Editor de la Máquina de Minsky

Juan Antonio Macías García

1 Introducción

Este programa es un editor de programas para la *máquina de Minsky* (también conocida como máquina de registros) programada en Python 3 por Juan Antonio Macías. Este programa se encuentra también alojado en GitHub de Juan A. Macías.

Este editor de programas trata de emular a la máquina de Minsky, una máquina teórica equivalente a la máquina de Turing. Fue formalizada por *Marvin Minsky* en 1961.

1.1 Características de la máquina

La máquina de registros consta de una sucesión infinita de registros $R_1, R_2, \ldots$ en los cuales se puede almacenar un número natural arbitrario.

Un programa para la máquina de registros consiste de un conjunto finito de estados $\{S_0, S_1, \ldots, S_n\}$ tales que, para cada i, hay una instrucción a ejecutar por la máquina cuando alcance el estado S_i. El estado S_0 es un estado reservado para indicar al programa que debe parar

Existen dos tipos de instrucciones:

1. Sumar 1 al registro R_j y proceder al estado S_k. Esta instrucción se representará por (j, +, k).

2. Examinar el registro R_j. Si hay un 0 en dicho registro, pasamos al estado S_l. En otro caso restar 1 y proceder al estado S_k. Este tipo de insrucción se representa por (j, -, k, l).

Consideraremos que el estado S_1 es el estado inicial de la máquina.

2 Uso del editor

Este editor está programado en Python 3.3.2 y los módulos necesarios para hacer uso de él son la suite SciPy para hacer uso de los arrays N-dimensionales y el módulo tabulate para mostrar correctamente tabulados los datos de los registros. (Este módulo se encuentra entre los archivos de esta versión pero no ha sido programado por Juan A. Macías).

2.1 Funcionamiento

Una vez ejecutado el programa, indicará que deben introducirse primero un conjunto de valores iniciales para el programa y las instrucciones que compondrán

el mismo. Se detalla la forma de introducción (entre paréntesis y separadas por comas para los valores, y como representados anteriormente y en orden para las instrucciones), y una vez terminado, se preguntará al usuario qué desea hacer a continuación. Los comandos disponibles son:

1. **Comandos**: Muestra la lista de los comandos con su explicación.
2. **Introducir valores**: Comando para introducir nuevos valores iniciales.
3. **Introducir instrucciones**: Comando para introducir un nuevo programa mediante sus instrucciones.
4. **Tabla registros**: Muestra una tabla con el programa una vez evaluado, en el que se tienen los contenidos de los registros en cada paso.
5. **Mostrar datos**: Muestra los datos introducidos por el usuario, esto es, los valores iniciales y las instrucciones.
6. **Modificar instrucción**: Permite modificar una única instrucción de un programa introducido.
7. **Obtener registro**: Este comando permite recuperar el contenido de un registro en un paso determinado del programa. También permite recuperar la instrucción en dicho paso o el vector de registros completos en un paso.
8. **Depurar**: El comando depurar indica si existen problemas con el programa introducido, como bucles infinitos, que el programa no tenga condición para terminar o que haya referencias a instrucciones que no existen.
9. **Limite bucle**: En caso de que el usuario necesite un máximo de iteraciones mayor que el establecido, este comando permite cambiarlo.

Para terminar la ejecución del programa, bastará con introducir "End" como comando. No es precisa la correcta capitalización de los comandos anteriores al introducirlos.

2.2 Funciones implementadas

- *enterinitval* : Esta función pide la introducción de los valores iniciales. Estos tienen que ser introducidos entre paréntesis y separados por comas. La función separará los valores y los introduce en un vector.
- *enterinstr* : Esta función pide la introducción de instrucciones para el programa deseado. La función las separará y modificará de manera que sean introducidas en una matriz, en la cual la primera fila está formada por -2 para facilitar el uso de varias funciones posteriores.
- *applyinst* : Toma una instrucción y un estado de la máquina (esto es, un vector de registros) y aplica la instrucción a dichos registros. Devuelve el vector con los registros nuevos y el número de instrucción siguiente a introducir.

- *regtable* : Dados un vector de valores iniciales y una matriz con instrucciones, la función calcula los registros en cada paso y los almacena en una matriz. A su vez, también almacena los estados en cada paso. Además, si cree que el programa ha entrado en un bucle sin fin, parará después de *looplim* iteraciones. Una vez calculado todo, la función transformará los datos en una tabla lista para ser mostrada. La función devuelve la matriz de registros, un vector con las instrucciones de cada paso, la tabla con los datos y una variable booleana que indicará si la función ha tenido que parar el programa o no.

- *getstate* : Esta función permite recuperar el estado de un registro en un paso concreto, recuperar la instrucción en la que se encuentra el programa en un paso u obtener el vector de registros en un paso. Los parámetros son la matriz de instrucciones, el vector de valores iniciales, el registro a consultar, el número de paso y la condición de límite de iteraciones.

- *editinstuc* : Toma como parámetros las instrucciones, el número de instrucción a modificar y la nueva instrucción en forma de string. La función separará y transformará en vector a la instrucción introducida y procederá a modificar la matriz de instrucciones.

- *printusrdata* : Toma como valores el vector de valores iniciales y la matriz de instrucciones. La función transforma estos datos en un formato agradable para ser mostrados por pantalla, tal y como se representan por escrito.

- *debugprogram* : Esta función depura el programa introducido como parámetro. Comprueba si existe un bucle infinito en el programa, si éste no tiene condición de acabado, esto es, existe un 0 en alguna instrucción, o si se hace referencia a alguna instrucción que no se encuentra en el programa. Si presenta alguno de estos problemas, se notifica cuáles de ellos y el paso a seguir para arreglarlo (si procede). En caso contrario, muestra un mensaje con la conformidad del programa.

- *main* : Esta es la función principal del programa. Se ejecutará cada vez que se ponga en funcionamiento. Mostrará una serie de mensajes para indicar al usuario el funcionamiento del programa. Este pedirá la introducción de un primer programa y después se le preguntará qué desea hacer. Si lo introducido por el usuario coincide con uno de los comandos preestablecidos, el programa ejecutará la función correspondiente de las de arriba pidiendo los valores necesarios si hiciera falta. En caso de que no sea un comando válido, se le pedirá que lo introduzca de nuevo. Los comandos no son susceptibles a la capitalización.

3 Recomendaciones

En el caso de trabajar con Windows, el entorno WinPython (en su versión 3.3) posee todo lo necesario para el correcto funcionamiento del programa. Además para este entorno no es necesario realizar una instalación completa, pudiendo ser un entorno portable.

Si se trabaja con Linux o Mac OSX, un entorno posible puede ser Algorete Loopy, el cual en su web existe una detallada guía de la instalación de la suite SciPy, necesaria para el funcionamiento de este editor.

Además se recomienda un compilador con soporte utf8 para la versión en español de este programa, ya que en caso contrario, se encontrarán caracteres extraños en las tildes y demás caracteres especiales a la hora de ejecutar el programa.