PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

SECCIÓN ELECTRICIDAD Y ELECTRÓNICA

Arquitectura de Computadoras

Laboratorio N°9

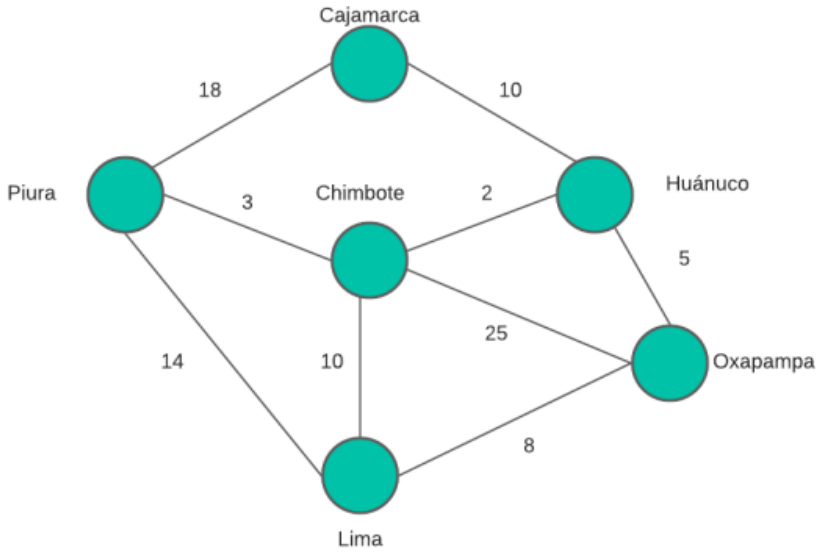
Nombre: Raúl Marcelo Samaniego Villarreyes

Código: 20181449

2022-2

Pregunta 2 **(4 puntos)**

Se tiene el siguiente gráfico, donde cada círculo representa una ciudad y las líneas representan las carreteras que unen a cada ciudad. Los números en cada línea representan la cantidad de horas que tomaría viajar por tierra entre cada ciudad.



Se desea averiguar cuál es el camino más corto para ir desde Lima (punto de partida) hasta Cajamarca (destino final).

Calcule el destino más corto usando el método de fuerza bruta: cree un thread por cada combinación posible y que cada thread calcule el tiempo del recorrido. Descargue la plantilla *pregunta2_lab9_plantilla.py* la cual contiene la lista de todas las combinaciones válidas posibles. Al final compare los tiempos de viaje para que su programa imprima la ruta más corta.

- La función implementada recibe la ruta y las distancias entre cada ciudad como argumentos. Luego, itera en el nombre la ruta que contiene implícitamente los pares de ciudades que recorre en cada tramo, para así obtener las distancias entre cada una y sumarlas de la siguiente manera:

```python
def calcular_distancia(ruta,distancias):
    ruta = ruta.split('_')
    dist = 0
    for i in range(1,len(ruta)):
        r = f"{ruta[i-1]}_{ruta[i]}"
        dist += distancias[r]
    return dist
```

- Una vez implementada la función, se procedió a elaborar los hilos para cada ruta:

```
dist = {}
thread1 = ReturnValueThread(target=calcular_distancia, args = (rutas_posibles[0],distancias))
thread2 = ReturnValueThread(target=calcular_distancia, args = (rutas_posibles[1],distancias))
thread3 = ReturnValueThread(target=calcular_distancia, args = (rutas_posibles[2],distancias))
thread4 = ReturnValueThread(target=calcular_distancia, args = (rutas_posibles[3],distancias))
thread5 = ReturnValueThread(target=calcular_distancia, args = (rutas_posibles[4],distancias))
thread6 = ReturnValueThread(target=calcular_distancia, args = (rutas_posibles[5],distancias))
thread7 = ReturnValueThread(target=calcular_distancia, args = (rutas_posibles[6],distancias))
thread8 = ReturnValueThread(target=calcular_distancia, args = (rutas_posibles[7],distancias))
thread9 = ReturnValueThread(target=calcular_distancia, args = (rutas_posibles[8],distancias))
thread10 = ReturnValueThread(target=calcular_distancia, args = (rutas_posibles[9],distancias))
```

- Cabe resaltar que se investigó la forma de obtener un hilo para función que retorna algún valor. Escogiendo como la mejor opción la siguiente clase que conserva la sintaxis convencional para crear hilos:

```
class ReturnValueThread(threading.Thread):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.result = None

    def run(self):
        if self._target is None:
            return  # could alternatively raise an exception, depends on the use case
        try:
            self.result = self._target(*self._args, **self._kwargs)
        except Exception as exc:
            print(f'{type(exc).__name__}: {exc}', file=sys.stderr)  # properly handle the exception

    def join(self, *args, **kwargs):
        super().join(*args, **kwargs)
        return self.result
```

- Una vez creado los hilos, se ejecutaron de la siguiente manera:

```
thread1.start()
thread2.start()
thread3.start()
thread4.start()
thread5.start()
thread6.start()
thread7.start()
thread8.start()
thread9.start()
thread10.start()
d1 = thread1.join()
d2 = thread2.join()
d3 = thread3.join()
d4 = thread4.join()
d5 = thread5.join()
```

```
    d6 = thread6.join()
    d7 = thread7.join()
    d8 = thread8.join()
    d9 = thread9.join()
    d10 = thread10.join()
    dist[rutas_posibles[0]] = d1
    dist[rutas_posibles[1]] = d2
    dist[rutas_posibles[2]] = d3
    dist[rutas_posibles[3]] = d4
    dist[rutas_posibles[4]] = d5
    dist[rutas_posibles[5]] = d6
    dist[rutas_posibles[6]] = d7
    dist[rutas_posibles[7]] = d8
    dist[rutas_posibles[8]] = d9
    dist[rutas_posibles[9]] = d10
    d = [d1,d2,d3,d4,d5,d6,d7,d8,d9,d10]
    #print(d)
    d.sort()
    #print(d)
    for i in dist:
        if dist[i] == d[0]:
            print(f"La ruta más corta es {i} con {d[0]} horas")
```

- Obteniendo en el terminal:

```
PS C:\PUCP\ARQUI\LAB9> python pregunta2_20181449.py
La ruta más corta es LIM_CHI_HUA_CAJ con 22 horas
PS C:\PUCP\ARQUI\LAB9>
```

Pregunta 3:

a. (3 puntos) Escriba una función que itere sobre todas las cadenas en la lista servidores_ntp[] para determinar cuál es el país cuya bolsa de valores está más próxima a abrir con respecto a nosotros(en Perú). Asuma que todas las bolsas de valores abren a las 08:00am en sus respectivos países.

La función implementada es:

```python
def calcular_diferencia(servidores):
    dif = {}
    sd = []
    for servidor in servidores:
        t = get_ntp_time(servidor)
        ahora = datetime.datetime.now()
        d = t - ahora
        dif[servidor] = d
        sd.append(d)
    sd.sort()
    for i in dif:
        if dif[i] == sd[0]:
            if i == '0.uk.pool.ntp.org':
                print(f"La bolsa más próxima a abrir es Londres con {sd[0]}")
            if i == '1.es.pool.ntp.org':
                print(f"La bolsa más próxima a abrir es Madrid con {sd[0]}")
            if i == '0.us.pool.ntp.org':
                print(f"La bolsa más próxima a abrir es Nueva York con {sd[0]}")
            if i == '0.hk.pool.ntp.org':
                print(f"La bolsa más próxima a abrir es Hong Kong con {sd[0]}")
            if i == '0.jp.pool.ntp.org':
                print(f"La bolsa más próxima a abrir es Tokyo con {sd[0]}")
```

- Obteniéndose:

```
PS C:\PUCP\ARQUI\LAB9> python pregunta3_20181449.py
Hora en UK: 2022-11-12 22:50:07
Hora en España: 2022-11-12 23:50:07
Hora en Estados Unidos: 2022-11-12 17:50:07
Hora en Hong Kong: 2022-11-13 06:50:07
Hora en Japón: 2022-11-13 07:50:08
La bolsa más próxima a abrir es Nueva York con 0:00:00.168871
El tiempo de ejecución sin threads es: 1.1226582527160645
```

b. (3 puntos) Haga lo mismo que en la parte a) pero en vez de iterar sobre la lista, use threads para cada uno de los elementos en la lista servidores_ntp[]. Una vez terminado los threads, su programa debe imprimir el país que ha calculado tener la hora más cercana. Imprima el tiempo de ejecución de esta función.

- Se hizo algunas modificaciones a la función anterior para ejecutar cada servidor dentro de un hilo:

```python
def calcular_diferencia_thread(servidor):
    dif = {}
    sd = []
    t = get_ntp_time(servidor)
    ahora = datetime.datetime.now()
    d = t - ahora
    return d
```

- Obteniéndose:

```
Hora en UK: 2022-11-12 22:52:03
Hora en España: 2022-11-12 23:52:03
Hora en Estados Unidos: 2022-11-12 17:52:03
Hora en Japón: 2022-11-13 07:52:03
Hora en Hong Kong: 2022-11-13 06:52:03
La bolsa más próxima a abrir es Nueva York con 0:00:00.371846
El tiempo de ejecución con threads es: 0.3269338607788086
```

Conclusiones:

- Los threads brindan gran ventaja al momento de ejecutar funciones recurrentes dentro de una iteración
- Con los threads se puede organizar las tareas o funciones que se requieran ejecutar en “paralelo”, estableciendo el inicio y punto de espera el final de ejecución de alguna tarea.