Dept. of Computer Science, University of California, Davis
ECS122b Instructor: Rob Gysel 5/25/2016

Program #2: Due dates below.

1 Overview

For this program, you will compare algorithms from one of the projects listed below. You may work in groups of up to 5 people for this project. This project has three milestones, with the following due dates:

Friday 5/27/16 11:59pm Proposal. Group size, project, and language.

Monday 5/30/16 11:59pm Division of Labor. Unnecessary if you work alone. See section 4 for details.

Monday 6/6/16 11:59pm Final Submission. Algorithm source(s), script(s), report (pdf).

You may write scripts in any language you wish.

2 Grading

This program is worth 100 points. 50 points come from your Gradescope submission and 50 points come from interactive grading. Your Gradescope submission is broken down as follows: Proposal & Division of Labor 5 points, Source Code 20 points, Report 25 points. Groups will be graded together during interactive grading.

3 Proposal (Due Friday 5/27/16 11:59pm)

Write a one-page proposal using LaTeX and submit it to Gradescope. In your proposal, be sure to include who is in your group, what project you will pick and what language you will use. Each team of two or more people need only submit one proposal, which should be a group submission on Gradescope.
Only use CSIF machines for your experiments as a last resort: the results will be unreliable. Give yourself *at least 4 days* to **run your programs**, gather data, debug, and write your report. Your report will be written in LaTeX. See the template program2_template.tex in the Google Drive and `https://www.latex-tutorial.com/tutorials/quick-start/` if you are new to LaTeX and want to save time. See section 6 for potential projects.

4 Division of Labor (Due Monday 5/30/16 11:59pm)

Write a one-page paper using LaTeX detailing the division of labor amongst your group. If you are working alone, you do not need to turn in this document. For each member of the group, list the following:

1. Programming responsibility. Note if code is original source or not. Estimate (roughly) time required.
2. Writing responsibility. Each member must help write at least one section of the report and proof-read at least one separate section of the report. A section may have multiple writers.

Submit your pdf to Gradescope as a group submission. Be prepared to answer questions on your portion of the project during interactive grading.

5 Final Submission (Due Monday 6/6/16 11:59pm)

See Section 6 for details on your report for a particular project. Your report must be written using LaTeX. Submit your pdf to Gradescope, and submit as a group submission if working on a team.

6 Projects

Pick one of the following projects. To submit a proposal for a project of your own choosing instead, e-mail Rob at rsgysel@ucdavis.edu by Thursday 5/26/16 11:59pm for feedback *before* writing your proposal. *Be sure to read this entire document before choosing a project.*

For projects that require hash functions or families of hash functions, see the next section.

QuickSelect vs DeterministicSelect

Overview Implement QuickSelect and DeterministicSelect. Implement finding the k^{th} largest element by sorting the array and taking the k^{th} element.

Sanity Checking Write a sanity checking script that randomly generates 100 test cases, runs them, and lets the user know if all implementations returned the same result or not. If not, the script should quit when a test case does not match in all three implementations, print the test case, and the result from all three implementations. Each test case should contain 1000 integers from the range $[1, 10^6]$.

Gather Data Run and time your code. Your study should investigate data sizes of 10^2, 10^3, 10^4, 10^5 and 10^6, with 200 samples each.

Report Write a report of at least three pages in length using LaTeX. You should have the following sections:

Introduction: Describe the problem, algorithms, and pseudo-code for your algorithms.

Empirical Studies: Describe what data was gathered and how it was gathered.

Conclusions: Display at least one plot that is derived from your data.

Citations: Cite any source you used for information. Sources may be textbooks or websites. **You must write your own source code for this project, and may not obtain it from any other source.**

Enumeration

Overview Implement subset enumeration and n-tuple enumeration as described in class. Write a program that constructs instances of the generalized Knapsack problem and solves them by brute-force using subset and n-tuple enumeration. Your brute-force algorithms should report the solution found, including items, total weight, and total value. For both sanity checking and data gathering (see below), your item values should be integers chosen randomly between 1 and 100 and your bag weight limit should be half the total value of *all* of the items in a given problem instance.

Sanity Checking Write a sanity checking script that randomly generates 100 test cases, runs them, and lets the user know if both implementations returned the same result or not. If not, the script should quit when a test case does not match in both implementations, print the test case, and the result from both implementations. Each test case should contain 6 items with 2 copies each.

Gather Data Run and time your code. Your study should investigate all combinations of the following parameters, with 50 samples each:

Number of items: 3, 4, and 5.

Max number of copies: 1 copy, 1 or 2 copies, $1-3$ copies, $1-4$ copies. For cases with a range of max copies, for each item, pick the copy number at random uniformly from the range given (e.g. pick a number in $1-3$ where each number is chosen with probability $\frac{1}{3}$).

Report Write a report of at least three pages in length using LaTeX. You should have the following sections:

Introduction: Describe the generalized Knapsack problem and algorithms you implemented (a brief overview and pseudo-code).

Empirical Studies: Describe what data was gathered and how it was gathered.

Conclusions: Display at least one plot that is derived from your data.

Citations: Cite any source you used for information. Sources may be textbooks or websites. **You must write your own source code for this project, and may not obtain it from any other source.**

Cuckoo Hashing

Overview Implement cuckoo hashing as described in class, except for re-hashing. The table will re-hash if a cycle occurs. This happens only when the a given key is put into the same spot (table & slot) twice. If this occurs, first check the stash for the key. When this occurs, you should stop inserting into the hash table, and insert it into small sized array that we will call a *stash*. Once the stash is full, a re-hash should occur. See the next section for hash function options. You will need two distinct hash functions for this project.

Gather Data Empirically study how many hashes occur until a re-hash must occur (hence you only need two hash functions, not a family of hash functions). Your study should investigate all combinations of the following parameters, with 50 samples each:

Hash table size: 10^2, 10^3, 10^4, 10^5 and 10^6.

Load factor: $\frac{1}{2}$, $\frac{3}{8}$ $\frac{1}{4}$, and $\frac{1}{8}$.

Stash size: 1, 3, 5, 10, 15.

Report Write a report of at least three pages in length using LaTeX. You should have the following sections:

Introduction: Describe the study, algorithm, and pseudo-code for your algorithm. Note which hash functions were used, but you do not need to describe them.

Empirical Studies: Describe what data was gathered and how it was gathered.

Conclusions: Display at least one plot that is derived from your data.

Citations: Cite any source you used for information. Sources may be textbooks or websites. **You may obtain source code for hash functions from on-line sources, but they must be cited.**

Probabilistic Data Structures

You may also write a proposal to study one of the probabilistic data structures discussed in class (Bloom Filters, LogLog, etc.). In your proposal be sure to include the following.

1. The probabilistic data structure to be studied.
2. An example application. You may use a source for this, but write the application in your own words, and be sure to cite your source.
3. How the data structure will be evaluated.
4. How data will be generated.

6.1 Hash Functions & Families

You can either implement one of the following hash functions or use the source code for an implementation posted on-line.

Md5 `http://people.csail.mit.edu/rivest/Md5.c`

SHA-256 `https://tls.mbed.org/sha-256-source-code`

MurmurHash `https://github.com/aappleby/smhasher`

SpookyHash `http://burtleburtle.net/bob/hash/spooky.html`

Skein Family `https://www.schneier.com/academic/skein`

CityHash `https://github.com/google/cityhash`

FarmHash `https://github.com/google/farmhash`