# Question png images | PNGWing

QUESTION BANK DOCKET

**Table of Contents**

[1 Source 2](#_Toc253285289)

[2 Category 3](#_Toc965934774)

[3 Tech Area 3](#_Toc613746458)

[4 Complexity 3](#_Toc904143571)

[5 Question 3](#_Toc1426514880)

[6 Answer 3](#_Toc441207452)

[7 References 3](#_Toc1838997797)

# Source

Leet Code

# Category

|  |
| --- |
| DOT NET |

# Tech Area

C#

# Complexity

Medium

# Question

1.Given an array of n integers nums, a **132 pattern** is a subsequence of three integers nums[i], nums[j] and nums[k] such that i < j < k and nums[i] < nums[k] < nums[j].

Return true *if there is a* ***132 pattern*** *in* nums*, otherwise, return* false*.*

**Example 1:**

**Input:** nums = [1, 2, 3, 4]

**Output:** false

**Explanation:** There is no 132 pattern in the sequence.

**Example 2:**

**Input:** nums = [3, 1, 4, 2]

**Output:** true

**Explanation:** There is a 132 pattern in the sequence: [1, 4, 2].

**Example 3:**

**Input:** nums = [-1, 3, 2, 0]

**Output:** true

**Explanation:** There are three 132 patterns in the sequence: [-1, 3, 2], [-1, 3, 0] and [-1, 2, 0].

**Constraints:**

1. n == nums.length
2. 1 <= n <= 2 \* 105
3. -109 <= nums[i] <= 109

**-------------------------------------------------------------------------------------------------------------------**

2. You are implementing a program to use as your calendar. We can add a new event if adding the event will not cause a **double booking**.

A **double booking** happens when two events have some non-empty intersection (i.e., some moment is common to both events.).

The event can be represented as a pair of integers start and end that represents a booking on the half-open interval [start, end), the range of real numbers x such that start <= x < end.

Implement the MyCalendar class:

* MyCalendar () Initializes the calendar object.
* boolean book(int start, int end) Returns true if the event can be added to the calendar successfully without causing a **double booking**. Otherwise, return false and do not add the event to the calendar.

**Example 1:**

**Input**

["MyCalendar", "book", "book", "book"]

[[], [10, 20], [15, 25], [20, 30]]

**Output**

[null, true, false, true]

**Explanation**

MyCalendar myCalendar = new MyCalendar();

myCalendar.book(10, 20); // return True

myCalendar.book(15, 25); // return False, It can not be booked because time 15 is already booked by another event.

myCalendar.book(20, 30); // return True, The event can be booked, as the first event takes every time less than 20, but not including 20.

**Constraints:**

* 0 <= start < end <= 109
* At most 1000 calls will be made to book.

----------------------------------------------------------------------------------------------------------------

3. You are given an array of non-overlapping axis-aligned rectangles rects where rects[i] = [ai, bi, xi, yi] indicates that (ai, bi) is the bottom-left corner point of the ith rectangle and (xi, yi) is the top-right corner point of the ith rectangle. Design an algorithm to pick a random integer point inside the space covered by one of the given rectangles. A point on the perimeter of a rectangle is included in the space covered by the rectangle.

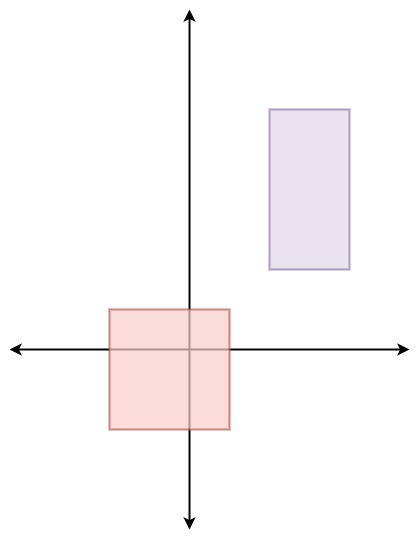
Any integer point inside the space covered by one of the given rectangles should be equally likely to be returned.

**Note** that an integer point is a point that has integer coordinates.

Implement the Solution class:

* Solution(int[][] rects) Initializes the object with the given rectangles rects.
* int[] pick() Returns a random integer point [u, v] inside the space covered by one of the given rectangles.

**Example 1:**



**Input**

["Solution", "pick", "pick", "pick", "pick", "pick"]

[[[[-2, -2, 1, 1], [2, 2, 4, 6]]], [], [], [], [], []]

**Output**

[null, [1, -2], [1, -1], [-1, -2], [-2, -2], [0, 0]]

**Explanation**

Solution solution = new Solution([[-2, -2, 1, 1], [2, 2, 4, 6]]);

solution.pick(); // return [1, -2]

solution.pick(); // return [1, -1]

solution.pick(); // return [-1, -2]

solution.pick(); // return [-2, -2]

solution.pick(); // return [0, 0]

**Constraints:**

* 1 <= rects.length <= 100
* rects[i].length == 4
* -109 <= ai < xi <= 109
* -109 <= bi < yi <= 109
* xi - ai <= 2000
* yi - bi <= 2000
* All the rectangles do not overlap.
* At most 104 calls will be made to pick.

----------------------------------------------------------------------------------------------------------------

4. Given the array orders, which represents the orders that customers have done in a restaurant. More specifically orders[i]=[customerNamei,tableNumberi,foodItemi] where customerNamei is the name of the customer, tableNumberi is the table customer sit at, and foodItemi is the item customer orders.

*Return the restaurant's “****display table****”*. The “**display table**” is a table whose row entries denote how many of each food item each table ordered. The first column is the table number and the remaining columns correspond to each food item in alphabetical order. The first row should be a header whose first column is “Table”, followed by the names of the food items. Note that the customer names are not part of the table. Additionally, the rows should be sorted in numerically increasing order.

**Example 1:**

**Input:** orders = [["David","3","Ceviche"],["Corina","10","Beef Burrito"],["David","3","Fried Chicken"],["Carla","5","Water"],["Carla","5","Ceviche"],["Rous","3","Ceviche"]]

**Output:** [["Table"," Burrito","Ceviche","Fried Chicken","Water"],["3","0","2","1","0"],["5","0","1","0","1"],["10","1","0","0","0"]]

**Explanation:**

The displaying table looks like:

**Table,Burrito,Ceviche,Fried Chicken,Water**

3 ,0 ,2 ,1 ,0

5 ,0 ,1 ,0 ,1

10 ,1 ,0 ,0 ,0

For the table 3: David orders "Ceviche" and "Fried Chicken", and Rous orders "Ceviche".

For the table 5: Carla orders "Water" and "Ceviche".

For the table 10: Corina orders "Beef Burrito".

**Example 2:**

**Input:** orders = [["James","12","Fried Chicken"],["Ratesh","12","Fried Chicken"],["Amadeus","12","Fried Chicken"],["Adam","1","Canadian Waffles"],["Brianna","1","Canadian Waffles"]]

**Output:** [["Table","Canadian Waffles","Fried Chicken"],["1","2","0"],["12","0","3"]]

**Explanation:**

For the table 1: Adam and Brianna order "Canadian Waffles".

For the table 12: James, Ratesh and Amadeus order "Fried Chicken".

**Example 3:**

**Input:** orders = [["Laura","2","Bean Burrito"],["Jhon","2","Beef Burrito"],["Melissa","2","Soda"]]

**Output:** [["Table","Bean Burrito",” Burrito","Soda"],["2","1","1","1"]]

**Constraints:**

* 1 <= orders.length <= 5 \* 10^4
* orders[i].length == 3
* 1 <= customerNamei.length, foodItemi.length <= 20
* customerNamei and foodItemi consist of lowercase and uppercase English letters and the space character.
* tableNumberi is a valid integer between 1 and 500.

----------------------------------------------------------------------------------------------------------------

5. You are given a stream of **records** about a particular stock. Each record contains a **timestamp** and the corresponding **price** of the stock at that timestamp.

Unfortunately due to the volatile nature of the stock market, the records do not come in order. Even worse, some records may be incorrect. Another record with the same timestamp may appear later in the stream **correcting** the price of the previous wrong record.

Design an algorithm that:

* **Updates** the price of the stock at a particular timestamp, **correcting** the price from any previous records at the timestamp.
* Finds the **latest price** of the stock based on the current records. The **latest price** is the price at the latest timestamp recorded.
* Finds the **maximum price** the stock has been based on the current records.
* Finds the **minimum price** the stock has been based on the current records.

Implement the StockPrice class:

* StockPrice() Initializes the object with no price records.
* void update(int timestamp, int price) Updates the price of the stock at the given timestamp.
* int current() Returns the **latest price** of the stock.
* int maximum() Returns the **maximum price** of the stock.
* int minimum() Returns the **minimum price** of the stock.

**Example 1:**

**Input**

["StockPrice", "update", "update", "current", "maximum", "update", "maximum", "update", "minimum"]

[[], [1, 10], [2, 5], [], [], [1, 3], [], [4, 2], []]

**Output**

[null, null, null, 5, 10, null, 5, null, 2]

**Explanation**

StockPrice stockPrice = new StockPrice();

stockPrice.update(1, 10); // Timestamps are [1] with corresponding prices [10].

stockPrice.update(2, 5); // Timestamps are [1,2] with corresponding prices [10,5].

stockPrice.current(); // return 5, the latest timestamp is 2 with the price being 5.

stockPrice.maximum(); // return 10, the maximum price is 10 at timestamp 1.

stockPrice.update(1, 3); // The previous timestamp 1 had the wrong price, so it is updated to 3.

// Timestamps are [1,2] with corresponding prices [3,5].

stockPrice.maximum(); // return 5, the maximum price is 5 after the correction.

stockPrice.update(4, 2); // Timestamps are [1,2,4] with corresponding prices [3,5,2].

stockPrice.minimum(); // return 2, the minimum price is 2 at timestamp 4.

**Constraints:**

* 1 <= timestamp, price <= 109
* At most 105 calls will be made **in total** to update, current, maximum, and minimum.
* current, maximum, and minimum will be called **only after** update has been called **at least once**.

--------------------------------------------------------------------------------------------------------------------------------------------

6. Given an integer array nums, return *the number of* ***reverse pairs*** *in the array*.

A **reverse pair** is a pair (i, j) where:

* 0 <= i < j < nums.length and
* nums[i] > 2 \* nums[j].

**Example 1:**

**Input:** nums = [1,3,2,3,1]

**Output:** 2

**Explanation:** The reverse pairs are:

(1, 4) --> nums[1] = 3, nums[4] = 1, 3 > 2 \* 1

(3, 4) --> nums[3] = 3, nums[4] = 1, 3 > 2 \* 1

**Example 2:**

**Input:** nums = [2,4,3,5,1]

**Output:** 3

**Explanation:** The reverse pairs are:

(1, 4) --> nums[1] = 4, nums[4] = 1, 4 > 2 \* 1

(2, 4) --> nums[2] = 3, nums[4] = 1, 3 > 2 \* 1

(3, 4) --> nums[3] = 5, nums[4] = 1, 5 > 2 \* 1

**Constraints:**

* 1 <= nums.length <= 5 \* 104
* -231 <= nums[i] <= 231 – 1

---------------------------------------------------------------------------

7. You are given two 2D integer arrays, items1 and items2, representing two sets of items. Each array items has the following properties:

* items[i] = [valuei, weighti] where valuei represents the **value** and weighti represents the **weight** of the ith item.
* The value of each item in items is **unique**.

Return *a 2D integer array* ret *where* ret[i] = [valuei, weighti]*,* *with* weighti *being the* ***sum of weights*** *of all items with value* valuei.

**Note:** ret should be returned in **ascending** order by value.

**Example 1:**

**Input:** items1 = [[1,1],[4,5],[3,8]], items2 = [[3,1],[1,5]]

**Output:** [[1,6],[3,9],[4,5]]

**Explanation:**

The item with value = 1 occurs in items1 with weight = 1 and in items2 with weight = 5, total weight = 1 + 5 = 6.

The item with value = 3 occurs in items1 with weight = 8 and in items2 with weight = 1, total weight = 8 + 1 = 9.

The item with value = 4 occurs in items1 with weight = 5, total weight = 5.

Therefore, we return [[1,6],[3,9],[4,5]].

**Example 2:**

**Input:** items1 = [[1,1],[3,2],[2,3]], items2 = [[2,1],[3,2],[1,3]]

**Output:** [[1,4],[2,4],[3,4]]

**Explanation:**

The item with value = 1 occurs in items1 with weight = 1 and in items2 with weight = 3, total weight = 1 + 3 = 4.

The item with value = 2 occurs in items1 with weight = 3 and in items2 with weight = 1, total weight = 3 + 1 = 4.

The item with value = 3 occurs in items1 with weight = 2 and in items2 with weight = 2, total weight = 2 + 2 = 4.

Therefore, we return [[1,4],[2,4],[3,4]].

**Example 3:**

**Input:** items1 = [[1,3],[2,2]], items2 = [[7,1],[2,2],[1,4]]

**Output:** [[1,7],[2,4],[7,1]]

**Explanation:**

The item with value = 1 occurs in items1 with weight = 3 and in items2 with weight = 4, total weight = 3 + 4 = 7.

The item with value = 2 occurs in items1 with weight = 2 and in items2 with weight = 2, total weight = 2 + 2 = 4.

The item with value = 7 occurs in items2 with weight = 1, total weight = 1.

Therefore, we return [[1,7],[2,4],[7,1]].

**Constraints:**

* 1 <= items1.length, items2.length <= 1000
* items1[i].length == items2[i].length == 2
* 1 <= valuei, weighti <= 1000
* Each valuei in items1 is **unique**.
* Each valuei in items2 is **unique**.

-------------------------------------------------------------------------------------------------------

8. Given an integer array nums, return *an integer array* counts *where* counts[i] *is the number of smaller elements to the right of* nums[i].

**Example 1:**

**Input:** nums = [5,2,6,1]

**Output:** [2,1,1,0]

**Explanation:**

To the right of 5 there are **2** smaller elements (2 and 1).

To the right of 2 there is only **1** smaller element (1).

To the right of 6 there is **1** smaller element (1).

To the right of 1 there is **0** smaller element.

**Example 2:**

**Input:** nums = [-1]

**Output:** [0]

**Example 3:**

**Input:** nums = [-1,-1]

**Output:** [0,0]

**Constraints:**

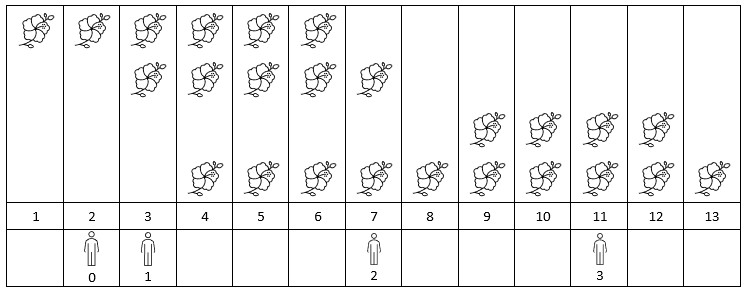
* 1 <= nums.length <= 105
* -104 <= nums[i] <= 104

---------------------------------------------------------------------------------------------------------------

9. You are given a **0-indexed** 2D integer array flowers, where flowers[i] = [starti, endi] means the ith flower will be in **full bloom** from starti to endi (**inclusive**). You are also given a **0-indexed** integer array persons of size n, where persons[i] is the time that the ith person will arrive to see the flowers.

Return *an integer array* answer *of size* n*, where* answer[i] *is the* ***number*** *of flowers that are in full bloom when the* ith *person arrives.*

**Example 1:**



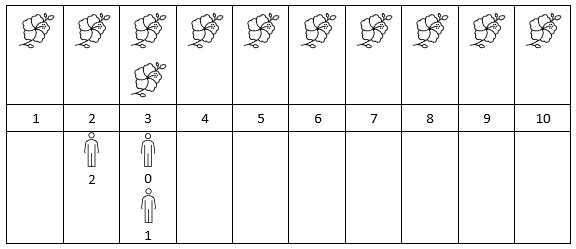
**Input:** flowers = [[1,6],[3,7],[9,12],[4,13]], persons = [2,3,7,11]

**Output:** [1,2,2,2]

**Explanation:** The figure above shows the times when the flowers are in full bloom and when the people arrive.

For each person, we return the number of flowers in full bloom during their arrival.

**Example 2:**



**Input:** flowers = [[1,10],[3,3]], persons = [3,3,2]

**Output:** [2,2,1]

**Explanation:** The figure above shows the times when the flowers are in full bloom and when the people arrive.

For each person, we return the number of flowers in full bloom during their arrival.

**Constraints:**

* 1 <= flowers.length <= 5 \* 104
* flowers[i].length == 2
* 1 <= starti <= endi <= 109
* 1 <= persons.length <= 5 \* 104
* 1 <= persons[i] <= 10

---------------------------------------------------------------------------------------------------------------

10. You are given an **integer** array pref of size n. Find and return *the array* arr *of size* n *that satisfies*:

* pref[i] = arr[0] ^ arr[1] ^ ... ^ arr[i].

Note that ^ denotes the **bitwise-xor** operation.

It can be proven that the answer is **unique**.

**Example 1:**

**Input:** pref = [5, 2,0,3,1]

**Output:** [5,7,2,3,2]

**Explanation:** From the array [5,7,2,3,2] we have the following:

- pref [0] = 5.

- Pref [1] = 5 ^ 7 = 2.

- Pref [2] = 5 ^ 7 ^ 2 = 0.

- Pref [3] = 5 ^ 7 ^ 2 ^ 3 = 3.

- Pref [4] = 5 ^ 7 ^ 2 ^ 3 ^ 2 = 1.

**Example 2:**

**Input:** pref = [13]

**Output:** [13]

**Explanation:** We have pref [0] = arr [0] = 13.

**Constraints:**

* 1 <= pref.length <= 105
* 0 <= pref[i] <= 106

**-------------------------------------------------------------------------------------------------------------------------------------------**

# Answer

1.

public class Solution {

public bool Find132pattern(int[] nums)

{

Stack<int> stack = new Stack<int>();

int k\_element = int.MinValue; // nums[k]

for(int i = nums.Length - 1; i >= 0; i--)

{

if (nums[i] < k\_element) return true;

while (stack.Count != 0 && stack.Peek() < nums[i])

{

k\_element = stack.Peek();

stack.Pop();

}

stack.Push(nums[i]);

}

return false;

}

2.

public class MyCalendar {

SortedList<int,int> events;

public MyCalendar() {

events = new();

}

public bool Book(int start, int end) {

int left =0, right = events.Count-1;

while(left <= right)

{

int mid = right + (left-right)/2;

int eStartTime = events.Keys[mid];

int eEndTime = events[eStartTime];

if(isTimeIntersected(start,end,eStartTime,eEndTime))

return false;

if(start > eStartTime)

left = mid +1;

else

right =mid-1;

}

events.Add(start,end);

return true;

}

private bool isTimeIntersected(int start, int end, int eventStart, int eventEnd)

=> (start >= eventStart && start < eventEnd) || (start< eventStart && end > eventStart);

}

3.

public class Solution {

private List<BigInteger> m\_Squares = new List<BigInteger>();

private List<(long x, long y, long w, long h)> m\_Rects = new();

public Solution(int[][] rects) {

BigInteger total = 0;

foreach (int[] rect in rects) {

long w = rect[2] - rect[0] + 1;

long h = rect[3] - rect[1] + 1;

m\_Rects.Add((rect[0], rect[1], w, h));

m\_Squares.Add(total += w \* h);

}

}

public (long x, long y) At(BigInteger value)

{

int index = m\_Squares.BinarySearch(value);

index = index < 0 ? ~index : index + 1;

var rect = m\_Rects[index];

long at = (long)(index == 0

? value

: value - m\_Squares[index - 1]);

long y = rect.y + at / rect.w;

long x = rect.x + at % rect.w;

return (x, y);

}

public BigInteger Count => m\_Squares[m\_Squares.Count - 1];

public int[] Pick() {

BigInteger value = (BigInteger)(((double)Count) \* Random.Shared.NextDouble());

var p = At(value);

return new int[] { (int)(p.x), (int)(p.y) };

}

}

4.

public class Solution {

public IList<IList<string>> DisplayTable(IList<IList<string>> orders)

{

IList<IList<string>> display = new List<IList<string>>();

display.Add(new List<string>());

display[0].Add("Table");

foreach(var food in orders.GroupBy(g => g[2]).OrderBy(o => o.Key, StringComparer.Ordinal).Select(s => s.Key))

{

display[0].Add(food);

}

foreach(var table in orders.GroupBy(g => g[1]).OrderBy(o => Int32.Parse(o.Key)))

{

IList<string> newTable = new List<string>();

newTable.Add(table.Key);

foreach(var food in display[0])

{

if(!String.Equals(food, "Table"))

{

newTable.Add(table.Where(w => w[2] == food).Count().ToString());

}

}

display.Add(newTable);

}

return display;

}

}

5.

public class StockPrice {

Dictionary<int, int> map;

int latestTS = 0;

SortedSet<(int price, int timeStamp)> mapPrice;

public StockPrice() {

map = new Dictionary<int,int>();

mapPrice = new SortedSet<(int,int)>();

}

public void Update(int timestamp, int price) {

latestTS = Math.Max(latestTS, timestamp);

if(map.ContainsKey(timestamp))

{

int incorrectPrice = map[timestamp];

mapPrice.Remove((incorrectPrice, timestamp));

mapPrice.Add((price,timestamp));

map[timestamp] = price;

}

else

{

map.Add(timestamp,price);

mapPrice.Add((price, timestamp));

}

}

public int Current() {

if(map.Count ==0) return -1;

return map[latestTS];

}

public int Maximum() {

if(mapPrice.Count == 0) return -1;

return mapPrice.Max.price;

}

public int Minimum() {

if(mapPrice.Count == 0) return -1;

return mapPrice.Min.price;

}

}

6.

public class Solution {

int count=0;

public int ReversePairs(int[] nums) {

MergeSort(nums,0,nums.Length-1);

return count;

}

void MergeSort(int[] nums,int left,int right)

{

if(left<right)

{

int mid=(left+right)/2;

MergeSort(nums,left,mid);

MergeSort(nums,mid+1,right);

Merge(nums,left,mid,right);

}

}

void Merge(int[] nums,int left,int mid,int right)

{

int i=left,j=mid+1;

while(i<=mid && j<=right)

{

if((long)nums[i]>(long)2\*nums[j])

{

count=count+(mid-i+1);

j++;

}

else

i++;

}

int[] temp=new int[right-left+1];

int k=0;

for(i=left;i<=right;i++)

temp[k++]=nums[i];

Array.Sort(temp);

k=0;

for(i=left;i<=right;i++)

nums[i]=temp[k++];

}

}

7.

public class Solution {

public IList<IList<int>> MergeSimilarItems(int[][] items1, int[][] items2) {

IList<IList<int>> result = new List<IList<int>>();

foreach(var item in items1.ToList()){

result.Add(item);

}

foreach(var item in items2.ToList()){

var common = result.FirstOrDefault(r=>r[0] == item[0]);

if(common != null){

common[1] += item[1];

}

else{

result.Add(item);

}

}

return result.OrderBy(p=>p[0]).ToList();

}

}

8.

public class Solution

{

private readonly int Size = 10000;

void update(int[] T, int pos)

{

pos += Size;

int n = T.Length / 2;

pos += n;

T[pos]++;

while (pos > 1)

{

pos /= 2;

T[pos] = T[2 \* pos] + T[2 \* pos + 1];

}

}

int query(int[] T, int left, int right)

{

left += Size;

right += Size;

int n = T.Length / 2;

left += n;

right += n;

int result = 0;

while (left < right)

{

if (left % 2 == 1) result += T[left++];

if (right % 2 == 1) result += T[--right];

left /= 2;

right /= 2;

}

return result;

}

public IList<int> CountSmaller(int[] nums)

{

int length = nums.Length, min = int.MaxValue;

int[] result = new int[length];

int[] T = new int[4 \* Size + 2];

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

{

int current = nums[i];

update(T, current);

min = Math.Min(min, current);

result[i] = query(T, min, current);

}

return result.ToList();

}

}

9.

public class Solution

{

public int[] FullBloomFlowers(int[][] flowers, int[] persons)

{

var changesQueue = new PriorityQueue<int, int>();

foreach (var bloomPeriod in flowers)

{

changesQueue.Enqueue(1, bloomPeriod[0]);

changesQueue.Enqueue(-1, bloomPeriod[1] + 1);

}

var changesHistory = new List<(int numberOfFlowersInFullBloom, int time)>();

changesHistory.Add((0, 0));

var inFullBloom = 0;

while (changesQueue.TryDequeue(out var change, out var time))

{

inFullBloom += change;

if (changesHistory.Count > 0 && changesHistory[^1].time == time)

{

changesHistory[^1] = (inFullBloom, time);

}

else

{

changesHistory.Add((inFullBloom, time));

}

}

return persons.Select(GetNumberOfFlowersInFullBloomByTime).ToArray();

int GetNumberOfFlowersInFullBloomByTime(int time)

{

var l = 0;

var r = changesHistory.Count - 1;

while (l < r)

{

var mid = (l + r + 1) / 2;

if (time >= changesHistory[mid].time)

{

l = mid;

}

else

{

r = mid - 1;

}

}

return changesHistory[l].numberOfFlowersInFullBloom;

}

}

}

10.

public class Solution {

public int[] FindArray(int[] pref) {

int[] result = pref.ToArray();

for (int i = 1, prior = result[0]; i < pref.Length; prior ^= result[i++])

result[i] = prior ^ pref[i];

return result;

}

}

# References

[Discussion Boards, Comments, Links, etc..]x

https://leetcode.com/tag/ordered-set/

Question 1: // Have the function QuestionsMarks(str) take the str string parameter, which

// will contain single digit numbers, letters, and question marks, and check

// if there are exactly 3 question marks between every pair of two numbers

// that add up to 10. If so, then your program should return the string true,

// otherwise it should return the string false.If there aren't any two numbers

// that add up to 10 in the string, then your program should return false as well.

// For example : if str is "arrb6???4xxbl5???eee5" then your program should return

// true because there are exactly 3 question marks between 6 and 4, and 3 question

// marks between 5 and 5 at the end of the string.

Answer:

|  |  |
| --- | --- |
| using System; | |
|  |  |
|  | namespace Coderbyte\_CSharp.Easy\_Challenges |
|  | { |
|  | class QuestionMarkSum |
|  | { |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | public string QuestionsMarks(string str) |
|  | { |
|  | string result = String.Empty; |
|  | bool pass = false; |
|  | int firstPos = -1; |
|  | int secondPos = -1; |
|  | int index = 0; |
|  |  |
|  | while (index < str.Length) |
|  | { |
|  | if (FindNumbersInString(str, index, out firstPos, out secondPos)) |
|  | { |
|  |  |
|  | int first = (int)Char.GetNumericValue(str[firstPos]); |
|  | int second = (int)Char.GetNumericValue(str[secondPos]); |
|  |  |
|  |  |
|  | // 2 integers must equal 10 |
|  | if ((first + second) == 10) |
|  | { |
|  | // There must be exactly 3 question marks between 2 integers |
|  | pass = IsQuestionMarksExist(str, firstPos, secondPos); |
|  | } |
|  |  |
|  | index = secondPos + 1; |
|  | } |
|  |  |
|  | // No numbers found in string |
|  | else |
|  | { |
|  | pass = false; |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | result = pass ? "true" : "false"; |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | protected bool FindNumbersInString(string str, int start, out int first, out int second) |
|  | { |
|  | bool result = false; |
|  | first = -1; |
|  | second = -1; |
|  |  |
|  | if (start == str.Length - 1) |
|  | { |
|  | return false; |
|  | } |
|  |  |
|  | for (int index = start; index < str.Length; index++) |
|  | { |
|  | if (Char.IsDigit(str[index])) |
|  | { |
|  | if (first == -1) |
|  | { |
|  | first = index; |
|  | } |
|  |  |
|  | else |
|  | { |
|  | second = index; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | result = (first != -1 && second != -1); |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | protected bool IsQuestionMarksExist(string str, int start, int end) |
|  | { |
|  | bool result = false; |
|  |  |
|  | int num1 = start + 1; |
|  | int num2 = end; |
|  | int count = 0; |
|  |  |
|  | if (num2 - num1 >= 3) |
|  | { |
|  | for (int index = num1; index < num2; index++) |
|  | { |
|  | if (str[index] == '?') |
|  | { |
|  | count++; |
|  | } |
|  | } |
|  |  |
|  | result = (count == 3); |
|  | } |
|  |  |
|  |  |
|  | return result; |
|  | } |
|  | } |
|  | } |

Question:2

//Have the function CodelandUsernameValidation(str) take the str parameter being

// passed and determine if the string is a valid username according to the

//following rules :

//1. The username is between 4 and 25 characters.

//2. It must start with a letter.

//3. It can only contain letters, numbers, and the underscore character.

//4. It cannot end with an underscore character.

//If the username is valid then your program should return the string true,

//otherwise return the string false.

Answer:

|  |
| --- |
| using System; |
|  |  |
|  | namespace Coderbyte\_CSharp.\_1\_Easy\_Challenges |
|  | { |
|  | class UsernameValidation |
|  | { |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  | public string CodelandUsernameValidation(string str) |
|  | { |
|  | string result = String.Empty; |
|  |  |
|  | bool rule1 = ValidateLength(str); |
|  | bool rule2 = ValidateStartWithLetter(str); |
|  | bool rule3 = ValidateOnlyValidCharacters(str); |
|  | bool rule4 = ValidateEndCharacter(str); |
|  |  |
|  | result = (rule1 && rule2 && rule3 && rule4) ? "true" : "false"; |
|  |  |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | protected bool ValidateLength(string str) |
|  | { |
|  | bool isValid = false; |
|  |  |
|  | int length = str.Length; |
|  |  |
|  | isValid = (length >= 4 && length <= 25); |
|  |  |
|  | return isValid; |
|  | } |
|  |  |
|  | protected bool ValidateStartWithLetter(string str) |
|  | { |
|  | bool isValid = false; |
|  |  |
|  | isValid = Char.IsLetter(str[0]); |
|  |  |
|  | return isValid; |
|  | } |
|  |  |
|  | protected bool ValidateOnlyValidCharacters(string str) |
|  | { |
|  | bool isValid = true; |
|  |  |
|  | foreach (char c in str) |
|  | { |
|  | if (!Char.IsLetterOrDigit(c) && c != '\_') |
|  | { |
|  | isValid = false; |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | return isValid; |
|  | } |
|  |  |
|  | protected bool ValidateEndCharacter(string str) |
|  | { |
|  | bool isValid = false; |
|  | int length = str.Length; |
|  |  |
|  | isValid = (str[length - 1] != '\_'); |
|  |  |
|  | return isValid; |
|  | } |
|  |  |
|  |  |
|  | } |
|  | } |

Question:3

// Have the function TimeConvert(num) take the num parameter being passed and

// return the number of hours and minutes the parameter converts to (ie. if

// num = 63 then the output should be 1:3). Separate the number of hours

// and minutes with a colon.

Answer:

|  |
| --- |
| using System; |
|  | using System.Text; |
|  |  |
|  | namespace Coderbyte\_CSharp.Easy\_Challenges |
|  | { |
|  | class TimeConverter |
|  | { |
|  |
|  |
|  |
|  |  |
|  | public String TimeConvert(int num) |
|  | { |
|  | string result = string.Empty; |
|  | StringBuilder sb = new StringBuilder(); |
|  |  |
|  | int hours = num / 60; |
|  | int minutes = num % 60; |
|  |  |
|  | sb.Append(hours); |
|  | sb.Append(":"); |
|  | sb.Append(minutes); |
|  |  |
|  | result = sb.ToString(); |
|  |  |
|  | return result; |
|  | } |
|  | } |
|  | } |

Question:4

// Have the function FindIntersection(strArr) read the array of strings stored

// in strArr which will contain 2 elements: the first element will represent a

// list of comma - separated numbers sorted in ascending order, the second

// element will represent a second list of comma - separated numbers(also sorted).

// Your goal is to return a comma - separated string containing the numbers that

// occur in elements of strArr in sorted order.If there is no intersection,

// return the string false.

// For example : if strArr contains["1, 3, 4, 7, 13", "1, 2, 4, 13, 15"] the

// output should return "1,4,13" because those numbers appear in both strings.

// The array given will not be empty, and each string inside the array will be

// of numbers sorted in ascending order and may contain negative numbers.

Answer:

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Text; |
|  |  |
|  | namespace Coderbyte\_CSharp.Easy\_Challenges |
|  | { |
|  | class StringIntersection |
|  | { |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  | public string FindIntersection(string[] strArr) |
|  | { |
|  | string result = String.Empty; |
|  | char delim = ','; |
|  |  |
|  | int[] first = TokenizeInt(strArr[0], delim); |
|  | int[] second = TokenizeInt(strArr[1], delim); |
|  |  |
|  | List<int> matches = new List<int>(); |
|  |  |
|  | // find matches between an arrays |
|  | foreach(int value1 in first) |
|  | { |
|  | foreach(int value2 in second) |
|  | { |
|  | if (value1 == value2) |
|  | { |
|  | matches.Add(value1); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // Convert list to output string |
|  | StringBuilder sb = new StringBuilder(); |
|  |  |
|  | int count = matches.Count; |
|  | foreach( int value in matches) |
|  | { |
|  | sb.Append(value); |
|  | if (value != matches[count-1]) |
|  | { |
|  | sb.Append(","); |
|  | } |
|  | } |
|  |  |
|  | result = sb.ToString(); |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | protected int[] TokenizeInt(string str, char delim) |
|  | { |
|  | string[] strValues = str.Split(delim); |
|  | int[] nums = new int[strValues.Length]; |
|  |  |
|  | for(int index = 0; index < strValues.Length; index++) |
|  | { |
|  | try |
|  | { |
|  | nums[index] = Int32.Parse(strValues[index]); |
|  | } |
|  | catch (FormatException e) |
|  | { |
|  | Console.WriteLine("Unable to parse {0}", strValues[index]); |
|  | Console.Error.Write(e.Message); |
|  | } |
|  | } |
|  |  |
|  | return nums; |
|  | } |
|  | } |
|  | } |

Question:5

// Have the function MinWindowSubstring(strArr) take the array of strings stored

// in strArr, which will contain only two strings, the first parameter being the

// string N and the second parameter being a string K of some characters, and your

// goal is to determine the smallest substring of N that contains all the characters

// in K.

//

// For example: if strArr is ["aaabaaddae", "aed"] then the smallest substring of N

// that contains the characters a, e, and d is "dae" located at the end of the string.

// So for this example your program should return the string dae.

// Another example: if strArr is ["aabdccdbcacd", "aad"] then the smallest substring

// of N that contains all of the characters in K is "aabd" which is located at the

// beginning of the string. Both parameters will be strings ranging in length from

// 1 to 50 characters and all of K's characters will exist somewhere in the string N.

// Both strings will only contains lowercase alphabetic characters.

// Examples

// Input: {"ahffaksfajeeubsne", "jefaa"}

// Output: aksfaje

// Input: {"aaffhkksemckelloe", "fhea"}

// Output: affhkkse

Answer:

|  |  |
| --- | --- |
| using System; | |
|  | using System.Collections.Generic; |
|  |  |
|  | namespace Coderbyte\_CSharp.Medium\_Challenges |
|  | { |
|  | class StringMinimumWindow |
|  | { |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  |  |
|  | public string MinWindowSubstring(string[] strArr, int arrLength) |
|  | { |
|  | string result = string.Empty; |
|  |  |
|  | string text = strArr[0]; |
|  | string pattern = strArr[1]; |
|  |  |
|  | Dictionary<char, int> countMap = CreateCountMap(pattern); |
|  | List<string> windows = ExtractWindows(text, pattern.Length); |
|  |  |
|  | foreach (string window in windows) |
|  | { |
|  | if (IsValidWindow(window, countMap)) |
|  | { |
|  | if (String.IsNullOrEmpty(result)) |
|  | { |
|  | result = window; |
|  | } |
|  | else if (window.Length < result.Length) |
|  | { |
|  | result = window; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | protected Dictionary<char, int> CreateCountMap(string str) |
|  | { |
|  | Dictionary<char, int> countMap = new Dictionary<char, int>(); |
|  |  |
|  | foreach (char c in str) |
|  | { |
|  | if (countMap.ContainsKey(c)) |
|  | { |
|  | countMap[c]++; |
|  | } |
|  |  |
|  | else |
|  | { |
|  | countMap.Add(c, 1); |
|  | } |
|  | } |
|  | return countMap; |
|  | } |
|  |  |
|  | protected List<string> ExtractWindows(string str, int length) |
|  | { |
|  | List<string> windows = new List<string>(); |
|  | int strLength = str.Length; |
|  |  |
|  |  |
|  | for (int start = 0; start < strLength; start++) |
|  | { |
|  | for (int end = start + length; end < strLength + 1; end++) |
|  | { |
|  | windows.Add(str.Substring(start, end - start)); |
|  | } |
|  | } |
|  |  |
|  | return windows; |
|  | } |
|  |  |
|  | protected bool IsValidWindow(string window, Dictionary<char, int> patternMap) |
|  | { |
|  | bool isValid = true; |
|  |  |
|  | Dictionary<char, int> windowMap = CreateCountMap(window); |
|  |  |
|  | foreach(var item in patternMap) |
|  | { |
|  | var key = item.Key; |
|  | var val = item.Value; |
|  |  |
|  | if (!windowMap.ContainsKey(key) || windowMap[key] < val) |
|  | { |
|  | isValid = false; |
|  | } |
|  | } |
|  |  |
|  | return isValid; |
|  | } |
|  | } |
|  | } |

Question:6

// For this challenge you will encode a given string following a specific rule.

// The function NumberEncoding(str) with str parameter and encode the

// message according to the following rule: encode every letter into its

// corresponding numbered position in the alphabet. Symbols and spaces will also

// be used in the input.

// For example: if str is "af5c a#!" then your program should return 1653 1#!.

Answer:

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Text; |
|  |  |
|  | namespace Coderbyte\_CSharp.Medium\_Challenges |
|  | { |
|  | class NumberEncoder |
|  | { |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | public string NumberEncoding(string str) |
|  | { |
|  | string result = string.Empty; |
|  | StringBuilder sb = new StringBuilder(); |
|  | Dictionary<char, int> alphabet = CreateAlphabetMap(); |
|  |  |
|  | foreach (char item in str) |
|  | { |
|  | // is a letter |
|  | if (Char.IsLetter(item)) |
|  | { |
|  | sb.Append(alphabet[item]); |
|  | } |
|  |  |
|  | // not a letter |
|  | else |
|  | { |
|  | sb.Append(item); |
|  | } |
|  | } |
|  |  |
|  | result = sb.ToString(); |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | protected Dictionary<char, int> CreateAlphabetMap() |
|  | { |
|  | Dictionary<char, int> alpha = new Dictionary<char, int>(); |
|  |  |
|  | int value = 1; |
|  |  |
|  | for (char key = 'a'; key <= 'z'; key++, value++) |
|  | { |
|  | alpha.Add(key, value); |
|  | } |
|  |  |
|  | return alpha; |
|  | } |
|  | } |
|  | } |

Question:7

// Have the function KaprekarsConstant(num) take the num parameter being passed

// which will be a 4-digit number with at least two distinct digits. Your program

// should perform the following routine on the number: Arrange the digits in

// descending order and in ascending order (adding zeroes to fit it to a 4-digit

// number), and subtract the smaller number from the bigger number. Then repeat

// the previous step. Performing this routine will always cause you to reach a

// fixed number: 6174. Then performing the routine on 6174 will always give you

// 6174 (7641 - 1467 = 6174). Your program should return the number of times

// this routine must be performed until 6174 is reached.

// For example: if num is 3524 your program should return 3 because of the

// following steps: (1) 5432 - 2345 = 3087, (2) 8730 - 0378 = 8352,

// (3) 8532 - 2358 = 6174.

Answer:

|  |  |
| --- | --- |
| using System.Collections.Generic; | |
|  | using System.Linq; |
|  |  |
|  | namespace Coderbyte\_CSharp.Hard\_Challenges |
|  | { |
|  | class Kaprekar |
|  | { |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | public int KaprekarsConstant(int num) |
|  | { |
|  | int count = 0; |
|  | int remainder = num; |
|  |  |
|  | while (remainder != 6174) |
|  | { |
|  | remainder = DescendInt(remainder) - AscendInt(remainder); |
|  | count++; |
|  | } |
|  |  |
|  | return count; |
|  | } |
|  |  |
|  | protected int AscendInt(int num) |
|  | { |
|  | int result = 0; |
|  |  |
|  | result = AdjustInt(num, false); |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | protected int DescendInt(int num) |
|  | { |
|  | int result = 0; |
|  |  |
|  | result = AdjustInt(num, true); |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | private int AdjustInt(int num, bool descend) |
|  | { |
|  | int result = 0; |
|  | int[] arr = new int[4]; |
|  | int temp = num; |
|  |  |
|  | arr[0] = temp / 1000; temp -= arr[0] \* 1000; |
|  | arr[1] = temp / 100; temp -= arr[1] \* 100; |
|  | arr[2] = temp / 10; temp -= arr[2] \* 10; |
|  | arr[3] = temp; |
|  |  |
|  | List<int> valueList = arr.ToList(); |
|  |  |
|  | if (descend) |
|  | { |
|  | valueList.Sort(SortDescend); |
|  | } |
|  | else |
|  | { |
|  | valueList.Sort(); |
|  | } |
|  |  |
|  | result = valueList[0] \* 1000 + valueList[1] \* 100 + valueList[2] \* 10 + valueList[3]; |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | private int SortDescend(int a, int b) |
|  | { |
|  | int result = 0; |
|  |  |
|  | if (a > b) |
|  | { |
|  | result = -1; |
|  | } |
|  |  |
|  | else if (a < b) |
|  | { |
|  | result = 1; |
|  | } |
|  | else |
|  | { |
|  | result = 0; |
|  | } |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  |  |
|  | } |
|  | } |

Question:8

/\*

Have the function WordCount(str) take the str string parameter being passed and return the number of

words the string contains (e.g. "Never eat shredded wheat or cake" would return 6). Words will be separated

by single spaces.

\*/

Answer:

|  |
| --- |
| using System; |
|  |  |
|  | public class Solution0013 |
|  | { |
|  | public static int WordCount(string text) |
|  | { |
|  | int result = 0; |
|  |  |
|  | string[] arrayOfText = text.Split(" "); |
|  |  |
|  | result = arrayOfText.Length; |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | static void Main() |
|  | { |
|  | Console.WriteLine(WordCount(Console.ReadLine())); |
|  | } |
|  | } |

Question:9

// Have the function MatrixDeterminant(strArr) read strArr which will be an

// array of integers represented as strings.Within the array there will also

// be "<>" elements which represent break points.The array will make up a

// matrix where the(number of break points + 1) represents the number of

// rows.Here is an example of how strArr may look : ["1", "2", "<>", "3", "4"].

//

// The contents of this array are row1 = [1 2] and row2 = [3 4]. Your program

// should take the given array of elements, create the proper matrix, and then

// calculate the determinant. For the example above, your program should return -2.

// If the matrix is not a square matrix, return -1. The maximum size of strArr

// will be a 6x6 matrix.The determinant will always be an integer.

|  |  |
| --- | --- |
| using System; | |
|  | using System.Collections.Generic; |
|  |  |
|  | namespace Coderbyte\_CSharp.Hard\_Challenges |
|  | { |
|  | class Determinant |
|  | { |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | public int MatrixDeterminant(string[] strArr, int size) |
|  | { |
|  | int determinant = 0; |
|  |  |
|  | List<List<int>> matrix = new List<List<int>>(); |
|  |  |
|  | ParseMatrix(strArr, size, ref matrix); |
|  |  |
|  |  |
|  | if (IsSquareMatrix(matrix)) |
|  | { |
|  | determinant = ComputeDeterminant(matrix); |
|  | } |
|  |  |
|  | else |
|  | { |
|  | Console.WriteLine("matrix is not square"); |
|  | } |
|  |  |
|  |  |
|  | return determinant; |
|  | } |
|  |  |
|  | protected bool IsSquareMatrix(List<List<int>> matrix) |
|  | { |
|  | bool isSquare = true; |
|  |  |
|  | // checking if input matrix is square |
|  | for (int row = 0; row < matrix.Count; row++) |
|  | { |
|  | if (matrix[row].Count != matrix.Count) |
|  | { |
|  | isSquare = false; |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | return isSquare; |
|  | } |
|  |  |
|  | protected void ParseMatrix(string[] strArr, int size, ref List<List<int>> matrix) |
|  | { |
|  | List<int> rowData = new List<int>(); |
|  |  |
|  | for (int x = 0; x < size; x++) |
|  | { |
|  | // condition to check for breakpoints |
|  | if (strArr[x] == "<>") |
|  | { |
|  | // Create a separate copy of rowData for matrix |
|  | matrix.Add(new List<int>(rowData)); |
|  | rowData.Clear(); |
|  | } |
|  | else |
|  | { |
|  | // converting to int |
|  | int value = 0; |
|  |  |
|  | // convert to snippet |
|  | try |
|  | { |
|  | value = Int32.Parse(strArr[x]); |
|  | } |
|  | catch (FormatException e) |
|  | { |
|  | Console.WriteLine("Unable to parse {0}", strArr[x]); |
|  | Console.Error.Write(e.Message); |
|  | } |
|  |  |
|  | // adding to our row |
|  | rowData.Add(value); |
|  | } |
|  | } |
|  | // adding last row |
|  | // Create a separate copy of rowData for matrix |
|  | matrix.Add(new List<int>(rowData)); |
|  | } |
|  |  |
|  | protected int ComputeDeterminant(List<List<int>> matrix) |
|  | { |
|  | int determinant = 0; |
|  |  |
|  | if (matrix.Count == 1) |
|  | { |
|  | determinant = matrix[0][0]; |
|  | } |
|  | // check for 2-by-2 matrix perform det calculation |
|  | else if (matrix.Count == 2) |
|  | { |
|  | determinant = (matrix[0][0] \* matrix[1][1]) - (matrix[0][1] \* matrix[1][0]); |
|  | } |
|  |  |
|  | // otherwise determine cofactor, submatrix and recursively compute determinant |
|  | else |
|  | { |
|  | int cofactor = 0; |
|  | // iterate across first row to set cofactor |
|  | for (int p = 0; p < matrix[0].Count; p++) |
|  | { |
|  | // create submatrix |
|  | List<List<int>> subMatrix = new List<List<int>>(); |
|  | int matrixSize = matrix.Count; |
|  |  |
|  | for (int i = 1; i < matrixSize; i++) |
|  | { |
|  | // iteration will start from row one cancelling the first row values |
|  | List<int> tempRow = new List<int>(); |
|  |  |
|  | for (int j = 0; j < matrixSize; j++) |
|  | { |
|  | // iteration will pass all cells of the i row excluding the j |
|  | //value that match p column |
|  | if (j != p) |
|  | { |
|  | tempRow.Add(matrix[i][j]);//add current cell to TempRow |
|  | } |
|  | } |
|  |  |
|  | // Adding each row to submatrix |
|  | if (tempRow.Count > 0) |
|  | { |
|  | subMatrix.Add(tempRow); |
|  | } |
|  | } |
|  |  |
|  | // recursively calculate the determinant value |
|  | cofactor = matrix[0][p]; |
|  | determinant += (int)(Math.Pow(-1, p)) \* cofactor \* ComputeDeterminant(subMatrix); |
|  | } |
|  | } |
|  |  |
|  | return determinant; |
|  | } |
|  |  |
|  | } |
|  | } |

Question:10

// For this challenge you will be searching a string for a particular substring.

// have the function KUniqueCharacters(str) take the str parameter being passed

// and find the longest substring that contains k unique characters, where k will

// be the first character from the string. The substring will start from the second

// position in the string because the first character will be the integer k.

// For example: if str is "2aabbacbaa" there are several substrings that all contain

// 2 unique characters, namely: ["aabba", "ac", "cb", "ba"], but your program

should

// return "aabba" because it is the longest substring. If there are multiple longest

// substrings, then return the first substring encountered with the longest length.

// k will range from 1 to 6.

Answer:

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  |  |
|  | namespace Coderbyte\_CSharp.Medium\_Challenges |
|  | { |
|  | class StringUniqueSubstring |
|  | { |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  | public string KUniqueCharacters(string str) |
|  | { |
|  | string result = String.Empty; |
|  | bool done = false; |
|  | int uniqueLength = (int)char.GetNumericValue(str[0]); |
|  | string localStr = str.Substring(1); |
|  | int substrLength = localStr.Length; |
|  | //List<string> substrings = new List<string>(); |
|  |  |
|  |  |
|  | while (!done) |
|  | { |
|  | int uniqueCount = 0; |
|  |  |
|  | List<string> substrings = CreateSubstrings(localStr, substrLength); |
|  | if (substrings.Count == 0) |
|  | { |
|  | continue; |
|  | } |
|  |  |
|  | foreach(string s in substrings) |
|  | { |
|  | uniqueCount = ComputeUniqueChars(s); |
|  | if (uniqueCount == uniqueLength && s.Length > result.Length) |
|  | { |
|  | result = s; |
|  | done = true; |
|  | } |
|  | } |
|  |  |
|  | substrings.Clear(); |
|  | substrLength--; |
|  | } |
|  | return result; |
|  | } |
|  |  |
|  | protected int ComputeUniqueChars(string str) |
|  | { |
|  | int count = 1; |
|  | Dictionary<char, int> charCount = new Dictionary<char, int>(); ; |
|  |  |
|  | foreach (char c in str) |
|  | { |
|  | if (charCount.ContainsKey(c)) |
|  | { |
|  | charCount[c]++; |
|  | } |
|  |  |
|  | else |
|  | { |
|  | charCount.Add(c, 1); |
|  | } |
|  | } |
|  |  |
|  | count = charCount.Count; |
|  |  |
|  | return count; |
|  | } |
|  |  |
|  | protected List<string> CreateSubstrings(string str, int length) |
|  | { |
|  | List<string> substrings = new List<string>(); |
|  | int strLength = str.Length; |
|  |  |
|  | if (length == strLength) |
|  | { |
|  | substrings.Add(str); |
|  | } |
|  |  |
|  | else if (length < strLength) |
|  | { |
|  | string substr; |
|  |  |
|  | for (int index = 0; index + length <= strLength; index++) |
|  | { |
|  | substr = str.Substring(index, length); |
|  | substrings.Add(substr); |
|  | } |
|  | } |
|  |  |
|  | return substrings; |
|  | } |
|  |  |
|  | } |
|  | } |



QUESTION BANK DOCKET

**Table of Contents**

[1 Source 2](#_Toc253285289)

[2 Category 3](#_Toc965934774)

[3 Tech Area 3](#_Toc613746458)

[4 Complexity 3](#_Toc904143571)

[5 Question 3](#_Toc1426514880)

[6 Answer 3](#_Toc441207452)

[7 References 3](#_Toc1838997797)

# Source

Leet Code

# Category

|  |
| --- |
| DOT NET |

# Tech Area

C#

# Complexity

Simple

# Question

1.Give a string s, return the longest Palindromic substring in s.

2. Given a string s, find the length of the longest substring without repeating characters.

3. Given an integer n, return  the number of prime numbers that are strictly less than n.

4. Given an integer array nums and an integer k, return *the* kth *largest element in the array*.

Note that it is the kth largest element in the sorted order, not the kth distinct element.

You must solve it in O(n) time complexity.

5. Given a string s and a dictionary of strings wordDict, return true if s can be segmented into a space-separated sequence of one or more dictionary words.

6. There are n gas stations along a circular route, where the amount of gas at the ith station is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from the ith station to its next (i + 1)th station. You begin the journey with an empty tank at one of the gas stations.

Given two integer arrays gas and cost, return *the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return* -1. If there exists a solution, it is **guaranteed** to be **unique**

7. Implement [pow(x, n)](http://www.cplusplus.com/reference/valarray/pow/), which calculates x raised to the power n.

8. Given an unsorted integer array numbers, return the smallest missing positive integer.

You must implement an algorithm that runs in O(n) time and uses constant extra space.

9.Given an array of strings strs, group **the anagrams** together. You can return the answer in any order**.**

An  Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

10. Given an integer array numbers, return the length of the longest strictly increasing subsequence.

# Answer

1.

public class Solution {

public string LongestPalindrome(string s) {

int maxLength = 0, startIndex = 0;

for (int i = 0; i < s.Length; i++) {

int start = i, end = i;

while(end < s.Length-1 && s[start] == s[end+1])

end++;

while(end < s.Length-1 && start > 0 && s[start-1] == s[end+1]) {

start--;

end++;

}

if(maxLength < end - start + 1) {

maxLength = end - start + 1;

startIndex = start;

}

}

return s.Substring(startIndex, maxLength);

}

}

2. public class Solution {

    public int LengthOfLongestSubstring(string s) {

        int maxLength = 0;

        StringBuilder subStr = new StringBuilder();

        for (int i = 0; i < s.Length; i++)

        {

            for (int j = i; j < s.Length; j++)

            {

                if (subStr.Length > 0)

                {

                    if (!subStr.ToString().Contains(s[j]))

                    {

                        subStr.Append(s[j]);

                    }

                    else

                    {

                        break;

                    }

                }

                else

                {

                    subStr.Append(s[j]);

                }

            }

            if (subStr.Length > maxLength)

                maxLength = subStr.Length;

            subStr = new StringBuilder();

        }

        return maxLength;

    }

}

3. public class Solution {

    public int CountPrimes(int n) {

        n--;

        bool[] prime = new bool[n+1];

        for (int i = 0; i <= n; i++) prime[i] = true;

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

        {

            if (prime[i] == true)

            {

                for (int j = i \* i; j <= n; j += i) prime[j] = false;

            }

        }

        int count = 0;

        for(int i=2;i<= n; i++) {

            if (prime[i] == true) count++;

        }

        return count;

    }

}

4.

public class Solution {

    public int FindKthLargest(int[] nums, int k) {

        Array.Sort(nums);

        return nums[nums.Length - k];

    }

}

5.

public class Solution {

    public bool WordBreak(string s, IList<string> wordDict) {

        var dic = new Dictionary<int, bool>();

        var rs = WordBreak(0, dic, s, wordDict);

        return rs;

    }

    private bool WordBreak(int index, Dictionary<int, bool> dic, string s, IList<string> wordDict)

    {

        if (dic.ContainsKey(index)) return dic[index];

        if (index == s.Length) return true;

        var list = GetWords(index, s, wordDict);

        for (int i = 0; i < list.Count; i++)

        {

            if (WordBreak(index + wordDict[list[i]].Length, dic, s, wordDict))

            {

                if (!dic.ContainsKey(index)) dic.Add(index, true);

                return true;

            }

        }

        if (!dic.ContainsKey(index)) dic.Add(index, false);

        return false;

    }

    private List<int> GetWords(int index, string s, IList<string> wordDict)

    {

        var rs = new List<int>();

        for(int i = 0; i < wordDict.Count; i++)

        {

            if (CheckWord(index, s, wordDict[i])) rs.Add(i);

        }

        return rs;

    }

    private bool CheckWord(int index, string s, string word)

    {

        if (word.Length - 1 + index >= s.Length) return false;

        for (int i = 0; i < word.Length; i++)

        {

            if (word[i] != s[index + i]) return false;

        }

        return true;

    }

}

6. public class Solution {

    public int CanCompleteCircuit(int[] gas, int[] cost) {

    int i=0, n=gas.Length;

        if(gas.Sum() <cost.Sum()){

            return -1;

        }

        while(i<n){

            if(gas[i]>=cost[i]){

                int start=i, left=gas[i]-cost[i], j=i+1;

                while(left+gas[j%n]>=cost[j%n]){

                    if(j%n==start)

                        return start;

                    left = left+gas[j%n]-cost[j%n];

                    j++;

                }

                i=j;

                continue;

            }

            i++;

        }

        return -1;

    }

}

7. public class Solution {

    Dictionary<int,double> d=new Dictionary<int,double>();

    public double MyPow(double x, int n) {

        if(n>=0)

            return helper(x,n);

         return helper(1/x,-n);

    }

    public double helper(double x, int n) {

        if(d.ContainsKey(n))

            return d[n];

        if(n==0){

            d[0]= 1;

             return d[0];}

        if(n==1){

            d[1]= x;

             return d[1];}

        if(n==2){

            d[2]= x\*x;

            return d[2];

        }

        if(n%2==0){

            d[n] =helper(x,n/2)\*helper(x,n/2);

            return d[n];}

            else{

        d[n]= helper(x,n/2)\*helper(x,n/2)\*x;

        return d[n];

                }

    }

}

8.

public class Solution {

    public int FirstMissingPositive(int[] nums)

    {

        bool[] bums = new bool[nums.Length+1];

        for (int i=0; i<nums.Length; i++) {

            if (nums[i] > 0 && nums[i] <= nums.Length)

            {

                bums[nums[i]] = true;

            }

        }

        for (int i=1; i<bums.Length; i++) {

            if (bums[i] == false)

            {

                return i;

            }

        }

        return bums.Length;

    }

}

9.

public class Solution {

   public IList<IList<string>> GroupAnagrams(string[] strs) {

        Dictionary<string, IList<string>> dic = new Dictionary<string, IList<string>>();

        for (int i = 0; i < strs.Length; i++)

        {

            char[] arr=strs[i].ToCharArray();

            Array.Sort(arr);

            string sorted=new string(arr);

            if(dic.ContainsKey(sorted))

            {

                dic[sorted].Add(strs[i]);

            }

            else

            {

                dic.Add(sorted, new List<string>() {strs[i]});

            }

        }

        var result= dic.Values.ToList();

        return result;

}

}

10.

public class Solution {

    public int LengthOfLIS(int[] nums) {

        int count = 0;

        int[] dp = new int[nums.Length];

        int last = nums.Length-1;

        Array.Fill(dp, 1);

        int res = 0;

        for(int i=last; i>=0; i--){

            for(int j=i+1; j<nums.Length; j++){

                if(nums[j] > nums[i]){

                    dp[i] = Math.Max(dp[i], dp[j]+1);

                }

            }

            res = Math.Max(res, dp[i]);

        }

        return res;

    }

}

# References

[Discussion Boards, Comments, Links, etc..]x

https://leetcode.com/problemset/all/