Huffman Image Compression and Hash Map
S19 CSCI332 Design and Analysis of Algorithms

Assigned: 2019-04-12
Reference: Sorting and Priority Queueing — Due Date: 2019-04-30 (midnight)

Summary:

Abstract: Compress an ASCII image and store (and then retrieve) from a Hash Map for rendering.

Objectives:

1. Huffman Tree and Code Words
2. Compression by Variable Bit Length
3. Priority Queues and Hash Maps
4. C++ Standard Template Library
5. Iterators and Iterator Algorithms
6. Algorithm Analysis

Grading: 45 pts - A $\geq$ 41.85; A- $\geq$ 40.50; B+ $\geq$ 39.15; B $\geq$ 37.35; B- $\geq$ 36; C+ $\geq$ 34.65; C $\geq$ 32.85; C- $\geq$ 31.75; D+ $\geq$ 30.15; D $\geq$ 28.35; D- $\geq$ 27

Outcomes: R1 (CAC-c,i,j,k; EAC-e,k,1); R2 (CAC-a,b,j; EAC-a,e,1); R6 (CAC-a,c,j; EAC-a,e,1) (see syllabus for description of course outcomes)

Description:

You have been given a sensitive image (ASCII art for our purposes) that must be stored with the minimum space footprint and in a coded manner such that, with out the code word translation table, elements from the storage container are meaningless. You have been provided a tool in the `utilities` folder to help you create the code words to be used for the image file(s) found in the `images` folder. This utility will create for you a lot of useful information to help you store the encoded image information.

Those seeking to secure the image information have decided you should use a *Hash Map* container to store your encoded image as it should provide for quick retrieval of the encoded image data when needed such that the image can be decoded and put back together. Take care ! Remember, the key for the hash map must be unique across all hash map entries, and you must use the information provided by the utility to craft of a hash map that is efficient as the speed of retrieval of the encoded image is vitally important and must be close to $O(1)$ as possible, while minimizing the storage footprint of the encoded image.

Input File: The input file will consist of the image(s) in the `images` folder. You will need to use the utility to generate the coded information from these images. Here is a sample of the image to be encoded:

```
  ,ad8888ba,    ad88888ba     ,ad8888ba,   88   ad888888b,   ad888888b,   ad888888b,
 d8"'    `"8b  d8"     "8b   d8"'    `"8b  88  d8"     "88  d8"     "88  d8"     "88
d8'           Y8,          d8'            88          a8P          a8P          a8P
88            `Y8aaaaa,    88             88        ,d8P"        aad8"        ,d8P"
88              `""""""8b,  88             88      a8P"           ""Y8,      a8P"
Y8,                   `8b  Y8,            88   a8P'                "8b   a8P'
 Y8a.    .a8P  Y8a     a8P   Y8a.    .a8P  88  d8"          Y8,     a88  d8"
  `"Y8888Y"'    "Y88888P"     `"Y8888Y"'   88  88888888888   "Y888888P'  88888888888
```

Of course, this is not the actual secret image, you need to first prove your algorithm's capability before those are disclosed to you!

In order to store this image in an encoded way in the hash map, you must create a *key* selection algorithm that is uniquely tied to each character (pixel) in the image. One way to think about this is the character's location (coordinate), but be careful ! - the keys must be unique across all entries in the hash map; so simply adding the coordinate values together is likely not a good key selection algorithm, for example.

Once you have your key selection algorithm, you must then create a hash map function, that maps the key from the image into the hash map index and then stores the code word for the image contents at the key provided to the hash function. You must do this to minimize the storage footprint for storing the encoded image.

Your Country is counting on you to get this working so secret encoded images can be sent in the plain sight without the fear of interception and successful decoding. This file will self-destruct in 29-days :-).

Obtaining Project Files:

1. Log into your account on the `gitlab.cs.mtech.edu` department server and fork this project `HuffmanImage` into your own account.
2. Go to your cloned project in your own account on the GitLab department server and select [settings] and then [members] for the project and add your instructor (and any teaching assistants for the course) as a `Developer` member.
3. (optionally - only need to perform this step once) If you are going to use the `ssh` protocol to obtain your project files from the GitLab department server, you need to make sure the `ssh` key from the machine on which you will be working with the project are copied to the list of valid keys in your account.
4. copy either the `ssh` or `http` url paths to your clipboard
5. Log into the `lumen.mtech.edu` department linux server with your department credentials. If this is the first time you have logged into the server, your username will be the first part of your campus email account and your default password will be your student id; make sure to change your password the first time you login using the `passwd` linux command.
6. Create a projects folder for the course using the command `mkdir -p ~/CSCI332/Projects`, and then change into this directory using the command `cd ~/CSCI332/Projects`.
7. Clone the project to your course project folder using the command `git clone <url>`, where `<url>` is the project url you copies to your clipboard. This will create a new directory for the project.
8. Your should use the command `cd` to change into the new project folder you just cloned.
9. Now proceed to the project activities in the next section.

Project Activities:

The following tasks need to be performed in order to complete the project:

1. Explore the project files, especially those in the `utilities` and `images` folders. Build the utilities and execute this on input, including images from the `images` folder. For example, if you execute,

   ```
   ./GenHuffmanCode
   a line of text
   ^D
   ```

   where `^D` is the control-d keyboard sequence to end close the terminal input to the program, you should get the following as output:

Symbol:	a	e	f	i	l	n	o	t	x
Count:	3	1	2	1	1	1	1	1	2	1
Frequency:	0.21	0.071	0.14	0.071	0.071	0.071	0.071	0.071	0.14	0.071
Code Word:	01	1010	110	1110	1111	1011	1000	000	001	1001

Notice the binary code words are prefix-free, that is if you were scanning a collection of the code words one bit at a time, the minute you have a bit pattern that matches a code word, you can translate back to the corresponding symbol.

2. Using the C++ STL `unordered_map` container, create your hash map to store the encoded symbols from the image(s) in the `images` folder.
3. You will need to develop two algorithms for this purpose:
   - an algorithm for taking an image element location, and representing this as a single key value that is unique across as entries in the hash map.
   - an algorithm for the hash function itself. You may not use the built in hash function for the STL container - you must provide your own. Make sure to use the information provided by the utilities to help you construct your hash function, and set the initial number of buckets for the STL container.
4. Write a program that,
   (a) reads in the code words for each symbol in the image to be stored,
   (b) reads each image element from the image file and uses your key selection algorithm to generate a key for the image element
   (c) store the code word for the image element as the value in the hash map using the key selected by your algorithm at an index in the hash map based on your hash function
5. Once all of the image elements are stored in the hash map, store the hash map to a file (in binary format) and then record the size of the file.
6. Write another program – or additional code in the same program (your choice) – that is provided the binary file containing your hash map data, and the symbols and their code words, and from these, regenerate the original image.
7. You should rely heavily on using the C++ STL where possible, including the algorithms.
8. If you use pointers in your solution, they must be smart pointers.
9. If you are creating simple data types, it is fine to use structs, but please create new types from them of the form,

```
using new_type_name = struct _new_type_name {
  typename field;
  typename field;
  ...
  typename field;
};
```

10. Provide an algorithm analysis of:

    - How does your key selection algorithm function? What is its complexity?
    - How does your hash function work? Does it provide a uniform distribution across the indices of the hash map? Explain.
    - How close to $\Theta(1)$ does your solution get to retrieval from the hash map?

Figure 1: Programming Project Grading Rubric

Attribute (pts)	Exceptional (1)	Acceptable (0.8)	Amateur (0.7)	Unsatisfactory (0.6)
Specification (10)	The program works and meets all of the specifications.	The program works and produces correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results, but does not display them correctly.	The program produces incorrect results.
Readability (10)	The code is exceptionally well organized and very easy to follow.	The code is fairly easy to read.	The code is readable only by someone who knows what it is supposed to be doing.	The code is poorly organized and very difficult to read.
Reusability (10)	The code could be reused as a whole or each routine could be reused.	Most of the code could be reused in other programs.	Some parts of the code could be reused in other programs.	The code is not organized for reusability.
Documentation (10)	The documentation is well written and clearly explains what the code is accomplishing and how.	The documentation consists of embedded comments and some simple header documentation that is somewhat useful in understanding the code.	The documentation is simply comments embedded in the code with some simple header comments separating routines.	The documentation is simply comments embedded in the code and does not help the reader understand the code.
Efficiency (5)	The code is extremely efficient without sacrificing readability and understanding.	The code is fairly efficient without sacrificing readability and understanding.	The code is brute force and unnecessarily long.	The code is huge and appears to be patched together.
Delivery (total)	The program was delivered on-time.	The program was delivered within a week of the due date.	The program was delivered within 2-weeks of the due date.	The code was more than 2-weeks overdue.

The *delivery* attribute weights will be applied to the total score from the other attributes. That is, if a project scored 36 points total for the sum of *specification*, *readability*, *reusability*, *documentation* and *efficiency* attributes, but was turned in within 2-weeks of the due date, the project score would be $36 \cdot 0.7 = 25.2$.

Project Grading:

The project must compile without errors (ideally without warnings) and should not fault upon execution. All errors should be caught if thrown and handled in a rational manner. Grading will follow the *project grading rubric* shown in figure 1.

Collaboration Opportunities:

You may optionally work with one (1) other fellow classmate to develop a single solution. You ***must*** indicate in your code (via comments) the contributions of each classmate in order for credit to be awarded. Failure to indicate the contributions of all collaborative work will likely result in credit not be awarded fairly - there will be no modification of award credit without source code justification of contribution - PERIOD.