Sample 3 MUM entrance exam solutions

Posted by: **rupaknpl** Posted date: **February 16, 2013** In: **Other** | comment : **0**

There are three questions on this exam. You have 2 hours to complete it.

1. An array is **zero-plentiful** if it contains at least one 0 and every sequence of 0s is of length at least 4.

Write a method named **isZeroPlentiful** which returns the number of zero sequences if its array argument is zero-plentiful, otherwise it returns 0.

If you are programming in Java or C#, the function signature is

int isZeroPlentiful(int[ ] a)

If you are programming in C or C++, the function signature is

int isZeroPlentiful(int a[ ], int len) where len is the number of elements in the array a.

Examples

a is	then function returns	reason
{0, 0, 0, 0, 0}1	1	because there is one sequence of 0s and its length >= 4.
{1, 2, 0, 0, 0, 0, 2, -18, 0, 0, 0, 0, 0, 12}1	2	because there are two sequences of 0s and both have lengths >= 4.
{0, 0, 0, 0, 1, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0}1	3	because three are three sequences of zeros and all have length >=4
{1, 2, 3, 4}1	0	because there must be at least one 0.
{1, 0, 0, 0, 2, 0, 0, 0, 0}	0	because there is a sequence of zeros whose length is less < 4.
{0}	0	because there is a sequence of zeroes whose length is < 4.
{}	0	because there must be at lea

2. A number is called **digit-increasing** if it is equal to n + nn + nnn + … for some digit n between 1 example 24 is digit-increasing because it equals 2 + 22 (here n = 2)

Write a function called **isDigitIncreasing** that returns 1 if its argument is digit-increasing otherwise, it returns 0.

The signature of the method is

READ PREVIOUS POST:
sample 2 question collections of MUM university MS computer science
There are three questions on this exam. You have two hours to complete it. 1. Write a function...

int isDigitIncreasing(int n)

Examples

if n is	then function returns	reason
7	1	because 7 = 7 (here n is 7)
36	1	because 36 = 3 + 33
984	1	because 984 = 8 + 88 + 888
7404	1	because 7404 = 6 + 66 + 666 + 6666

3. An integer number can be encoded as an array as follows. Each digit n of the number is represented by n zeros followed by a 1. So the digit 5 is represented by 0, 0, 0, 0, 0, 1. The encodings of each digit of a number are combined to form the encoding of the number. So the number 1234 is encoded as the array {0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1}. The first 0, 1 is contributed by the digit 1, the next 0, 0, 1 is contributed by the digit 2, and so on.

There is one other encoding rule: if the number is negative, the first element of the encoded array must be -1, so -201 is encoded as {-1, 0, 0, 1, 1, 0, 1}. Note that the 0 digit is represented by no zeros, i.e. there are two consecutive ones!

Write a method named **decodeArray** that takes an encoded array and decodes it to return the number.

You may assume that the input array is a legal encoded array, i.e., that -1 will only appear as the first element, all elements are either 0, 1 or -1 and that the last element is 1.

If you are programming in Java or C#, the function prototype is

int decodeArray(int[ ] a)

If you are programming in C or C++, the function prototype is

int decodeArray(int a[ ], int len);

Examples

a is	then function returns	reason
{1}	0	because the digit 0 is represented by no zeros followed by a one.
{0, 1}	1	because the digit 1 is represented by one zero followed by a one.
{-1, 0, 1}	-1	because the encoding of a negative number begins with a -1 followed by the encoding of the absolute value of the number.
{0, 1, 1, 1, 1, 1, 0, 1}	100001	because the encoding of the first 1 is 0, 1, the encoding of each of the four 0s is just a 1 and the encoding of the last 1 is 0, 1.
{0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1}	999	because each 9 digit is encoded as 0,0,0,0,0,0,0,0,0,1.

+1 at Google Plus

g+1 Recommend this on Google

This exam consists of three questions. You have two hours in which to complete it.

1. An **onion** array is an array that satisfies the following condition for all values of j and k:

if j>=0 and k>=0 and j+k=length of array and j!=k then a[j]+a[k] <= 10

Write a function named **isOnionArray** that returns 1 if its array argument is an onion array and returns 0 if it is not.

Your solution must not use a nested loop (i.e., a loop executed from inside another loop). Furthermore, once you determine that the array is not an onion array your function must return 0; no wasted loops cycles please!

If you are programming in Java or C#, the function signature is

int isOnionArray(int[ ] a)

If you are programming in C or C++, the function signature is

int isOnionArray(int a[ ], int len) where len is the number of elements in the array a.

Examples

a is	then function returns	reason
{1, 2, 19, 4, 5}	1	because 1+5 <= 10, 2+4 <=10
{1, 2, 3, 4, 15}	0	because 1+15 > 10
{1, 3, 9, 8}	0	because 3+9 > 10
{2}	1	because there is no j, k where a[j]+a[k] > 10 and j+k=length of array and j!=k
{}	1	because there is no j, k where a[j]+a[k] > 10 and j+k=length of array and j!=k
{-2, 5, 0, 5, 12}	1	because -2+12 <= 10 and 5+5 <= 10

2. A number n is called **prime happy** if there is at least one prime less than n and the sum of all primes less than n is evenly divisible by n.

Recall that a prime number is an integer > 1 which has only two integer factors, 1 and itself

The function prototype is int **isPrimeHappy(int n)**;

Examples:

if n is	return	because
5	1	because 2 and 3 are the primes less than 5, their sum is 5 and 5 evenly divides 5.
25	1	because 2, 3, 5, 7, 11, 13, 17, 19, 23 are the primes less than 25, their sum is 100 and 25 evenly divides 100
32	1	because 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31 are the primes less than 32, their sum is 160 and 32 evenly divides 160
8	0	because 2, 3, 5, 7 are the primes less than 8, their sum is 17 and 8 does not evenly divide 17.
2	0	because there are no primes less than 2.

3. An integer number can be encoded as an array as follows. Each digit n of the number is represented by n

zeros followed by a 1. So the digit 5 is represented by 0, 0, 0, 0, 0, 1. The encodings of each digit of a number are combined to form the encoding of the number. So the number 1234 is encoded as the array {0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1}. The first 0, 1 is contributed by the digit 1, the next 0, 0, 1 is contributed by the digit 2, and so on. There is one other encoding rule: if the number is negative, the first element of the encoded array must be -1, so -201 is encoded as {-1, 0, 0, 1, 1, 0, 1}. Note that the 0 digit is represented by no zeros, i.e. there are two consecutive ones!

Write a method named **encodeArray** that takes an integer as an argument and returns the encoded array.

If you are programming in Java or C#, the function prototype is

int[] encodeArray(int n)

If you are programming in C or C++, the function prototype is

int * encodeArray(int n);

Hints

Use modulo 10 arithmetic to get digits of number

Make one pass through the digits of the number to compute the size of the encoded array.

Make a second pass through the digits of the number to set elements of the encoded array to 1.

Examples

n is	then function returns	reason
0	{1}	because the digit 0 is represented by no zeros and the representation of each digit ends in one.
1	{0, 1}	because the digit 1 is represented by one zero and the representation of each digit ends in one.
-1	{-1, 0, 1}	because the encoding of a negative number begins with a -1 followed by the encoding of the absolute value of the number.
100001	{0, 1, 1, 1, 1, 1, 0, 1}	because the encoding of the first 1 is 0, 1, the encoding of each of the four 0s is just a 1 and the encoding of the last 1 is 0, 1.
999	0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1	because each 9 digit is encoded as 0,0,0,0,0,0,0,0,0,1.

1. An array is called **systematically increasing** if it consists of increasing sequences of the numbers from 1 to n.

The first six (there are over 65,000 of them) systematically increasing arrays are:

{1}

{1, 1, 2}

{1, 1, 2, 1, 2, 3}

{1, 1, 2, 1, 2, 3, 1, 2, 3, 4}

{1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5}

{1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6}

Write a function named **isSystematicallyIncreasing** which returns 1 if its array argument is systematically increasing. Otherwise it returns 0.

If you are programming in Java or C#, the function signature is

int isSystematicallyIncreasing(int[ ] a)

If you are programming in C or C++, the function signature is

int isSystematicallyIncreasing(int a[ ], int len) where len is the number of elements in the array a.

Examples

a is	then function returns	reason
{1}	1	because 1 is a sequence from 1 to 1 and is the only sequence.
{1, 2, 1, 2, 3}	0	because it is missing the sequence from 1 to 1.
{1, 1, 3}	0	because {1, 3} is not a sequence from 1 to n for any n.
{1, 2, 1, 2, 1, 2}	0	because it contains more than one sequence from 1 to 2.
{1, 2, 3, 1, 2, 1}	0	because it is “backwards”, i.e., the sequences from 1 to n are not ordered by increasing value of n
{1, 1, 2, 3}	0	because the sequence {1, 2} is missing (it should precede {1, 2, 3})

2. A positive, non-zero number n is a **factorial prime** if it is equal to factorial(n) + 1 for some n and it is prime. Recall that factorial(n) is equal to 1 * 2 * … * n-1 * n. If you understand recursion, the recursive definition is

factorial(1) = 1;

factorial(n) = n*factorial(n-1).

For example, factorial(5) = 1*2*3*4*5 = 120.

Recall that a prime number is a natural number which has exactly two distinct natural number divisors: 1 and itself.

Write a method named **isFactorialPrime** which returns 1 if its argument is a factorial prime number, otherwise it returns 0.

The signature of the method is

int isFactorialPrime(int n)

Examples

if n is	then	reason

	function returns	
2	1	because 2 is prime and is equal to factorial(1) + 1
3	1	because 3 is prime and is equal to factorial(2) + 1
7	1	because 7 prime and is equal to factorial(3) + 1
8	0	because 8 is not prime
11	0	because 11 does not equal factorial(n) + 1 for any n (factorial(3)=6, factorial(4)=24)
721	0	because 721 is not prime (its factors are 7 and 103)

3. Write a function named **largestDifferenceOfEvens** which returns the largest difference between even valued elements of its array argument. For example largestDifferenceOfEvens(new int[ ]{-2, 3, 4, 9}) returns 6 = (4 – (-2)). If there are fewer than 2 even numbers in the array, largestDifferenceOfEvens should return -1.

If you are programming in Java or C#, the function signature is

int largestDifferenceOfEvens(int[ ] a)

If you are programming in C or C++, the function signature is

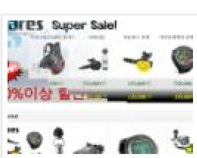
int largestDifferenceOfEvens(int a[ ], int len) where len is the number of elements in the array a.

Examples

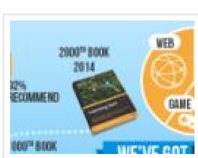
a is	then function returns	reason
{1, 3, 5, 9}	-1	because there are no even numbers
{1, 18, 5, 7, 33}	-1	because there is only one even number (18)
{[2, 2, 2, 2]}	0	because 2-2 == 0
{1, 2, 1, 2, 1, 4, 1, 6, 4}	4	because 6 – 2 == 4

‹ Previous Next ›

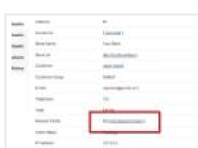
Related posts



Scubattokki.com checkout system and many vqmod conflicts removed successfully



2000th title of Packt Publishing are launching an exciting campaign until 26th Mar 2014



How Reward Points works in opencart 1.5+ eCommerce site

MUM test entrance exam solutions

Leave a Comment

Name*

Email*

Website

Submit Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.