UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Herramienta para medir la eficiencia de código en Python

Presentado por Guillermo Calvo Álvarez
en Universidad de Burgos — 1 de julio de 2019
Tutor: Jesús Enrique Sierra García

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

Jesús Enrique Sierra García, profesor del departamento de Ingeniería Civil, área de lenguajes y Sistemas informáticos.

Expone:

Que el alumno D. Guillermo Calvo Álvarez, con DNI 71302497J, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Herramienta para medir la eficiencia de código en Python.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 1 de julio de 2019

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

La eficiencia, definida como la capacidad de conseguir un objetivo con el mínimo numero de recursos posibles, es algo que se busca en todos los ámbitos con mayor o menor prioridad.

En el caso de la informática, es algo muy importante, ya que de ello depende muchas veces que un programa sea viable para la puesta en práctica en un entorno laboral o productivo, sobre todo si hablamos de sistemas de tiempo real.

Teniendo en cuenta esa importancia lo que se pretende es conseguir una herramienta capaz de mostrar los datos necesarios para hacer un análisis de eficiencia de códigos, enfocando la idea en el entorno Python. Utilizando archivos diseñados en lenguaje Python. (Archivos .py).

Para ello se investigará sobre los diferentes métodos de medición de eficiencia existentes, la forma de interpretar código en lenguaje Python y, diferentes maneras de realizar un análisis y hacer representaciones con la información resultante.

Cabe resaltar que en base a este trabajo se realizó un artículo sobre la metodología y la herramienta desarrolladas y fue aceptado y presentado en el congreso EuroSim 2019, el cual esta enfocado a presentar novedades en el ámbito de las simulaciones. Este artículo se incluye incluido en los anexos.

Descriptores

Eficiencia, Python, Interpretar, Análisis, Medición, Artículo.

Abstract

Efficiency, defined as the ability to achieve an objective with the minimum number of possible resources, is something that is sought in all areas with greater or lesser priority.

In the case of computer science, it is very important, since it often depends on whether a program is viable for implementation in a work or productive environment, especially if we are talking about real-time systems.

Taking into account this importance, we want to obtain a tool capable of displaying the necessary data to make a code efficiency analysis, focusing the idea on the Python environment. Using files designed in Python language. (.Py files).

To do this, we will investigate the different methods of measuring efficiency, the way of interpreting code in the Python language, and different ways of performing an analysis and making representations with the resulting information.

It should be noted that based on this work an paper was made on the methodology and tool developed and was accepted and presented at the EuroSim 2019 congress, which is focused on presenting novelties in the field of simulations. This article will be included in the annexes.

Keywords

Efficiency, Python, Interpret, Analysis, Measurement, Article.

Índice general

Índice de figuras

Índice de tablas

Introducción

La eficiencia[1] en el mundo informático, es un termino fundamental y con mucho peso, que condiciona en la gran mayoría de ocasiones todo lo relacionado a este. Cuanta mayor eficiencia menos recursos se gastan para poder realizar un objetivo y estos recursos pueden ser utilizados para otras metas. La búsqueda de una buena eficiencia tiene una gran importancia dentro del ámbito laboral, donde cada recurso ahorrado puede ser fundamental, para el desarrollo de otras tareas. Un gran ejemplo de esto son los sistemas de tiempo real, donde el tiempo de respuesta debe ser mínimo, lo que hace del tiempo un recurso muy valioso, por lo cual la eficiencia de este recurso es esencial

En este caso nos vamos a centrar especialmente en la eficiencia aplicada a código Python, ya que es un lenguaje muy utilizado, ofrece muchas posibilidades de desarrollo y la eficiencia, es algo que se puede aplicar a casi cualquier campo conocido. Para poder medir la eficiencia de un código hecho en lenguaje Python, primero se tendrá que empezar por determinar y desarrollar que es lo que hace que un código sea mas eficiente que otro y el como conseguir esa información.

A la primera pregunta la respuesta sería, "depende". Ya que según la finalidad para la que haya sido creado un programa, esta determina que factores hacen que el programa se considere mas eficiente. Por ello existen diferentes métricas[2] para según que casos.

He aquí algunas de ellas:

- Medir el tiempo de ejecución: Esta es la forma de medir la eficiencia más utilizada, esto se debe a la facilidad de llevarla a cabo, pero como gran inconveniente se podría decir que es una manera la cual devuelve resultados difícilmente repetibles. Esto es por la gran influencia que tienen algunos factores externos en los resultados que puede devolver. El mayor de estos factores son las condiciones de pruebas donde se ejecuta el código del programa, ya que si hacemos una ejecución en un mismo ordenador pero en momentos diferentes, es muy probable que algoritmos con una cierta complejidad tarde mas en ser ejecutados en un momento que en el otro. Esto se debe a que con este método para poder sacar resultados con una gran fiabilidad habría que tener un gran control sobre el entorno, lo cual requiere de una gran preparación previa.

- Complejidad O[3] Esta forma realmente nos sirve para conocer la complejidad del código, ya que mide el numero de iteraciones que hace un programa, pero por otra parte no se llega a considerar lo que tarda en ejecutar cada iteración en concreto.

- Cuenta de Operaciones: Esta métrica se basa en contar el numero de operaciones que son necesarias en la ejecución del código. No es tan fácil de llevarla a cabo como la métrica de medir el tiempo, pero a cambio los resultados son repetibles.

La métrica elegida para ser implementada en la herramienta de este trabajo a sido la cuenta de operaciones

Se a decidido implementar la medición con operaciones por ser una manera de obtener soluciones invariables sea cual sea el entorno en el que se haga y devuelve datos con una profundidad suficiente como para poder hacer todos los análisis que se requieran.

Otra ventaja de utilizar esta medición es que se puede obtener la complejidad O de un programa al hacer varios análisis de un mismo fichero con diferentes valores cada vez.

En el caso de este trabajo se ha decidido que para conseguir la información de cuantas operaciones hay en la ejecución de un programa se utilizara un interprete[4], que como bien su nombre indica, será el responsable de interpretar el código que forma el programa y tras hacerle unas modificaciones nos guardara la información que se desea.

Objetivos del proyecto

2.1. Objetivos del Proyecto

Objetivo general

Desarrollar un software capaz de dar la información necesaria para realizar un análisis de eficiencia sobre un código de extension .py.

- Poder hacer análisis de un código python a través del cálculo de operaciones.
- Mostrar esa información de una manera agradable y entendible.

Objetivos funcionales

- El usuario podrá elegir el tipo de análisis que quiera realizar: de un fichero individual o comparando varios ficheros.
- El usuario podrá elegir las operaciones que quiera que se tengan en cuenta en los resultados de los análisis.
- El usuario puede cambiar la métrica de los análisis, decidiendo como pondera cada tipo de operación
- El usuario podrá realizar varios análisis sin necesidad de salir de la herramienta
- El usuario podrá hacer una parametrización a través de los nombres de los ficheros

Objetivos técnicos

- Obtener el numero de operaciones a través del análisis por parte del interprete
- Diseñar una manera de guardar los niveles de eficiencia de cada tipo de operación.
- Obtener los datos finales con los que realizar las gráficas a través de la ponderación
- Utilizar Tkinter para realizar la interfaz de la herramienta
- Utilizar Matplotlib para generar las gráficas tras los respectivos análisis
- Entender el funcionamiento de ByteCode en Python

Conceptos teóricos

3.1. Fundamentos Básicos

La eficiencia

La eficiencia vista desde un ámbito general se definiría como el uso racional de los recursos para conseguir un objetivo. Esta definición puede ser aplicada a cualquier ámbito en el que se hable de eficiencia pero en el caso de este proyecto se aplicará esta definición en el ámbito computacional. La definición citada anteriormente menciona la palabra recursos, en el ámbito que estamos tratando, cuando se habla de recursos la palabra que más suele asociarse es tiempo. Esto no quiere decir que sea la única medida de eficiencia, ni mucho menos, también se podría poner como recurso el espacio en memoria, pero el tiempo suele ser el recurso que más condiciona y la gente más en cuenta tiene en las características de los programas.

Aquí llegamos a la razón de existir de este proyecto, ¿cómo medimos lo que tarda un programa en ejecutarse de manera fiable?. La manera más extendida para hacer esto, es ejecutar el programa que se desee y esperar a que termine, contado el tiempo pasado desde el comienzo hasta el final de este. Esta manera a pesar de ser sencilla, a la hora de la verdad es muy poco fiable. Para que los resultados obtenidos por este método sean considerados verídicos se tendría que asegurar que el entorno de pruebas tenga unas condiciones idóneas, como por ejemplo que el procesador que del ordenador donde se ejecuta el programa no este gastando recursos en otros procesos que estén activos en segundo plano. El problema de esto es que conseguir un entorno así muchas veces no es tarea fácil. Y como respuesta a este problema surge la idea principal de este proyecto, medir la eficiencia de un programa

de una manera fiable, que no se vea afectado por factores externos a este.

Para lograr un método para medir la eficiencia de un programa sin que su valor se vea afectado por el cuándo y dónde se haga el análisis, se decidió hacerlo a través de la identificación y el conteo operaciones que se ejecutan en el código del programa. Tras esto una vez se obtengan estas operaciones, se hará una ponderación a cada tipo de operación según el nivel de complejidad que se declare.

El intérprete

Un intérprete es un programa que analiza y ejecuta el código de otro programa instrucción a instrucción. Esta es la manera con la que se pueden detectar los diferentes elementos que componen un programa. Al realizar este trabajo en Python hay que comentar antes que el interprete se trata de una máquina de pila[5], lo que no deja de ser una máquina virtual que emula el funcionamiento de un ordenador. El funcionamiento principal de este se basa en almacenar y gestionar información que va recogiendo en las diversas pilas que tiene, para de esta manera poder hacer sus operaciones.

El interprete elegido para este trabajo se trata de ByteRun[6], un interprete hecho en código python y para código python. Este fue elegido por ser un interprete que fue creado con el objetivo principal el aprendizaje, lo cual conlleva que su complejidad no sea muy alta y por lo tanto es relativamente fácil implementar nuevas funcionalidades en este. Por otro lado en contra parte de esa baja complejidad hay que mencionar que ByteRun no es un intérprete muy rápido, como por ejemplo podría ser Cpython, pero como en este trabajo el principal requisito que se busca en un interprete es el de poder implementar nuevas funcionalidades en este para conseguir una forma de poder detectar las operaciones ejecutadas y guardarlas. ByteRun era una opción muy acertada que se adaptaba perfectamente a los requisitos del desarrollo.

Bytecode

ByteRun al ser un interprete de Python, no analiza directamente el código de alto nivel, lo que este analiza es un código intermedio llamado Bytecode.
El ByteCode es un código intermedio sacado a través de la traducción de un código de alto nivel al ser tratado por una serie de procesos (lexer,compiller...).

Para sacar este código intermedio se ha utilizado un modulo de Python que actúa de desensamblador y descompone el código de alto nivel.

3.2. De código a Tiempo Computacional

Una vez explicados los conceptos anteriores se para a contar los pasos llevados a cabo para conseguir traducir el código de alto nivel a una medida con la que se pueda medir la eficiencia, el cual en el caso de este trabajo se ha optado por los ciclos de reloj.
Los pasos para hacer la transformación son los siguientes:

1. Transformar el código de alto nivel a ByteCode a través del compilador.
2. Traducir y ejecutar el ByteCode a través del intérprete.
3. Obtener el diccionario de operaciones, mientras el interprete ejecuta el ByteCode.
4. Ponderar los valores recogidos en el diccionario de operaciones con los procesadores teóricos para obtener el tiempo computacional.

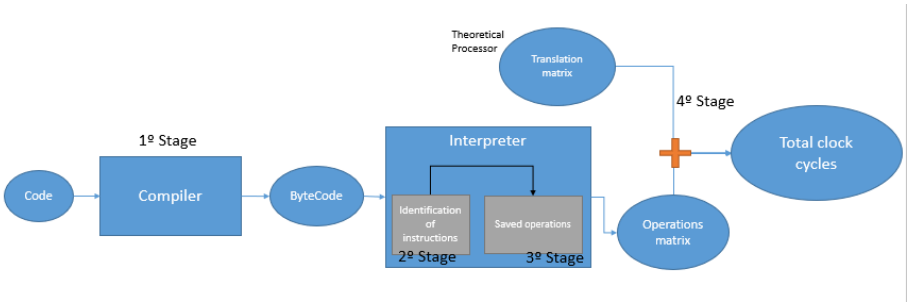


Figura 3.1: Diagrama de flujo de la transformación del código de alto nivel al tiempo computacional

1º Paso: Obtención del ByteCode

.

Como ya se ha comentado antes el ByteCode se trata de un código intermedio sacado a partir de la intervención de un compilador, para conseguir esto se cuenta con un modulo de Python llamado Dis que hace esta labor

de compilador.
De primeras el aspecto de este código se trata de una serie de bytes que a simple vista una persona no podría entender.

```
x=10
y=8
z=x*y
```

Figura 3.2: Ejemplo de un código de alto nivel

Pero el modulo Dis, cuenta con una función que permite ver el ByteCode en una especie de lenguaje ensamblador, lo cual resulta muy útil para entender la estructura del código.

```
1           0 LOAD_CONST               0 (10)
            3 STORE_NAME               0 (x)

2           6 LOAD_CONST               1 (8)
            9 STORE_NAME               1 (y)

3          12 LOAD_NAME                0 (x)
           15 LOAD_NAME                1 (y)
           18 BINARY_MULTIPLY
           19 STORE_NAME               2 (z)
           22 LOAD_CONST               2 (None)
           25 RETURN_VALUE
```

Figura 3.3: Ejemplo del ByteCode en vista de ensamblador

La forma de entender este tipo de código es la siguiente:

Como ejemplo usaremos la primera fila donde se encuentra la instrucción ”LOAD CONTS”, la primera columna muestra el número 1 que representa la línea donde está la instrucción en el código de Python, la segunda columna es un índice que indica que la instrucción ”LOAD CONTS”, en En este caso, está en la posición 0, la tercera columna es el nombre de la instrucción en sí misma, mostrándola con un nombre comprensible para una persona, la cuarta columna indica la posición de ese argumento en la pila y la última columna muestra cuál es el argumento para la instrucción.

2º Paso: Obtención del ByteCode

Una vez obtenido el Bytecode entra en acción el interprete, en nuestro caso el ByteRun, el cual es capaz de ir analizando y ejecutando el bytecode obtenido.

3º Paso: Matriz de operaciones

El ByteRun no es capaz ir almacenando las operaciones que va traduciendo. En este momento es donde se debe implementar una nueva funcionalidad en el intérprete para que sea capaz de hacer esto, para así mas tarde poder hacer un cálculo de la eficiencia

Cuando el ByteRun lee una instrucción y lo primero que tiene que hacer es identificar que tipo de instrucción esta leyendo, hay muchos tipos diferentes desde un "LOAD", a un "LOOP", o un "ADD", pero a nosotros no nos interesan todas las instrucciones, solo queremos almacenar aquellas que son consideradas por nosotros como "operaciones", o lo que es lo mismo, instrucciones que necesiten de algún tipo de calculo por parte del procesador, y que por lo tanto tenga un tiempo de ejecución.

Por suerte para nosotros el interprete ya hace distinciones entre los diferentes tipos de instrucciones y guarda en una lista todos aquellos valores en los cuales requiere hacer una operación.

Aprovechando esto, cada vez que alguna de esas instrucciones es llamada guardamos el nombre que tiene esta al ser visualizado el bytecode mediante el modulo dis[7] en una matriz, el cual decidimos llamar matriz de operaciones.

Esta matiz tiene la siguiente estructura:

Operación	Operador	INT	STR	FLOAT	BOOL
ADD	int	1001	0	200	0
ADD	flo	2004	0	2450	0
MULTIPLY	int	0	70	0	0
MULTIPLY	flo	0	0	555	0
DIVIDE	int	600	0	30	0

Tabla 3.1: Ejemplo de la estructura básica de la Matriz de operaciones.

La estructura de la matriz se basa en el hecho de que un operación necesita de dos valores con los que operar, y dependiendo del tipo de estos valores, dos mismas operaciones con diferentes valores, pueden llegar a tener nivel de eficiencia mucho mas bajo o alto que el otro.

Como se ve en la tabla 3.1, las dos primeras columnas de la matriz hacen de índice para las tuplas, la primera columna guarda el tipo de operación mientras que la segunda guarda el tipo del primer operador. Y la primera tupla de la matriz hace de indice para las columnas guardando los tipos del segundo operador, para así poder determinar la celda exacta donde se lleva la cuenta de una operación concreta.

Según el ByteRun va analizando todo el código cuando detecta una instrucción dentro de la lista de operaciones este llama a una función, la cual busca la posición celda correspondiente a la operación encontrada, una vez hallada incrementa en 1 el valor que se encuentre en la celda.

De esta manera cuando el ByteRun acaba de analizar todas las operaciones, conseguimos tener una matriz con el registro total de operaciones que han surgido a lo largo de la ejecución del programa.

4º Paso: Ponderación

Pero con solo tener los tipos de operaciones no es suficiente. Para hacer un análisis de la eficiencia es necesario saber el nivel de eficiencia que dispone cada tipo de operación, ya que no todas las operaciones tiene la misma complejidad y el procesador necesita hacer mas cálculos para ejecutarlos. Esto se consigue gracias al procesador teórico.

Procesadores Teóricos

El procesador teórico tiene la función de darle un nivel de eficiencia a cada tipo de operación. Se le puso este nombre por ser la parte que simula la complejidad que le supondría a un determinado procesador, de esta manera según los valores que se guarden en el procesador se pueden simular diferentes tipos de entornos.

Estos procesadores se tratan de unos ficheros de tipo .csv que guardan una matriz, llamada matriz de traducción. Esta matriz tiene una estructura pensada para poder almacenar los diferentes niveles de eficiencia que se les quiere asignar a cada operación en cada celda según se desee.

Sigue la misma estructura que la matriz de operaciones. Las dos primeras celdas de la matriz hacen de índice para las tuplas, la primera columna guarda el tipo de operación mientras que la segunda guarda el tipo del primer operador. Siguiendo esta estructura si tuviéramos por ejemplo una suma de un entero con un decimal, para encontrar el nivel de eficiencia que habría que aplicarle a esta operación en concreto la herramienta busca primero el tipo del operador, en este caso seria "ADD" y seguido a esto busca en la segunda columna deja el valor "int", una vez encontrados estos dos valores ya tendría localizada la tupla donde se encuentra el valor deseado, y aquí es donde entra en juego el indice de las columnas que se haya en la primera fila de la matriz, esta guarda los diferentes tipos que puede tomar el segundo operador, en este caso "flo", y con esto puede localizar que celda de la tupla es la que se necesita.

Operación	Operador	INT	STR	FLOAT	BOOL
ADD	int	1	4	2	3
ADD	int	2	2	2	2
MULTIPLY	int	1	2	3	2
MULTIPLY	int	1	4	2	2
DIVIDE	int	2	5	3	3

Tabla 3.2: Ejemplo de la estructura básica de la Matriz de traducción.

Calculo de la Ponderación

Una vez tenemos tanto la matriz de operaciones como un Procesador teórico ya solo falta realizar la ponderación para obtener los datos necesarios, en este caso los ciclos de reloj, para hacer los análisis oportunos.

La formula aplicada para hacer la ponderación es la siguiente:

$$Tcloc = \sum_{i,j}^{(N,M)} M_O(i,j) M_T(i,j)$$

Figura 3.4: Formula aplicada para hacer la ponderación.

Donde N y M son el número de filas y columnas de M O. Debe prestarse atención al tamaño de M T debe ser más grande que el tamaño de M O. Aunque esta herramienta se ha desarrollado completamente en Python, y actualmente solo puede analizar archivos .py exclusivamente, una de las ideas principales es que luego se pueda usar en diferentes entornos, para diferentes tipos de lenguajes y pueda realizar una variedad más amplia de análisis, para así adaptarse mejor a las diferentes demandas que puede requerir el análisis en profundidad de la eficiencia de un código, en un entorno más exigente. Se decidió comenzar con este tipo de lenguaje, ya que Python hoy en día se usa ampliamente en diferentes campos como por ejemplo la robótica, gracias a la gran variedad de bibliotecas que tiene. Y sería muy interesante ver los resultados que se podrían obtener en todos estos campos.

3.3. Tipos de Análisis

Con los datos obtenidos ya se pueden hacer diferentes tipos de análisis, en el caso de esta herramienta se han implementado dos tipos:

- Análisis individual
- Análisis comparativo

Análisis Individual

En este análisis solo se permite la entrada de un fichero,de formato .py, el cual tras pasar por los pasos citados en los puntos anteriores, muestra una gráfica circular con las diferentes operaciones que el interprete(ByteRun)a detectado al ejecutarlo. Y ademas de esto deja recalcular el resultado mostrado en el gráfico según las operaciones que queramos mostrar o no y el procesador con el que se quiera hacer la ponderación.

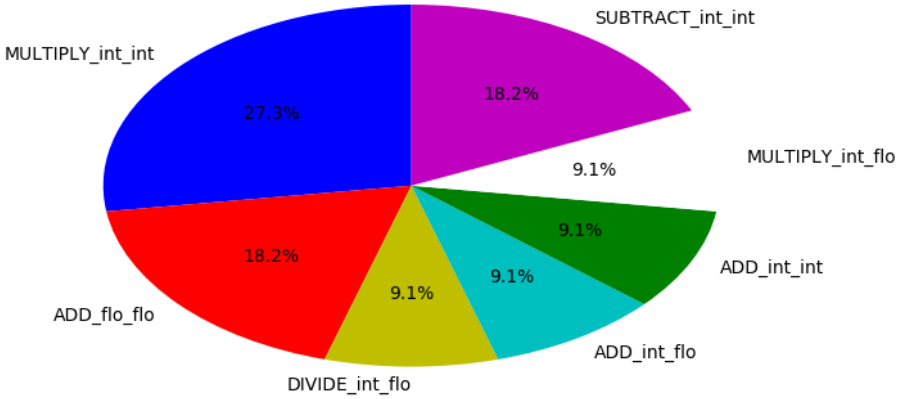
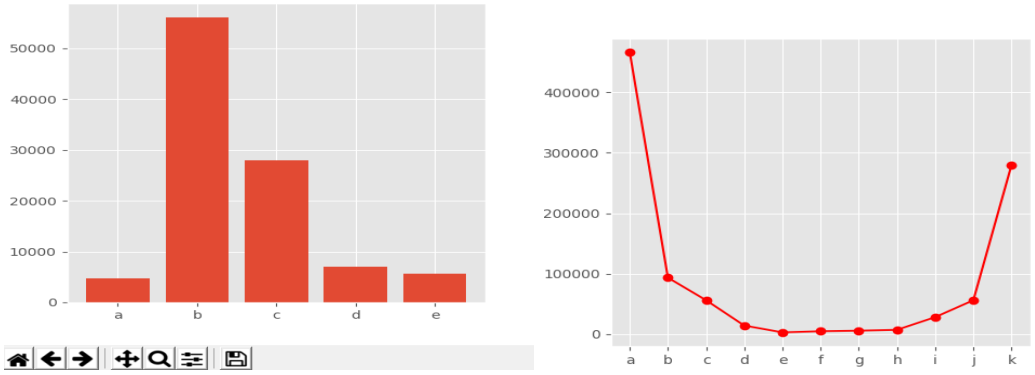


Figura 3.5: Gráfica resultante del análisis individual

Este análisis esta pensado para optimizar el código del programa, ya que de esta manera puedes detectar que operaciones son las que mas recursos consumen.

Análisis Múltiple

En este análisis se eligen varios ficheros, los cuales son ejecutados y analizados para calcular los ciclos de reloj que tarda en ejecutarse cada uno. A la hora de mostrar los resultados puede variar la forma de represar la información según el numero de ficheros que se hayan escogido. En caso de ser menos de 10 saldrá una gráfica de barras en la cual cada barra representa el total de ciclos de reloj de cada fichero. Y en caso de que se metan mas de 10 se cambiaría por una gráfica de puntos para facilitar la visibilidad de los resultados. Al igual que en el análisis individual en todo momento se pueden cambiar los resultados de la gráfica cambiando las operaciones que queramos que se tengan en cuenta y el procesador con el que se quiera ponderar las operaciones.



(a) Con menos de 10 ficheros (b) Con más de 10 ficheros

Figura 3.6: Gráficas resultantes del análisis múltiple

Este análisis esta pensado para compara ficheros que tengan como finalidad hacer lo mismo, y de esta manera saber cual de ellos tiene en que clase de entornos mejor eficiencia.

3.4. Caso práctico de un Análisis

Para entender mejor todos los conceptos teóricos a continuación se explicara un caso practico de un análisis múltiple.

En este análisis utilizaremos ficheros con el mismo código, pero a cada uno de estos se le cambiara algún parámetro, para así averiguar cual responde con mayor eficiencia. El código utilizado se muestra en la figura 3.6.

```
tEnd=100; ts=0.1
KP=2; KD=8; KI=1
t=0; e=0; eSum=0; eOld=0
while t < tEnd:
        t=t+ts
        e=SIGNAL-REF
        OUT=KP*e + KD*(e-eOld)/ts + KI*eSum*ts
        eOld=e
        eSum+=e
```

Figura 3.7: Vista del código de alto nivel que se utiliza para hacer el análisis

Una vez seleccionados los archivos que queremos analizar, se llama al modulo dis para traducir los códigos a ByteCode (figura 3.8).

Cuando ya se ha logrado el ByteCode, es cuando el intérprete comienza a realizar sus funciones y genera el contenido de la matriz de operaciones (tabla 3.3).

Operación	Operador	INT	STR	FLOAT	BOOL
ADD	int	2001	0	1002	0
ADD	int	0	0	1001	0
MULTIPLY	int	3003	0	1001	0
SUBTRARCT	int	2002	0	0	0
DIVIDE	int	0	0	1001	0

Tabla 3.3: Ejemplo de la estructura básica de la Matriz de operaciones.

En la tabla anterior, una de las matrices de operación se ha mostrado de manera resumida, solo se muestran las tuplas que contienen algún valor. Cabe señalar que habría n Matrices, una para cada archivo analizado.

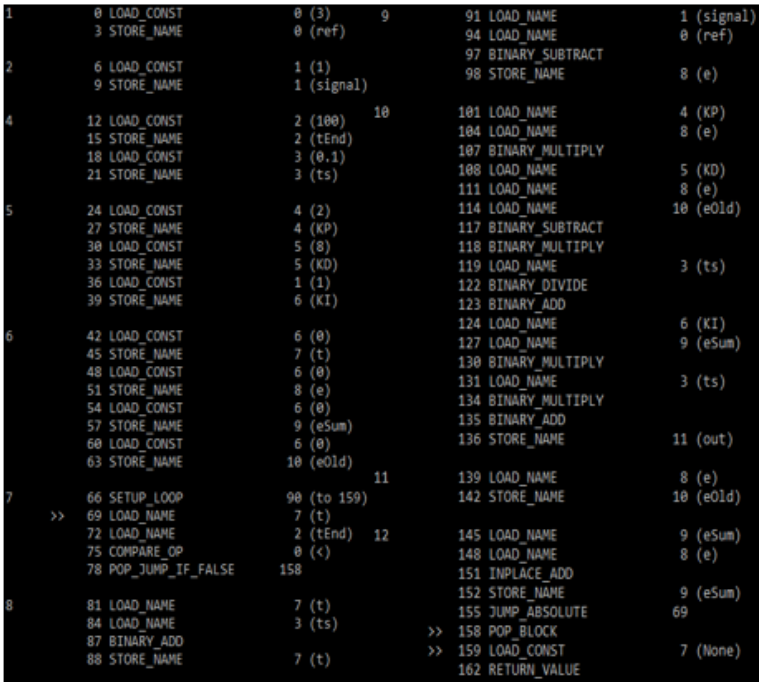


Figura 3.8: Vista del ByteRun resultante de uno de los ficheros

Después de calcular la matriz de operaciones, debemos elegir el procesador deseado para ponderar las operaciones contenidas en la matriz de operaciones.

Aquí elegimos de acuerdo al campo en el que queremos que funcione el código. Y para eso tenemos que saber qué valores hay dentro de la matriz de traducción (tabla 3.4).

Operación	Operador	INT	STR	FLOAT	BOOL
ADD	int	1	4	2	3
ADD	int	2	0	2	2
MULTIPLY	int	1	2	3	2
SUBTRARCT	int	1	4	2	2
DIVIDE	int	2	5	3	3

Tabla 3.4: Ejemplo de la estructura básica de la Matriz de traducción.

Como en la vista de la matriz de operaciones, solo se muestran en esta vista las tuplas que tienen un valor relevante para el ejemplo que estamos ejecutando. En una vista total de la matriz de traducción hay muchas más filas.

Para verificar la validez de la técnica, la herramienta se ejecuta en un análisis de varios archivos, cada archivo tiene diferentes valores de tEnd y ts para el código de la figura 3.7. Los resultados se muestran en la tabla 3.5.

File	tEnd	ts	Clock cycles
A	500	0.015	466676
B	100	0.015	93338
C	400	0.1	56000
D	100	0.1	14014
E	100	0.5	2800
F	100	0.3	4676
G	200	0.5	6000
H	100	0.2	7000
I	100	0.05	28014
J	100	0.025	56000
K	500	0.025	280014

Tabla 3.5: Vista del valor de los parámetros de cada fichero con los ciclos de reloj resultantes.

Con todos estos datos, la herramienta hace su análisis y nos muestra los resultados de la figura:

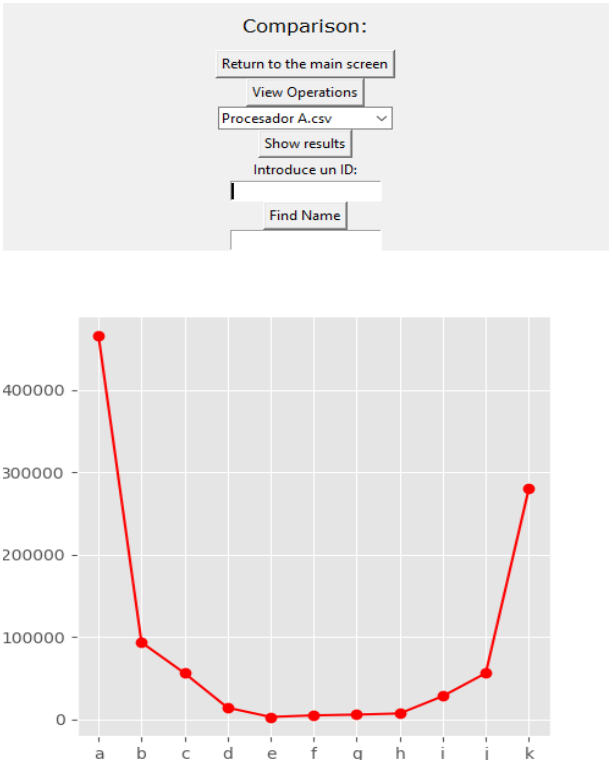


Figura 3.9: Vista de los resultados del análisis de eficiencia

Como puede verse en la figura 3.9 y la tabla 3.5, los mejores resultados se obtienen con el archivo e, con los parámetros tEnd = 100 y ts = 0.5.

Técnicas y herramientas

4.1. Herramienta de documentación

Latex

Es un sistema de composición de textos especializado en los textos técnicos y científicos. Se decidió hacer este trabajo con este sistema para componer la documentación, porque facilita de gran manera la buena estructuración del documento, gracias a su , además de ofrecer una gran cantidad de aspectos tipográficos, que consiguen dar una gran calidad y profesionalidad a los documentos resultantes y al componer Latex texto mediante marcas en un archivo fuente, esto permite previsualizar el documento desde cualquier entorno sin perder el formato, lo resulta extremadamente útil para el desarrollo de documentos como el que estas leyendo.

En mi caso he utilizado como editor de texto TexMaker, ya que me gustaba la disposición que tenia su interfaz y me facilitaba mucho su uso.

Version: 5.0.3
Licencia Open Source
PaginaWeb: https://www.xm1math.net/texmaker/download.html

La distribución de Latex utilizada a sido miktex, ya que esta contiene un gran número de paquetes tipográficos y es fácil de instalar:
Version: 2.9.6
Licencia Open Source
PaginaWeb: https://miktex.org/download

4.2. Herramientas de Gestión

Kanban

Para mantener un flujo de trabajo en el desarrollo del proyecto se decidió utilizar la metodología Kanban[8] para así mantener un progreso en todo momento.

La metodología Kanban se basa en hacer visible el flujo de trabajo a trabes de una tabla. La tabla Kanban se puede dividir en diferentes filas y columnas, las filas sirven para identificar diferentes tipos de actividades y las columnas para identificar cada paso por un proceso.

Para utilizar esta metodología se utilizó la aplicación web llamada trello. Ya que consta de varias herramientas interesantes, como por ejemplo hacer descripciones y comentarios en cada tarea y añadir elementos internos, para segmentar los pasos de una tarea.

Link: https://trello.com/

En el caso de la tabla que se ha realizado para este proyecto solo se ha utilizado una fila, ya que al solo tener 2 actores en el proyecto (Mi tutor y Yo), identificar el tipo de actividades no se creyó necesario. Por otra parte se implementaron 4 columnas:

- Por hacer.
- En proceso.
- Revisión.
- Hecho.

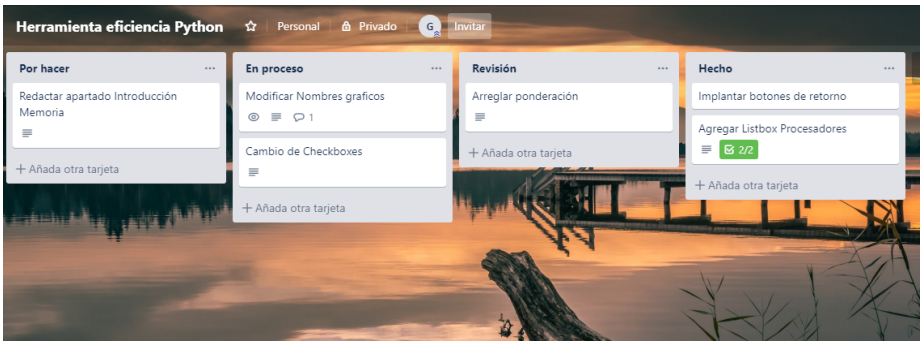


Figura 4.10: Imagen de la tabla kanban en mitad del desarrollo

Con esta tabla el sistema de trabajo ha sido el siguiente:

Primero se hacia una reunión con una frecuencia de una o dos semanas, aquí se planteaba que tareas había que hacer. Estas tareas eran las que se colocaban en la columna ”Por hacer”, una vez hecho esto las tareas que se empezaban a desarrollar pasaban a ”En Proceso”, con un limite de 2 tareas simultaneas en esa columna, no se pueden estar desarrollando mas de dos tareas a la vez. Esto se decidió así para que no se concentrase tanto el flujo en ese proceso. Una vez una tarea se acababa su desarrollo pasaba a Revisión, lo que significaba que hasta que en la siguiente reunión no se le diese el visto bueno por parte del tutor, no pasaba a Terminado.

GitHub

Se trata de una plataforma online donde la gente puede almacenar y gestionar los proyectos que estén desarrollando mediante gestión de versiones.

Ha sido la herramienta de gestión principal elegida debido a que es la que nos han enseñado su uso en la Universidad y por lo tanto con la que más practica se tenía de antemano, además de ser gratuita para proyectos de código abierto.

Pagina web: https://github.com/

La hemos utilizado para albergar el código del proyecto, y desde ahí poder gestionar el avance del desarrollo.

Link Repositorio: https://github.com/Guillecal/TFG-Herramienta_para_medir_la_eficiencia_de_codigo_python

Se han ido añadiendo los commits pertinentes según se han ido completando las tareas planificadas.

GitHub Desktop

Esta es la herramienta utilizada para gestionar de mejor manera el repositorio donde se encuentra el trabajo, ya que facilita mucho el poder realizar los commits, con sus respectivos comentarios e incluso dejar gestionar su contenido.

Pagina web: https://desktop.github.com/

Una vez descargado e instalado, la primera vez que lo ejecutamos, podemos clonar desde aquí el proyecto desde el repositorio.

4.3. Herramientas de desarrollo

Python

Se a utilizado este lenguaje de programación para realizar el trabajo, esto se debe a las gran cantidad de librerías que dispone este lenguaje para poder operar con una gran variedad de componentes, lo que ya de por si nos ofrece mucha flexibilidad. También hay que destacar que se eligió este lenguaje por ser un tipo de lenguaje interpretado, lo cual hacía que fuese mas fácil tratar el tema del interprete. La versión con la que se ha contado a lo largo del proyecto ha sido la 2.7.16. Se intentó utilizar un versión 3.7 y 3.6, pero tras tener varios problemas de consistencia entre este y el interprete que elegimos para la realización del proyecto, al final esto nos forzó a tener que probar una versión 2.

Version:2.7.16
Licencia: Open Source
PaginaWeb: https://www.python.org/downloads/

TK

Es una librería de Python para el desarrollo de interfaces gráficas. En principio se barajó la posibilidad de utilizar algún otro tipo de librería para realizar la interfaz como por ejemplo PyQt5, pero al tener dificultades de implementar otras librerías la versión de Python 2.7.16, y no tener el modulo Tkinter[9] una gran dificultad de aprendizaje para poder realizar una interfaz básica.

Tkinter

Es un modulo de esta librería que se ha utilizado en esta practica para generar el entono de la interfaz. Version: 8.5

Se utilizo un modulo del Tkinter ttk[10]

MatPlotLib

Es una librería diseñada para representar datos en gráficos 2D. Es una librería clave en el proyecto para la representación de los gráficos resultantes de los análisis.

Version: 2.2.4
Licencia: Open Source
PaginaWeb: https://matplotlib.org/downloads.html

Microsoft Excel

Es una aplicación de hoja de cálculos, utilizada mayoritariamente para tareas financieras y de logística. En el caso de este proyecto se utilizo únicamente para dar valores a las matrices de traducción, ya que la estructura de celdas que tiene esta aplicación facilitaba mucho hacer esta tarea.

Version: 1905, Office 2016
Licencia: Comercial, Propietario
PaginaWeb: https://products.office.com/es-es/try

Csv

Modulo de Python que permite abrir,leer y escribir en ficheros .csv. En el caso de este trabajo ha sido utilizado para poder leer los procesadores teóricos y poder sacar los valores deseados para realizar la ponderación.

os

Se trata de un modulo de Python, que permite acceder a funciones que permite leer y escribir archivos y acceder al sistema de archivos. En el caso de este proyecto se ha utilizado para acceder al sistema de archivos y poder recoger las rutas de algunas carpetas de forma automática.

Dis

Un modulo de Python que desensambla el código origen y lo transforma a ByteCode descomponiendo cada parte del código original. Este modulo es vital para el funcionamiento del ByteRun ya que es el encargado de transformar el código de alto nivel a ByteRun

4.4. IDE

Como entorno de desarrollo se podría se puede utilizar el IDE de Python, pero para el desarrollo de la herramienta se ha utilizado Spyder.

Spyder

Spyder es un entorno de desarrollo diseñado para desarrollar código Python, el cual cuenta con un terminal donde se pueden probar los códigos y ya incluye algunas de las librerías mas utilizadas de Python

Link descaga: https://www.spyder-ide.org/

Una vez descargado e instalado este IDE ya viene listo para poder empezar a trabajar, pero en caso de que se requiera, se pueden cambiar algunas configuraciones según el gusto de cada uno desde la pestaña Herramientas, en el apartado preferencias.

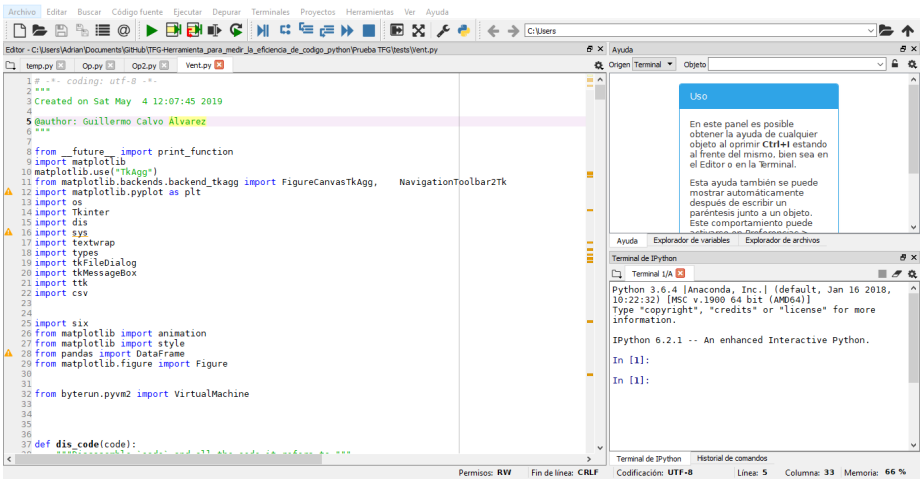


Figura 4.11: Imagen de la interfaz de Spyder

Esta es una recomendación, si se está acostumbrado utilizar otro tipo de IDE que sirva para el lenguaje Python no habría ningún problema.

Aspectos relevantes del desarrollo del proyecto

5.1. Conceptos usados aprendidos en la Universidad

Programación en lenguaje Python

Python es un lenguaje que ha sido utilizado en diversas asignaturas a lo largo de la carrera como por ejemplo Algoritmia, Sistemas Inteligentes, Gestión de la Información o Computación Neuronal y Evolutiva. En cada una de estas asignaturas han sido enseñadas diferentes librerías como por ejemplo Matplotlib la cual es fundamental para mostrar los resultados de este trabajo.

Se agradecen también las horas de practicas de estas asignaturas las cuales han sido muy importantes a la hora de conseguir cierta soltura para programar en este lenguaje.

Conocimientos sobre intérpretes y compiladores

La forma en que un compilador trata la información para transformarla y el uso que tiene el intérprete ha sido un tema central que se nos ha enseñado en la asignatura de Procesadores del lenguaje.

Programación eficiente

En asignaturas como Programación y Metodología de la programación, se han enseñado maneras de llevar a cabo una programación que se han

llevado a cabo en este desarrollo.

Gestión del desarrollo

Hay también que nombrar a la asignatura Gestión de Proyectos, donde se aprendió sobre la metodología Kanban utilizada en este proyecto y también sobre como utilizar la aplicación Github.

5.2. Transcurso del desarrollo

El planteamiento inicial del proyecto era crear una herramienta para medir la eficiencia del código Python, en base a los ciclos de reloj y el espacio en memoria que este ocupa. Al ser este un planteamiento previo al inicio del desarrollo, este ha ido dando algunos cambios según nuevos requisitos han ido surgiendo o restricciones con las que no habíamos contado han ido apareciendo. En este apartado se intentara contar como han surgido esos cambios inesperados y como han afectado este al desarrollo del proyecto.

El intérprete

En primer lugar, la base por donde se partió el proyecto fue el intérprete. Se buscó un intérprete capaz de satisfacer las necesidades que se plantearon. En esta etapa surgieron varias propuestas, pero tras algunas puestas en común, se llegó a la conclusión de que el intérprete ByteRun era la mejor idea para hacer un primer intento.

Requisitos que cumple

ByteRun al final fue el intérprete elegido porque nos proporcionaba una serie de requisitos que encajaban a la perfección con lo que teníamos planeado previamente. He aquí alguno de esos requisitos:

- Estaba desarrollado en Python: Al ser Python un lenguaje conocido y que ha sido impartido en la Universidad, era preferible que estuviese hecho en un lenguaje que ya tuviésemos conocimientos previos. Además que un intérprete de Python hecho en Python nos parecía que tenía un cierto atractivo.

- Los autores de este interprete contaban con una buena documentación sobre su funcionamiento[11], lo cual nos venía como anillo al dedo, para así entender mejor su funcionamiento y poder hacer mejor el upgrade a este.
- Es un intérprete hecho con la finalidad del aprendizaje, esto supone que no sea el más rápido, pero también conlleva el hecho de que no sea muy complejo. Y como el objetivo del proyecto no es tener la herramienta más rápida, si no que funcione y seamos capaces de implementar los cambios necesarios, esto para nosotros era visto como una gran ventaja.

Una vez se tenía claro que el ByteRun era el intérprete que haría de base para desarrollar la herramienta, lo primero era entender su funcionamiento y hacer pruebas con diferentes ficheros para comprobar que de verdad funcionaba.

Pruebas

En este punto nos encontramos con los primeros imprevistos, los autores del intérprete ya contaban con unos tests hechos, pero para poder ejecutar estos test utilizaban una herramienta para Python llamada tox.

Tox en algunos tipos de versiones de Python no funciona adecuadamente, tras informarnos sobre esta, y hacer pruebas en diversas versiones de python, al final decidimos implementarlo en Python 2.7.16 la última de las versiones 2 de Python actualmente.

Una vez solucionado esto se probaron diferentes tests que estaban ya creados, los cuales funcionaban correctamente. Pero para poder medir los límites del interprete se generaron otra serie de tests con los cuales se pudo demostrar alguna de las limitaciones del interprete.

Upgrade

Seguido de las pruebas se pasó a plantear la modificación del intérprete. Al principio como la intención del proyecto era poder medir la eficiencia a través de las operaciones y de las direcciones de memoria, nos pusimos a

investigar en que parte del intérprete conseguiríamos la información necesaria para esto, y la conclusión a la que llegamos no fue la esperada. Esto se debió a que al ser el ByteRun un intérprete de Pila, este no guarda la información que va procesando directamente en direcciones de memoria, si no que las guarda en su sistema de pilas interno. Por culpa de esto nos vimos obligados a cambiar el planteamiento principal y centrarnos en la eficiencia a través de las operaciones y diferentes formas sacar resultados a partir de esto.

Una vez centrados en sacar las operaciones del interprete, nos topamos con algo que esperábamos. El interprete para saber como actuar según el tipo de instrucción que fuera a ejecutar distinguía entre diferentes tipos de instrucción, y para las que nosotros llamaríamos operaciones, tenia un par de listas con los nombres de estas según el ByteCode. Con esta diferenciación, nosotros tendríamos que guardar una instrucción cuando esta se encuentra en alguna de estas listas.

El proceso de guardado de matrices pasó por varias etapas para al final conseguir, el que es nuestro criterio, el método mas óptimo de entre todos ellos.

En la primera idea implementada solo se crearon unas variables por cada tipo de operación disponible, esto a parte de ser poco eficiente, limitaba mucho el tipo de operaciones a analizar, ya que no se tenían en cuenta los tipos de los operadores.

La segunda idea implementada fue la de la matriz de operaciones: se decidió guardar las operaciones en una matriz con la misma estructura que posteriormente se aplicaría a la matriz de traducción, como se puede ver en la tabla 3.3.

Esta idea ya daba una mayor variedad de operaciones, ya que si tenía en cuenta los tipos de los operadores, pero por otra parte era incluso menos eficiente que cuando solo había variables. La matriz hacía tener un gran numero de posiciones en la matriz sin ningún tipo de uso, ya que la matriz tenia unas 350 posiciones de las cuales por fichero se solían llenar entre 6 y 12. Esto equivalía un porcentaje muy grande de desuso, por lo que se decidió cambiar este método por otro.

Por último, para mejorar la eficiencia de la matriz de operaciones se hizo una implementación en un diccionario. Este diccionario guarda un contador por cada operación diferente, teniendo en cuenta los operadores. Y de esta manera nos ahorramos las celdas vacías que nos surgían en la matriz.

Operación	Contador
ADD int int	100
ADD int flo	50
MULTIPLY int int	60
MULTIPLY str int	150
DIVIDE int int	200

Tabla 5.6: Ejemplo de la estructura básica del diccionario.

El diccionario esta compuesto por una lista de clave primaria, y los componentes de esta lista son los siguiente:

- Nombre de la operación detectada por la interfaz
- Primeras 3 Iniciales del tipo del primer operador
- Primeras 3 Iniciales del tipo del segundo operador

Esta estructura para las keys del diccionario se basa en el hecho de que un operación necesita de dos valores con los que operar, y dependiendo de el tipo de estos valores, dos mismas operaciones con diferentes valores, pueden llegar a tener nivel de eficiencia mucho mas bajo o alto que el otro.

Según el ByteRun va analizando todo el código cuando detecta una instrucción dentro de la lista de operaciones este llama a una función, que en caso de no tener ninguna entrada de esa operación y sus respectivos operadores la genera y pone de valor un 1. Posteriormente si se va encontrando mas operaciones iguales en vez de crear otra entrada en el diccionario de operaciones, incrementa en 1 el valor enlazado con la clave que la corresponde.

Análisis

Tras implementar esto empezamos a plantearnos los distintos tipos de análisis que se harían con los datos obtenidos y la conclusión a la que llegamos fue el de hacer un análisis individual en el que solo mostrásemos los porcentajes de eficiencia de un fichero dado y otro en el pudiésemos comparar la eficacia entre varios varios ficheros.

Primero empezamos por el análisis individual, y en este caso como el controlador y la vista están unidas, se podría decir que era un análisis en paralelo, según se hacia esa parte del controlador, se implementaban los parámetros de la interfaz.

La interfaz también pasó por una serie de versiones: Al principio la interfaz tan solo se componía de una ventana donde se añadían elementos y a veces se superponían, literalmente era un caos. Ademas de esto para indicar que fichero se quería analizar había que pasar la dirección del directorio.

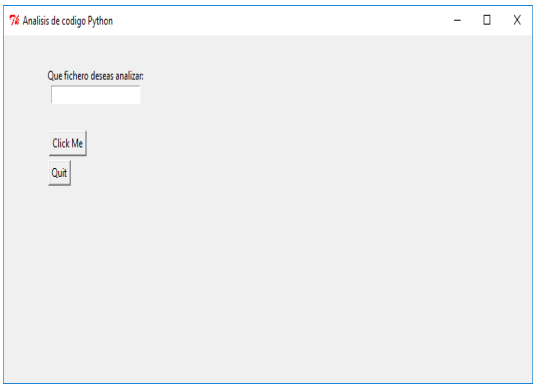


Figura 5.12: Ventana principal V.1 Interfaz

Para evitar que los elementos colisionasen entre ellos , se pasó a otro método donde cada vez que se se activasen algunos botones en particular, se añadiese una nueva ventana, esto evitaba el problema de las colisiones, pero por otra parte la final de los análisis era muy molesto, porque podías terminar con 4 ventanas diferentes abiertas, lo que al final era muy lioso.

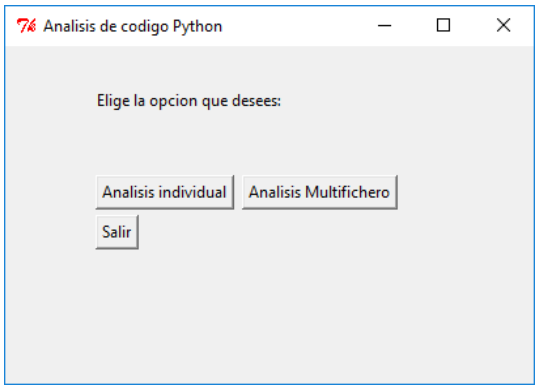


Figura 5.13: Ventana principal V.2 Interfaz

Para evitar los problemas tanto de la primera versión de la interfaz como de la segunda, se investigó mas el funcionamiento de la biblioteca Tkinter y sus diferentes funcionalidades, y de esta manera se consiguió realizar una interfaz que utilizase solo una ventana[12], pero el contenido de esta se fuese cambiando según interactuemos con esta. A decir verdad la interfaz por dentro crea un numero de marcos, por así llamarlos, donde ya están todos los elementos que se tiene que mostrar en pantalla, cada marco representa un tipo distinto de ventana que puede surgir según se desarrolle la interacción con la herramienta, lo único que solo se muestran por pantalla aquel marco que corresponda con la funcionalidad que se este mostrando en ese momento.

Fotos

En concreto esta herramienta cuenta con un total de 9 Marcos que son los siguientes:

- VentanaPrincipal
- VentanaIndividual
- VentanaAnalisis
- VentanaMultiple
- VentanaComparacion
- VentanaComparacion2

- VentanaOperaciones
- VentanaOperaciones2
- VentanaOperaciones3

Es una manera muy útil de hacer una interfaz mucho mas fluida y sencilla para la vista. La única pega que se le podría poner a este método, es que al iniciar la aplicación tiene que generar los marcos para ir rotando entre ellos. Esto ha condicionado más de una vez algunas implementaciones hechas en la interfaz ya que a veces quedan huecos innecesarios.

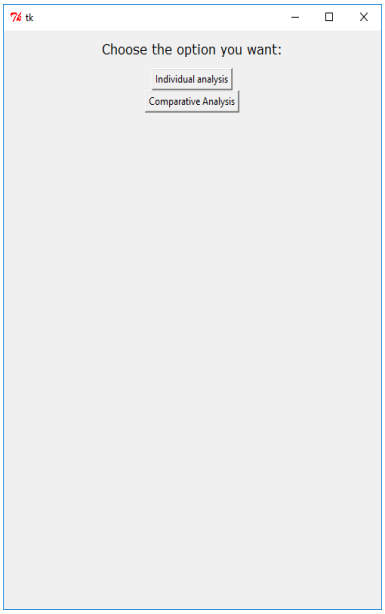


Figura 5.14: Ventana principal V.3 desconfigurada

Para acabar con el desarrollo de las funcionalidades principales se crearon los llamados procesadores teóricos a través de la matriz de traducción. Como ya se ha explicado en puntos anteriores los procesadores contienen una matriz de traducción que contiene valores que dan un nivel de eficiencia a los distintos tipos de operaciones. Para guardar la matriz se decidió utilizar el formato csv. Hay un par de motivos detrás de esta elección:

1. La simplicidad del formato hace posible poder configurarlo con un gran número de aplicaciones, aunque en el caso de este proyecto se optó por Microsoft Excel.

2. Hay un modulo de Python en el que ya teníamos algo de experiencia, llamado también csv, que permitía leer y escribir en los tipos de archivos .csv.

Artículo

Una vez terminado el desarrollo de las funcionalidades principales se le dieron unas pinceladas a algunos aspectos de este como proporciones de algunos elementos en la interfaz, cambios en las nomenclaturas de variables, etc. Y mientras esto ocurría empezamos a desarrollar un artículo sobre el propio proyecto, para exponerlo en el congreso europeo, EuroSim 2019. Pensamos que sería una oportunidad muy buena de ver si el trabajo hecho hasta el momento era visto atractivo y útil por otras personas interesadas en el sector informático y de esta manera también poder aprender del feedback que nos transmitieran.

Retoques

Una vez terminado el artículo se pasó a una fase final de depuración donde se hicieron algunos cambios de diseño o alguna pequeña funcionalidad de última hora. Como por ejemplo cambiar la forma en la que se mostraban los checkboxes[13][14] o la parametrización a través del nombre de algunos ficheros.

5.3. Estructura interna

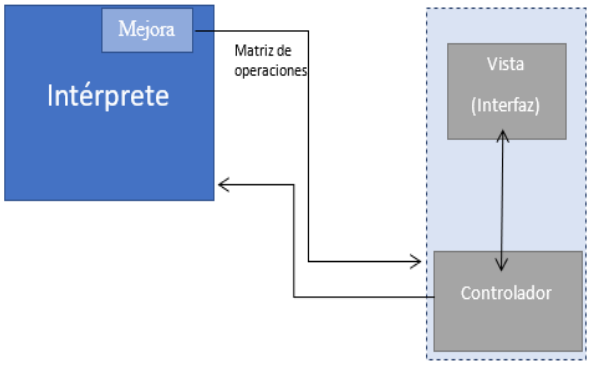


Figura 5.15: Estructura interna de la herramienta

La estructura interna del programa se muestra en la figura 4. Como se puede observar en la figura 4, es un modelo vista controlador . En este caso, el controlador y la vista se implementan de tal manera que comparten algunos componentes, por lo tanto, comparten un cuadro en la imagen, pero aún así, el concepto del modelo es el típico en el que el controlador actúa como un intermediario. Esta unión surgió del desarrollo, se fueron haciendo partes del controlador junto con la interfaz.

En la herramienta, el controlador actúa como un intermediario entre la interfaz gráfica y el interprete. La interfaz permite interactuar y mostrar los resultados del análisis de eficiencia, mientras que el intérprete, el ByteRun en este caso, se ocupa de analizar y ejecutar el código.

Trabajos relacionados

El tema tratado en este trabajo ha sido tratado desde ya hace bastante tiempo, ya que la eficiencia un campo que lleva teniendo importancia ya años atrás, a la par de que es posible abordarla desde un gran numero de puntos de vista diferentes.

Aquí se hará mención a un par de trabajos o proyectos que tengan que ver con mas con la eficiencia en el campo de la informática y se consideren interesantes para combinar y ampliar los conocimientos expuestos en este trabajo.

Vamos a mencionar un articulo que explica diferentes formas de medir la eficiencia de código para código java de android, este es un trabajo que es muy interesante que puede servir para entender otro tipo de medición y enfocado para un entorno mas específico como son las aplicaciones móviles[15].

Otro proyecto que destacamos es uno en el que se ha creado una herramienta para Chrome capaz de medir la eficiencia de aplicaciones Rect. Destacamos este proyecto por la gran cantidad de opciones que posee y algunas funcionalidades, como por ejemplo el hecho de que puede enseñar información en tiempo real según la va analizando[16].

A parte de lo mencionado anteriormente también se han encontrado algunos proyectos más que tratan la medición de eficiencia en diferentes lenguajes y pueden ser interesantes:

- https://github.com/DaKnOb/Zipper.
- https://github.com/MobileComputing2016/EfficiencyMeasurement

Conclusiones y Líneas de trabajo futuras

Para terminar se expondrán las conclusiones obtenidas tal el desarrollo de todo este proyecto y las posibilidades que deja abierta para desarrollar mas cosas en el futuro:

7.1. Conclusiones

- Se consiguieron realizar los objetivos principales propuestos para este proyecto. Entre estos están la implementación de una forma de medir la eficiencia de una manera flexible, con el propósito de que sea útil en un mayor numero de casos de usos.

- La labor de comprensión realizada para entender un código hecho por otras personas, en este caso concreto para entender el funcionamiento del ByteRun, fue más duradero de lo esperado ya que a pesar de no ser un interprete muy complejo tenía una serie de conceptos sobre las pilas y el Bytecode, al final fue una buena manera de aprender algo más sobre los diferentes tipos de interpretes que había y en concreto saber más sobre los interpretes de pila.

- Otro gran reto de esta aplicación fue la interfaz gráfica, ya que casi no había tenido experiencia creando una. Tanto elegir las herramientas necesarias para hacerla como el posterior aprendizaje sobre el funcionamiento de Tkinter, es otra de las cosas que me llevo aprendidas tras este proyecto Aunque no todas las ideas que se plantearon para este proyecto salieron finalmente a la luz, ha sido muy satisfactorio y

enriquecedor, ver como ideas que vas planteando se van plasmando poco a poco en algo que vas invirtiendo horas para finalmente formar aquello que estabas intentando lograr. Personalmente también me ha servido para aprender sobre la gestión que hay que llevar en un proyecto de estas características y como reaccionar ante los diferentes obstáculos que uno se va encontrando en el desarrollo del mismo.

7.2. Líneas de trabajo futuras

Este proyecto tiene un fuerte potencial para seguir siendo desarrollado, ya que a partir de este punto puede evolucionar hacia diferentes frentes:

En la herramienta se han implementado aquellos análisis que se han creído mas interesantes en una primera instancia que pueden ser usado en una ámbito mas general. Pero se pueden implementar más análisis dependiendo del objetivo que se tenga detrás, por poner un ejemplo, un análisis que estuvo a punto de ser implementado también en este proyecto fue, el análisis de operaciones individuales. Este análisis consistiría en mostrar como aumenta la cantidad de un tipo de operador en concreto cada vez que el interprete encuentra un operador. Este análisis sería una buena forma de ver en que parte de la ejecución un código se aglomeran mas tipos de operaciones. Y este es solo un ejemplo de los muchos tipos de análisis más que se pueden implementar.

Y en lo que más potencial creemos que puede derivar en un futuro esta herramienta sería el poder medir la eficiencia de códigos de casi cualquier tipo, esto sería relativamente sencillo tras haber desarrollado esta herramienta, debido a la forma interna en la que esta construida . La interfaz está separada del resto del programa y gracias a esto se podrían añadir interpretes que pudieran analizar tipos de lenguaje de programación, y al resto de la herramienta habría que hacerle unos cambios mínimos para que permitiese la entrada de ficheros con otro tipo de formato que no sea solamente .py. Esto haría que la herramienta fuese útil para casi cualquier tipo de desarrollo.

Como últimos planteamiento también se podría mirar la migración de version, de la version 2.7 de Python a alguna de las versiones 3. Algunos elementos de la interfaz podrían se mejorables como el evitar la superposición de los nombres de los tipos de operaciones detectados en el análisis individual cuando se trata de un fichero con muchos tipos.

Bibliografía

[1] Wikipedia: Eficiencia,
https://es.wikipedia.org/wiki/Eficiencia

[2] Soge: Tiempo de ejecución y eficiencia de algoritmos,
http://verso.mat.uam.es/~pablo.angulo/doc/laboratorio/b2s2.html

[3] Wikipedia: Cota superior asintótica,
https://es.wikipedia.org/wiki/Cota_superior_asint%C3%B3tica

[4] Wikipedia: Intérprete(informática),
https://es.wikipedia.org/wiki/Int%C3%A9rprete_(inform%C3%A1tica)

[5] Wikipedia: Máquina de pila,
https://es.wikipedia.org/wiki/M%C3%A1quina_de_pila

[6] Allison Kaptur: A Python Interpreter Written in Python,
https://www.aosabook.org/en/500L/a-python-interpreter-written-in-python.html

[7] Python: Disassembler for Python bytecode,
https://docs.python.org/2/library/dis.html

[8] Kanbanize: Fundamentos, que es un tablero kanban,
https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-tablero-kanban/

[9] Recursos Python: Panel de pestañas (Notebook) en Tcl/Tk (tkinter),
https://recursospython.com/guias-y-manuales/panel-de-pestanas-notebook-tkinter/

[10] Tutorialesprogramacionya: módulo ttk,
https://www.tutorialesprogramacionya.com/pythonya/detalleconcepto.php?punto=63&codigo=63&inicio=60

[11] GitHub,Ned Batchelder: Byterun,
https://github.com/nedbat/byterun

[12] Youtube,sentdex: Object Oriented Programming Crash course with Python 3,
https://www.youtube.com/watch?v=A0gaXfM1UN0

[13] Python Course: Python Tkinter Course,
https://www.python-course.eu/tkinter_checkboxes.php

[14] PerlMonks : Tk List of Checkboxes,
https://www.perlmonks.org/?node_id=172934

[15] Nugroho Satrijandi, Yani Widyani: Efficiency measurement of Java Android code,
https://ieeexplore.ieee.org/abstract/document/7062696/keywords#keywords

[16] johnnycoyle: react-rpm,
https://github.com/react-rpm/react-rpm