CS Scholars Programming Hw4 - Written

Due Date: Friday 07/30 EOD

Name:

AndrewID:

For full credit on the assignment, complete either all Review + Core problems (#1-#4, #7-#12) or all Core + Spicy problems (#3-#12).

Bonus problems are related to the Advanced Track content, and are optional.

Review Problems - Written [20pts]

#1 - String Code Tracing - 10pts

Can attempt after Strings and Lists lecture

Assuming that the following two lines of code have been run:

```
s1 = "15-110 is cool"
s2 = "CMU rocks!"
```

What will each of the following expressions evaluate to? Don't just run the code in the editor- try to figure out the answer on your own.

Expression	**Value**
`s1[7] + s2[6]`	
`s1[1] + s2[len(s2)-1]`	
`s1[4:8]`	
`s2[2:len(s2)-2]`	
`s1[::4]`	

#2 - Files in Code - 10pts

Can attempt after Data Analysis lecture

We've written some code to count the number of times the string `"Hello World"` occurs within a file:

```
f = open("sample.txt", "r")
print(f.count("Hello World"))
f.close()
```

There is a bug in this code that will make it raise a runtime error, even if sample.txt is located in the same directory as the code. What is the bug, and how could you fix it?

Core Problems - Written [20pts]

#3 - Aliasing and Mutability - 10pts

Can attempt after References and Memory lecture

The following code creates and modifies lists. Determine each list's values after the code has run.

```
a = [ "apple", "banana", "carrot", "donut" ]
b = a
b.remove("apple")
c = a + [ "eclair" ]
d = c[1:]
d.insert(2, "fig")
```

Variable	**List Values**
a	
b	
c	
d	

Select all of the pairs of lists that are **aliased** at the end of the code.

- ☐ **a** and **b**
- ☐ **a** and **c**
- ☐ **a** and **d**
- ☐ **b** and **c**
- ☐ **b** and **d**
- ☐ **c** and **d**
- ☐ None of the lists are aliased

#4 - Parsing Data - 10pts

Can attempt after Data Analysis lecture

You have been given a set of data about CMU classes in the following format (whitespace added for clarity):

```
Professor,      ClassNum,  Days,      Time
Cortina,        15104,     MTWF,      09:20-10:10
Rivers,         15110,     MWThF,     16:00-16:50
Touretzky,      15110,     MWThF,     17:20-18:10
Touretzky,      15394,     MW,        19:00-20:20
```

Assume you've already split the string on `"\n"` and used the variable `row` to iterate through each class one line at a time.

Each class number represents the class's department (first two digits) and identifying number (last three digits). How would you determine which department each class is in?

- ☐ `row.split(',')[1][0:1]`
- ☐ `row.split(',')[1][0:2]`
- ☐ `row.split(',')[1][2:]`

How would you determine the start time and end time of a class and set those times as strings in the variables `start` and `end`? **Select all lines that are needed.** Assume that the code is run from the top selected line to the bottom selected line.

- ☐ `times = row.split(',')[0]`
- ☐ `times = row.split(',')[3]`

- ☐ `start = row.split('-')[0]`
- ☐ `start = times.index('-') - 1`
- ☐ `start = times.split('-')[0]`

- ☐ `end = times.index('-') + 1`
- ☐ `end = times.split('-')[1]`
- ☐ `end = start + "1:00"`

Spicy Problems - Written [20pts]

#5 - Advanced String Code Tracing - 10pts

Can attempt after Strings and Lists lecture

What will the following code print? Don't just type up the code and run it- try to figure out the answer on your own.

```
def mystery(s):
    a = ""
    b = ""
    for i in range(len(s)):
        if s[i] in s[i+1:]:
            n = s[i:].count(s[i])
            print(str(i) + "-" + str(n))
            a = a + s[i]
        else:
            b = s[i] + b
    return a + "," + b

print(mystery("xyzxyx"))
```

#6 - Advanced Files in Code - 10pts

Can attempt after Data Analysis lecture

Consider the following file code, which is supposed to take a file containing lines of text and write a file with the same text, but with each word capitalized:

```
f = open("test.txt", "r")
text = f.read()

words = text.split(" ")
for i in range(len(words)):
    words[i] = words[i].capitalize()
newText = " ".join(words)

f = open("test.txt", "w")
f.write(newText)
f.close()
```

This code can run without raising an error message, and it even works properly on some files. But it isn't **robust** - it may cause errors (runtime or logical) in some conditions.

Identify at least two robustness problems in the code, and explain what you would do to fix each of them.

HINT: consider what the code would do if you ran it on a file containing the text:

```
Hello world, how are you today?
I'm doing great, thanks!
```

Programming Problems

Each of these problems should be solved in the starter file available on the course website. They should be submitted to the Gradescope assignment Hw4 - Programming to be autograded.

Make sure to 'Run file as Script' to check your work before submitting, then check the autograder feedback after you submit!

Core Problems - Programming [60pts]

#7 - `getSecretMessage(s, key)` - 10pts

Can attempt after Strings and Lists lecture

You can hide a secret message in a piece of text by setting a specific character as a key. Place the key before every letter in the message, then fill in extra (non-key) letters between key-letter pairs to hide the message in noise.

For example, to hide the message "computer" with the key "q", you would start with "computer", turn it into "q**c**q**o**q**m**q**p**q**u**q**t**q**e**q**r**", and then add extra letters as noise, perhaps resulting in "orup**qc**rzyp**qo**m**qm**hcy**qp**whh**qu**t**qt**xt**qe**ye**qr**pa". To get the original message back out, copy every letter that occurs directly after the key, ignoring the rest.

Write a function `getSecretMessage(s, key)` that takes a piece of text holding a secret message and the key to that message and returns the secret message itself. For example, if we called the function on the long string above and "q", it would return "computer". You are guaranteed that the key does not occur in the secret message.

Hint: loop over every character in the string. If the character you're on is the key, add the **next** character in the string to a result string.

#8 - `sumAnglesAsDegrees(angles)` - 10pts

Can attempt after Strings and Lists lecture

When analyzing data, you need to convert the data from one format to another before processing it. For example, you might have a dataset where angles were measured in radians, yet you want to find the sum of the angles in degrees.

Write the function `sumAnglesAsDegrees(angles)` which takes a list of angles in radians (floats) and returns the sum of those angles **in degrees** (an integer). To do this, you will need to change each angle from radians to degrees before adding it to the sum. You can do this with the library function `math.degrees()`. Make sure to round the final result to get an integer answer.

For example, `sumAnglesAsDegrees([math.pi/6, math.pi/4, math.pi])` should convert the radians to approximately `30.0`, `45.0`, and `180.0`, then return `255`.

#9 - `getCharacterLines(script, character)` - 10pts

Can attempt after Strings and Lists lecture

Assume you're provided a string script that has been formatted in a specific way. Each line of the script begins with a character's name, followed by a colon, followed by their line of dialogue. Lines are separated by newlines, which are represented in Python by the string `'\n'`. For example:

```
'''Buttercup: You mock my pain.
Man in Black: Life is pain, Highness.
Man in Black: Anyone who says differently is selling something.'''
```

Write the function `getCharacterLines(script, character)`, which takes a script and a character name (both strings) and returns a list of the lines spoken by that character. The lines should be stripped of the leading character name and any leading/trailing whitespace. So if we use the following script:

```
'''Burr: Can I buy you a drink?
Hamilton: That would be nice.
Burr: While we're talking, let me offer you some free advice: talk less.
Hamilton: What?
Burr: Smile more.
Hamilton: Ha.
Burr: Don't let them know what you're against or what you're for.
Hamilton: You can't be serious.
Burr: You want to get ahead?
Hamilton: Yes.
Burr: Fools who run their mouths oft wind up dead.'''
```

Then:

```
getCharacterLines(script, "Hamilton") ==
[ "That would be nice.", "What?", "Ha.", "You can't be serious.", "Yes." ]
```

Hint: you'll want to use string and list **methods** and **operations** to make this problem more approachable. Specifically:

- `split` can help you separate the lines of text
- `index` can help you locate where a line of text switches from name to dialogue
- slicing can help you separate the name from the dialogue
- `strip` can remove excess whitespace from the front and end of the string

#10 - `onlyOdds(lst)` and `removeEvens(lst)` - 10pts

Can attempt after References and Memory lecture

First, write a **non-destructive** function `onlyOdds(lst)` that takes a list of integers and returns a **new** list containing only the odd elements of `lst`. Note that this should not return the odd indexes- it should return the odd **elements**!

For example, `onlyOdds([1, 2, 3, 4, 5, 6])` returns `[1, 3, 5]`, and `onlyOdds([4, 1, 70, 35, -9])` returns `[1, 35, -9]`

Then write a **destructive** function `removeEvens(lst)` that takes a list of integers and destructively removes the even elements of the provided list so that it contains only the original odd elements at the end of the function. This function should return `None` instead of the list; we'll test it by checking whether the input list was modified properly.

For example, `removeEvens([1, 2, 3, 4, 5, 6])` modifies the list to be `[1, 3, 5]`, while `removeEvens([4, 1, 70, 35, -9])` modifies the list to be `[1, 35, -9]`.

Hint: this is tricky because `lst` will change as the function runs. You should use an appropriate loop to account for this - see the course slides! Also, make sure to check for aliasing issues.

#11 - `drawPixelArt(root, template, pixelSize, colors)` - 10pts

Can attempt after References and Memory lecture

Pixel art is an art style where you draw pictures by piecing together oversized pixels (dots of color) into an image. When we draw graphics in Tkinter, we're technically drawing with pixels, but the pixels are so small that it's hard to tell.

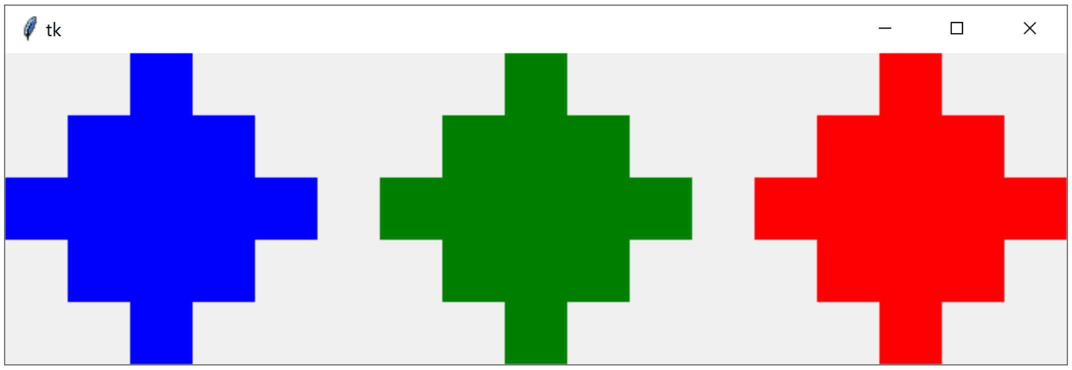
Write the function `drawPixelArt(root, template, pixelSize, colors)` which takes a Tkinter root, a template string, an integer pixel size, and a list of colors (strings), and generates a piece of pixel art based on that information.

The template is a string that contains only spaces, newlines, and numbers. Each character represents a square pixel of size `pixelSize` (with `'\n'` splitting the lines). If a pixel is represented by a space, nothing is drawn there. If it's represented by a number, the color at that index of the `colors` list is drawn there as a square.

For example, if we call the function on the template:

```
  0     1     2
 000   111   222
00000 11111 22222
 000   111   222
  0     1     2
```

With the color list `["blue", "green", "red"]`, we should get:



[continued on next page]

Unlike other Tkinter functions you've written, you need to generate the canvas yourself for this problem. To do this, use this line of code, which calls a helper function:

```
canvas = makeCanvas(root, width, height)
```

If you convert the template to a list (by splitting over the newlines), it's sort of like a 2D list- each row is just represented as a string instead of a list. `width` and `height` can be calculated as the width (length of each line) and height (number of lines) of that 2D list, modified by `pixelSize`. (You're guaranteed that all lines in the template will have the same number of pixels).

For example, in the pixel art above, if the pixel size is set to 40, the width of the screen should be 17*40 = 680 and the height of the screen should be 5*40 = 200.

Once you've generated the canvas, you can iterate over the template, access each character, and draw each pixel using similar logic to how we drew a grid in a previous lecture. Make sure to use `pixelSize` to make the pixels the appropriate size!

Bonus: for extra fun, try setting up your own pixel art string to call with your function! We've set up a function `myPixelArt` where you can insert your own template and color list. Cool pixel art will be eligible for a max of three bonus points on the homework assignment.

#12 - `gradebookSummary(filename)` - 10pts

Can attempt after Data Analysis lecture

Assume that you have a gradebook for a class stored in a `.csv` file. Each line of the gradebook contains a student's ID followed by one or more grades, all separated by commas. A gradebook always contains at least one student, and each row always contains at least one grade. Gradebooks can also contain blank lines.

Write the function `gradebookSummary(filename)` that takes the filename of a gradebook as an argument and writes a new file (with the name `summary_` followed by the original filename) containing a summary of the gradebook, returning `None`. This summary file should also be a csv, but with just three columns: student IDs, mean grades, and median grades.

In the result file, students should be listed in their original order (separated by newlines), grades should be rounded to the hundredth digit (with the `round` function), and there should be no extra blank lines. Also, there should be a header at the top of the file with the column names: **ids**, **mean**, and **median**.

For example, if the file `gradebook1.txt` contains the following text:

```
wilma,91,93,94
fred,80,85,90,97,100
betty,88
```

Then `gradebookSummary("gradebook1.txt")` should create a file `summary_gradebook1.txt` with the following text:

```
id,mean,median
wilma,92.67,93
fred,90.4,90
betty,88,88
```

Hint: try using the `csv` and `statistics` libraries we discussed in class! They will make this problem much easier, especially if you make a list of grades (converted to integers) for each student.

Bonus Problems [10pts]

Advanced Programming 1 - Dictionaries - 2pts

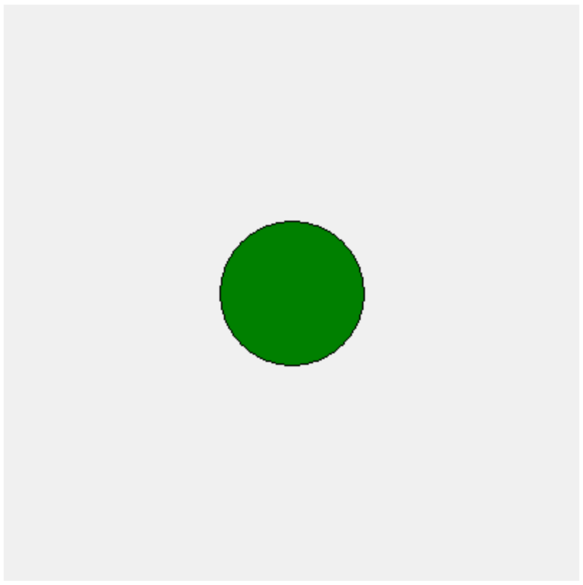
Write the tkinter function `generateBubbles(canvas, bubbleList)` which takes a tkinter canvas and a **list of dictionaries**, `bubbleList`, and draws bubbles as described in `bubbleList`.

Each dictionary in the bubble list contains exactly four keys: `"left"`, `"top"`, `"size"`, and `"color"`. The first three all map to integers (the left coordinate, top coordinate, and diameter size of the bubble), and the fourth maps to a string (its color). Use this information to draw the bubble (with `canvas.create_oval`) in the appropriate location, with the correct size and color.

For example, if we make run the function with the bubble list from the first test:

```
bubbleList1 = [ {"left":150, "top":150, "size":100, "color":"green"} ]
```

We'll get:

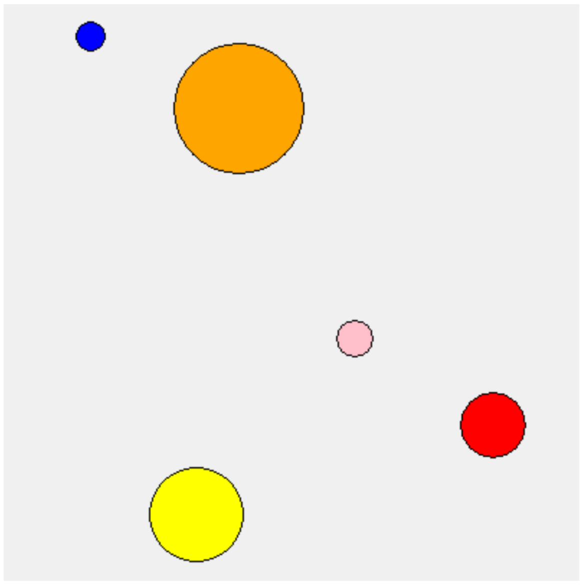


[continued on next page]

And the second test, which has:

```
bubbleList2 = [
     {'left': 317, 'top': 269, 'size': 45, 'color': 'red' },
     {'left': 118, 'top': 27, 'size': 90, 'color': 'orange'},
     {'left': 101, 'top': 321, 'size': 65, 'color': 'yellow'},
     {'left': 231, 'top': 219, 'size': 25, 'color': 'pink' },
     {'left': 50, 'top': 12, 'size': 20, 'color': 'blue' } ]
```

Should produce this:



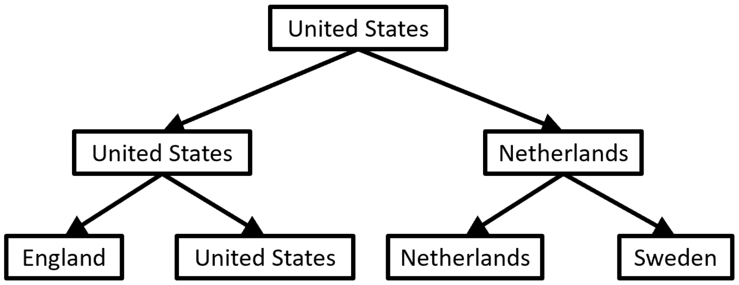
The third test randomly generates 10 bubbles using the provided `makeNBubbles(n)` function. Try changing the size of `n` to generate more or less bubbles, and see how it looks! Your bubbles will be different every time.

Hint: a list of dictionaries might sound intimidating at first, but it's not so bad! Just loop over the list, access the dictionary using the loop control variable, then index into the dictionary to get the needed values.

Advanced Programming 2 - Trees - 1.5pts

We can represent a tournament bracket from a sports competition as a binary tree. To do this, store the winning team as the root node. Its children are the winning team again, as well as the second-place team. In general, every node represents the winner of a match, and its two children are the two teams that competed in that match.

For example, the following bracket represents the last two rounds of the Women's World Cup in 2019.



In our binary tree dictionary format, this would look like:

```
t1 = { "contents" : "United States",
       "left" : { "contents" : "United States",
           "left"  : { "contents" : "England", "left" : None, "right" : None },
           "right" : { "contents" : "United States", "left" : None, "right" : None}},
       "right" : { "contents" : "Netherlands",
           "left"  : { "contents" : "Netherlands", "left" : None, "right" : None },
           "right" : { "contents" : "Sweden", "left" : None, "right" : None } }
     }
```

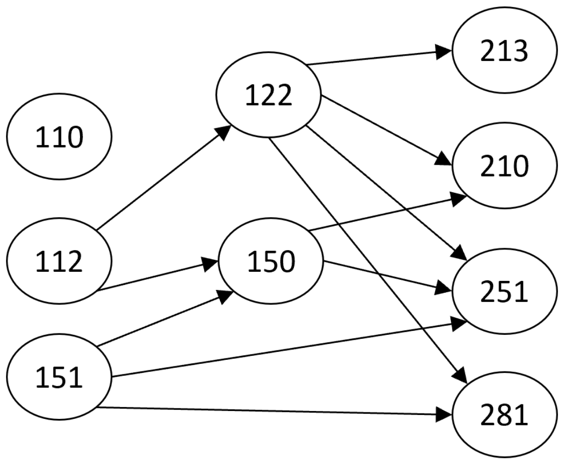
Write the function `getInitialTeams(bracket)` which takes a tournament bracket and returns a list of all the teams that participated in that tournament. For example, if the function is called on the tree above it might return `[ "England", "United States", "Netherlands", "Sweden" ]`. You will need to implement this function **recursively** to access all the nodes. We recommend that you start by looking at the `sumNodes` and `listValues` examples from the slides.

Hint 1: how can we get all of the teams to show up in the list exactly once? Every team occurs at the very beginning of the tournament, in the first set of matches. In the tree, this is represented by the **leaves,** so you should not include values on non-leaf nodes.

Hint 2: make sure the **type** you return is the same in both base and recursive cases!

Advanced Programming 3 - Graphs - 1.5pts

In college, you get to select your own schedule of classes. However, to take certain classes you must first pass an earlier class in the course sequence- a **prerequisite**. These prerequisites are notoriously complicated. However, we can make them a little easier to understand by representing the course dependency system as a **directed graph**, where the nodes are courses and an edge leads from course A to course B if A is a prerequisite of B. For example, the core Computer Science courses (almost) produce the following prereq graph:



Which would be represented in code as:

```
g = { "110" : [],
      "112" : ["122", "150"],
      "122" : ["213", "210", "251", "281"],
      "151" : ["150", "251", "281"],
      "150" : ["210", "251"],
      "213" : [],
      "210" : [],
      "251" : [],
      "281" : [] }
```

Write the function `getPrereqs(g, course)` that takes a directed graph (in our adjacency list dictionary format, without weights) and a string (a course name) and returns a list of all the immediate prerequisites of the given course. If we called `getPrereqs` on our graph above and `"210"`, for example, the function should return `["122", "150"]`.

Hint: you can't just return the neighbors of the course, because the edges are going in the opposite direction! Instead, iterate over all the nodes to find those that have the course as a neighbor. Construct a new list out of these nodes as the result.

Advanced Computer Science 1 - Best Case, Worst Case - 2pts

For each of the following functions, describe an input that would result in **best-case efficiency,** then describe an input that would result in **worst-case efficiency**. This generic input must work at **any possible size**; don't answer 1 for `isPrime`, for example.

```python
def getEmail(words):
    # words is a list
    for i in range(len(words)):
        if "@" in words[i]:
            return words[i]
    return "No email found"
```

```python
def isPrime(num):
    for factor in range(2, num):
        if num % factor == 0:
            return False
    return True
```

What is a **best case input** for `getEmail`?

What is a **worst case input** for `getEmail`?

What is a **best case input** for `isPrime`?

What is a **worst case input** for `isPrime`?

Advanced Computer Science 2 - Big-O Runtimes - 2pts

For each of the following functions, check the **Big-O function family** that function belongs to. You should determine the function family by considering how the number of steps the algorithm takes grows as the size of the input grows.

```
def countEven(L): # n = len(L)
  result = 0
  for i in range(len(L)):
    if L[i] % 2 == 0:
      result = result + 1
  return result
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L)
def sumFirstTwo(L):
  if len(L) < 2:
    return 0
  return L[0] + L[1]
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L1) = len(L2)
def binarySearchAll(L1, L2):
  count = 0
  for item in L1:
    # Hint: what's the complexity of
    # binary search?
    if binarySearch(L2, item) == True:
      count = count + 1
  return count
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L); original call has i = 0
def recursiveSum(L, i):
  if i == len(L):
    return 0
  else:
    return L[i] + recursiveSum(L, i+1)
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

Advanced Computer Science 3 - Hashing - 1pt

Recall our discussion of what hash functions are and what they are used for. Below we've listed four different scenarios. Each scenario contains a data set, a hash function, and which values will need to be looked up in the hashtable. Select **all** the scenarios where you will generally be able to look up the given values in **constant time**.

- ☐ Given a set of integer phone numbers, hash a phone number based on the phone number itself. Use the hashtable to look up an individual phone number.
- ☐ Given a set of all the college essays sent to CMU (as strings), hash an essay based on the first character of the essay ("I want to go to CMU because.." hashes based on "I"). Use the hashtable to look up an individual essay.
- ☐ Given a set of string full names (like "Farnam Jahanian"), hash a name by adding together the numeric ASCII values of all the letters in the name. Use the hashtable to look up an individual name.
- ☐ Given a set of lists of high scores (so each list contains integers), hash a list based on the sum of its scores. Lists can be updated after hashing when new high scores are added. Use the hashtable to look up an individual high-score list.
- ☐ None of the situations described above can be searched in constant time.