Astrada Felicitas
Jenkins Dafydd Alan
Cannestraci Valentino

Trabajo Práctico N2
Introducción a la programación

Aclaraciones

```
53      def distancia(self, lat:float , lng:float) -> float:
54          '''
55          Requiere: nada.
56          Devuelve: la distancia entre un punto dado y el alojamiento.
57          '''
58          alojamiento: Tuple[float, float] = (self.latitud, self.longitud)    #O(1)
59          punto: Tuple[float, float] = (lat, lng)                             #O(1)
60          distancia: float = haversine(alojamiento, punto, unit= Unit.METERS) #O(1)
61          return distancia
62
```

— El método distancia está compuesto por tres líneas de código cuya complejidad es O(1), ya que ocupan un tiempo fijo; creación de tupla a partir de atributos de la clase, creación de tupla a partir de floats pasados como parámetro y creación de una variable de tipo float utilizando la función haversine, cuya complejidad es O(1). Por lo tanto, la complejidad total es O(MAX(1, 1, 1) = O(1).

— Para testear los métodos distancia() de la clase AlojamientosTuristicos, y tres_bolices_mas_cercanos() y boliche_proximo_de_rubro() de la clase DataSetTuristico, utilizamos una calculadora de distancias haversine online (https://acortar.link/noBabl). Con esa calculadora nos aseguramos que las distancias que devolvía haversine eran correctas. Así mismo, para los ultimos dos metodos ingresamos cada uno de los alojamientos y establecimientos testeados, comparando y ordenando las distancias manualmente.

Órdenes de complejidad

```
8   def __init__(self,
9                filename_alojamientos:str,
10               filename_est_gastronomicos:str):
11      ''' Inicializa un Dataset Turistico que englova a las clases Alojamiento Turistico
12          y Establecimento Gastronomico, con atributos:
13          -lista ordenada de Alojamientos Turisticos alojamientos_t
14          -lista ordenada de Establecimientos Gastronomicos establecimientos_g
15      '''
16
17      a:TextIO = open(filename_alojamientos, 'r', encoding = 'utf-8')
18      e:TextIO = open(filename_est_gastronomicos, 'r', encoding = 'utf-8')
19      self.alojamientos_t:List[AlojamientoTuristico] = [] #O(1)
20      self.establecimientos_g:List[EstablecimientoGastronomico] = [] #O(1)
21      for i in csv.DictReader(a): #O(A)
22          if i['Lat'] != '' and i['Lat'] != ' ' and i['Long'] != '' and i['Long'] != ' ': #O(1)
23              nom:str = i['nombre'] #O(1)
24              lat:float = float(i['Lat']) #O(1)
25              long:float = float(i['Long']) #O(1)
26              nom_calle:str = i['calle'] #O(1)
27              alt_calle:int = int(i['altura']) #O(1)
28              tel:str = i['telefono'] #O(1)
29              email:str = i['email'] #O(1)
30              alo:AlojamientoTuristico = AlojamientoTuristico(nom, lat, long, nom_calle, alt_calle, tel, email) #O(1)
31              self.alojamientos_t.append(alo) #O(1)
32      a.close()
33      self.alojamientos_t.sort() #O(A*log(A))
34      for i in csv.DictReader(e): #O(E)
35          if i['lat'] !='' and i['lat'] !=' ' and i['long'] !='' and i['long'] != ' ' and i['altura'] != '' and i['altura'] != ' ': #O(1)
36              nom: str = i['establecimiento'] #O(1)
37              lat:float = float(i['lat']) #O(1)
38              long:float = float(i['long']) #O(1)
39              rubro: str = i['rubro'] #O(1)
40              calle: str = i['calle'] #O(1)
41              altura: int = int(i['altura']) #O(1)
42              est: EstablecimientoGastronomico = EstablecimientoGastronomico(nom, calle, altura, rubro, long, lat) #O(1)
43              self.establecimientos_g.append(est) #O(1)
44      e.close()
```

Astrada Felicitas
Jenkins Dafydd Alan
Cannestraci Valentino

Trabajo Práctico N2
Introducción a la programación

O(MAX(A*log (A), E))

- En las líneas 19 y 20, la complejidad es O(1) por ser operaciones simples (creación de listas).
- Dentro del primer ciclo (líneas 22 a 31), todas las operaciones tienen O(1). Si bien, de la línea 22 a la 29 estamos obteniendo valores asociados a las claves de un diccionario, la longitud de este (cantidad de claves) es acotada, por lo que siempre será 12. En consecuencia, se ejecutan en cierto tiempo fijo. Por otro lado, las líneas 30 y 31, tienen O(1) por ser operaciones simples (creación de un nuevo objeto y append a una lista).
- Dicho ciclo se ejecuta A cantidad de veces, siendo A la cantidad de alojamientos en el csv (línea 21).
- En la línea 33 se ordena la nueva lista creada de alojamientos, cuya longitud es A, por lo que esa línea de código tiene O(A*log (A)).
- Durante el segundo ciclo (líneas 35 a 43), todas las operaciones tienen O(1) por el mismo razonamiento que explicado en el primer ciclo.
- El segundo ciclo se ejecuta E cantidad de veces, siendo E la cantidad de establecimientos gastronómicos en el csv (línea 34).
- Finalmente, la complejidad del método es la secuenciación de dos líneas con O(1), un ciclo O(A), una línea con O(A * log (A)) y un ciclo con O(E). Por lo tanto, la complejidad total será O(MAX(1, 1, A, A* log (A), E)) lo que equivale a **O(MAX(A * log(A), E)**, ya que O(A) y O(1) son menores a O(A * log (A)).

```
46      def alojamientos(self) -> List[AlojamientoTuristico]:
47          ''' Requiere: Nada.
48              Devuelve: Una lista con los alojamientos en el dateset ordenados alfabeticamente.
49          '''
50          return self.alojamientos_t #O(1)
```

O(1)

- El método consta de una sola línea, en la cual la complejidad es O(1) por ser una expresión simple (return de un valor pre-calculado).

```
52      def alojamiento_por_nombre(self, nom:str) -> AlojamientoTuristico:
53          ''' Requiere: nom es el nombre de un alojamiento existente en dataset.
54              Devuelve: El AlojamientoTuristico correspondiente al nombre dado.
55          '''
56          izq:int = 0 #O(1)
57          ls:List[AlojamientoTuristico] = self.alojamientos_t #O(1)
58          der:int = len(ls) #O(1)
59          while izq < der: #O(log(A))
60              med:int = (izq + der) // 2 #O(1)
61              if ls[med].nombre == nom: #O(1)
62                  return ls[med] #O(1)
63              elif ls[med].nombre < nom: #O(1)
64                  izq = med + 1 #O(1)
65              else:
66                  der = med #O(1)
```

Astrada Felicitas
Jenkins Dafydd Alan
Cannestraci Valentino

Trabajo Práctico N2
Introducción a la programación

O(log A)

— En las líneas 56 a 58, la complejidad es O(1) por ser expresiones simples (definición de variables tipo entero y creación de lista).
— Dentro del ciclo (líneas 60 a 66), la complejidad es de O(1). La línea 60 tiene O(1) por ser una operaciones simples entre enteros y asignación del valor resultante a una variable. Si bien, en las líneas 61 y 63 se están comparando strings, los mismos se consideran acotados, ya que son palabras pertenecientes al idioma español que tienen una longitud máxima, por lo que se ejecutan en cierto tiempo acotado. Así mismo, por encontrarse dentro de un condicional, se toma el orden máximo entre sus posibles casos, el cual es de O(1).
— El ciclo se repite O(log (A)) veces, siendo A la cantidad de alojamientos en el csv. Esto se debe a que, en cada iteración del ciclo, la longitud de la lista recorrida se divide por dos. Así la complejidad final del ciclo es de O(1 * log(A)) = O(log (A)).
— Finalmente, el método es la secuenciación de tres líneas con O(1) y un ciclo con O(log(A)), es decir su complejidad es O(MAX(1, 1, 1, log (A)) = O(log (A)).

```
68      def tres_boliches_cercanos(self, aloj:AlojamientoTuristico) -> Tuple[EstablecimientoGastronomico,
69                                                                          EstablecimientoGastronomico, EstablecimientoGastronomico]:
70          ''' Requiere: aloj es un Alojamiento Turistico existente en el Dataset.
71              Devuelve: los Establecimientos Gastronomicos mas cercanos a aloj.
72          '''
73          res:Tuple[EstablecimientoGastronomico, EstablecimientoGastronomico, EstablecimientoGastronomico] = tuple() #O(1)
74          distancias:List[Tuple[EstablecimientoGastronomico, float]] = [] #O(1)
75          menores:List[Tuple[EstablecimientoGastronomico, float]] = [] #O(1)
76
77          for i in self.establecimientos_g: #O(E)
78              distancia:Tuple[EstablecimientoGastronomico, float] = (i, aloj.distancia(i.latitud, i.longitud)) #O(1)
79              distancias.append(distancia) #O(1)
80
81          for j in range(3): #O(1)
82              pos:int = 0  #O(1)
83              for i in range(0, len(distancias)): #O(len(distancias))
84                  if distancias[i][1] < distancias[pos][1]: #O(1)
85                      pos = i #O(1)
86              menores.append(distancias[pos]) #O(1)
87              distancias.pop(pos) #O(len(distancias))
88
89          res = (menores[0][0], menores[1][0], menores[2][0]) #O(1)
90          return res
```

O(E)

— En las líneas 73 a 75, la complejidad es O(1), por ser expresiones simples, es decir, definición de listas vacías y creación de una tupla.
— Dentro del primer ciclo (líneas 78 y 79), la complejidad es de O(1), porque estamos generando una variable de tipo tupla utilizando el método distancia de la clase alojamiento turístico, cuya complejidad es O(1) (ver sección “Aclaraciones”). En la siguiente, utilizamos el append a dicha lista, operación que también tiene O(1). Ese ciclo (línea 77) se repite E veces, por lo que la complejidad es O(1 * E) = O(E).
— En el segundo ciclo (líneas 82 a 87), el código es la secuenciación de:
    - Las líneas 82 y 86 cuentan con complejidad O(1), por ser expresiones simples (la definición de una variable como entero y el append a una lista).
    - Un ciclo (líneas 83 a 85), en el cual la complejidad es O(1). La evaluación de la condición tiene O(1), por estar comparando floats y la línea 85, es una expresión simple, en la cual se da un valor entero a una variable. Dicho ciclo se repite len(distancias) veces, por lo tanto la complejidad final del ciclo es de O(1 * len(distancias)) = O(len(distancias)).
    - La línea 87 con complejidad O(len(distancias)), ya que se está aplicando el pop intermedio a una lista.

Finalmente la complejidad total del segundo ciclo es O(MAX(1, len(distancias), len(distancias)), lo que equivale a O(len(distancias)).

— La línea 89 tiene O(1), ya que es una asignación de tupla a partir de elementos de una lista en una posición dada.
— Finalmente, la complejidad del método es O(MAX(1, E, len(distancias))). Como distancias es una lista con cantidad de tuplas igual o menor a la cantidad de alojamientos en el csv, O(len(distancias)) es menor o igual a O(E). Así, la complejidad total es O(E).

```
92      def boliche_proximo_de_rubro(self, aloj:AlojamientoTuristico, rub:str) -> EstablecimientoGastronomico:
93          '''Requiere: aloj es un Alojamiento Turistico existente en el Dataset,
94              y rub es un rubro perteneciente a algun Establecimento Gastronomico en el Dataset.
95              Devuelve: el Establecimiento Gastronomico con rubro rub mas cercano a aloj.
96          '''
97
98          candidatos:List[EstablecimientoGastronomico] = [] #O(1)
99          distancias:List[Tuple[EstablecimientoGastronomico, float]] = [] #O(1)
100         pos:int = 0 #O(1)
101         for i in self.establecimientos_g: #O(E)
102             if i.rubro == rub: #O(1)
103                 candidatos.append(i) #O(1)
104         for i in candidatos: #O(len(candidatos))
105             distancia:Tuple[EstablecimientoGastronomico, float] = (i, aloj.distancia(i.latitud, i.longitud)) #O(1)
106             distancias.append(distancia) #O(1)
107         for i in range(0, len(distancias)): #O(len(distancias))
108             if distancias[i][1] < distancias[pos][1]: #O(1)
109                 pos = i #O(1)
110         res:EstablecimientoGastronomico = distancias[pos][0] #O(1)
111         return res
```

O(E)

— Las líneas 98 a 100 tienen O(1), debido a que son creaciones de listas vacías y definición de una variable como entero.
— El primer ciclo (líneas 101 a 103) está compuesto por un condicional, cuyas líneas son O(1) gracias a que son una comparación de strings acotados y el append a una lista. Éste código se repite E veces, por lo que la complejidad es O(E * 1) = O(E).
— Durante el segundo ciclo (líneas 105 y 106) todas sus líneas tienen una complejidad O(1), por ser expresiones simples; definición de una variable de tipo tupla utilizando el método distancia (que tiene O(1), ver sección “Aclaraciones”), y append a una lista. Este ciclo (línea 104), se repite len(candidatos) cantidad de veces, por lo que su complejidad es O(len(candidatos) * 1) = O(len(candidatos)).
— El tercer ciclo (líneas 107 a 110) está compuesto por un condicional, cuyas líneas tienen complejidad O(1), ya que son expresiones simples (comparación de floats y asignación de una variable como entero). Este ciclo se repite O(len(distancias)) veces, por lo que la complejidad total del ciclo es de O(1 * len(distancias)) = O(len(distancias)).
— La línea 110 tiene complejidad O(1), ya que se está asignando un valor de una lista en una posición fija a una variable.
— Finalmente, la complejidad total del método es O(MAX(1, E, len(candidatos), len(distancias))), lo cual equivale a O(E) debido a que tanto len(candidatos) y len(distancias) son menores o iguales a E, por ser candidatos y distancias listas con cantidad de elementos menor o igual a la cantidad de Establecimientos Gastronomicos en el csv.