2020_Entrega1

Jump to bottom

Juan Pedro Fisanotti edited this page 13 days ago · 17 revisions

Mercado Artificial

Un sitio que permite la compra y venta de productos entre usuarios (que no existe en la realidad, esta es una idea nueva y original), desea optimizar el recorrido que sus vehículos realizan al buscar y entregar los diferentes productos vendidos.

Cada producto tiene una ciudad de origen y una ciudad de destino, y en cada momento dado hay una gran candidad de productos esperando ser transportados. Debido a ello, la empresa dispone de una flota de camiones que pueden recorrer la región juntando y entregando paquetes al mismo tiempo.

Se desea optimizar el costo de los transportes reduciendo lo más posible el combustible que los camiones consumen.

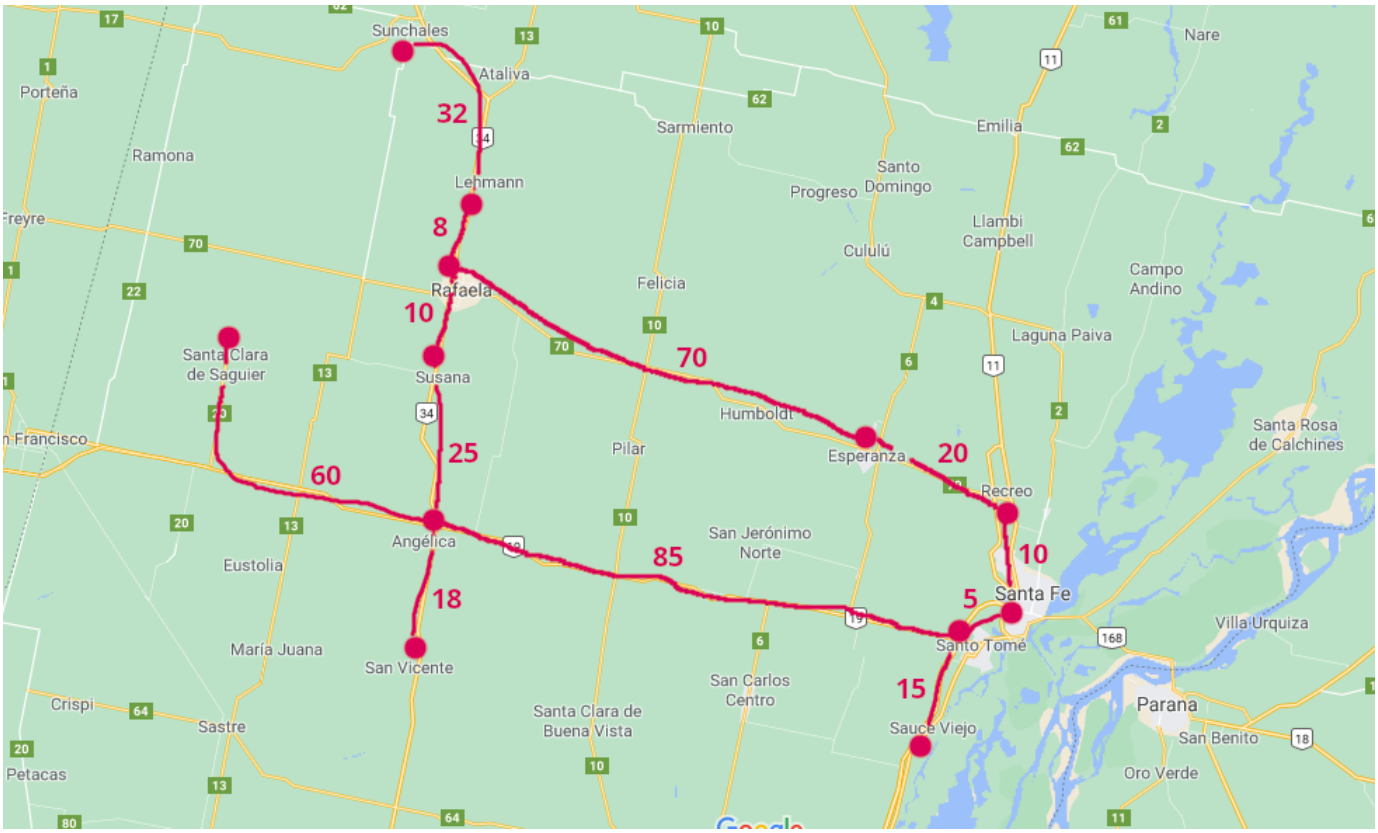
Un aspecto importante a tener en cuenta, es que los camiones consumen combustible, y que por conveniencia de precio solo se permite que recarguen combustible en las sedes de la empresa: Santa Fe y Rafaela. Por ende el viaje de cada camión no termina hasta que no haya regresado a alguna de las sedes. Si un camión se queda sin combustible antes de poder regresar a alguna sede, la solución propuesta no se considera válida.

Se asume que cada vez que un camión pasa por una sede, su tanque de combustible se llena "automáticamente" (no hace falta modelarlo como una operación separada).

Como el problema completo es demasiado complejo, se realizará una primer aproximación con las siguientes simplificaciones:

- Se asume que el consumo de combustible depende solamente de las distancias recorridas entre las ciudades, sin contar el combustible que se gasta dentro de cada ciudad.
- Se asume que los camiones tienen capacidad infinita de carga.
- Solo se considerará un subconjunto de ciudades de la región central de la provincia de Santa Fe, con los ids: rafaela, sunchales, lehmann, susana, sc_de_saguier, esperanza, recreo, santa_fe, san_vicente, santo_tome, angelica y sauce_viejo (todos en minúsculas, sin espacios, y sin acentos).

- Solo consideraremos que se puede ir de una ciudad a otra si están conectadas en el mapa adjunto, y las distancias que utilizaremos son las que figuran en el mismo (en kms).
- El consumo de combustible es de 1 litro cada 100km recorridos.



Objetivo e implementación

El objetivo consiste en tener una función `planear_camiones` que dada una combinación de camiones (con sus ciudades iniciales y capacidad de combustible) productos en espera (cada uno indicando su ciudad de origen y de destino), y método de búsqueda a utilizar, se devuelva un itinerario a recorrer por cada camión solicitado. El itinerario debe dar como resultado la lista de viajes con su consumo de combustible, y los paquetes que se deben llevar en cada uno.

Por ejemplo, que se pueda llamar a la función de esta manera:

```
itinerario = planear_camiones(
  # método de búsqueda a utilizar. Puede ser: astar, breadth_first, depth_first, uni
  metodo="astar",
  camiones=[
    # id, ciudad de origen, y capacidad de combustible máxima (litros)
    ('c1', 'rafaela', 1.5),
    ('c2', 'rafaela', 2),
    ('c3', 'santa_fe', 2),
  ],
  paquetes=[
    # id, ciudad de origen, y ciudad de destino
    ('p1', 'rafaela', 'angelica'),
    ('p2', 'rafaela', 'santa_fe'),
```

```
        ('p3', 'esperanza', 'susana'),
        ('p4', 'recreo', 'san_vicente'),
    ],
)
```

Y que el resultado respete esta estructura (ejemplo con datos sin demasiado sentido, es un itinerario que no resolvería el problema porque quedan paquetes sin entregar):

```
print(itinerario)
[
    # viaje 1: c1 va de rafaela a esperanza llevando p1 y p2
    ('c1', 'esperanza', 0.7, ('p1', 'p2')),

    # viaje 2: c1 vuelve de esperanza a rafaela llevando p3 (por lo que p1 y
    # p2 quedan en esperanza)
    ('c1', 'rafaela', 0.7, ('p3', )),

    # viaje 3: c3, otro camion, va de santa fe (ciudad inicial) a recreo sin
    # llevar nada
    ('c3', 'recreo', 0.1, ()),

    # viaje 4: c1 va de rafaela a susana llevando p3
    ('c1', 'susana', 0.1, ('p3', )),

    # viaje 5: c1 vuelve a rafaela sin traer nada (por lo que p3 queda en
    # susana, entregado)
    ('rafaela', 0.1, ()),

    # (c2 no hizo ningun viaje)
]
```

Como puede verse en el ejemplo, el resultado es una lista de viajes ordenada temporalmente.

Cada viaje es una tupla con 4 datos:

- El id del camión que está viajando.
- La ciudad de destino (su origen se asume como la ciudad anterior en la que estaba, o la de origen del camión en el caso del primer viaje).
- Los litros de combustible que ese viaje en particular va a consumir (por ejemplo, 0.1 litros para viajar de Rafaela a Susana).
- Una tupla de los paquetes que van a viajar en el camión durante ese viaje en particular.

Como se puede ver en el ejemplo, no hace falta indicar cuándo se junta o deja cada paquete, simplemente se debe indicar qué paquetes son transportados en cada viaje. Se asume que si un paquete se deja de transportar, significa que fue dejado en la de origen. Y se asume que si un nuevo paquete se comienza a transportar, entonces es porque tuvo que ser juntado en la ciudad de origen.

En la solución devuelta se debe garantizar que:

- Todos los paquetes queden en su ciudad de destino.
- Todos los camiones terminen en alguna sede de la empresa (rafaela o santa_fe, no hace falta que coincida con la ciudad de inicio del camión)

Ejercicios:

1. Implementar la formulación del problema como problema de búsqueda tradicional para ser resuelto con SimpleAI, incluyendo definición de la clase problema y sus métodos: `cost`, `actions`, `result`, `is_goal` y `heuristic` (la heurística puede ser poco precisa, pero **debe** ser admisible).

2. Implementar la función `planear_camiones` con exactamente la api detallada en la sección anterior (tanto para los datos esperados de entrada, como para el resultado devuelto).

Formato de entrega:

Se debe informar por mail al grupo de la materia: el número de grupo, los integrantes del mismo, y la url del repositorio git donde realizarán la entrega.

La resolución del ejercicio debe realizarse en un archivo llamado `entrega1.py`, que debe ser subido a la raíz del repositorio git del grupo.

El módulo debe tener una función llamada `planear_camiones` que reciba los parámetros mostrados anteriormente, y devuelva el resultado esperado en el formato explicado anteriormente.

Respetar nombres de archivos, funciones, parámetros y tipos de datos **exactamente** como se explican en este enunciado. Cualquier falla por no respetar la interfaz definida, se considera no entregado.

Desde la materia desarrollamos un conjunto de unit tests que les validan muchísimo del funcionamiento de la entrega. La entrega debería pasar todos los tests antes de ser presentada, ya que una entrega que no pasa los tests se considera no resuelta.

Estos tests son también muy muy MUY útiles durante el desarrollo, por lo que les recomendamos que los ejecuten cada vez que quieran probar cosas, y que si tienen problemas nos envíen siempre el resultado de correrlos.

Para correr los tests, deben bajar el archivo `tests_entrega1.py` del directorio `2020` en el repo de la cátedra, y ubicarlo en la raiz del repositorio del grupo. Una vez allí, instalen las dependencias necesarias para correrlo dentro de un virtualenv:

```
pip install pytest pytest-dependency
```

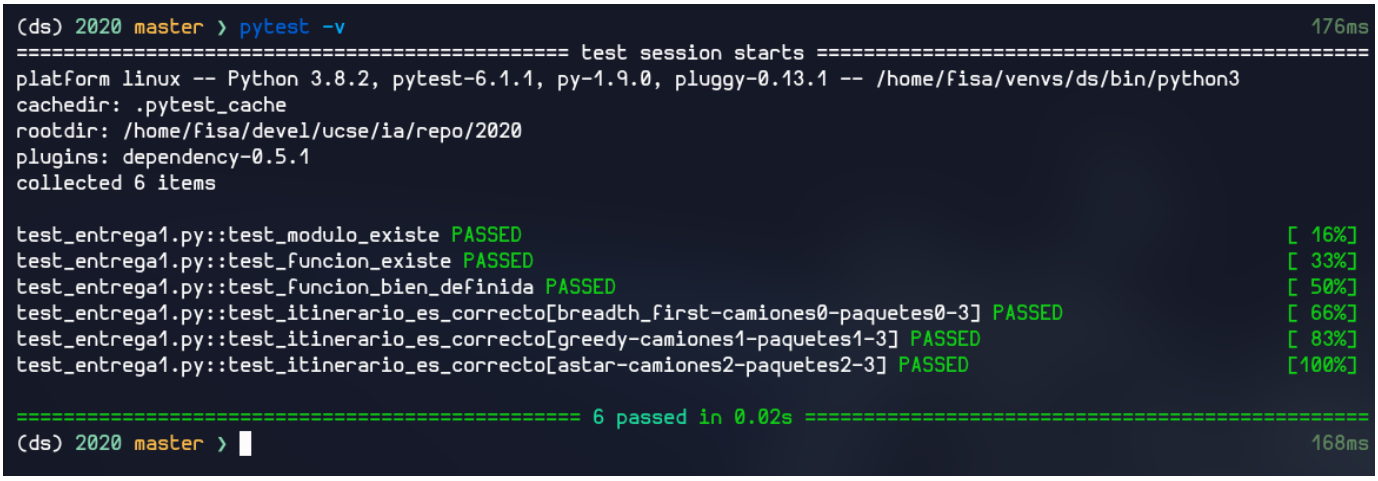
(info sobre cómo usar virtualenvs)

Y luego pueden correr los tests de esta forma:

```
pytest -v
```

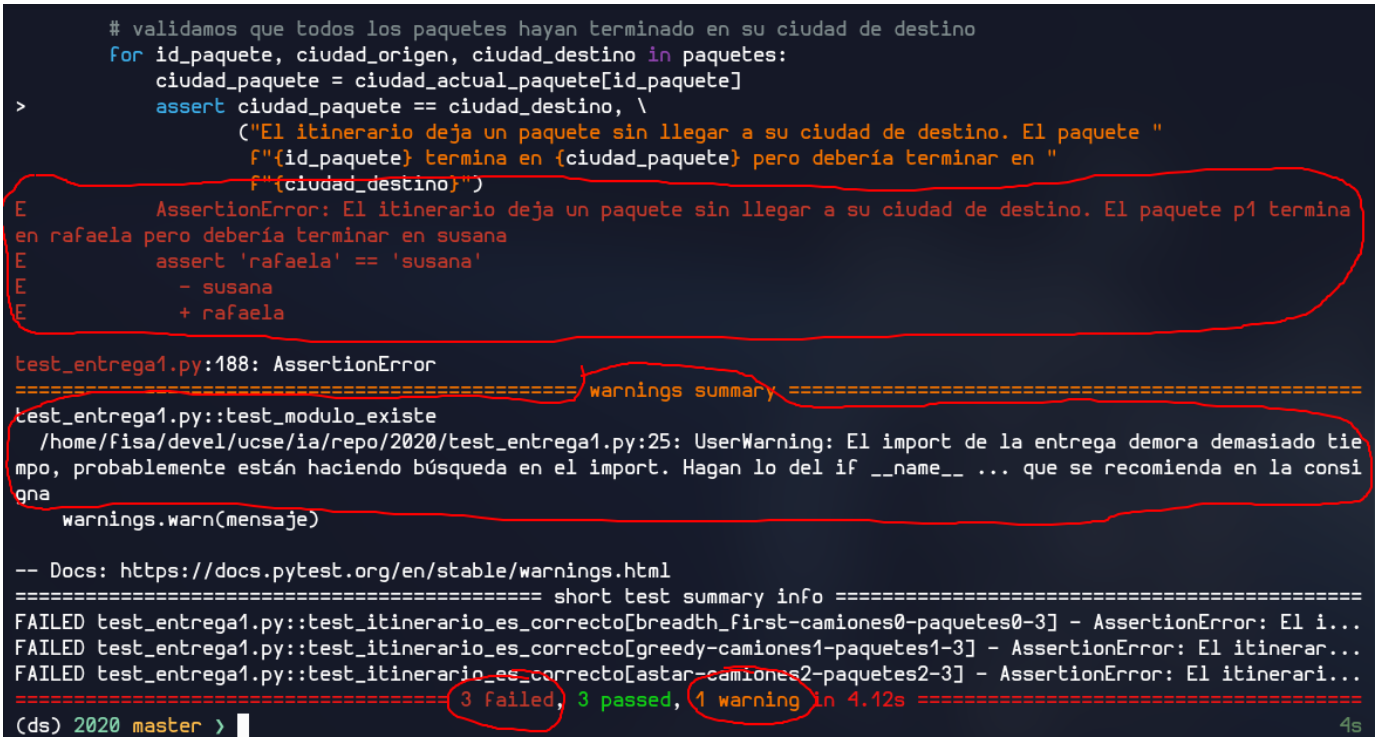
Si eso no funciona, pueden estar seguros de que algo no están haciendo bien. En los casos de error más comunes, los tests pueden explicarles lo que están haciendo mal. En casos más raros, no tanto. **Recuerden que pueden preguntar en el grupo todo lo que necesiten!**

El resultado de una corrida exitosa de tests se ve así:



(ejemplo con pocos tests, pero la cantidad de tests va a ir creciendo a medida que agreguemos casos de ejemplo. Mantengan actualizado el archivo de tests que tienen bajado!)

Mientras que el resultado cuando hay errores o advertencias (recuerden mirar ambas cosas!) se ve así:



Como pueden ver, es super útil para entender qué pueden estar haciendo mal.

Notas útiles:

- Los viewers son útiles para probar y visualizar cosas, encontrar problemas, etc. Pero **recuerden** desactivarlos para la versión entregada, de lo contrario cuando la corrección automática trate de llamar a los algoritmos, se va a quedar tildada esperando o va a demorar demasiado.
- También **recuerden** que el módulo no debe ejecutar ninguna búsqueda al ser importado. Para ello, utilicen el "truco" del `if __name__ == '__main__':` que vimos en clases (está en algunos ejemplos del repo).