**DOCUMENTO DE ANALISIS**

**ESTUDIANTE 1: DIEGO ALEJANDRO GONZALEZ VARGAS**

**ESTUDIANTE 2: SEBASTIAN GUERRERO RIOS**

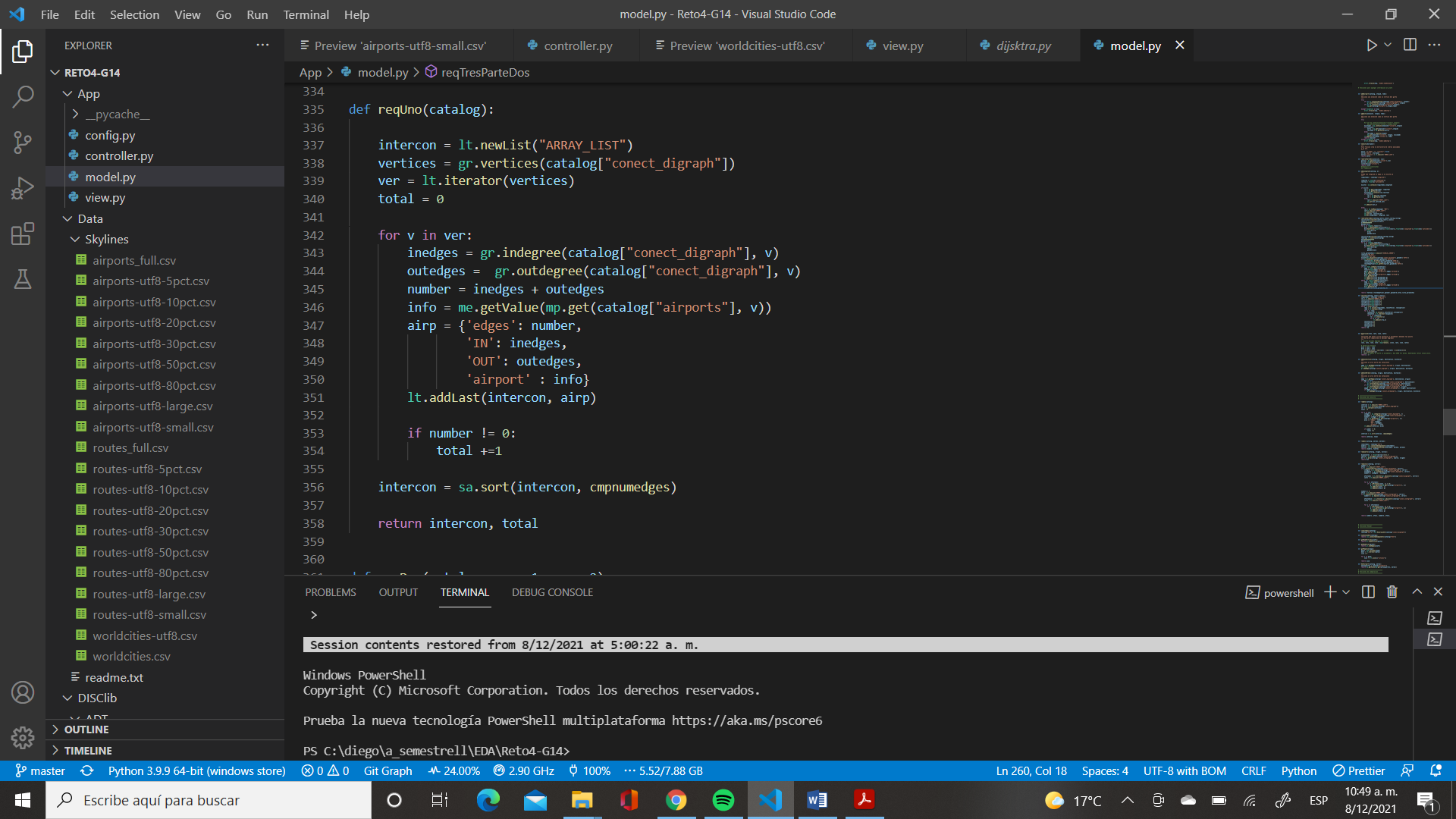
**Maquina 1:**

Procesador: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz

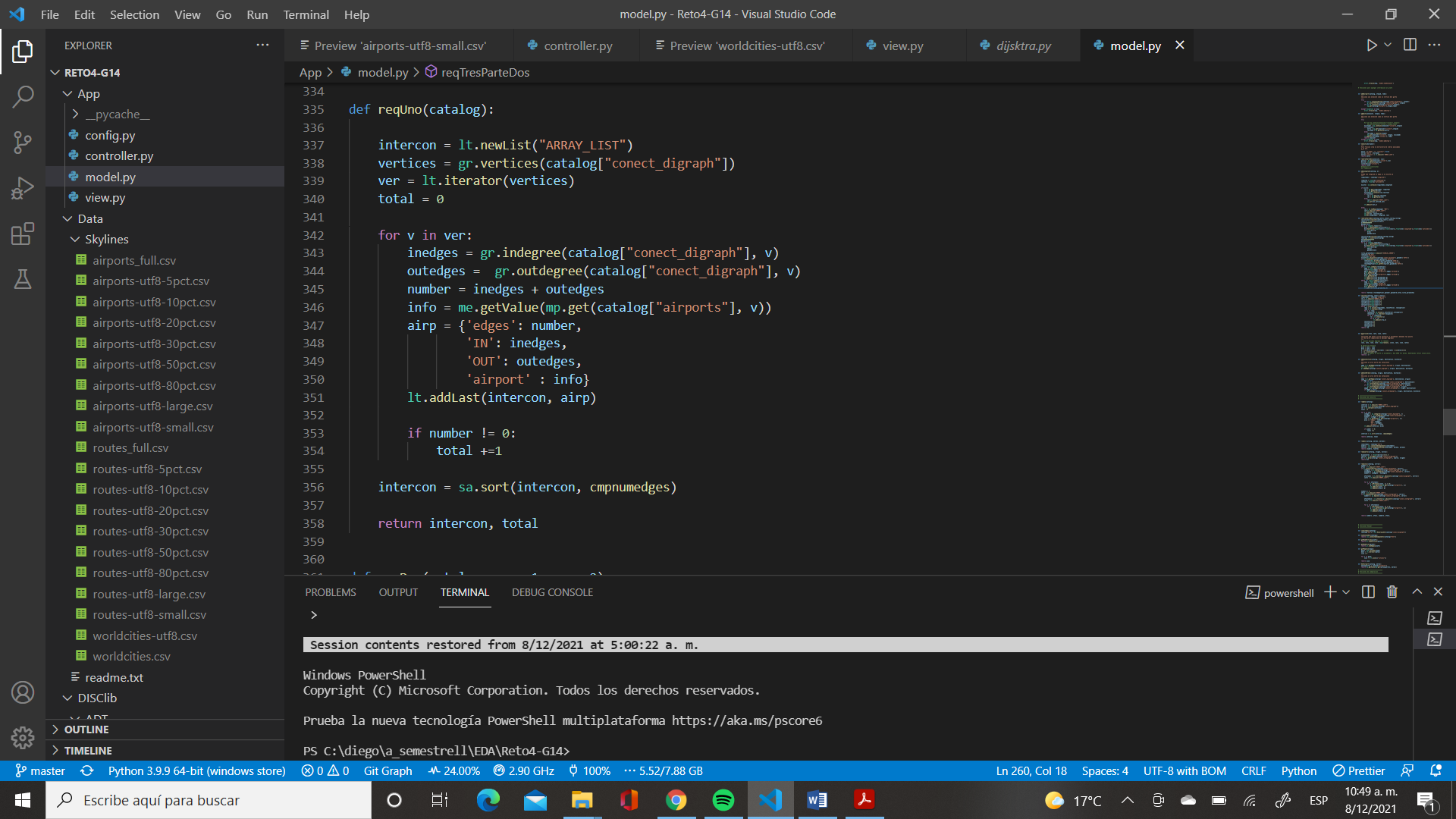
Ram: 8gb

**REQUERIMIENTO 1:**

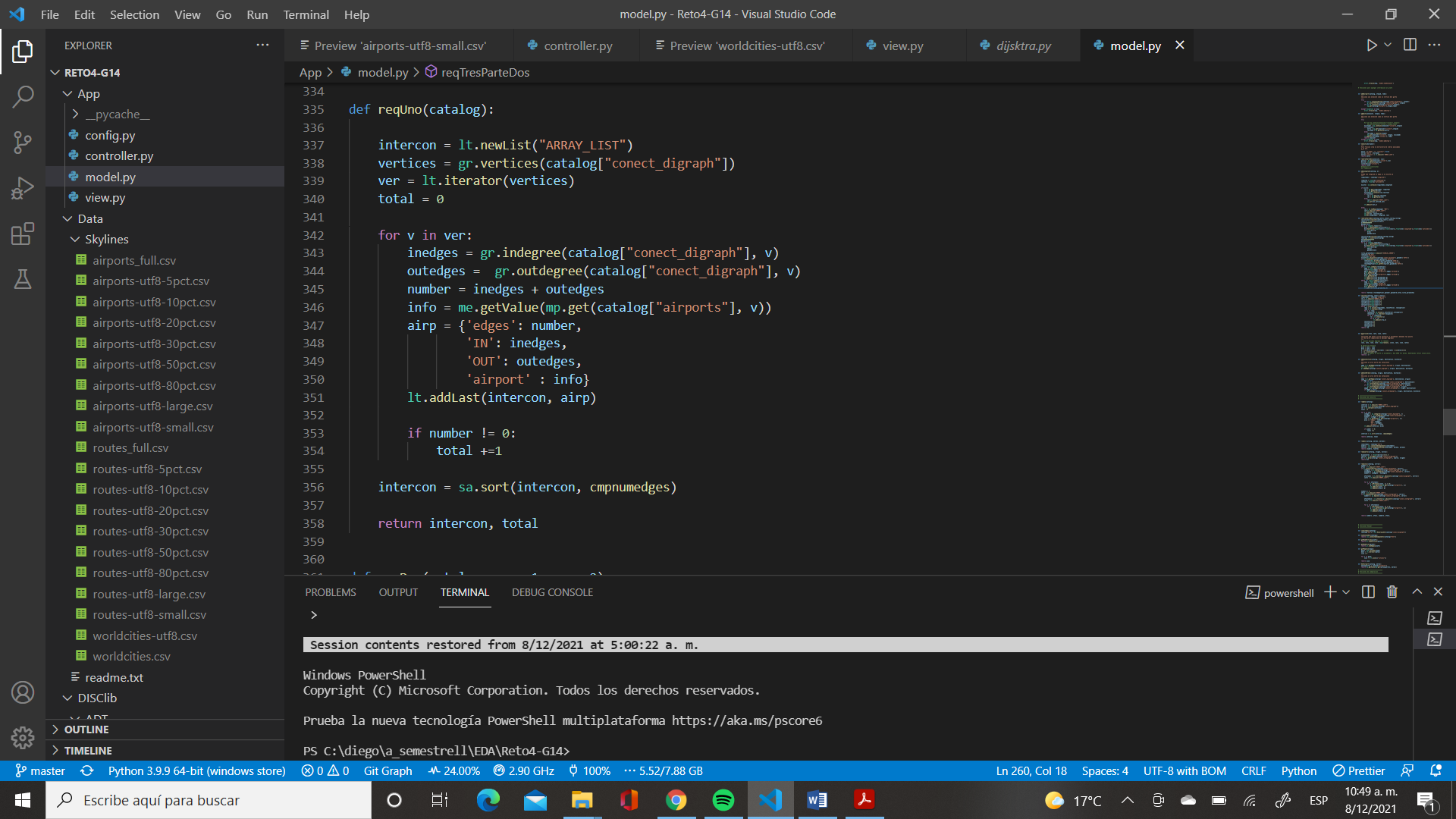
Para poder dar respuesta a la complejidad algorítmica de este requerimiento se debe de hacer un análisis de cada una de las funciones de su código, así como de la construcción de la respuesta. En este sentido, se debe tener presente que el enunciado solicitaba un ranking de los 5 aeropuertos con más conexiones del grafo creado para la sección de cargar. Así las cosas, una vez se construía los grafos tanto dirigidos como no dirigidos, se tomaba el primero mencionado (digraph), y se iniciaba la operación.



Para empezar, se solicitaba la función de vértices del API de grafo para poder tener un TAD lista con cada uno de los vértices del mismo. Posteriormente, se recorría cada uno de los elementos de la lista, y se configuraba un objeto de tipo diccionario nativo con la información exclusiva de las conexiones de cada uno de los vértices. Esta información sería posteriormente indexada a un nuevo TAD lista con toda la información de estos objetos. Esta operación tiene complejidad de o(V), donde v hace referencia a todos los vértices del grafo.



Finalmente, una vez se tiene la lista con toda la información de conexiones, se procede a organizar la misma para cumplir con el requerimiento de ranking. Para ello, se utiliza una función tradicional de ordenamiento Shell Sort. Este ordenamiento tiene una complejidad de o(V^1,25) en promedio.



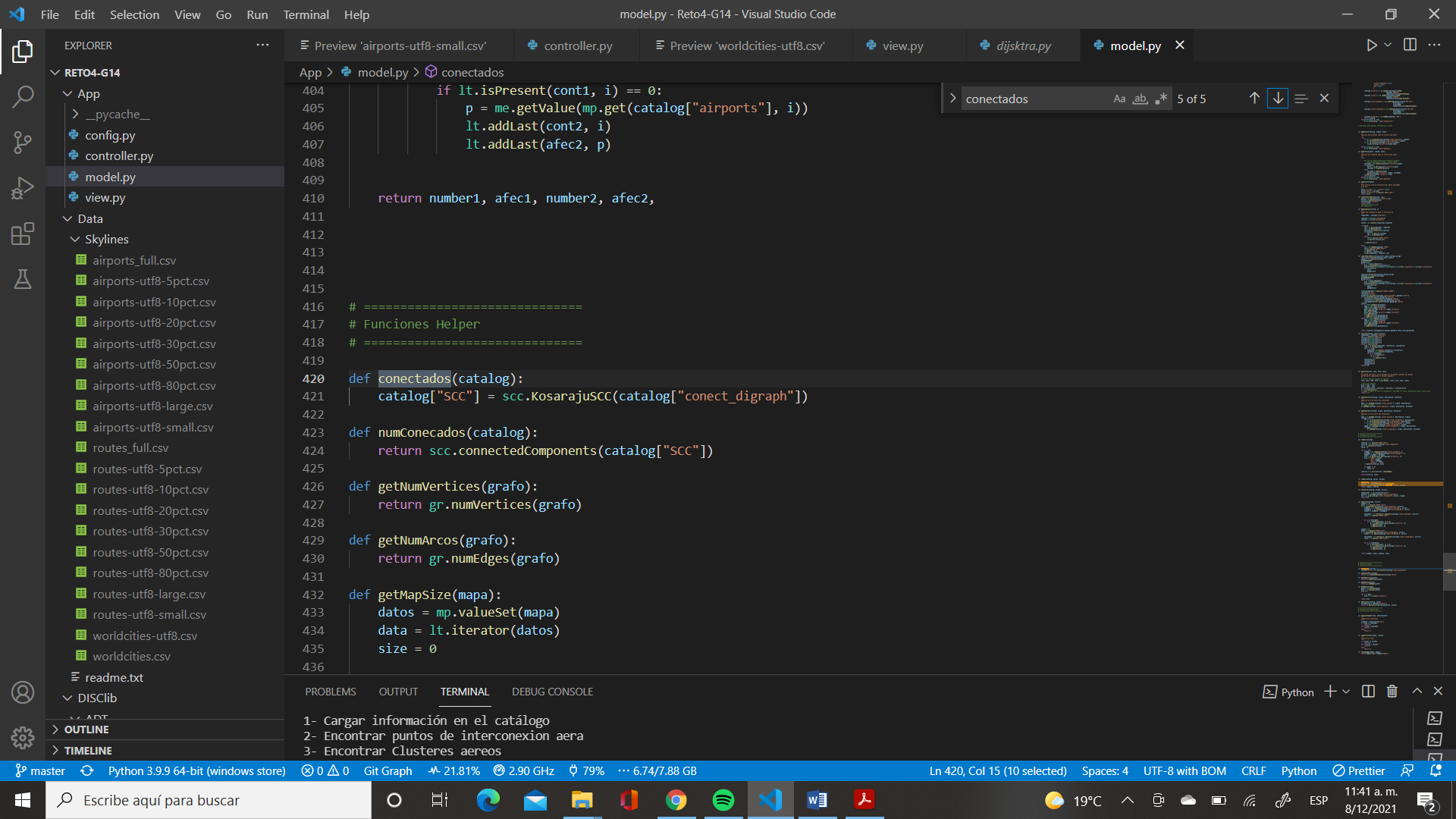
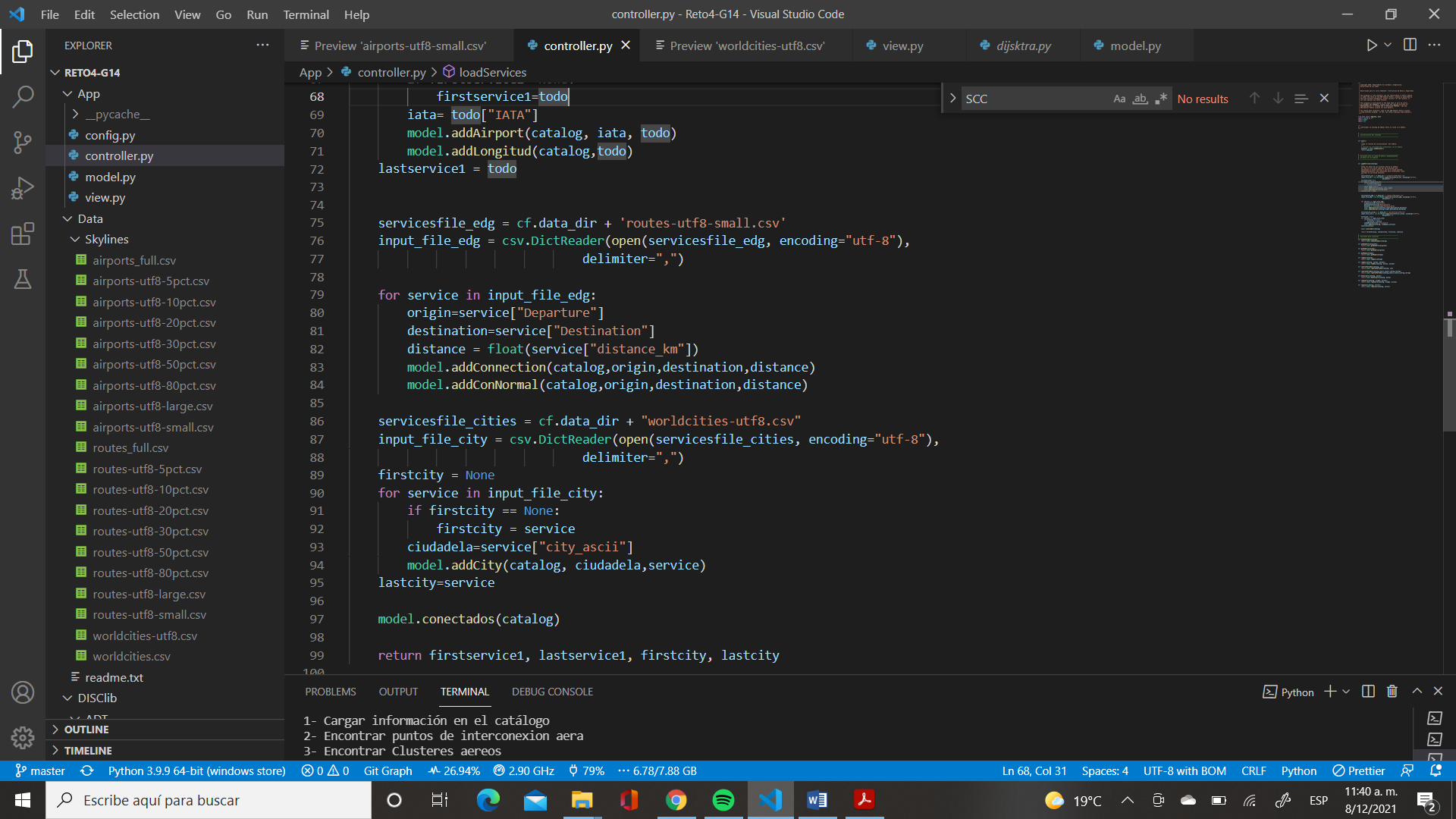
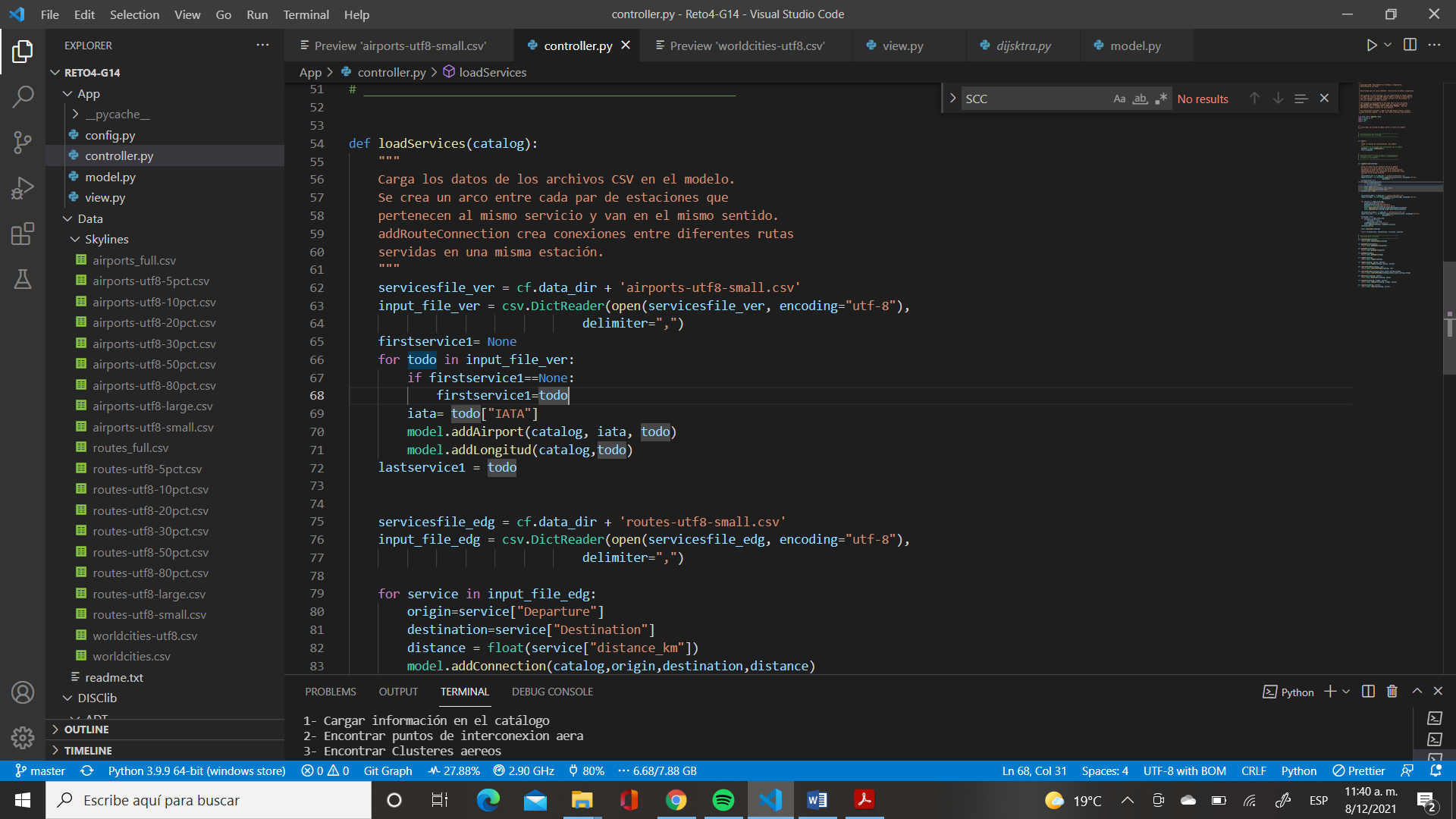
**En conclusión, se puede decir que la complejidad algoritmica es de o(V)+o(V^1,25), la cual se puede resumir y simplificar a o(V^1,25) como complejidad final**

Máquina 1:

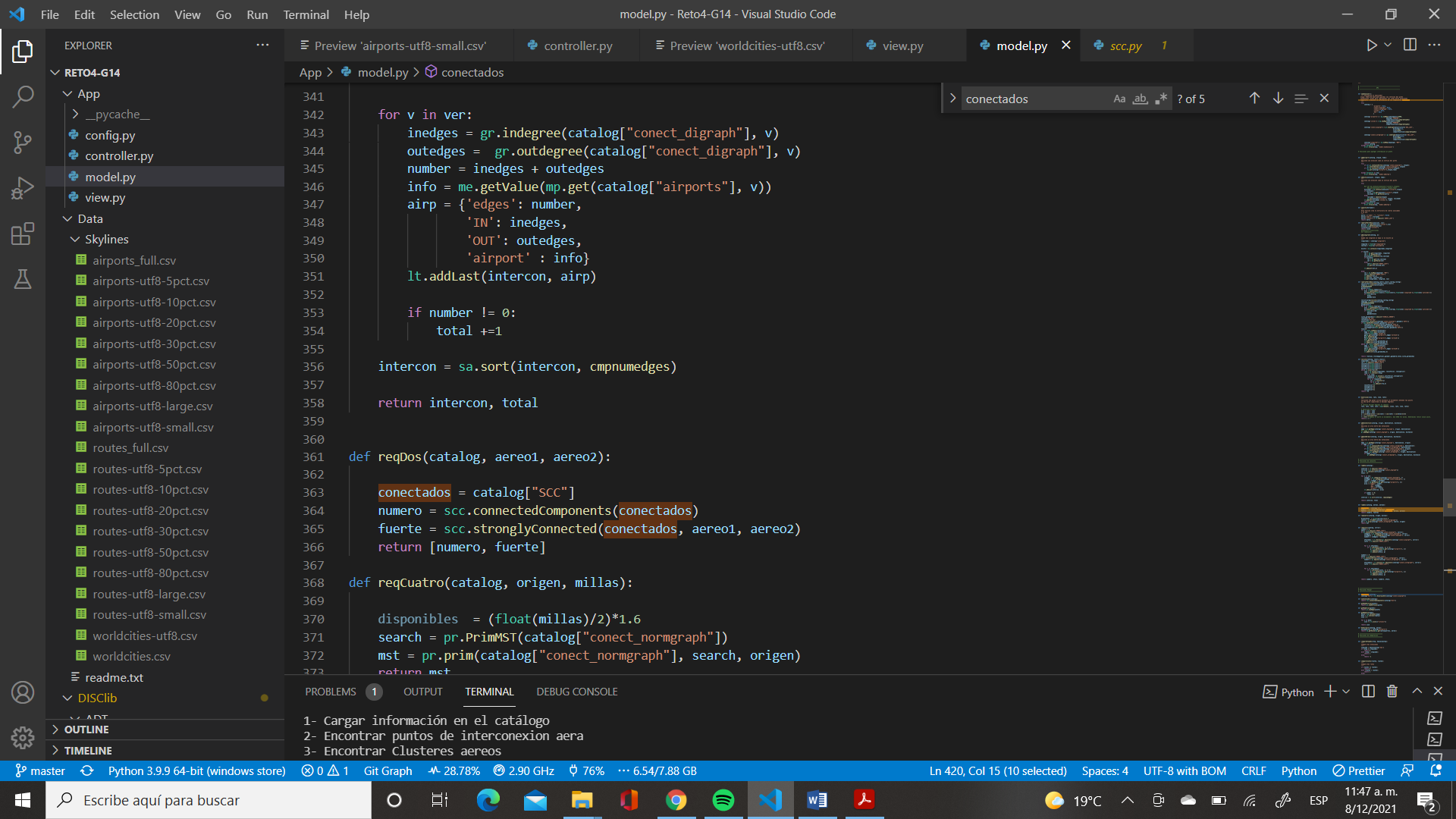
|  |  |
| --- | --- |
| ARCHIVO(%) | TIEMPO(SEG) |
| 10 | 0,0 |
| 20 | 0,0 |
| 30 | 0,0125 |
| 50% | 0,0125 |
| 80% | 0,175 |
| 100% | 0,328125 |

**Requerimiento 2:**

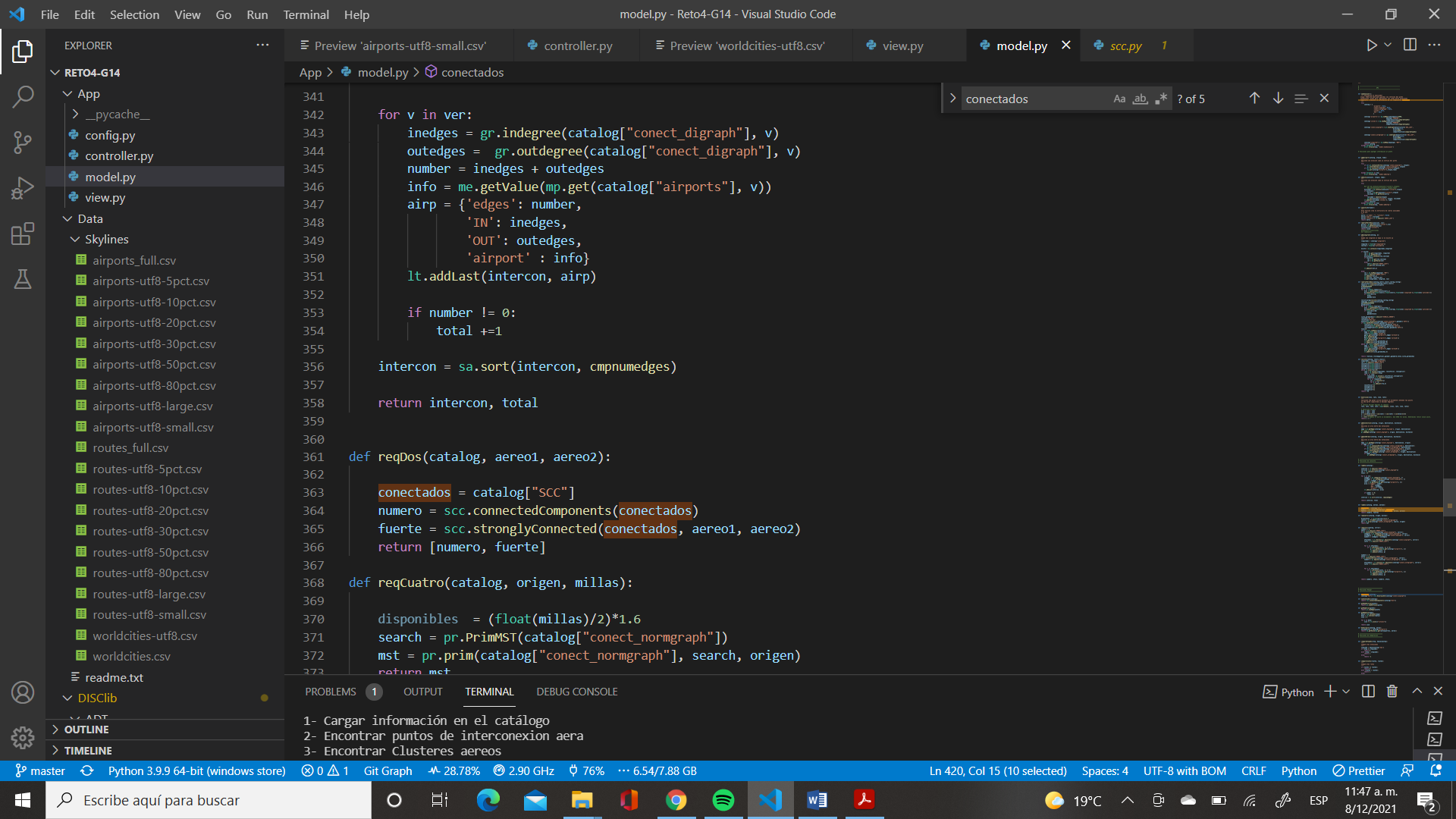
Para poder dar respuesta a la complejidad algorítmica de este requerimiento, una vez se realizará el análisis de cada una de las partes que componen la estructura de código del mismo. En este sentido, se debe mencionar que el requerimiento pedía una determinación de la cantidad de clústeres o componentes fuertemente conectados que tiene el grafo. Para ello, se debe tener en cuenta que la resolución de este algoritmo se realizó a partir de la utilización del algoritmo de Kosaraju.

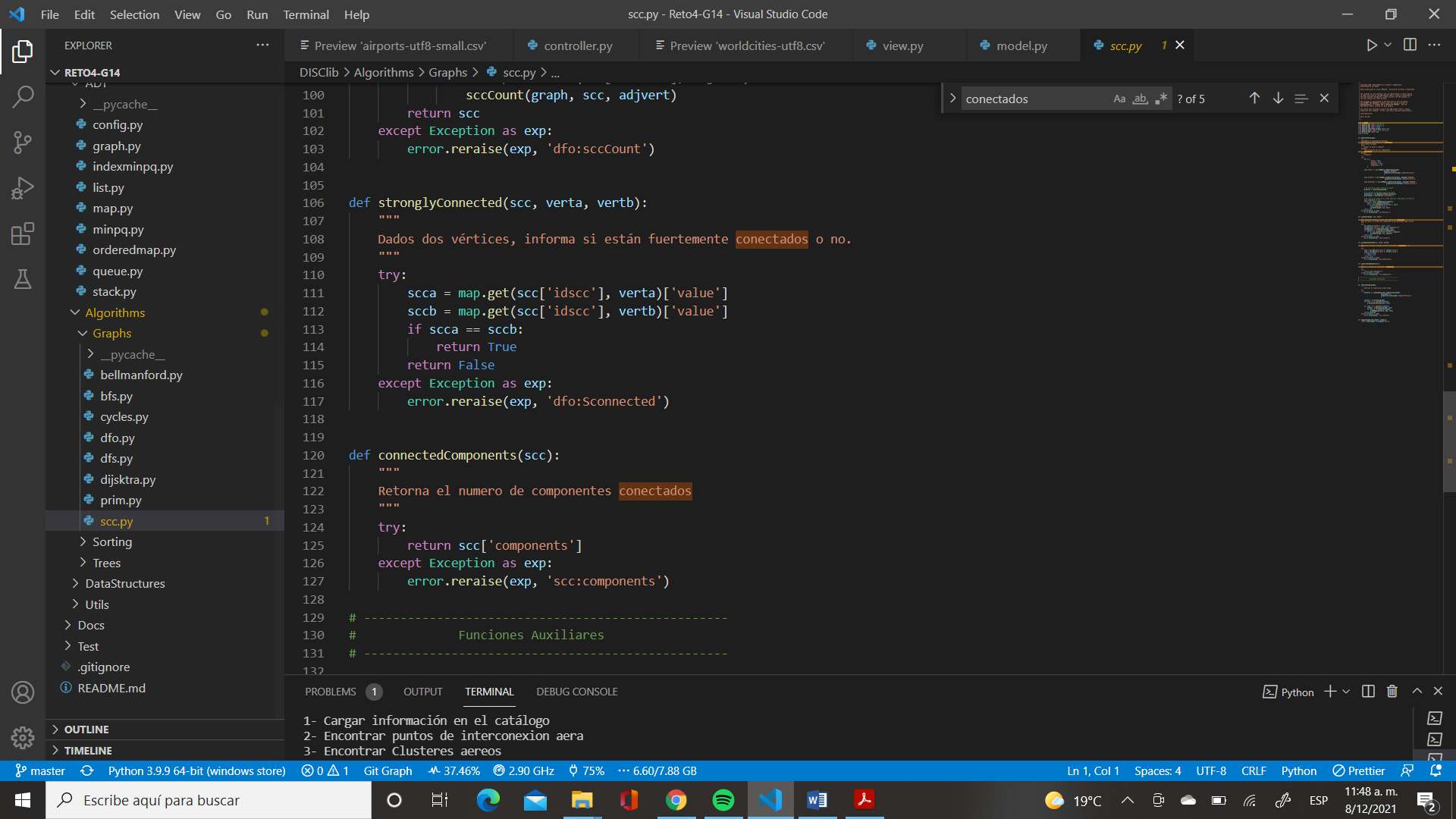


Sin embargo, este algoritmo se utiliza en la sección de cargar, por lo que a pesar de que su complejidad algorítmica es de o(E+V), esta no se toma en cuenta. Posteriormente, se procede a hacer uso de toda la información que provee el API SCC, de donde se obtiene en orden o(1) el número de componentes fuertemente conectados del gráfico.



Por otra parte, para la verificación de la conexión entre 2 aeropuertos dentro de un mismo cluster, se utiliza la función correspondiente del API, que como se puede ver hace 2 llamados a un mapa, por lo que se considera que su complejidad también será de O(1)





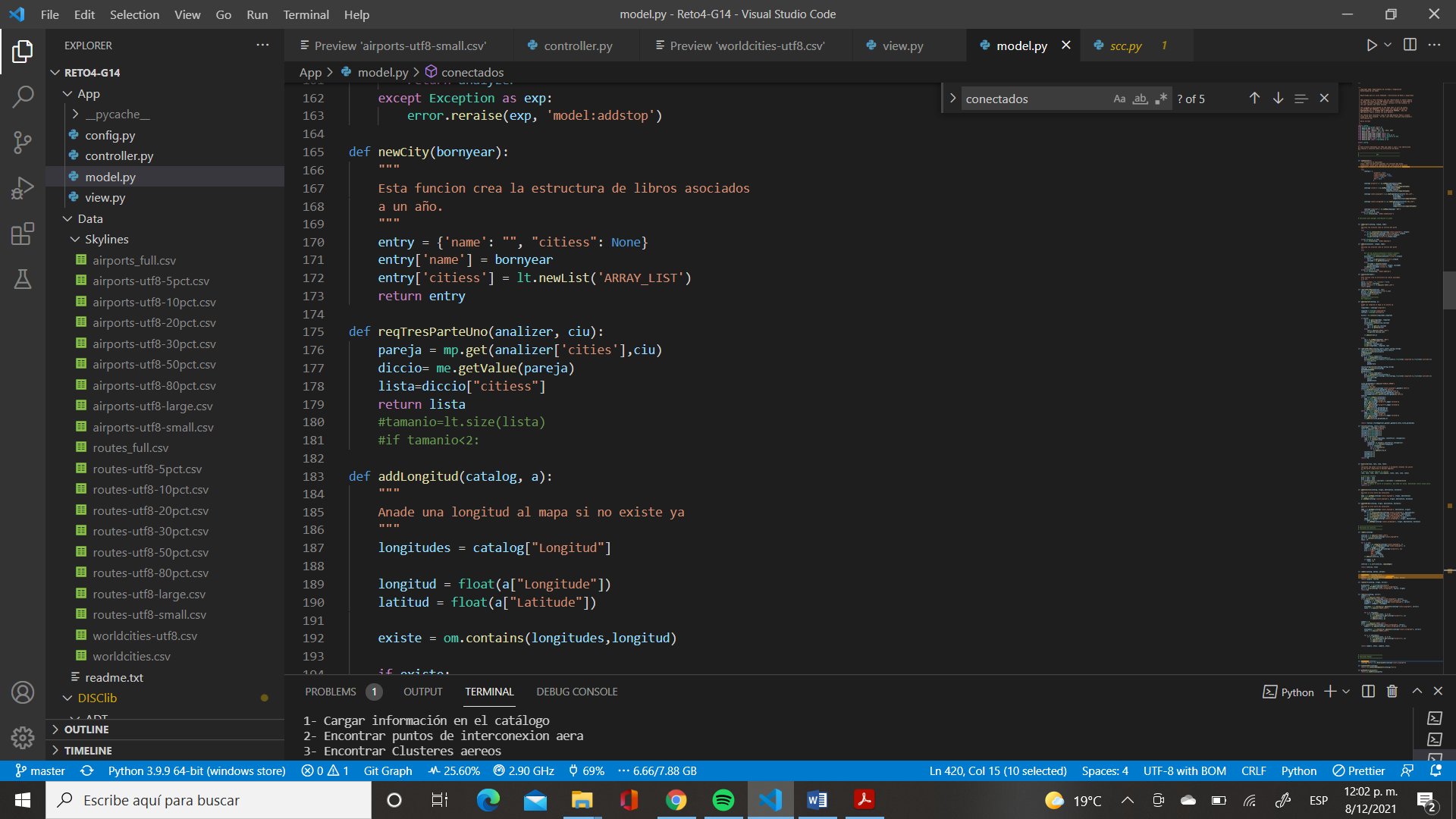
**En conclusión, se puede asegurar que la complejidad algorítmica general de este requerimiento es O(1)**

Máquina 1:

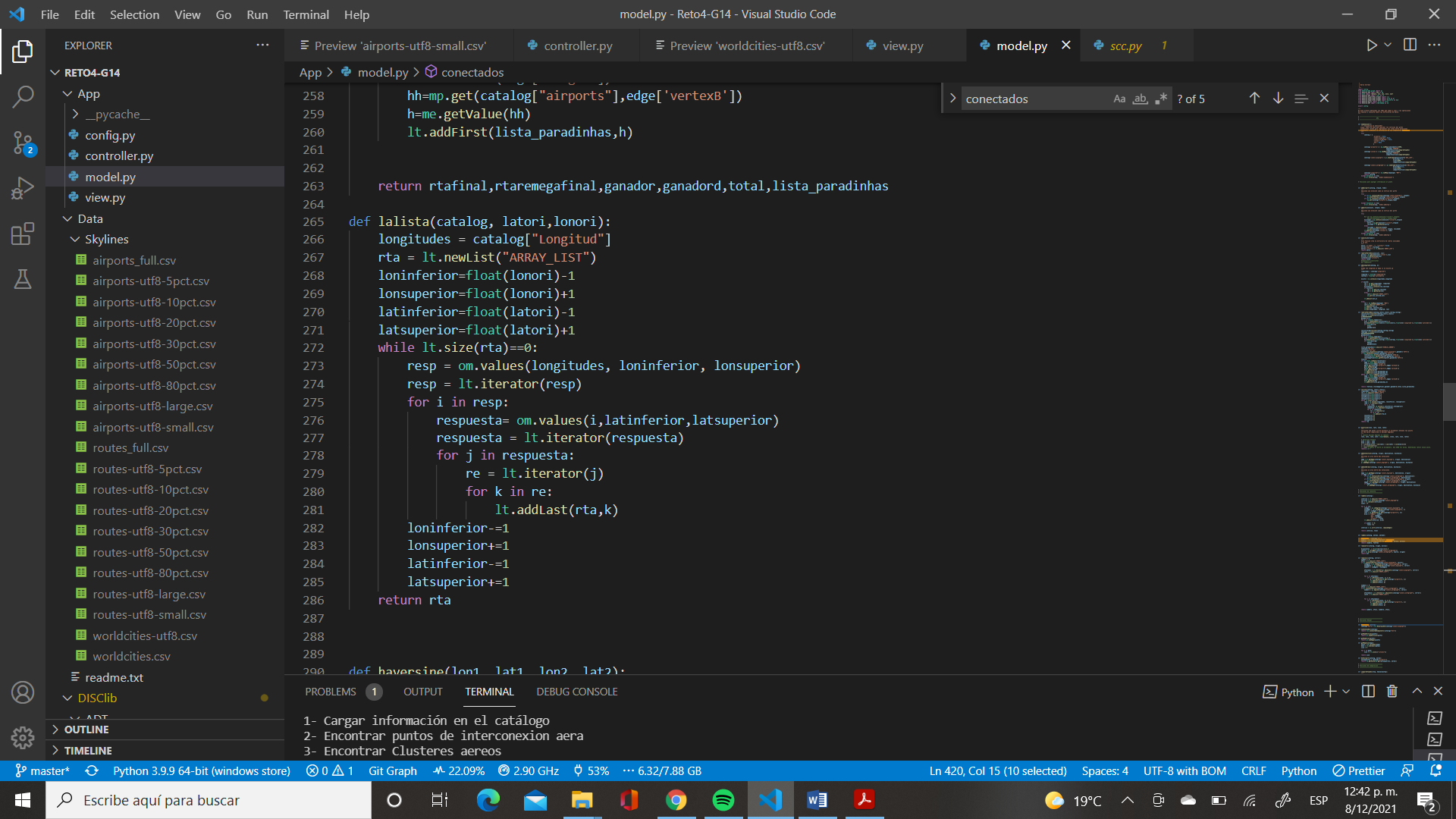
|  |  |
| --- | --- |
| ARCHIVO(%) | TIEMPO(SEG) |
| 10 | 0,0 |
| 20 | 0,0 |
| 30 | 0,0 |
| 50% | 0,0 |
| 80% | 0,0 |
| 100% | 0,0 |

**Requerimiento 3:**

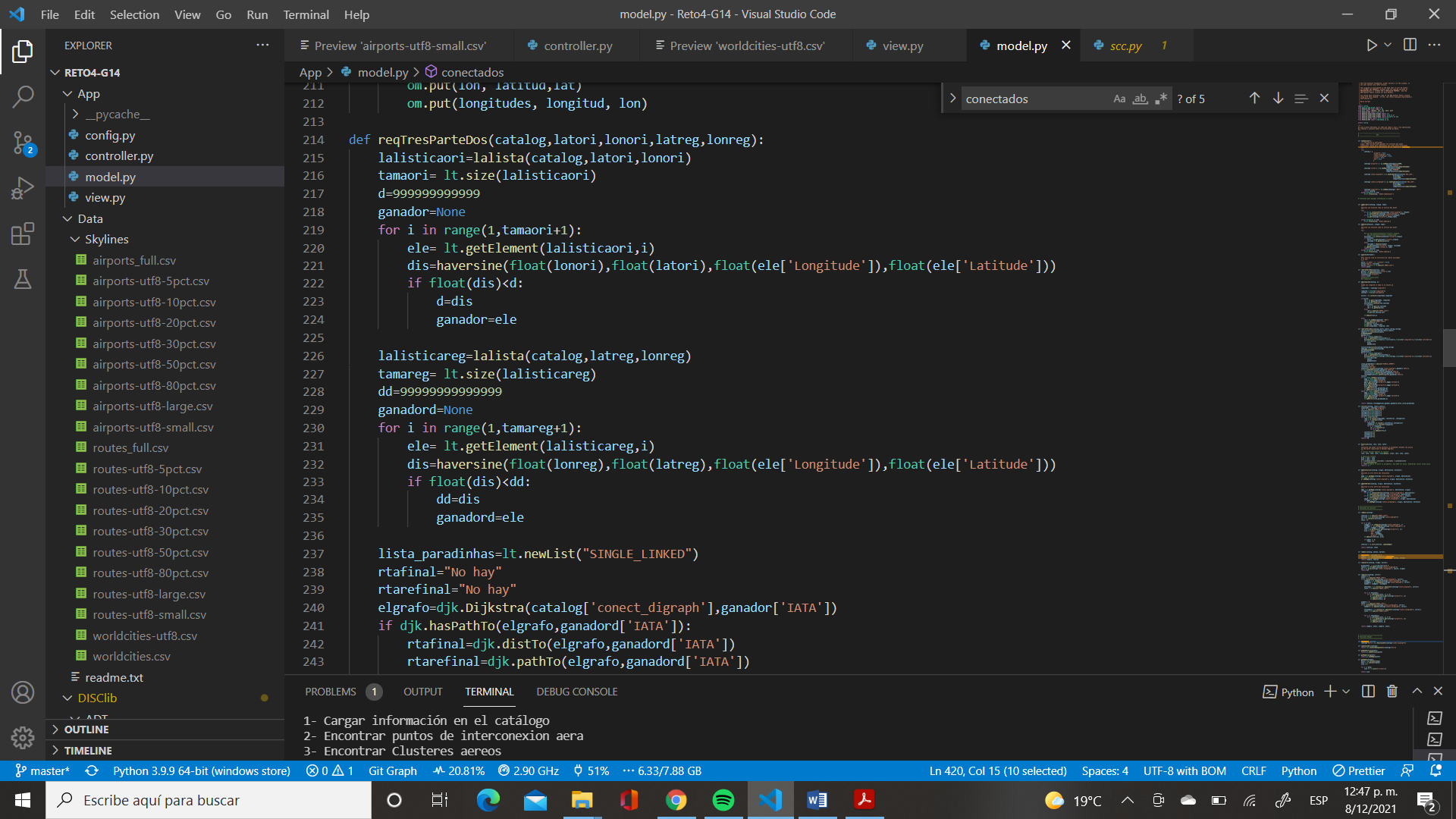
Para poder dar respuesta a la complejidad algorítmica de este requerimiento se debe de tener en cuenta nuevamente un análisis de cada una de las partes que estructuran el mismo. En este sentido, se sabe que la petición en este caso estaba relacionada con la determinación de un camino de peso mínimo que conectara 2 ciudades. Para ello, la primera aclaración que se hacía iba relacionada con la existencia de ciudades homónimas y la oportunidad del usuario de escoger específicamente cual necesitaba. En consecuencia, se construyó desde la función de cargar un mapa con los nombres de las ciudades como llaves y una lista con todas las ciudades homónimas como valor. Así las cosas, una vez el usuario colocaba el nombre de las ciudades de interés, solo se llamaba al mapa y se obtenía la lista correspondiente en una operación de o(1).



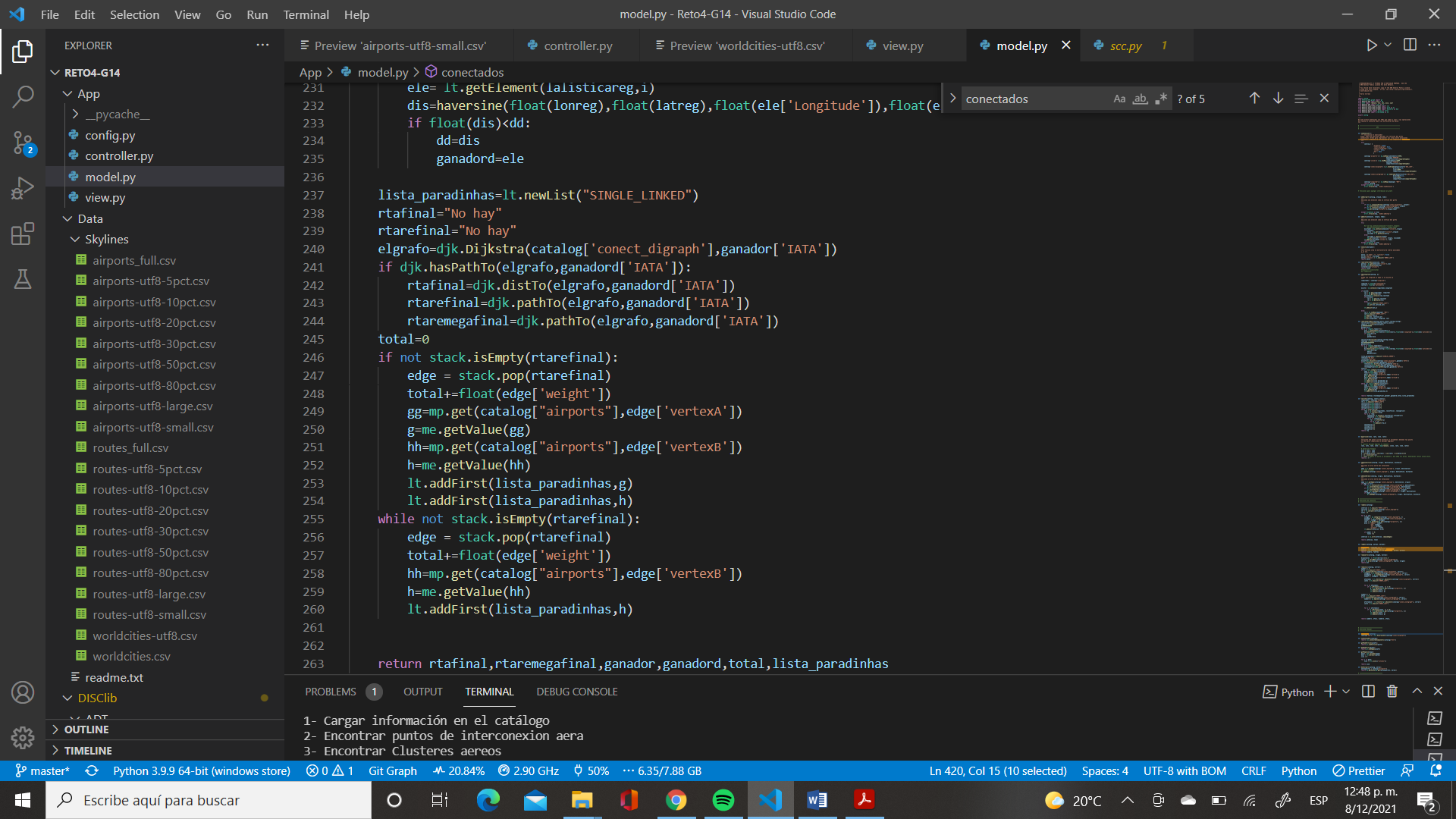
Luego, una vez determinada la ciudad, se procede a encontrar el aeropuerto más cercano. Para ello, de acuerdo con lo propuesto por el enunciado, se utilizó un barrido por secciones del plano coordenado. En consecuencia, se utiliza una aproximación de que cada punto de longitud y latitud equivale a 11000km en promedio, y a partir de ellos se toma la información de todos los aeropuertos en esa zona. Adicionalmente, cabe mencionar que para la realización de este requerimiento se creó un mapa ordenado en la función cargar donde cada una de las llaves corresponden a los valores de la longitud, y los valores nuevamente a otros mapas ordenados donde la llave son las latitudes y los valores los aeropuertos.



Se considera que para casos promedio esta función debe ser o(1), pues reúne y recorre una cantidad de valores bastante inferior a la total del catálogo. De igual manera luego de hacer esto, se analiza y se filtra la lista de aeropuertos del área de interés para encontrar el más cercano a las coordenadas de la ciudad de interés. Para ello se utilizará la ecuación del semiverseno propuesta en el enunciado. Este recorrido, tomando en cuenta que las listas deben tener una cantidad de elementos inferior a la cantidad de ciudades o aeropuertos totales, sigue siendo o(1)



Finalmente, se realiza la parte de mayor interés y relación con grafos del enunciado. Una vez determinado el aeropuerto de origen y de destino, se ejecuta el algoritmo de Dijkstra con en el vértice de inicio. Esta sección tiene una complejidad de o(ElogV). Eventualmente, con la estructura de búsqueda armada, se ejecutan 2 funciones del API para determinar la existencia de la ruta que conecte los aeropuertos de interés, así como sus paradas y peso total. Estas funciones tienen una complejidad algorítmica de o(1). Además, cabe agregar también que aunque el API devuelve una pila con las paradas, se le hace un tratamiento para convertirla en una lista y para obtener el valor total de cada una de las paradas.



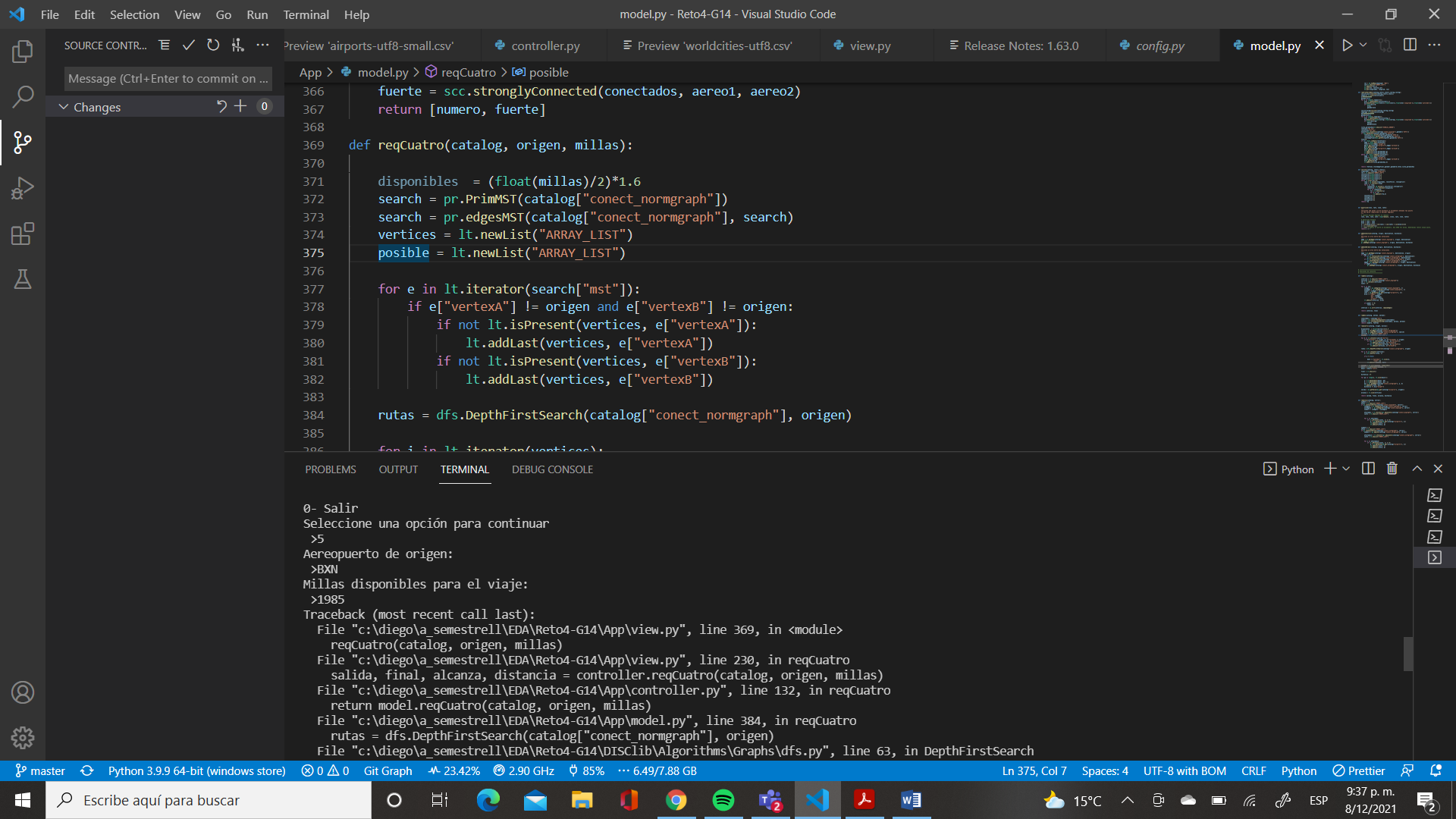
**En conclusión, la complejidad algorítmica total será de o(E log V)**

Máquina 1:

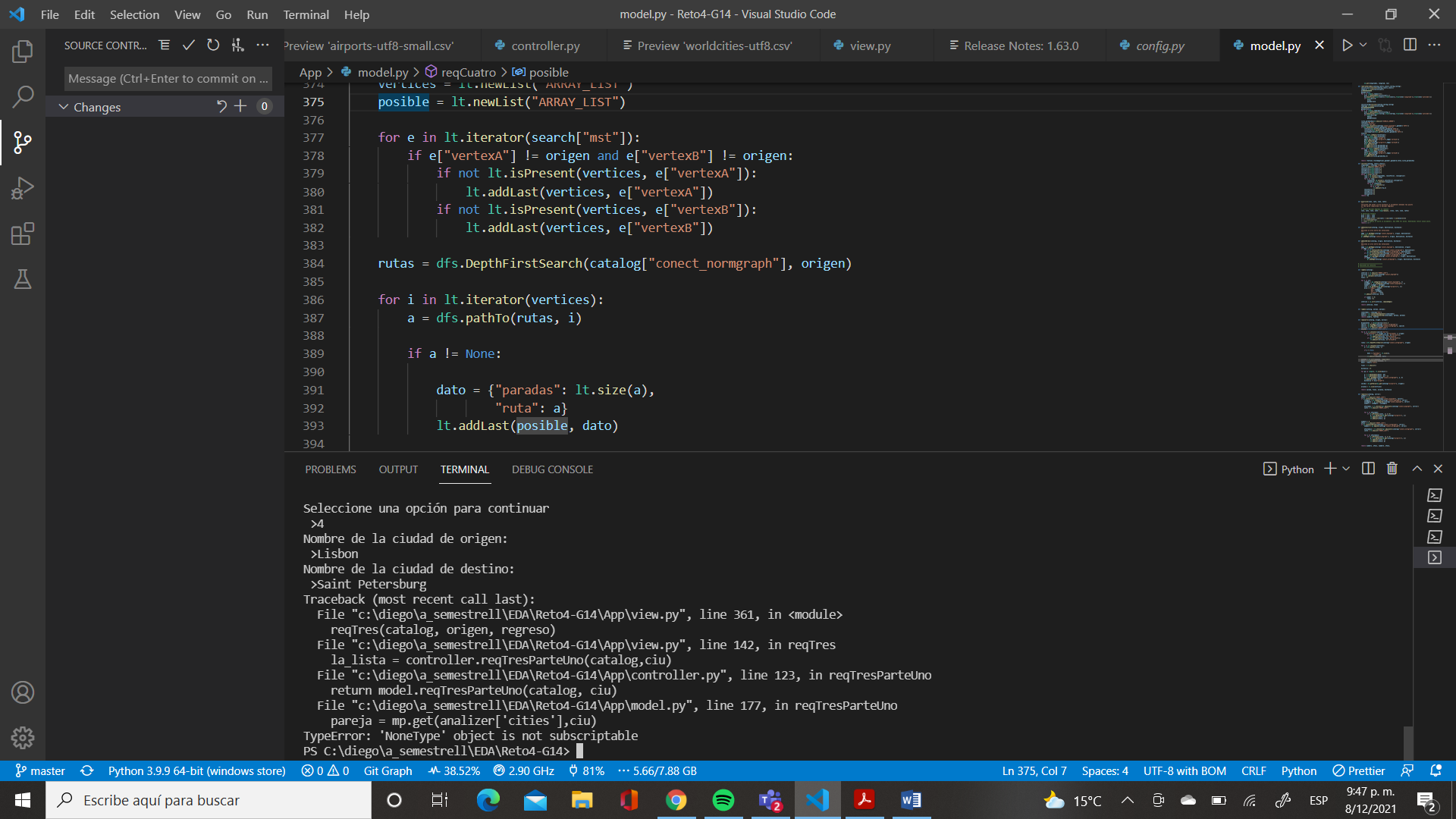
|  |  |
| --- | --- |
| ARCHIVO(%) | TIEMPO(SEG) |
| 10 | 0,0 |
| 20 | 0,475 |
| 30 | 0,500 |
| 50% | 1,485 |
| 80% | 1.87125 |
| 100% | 1.984375 |

**Requerimiento 4:**

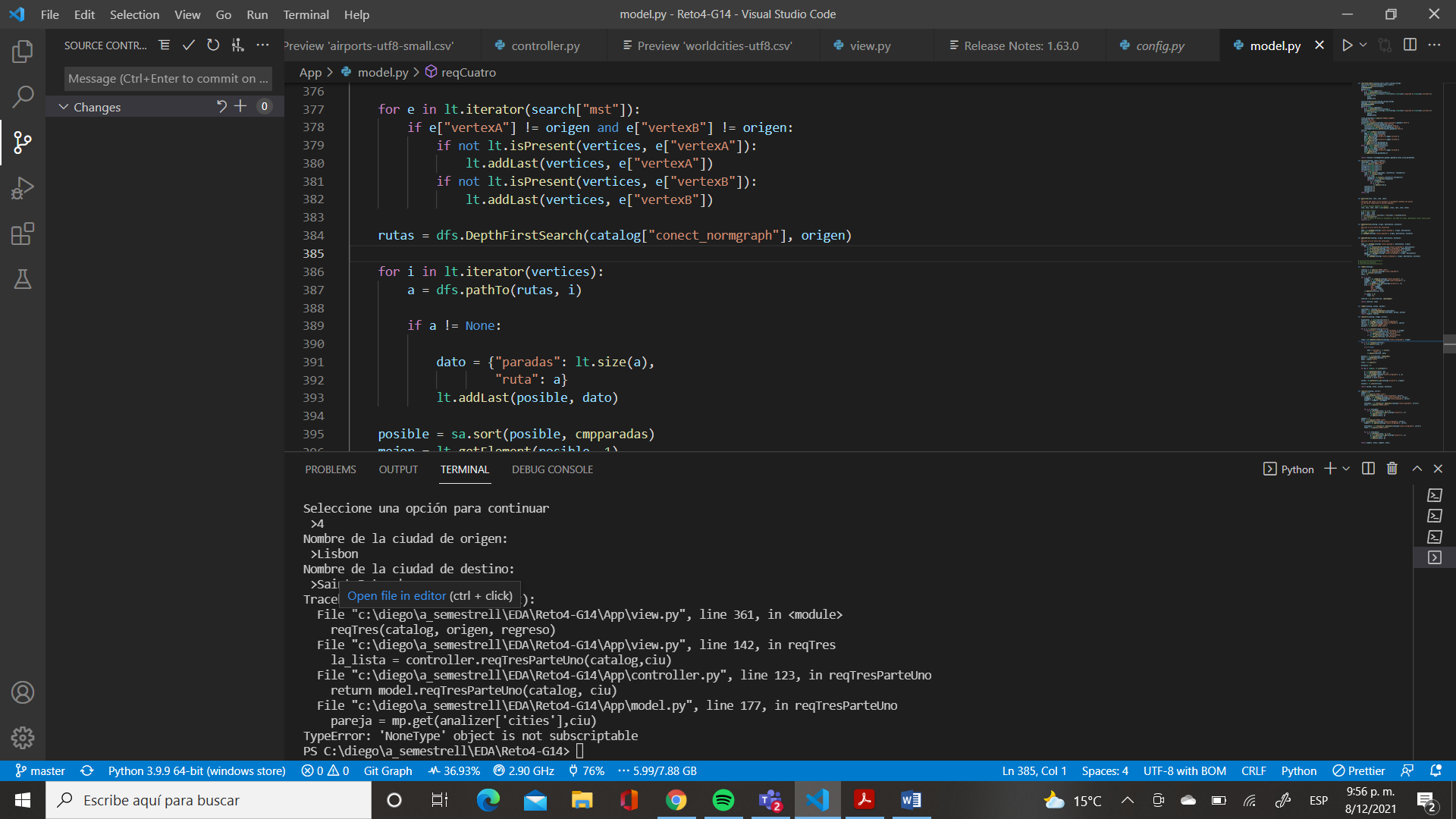
Para poder dar respuesta a la complejidad algorítmica de este requerimiento, una vez más se utiliza el análisis diferenciado de cada una de las secciones que componen el código del requerimiento. Para empezar, se debe recordar que en este requerimiento se nos pedía la creación de un árbol de expansión mínima y la evaluación de la rama más larga del mismo. Adicionalmente, también se pedía la utilidad de dicha respecto a una ciudad específica. Para ello, se decidió hacer uso del algoritmo Prim, que permitía la creación inmediata de dicho árbol.



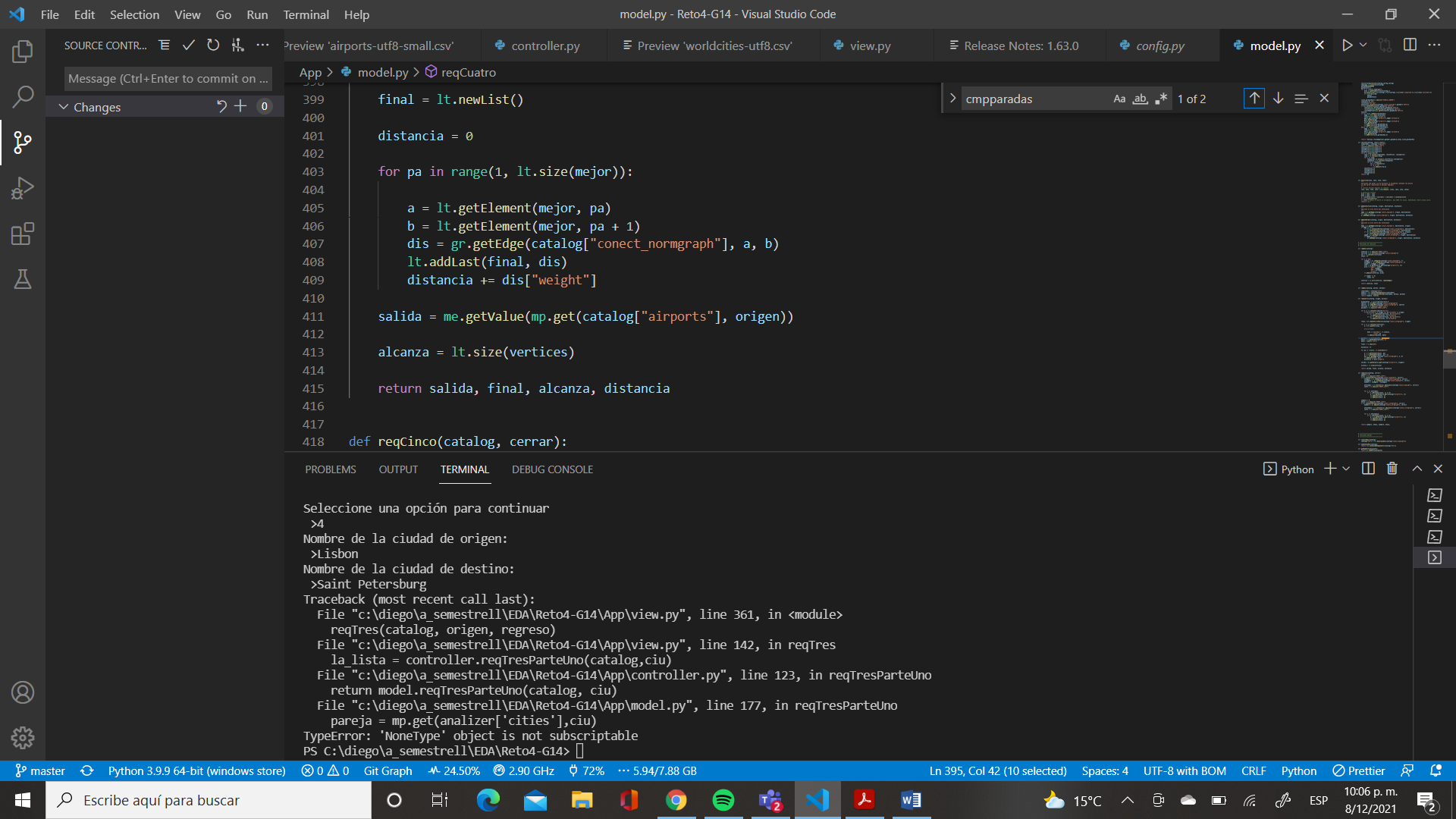
No obstante, es importante mencionar que la aplicación de este algoritmo se realizó sobre el grafo no dirigido, para garantizar que pueda hacer un recorrido circular desde cualquier vértice. Este paso toma una complejidad algorítmica de o(ElogE) donde E hace referencia a los arcos totales. Posteriormente, se realiza una iteración dentro del MST resultante para entender los vértices de relevancia dentro del requerimiento y agregarlos a un TAD lista tradicional para almacenarlos. Este paso en el peor de los casos en el que todos los enlaces este en el árbol de recubrimiento puede ser o(E), pero para casos promedio será de o(1)



Luego de esto, con la ayuda de la un DFS, se determinan todas las rutas posibles dentro del grafo original del catálogo no dirigido. Esta operación tiene un costo temporal de: O(V+E)



A partir del DFS, se realiza un recorrido que permita determinar las rutas de cada uno de los vértices al vértice de interés, operación de O(V). Posteriormente, cada una de esas posibles rutas se ordena de acuerdo con la cantidad de paradas de la misma, en una operación que, como se utiliza un tradicional Shell sort, tiene una complejidad de O(V^1.25). Finalmente, se itera dentro de la considerada mejor ruta para poder obtener el resto de información de interés para la impresión.



**En conclusión, se considera que el algoritmo tiene una complejidad de O(E+V) como su máxima dentro de las operaciones que realiza**

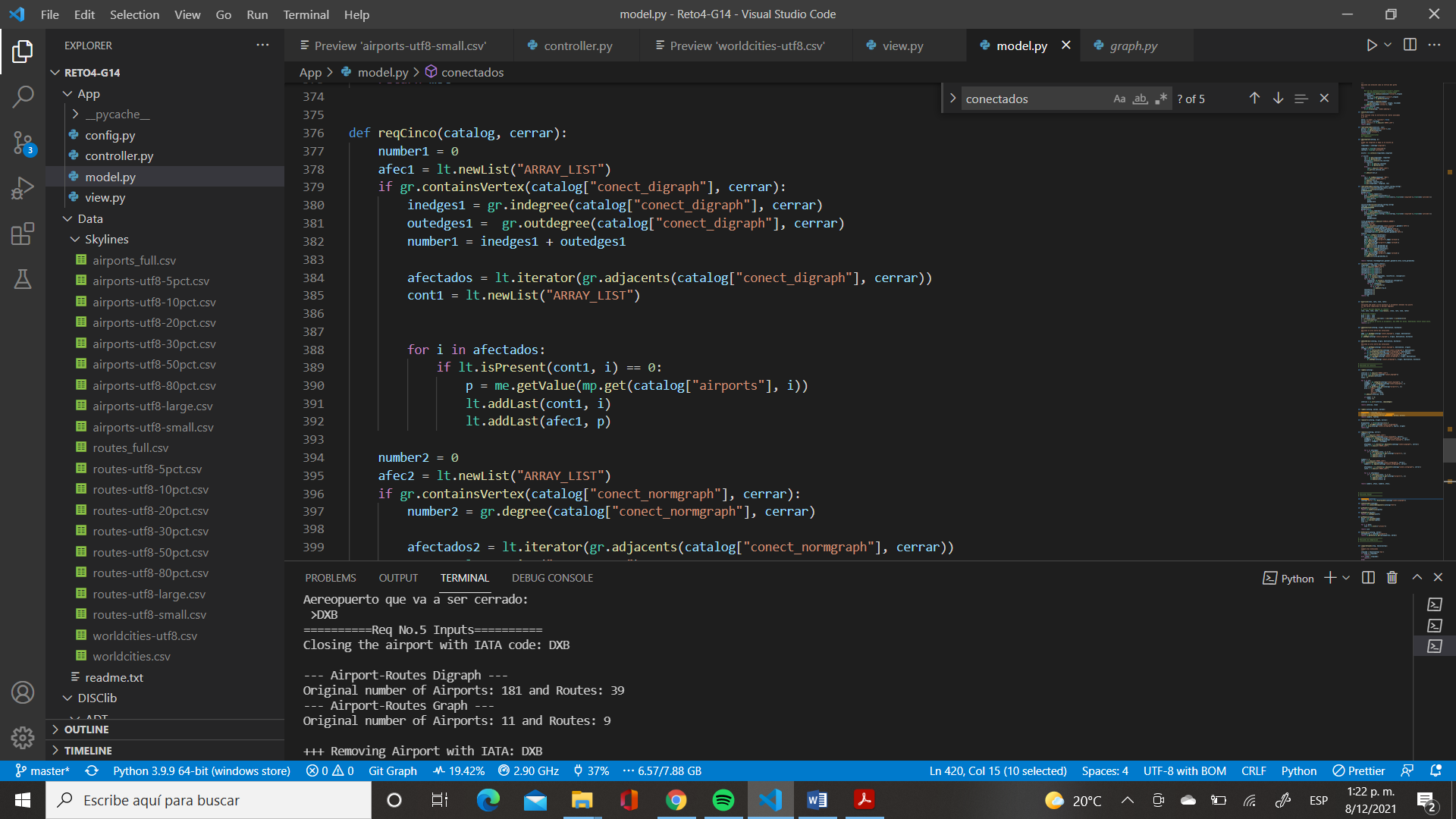
Maquina 1

|  |  |
| --- | --- |
| ARCHIVO(%) | TIEMPO(SEG) |
| 10 | 0,75seg |
| 20 | 3,125 |
| 30 | 5,00seg |
| 50% | 17,7125 |
| 80% | 20,025 |
| 100% | 24.075 |

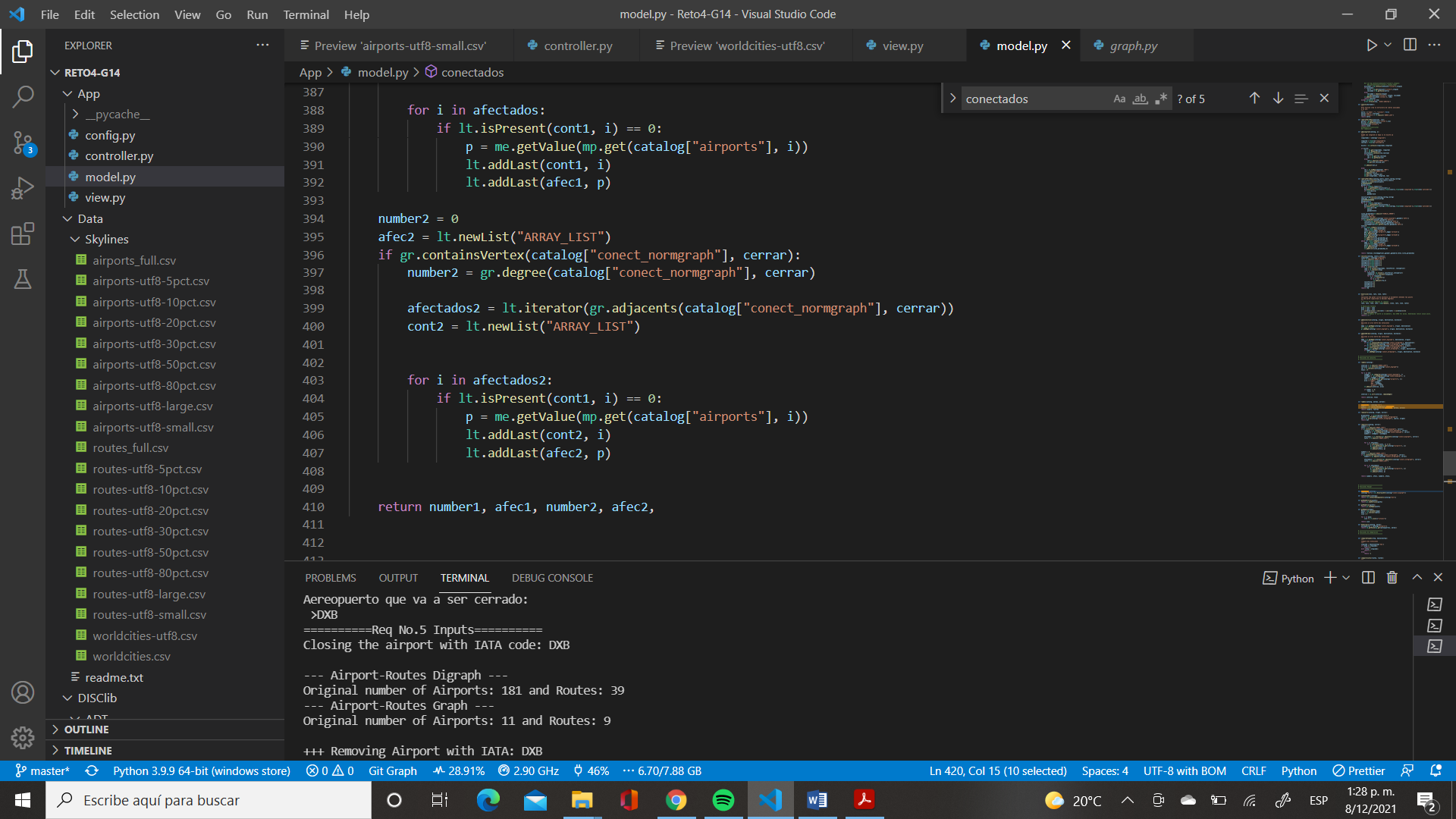
**Requerimiento 5:**

Para poder dar un análisis de complejidad de este requerimiento, nuevamente se utiliza un análisis diferenciado de cada una de las secciones que componen la elaboración del mismo. En este sentido, se sabe que el requerimiento solicitaba la información de la afectación producto de la eliminación de un aeropuerto del grafo. Para poder evaluar esto, solo se hizo necesario de la utilización de propiedades básicas de los grafos, como se mostrara a continuación:

Para empezar, se busca la existencia del aeropuerto que se quiere eliminar, posteriormente se determinan todas las rutas con las que tiene contacto el vértice a partir de la determinación de su grado en una operación que tiene una complejidad algorítmica de o(1). Luego de esto, se determina la lista de adyacencias del vértice de interés y se anexan cada uno de sus elementos a un TAD lista, que servirá como el TAD con todos los aeropuertos afectados, obteniendo la información completa de cada uno de los aeropuertos del mapa realizado en la función cargar. Esta función, tomando en consideración que cada una de las listas de adyacencias no tiene una cantidad de elementos comparable con el total de aeropuertos, se considera o(1).



Finalmente, se realiza un proceso similar, pero con las rutas y vértices del grafo no dirigido:



**Se concluye que la complejidad algorítmica de este requerimiento es o(1)**

Máquina 1:

|  |  |
| --- | --- |
| ARCHIVO(%) | TIEMPO(SEG) |
| 10 | 0,0 |
| 20 | 0,0 |
| 30 | 0,0 |
| 50% | 0,0125 |
| 80% | 0,05 |
| 100% | 0.078125 |