Proyecto final: el planificador de entregas de PyEnvíos

Durante el curso resolviste por separado cada problema de PyEnvíos: buscar, medir, cargar la camioneta y moverse por la ciudad. En este proyecto los unes en una sola herramienta: un planificador que recibe las zonas del día y devuelve el orden de visita, su duración y qué tan lejos está del mejor orden posible. Se usa Python 3.11 o superior y solo la biblioteca estándar (`heapq`, `itertools`, `functools`, `math`); networkx y matplotlib se permiten únicamente para verificar y dibujar. El notebook `proyecto-rutas` contiene una solución de referencia completa: consúltalo al terminar, no antes.

Contexto

Cada mañana la camioneta de PyEnvíos sale del Almacén, entrega en varias zonas de la ciudad y regresa al Almacén. Hoy hay que visitar **las 11 zonas** restantes. El orden importa: la misma lista de entregas puede costar 130 minutos o 89 según cómo se recorra. Encontrar el orden óptimo es el problema del agente viajero (TSP), y con 11 paradas hay 11! = 39,916,800 órdenes posibles. Probarlos todos es lento; adivinar es caro. Tu planificador debe hacer algo mejor que ambas cosas y demostrarlo con cifras.

Objetivo

Construir y comparar cuatro estrategias para ordenar las entregas: el orden de la lista, el vecino más cercano, la mejora local 2-opt y el algoritmo exacto de Held-Karp (Bellman, y Held y Karp, 1962), que usa programación dinámica con memoización. El reporte final debe responder: ¿cuánto se pierde con cada heurística frente al óptimo y cuánto trabajo cuesta el óptimo?

Datos

El mapa es ficticio y está fijo: 12 zonas y 20 calles de doble sentido con su tiempo en minutos. Copia esta celda al inicio de tu notebook:

```
CIUDAD = [
    ("Almacén", "Centro", 7),
    ("Almacén", "Industrial", 4),
    ("Almacén", "Norte", 12),
    ("Centro", "Norte", 6),
    ("Centro", "Mercado", 3),
    ("Centro", "Universidad", 9),
    ("Industrial", "Puerto", 11),
```

```
    ("Industrial", "Mercado", 8),
    ("Mercado", "Universidad", 4),
    ("Mercado", "Parque", 6),
    ("Norte", "Colinas", 10),
    ("Universidad", "Colinas", 5),
    ("Universidad", "Hospital", 7),
    ("Parque", "Hospital", 3),
    ("Parque", "Puerto", 9),
    ("Hospital", "Estadio", 6),
    ("Colinas", "Estadio", 8),
    ("Puerto", "Estadio", 15),
    ("Norte", "Aeropuerto", 20),
    ("Colinas", "Aeropuerto", 9),
]

def grafo_ciudad():
    g = {}
    for a, b, w in CIUDAD:
        g.setdefault(a, []).append((b, w))
        g.setdefault(b, []).append((a, w))
    return g

paradas = sorted(z for z in grafo_ciudad() if z != "Almacén")
print(len(paradas))  # 11
print(paradas[0], paradas[-1])  # Aeropuerto Universidad
```

Para los extras se usan además los generadores del curso: `generar_paquetes()` (20 paquetes con peso y valor) y `generar_cuadricula()` (barrio de 30 × 30 con edificios).

Requisitos mínimos

1. **Matriz de distancias.** Implementa Dijkstra con `heapq` y ejecútalo desde cada una de las 12 zonas para obtener la matriz de minutos mínimos entre todos los pares. Verifica al menos tres distancias con `networkx`.
2. **Costo de una ruta.** Una función `costo_ruta(orden)` que sume los minutos de Almacén → paradas en ese orden → Almacén, usando la matriz.
3. **Línea base.** El costo de visitar las paradas en orden alfabético, tal como vienen en `paradas`.
4. **Vecino más cercano.** Desde el Almacén, ir siempre a la parada pendiente más cercana. Rompe los empates por nombre (`min` con la clave `(distancia, nombre)`) para que el resultado sea reproducible.

5. **2-opt.** Partiendo de la ruta del vecino más cercano, recorre todos los pares de posiciones i < j e invierte el tramo de i a j si eso reduce el costo; acepta cada mejora en cuanto aparece y repite pasadas completas hasta que ninguna inversión mejore.
6. **Held-Karp.** Implementa la recurrencia sobre estados (subconjunto de paradas visitadas, última parada) con `functools.lru_cache`, representando el subconjunto como máscara de bits. Reconstruye el orden óptimo y cuenta cuántos estados distintos se evaluaron.
7. **Validación con fuerza bruta.** En el caso pequeño de 6 paradas (Mercado, Hospital, Colinas, Puerto, Aeropuerto, Estadio), compara Held-Karp con `itertools.permutations`: ambos deben dar el mismo costo.
8. **Comparación y reporte.** Una tabla con cada estrategia, su ruta, sus minutos, su porcentaje sobre el óptimo y su trabajo (rutas evaluadas, pasadas o estados).

Cifras esperadas

Tu solución debe reproducir estos valores:

- Desde el Almacén: Industrial 4, Centro 7, Mercado 10, Norte 12, Universidad 14, Puerto 15, Parque 16, Colinas 19, Hospital 19, Estadio 25 y Aeropuerto 28 minutos. El par más distante es Puerto–Aeropuerto, a 32 minutos.
- Orden alfabético: 166 minutos (86.5 % por encima del óptimo).
- Vecino más cercano: 130 minutos, con el orden Industrial, Mercado, Centro, Norte, Colinas, Universidad, Hospital, Parque, Estadio, Puerto, Aeropuerto.
- 2-opt sobre esa ruta: 89 minutos en 3 pasadas (la tercera ya no mejora nada), con el orden Industrial, Puerto, Parque, Hospital, Estadio, Aeropuerto, Colinas, Universidad, Mercado, Centro, Norte.
- Held-Karp: óptimo exacto de 89 minutos, la misma ruta, con 11,264 estados evaluados (11×2^{10}).
- Vecino más cercano queda 46.1 % por encima del óptimo; 2-opt, 0.0 %.
- Caso de 6 paradas: 720 rutas por fuerza bruta, óptimo de 78 minutos (Mercado, Colinas, Aeropuerto, Estadio, Hospital, Puerto); Held-Karp lo alcanza con 192 estados. El vecino más cercano da 105 minutos.

La lectura que se espera

El vecino más cercano es rápido y fácil de explicar, pero su decisión miope al final de la ruta (ir a Puerto y luego al Aeropuerto, el par más lejano de la ciudad) le cuesta 41 minutos de más. Una sola mejora local, 2-opt, recupera todo en este mapa; un buen reporte aclara que eso es suerte del caso y no una garantía: 2-opt se detiene en un óptimo local. Held-Karp sí garantiza el óptimo y lo logra con 11,264 estados frente a 39,916,800 rutas, pero su costo crece como $O(n^2 \cdot 2^n)$: con 20 paradas ya son

10,485,760 estados y con 30, más de 16 mil millones. El planificador real combina ambas ideas: exacto mientras se pueda, heurística verificada cuando no.

Extras opcionales

- **Ventanas de horario.** Asigna a algunas zonas una hora límite de entrega (por ejemplo, Hospital antes del minuto 40) y adapta el vecino más cercano o Held-Karp para descartar rutas que la incumplan. Reporta cuánto se alarga la ruta.
- **Capacidad con mochila.** Con `generar_paquetes()` y una camioneta de 50 kg, decide qué paquetes salen en el primer viaje con programación dinámica. Valor esperado: 10,581 MXN con 11 paquetes y 50 kg exactos.
- **Última milla con A*.** Dentro de la zona de destino, usa `generar_cuadricula()` y A* con distancia Manhattan para ir de (0, 0) a (29, 29). Valor esperado: 58 pasos con 282 celdas expandidas, contra 635 de Dijkstra.
- **Escalamiento.** Mide cuántos estados evalúa Held-Karp con 6, 8, 10 y 11 paradas y grafica el crecimiento junto a n!. Valores esperados: 192, 1,024, 5,120 y 11,264 estados, contra 720, 40,320, 3,628,800 y 39,916,800 rutas.
- **Dibujo.** Dibuja el mapa con networkx y resalta la ruta óptima.

Entregables

1. **Un notebook** que corra de principio a fin sin errores, con la estructura: datos, matriz de distancias, estrategias, validación, comparación y conclusiones.
2. **Una tabla de resultados** con estrategia, ruta, minutos, porcentaje sobre el óptimo y trabajo realizado.
3. **Un resumen** de media página para el jefe de operaciones de PyEnvíos: qué estrategia usar hoy, cuánto tiempo ahorra frente a la ruta actual y hasta cuántas paradas el método exacto sigue siendo práctico.

Rúbrica (100 puntos)

- **Matriz de distancias (15 puntos):** Dijkstra propio con `heapq`, matriz completa y verificación con networkx.
- **Heurísticas (20 puntos):** línea base, vecino más cercano con desempate reproducible y 2-opt correcto, con las cifras esperadas.
- **Held-Karp (25 puntos):** recurrencia correcta con máscara de bits y memoización, reconstrucción de la ruta y conteo de estados.
- **Validación (10 puntos):** fuerza bruta en 6 paradas que coincide con Held-Karp.
- **Análisis de complejidad (15 puntos):** conteos de trabajo de cada estrategia y explicación de por qué Held-Karp no escala a 50 paradas.
- **Comunicación (15 puntos):** tabla clara, resumen ejecutivo con cifras y conclusiones honestas sobre lo que 2-opt garantiza y lo que no.

Los extras suman hasta 10 puntos adicionales y no sustituyen a los requisitos mínimos.