }
  if (success == 1)
   return i;
 }

 return -1;
}

//calculate KMP array
public int[] getNext(String needle) {
 int[] next = new int[needle.length()];
 next[0] = 0;

 for (int i = 1; i < needle.length(); i++) {
  int index = next[i - 1];
  while (index > 0 && needle.charAt(index) !=
needle.charAt(i)) {
   index = next[index - 1];
  }

  if (needle.charAt(index) == needle.charAt(i)) {
   next[i] = next[i - 1] + 1;
  } else {
   next[i] = 0;
  }
 }

 return next;
}
```

Question: List of string that represent class names in CamelCaseNotation.

Write a function that given a List and a pattern returns the matching elements.

```
['HelloMars', 'HelloWorld', 'HelloWorldMars', 'HiHo']

H -> [HelloMars, HelloWorld, HelloWorldMars, HiHo]
HW -> [HelloWorld, HelloWorldMars]
Ho -> []
HeWorM -> [HelloWorldMars]
```

Full Implementation;

```
package codespaghetti.com;

import java.util.ArrayDeque;
import java.util.Arrays;
import java.util.List;
import java.util.Queue;
import java.util.stream.Collectors;

public class CamelCaseNotation
{

 private String[] camelCaseWords = { "HelloMars", "HelloWorld", "HelloWorldMars",
"HiHo" };

 public List testCamelCase(final String pattern)
 {
  return Arrays.stream(camelCaseWords).filter(word -> isMatchingCamelCase(pattern,
word)).collect(Collectors.toList());
 }

 private Queue toQueue(final String word)
 {
  final Queue queue = new ArrayDeque<>(word.length());
  for (final char ch : word.toCharArray())
  {
   queue.add(String.valueOf(ch));
  }
  return queue;
 }

 private boolean isMatchingCamelCase(final String pattern, final String word)
 {
  if (pattern.length() > word.length())
  {
   return false;
  }

  final Queue patternQueue = toQueue(pattern);
  final Queue wordQueue = toQueue(word);
  String ch = patternQueue.remove();
  while (!wordQueue.isEmpty())
  {
   if (ch.equals(wordQueue.remove()))
   {
    if (patternQueue.isEmpty())
    {
     return true;
    }
    ch = patternQueue.remove();
    continue;
   }

   if (!Character.isUpperCase(ch.charAt(0)))
   {
    return false;
   }
  }
  return false;
 }
}
```

Question: Print all the characters in string only once in a reverse order.

Print all the characters present in the given string only once in a reverse order. Time & Space complexity should not be more than O(N). e.g.

1. Given a string aabdceaaabbbcd the output should be - dcbae
2. Sample String - aaaabbcddddccbbdaaeee Output - eadbc
3. I/P - aaafffcccddaabbeeddhhhaaabbccddaaaa O/P - adcbhef

Full Implementation

```
public class StringReverse {

    public static void main(String[] args) {
        String input = "aabdceaaabbbcd";
        reverseWithOneCharacterOnce(input);
    }

    private static void reverseWithOneCharacterOnce(String input) {

        Set alreadyPrintedCharacter = new HashSet();
        String reversed = "";

        for (int index = input.length() - 1; index >= 0; index--) {
            Character ch = input.charAt(index);
            if (!alreadyPrintedCharacter.contains(ch)) {
                alreadyPrintedCharacter.add(ch);
                reversed = reversed + ch;
            }
        }

        System.out.println(reversed);

    }

}
```

Question: Print out all of the unique characters and the number of times it appeared in the string in Java?

Pseudo code for printing unique numbers and their frequency

```
s = "some string with repeated characters"
map<char, int> m

for each c in some
 m[c]++

for each e in m
 print e.value
```

Full Implementation

```
public class DuplicateFrequency {

 public static void main(String[] args) {

  String str = "I am preparing for interview";

  char [] strArray = str.toCharArray();

  int [] ar = {1,16,2,3,3,4,4,8,6,5,4};

  Map<Character, Integer> map = new TreeMap<Character, Integer>();

  for (int i=0;  i System.out.println ("key ="+ k + ", No. of time Repeated "+ v) );

 }
```

Question: Rearrange characters in a string to form a lexicographically first palindromic string

---

Isomorphic Strings in Java

Given two strings s and t, determine if they are isomorphic. Two strings are isomorphic if the characters in s can be replaced to get t.
For example,"egg" and "add" are isomorphic, "foo" and "bar" are not. **Analysis** We can define a map which tracks the char-char mappings. If a value is already mapped, it can not be mapped again. **Java Solution**

```
public boolean isIsomorphic(String s, String t) {
    if(s==null||t==null)
        return false;

    if(s.length()!=t.length())
        return false;

    HashMap<Character, Character> map = new HashMap<Character,
Character>();


    for(int i=0; i<s.length(); i++){
        char c1 = s.charAt(i);
        char c2 = t.charAt(i);

        if(map.containsKey(c1)){
            if(map.get(c1)!=c2)// if not consistant with previous ones
                return false;
        }else{
            if(map.containsValue(c2)) //if c2 is already being mapped
                return false;
            map.put(c1, c2);
        }
    }

    return true;
}
```

Time is O(n).

Question: Shuffle the string so that no two similar letters are together.

Given a string e.g. ABCDAABCD. Shuffle he string so that no two smilar letters together. E.g. AABC can be shuffled as ABAC.

```
void ScrambleString( string &inputString, int iter = 4 )
{
    if( iter == 0 )
        return;
    cout << "Iteration : " << iter << endl;

    size_t strLen = inputString.size();
    for( int i = 1; i < strLen; i++ )
    {
        if( inputString[i-1] == inputString[i] )
        {
            for( int k = i+1; k < strLen; k++ )
            {
                if( inputString[i] == inputString[k] )
                    continue;

                char temp = inputString[i];
                inputString[i] = inputString[k];
                inputString[k] = temp;
                break;
            }
        }
    }
    iter--;
    bool iterateFlag = false;
    for( int i = 1; i < strLen; i++ )
    {
        if( inputString[i-1] == inputString[i] )
        {
            iterateFlag = true;
            break;
        }
    }

    if( iterateFlag )
    {
        std::reverse(inputString.begin(), inputString.end());
        ScrambleString(inputString, iter );
    }
    return;
}
```

Question: How To UnScramble a String in Java?

```
package codespaghetti.com;

import java.util.HashSet;
import java.util.Set;

public class GoogleInterviewQuestion {
 //swap positions. used for makeSentence() function
 public static String swapStringIndexes(String s, int i, int j){
  char[] arr=s.toCharArray();
  char dummy=arr[i];
  arr[i]=arr[j];
  arr[j]=dummy;
  return new String(arr);
 }
 //Generates permutations of string and returns a string if it is found in
dictionary or return ""
 public static String wordExists(String s,Set dictionary,int i,int j){
  if(i==j){
   if(dictionary.contains(s)){
    return s;
   }
  }else{
   for(int k=i;k<j;k++){
    String found=wordExists(swapStringIndexes(s, i, k),dictionary,i+1,j);
    if(dictionary.contains(found)){
     return found;
    }
   }
  }
  return "";
 }
 //return sentence if can be formed with the given string or return ""
 public static String makeSentence(String s,Set dictionary,String sentenceBuilder){
  if(s.isEmpty()){
   return sentenceBuilder; //sentenceBuilder;
  }else{
   for(int i=1;i<=s.length();i++){
    String first=s.substring(0,i);
    String second=s.substring(i);
    String foundWord=wordExists(first, dictionary, 0, i);
    if(!foundWord.isEmpty()){
     String sentence=makeSentence(second, dictionary, sentenceBuilder+foundWord+"
");
     if(!sentence.isEmpty()){
      return sentence;
     }
    }
   }
  }
  return "";
 }

 public static void main(String[] args) {
  Set dictionary=new HashSet<>();
  dictionary.add("hello");
  dictionary.add("he");
  dictionary.add("to");
  dictionary.add("the");
  dictionary.add("world");
  System.out.println(makeSentence("elhloothtedrowl", dictionary,""));
 }

}
```

Question: Write 2 functions to serialize and deserialize an array of strings in Java

Write 2 functions to serialize and deserialize an array of strings. Strings can contain any unicode character. Do not worry about string overflow.

Java implementation:

```
public class ArraySerializerDeserializer {

 public static String serialize(String[] a) {
  StringBuilder output = new StringBuilder();
  int maxLenght = 0;
  for (String s : a)
   if (s.length() > maxLenght)
    maxLenght = s.length();
  maxLenght++;
  output.append(maxLenght).append(":");
  String delimiter = generateRandString(maxLenght);
  for (String s : a)
   output.append(delimiter).append(s.length()).append(":").append(s);
  System.out.println(output.toString());
  return output.toString();
 }

 public static String[] deserialize(String s, int size) {
  String[] output = new String[size];
  StringBuilder sb = new StringBuilder();
  StringBuilder num = new StringBuilder();
  int i = 0;
  while (s.charAt(i) != ':') {
   num.append(s.charAt(i));
   i++;
  }
  i++;

  int maxWordSize = Integer.valueOf(num.toString());
  num = new StringBuilder();

  boolean parsingNum = false;
  boolean parsingDelimiter = true;
  int charCount = 0;
  int nextWordLenght = 0;
  int wordCount = 0;
  while (i < s.length()) {
   if (parsingDelimiter) {
    while (charCount < maxWordSize) { i++; charCount++; } parsingDelimiter = false;
parsingNum = true; charCount = 0; } else if (parsingNum) { while (s.charAt(i) !=
':') { num.append(s.charAt(i)); i++; } parsingNum = false; nextWordLenght =
Integer.valueOf(num.toString()); num = new StringBuilder(); // Emptying. i++; } else
{ while (nextWordLenght > 0) {
     sb.append(s.charAt(i));
     i++;
     nextWordLenght--;
    }
    parsingDelimiter = true;
    output[wordCount] = sb.toString();
    wordCount++;
    sb = new StringBuilder(); // Emptying.
   }
  }
  return output;
 }

 private static String generateRandString(int size) {
  StringBuilder sb = new StringBuilder();
  for (int i = 0; i < size; i++) {
   sb.append((char) (65 + (26 * Math.random())));
  }
  return sb.toString();
 }

 public static void main(String[] args) {
  String[] a = { "this", "is", "very", "nice", "I", "like" };
  String s = serialize(a);
  String[] output = deserialize(s, a.length);
  for (String out : output)
   System.out.print(out + " ");
 }

}
```

Arrays Algorithms Questions

Array algorithm questions

ArrayList Algorithm Questions

Arraylist algorithm questions

LinkedLists Algorithm Questions

Linkedlist algorithm questions

Keys to interview success

The ultimate secret of success in programming interviews is your ability to understand and master the most important algorithms.

Do not be one of the many who mistakenly believe that they have sufficient knowledge and grip on algorithms.

The world of algorithms, will always to some extent remain mysterious. It takes a serious effort to master them. If you are going for interview in a big company like, Google, Microsoft , Amazon or Facebook.

Then you must be prepare very thoroughly about these. You do not need some surface knowledge but very deep understanding of each aspects of algorithms.

In a world growing increasingly competent your survival in interview can only be ensured by the mastery of algorithms.

About The Author

References

- https://www.amazon.com/Data-Structures-Algorithms-Java-2nd/dp/0672324539
- http://introcs.cs.princeton.edu/java/40algorithms/
- https://en.wikipedia.org/wiki/Search_algorithm
- https://www.quora.com/
- https://betterexplained.com/articles/sorting-algorithms/
- http://www.javaworld.com/article/2073390/core-java/datastructures-and-algorithms-part-1.html
- https://www.quora.com/What-algorithm-questions-were-you-asked-at-an-Amazon-Microsoft-Google-interview
- https://www.quora.com/How-can-one-be-well-prepared-to-answer-data-structure-algorithm-questions-in-interviews