CS325: Close to zero project

Stephen Redfield

Due: Week 3, Sunday

Your report must be a typeset, *PDF document. Each team member's name must be listed as well as any resources used to finish the project. For questions regarding this project, please contact your TA.*

For this project, you will design, implement and analyze (both experimentally and mathematically) *two* algorithms for the close-to-zero problem:

> Given array of small integers $a[0, 1, \ldots, n-1]$, compute
>
> $$\min_{0 \le i \le j < n} \left| \sum_{k=i}^{j} a[k] \right|.$$
>
> That is, given any array, find a subarray whose sum is closest to zero.
> For example, CloseToZero$([31, -41, 59, 26, \mathbf{-53}, \mathbf{58}, \mathbf{-6}, 97, -93, -23, 84]) = 1$

You may use any language you choose to implement your algorithms, but both algorithms should be implemented in the same language. Be sure that your algorithm is *correct* for any input.

Instructions

Your two algorithms are to be based on these ideas:

Algorithm 1: Enumeration Loop over each pair of indices i, j and compute the sum, $\sum_{k=i}^{j} a[k]$. Keep the best sum you have found so far.

Algorithm 2: Better Enumeration Notice that in the previous algorithm, the same sum is computed many times. In particular, notice that $\sum_{k=i}^{j} a[k]$ can be computed from $\sum_{k=i}^{j-1} a[k]$ in $O(1)$ time, rather than starting from scratch. Write a new version of the first algorithm that takes advantage of this observation.

Testing for correctness Above all else, your algorithms should be correct. A file containing test sets (test_case.txt) can be found on Blackboard: The file has one test case per line (10 cases each with 100 entries). A line corresponding to the example above would be:

```
[31, -41, 59, 26, -53, 58, -6, 97, -93, -23, 84], [-53,58,-6],1
```

with the input array followed by the sum of the closest-to-zero subarray and the corresponding start and end indices (with indices start at 0). You may use this test file to check that your code is correct. You should also test your code on small hand-generated instances. **This is the format which your program should output its results.**

Experimental analysis For the experimental analysis you will plot running times as a function of input size. Every programming language should provide access to a clock (not necessarily in seconds). Run each of your algorithms on input arrays of size $100, 200, 300, \ldots, 900$ and $1000, 2000, 3000, \ldots, 9000$ (that is, you should have 18 data points for each algorithm). The first enumerative algorithm may be frustratingly slow, so you may compute running times for sizes $100, 200, 300, \ldots, 900$.

To do this, generate random instances using a random number generator as provided by your programming language. Remember to include both positive and negative numbers! For each data point, you should take the average of a small number (say, 10) runs to smooth out any noise. For example, for the first data point, you will do something like:

```
for i = 1:10
    A = random array with 100 entries
    start clock
    CloseToZero(A)
    pause clock
return elapsed time
```

Note that you should not include the time to generate the instance.

Plot the running times as a function of input size for each algorithm in a single plot. Label your plot (axes, title, etc.). Include an additional plot of the running times on a log-log axis. See here for an explanation: `http://en.wikipedia.org/wiki/Log-log_graph` Note that if the slope of a line in a log-log plot is m, then the line is of the form $O(x^m)$ on a linear plot. You may also find these videos helpful: `http://www.khanacademy.org/math/algebra/logarithms/v/logarithmic-scale` and `https://www.khanacademy.org/math/probability/regression/regression-correlation/v/fitting-a-line-to-data`

For an example of an experimental analysis in java, see `http://algs4.cs.princeton.edu/14analysis/`

Project Report

For each of the above algorithms, your report **must** include:

Run-time analysis Give pseudocode for each algorithm and an analysis of the asymptotic running-times of the algorithms.

Testing To illustrate that your code is correct, determine the solution *and* value for each instance in the file: (CtZ_Problems.txt) Each line of this file is a different input array.

Experimental analysis Perform an experimental analysis and include plots as described above. *Note: Keep the data used for these plots; you will use them in the next project.*

Extrapolation and interpretation Use the data from the experimental analysis to answer the following questions:

1. For each algorithm, what is the size of the biggest instance that you could solve with your algorithm in one hour?
2. Determine the slope of the lines in your log-log plot and from these slopes infer the experimental running time for each algorithm. Discuss any discrepancies between the experimental and asymptotic running times.

What to Submit

Your elected submitter must upload the following to Blackboard and T.E.A.C.H.

Project Report: Submit the project report as specified above.

Code: Submit your fully-functioning program which must run on our the engr server FLIP for verification, and a README with directions describing exactly how to use it, combined in a ZIP file.

Results: Submit your results for the CtZ_Problems.txt file, in the form of a .txt file CtZ_Results.txt.