Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignaturas: Diseño de Algoritmos - Algoritmos II
Primer Cuatrimestre de 2017

Trabajo Práctico No. 1

La resolución de este trabajo práctico debe ser enviada a través del moodle de la materia `http://dc.exa.unrc.edu.ar/moodle`, antes de las **23:55 del viernes 19 de Mayo de 2017**.
El código provisto como parte de la solución a los ejercicios deberá estar documentado apropiadamente (por ejemplo, con comentarios en el código). Aquellas soluciones que no requieran programación, como así también la documentación adicional de código que se desee proveer, debe entregarse en **archivos de texto convencionales o archivos en formato PDF** unicamente, con nombres que permitan identificar facilmente su contenido.
Tanto la calidad de la solución, como el código y su documentación serán considerados en la calificación. Recuerde además que los trabajos prácticos **no tienen recuperación**, y que el trabajo en la resolución de los ejercicios debe realizarse en grupos de 3 personas.

Ej. 1. Los autómatas finitos determinísticos son máquinas de estados, reconocedoras de lenguajes regulares, con diversas aplicaciones en Ciencias de la Computación, en la construcción de compiladores, en ingeniería de software y otras áreas. Formalmente, un autómata finito determinístico $M = (Q, \Sigma, \delta, q_0, F)$ consta de:

- $Q = \{q_1, q_2, ..., q_m\}$, un conjunto finito de estados,
- $\Sigma = \{a_1, a_2, ..., a_n\}$, un conjunto de símbolos (el alfabeto),
- $\delta : Q \times \Sigma \to Q$, una función de transición de estados,.
- $q_0 \in Q$, un estado inicial, y
- $F \subseteq Q$, un conjunto de estados finales.

Definir un autómata que reconozca *exactamente* un lenguaje es una tarea en muchos casos compleja, que demanda refinamientos tanto del lenguaje a capturar como de la definición de la máquina de estados en sí. Para muchos desarrolladores, muchas veces es fácil identificar cadenas que uno quisiera que la máquina a construir reconozca, y otras que quisiera que la máquina *no* reconozca, pero construir la máquina propiamente dicha es una tarea sustancialmente más compleja.

Para ayudar a resolver este problema, desarrollaremos una aplicación que, dado un alfabeto Σ, una cantidad máxima de estados k,y dos conjuntos de cadenas *pos* y *neg*, intente construir el autómata con menor número de estados, acotado por k, que reconoce las cadenas en *pos* y *no* reconoce las cadenas en *neg*. Esta aplicación apunta a sintetizar autómatas a partir de ejemplos positivos y negativos.

Ej. 2. Scrum es una de las metodologías ágiles de desarrollo de software más utilizadas en la actualidad. Algunos de los principios fundamentales que rigen la misma son la construcción rápida de prototipos, la entrega constante de nuevas funcionalidades y la flexibilidad para redefinir requisitos mientras el desarrollo se encuentra en ejecución. Para esto, Scrum sugiere planificar ciclos cortos de desarrollo, en los cuales se trata de maximizar la cantidad de funcionalidades a implementar, en función de las prioridades que el cliente le asigna a las mismas, y los recursos disponibles en el equipo para desarrollarlas. Los ciclos de desarrollo (*sprints* en Scrum) tienen una longitud acotada (generalmente, entre 2 y 4 semanas). Al finalizar un sprint, el cliente recibirá una entrega (*release*) del sistema enteramente funcional, con las funcionalidades que se establecieron como prioritarias y se definieron para ese sprint de desarrollo.

Scrum requiere mantener un *backlog* de producto, que consiste de una lista de características a incorporar al producto de software, acompañadas de una descripción, un valor (vinculado a la prioridad de la misma, y a la utilidad que le proveería al cliente su desarrollo), y una estimación del costo de desarrollo, e.g., en horas hombre. Un problema relevante en este contexto es la producción del *backlog de sprint*, es decir, la selección de qué características se desarrollarán en el próximo sprint, maximizando el valor global de las mismas, y respetando la duración del sprint (es decir, que de acuerdo a la estimación de costos en tiempo, pueda completarse dentro de la duración del sprint).

Se desea desarrollar una aplicación que, dado un backlog de producto, con cada funcionalidad acompañada de su valor y su estimación de costo en tiempo, y una duración para el sprint, produzca eficientemente un backlog de sprint, que maximice el valor global de las características a desarrollar, y que pueda completarse, de acuerdo a las estimaciones, en el tiempo previsto para el sprint.