Exercise 6 (for grade) ~ Wednesday, October 26, 2022 ~ CPSC 535 Fall 2022

Write one submission for your entire group, and write all group members' names on that submission. Turn in your submission before the end of class. The ❎ symbol marks where you should write answers.

Recall that our recommended problem-solving process is:

1. **Understand** the problem definition. What is the input? What is the output?
2. **Baseline** algorithm for comparison
3. **Goal** setting: improve on the baseline how?
4. **Design** a more sophisticated algorithm
5. **Inspiration** (if necessary) from patterns, bottleneck in the baseline algorithm, other algorithms
6. **Analyze** your solution; goal met? Trade-offs?

Follow this process for each of the following computational problems. For each problem, your submission should include:

a. State are the input variables and what are the output variables
b. Pseudocode for your baseline algorithm, that needs to include the data type and an explanation for any variable other than input and output variables
a. The Θ-notation time complexity of your baseline algorithm, with justification.

and if you manage to create an improved algorithm:

c. Answer the question: how is your improved algorithm different from your baseline; what did you change to make it faster?
d. Pseudocode for your improved algorithm, that needs to include the data type and an explanation for any variable other than input and output variables
a. The Θ-notation time complexity of your improved algorithm, with justification.

Today's problems are:

1.

Given the text T=Jimy_ran_and_hailed_the_monkey_to_stop and P=monkey, show how Horspool or Boyer-Moore algorithm works for searching the pattern P in the text T.

a) Construct the shift / bad shift table

b) Apply the Horspool or Boyer-Moore algorithm and show the shift after each unsuccessful match.

2.

Given the text T = t= 2359024411526739921 and the pattern P = 44115

(a) State the value of the prime q chosen to compute the hash values for the pattern and all the substrics T[s..s+2]

(b) Apply the Rabin-Karp algorithm to compute the indices where the pattern P exists in the text.

(c) State the number of valid matches and spurious hits in the Rabin-Karp algorithm

3.

Design an algorithm that, given an array ***S*** of ***n*** integers, identifies all integers that do not repeat, i.e. that show up only once in the array ***S***.

Your goal: baseline $O(n^2)$ time, improved to either O(n log n) worst-case time or O(n) expected time.

Names

Write the names of all group members below.

☒ Angelo Salac

Joseph Maa

Rosa Cho

Exercise 1: Solve and provide answer

☒ Bad Shift Table:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
6	6	6	6	1	6	6	6	6	6	2	6	5	3	4	6	6	6	6	6	6	6	6	6	6	6

Apply Horspool:

T = Jimy_ran_and_hailed_the_monkey_to_stop

P = monkey   length = 6

Shift Table

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
6	6	6	6	1	6	6	6	6	6	2	6	5	3	4	6	6	6	6	6	6	6	6	6	6	6

```
Jimy_ran_and      No Match
monkey
```

shift by T(R) = 6

```
Jimy_ran_and_hailed_the_monkey
      monkey
                    No Match
```

Shift by T(D) = 6

```
Jimy_ran_and_hailed_the_monkey
            monkey    No Match
```

Shift by T(D) = 6

```
Jimy_ran_and_hailed_the_monkey
                  monkey   No Match
```

Shift by T(M) = 5

```
Jimy_ran_and_hailed_the_monkey    MATCH
                        monkey
```

check y=y, e=e, k=k, n=n,
o=o, m=M

Exercise 2: Solve and provide answer

☒ a) Given pattern t.length = 19, the value of prime q would be 19.

.

b) Input: text T = t= 2359024411526739921, pattern P = 44115, q = 19

Output: Matching numbers/pattern found at a specific index

Pseudocode:

```
//find hash value for pattern and text

m = length of P
n = length of t
d = 0                       // a designated number to help calculate the hash value
q = prime number

for i=0 in range(m):
        p = d^m-1 %q            //hash value of pattern p
        t = d^m-1 %q            //hash value of text t

//find matches

For s=0 to range(n-m+1):
        if p[1….m] = t [s+1….s+m]:
                print ("Match at ", s)

        if s < n-m:
                t  + 1 =  (d (ts - t[s + 1](d^m-1 % q)) + t[s + m + 1]) % q
```

Efficiency:

Its runtime is in O(n).

c) The number of valid matches would be: 1

The number of spurious hits of text t to P % q would be: 3

Given

t= 2359024411526739921

P = 44115

P % q = 16

Exercise 3: Solve and provide answer

Understand

Input/Output

Read and discuss the problem statement, then answer: **Describe the input and output definition in your own words.**

The input is an array of ints. The output is also an array of ints.

Examples

Write out a concrete input value for the problem, and the concrete correct output. Repeat this at least once (so there are at least two input/outputs).

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

Inputs on the left, outputs on the right.

Corner Cases

A corner case is a special case of input that is particularly unusual or challenging. Are there corner cases for this problem? If so, describe them.

This algorithm might have a hard time comparing floats for input to the dictionary as different architectures might slightly differ in their errors, but as it stands with ints, there should be no corner cases.

Use Case

Think of a specific use case for an algorithm that solves this problem. What kind of feature could it support in practical real-world software?

This could be useful for counting the number of unique items you have in an inventory.

Similar Problems

Does this problem resemble any other problems you remember? If so, which problem? Does that problem have an efficient algorithm? If so, what is the run time of that algorithm, inΘ-notation?

❌ This looks similar to this leetcode problem that runs in O(n) time.

Baseline

Design Overview

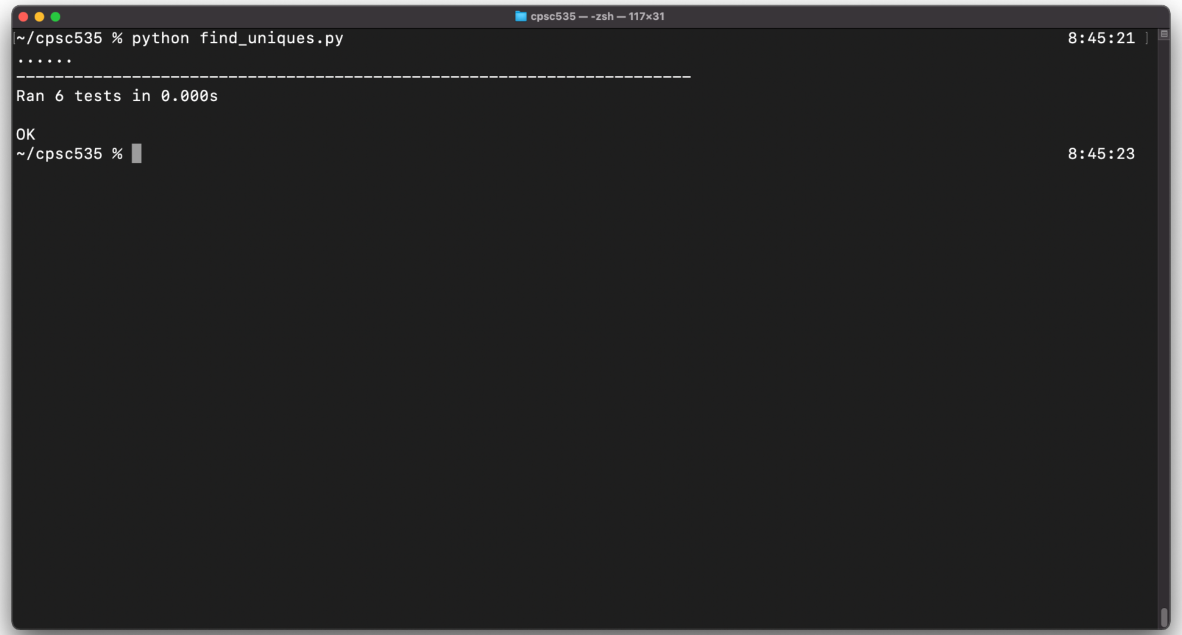
Design a naïve algorithm that solves the problem. This should be a simple algorithm that is easy to describe and understand. First, describe the basic idea of your algorithm in 1-2 sentences.

❌ The algorithm should build a counts of the array mapping, then add the unique values to a new array.

Pseudocode

Now, write clear pseudocode for your baseline algorithm.

```
from collections import Counter
import unittest
from parameterized import parameterized


def find_uniques(arr: list[int]) -> list[int]:
    """
    Design an algorithm that, given an array S of n  integers, identifies all integers
that do not repeat, i.e. that show up only once in the array S.
    Your goal: baseline O(n2) time, improved to either O(n log n) worst-case time or
O(n) expected time.

    This algorithm runs in O(n) time by constructing a counts map of values, then
re-iterating over the counts.
    """
    counts = Counter()
    for num in arr:
        counts[num] += 1

    res = []
    for key, val in counts.items():
        if val == 1:
            res.append(key)

    return res


class test_find_uniques(unittest.TestCase):
    @parameterized.expand(
        [
            [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]],
            [[1, 2, 3, 4, 4], [1, 2, 3]],
            [[1, 1, 2, 2, 3], [3]],
            [[1, 1, 1, 1, 1], []],
            [[1, 2, 1, 2, 1], []],
            [[1, 1, 2, 3, 5], [2, 3, 5]],
        ]
    )
    def test_find_unique(self, arr: list[int], expected: list[int]):
        self.assertEqual(find_uniques(arr), expected)


if __name__ == "__main__":
    unittest.main()
```



```
~/cpsc535 % python find_uniques.py                                          8:45:21
......
----------------------------------------------------------------------
Ran 6 tests in 0.000s

OK
~/cpsc535 %                                                                 8:45:23
```

Efficiency

What is the Θ-notation time complexity of your baseline algorithm? Justify your answer.

☒ The algorithm runs in O(n) time. It takes O(n) time to generate the counts map and O(n) time to iterate over the counts map again to find unique values.

Analysis

Did you meet your goal? Why or why not?

☒ Yes, the algorithm runs in O(n) time.