Computer Science AP/X — CSCI-140/242
Battle of the Bands — Lab 3

8/9/2022

1 In-Lab Activity

The in-lab activity can be found at `https://www.cs.rit.edu/~csapx/Labs/03-Quickselect/inlab.pdf`. The goal is to implement the *Quicksort* version of this lab. When this part is finished, upload your `bands.py` program to the MyCourses Assignment dropbox.

2 Quickselect

There is an algorithm called *Quickselect* (often called *k-select*) which is better suited for the problem of finding the most mediocre band. The algorithm is used to find the `k`th smallest element in an unordered list. In our problem, `k` is the index of the median element, which we defined as:

```
k = len(lst) // 2
```

The algorithm is similar to Quicksort, but it does not require a complete sort of the data in order to find the k-th element.

2.1 Quickselect example

Consider a list of 9 numbers and finding the index of the 2nd smallest element (i.e. index 1 for a sorted Python list).

```
data = [65, 28, 59, 33, 21, 56, 22, 95, 56]
k = 1
```

First, choose a pivot from the list. Note that it is a good practice to pick a random pivot but picking the first index is sufficient for this lab. We will determine the "sorted" location of the pivot element and compare it to `k`:

```
pivot = 56
```

The elements smaller than the pivot are in the left subarray, `smaller`:

```
smaller = [28, 33, 21, 22]
```

A count of the elements that are equal to the pivot are in `count`:

```
count = 2
```

The larger elements are in the right subarray, `larger`:

```
larger = [65, 59, 95]
```

We now know the pivot's exact location is `m` if the list was sorted:

```
m = len(smaller) = 4
```

There are three possible outcomes after the partition:

1. If `k` is between `m` (inclusive) and `m+count` (exclusive), the `k`-th element has been found. It's the `pivot`.
2. If `m` is greater than `k`, the `k`-th smallest element is in `smaller`. We have eliminated `count + len(larger)` elements, but have not found the `k`-th value. Perform a similar partition on `smaller` using the same value for `k`.
3. Otherwise the `k`-th smallest element is in `larger`. We eliminated `len(smaller) + count` elements. Perform a similar partition on `larger` using `k - m - count` for `k`.

In this example we have the second outcome, `k = 1` is less than `m = 4`, so we perform another partition using the smaller list `smaller = [28, 33, 21, 22]` with index `k = 1`. The element in the second position will eventually be found, `22`, at index 1, starting at 0 in the list.

2.2 Quickselect complexity

Quickselect can suffer from the same problem as Quicksort, i.e. there is a chance that the pivot does not split the list into similar sized subsets and the problem is not reduced by $\sim \frac{1}{2}$ at each iteration. Therefore, the worst case performance is $\mathcal{O}(n^2)$. In practice, working with random data and pivots yields an average/best-case performance of $\mathcal{O}(n)$. The proof of this is outside the scope of our course but it is relatively easy to derive with an understanding of geometric series. This is why Quickselect is much faster than Quicksort's $\mathcal{O}(n \log n)$ performance.

Quickselect is usually implemented like Quicksort, using an in-place algorithm, which means when the `k`-th element is found, the list of data will be partially sorted. For this lab, it is okay to create new lists during each partitioning. This method will consume more memory to store the temporary lists during the search, and run a little slower than in-place, but it will still maintain linear complexity for performance.

3 Implementation

Continue working on the same program from your in-lab, `bands.py`. It will be modified to demonstrate the performance differences between using Quickselect and Quicksort to solve the problem of finding the most mediocre band.

3.1 Command Line

The program is run on the command line as:

```
$ python3 bands.py [slow|fast] input-file
```

If the number of command line arguments is incorrect, display a usage message and exit:

```
Usage: python3 bands.py [slow|fast] input-file
```

If `slow` is specified, use Quicksort to find the band at the median amount of votes otherwise assume `fast` is specified and use Quickselect.

Assume the input file exists, and it contains at least one band. You should test your lab with the test files you downloaded in the in-lab.

3.2 Program Output

When the program runs, it will produce 4 lines of output:

- The search type that was specified on the command line:

  ```
  Search type: [slow|fast]
  ```

- The total number of bands in the input file:

  ```
  Number of band: #
  ```

- The time it took to perform the search, in seconds:

  ```
  Elapsed time: # seconds
  ```

- The band at the median:

  ```
  Most Mediocre Band (name='XXX', votes=#)
  ```

3.3 Timing

The calculation of the elapsed timing should only include the time it takes to perform the search. It should not include the time to read in the file, or to display the final output.

Use the `time` module in python to determine the elapsed time, in seconds:

```
>>> import time
>>> start = time.perf_counter()
>>> time.perf_counter() - start
0.0016670000000000018
```

3.4 Sample Runs

Here are sample runs using the million band data set, `test-1M.txt`. Results should be similar depending on the computer's speed.

3.4.1 Quicksort

```
$ python3 bands.py slow test-1M.txt
Search type: slow
Number of bands: 1000000
Elapsed time: 9.484235463000005 seconds
Most Mediocre Band: Band(name='The Dissevering Upbeat
   Batrachomyomachias', votes=50000804)
```

3.4.2 Quickselect

```
$ python3 bands.py fast test-1M.txt
Search type: fast
Number of bands: 1000000
Elapsed time: 0.6002664689985977 seconds
Most Mediocre Band: Band(name='The Dissevering Upbeat
   Batrachomyomachias', votes=50000804)
```

4 Grading

This assignment will be graded using the following rubric:

- (15%) Problem-solving
- (10%) In-lab Activity
- (10%) Code Design
- (55%) Functionality:
  - (20%) Quicksort solution
  - (35%) Quickselect solution
- (10%) Code Style and Documentation: Proper commenting of modules, classes and methods (e.g. using docstring's).

5 Submission

Go to your project's `src` folder and zip up the `bands.py` file. Rename the zip file `lab3.zip` and upload it to MyCourses Assignments by the due date.