Algorithms and Data Structures

Laboratory 4

**Exercise 1 Spell checking (6+4+4+2+4)**

1. Implement functions that calculate distance between two strings:
   1. Levenshtein distance
   2. Hamming distance
   3. Indel distance
2. For Levenshtein and Hamming distance, suggest modification so that more probable spelling errors (basing on your experience in typing) have less impact on the distance between two strings (e.g. it is easy to mix “o” and “p” when typing)
3. Basing on these distances, or any other you may want to add, design and implement a method of suggesting a correct word from the file “words\_alpha.txt”, when an incorrect word is inserted.
4. Implement a program that takes a text file as an input and returns the corrected text in another text file. The file should contain only lowercase alphabetical characters with words separated by single space.
5. Using the word database, implement auxiliary structures to speed up the search (hint: make use of character frequency, relative distance between words)

**Exercise 2 (Huffman coding) (3+4+2 points)**

Starting with an array, implement a binary heap. With use of this heap, implement a priority queue. Then, use this priority queue to generate Huffman codes for symbols in the string taken as input. Your program should return the input string encoded according to Huffman algorithm. You should also print Huffman codes for each symbol. Calculate the compression ratio for the input string.

Evaluate your program on the following expression:

I love data structures

Encoded (note: this is one example of correct encoding): 0110011110001001111010101111110111100011000101010111100100011110000

0110011110001001111010101111110111100011000101010111100100011110000

0100001010100011111010110111100110001101111100010111010011111110000

You may be asked to evaluate other expressions. Do not use any built-in heap implementation.