CSCI-B 505 Applied Algorithms (3 cr.) № 2

Dr. H. Kurban

Computer Science

School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

September 7, 2021

Contents

Introduction

This homework is meant as precursor to the course: developing working knowledge of Python and being exposed to some of the concepts of *Applied Algorithms* as we move through the semester. Because of the consequences of Covid, these first few homeworks will not be graded; rather, they are to help you prepare. The solutions are given at the end, but as graduate students you know that skipping to the end without trying to solve the problems makes the exercise unhelpful. We urge you, in the strongest terms, to treat these as though they were homework and learn from them. With respect to this, we are not giving you the code explicitly–you should type it in–this will help you immensely, since you'll have bugs and bugs are always gifts to better understand code! You are also encouraged to explore Python beyond what we present here–we simply do not have time to teach the language completely–as computer and data scientists we know that *if you know one language, you know them all*. We list here the salient topics for this particular homework:

- Introduction to what applied algorithms is
- Dictionaries
- Runtime
- I/O

We end with three problems drawn from content above. Solutions are presented at the end. As graduate students you are aware that there are often no single correct answer; rather, collections of correct answers. If you have found a solution that's different from what we present, that's great! The important point is to make sure you understand the underlying concept. If you're unsure, contact us.

Problem 1: What is Applied Algorithms

We will delve more deeply into algorithms throughout the course, but two principle questions are answered in applied algorithms: if we have a computing solution to a problem with data D, how does it scale in space and time as D increases in size. For example, suppose we have a list of positive integers $[x_1, x_2, \ldots, x_n]$ and want to determine if 100 is in the list, we might the value at first, second, ..., n^{th} positions or find it isn't present at all. In this example, we'll create a large list of positive integers and search for -1. We will use Python's time module to get an approximate time value in seconds.

1. Create a folder `Assignment_2`.
2. In this folder, write this program

```
1  import time
2
3  X, size = [], 10000000
4
5  def make_data_list(X,size):
6      for i in range(1,size+1):
7          X.append(i)
8
9
10 def query1(item, X):
11     for i in X:
12         if i == item:
13             return "found"
14     return "not found"
15
16 def query2(item, X):
17     return "found" if item in X else "not found"
18
19 def run_time(query, X, item):
20     start = time.time()
21     print(query(item, X))
22     print(query.__name__ +  " runtime: {0:.4} sec.".format(time.time↩
           () - start))
23
24 if __name__ == '__main__':
25
26     make_data_list(X,size)
27
28     for q in [query1, query2]:
29         run_time(q, X, -1)
```

and save it as `search_1.py`.

3. Open a shell and execute the script:

```
1  PS D:\505> py .\Assignment_2\search_1.py
2  not found
3  query1 runtime: 0.246 sec.
4  not found
5  query2 runtime: 0.11 sec.
6  PS D:\505>
```

On our machine we abbreviated python to py. You do not need to do this. We looked for our value in two ways, using our own loop and then letting Python use its approach. The time module allows you to get the time (line 20), then find the elapsed time by subtracting that time from the current time (line 22). As you can observe, there is more than a doubling of time between the approaches. If you run this several times, you'll get answers that differ slightly due to particular elements of your machine. You should go to
`https://docs.python.org/3/library/__main__.html`
to read about lines 25-30. This method should be your standard approach when writing Python. Here is an interesting post on loops in Python
`https://www.blog.duomly.com/loops-in-python-comparison-and-performance/`.
Being able to connect with the Python community is the principle reason Python is currently the most popular language.

Problem 2: Dictionaries and Measuring Runtime

Dictionaries are containers, but unlike lists are *hashed*. They are pairs of key, values that allow constant time access. Here is a common use: counting. Suppose we have data on a ballot (0 = no, 1 = yes) and have this as data as ten votes. We want to write a program that finds if the ballot passed, failed, or tied.

1. Inside `Assignment_2`, write this program:

```
import random as rn

data = [rn.randint(0,1) for _ in range(10)]
vote = {}
vote[0], vote[1] = 0,0

if __name__ == '__main__':

    print(f"Data: {data}")
    for d in data:
        vote[d] += 1

    print(f"Vote Dictionary: {vote}")
    if vote[0] > vote[1]:
        print("Measure failed...")
    elif vote[0] < vote[1]:
        print("Measure passed...")
    else:
        print("Tie vote...")
```

   and name it `dic_1.py`. Here are some runs of the program:

```
PS D:\505> py .\Assignment_2\dic_1.py
Data: [0, 0, 0, 1, 1, 1, 1, 1, 0, 0]
Vote Dictionary: {0: 5, 1: 5}
Tie vote...
PS D:\505> py .\Assignment_2\dic_1.py
Data: [1, 1, 0, 1, 1, 1, 1, 0, 0, 1]
Vote Dictionary: {0: 3, 1: 7}
Measure passed...
PS D:\505> py .\Assignment_2\dic_1.py
Data: [0, 0, 1, 0, 0, 0, 0, 0, 0, 1]
Vote Dictionary: {0: 8, 1: 2}
Measure failed...
PS D:\505>
```

Line 3 is the Pythonic way to build lists called *list comprehension*. Please read about this. List comprehension is typically very efficient. Line 4 creates an empty dictionary. Line 5 creates two entries into the dictionary 0:0 and 1:0. Remember, these are key, value pairs. Keys must be unique an *immutable*. Please review mutability in Python. Line 9 is often an easier way to display variable vaules. Lines 10-11 increment the count of either 0 or 1. Lines 14-19 check the dictionary for the three outcomes.

Problem 3: I/O

Python uses the string type to read and write to storage. This requires that you have both the location and the name. One way is to directly set your path. We'll do that here.

1. Inside 505 directory, create a folder `Sunshine`. Use your command line to see that the folder exists:

```
PS D:\505> tree
Folder PATH listing for volume DATA
Volume serial number is 0A26-4A0D
D:.
|------Assignment_1
|------Assignment_2
|------Sunshine
PS D:\505>
```

   Write the following program in the Assignment_2 directory and name it `input_output.py`.

```
import os

if __name__ == '__main__':

    path = os.getcwd()
    print("current directory", path)
    new_location = input("Enter new path: ")
    os.chdir(new_location)
    path = os.getcwd()
    print("current directory", path)
```

   Here is my run:

```
PS D:\505> py .\Assignment_2\input_output_1.py
current directory D:\505
Enter new path: d:\505\Sunshine
current directory d:\505\Sunshine
PS D:\505>
```

   Create an ASCII file called `data.csv` in the Assignment_2 directory like this (exactly):

```
1, 10
3, 21
4, 31
5, 45
```

2. The following should be put into the Assignment_2 directory named `input_output.py`:

```
import csv

if __name__ == '__main__':

    data = []
    location_name = input("file: ")
    with open(location_name, 'r') as f:
        fcsv = csv.reader(f)
        for lines in fcsv:
            data.append(list(map(int, lines)))

    print(data)
```

Because Python reads the data as strings, we need to cast them as ints. Please read about map and int. Here is a run:

```
file: d:\505\Sunshine\data.csv
[[1, 10], [3, 21], [4, 31], [5, 45]]
```

Let's denote the data by (x, y). We need to create a file $(x^2, y + 1)$ so the new file will be:

```
1, 11
9, 22
16, 32
25, 46
```

Here is the extended program:

```
import csv

if __name__ == '__main__':

    data = []
    location_name = input("file: ")
    with open(location_name, 'r') as f:
        fcsv = csv.reader(f)
        for lines in fcsv:
            data.append(list(map(int, lines)))

    print(data)

    new_location_name = input("file: ")
    with open(new_location_name, 'w') as g:
```

```
16          for x,y in data:
17              g.write(f"{x**2}, {y+1}\n")
```

We created a list of pairs of numbers, then wrote the new values back to another file. Here is a run:

```
1  file: d:\505\Sunshine\data.csv
2  [[1, 10], [3, 21], [4, 31], [5, 45]]
3  file: d:\505\Sunshine\change.csv
```

Here is the file `change.csv`:

```
1, 11
9, 22
16, 32
25, 46
```

This is actually one long string:

`'1, 11\n9,22\n16, 32\n25, 46\n'.`

Problem 4: Practice

1. Often times, our code gives explicit directives to the compiler to improve speed. For this problem, modify `search_1` to compare two ways to find this sum:

$$sum = \sum_{i=1}^{10000000} i^2 \tag{1}$$

The right side of Eq. 1 can be written in two ways:

```
1  i * i
2  i ** 2
```

Compare the runtimes and discuss the difference.

2. You've seen the recursive formula for the Fibonacci sequence:

$$f_0 = 0 \tag{2}$$

$$f_1 = 1 \tag{3}$$

$$f_n = f_{n-1} + f_{n-2} \tag{4}$$

This can easily be converted to a bounded loop ($\leftarrow$ means assigned to)

$$f_0 \leftarrow 0 \tag{5}$$
$$f_1 \leftarrow 1 \tag{6}$$
$$f_0, f_1 \leftarrow f_1, f_0 + f_1 \tag{7}$$

This is particular to Python–the assignments are done in parallel. There is also a clever trick using linear algebra:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix} \tag{8}$$

Write a Python script for the second two functions. Add a third function that is a dictionary which returns the n^{th} Fibonacci if it's in it, otherwise use the linear function to add to the dictionary. Time these functions for $f(200)$. You are encouraged to ready about the numpy matrix multiplication module, but you're free to write your own multiplication.

3. Read in the data from `data.csv` and visualize it using the plot command in matplotlib. You should use the code from last homework as a starting point.

4. Write a Python script that reads the data from the file you created, but now plots it (read about scatter) and saves it to a directory and name of your choice using $(x^2, y+1)$ for the new values. Here is a partial skeleton of the code:

```
1
2     #code to read in data
3
4     fig = plt.figure()
5     ax = fig.add_subplot(1, 1, 1)
6
7     plt.title("Data Plot")
8     X,Y = [],[]
9     #create a list of first value of pairs X
10    #create a list of second value of pairs Y
11
12    plt.scatter(X,Y)
13
14    new_location_name = input("file: ")
15    plt.savefig(new_location_name, dpi=300, bbox_inches='tight')
16    plt.show()
```