Fall 2023 - Analysis and Design of Algorithms
Programming Assignment 2
Due: Thursday, Dec 21 2023

Part 1 - Weighted Activity Selection (20%)

Implement a dynamic programming solution for problem 1 in sheet 6 (weighted activity selection). Any dynamic programming solution will be accepted as long as it's $O(n^3)$, but you are encouraged to implement an $O(n \lg n)$ solution. The $O(n \lg n)$ is not difficult.

Specifications

- It is required to submit a runnable jar with the name **activity_<id>.jar**, where <id> is your academic id number. The jar must include the source code.

- The absolute path of an input file will be provided as a command-line argument to your program. Your jar will be run using the following command:

  ```
  java -jar activity_<id>.jar absolute_path_to_input_file
  ```

  The input file will have the number of activities *n* in the first line, followed by *n* lines. Each line will contain the start time, finish time and weight of one of the input activities. All data will be integers and will be separated by spaces, e.g.,

  ```
  3
  1 2 1
  2 3 2
  3 4 5
  ```

- Your program should output a file that has the maximum possible weight that can be obtained by choosing a mutually-compatible set of activities. The file should appear in the same directory as the input file. If the input file name is test1.in, then the output should be **test1_<id>.out**.

Part 2 - Huffman's Algorithm (80%)

In this part, it is required to implement Huffman's algorithm that we discussed. Your implementation should allow compressing and decompressing arbitrary files. As discussed in class, the implementation should collect statistics from the input file first, then apply the compression algorithm. Note that you will need to store a representation of the codewords in the compressed file, so that you can decompress the file back.
Your program should have the capability of considering more than one byte. For example, instead of just collecting the frequencies and finding codewords for single bytes. The same can be done assuming the basic unit is *n* bytes, where *n* is an integer.

Specifications

- You will submit a single runnable jar that will be used for both compression and decompression. Your jar should be named as **huffman_<id>.jar**. Replace id with your id number. The jar must include the source code files.
    - To use it for compressing an input file, the following will be called:
      `java -jar huffman_<id>.jar c absolute_path_to_input_file n`
        - c means compressing the file.
        - n is the number of bytes that will be considered together.
    - To use it for decompressing an input file, the following be called:
      `java -jar huffman_<id>.jar d absolute_path_to_input_file`

- If the user chooses to compress a file with the name **abc.exe**, the compressed file should have the name **<id>.<n>.abc.exe.hc** where <id> should be replaced by your id number, and <n> should be replaced by **n** (the number of bytes per group). The compressed file should appear in the same directory of the input file. The program should print the compression ratio and the compression time.
- If the user chooses to decompress a file with name **abc.exe.hc**, the output file should be named **extracted.abc.exe.** This should appear in the same directory of the input file. You don't need to include the id number here.
  The program should print the decompression time in this case.

Analysis Requirements

- Submit a report that shows the compression ratio when running your implementation on the following files for different values of n = 1, 2, 3, 4, 5.
    - File 1: This file is from the NIH genetic sequence database. Note that the file is already compressed. Extract the file first, then run the experiments on the uncompressed file (gbbct10.seq). Note that the experiments here could take time, as the file is large. You should test your program and make sure of correctness on small inputs first. Also, try to implement your code efficiently to save time.
    - File 2: The PDF of the greedy algorithms lecture.
- Compare the compression ratio you got in both cases with the compression ratio of 7-zip: https://www.7-zip.org/. Try to explain the observations.
  Note: The compression ratio can be defined using multiple ways. You can calculate it in the same way as 7-zip does, which is the size of the compressed file divided by the size of the original file.

Policies and Submission Requirements

- Use Java for your implementation.
- You will submit your work as runnable jars and a report that includes the analysis results of the second part. The jars should include the source files as mentioned previously.
- Please follow all the academic integrity guidelines. You are expected to write the code individually without any unauthorized help and without checking online implementations that solve the same problems.

  In order for your assignment to be graded, please include the following statement in your report and the main() files of your code.

  “I acknowledge that I am aware of the academic integrity guidelines of this course, and that I worked on this assignment independently without any unauthorized help”.
- Late submissions are not allowed unless there is a valid documented excuse.
- More details about the submission will be posted by the TAs near the deadline.

Progress Update

To make sure that you will start early, you will be required to submit a progress report on Thursday, December 14th. You will send a **secret** gist link that has the files that you worked on so far, plus a simple readme file that describes what you have done. The code does not have to be working, but it must show a decent start. Here is a dummy example to show how a progress update could look like.
<**The progress update form will be updated soon**>. This is not optional. Your assignment won't be graded if you don't submit a checkpoint.