# Computación Paralela y Distribuída
# Proyecto 2022-I

**Prof. José Fiestas**
jfiestas@utec.edu.pe
UTEC

---

Instrucciones para desarrollar el proyecto:

- grupos de 3 estudiantes
- se presentan dos tipos de proyecto: **(A)** Desarrollo de código en paralelo y **(B)** Análisis de Performance
- debe escoger uno de los 4 proyectos propuestos. También puede proponer un nuevo proyecto de su interés. La calificación será sobre 20
- la entrega consiste en: el proyecto programado en C/C++, y un informe en LaTeX
- el informe escrito debe contener capítulos correspondientes a: **Introducción** (con la descripción del problema), **Método** (descripción del procedimiento), **Resultados** (presentación de partes relevantes del código, resultados y logros del proyecto) y **Conclusiones** (resumen del proyecto y propuestas de mejora)
- la presentación oral consistirá de 5 minutos por integrante, que escogerá algún tema del proyecto, utilizando una presentación o el mismo informe, y una pregunta final a cada estudiante.
- Entrega: 03.07., por Gradescope

Se utilizará la siguiente rúbrica para le calificación del proyecto

| Criterio | Puntaje |
|---|---|
| Desarrollo de código/PRAM | 6.0 |
| Optimización | 4.0 |
| Presentación escrita (informe) | 6.0 |
| Presentación oral | 4.0 |
| Total | 20.0 |

---

# 2 TSP (Proyecto tipo A)

Resuelva el problema del agente viajero en paralelo utilizando el método *branch and bound*
Se trata de un vendedor que debe minimizar el recorrido entre n ciudades. Puede empezar en cualquiera de ellas y terminar en la misma luego del recorrido. Cada ciudad debe ser visitada solo una vez.
Según el método de búsqueda primero en profundidad (DFS en inglés), se consideran las posibles ciudades que se pueden visitar desde una ciudad de partida. Asi sucesivamente, y calcular el camino mínimo de todas las posibilidades. Por ello, para n ciudades, obtenemos una cantidad total de caminos de (n-1)!
El algoritmo recursivo correspondiente es el siguiente:

```
DFS(camino){
if (camino tiene longitud n) {
.       if (camino es el mejor camino) {
.               mejor_camino = camino
.       }
.       else: {
.               for (para todos los caminos aún no recorridos i)
.                       DFS(camino ∪ i)
.               }
.      }
}
```

En el que **camino** es una estructura que contiene las ciudades a visitar.
En **branch and bound** se verifica si el camino encontrado hasta el momento, es mayor que el mejor camino encontrado. En caso lo sea, se detiene la búsqueda por ese camino.

```
BB(camino){
if (camino tiene longitud n) {
.       if (camino es el nuevo mejor camino) {
.               mejor_camino = camino
.       }
.       else: {
.               for (para todos los caminos aún no recorridos i)
.                       if (camino ∪ i es menor que el actual mejor camino )
.                               BB(camino ∪ i)
.               }
.      }
}
```

En su versión no-recursiva el algoritmo realiza lo siguiente:

```
camino={0}
push(camino)
while pop(camino){
if (camino tiene longitud n) {
.       if (camino es el nuevo mejor camino) {
.               mejor_camino = camino
.       }
.       else: {
.               for (para todos los caminos aún no recorridos i)
.                       if (camino ∪ i es menor que el actual mejor camino )
.                               push(camino ∪ i)
.               }
.   }
}
```

Para su implementación se utiliza una pila (stack), que se inicializa con la ciudad **0**. En cada iteración se retira la ciudad del tope de la pila. Si el camino tiene longitud n se evalua, de lo contrario se añaden ciudades si la longitud del camino no es mayor que el del mejor actual. El programa termina cuando la pila está vacía.

Considere el siguiente caso:
Una agencia de transporte de materiales, quiere optimizar sus operaciones en Lima. La agencia trabaja en los siguientes distritos: Lima Centro, Lince, Miraflores, Barranco, Rimac, Los Olivos, La Molina, La Victoria, Magdalena, San Borja.
Se necesita elaborar un software en paralelo que resuelva el problema TSP entre estos distritos de Lima. Para ello debe elaborar una matriz con distancias entre estos. Puede tomar distancias referenciales (e.g. desde Google Maps).

Puede escribir el código desde cero o utilizar algún código fuente para solucionar el problema, pero necesita validarlo, mostrando su precision antes de usarlo para el caso planteado.

El desarrollo de la solución del problema debe contener lo siguiente:
a) Desarrollar un código en C++, correspondiente al paradigma apropiado de paralelismo
b) Registro del desarrollo del código en por lo menos 3 pasos (versiones beta). En caso utilice una fuente externa, mostrar la validación del código
c) Utilice la matriz de distritos de Lima para implementar el problema y añada un factor adicional de costo, e.g. combustible, a los caminos trazados.
d) **Optimización:** Permita que el software sea interactivo para poder ingresar dos o mas ciudades y encontrar el camino óptimo entre ellas