1 Programming Project Logistics

Warning: BEFORE YOU START WORKING WITH THE ALGORITHMIC PART OF THE PROJECT, WE URGE YOU TO START WORKING ON SOME AUXILIARY COMPONENTS, ESPECIALLY IF YOU ARE NOT FAMILIAR WITH THEM, SUCH AS COMMAND LINE PROCESSING and FILE-BASED BINARY INPUT/OUTPUT. IMPLEMENTATION ERRORS AT SUCH LEVEL CAUSE SEVERAL SUBMISSIONS TO RECEIVE 0 POINTS BECAUSE THEY MAKE ALTERNATE TESTING BY THE GRADER DIFFICULT.

STEP-1. Read carefully Handout 2 and follow all the requirements specified there.

STEP-2. When the submission archive is ready, upload it to moodle

BEFORE NOON-time of Tue 4th December, 2018

For penalties check Handout 1 (Syllabus). If you submit the day before the deadline, you would have sufficient time to fix other problems. If you want to get 0 pts ignore Handout 2.

You may do one or the other option or both. You may utilize the same language or not.

OPTION 1. Do the programming part related to Huffman coding in Java, C, or C++.

OPTION 2. Do the programming part related to Kleinberg's HITS, and Google's PageRank algorithms in Java, C, or C++.

2 OPTION 1: Huffman Coding (134 points)

Huffman coding. Implement Huffman coding as explained in class and described in the Subject notes. This assumes that you will also provide an implementation of a BinaryHeap (aka array with the required Heap properties) with operations as described in class. Non-compliance to this will automatically gain you a zero. (See Handout 2.) Your implementation must work with arbitrary files binary or not. We will test it minimally on a text file, PDF, and JPG or MPEG. Avoid getting distracted with the Wikipedia code: it is not PrP compliant, and its use will cause a violation of the NJIT Honors code (see the last page of Handout 1/Syllabus under Collaboration).

Argument `filename` in the command-line below is an arbitrary file-name. It can be `p1610s18.pdf`, or `p1610s18.tex`, or `image1.jpg`, or say `01aug25_flare.mpg`, or something else. The suffix (sometimes called the extension) is what follows the dot (eg pdf, tex, jpg, mpg). Although we do not ask for your implementation to be optimal and fast we require that it is relatively efficient. Thus Huffman coding (encoding or decoding) of a 2MiB file should not take more than 15-30 seconds in Java, C, or C++ on an AFS machine with reasonable load. Moreover your implementation should result in some compression savings for reasonable test files (though do not expect any in small text files or jpg files).

As stated under RULE 1, resolve issues with command-line processing or file-based input/output early in the semester. Never prompt for input as testing would be automated to some degree. So command line processing to the specifications is a MUST. My suggestion is to always read files in binary. If you don't know what i mean, pick a pdf file build a histogram of its bytes add up its frequencies and make sure that sum is equal to the size of the file as reported by an `ls -l` in a UNIX/Linux/OSX system. Write code to copy the file into another file and compare them using md5sum (a Unix/Linux/etc command).

```
 // Encode : henc  encodes filename     into    filename.huf  and filename     gets erased
 //          thus x.pdf generates x.pdf.huf and then x.pdf gets erased
 // Decode : hdec  decodes filename.huf into    filename      and filename.huf gets erased
 //          if  filename already exists, it gets overwritten
 //          thus x.pdf.huf generates x.pdf and then x.pdf.huf gets erased
 // Note   : Per Handout 2  you should use   hencWXYZ or hdec_WXYZ for henc and hdec
% java henc filename.pdf
% java hdec filename.pdf.huf
% ./henc    filename.pdf
% ./hdec    filename.pdf.huf
  //   EXAMPLE testing steps
% cp p1610s18.pdf  file1        // copy into file1
% java henc file1               // Huffman encode file1 into file1.huf
% rm -rf file1                  // If not already removed, we remove it to continue testing
% java hdec file1.huf           // Huffman decode file1.huf
% diff   p1610s18.pdf file1     // If different something went wrong! diff might not work
% md5sum p1610s18.pdf file1     // Should have same signatures; this is better
  //   EXAMPLE C or C++
% cp p1610s18.pdf  file1
% ./henc file1
% rm -rf file1
% ./hdec file1.huf
% diff   p1610s18.pdf file1
% md5sum p1610s18.pdf file1
```

Deliverables. Read and follow Handout 2. At a minimum for a Java implementation, `hencWXYZ.java`, `hdecWXYZ.java`, `prp_WXYZ.txt`, inside an archive `prp_WXYZ.zip` or `prp_WXYZ.tar`. Or say for a C++ implementation, `hencWXYZ.cpp`, `hdecWXYZ.cpp`, `prp_WXYZ.txt`, inside an archive `prp_WXYZ.zip` or `prp_WXYZ.tar`.