Project 2

Rainbow Table

Admin

- Deadline: 12noon, (Thursday) 12 Nov 2014.
- Submit
  - A simple (say one or two pages) report in pdf format + demo.
  - Source code.

  Likely that I would not read the source code. If you want to highlight your algorithm, describe it in the report, not the source code. Since there would be a demo session, you can also explain your program during the demo. See Forum on the arrangement of demo session.

- Upload to IVLE. Put everything into a single file. The file name should be *MatricNumber_Name.extension* for e.g,

  A000007A_AliceChang.zip

  The report (pdf) file name should be: *MatricNumber_Name*.pdf for e.g.

  A000007A_AliceChang.pdf

  If you prefer to hand in some handwritten notes, or hand-drawn figures, pass the report to me.

- Programming language: no restriction. (Java, C, C++. Not clear which has advantage in performance. For C++, templates of the program will be given).

- Machines: No restriction

Goal

- For simplicity, a word in our dictionary is a 24-bit string. So there are a total of 2^{24} possible words.
- Every word is hashed using SHA1.
- You are tasked to build a datastructure `RAINBOW`, and give a program `invert`, that on input of 1000 digests, with the help of the pre-processed RAINBOW, finds the corresponding 1000 words. (see SAMPLE_INPUT.data SAMPLE_OUTPUT.data for format).

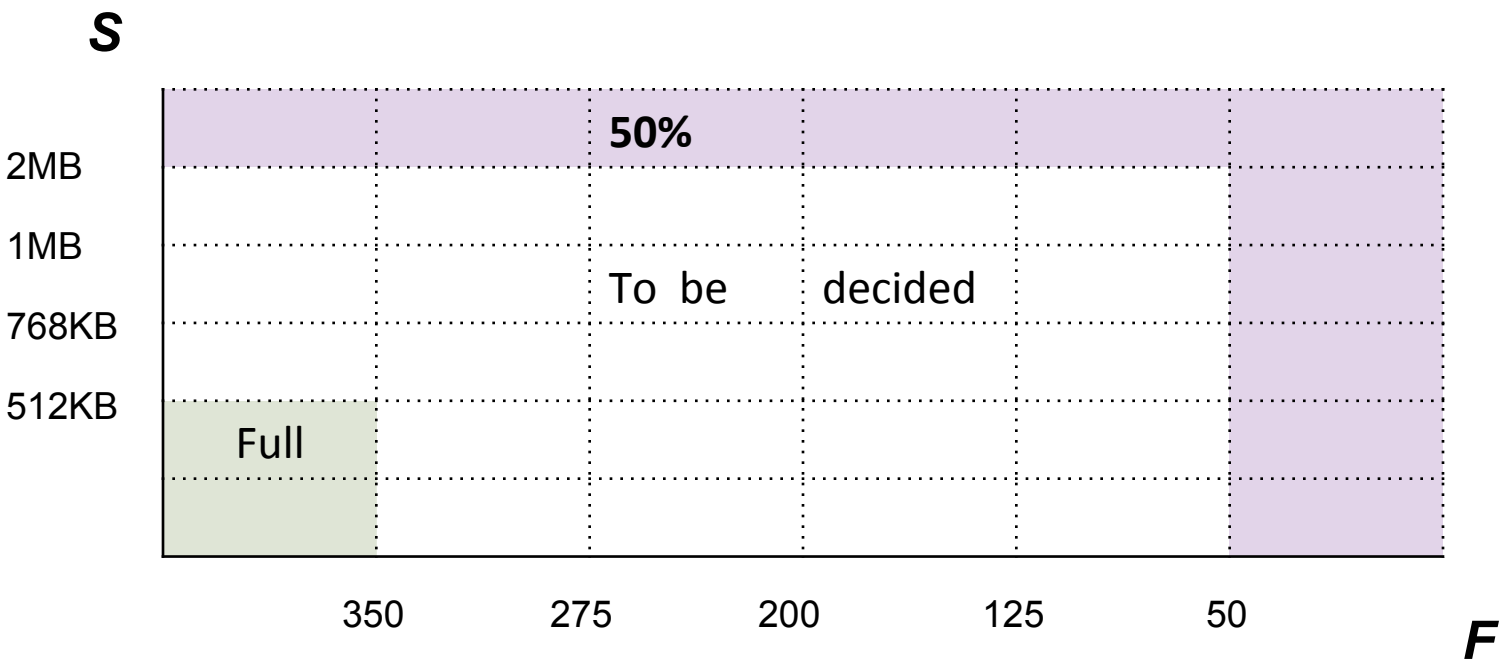
- Measure/derive the followings:
  - ***t***, the time taken by `invert`.
  - ***C***, the percentage of words correctly found by `invert`.
  - ***S***, the size of `RAINBOW` in bytes, as stored in the file directory
  - ***T***, the time taken by your machine to compute 2^{23} SHA1 operations. This is the expected time taken to invert the digest directly.
  - ***F***, speedup factor of your program, $F = 1000\ T / t$.

- Requirements:

  $$C > 45\%$$

  Your grade will depend on the speedup factor ***F*** and ***S***

Grading



Deriving the performance.

- You can "derive" the value of ***F***,***C***,***S***, instead of directly measure them.

- Method 1. Suppose the data structure is a table, where each row consist of
  (a) a 24-bit string and
  (b) a 30-bit string.

So each row can be packed into a 54-bits string. If your table consists of 1000 rows, then it is possible to packed everything into 6,750 bytes. However, although possible, it is very tedious in programming to pack them into 6,750 bytes. In your report, you can "derive" and state that the size is 6750 bytes, instead of giving the actual size of the file generated by your program.

- Method 2. Note that accuracy ***C*** can be improve simply by having multiple data-structure, but with the expenses of increasing the speedup factor ***F,*** and size ***S.*** With ***x*** duplicated data-structures (but with different "reduce" functions), the chance of misses reduces to $(1-\boldsymbol{C})^x$, and thus accuracy improves to $1-(1-\boldsymbol{C})^x$ but the size increases to $x\boldsymbol{S}$ and the time ***t*** increases to $\boldsymbol{Ct}+ 2\boldsymbol{C}(1-\boldsymbol{C})\boldsymbol{t} + 3\boldsymbol{C}(1-\boldsymbol{C})^2\boldsymbol{t} + \ldots + + (x-1)\boldsymbol{C}(1-\boldsymbol{C})^{x-2}\boldsymbol{t}+ (1-\boldsymbol{C})^{x-1}\boldsymbol{t}$. (Note that the last term doesn't contain the factor C. To understand this formula, it would be helpful to write down the special of x=2, and x=3..) Likewise, the speedup factor reduce by a factor of $\boldsymbol{C}+ 2\boldsymbol{C}(1-\boldsymbol{C}) + 3\boldsymbol{C}(1-\boldsymbol{C})^2 + \ldots + + (x-1)\boldsymbol{C}(1-\boldsymbol{C})^{x-2}+ (1-\boldsymbol{C})^{x-1}$

Eg.

S=100KB, ***C***=12%, ***F***=500 → duplicate 5 times → ***S***=500KB, ***C***=47.2%, ***F***=127.05

Grading & Report

- In the report's first page, clearly state

  1. The measured S,C ,T, t and the derived S,C,F.
  2. The configuration of machine and programming environment for e.g.

     A desktop. CPU: Core2 Quad CPU@3 GHz. Memory: 4GB Java.

- Total mark is 20. This form 20% of overall grade. (Recall that 60% CA, 40% Exam).

- Grading is based on the factor F.

Input/output format of `invert`

- Your program `invert` must able to read in an input file that contains 1000 digests (see IVLE for sample). The outputs of `invert` lists the corresponding 24-bit words in hex format. If a digest cannot be found, then it is replaced with 0. The last line of the output is:

  `The total number of words found is: xxx`

  where xxx is the number of words found (in decimal format).

For example

```
1F3DF
FED3A3
0
0
123D00
…
…
…
113A4B
The total number of words found is: 476
```

What's in the IVLE workbin

- A sample input.
- A sample output
- A C SHA1 library
- A C++ SHA1 library
- The C++ program that generates the input
- A C++ template that build the table.
- A C++ template that invert the digest.
- This file.

Remarks/Guides

- In my machine (OSX, 2.9GH Intel Core i7, 8GB 1600 GHz DDR3, using GNU g++) $T \approx 3.1$ seconds. My program took a few minutes to build the table.

- The main parameter to adjust is the length of the chain. In my version that achieves F=200, the average length of the chains is approximately 180.

- It is possible to complete this project using rainbow table. The original time-memory trade-off may able to achieve reasonable speedup.

- It is important to employ an efficient data structure that supports fast insertion and searching. For example, the standard library `<unordered_map>` in C++. A simple method of using a large table with 2^{32} entries might also work.

- For Java, use the standard library if possible. In particular, on SHA1.

Non-acceptable solutions…

- Note that the choice of 24-bits words is for simplicity.
- This solution is not accepted:
  - No rainbow table is being built.
  - On input of 1000 digests X, (1) built the full table (there is only 2^{24} words) within seconds. (2) For each digest in X, search for the word in the table.
- Although the above seems to work, that is because there are 1000 digests, and the speedup factor could reach 500. In practical scenario, we only need to invert one digest. If there is only one digest, the speedup factor will be less than 1.