Paradigmas de Programación
Licenciatura en Ciencia de la Computación, USACH
Profesor: Pablo Pérez Lantero

Trabajo de Laboratorio 1

(Fecha de entrega: 4 de mayo de 2017)

Introducción

Para el desarrollo de un nuevo robot, se pretende desarrollar un software que automatice el proceso de conversación con los humanos. El proceso es simple, el humano puede decirle frases (hechos) al robot que serán aprendidas o memorizadas por este, y además el humano podrá hacer preguntas al robot que serán respondidas por este último de acuerdo al conocimiento adquirido durante la conversación con el humano. La conversación humano-robot se realizará en el idioma Inglés.

El funcionamiento del algoritmo general para realizar este tipo de conversaciones con el robot se explica a continuación. La voz del humano es analizada a través de un Módulo de Reconocimiento de Voz (MRV) y convertida a texto plano. El texto es después analizado por un Algoritmo de Respuesta Automática Adaptativo (ARAA). Las preguntas hechas por el humano son reconocidas y respondidas automáticamente. Las respuestas son posteriormente convertidas a sonido por un Módulo de Conversión Texto-Sonido (MCTS). Los módulos MRV y MCTS ya han sido implementados, solo falta implementar el módulo ARAA.

A usted se le ha dado la tarea de implementar el módulo ARAA. Como su algoritmo debe adaptarse a las frases o hechos recibidos en la conversación con el humano, usted no cuenta con ningún conocimiento básico de antemano, pero debe ser capaz de analizar las frases en Inglés recibidas y recordar los hechos mencionados en ellas. Cuando alguna pregunta sobre algún hecho es realizada, se debe responder correctamente con la información que tenga al respecto en la base de conocimiento adquirido hasta el momento. Como los módulos MRV y MCTS ya han sido implementados, la entrada y salida del módulo ARAA se hará en forma de texto.

Se requiere implementar en `Python` el programa `araa.py` que si no se le pasan parámetros al mismo, lea el archivo de entrada `in.txt` y produzca el archivo de salida llamado `out.txt`. De lo contrario, se le pasarán dos parámetros de entrada, uno con el nombre del archivo de entrada y otro con el nombre del archivo de salida. Es decir, el programa `araa.py` debe poder ser ejecutado de las siguientes dos formas.

```
1  \> python araa.py
2  \> python araa.py input output
```

En la primera el archivo `in.txt`, que debe estar en la misma carpeta que `araa.py`, es cargado y se genera dentro de la misma carpeta la salida `out.txt`. En la segunda forma, el parámetro `input` es el nombre (o ruta) del archivo de entrada, el cual es cargado y se genera como salida el archivo con nombre (o ruta) especificado en el parámetro `output`. Siempre que se ejecute el programa, si el archivo a generar ya existe entonces se sobreescribe automáticamente.

Descripción del archivo `in.txt`

Este archivo está formado por un entero positivo T en la primera línea, que indica la cantidad de diálogos que se desean analizar. Cada diálogo consiste de cero o más oraciones las cuales pueden ser una afirmación o una pregunta. Las afirmaciones terminan con un punto (.) y las preguntas con un signo de interrogación `?`. Existe una línea extra por cada diálogo, la cual termina con un signo de exclamación `!`. El signo `!` significa que se acabó el diálogo y se pasa al diálogo siguiente o se termina la entrada en dependencia del entero positivo T.

Las oraciones pueden contener palabras, espacios y signos de puntuación. Todas las palabras contienen letras del alfabeto Inglés y son "**case-insensitive**" (no existe diferenciación entre las mayúsculas y las minúsculas) por lo que usted debe **garantizar que todos los textos estén en letras minúsculas antes de ser procesados**. No hay espacios innecesarios entre las palabras y una línea no contendrá más de 128 caracteres. Las bases de conocimientos de los diálogos son independientes, o sea, una afirmación hecha en un diálogo no tiene validez en otro.

Cada **afirmación** tiene una de las siguientes dos formas ("_" denota el caracter espacio):

```
1  (+|-)subject_predicate[s][_object].
2  (+|-)subject_(don't|doesn't)_predicate[_object].
```

Los corchetes denotan partes opcionales, y la línea vertical dos posibles variantes. Por ejemplo `(+|-)` indica que puede ser el símbolo `+` o el símbolo `-`. El símbolo `+` indica que la afirmación dicha por el humano comienza a tener validez para el robot, y el símbolo `-` indica que la afirmación deja de tener validez. Cuando una afirmación deja de tener validez, a partir de ese momento, no se tendrá en cuenta para responder a las preguntas realizadas (como si nunca hubiese aparecido la afirmación). `subject` es una sola palabra, nombre o pronombre en singular. `predicate` es un verbo (de una sola palabra) que denota alguna actividad. `object` puede ser cualquier texto. Cualquier par `predicate + object` determina una única actividad. Las oraciones sin la parte `_object` pueden ser consideradas como oraciones cuyo `_object` es vacío. Una afirmación donde aparezca un verbo cuyo `_object` es vacío tiene un significado independiente al de una oración donde aparezca el mismo verbo con una parte `_object` no vacía.

La primera forma de las afirmaciones denota afirmaciones positivas. La palabra `predicate[s]` significa un verbo escrito de acuerdo con el sujeto de la oración. Si el sujeto es `I` o `you` entonces el verbo tiene la misma forma que el infinitivo (e.g. like, love, etc). Con cualquier otro sujeto, la letra `s` es siempre añadida al final del verbo (e.g. likes, loves, etc.). En las oraciones no aparecerán verbos irregulares.

La segunda forma denota afirmaciones negativas. La cadena `don't` o `doesn't` también será utilizada según sea el sujeto. `don't` es usado cuando el sujeto es `I` o `you`, en cualquier otro caso es utilizada la cadena `doesn't`.

También pueden aparecer en `subject` los sujetos genéricos `everybody` y `nobody`. Al utilizar el primero se especifica que la afirmación en la que aparece se cumple para todo el mundo, mientras que al utilizar el segundo se especifica que la afirmación en la que aparece no se cumple para nadie. Ambos sujetos genéricos solo pueden ser utilizados en afirmaciones positivas. Las oraciones `nobody likes something` y `everybody doesn't like anything` dicen exactamente lo mismo pero la segunda nunca aparecerá en la entrada.

Cada **pregunta** tiene una de las tres formas siguientes:

```
(do|does)_subject_predicate[_object]?
who_predicates[_object]?
what_(do|does)_subject_do?
```

Descripción del archivo `out.txt`

Por cada diálogo del fichero de entrada, su módulo ARAA debe imprimir la línea "Dialogue #D:", donde D es el número en la secuencia del diálogo, comenzado por 1. Después imprimirá exactamente tres líneas por cada pregunta: la primera línea repite la pregunta, la segunda contiene la respuesta y la tercera línea estará vacía. No se imprimirá nada al analizar las afirmaciones. Después de cada diálogo, imprimirá la misma línea, marcada con el signo de exclamación `!` que se utilizó al finalizar el diálogo en la entrada. Imprima después una línea extra vacía.

La respuesta debe estar correctamente formateada para poder ser entendida por el modulo MCTS. Solo las afirmaciones que aparecen antes de una pregunta deben ser usadas para la correspondiente respuesta. Si existe alguna contradicción entre las afirmaciones, la respuesta será siempre

```
I am abroad.
```

Si la pregunta y las afirmaciones consideran el sujeto `you`, este deberá ser remplazado en la respuesta por `I`. Si la pregunta y las afirmaciones consideran el sujeto `I`, este deberá ser remplazado en la respuesta por `you`. El verbo siempre debe estar en correspondencia con el sujeto en la oración de respuesta. La forma exacta de las respuestas correctas depende del tipo de pregunta que se haya realizado. A continuación se detalla cada caso:

`1. (do|does)_subject_predicate[_object]?`

Si existe alguna afirmación positiva acerca del sujeto mencionado (o del sujeto genérico `everybody`) y el par `predicate + object`, la respuesta será:

```
yes,_subject_predicate[s][ _object].
```

Si existe alguna afirmación negativa sobre el sujeto mencionado (o sobre el sujeto genérico `nobody`) y el par `predicate + object`, la respuesta será:

```
no,_subject_(don't|doesn't)_predicate[_object].
```

En otro caso, la respuesta será

```
maybe.
```

El sujeto en la respuesta es siempre el mismo que en la pregunta.

`2. who_predicate[s]_object?`

La respuesta será de la forma:

```
subject_predicate[s][_object].
```

Si existe más de un sujeto que haya sido asociado positivamente a la acción `predicate + object`, entonces se enumerarán en la respuesta los sujetos, en el mismo orden en que las afirmaciones correspondientes aparecieron en la entrada. Los sujetos estarán separados por una coma y un espacio, excepto los dos últimos que estarán separados por `and`. Si el sujeto genérico `everybody` pertenece al grupo de los sujetos, entonces estos no deben ser enumerados, solo se debe poner el sujeto `everybody`. Si la enumeración contiene al menos dos sujetos, entonces el verbo en la respuesta se escribirá sin la `s` al final. En otro caso, el verbo siempre se debe escribir en correspondencia con el sujeto. Es decir, si hay tres sujetos la respuesta sería:

```
subject1,_subject2_and_subject3_predicate[_object].
```

Si existe alguna afirmación que considere el sujeto genérico `nobody`, y el par `predicate + object` especificado, entonces la respuesta será:

```
nobody_predicates[_object].
```

En otro caso la respuesta será:

```
I don't know.
```

3. what_(do|does)_subject_do?

Si existe una o más afirmaciones (tanto positivas como negativas) que consideren al sujeto especificado (o a los sujetos `everybody` o `nobody`), todos los pares `predicate + object` de tales afirmaciones deben ser incluidos en la respuesta en el mismo orden en que las afirmaciones correspondientes aparecieron en la entrada. Ningún par `predicate + object` puede estar incluido más de una vez (la segunda aparición en caso de existir debe ser omitida). Los pares `predicate + object` deberán estar separados en la respuesta por una coma y un espacio, excepto los dos últimos que estarán separados por la palabra `and`. Las respuestas negativas tienen la misma estructura que las positivas, pero utilizando en este caso `don't` y `doesn't`. Ejemplos:

```
subject[_(don't|doesn't)]_predicate1[s][_object1],
[_(don't|doesn't)]_predicate2[s][_object2]
_and[_(don't|doesn't)]_predicate3[s][_object3].

subject[_(don't|doesn't)]_predicate1[s][_object1]
_and[_(don't|doesn't)]_predicate2[s][_object2].

subject[_(don't|doesn't)]_predicate[s][_object].
```

En otro caso la respuesta será

```
I don't know.
```

Ejemplo

in.txt	out.txt
1 +I like hotdogs. +nobody likes to work. +everybody smiles. what do I do? who smiles? what do you do? does Joe smile? do I like to work? +everybody hurts sometimes. who walks there? +Michal walks there. who walks there? what does Michal do? -Michal walks there. who walks there? what does Michal do? do you understand? +Jonh walks there. +nobody walks there. do you understand now? -nobody walks there. do you understand now? bye!	Dialogue #1: what do I do? you like hotdogs, don't like to work and smile. who smiles? everybody smiles. what do you do? I don't like to work and smile. does Joe smile? yes, Joe smiles. do I like to work? no, you don't like to work. who walks there? I don't know. who walks there? Michal walks there. what does Michal do? Michal doesn't like to work, smiles, hurts sometimes and walks there. who walks there? I don't know. what does Michal do? Michal doesn't like to work, smiles and hurts sometimes. do you understand? maybe. do you understand now? I am abroad. do you understand now? maybe. bye!

Reglas del Juego

(1) El trabajo se realizará en equipos de a dos. La nota será única, es decir, los miembros del equipo reciben todos la misma nota. Para aprobar el trabajo es necesario haber implementado satisfactoriamente el programa, que cada integrante del equipo haya participado en la solución y domine la solución, y que además las solución sea original.

(2) La fecha de entrega es el día **viernes 4 de mayo de 2017, hasta las 23:59 horas**. La entrega se hará por correo electrónico a la dirección del profesor. En la siguiente semana cada equipo de alumnos expondrá el trabajo realizado al profesor/ayudante en la oficina. Entregas fuera de fecha serán rechazadas.

(3) Respetar la sintaxis es **obligatorio**, de lo contrario el programa no va a funcionar con ejemplos de prueba creados por el profesor. Cada ejemplo de prueba suma puntos a la nota del trabajo. Fíjese en los signos de puntuación, los espacios (un caracter espacio entre palabras), el signo `'` del `don't` es comilla simple (ASCII 039), etc.

(4) Respetar tanto el nombre del programa como la forma de usarlo descrita en la introducción, es **obligatorio**. El programa implementado será testeado automáticamente por un programa de testeo desarrollado por el profesor y el ayudante. No pasar los tests significa que se **reprueba** el trabajo.

(5) En el programa tanto las estructuras de datos como los algoritmos tienen que ser eficientes, y esto suma puntos a la nota del trabajo. El programa implementado será testeado con archivos de muchas líneas para verificar la eficiencia en tiempo y memoria del mismo.

(6) El uso de la estructura de datos diccionario es **obligatorio**. Para ello pueden dirigirse, por ejemplo, al tutorial `https://www.python-course.eu/python3_dictionaries.php`.