

pol-voraces

March 24, 2024

1 Algoritmia

1.1 Práctica Obligatoria 1

1.1.1 Curso 2023 - 2024

1.2 ##### Métodos Voraces

Autores:

- Cristian Fernández Martínez
- Alicia García Pérez

Resuelva la siguiente práctica.

Importe las librerías que desees **Recuerda:** * Solamente puedes utilizar bibliotecas nativas (<https://docs.python.org/es/3.8/library/index.html>) * Las funciones que importes no son “gratis”, cada una tendrá una complejidad temporal y espacial que se tendrá que tener en cuenta.

```
[1]: #testeable
      # Imports
```

```
[2]: #testeable
class Video:
    """
    Clase Video.
    Representa una serie o película.
    """

    def __init__(self, name, size):
        """Crea un objeto de clase Video

        Parameters
        -----
        name : str
            Nombre de la serie/película
        size : number
            Tamaño en memoria de la serie/película
        """
```

```

self.name = name
self.size = size
self.users = {}

def __hash__(self):
    """Genera el valor hash identificativo del video

    Returns
    -----
    int
        Valor hash
    """
    return hash((self.name, self.size))

def __str__(self):
    """Genera una cadena descriptiva del objeto

    Returns
    -----
    str
        Cadena descriptiva
    """
    return f'Nombre del video: {self.name}, tamaño: ({self.size} MB)'

def __repr__(self):
    """Genera una cadena descriptiva del objeto dentro de colecciones

    Returns
    -----
    str
        Cadena descriptiva
    """
    return f'Nombre del video: {self.name}, tamaño: ({self.size} MB); hash:␣
↪{hash(self)}'

def set_users(self, country, users):
    """Dado un país y un número de usuarios
        almacena para este video la cantidad de espectadores que tiene.

    Parameters
    -----
    country : str
        País desde donde se ve la serie/película
    users : int
        Número de espectadores
    """

```

```

        self.users[country] = users

    def get_users(self, country):
        """Dado un país, obtiene el número de usuarios.

        Parameters
        -----
        country : str
            País desde donde se ve la serie/película

        Returns
        -----
        int
            Número de espectadores para el país `country`
        """
        return self.users.get(country, 0) #Si el país no existe, devuelve 0

```

```

[3]: #testable
class ServidorCache:
    """
    Clase del servidor caché donde se almacenan parte de series/películas.
    """

    def __init__(self, identifier, country, capacity):
        """Instancia un Servidor de Caché

        Parameters
        -----
        identifier : int
            Valor que identifica un servidor.
        country : str
            País donde está el servidor.
        capacity : int
            Cantidad de memoria de almacenamiento disponible.
        """
        self.identifier = identifier
        self.country = country
        self.capacity = capacity
        self.videosAlmacenados = {} #Diccionario con los videos almacenados y
        ↪ la cantidad almacenada de cada uno.

    def __hash__(self):
        """Genera el valor hash identificativo del servidor

        Returns
        -----
        int

```

```

        Valor hash
        """
        return hash((self.identifier, self.country, self.capacity))

def __str__(self):
    """Genera una cadena descriptiva del objeto

    Returns
    -----
    str
        Cadena descriptiva
    """
    return f'Servidor {self.identifier} en {self.country} con capacidad_
↪{self.capacity} MB'

def __repr__(self):
    """Genera una cadena descriptiva del objeto en colecciones

    Returns
    -----
    str
        Cadena descriptiva
    """
    return f'Servidor {self.identifier} en {self.country} con capacidad_
↪{self.capacity} MB; hash: {hash(self)}'

def rellena(self, videos):
    """Dada una colección de videos,
        seleccionar de cada uno cuanta cantidad (entre 0 y 1)
        se almacena en el servidor.
        Se ha de optimizar para que el tiempo de emisión
        sea el máximo posible.

    Parameters
    -----
    videos : collection
        Colección de videos que se quieren almacenar en el servidor.
    """
    videos_valor = {}
    densidad = 0
    for video in videos: #O(n)
        # Calculamos el valor de cada video: (tam*esp/tam = esp)
        valor = video.get_users(self.country)
        videos_valor[video] = valor
        # Ordenamos los videos por valor
        videos_valor = sorted(videos_valor.items(), key=lambda x: x[1],
↪reverse=True) #O(nlogn)

```

```

pesoUtilizado = 0
# Vamos almacenando los videos en el servidor
while pesoUtilizado < self.capacity: #O(n)
    for video in videos_valor: #O(n)
        # Si cabe el video completo
        if video[0].size <= self.capacity - pesoUtilizado:
            self.videosAlmacenados[video[0]] = 1
            pesoUtilizado += video[0].size
        # Si no cabe el video completo
        else:
            entra = (self.capacity - pesoUtilizado) / video[0].size
            self.videosAlmacenados[video[0]] = entra
            pesoUtilizado += video[0].size * entra
            break
    return self.videosAlmacenados

def disponible(self, video):
    """Obtiene la cantidad de video disponible en el servidor.

    Parameters
    -----
    video : Video object
        Video del cual se quiere saber la disponibilidad

    Returns
    -----
    float
        Cantidad del video disponible
    """
    self.videosAlmacenados.get(video, 0)

def almacenados(self):
    """Material almacenado en el servidor

    Returns
    -----
    set
        Conjunto de tuplas (video, cantidad) de los videos ALMACENADOS en
    el servidor.
    """
    tuplas = set()
    for video in self.videosAlmacenados:
        tuplas.add((video, self.videosAlmacenados[video]))
    return tuplas

```

```

def tiempo_emision(self):
    """A partir de los datos almacenados
        devolver el tiempo de emisión
        siguiendo la fórmula:
        \sum_{i \sim v} \square
    ↪ \text{espectadores}_i * \text{tamaño}_i * \text{porcionAlmacenada}_i

    Returns
    -----
    number
        Tiempo de emision disponible
    """
    tiempo = 0
    for video in self.videosAlmacenados:
        tiempo += video.get_users(self.country) * video.size * self.
    ↪ videosAlmacenados[video]
    return tiempo

```

```

[4]: #testable
class ServidorMaestro:
    """
        Servidor central que gestiona las conexiones entre servidores cache
    """

    def __init__(self, servidores, distancias):
        """Instancia el servidor central

        Parameters
        -----
        servidores : Iterable
            Conjunto de servidores cache disponibles
        distancias : dict{ServidorCache: dict{ServidorCache: int}}
            Grafo de distancias en milisegundos entre servidores.
        """
        self.servidores = set(servidores)
        self.distancias = distancias
        self.grafo_simplificado = {}

    def get_grafo(self):
        """Devuelve el grafo de distancias recibido

        Returns
        -----
        dict{ServidorCache: dict{ServidorCache: int}}
            Grafo de distancias en milisegundos entre servidores.
        """

```

```

        return self.distancias

def get_grafo_simplificado(self):
    """Devuelve el grafo de distancias simplificado

    Returns
    -----
    dict{ServidorCache: dict{ServidorCache: int}}
        Grafo de distancias en milisegundos entre servidores.
    """

    return self.grafo_simplificado

def simplifica_grafo(self):
    """A partir del grafo de distancias
        hacer una simplificación de la estructura
        de datos para ahorrar espacio y tiempo.
    """
    # Inicializar el grafo simplificado y la lista de servidores visitados
    self.grafo_simplificado = {servidor: {} for servidor in self.
↪servidores} #O(n)
    visitados = set()

    # Comenzar con un servidor arbitrario
    actual = next(iter(self.servidores))
    visitados.add(actual)

    # Inicializar los diccionarios de servidor más cercano y costo mínimo
    mas_cerca = {servidor: actual for servidor in self.servidores} #O(n)
    menor_coste = {servidor: self.distancias[actual].get(servidor, None) for
↪servidor in self.servidores} #O(n)

    while len(visitados) < len(self.servidores): #O(n)
        # Encontrar el servidor no visitado con el costo más pequeño
        actual = min((servidor for servidor in self.servidores if servidor
↪not in visitados), key=menor_coste.get)
        visitados.add(actual)

        # Añadir la arista al grafo simplificado
        cerca = mas_cerca[actual]
        self.grafo_simplificado[actual][cerca] = menor_coste[actual]
        self.grafo_simplificado[cerca][actual] = menor_coste[actual]

        # Actualizar los diccionarios de servidor más cercano y costo mínimo
        for otro_servidor in self.servidores: #O(n)
            if otro_servidor not in visitados and self.
↪distancias[actual][otro_servidor] < menor_coste[otro_servidor]:

```

```

        mas_cerca[otro_servidor] = actual
        menor_coste[otro_servidor] = self.
↪distancias[actual][otro_servidor]

def mas_cercano(self, servidor):
    """Reporta el servidor más cercano al dado por parámetro

    Parameters
    -----
    servidor : ServidorCache

    Returns
    -----
    ServidorCache
        Servidor más cercano
    """
    # Inicializar el servidor más cercano y la distancia mínima con valores
↪nulos
    servidor_mas_cercano = None
    distancia_minima = float('inf')
    # Recorrer todos los servidores
    for otro_servidor in self.servidores:
        # Ignorar el servidor dado
        if otro_servidor == servidor:
            continue

        # Si la distancia al otro servidor es menor que la distancia mínima
↪actual, actualizar el servidor más cercano y la distancia mínima
        if self.distancias[servidor][otro_servidor] < distancia_minima:
            servidor_mas_cercano = otro_servidor
            distancia_minima = self.distancias[servidor][otro_servidor]

    # Devolver el servidor más cercano
    return servidor_mas_cercano

```

1.2.1 Caso de ejemplo

```

[5]: import unittest
import json

def carga_dataset(data):
    with open(data) as f:
        test_datasets = json.load(f)

    videos = list()

```



```

for v in test_datasets["videos"]:
    v_obj = Video(v["name"], v["size"])
    for c, u in v["users"].items():
        v_obj.set_users(c, u)
    videos.append(v_obj)

servers = dict()
for s in test_datasets["servers"]:
    servers[s["country"]] = ServidorCache(s["identifier"], s["country"],
↪s["size"])

pings = test_datasets["pings"]
p_ = dict()
for s in servers.values():
    p_[s] = dict()
    for p in pings[s.country]:
        p_[s][servers[p]] = pings[s.country][p]
maestro = ServidorMaestro(servers.values(), p_)

return videos, servers, maestro

```

```

[6]: class TestBasico(unittest.TestCase):

    def test_carga_simple(self):

        v, s, m = carga_dataset("toy.json")

        spain = s["Spain"]
        spain.rellena(v)
        self.assertEqual(spain.tiempo_emision(), 578000)
        almacenados = spain.almacenados()
        self.assertIn((v[3], 0.5), almacenados)

        m.simplifica_grafo()
        self.assertEqual(m.mas_cercano(s["Spain"]), s["France"])
        m.simplifica_grafo()
        self.assertEqual(m.mas_cercano(s["Spain"]), s["France"])
        self.assertEqual(m.mas_cercano(s["France"]), s["Spain"])

if __name__ == "__main__":
    unittest.main(argv=['first-arg-is-ignored'], exit=False)

```

.

 Ran 1 test in 0.006s

OK

```
[7]: class TestBasico(unittest.TestCase):

    def test_carga_simple(self):

        v, s, m = carga_dataset("toy.json")

        spain = s["Spain"]
        spain.rellena(v)
        self.assertEqual(spain.tiempo_emision(), 578000)
        almacenados = spain.almacenados()
        self.assertIn((v[3], 0.5), almacenados)

        m.simplifica_grafo()
        self.assertEqual(m.mas_cercano(s["Spain"]), s["France"])
        m.simplifica_grafo()
        self.assertEqual(m.mas_cercano(s["Spain"]), s["France"])
        self.assertEqual(m.mas_cercano(s["France"]), s["Spain"])

if __name__ == "__main__":
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.
-----
```

Ran 1 test in 0.003s

OK

Tests Para probar que tu solución pasa los tests. Utilice el comando:

```
$ python tests-py3<version de python> <mi notebook>
```

Los tests necesitan de las librerías **networkx** y **nbformat**

```
$ pip install networkx nbformat
```

Explicación de los tests

- **test_ejemplo:** Es el mismo que el caso de ejemplo.
- **test_ej1_emision_correcta:** Comprueba que el tiempo de emisión del servidor caché es correcto.
- **test_ej1_sin_espacio:** Comprueba que ante un servidor sin espacio, el tiempo de emisión es 0.
- **test_ej1_espacio_infinito:** Comprueba que ante un servidor con espacio infinito, el tiempo de emisión es el máximo.
- **test_ej1_pais_no_existe:** Comprueba que ante pais que no tiene servidor cache, el tiempo de emisión es 0.
- **test_ej2_estructura_datos_mas_simple:** Comprueba que la estructura de datos que se utiliza para almacenar la red de servidores es más simple que la original.

- **test_ej2_red_servidores_consistente:** Comprueba que la red de servidores es consistente con el mapa original, es decir, no hay conexiones nuevas y los costes son los mismos.
 - **test_ej2_sistema_conexo:** Comprueba que la red de servidores cache es conexa.
-

1.2.2 Informe

Contesta a las siguientes preguntas.

Complejidad

1. Método `ServidorCache.rellena`
 - **Complejidad temporal:** en el peor de los casos $\rightarrow O(n^2)$ y en el mejor de los casos $\rightarrow O(n \log n)$
2. Método `ServidorMaestro.simplifica_grafo`
 - **Complejidad temporal:** en el peor de los casos $\rightarrow O(n^2)$ y en el mejor de los casos $\rightarrow O(n)$

Servidores cache.

- ¿La solución es óptima (maximiza siempre el tiempo de emisión) o es aproximada (encuentra un máximo local)?

Es una solución aproximada, porque se ordenan los videos según su valor, y luego se intenta almacenar tanto como sea posible en el servidor, priorizando los videos de mayor valor.

- ¿Qué ocurriría si solo se admitiese almacenar vídeos completos en cada servidor?

Si solo se admitiese almacenar videos completos en cada servidor el algoritmo tendrá que tener un par de modificaciones para asegurarse de que siempre quepan los videos completos en la capacidad disponible.

Red de servidores cache

- ¿La solución es óptima (la red es lo más simple posible) o es aproximada (encuentra un mínimo local)?

La solución es aproximada, porque no considera el impacto de cada seleccion en la estructura final del grafo, y no explora exhaustivamente todas las posibles configuraciones del grafo para garantizar que se ha obtenido la óptima.

- ¿Cómo afecta el número de conexiones entre servidores a la complejidad temporal del algoritmo empleado?

Cuanto mayor sea el número de conexiones entre servidores, la complejidad temporal también es mayor. Hay que tener en cuenta esta relación al diseñar los sistemas que involucren la gestión de redes de servidores cache y con ello buscar el algoritmo más eficiente de todos que pueda manejar los datos sin problema y de manera óptima.