Problem Set 2

Course: Algorithms for optimization and statistical inference (2014)

1. Given two non-deterministic finite automata (NDFSM) deciding languages $L_1, L_2 \subset \Sigma^*$, build a new NDFSM that decides $L_1 \cap L_2$.

2. Devise a (polynomial) method to: given a NDFSM, find an input accepted by the machine (if it exists) or to conclude that there is none (and thus the machine decides the language $L = \emptyset$)

3. Prove that the language $L = \{a^n b^n : n \in \mathbb{N}\}$ is not regular (hint: assume that is decided by a deterministic FSM and find a contradiction).

4. Prove that the language $L' = \{w \in \{a, b\}^* : \# \{i : w_i = a\} = \# \{i : w_i = b\}\}$ is not regular. (hint: write L of the previous exercise as $L' \cap L''$ for some regular language L'')

In the programming language of your choice, implement:

5. (diff) The Edit Distance algorithm, with recursion given by

$$L(s_1, \ldots, s_k; t_1, \ldots, t_n) = \min \left\{ \begin{array}{c} L(s_1, \ldots, s_{k-1}; t_1, \ldots, t_{n-1}) + (1 - \delta_{s_k t_n}) \\ L(s_1, \ldots, s_{k-1}; t_1, \ldots, t_n) + 1 \\ L(s_1, \ldots, s_k; t_1, \ldots, t_{n-1}) + 1 \end{array} \right\}$$

Your program should accept two strings as inputs and output the value of the distance L. Make your program show the following additional output:

   (a) the sequence of modifications needed to convert the first string to the second.

   (b) The shortest *alignment* between the two inputs: the program should output two lines of the same length; each will have one of the input strings with additional internal space padding in such a way that the the hamming distance (number of different entries) of the two padded strings is the smallest possible. As an example, $L(ACG, CTG) = 2$ and the output should be

```
AC.G
.CTG
```

6. Program a generic NDFSM M: decide data structures to store a generic NDFSM in memory (this is similar to storing a graph), and program an algorithm to "run" M on a given input w: for each letter read, you program should update the set of possible internal states on which the machine could be (this set can be stored in memory simply as a 0/1 flag for each internal state). The automata will accept the input if one of the states after the full string was read is final.

7. (grep) Using the simple regular expression parser given in the *materials* and problem 4, build a regular expression *recognizer*; that is; your program should accept a regular expression R and an input string w and build the corresponding NDFSM M, and then run M on w, answering **yes** if and only $w \in L(M)$.

8. Implement the method of Problem 2.

9. (sort) Implement **quicksort,** and run it on 1000 instances of N random numbers with $N = 100, 200, 400, 800, 1600, 3200, 6400$. For each instance, make the program compute the number of comparisons c_N; plot the mean $\mu_N = \langle c_N \rangle$ and the standard deviation $\sigma_N = \sqrt{\langle c_N^2 \rangle - \langle c_N \rangle^2}$ of the number of comparisons vs. N. Then, make the plot in log-log scale. Remember that for a given list $[a_1, \ldots, a_n]$, quicksort implements the following recursion:

$$\text{qsort}\left([a_1, \ldots, a_n]\right) = \text{qsort}\left([a_i : i > 1 \wedge a_i \leq a_1]\right) \cdot [a_1] \cdot \text{qsort}\left([a_i : a_i > a_1]\right)$$

where $\cdot$ means concatenation of lists.