```java
package PocketGem;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;


public class Find_Path {
    public static List<String> parseLines(List<String> lines) {
        List<String> ans = new ArrayList<String>();

        if(lines == null || lines.size() == 0){
            return ans;
        }

        String[] array = lines.get(0).split(" ");
    String start = array[0];
        String end = array[1];
        Map<String, List<String>> graph = new HashMap<String, List<String>>();
        Set<String> visited = new HashSet<String>();

        for(int i = 1; i < lines.size(); i++) {
            String line = lines.get(i);
            String[] nodes = line.trim().split(":");
            List<String> list = Arrays.asList(nodes[1].trim().split("\\s{1,}"));
            graph.put(nodes[0].trim(), list);
        }

        String path = start;
        backtrack(ans, path, visited, graph, start, end);
        return ans;
    }

   public static void backtrack(List<String> ans, String path, Set<String> visited,
Map<String, List<String>> graph, String start, String target){
        if(visited.contains(start)){
            return ;
        } else if(start.equals(target)){
            ans.add(path);
            return ;
        } else if(!graph.containsKey(start)){
            return ;
        }

        visited.add(start);

        for(String str : graph.get(start)){
            backtrack(ans, path + str, visited, graph, str, target);
        }
```

```
        visited.remove(start);
    }


    /************************************* main
*****************************************/
    public static void main(String[] args) throws FileNotFoundException,
IOException {
        String filename = "input3.txt";
        if (args.length > 0) {
        filename = args[0];
        }

        List<String> answer = parseFile(filename);
        System.out.println(answer);
    }

    static List<String> parseFile(String filename) throws FileNotFoundException,
IOException {
    /*
     *  Don't modify this function
     */
        BufferedReader input = new BufferedReader(new FileReader(filename));
        List<String> allLines = new ArrayList<String>();
        String line;
        while ((line = input.readLine()) != null) {
        allLines.add(line);
        }
        input.close();

        return parseLines(allLines);
    }

//  public static void main(String[] args){
//      List<String> file = new ArrayList<String>();
//      file.add("A E");
//      file.add("A:B C D");
//      file.add("B:C");
//      file.add("C:E");
//      file.add("D:B");
//
//      List<String> ans = parseFile(file);
//
//      for(String path : ans){
//          System.out.println(path);
//      }
//  }
}
```

```java
package PocketGem;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.PriorityQueue;
import java.util.Queue;


public class TimePercentage {
    /*****************************************************************/
    // multi-user
    static String parseLines(String[] lines) {
        if(lines == null || lines.length == 0){
            return "";
        }

        Map<String, Integer> map = new HashMap<String, Integer>();
        map.put("START", 3);
        map.put("CONNECTED", 4);
        map.put("DISCONNECTED", 2);
        map.put("SHUTDOWN", 1);

        Queue<Pair> heap = new PriorityQueue<Pair>(1, new Comparator<Pair>() {
            public int compare(Pair left, Pair right){
                if(left.timeStamp != right.timeStamp){
                    if(left.timeStamp < right.timeStamp){
                        return -1;
                    } else if(left.timeStamp > right.timeStamp){
                        return 1;
                    } else {
                        return 0;
                    }
                } else {
                    return map.get(left.status) - map.get(right.status);
                }
            }
        });

        SimpleDateFormat formatter = new
SimpleDateFormat( "yyyy/MM/dd-HH:mm:ss");
        Date date = new Date();
        long startTime = 0, endTime = 0;


        for(String str : lines){
            String[] array = str.split("::");
            array[0] = array[0].trim();
```

```
            array[0] = array[0].substring(1, array[0].length() - 1);
            array[1] = array[1].trim();

            try {
                date =  formatter.parse(array[0]);
            } catch (ParseException e) {
                e.printStackTrace();
            }

            long timeStamp = date.getTime();
            heap.offer(new Pair(timeStamp, array[1]));

            if(array[1].equals("START")){
                startTime = timeStamp;
            }
            if(array[1].equals("SHUTDOWN")){
                endTime = timeStamp;
            }
        }

        long connectStart = -1;
        long connectTime = 0;
        boolean startFlag = false;
        int connectCount = 0;

        while(!heap.isEmpty()){
            Pair node = heap.poll();

            if(startFlag == false && !node.status.equals("START")){
                continue;
            } else if(node.status.equals("START")){
                startFlag = true;
            } else if(node.status.equals("SHUTDOWN")){
                if(connectStart != -1){
                    connectTime += node.timeStamp - connectStart;
                }
                break;
            } else if(node.status.equals("CONNECTED")){
                if(connectCount == 0){
                    connectStart = node.timeStamp;
                }
                connectCount++;
            } else if(node.status.equals("DISCONNECTED")){
                connectCount--;

                if(connectCount == 0){
                    connectTime += node.timeStamp - connectStart;
                    connectStart = -1;
                }
            }
        }

        double ratio = (double) connectTime / (endTime - startTime) * 100;
        return String.format("%d%s", (int) ratio, "%");
    }

    static class Pair {
        long timeStamp;
```

```
        String status;

        public Pair(long timeStamp, String status){
            this.timeStamp = timeStamp;
            this.status = status;
        }
    }



    /*************************************************************/
    // single-user
    static String parseLines2(String[] lines) {
    if (lines == null || lines.length == 0) {
            return "";
        }

        Map<String, Integer> status = new HashMap<String, Integer>();
        status.put("START", 0);
        status.put("CONNECTED", 1);
        status.put("DISCONNECTED", -1);
        status.put("SHUTDOWN", -1);
        List<Date> timeStamps = new ArrayList<Date>();
        List<String> Logged = new ArrayList<String>();

        for (int i = 0; i < lines.length; i++) {
            String[] line = lines[i].split(" :: ");

            if (!status.containsKey(line[1])) {
                continue;
            }

            timeStamps.add(getDate(line[0].substring(1, line[0].length() - 1)));
            Logged.add(line[1]);
        }

        long totalTime = timeStamps.get(timeStamps.size() - 1).getTime() -
timeStamps.get(0).getTime();
        long connectedTime = 0;
        long lastTimeStamp = 0;

        for (int i = 1; i < timeStamps.size(); i++) {
            String currentEvent = Logged.get(i);
            long currentTime = timeStamps.get(i).getTime();

            if (status.get(currentEvent) > 0) {
                lastTimeStamp = currentTime;
            } else if (lastTimeStamp > 0) {
                connectedTime += currentTime - lastTimeStamp;
                lastTimeStamp = -1;
            }
        }

        double ratio = (double) connectedTime / totalTime * 100;
        return String.format("%d%s", (int) ratio, "%");
    }

    private static Date getDate(String dateStr) {
```

```
        DateFormat formatter = new SimpleDateFormat("MM/dd/yyyy-hh:mm:ss");
        Date date = new Date();

        try {
        date = formatter.parse(dateStr);
        } catch (ParseException exception) {
        exception.printStackTrace();
        }

        return date;
    }



    /****************************************** main
*********************************************/
    public static void main(String[] args) throws FileNotFoundException,
IOException {
        String filename = "test2.txt";
        /*if (args.length > 0) {
        filename = args[0];
        }*/

        String answer = parseFile(filename);
        System.out.println(answer);
    }

    static String parseFile(String filename)
            throws FileNotFoundException, IOException {
        /*
         *  Don't modify this function
         */
        BufferedReader input = new BufferedReader(new FileReader(filename));
        List<String> allLines = new ArrayList<String>();
        String line;
        while ((line = input.readLine()) != null) {
            allLines.add(line);
        }
        input.close();

        return parseLines(allLines.toArray(new String[allLines.size()]));
    }
}
```

```java
public class FirstOccurrenceOfBinarySearch {
    private int firstOccurrenceBinarySearch(int[] source, int needle) {
        int low = 0;
        int high = source.length - 1;
        int firstOccurrence = Integer.MIN_VALUE;
        while (low <= high) {
            int middle = low + ((high - low) >>> 1);
            if (source[middle] == needle) {
                // key found and we want to search an earlier occurrence
                firstOccurrence = middle;
                high = middle - 1;
            } else if (source[middle] < needle) {
                low = middle + 1;
            } else {
                high = middle - 1;
            }
        }
        if (firstOccurrence != Integer.MIN_VALUE) {
            return firstOccurrence;
        }
        return -(low + 1); // key not found
    }
}


public class InorderSuccessorInBST {
    public static TreeNode inorderSuccessor2(TreeNode root, TreeNode node) {
        if (node.right != null) { // 有右孩子子，直接找右子子树的最小小节点
        return minValue(node.right);
        }
        TreeNode succ = null;
        while(root != null) {
        if(root.val > node.val) { // 继续找更小小的
        succ = root; // 后继节点必然比比node要大大，所以只能在这里里保存
        root = root.left;
        }
        else if(root.val < node.val){ // 继续找更大大的
        root = root.right;
        }
        else{ // root节点和node节点重复，停止止
        break;
        }
        }
        return succ;
        }
    public static TreeNode minValue(TreeNode node) {
        TreeNode cur = node;
        // 最小小节点必定在最左下角角
        while (cur.left != null) {
        cur = cur.left;
        }
        return cur;
        }
    public TreeNode successor(TreeNode root, TreeNode p) {
          if (root == null)
            return null;

          if (root.val <= p.val) {
```

```
            return successor(root.right, p);
          } else {
            TreeNode left = successor(root.left, p);
            return (left != null) ? left : root;
          }
        }
    public TreeNode predecessor(TreeNode root, TreeNode p) {
          if (root == null)
            return null;

          if (root.val >= p.val) {
            return predecessor(root.left, p);
          } else {
            TreeNode right = predecessor(root.right, p);
            return (right != null) ? right : root;
          }
        }
}



public class LowestCommonAncestorOfBinarySearchTree {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q)
{
        if (root == null)
            return root;
        if ((p.val < root.val && q.val > root.val) || (p.val > root.val && q.val
< root.val)) {
            return root;
        } else if (p.val < root.val && q.val < root.val) {
            return lowestCommonAncestor(root.left, p, q);
        } else {
            return lowestCommonAncestor(root.right, p, q);
        }
    }
}



public class LowestCommonAncestorOfBinaryTree {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q)
{
        if (root == null) {
            return null;
        }
        if (root == p || root == q) {
            return root;
        }
        TreeNode l = lowestCommonAncestor(root.left, p, q);
        TreeNode r = lowestCommonAncestor(root.right, p, q);
        if (l != null && r != null) {
            return root;
        }
        return l != null ? l : r;
    }
}
```

```java
public class MaximumProductSubarray {
     public int maxProduct(int[] A) {
           if(A == null || A.length == 0)
           return 0;
           int max = A[0];
           int currmax = A[0];
           int currmin = A[0];
           for(int i = 1; i < A.length; i++){
               int temp = currmax;
               currmax = Math.max(A[i], Math.max(currmax * A[i], currmin * A[i]));
               currmin = Math.min(A[i], Math.min(temp * A[i], currmin * A[i]));
               max = Math.max(currmax, max);
           }
           return max;
       }
}



public class SortColor {
     public void sortColors(int[] A) {
           int redpt=0;
           int bluept=A.length-1;
           int RED=0;
           int BLUE=2;
           int i=0;
          while(i<=bluept){
               if(A[i]==RED){
                   swapColor(A, redpt, i);
                   redpt++;
                   i++;
               }
               else if(A[i] == BLUE){
                   swapColor(A, bluept,i);
                   bluept--;
               }
               else
                   i++;
           }
       }
       private void swapColor(int[] A, int from, int to){
           int temp = A[from];
           A[from] = A[to];
           A[to] = temp;
       }
}

public int strStr(String haystack, String needle) {
       int l1 = haystack.length(), l2 = needle.length();
       if (l1 < l2) return -1;
       if (l2 == 0 || needle == null) return 0;
       int threshold = l1 - l2;
       for (int i = 0; i <= threshold; ++i) {
       if (haystack.substring(i,i+l2).equals(needle)) {
       return i;
       }
       }
       return -1;  }
```

```java
public static TreeNode solve(String s) {
        if (s == null || s.length() == 0)
            return null;
        if (s.length() == 1)
            return new TreeNode(s.charAt(0));
        int flag = 0;
        int mid = 0;
        for (int i = 2; i <= s.length() - 1; i++) {
            if (s.charAt(i) == '?')
                flag++;
            else if (s.charAt(i) == ':') {
                if (flag == 0) {
                    mid = i;
                    break;
                } else
                    flag--;
            }
        }
        TreeNode head = new TreeNode(s.charAt(0));
        TreeNode temp_left = solve(s.substring(2, mid));
        TreeNode temp_right = solve(s.substring(mid + 1, s.length()));
        head.left = temp_left;
        head.right = temp_right;
        return head;
    }


public static List<Integer> getSums(int[] nums, int k) {
        List<Integer> result = new ArrayList<>();//注意(arraylist == null ||
arraylist.size() == 0)
        //要return一个已经初始化的arrayList而不是null，否则会有一个test case过不去
        if (nums == null || nums.length < k) {
            return result;
        }
        int sum = 0;
        for (int i = 0; i < nums.length; i++) {
            sum += nums[i];
            if (i >= k) {       //first
                sum -= nums[i - k];
            }
            if (i >= k - 1) {   //second
                result.add(sum);
            }
        }
        return result;
    }

public class WordBreak {
    public boolean wordBreak(String s, Set<String> wordDict) {
          boolean[] t = new boolean[s.length()+1];
          t[0]=true;
          for(int i=0; i<=s.length(); i++){
              for(int j=i-1; j>=0; j--){
                  if(t[j]&&wordDict.contains(s.substring(j,i))){
                  t[i]=true;
                  break;
                  }
              }       }       return t[s.length()];    } }
```