Java

TopCoder - KawigiEdit - Configuration: General/Testing  
C:\Users\owner\Documents\Visual Studio 2008\Projects\TopCoderCompetition - for C#  
**C:\Users\owner\My Programs\JavaEclipse\TopCoder\src** - for Java  
Change language from C# to Java by choosing regular editor, then Display Preferences, then Editors, Java

### Comparable interface & compareTo() method; Comparator interface & compare() method

**class** Player **implements** Comparable<Player>

{

**...**

**public** Player(String nam, **int** percent, **int** height)

{

**...**

}

//Comparable interface: sort using a type's natural order.

//The type Player must implement the inherited abstract method

//Comparable<Player>.compareTo(Player)

**public** **int** compareTo(Player that)

{

if (this.percent == that.percent) return that.height - this.height;

return that.percent - this.percent;

**if** (**this**.percent > that.percent) **return** -1;

**if** (**this**.percent < that.percent) **return** 1;

**if** (**this**.percent == that.percent)

**if** (**this**.height > that.height) **return** -1; **else** **return** 1;

**return** 0;

}

}

Comparable interface: sort using a type's natural order. Definition is within the datatype.

Comparator interface: sort using an alternate order or many different orders on the same data. Implement a method compare that compares 2 different Keys. Must be a total order. For example, sort strings by natural order, by case insensitive order, by song name, by song length. Decouples the definition of the data type from the definition of what it means to compare two objects of that type. Comparators are done outside of the datatype, even at some later time.

Create Comparator object. Pass as second argument to Arrays.sort().  
Arrays.sort(a, String.CASE\_INSENSITIVE\_ORDER);

//Comparator interface: sort using an alternate order, from the one

//established by default or by Comparable<>.compareTo()

Arrays.*sort*(player, **new** Comparator<Player>()

{

@Override

**public** **int** compare(Player a, Player b) {

//Sort according to highest percent then height (total)

**if** (a.percent > b.percent) **return** -1;

**if** (a.percent < b.percent) **return** 1;

**if** (a.percent == b.percent)

**if** (a.height > b.height) **return** -1; **else** **return** 1;

**return** 0;

}

});

### Comparator interface & compare() method

**class** ClsComparator **implements** Comparator<Integer>

{

//compare method must be public and have two objects of type Integer (matching

Comparator<>)

**public** **int** compare(Integer a, Integer b)

{

**int** dif = Integer.*bitCount*(a) - Integer.*bitCount*(b);

**if** (dif == 0) **return** a-b;

**return** dif;

}

}

**public** **class** SRM166\_Div2\_L2\_BinaryCardinality\_Cmp

{

**public** **int**[] arrange(**int**[] numbers)

{

**int** n = numbers.length;

Integer[] nums = **new** Integer[n];

**for** (**int** i = 0; i < n; i++) nums[i] = numbers[i];

Arrays.*sort*(nums, **new** ClsComparator()); //version 1 - separate class

Arrays.*sort*(nums, **new** Comparator<Integer>() //version 2 - embedded class

{

**public** **int** compare(Integer a, Integer b)

{

**int** dif = Integer.*bitCount*(a) - Integer.*bitCount*(b);

**if** (dif == 0) **return** a-b;

**return** dif;

}

});

**for** (**int** i = 0; i < n; i++) numbers[i] = nums[i];

**return** numbers;

}   
}

### Comparable interface & compareTo() method;

**class** Nums **implements** Comparable<Nums>

{

**int** num;

**public** Nums(Integer num) {**this**.num = num;}

**public** **int** compareTo(Nums b)

{

**int** dif = Integer.*bitCount*(**this**.num) - Integer.*bitCount*(b.num);

**if** (dif == 0) **return** **this**.num - b.num;

**return** dif;

}

}

**public** **class** Comparable\_SRM166\_Div2\_L2\_BinaryCardinality\_Cmp

{

**public** **int**[] arrange(**int**[] numbers)

{

**int** n = numbers.length;

Nums[] nums = **new** Nums[n];

**for** (**int** i = 0; i < n; i++) nums[i] = **new** Nums(numbers[i]);

Arrays.*sort*(nums); //version 1 - separate class

**for** (**int** i = 0; i < n; i++) numbers[i] = nums[i].num;

**return** numbers;

}

### Build Int Array from ArrayList

**public** **int**[] buildIntArrayFromArrayList(ArrayList<Integer> nums)

{

**int** n = nums.size();

**int**[] intArray = **new** **int**[n];

**for** (**int** i = 0; i < n; i++) intArray[i] = nums.get(i);

**return** intArray;

}

int x = Integer.bitCount(num); //gets number of bits in base 2 version of num

import java.io.BufferedReader;  
import java.io.InputStreamReader;

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));   
int number=Integer.parseInt(br.readLine());

**public** **static** **void** main(String[] args) **throws** IOException {

//Scanner in = new Scanner(System.in);

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(System.*in*));

PrintWriter out = **new** PrintWriter(System.*out*);

//Reading 100,000 numbers on a line takes 3 times longer with Scanner

//Scanner = 436ms; BufferedReader w/split = 186ms; w/o split = 124ms

//For 200,000 numbers on a line: Scanner = 810ms; BR w/split = 280ms.

//Thus, use BufferedReader instead of Scanner if time is critical.

*solve*(in, out);

out.close();

in.close();

}

TopCoder automatically adds 4 imports:

**import** java.util.\*;

**import** java.util.regex.\*;

**import** java.text.\*;

**import** java.math.\*;

import java.awt.geom.\*; //I've seen some add this one

Wata uses the following:  
import java.util.\*;  
import static java.lang.Math.\*;  
import static java.util.Arrays.\*;

import java.util.\*; //for ArrayList<>, Arrays.sort

For some reason, CodeChef does not like "public" preceding the class and needs to throw exception:

**~~public~~ class** CC2013\_10\_B\_PPNUM {

**public** **static** **void** main(String[] args) **throws** IOException {

Scanner sc = **new** Scanner(System.*in*);

System.*out*.println("Working Directory = " + System.*getProperty*("user.dir"));

Using StringBuilder as a buffer is 30% faster than simply writing every character, horizontally.

In another test CF R215\_Div2\_C.java and R187\_Div2\_B, there was no difference when displaying many println vs embedding \r\n in StringBuilder; thus println is fine.

//The following is fine in CF, with extra space at end and no line-feed

**for** (**int** i = 1; i <= n; i++)

System.*out*.print(c[i] + " ");

//Alternatively, StringBuilder is 30% faster;

//StringBuilder sb = new StringBuilder();

//for (int i = 1; i <= n; i++)

//{

// if (i > 1) sb.append(" ");

// sb.append(c[i]);

//}

//System.out.println(sb);

**System.out.println(sb.toString().trim()); //trim space at beginning/end**

To concatenate characters to make a string, transform char to string using "" after each character, otherwise it will add the ASCII values together:

String a = c[i - 3] + "" + c[i - 2] + "" + c[i - 1] + "a";

// Fast Reader - uses Buffered Reader and special Input Reader class;

// It is 30% faster than just using Buffered Reader, which itself is

// 100% faster than using Scanner.

Scramble array before sorting large amount of numbers to avoid O(n^2)

Random rnd = **new** Random();

**int** ra, temp;

**for** (**int** i = 0; i < n; i++) {

ra = rnd.nextInt(i + 1);

temp = a[i];

a[i] = a[ra];

a[ra] = temp;

}

Arrays.*sort*(a);

### Find Unique number of strings

**public** **int** find(String[] questions)

{

//HashSet<String> hs = new HashSet<String>(Arrays.asList(questions));

HashSet<String> hs = **new** HashSet<String>();

**for** (String s : questions)

hs.add(s);

**return** 1 << hs.size();

}

**Prime Factorization using Map / TreeMap**

//import java.util.HashMap;

import java.util.Map;

//import java.util.SortedMap;

import java.util.TreeMap;

public class PrimeFactorization\_Map {

public static void main(String[] args) {

int n = 3000; // 2\*3\*5\*7\*11\*13\*17\*23;

Map<Integer,Integer> pf = getPrimeFactorization(n);

System.out.println("Prime Factorization of " + n);

System.out.println("# of unique primes: " + pf.size());

//Iterate through the map set and display each

//prime factor x num of occurrences

for (Map.Entry<Integer, Integer> entry : pf.entrySet())

System.out.println(entry.getKey() + " x " + entry.getValue());

for (int i : pf.keySet()) System.out.println(i);

for (int i : pf.values()) System.out.println(i);

}

private static TreeMap<Integer, Integer> getPrimeFactorization(int n)

{

// 3000 = 2x3 \* 3x1 \* 5x3

TreeMap<Integer,Integer> pf = new TreeMap<Integer,Integer>();

for (int i = 2; i \* i <= n; i++)

{

int cnt = 0;

while (n % i == 0)

{

cnt++;

n /= i;

// if (pf.get(i) != null)

// pf.put(i, pf.get(i)+1);

// else

// pf.put(i, 1);

}

if (cnt > 0) pf.put(i, cnt);

}

if (n > 1) pf.put(n, 1);

return pf;

}

}

**Graph**

ArrayList<Edge> [] list;

class Edge {

int from, to, cost;

public Edge(int from, int to, int cost) {

super();

this.from = from;

this.to = to;

this.cost = cost;

}

}

list = new ArrayList[n];

for(int i = 0; i < n; i++){

list[i] = new ArrayList<>();

}

class Road {

int to;

boolean pr;

Road(int to, int type) {

this.to = to;

this.pr = type == 2;

}

}

ArrayList<Road>[] g;

g = new ArrayList[n];

ans = new boolean[n];

for (int i = 0; i < n; i++)

g[i] = new ArrayList<>();

class Edge {

int to;

boolean bad;

public Edge(int to, boolean bad) {

super();

this.to = to;

this.bad = bad;

}

}

List<Edge>[] graph;

boolean[] take;

boolean dfs(int u, int p) {

boolean hasChild = false;

for (Edge e : graph[u]) {

if (e.to == p) {

continue;

}

if (dfs(e.to, u)) {

hasChild = true;

} else {

if (e.bad) {

take[e.to] = true;

hasChild = true;

}

}

}

return hasChild;

}

public void solve() {

int n =in.nextInt();

graph = new List[n];

for (int i = 0; i < n; i++) {

graph[i] = new ArrayList<>();

}

for (int i = 0; i < n - 1; i++) {

int from = in.nextInt() - 1, to = in.nextInt() - 1;

boolean bad = in.nextInt() == 2;

graph[from].add(new Edge(to, bad));

graph[to].add(new Edge(from, bad));

}

take = new boolean[n];

dfs(0, -1);

int size = 0;

for (boolean b : take) {

size += b ? 1 : 0;

}

out.println(size);

for (int i = 0; i < n; i++) {

if (take[i]) {

out.print((i + 1) + " ");

}

}

}

**String vs. StringBuilder**

s = s + "a" is linear sb.append("a") is constant (amortized)

s.substring(i,j) is constant sb.substring(i,j) is linear

**StringBuilder performance vs println - No difference in Java with repeated prints**

//sb.append(max);

//sb.append("\r\n");

out.println(max);

}

//out.println(sb);

**Memory**

//R267\_Div2\_C.java

//Java 7 solution - gets a Memory limit exceeded on test 22

// Input: 5000 1 5000, thus n=5000, k=5000

// Using 262,100 KB memory

//Java 8 solution - Accepted, Using max of 204,700 KB memory

long[][] dp = new long[n+1][k+1];

**To enable Java assertions (assert) in Eclipse:**

