Home Work № 2

CSCI B505 Applied Algorithms 09/09/2022

Instructions

1. Under no circumstance the deadline will NOT be extended
2. Code should be written in python language in a Jupyter notebook
3. Upload Jupyter notebook and html file of the Jupyter notebook in canvas. No other forms submissions is considered as valid submission. Make sure to have the output of the required cells of the jupyter notebook and its html version before making submission.
4. Plagiarism is unacceptable and we have ways to find it. So do not do it.
5. There are eight problems. Problem 2 and Problem 7 - 20 points , rest of the problems 10 points each.
6. The code should readable with variables named meaningfully
7. Write test cases wherever required so that they cover all scenarios.

Problem 1

Fräulein Anja and Julia bought a rectangular Black Forest cake.
Black Forest cakes have cherries on top of them which they both love. Anja would like your help to cut the cake into two smaller rectangular pieces in such a way that she gets the maximum possible cherries after Julia takes the best piece (the one with more cherries) out of the two pieces.

Input will be a matrix representing the cake. Here '#' represents a cherry and '.' represents no-cherry (just cake). Your output will be the maximum possible number of cherries on Anja's piece after Julia takes the piece with more cherries. Note that you are allowed to cut the cake only once, and the cut has to be either horizontal or vertical.

```
Example 1:

Input:  [['.', '#', '.'], ['#', '.', '.'], ['#', '#', '#']]

. # .
# . .
# # #
```

Output: 2

Explanation:

If we cut the cake this way:

```
. # .
------
# . .
# # #
```

Julia will take the piece with 4 cherries and Anja will be left with only 1 cherry. Instead, if we cut the this way, Anja gets 2 cherries after Julia takes the piece with three cherries.

```
. # .
# . .
------
# # #
```

Example 2:

Input: [['.', '#', '#', '#', '#'], ['.', '.', '.', '.', '.']]

```
. # # # #
. . . . .
```

Output: 2

Explanation:

Answer is 2 because we can cut the cake like this:

```
      |
. # # | # #
. . . | . .
      |
```

```
1
2 def max_cherries(cake):
3     #***logic****#
4
5     #************#
6     return (max possible no of cherries on Anja piece)
```

Problem 2

Philippa (more formally, Philippa Georgiou Augustus Iaponius Centarius,
her most Imperial Majesty, Mother of the Fatherland, Overlord of Vulcan, Dominus of Qo'noS, Regina Andor) has finally found the locations of the resistance leaders (a.k.a traitors).
They are all hiding in a small Terran city that has a weird geometry. All the buildings are on a straight line! She would like to spy on the resistance leaders in the city. Your job, as Philippa's adviser, is to find the buildings on which to post spies such that you can spy on the entire city using as few spies as possible.

A spy posted on a building can spy on people in the buildings to the right which are shorter than the building she is posted on (Of course, she can also spy on the people in her building).

You may assume that all buildings have distinct heights. Your input will be the heights of the buildings, and your output has to be the locations (indices) of the buildings on which to post spies.

```
Example 1:

Input: [4, 3, 2, 1]

Output: [0]

The building at 0 is the tallest building, if we post a spy on it, we can spy on all
the buildings to the right.

Example 2:

Input: [1, 2, 5, 7]

Output: [0, 1, 2, 3]

We need spies on every building because every building to the right is taller than
the one before it. For instance, a spy posted on building at index 2 (with height 5)
cannot see building at index 3 (with height 7).
```

```
Example 3:

Input: [4, 3, 7, 6, 9]

Output: [0, 2, 4]

We need a spy on building 0 as there are no buildings to its left.
We don't need a spy on building 1 as the spy on building 0 can spy on building 1.
We need one on building 3 as spy on building 0 cannot spy on building 3 an so on...
```

```
1
2 def spy_locations(heights):
3     #***logic****#
4
5     #************#
6     return (location of buildings which to post spies)
```

Problem 3

Veidt and Walter are taking turns while playing a game. Given a string pattern p of length n and a string s of length N>n on each turn a player has to remove a character either from the beginning or the end of the string s. Game stops when the size of string s becomes n. Walter wins if the pattern p and the resultant string s are identical. Else Veidt wins. Assuming Veidt always makes the first move and both the players always play using optimal strategy, return the winner of the game. Expected Time Complexity O(n)

```
Example 1:
    S= ''Abcdefghij''
    P=''cdef''
    Answer: Veidt
```

```
1     def whos_the_winner(s):
2         #***logic****#
3
4          #************#
5         return winner(Veidt or Walter)
```

Problem 4

Given a pattern p and a text string s, remove the pattern from the string s whenever it's instance is found as a substring and concatenate the remaining two parts of the string together. Repeat the above process till there are no more substrings in s that are equal to p and return the resultant string

```
Example 1:
 S= ‘‘aababccbc’’
 P=’’abc’’
 Answer: ‘‘’’
```

```
1    def return_resultant_string(s):
2        #***logic****#
3
4         #***********#
5        return result
```

Problem 5

Given a binary string of n characters and a list of strings queries where each element in the list is either “flip” or “get”. For each element query in “queries”, if the query is “flip” you have to find the first “1” from the left, set it to ‘0’ and set all the ‘0’s before it to ‘1’.
Whenever the query is “get” you have to append the number of ‘1’s in the binary string to a “result” list. After accounting for all the queries return the list of integers result. The operations “flip” and “get” are to be implemented in O(1) complexity.
One may safely assume there is at least one ‘1’ in the string when either of the above functions is called.

```
Example 1:
    s=’’0000101011’’
    queries=[‘‘get’’,’’flip’’,’’flip’’,’’get’’,’’flip’’,’’flip’’,’’flip’’,’’get’’]
    Answer: [4,6,6]
```

```
1    def flip_game(s,queries):
2        #***logic****#
3
4        #***********#
5      return result_list
```

Problem 6

You are provided with a scoresheet, Calculate the total number of points scored at the end of the game by following the rules.

Rules:

1. An integer x - Record a new score of x.
2. [+, -, *, %, /]- Record a new score by performing a given operation on previous two scores. It is guaranteed there will always be two previous scores.
3. "D" - Record a new score that is double the previous score. It is guaranteed there will always be a previous score.
4. "I" - Invalidate the previous score, remove it from the record. It is guaranteed that there will always be a previous score.

   Note: - For "/" operation consider the floor division of the given 2 numbers.

```
Example 1:
    Input: scoresheet = ["5","2",’’I","D","+"]
    Output: 30
Explanation:
    "5" - Add 5 to the record, record is now [5].
    "2" - Add 2 to the record, record is now [5, 2].
    "I" - Invalidate and remove the previous score, record is now [5].
    "D" - Add 2 * 5 = 10 to the record, record is now [5, 10].
    "+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].
    The total sum is 5 + 10 + 15 = 30.

Example 2:
    Input: scoresheet = ["5","-2","4","I","D","9","+","/"]
    Output: 14
Explanation:
    "5" - Add 5 to the record, record is now [5].
    "-2" - Add -2 to the record, record is now [5, -2].
    "4" - Add 4 to the record, record is now [5, -2, 4].
    "I" - Invalidate and remove the previous score, record is now [5, -2].
    "D" - Add 2 * -2 = -4 to the record, record is now [5, -2, -4].
    "9" - Add 9 to the record, record is now [5, -2, -4, 9].
    "+" - Add -4 + 9 = 5 to the record, record is now [5, -2, -4, 9, 5].
    "/" - Add 9 / 5 = 1 to the record, record is now [5, -2, -4, 9, 5, 1].
    The total sum is 5 + -2 + -4 + 9 + 5 + 1 = 14.
```

```
    def calculate_points (list):
        #***logic****#

        #***********#
    return sum(scoresheet)
```

Problem 7

Write a program which performs following operations in the Linked List.
Insert: Given an element and position , insert the element at that position in the linked list. If the position is not valid , do nothing

```
Example : 1 -> 2 -> 3 -> 4
Insert 5 at 2
Output : 1 -> 2 -> 5 -> 3 -> 4
```

Delete: Given an element, delete that element. If given search element is not available then do nothing

```
Example : 1-> 2-> 3->4
delete 2
Output : 1-->3->4
```

Update : Given an element replace with the new element. If given search element is not available then do nothing

```
Example: 1->2->3->5
Update 2 with 7
Output: 1->7->3->5
```

Search: Given an element return True if it is in the linked List else False.

```
Example: 1->2->3->5
Search 6
Output: False


Example: 1->2->3->5
Search 5
Output: True
```

Given two sorted linked list, return a sorted new linked list with merging of given linked lists.

```
Example: a) 1->3->5 b) 2->4->6
Output: 1->2->3->4->5->6
```

Note: You are Free to choose Single Linked List or Double Linked List; The elements in the linked list are unique; For every operation print the linked List.

```
1     class Node:
2         def __init__(self, num, next = None ):
3             self.num = num
4             self.next = next
5
6     class Linked_List():
7
8         def Insert( self, element, position):
9             #***logic****#
10
11        def delete(self, element):
12            #***logic****#
13
14        def Update(self, search_element, replace_element):
15            #***logic****#
16
17        def Search(self, element):
18            #***logic****#
19
20        def merge_linkedlists(self, Linkedlist);
21            #***logic****#
22            return(Sorted Linked List)
23
24        def print(self):
25            #***logic****#
```

Problem 8

There is a colony of ants where each ant has been assigned a positive number as its identity. Ants returning from food hunt form a line outside the nest. Their nest has a single narrow entrance where only one ant, at a time can enter the nest through it.
The colony has a rule, where any ant with number 'x', arriving at the entrance cannot enter the nest before all the preceding numbered ants have entered (x-1,x-2,...) the nest [ant 4 cannot enter till ants 3,2 and 1 have entered the nest].
There is a narrow tunnel near the entrance where any number of ants can 'wait' if required, till it's their turn to enter the nest. Any ant that enters the waiting tunnel last must necessarily leave the tunnel first, and any ant leaving the waiting tunnel can only go to the entrance of the nest, and cannot re-enter the arrival queue.
You are given the order in which 'n' ants arrive at the entrance in an array containing numbers 1 to n. write a function that returns 1 if all ants can enter the nest without breaking the rule(i.e

in order 1,2,3,.. etc.) and 0 if they cannot.

```
Example 1:
    Arrival queue = [5,3,2,1,4] (order of arrival is 5,3,2,1 & 4)
    ans: 1
    explanation:
    first arriving ant is 5, which cannot enter as 4,3,2,1
    have not entered the nest yet. So 5 goes into the waiting tunnel.
    Lets call it 'T'.
    Arrival queue = [3,2,1,4]
    T =[5]
    Next ant in line 3 also cannot enter the nest as 2 and 1
    have not entered the nest yet. So 3 goes into the waiting tunnel.
    Arrival queue = [2,1,4]
    T =[3,5]
    Next ant in line 2 also cannot enter the nest as 1 has not entered the nest yet.
    So 2 goes into the waiting tunnel.
    Arrival queue = [1,4]
    T =[2,3,5]
    Next ant in line 1 can enter the nest.
    Arrival queue = [4]
    T =[2,3,5]
    nest=[1]
    Now ant 2 comes out of the waiting tunnel and goes into the nest,
    then 3 follows the similar path:
    Arrival queue = [4]
    T =[5]
    nest=[3,2,1]
    Now 4 can enter the nest from the arrival queue,
    then finally 5 can exit the tunnel and enter nest:
    Arrival queue = []
    T =[]
    nest=[5,4,3,2,1]

Example 2:
    Arrival queue = [4,1,5,3,2]
    ans: 0
    Explanation: following the similar logic, we cannot progress beyond:
    Arrival queue = []
    T =[5,4]
    nest=[3,2,1]
    at this point, 5 cannot go into nest from the waiting tunnel
```

(as 4 is not in the nest) or it cannot re-enter the arrival queue to let 4 exit the tunnel.

```
    def validate_order(nums):
        # nums: array of length n containing integers from 1 to n
        ####logic#####

        ## ans: 0 or 1#
    return ans
```