**NOMBRE:**

**1** **2 puntos**

El problema de la búsqueda de un ciclo hamiltoniano se puede resolver mediante la técnica de Vuelta Atrás. Como el ciclo hamiltoniano puede empezar en cualquier vértice, es una práctica habitual empezar la comprobación por el primer vértice del grafo (asumimos que es el 0). Instancia los métodos siguientes para resolver el problema, utilizando una representación de grafo en forma de listas de adyacencia:

```
self.is_feasible(s)
self.branch(s)
self.is_promising(s)
```

Te proporcionamos el código del esquema general de Vuelta Atrás:

```
def backtracking(self, s):
  """Explora el subarbol que tiene por raiz a s y devuelve la primera
  solucion factible que encuentra (si la hay). Si s no contiene al
  menos una solucion factible, devuelve None"""
  if self.is_complete(s):
    if self.is_feasible(s):
      return s
  else:
    for child in self.branch(s):
      if self.is_promising(child):
        found = self.backtracking(child)
        if found != None:
          return found
    return None

def solve(self, s):
  return self.backtracking([0])

def is_complete(self, s):
  return len(s)==len(self.G)+1
```

y un ejemplo de grafo dirigido representado mediante listas de adyacencia:

```
self.G = [[1,2,3],       # del vertice 0 vamos a los vertices 1,2,3
          [0,3,4],       # del vertice 1
          [0,3,5],       # del vertice 2
          [0,1,2,4,5,6], # del vertice 3
          [1,3,7],       # del vertice 4
          [2,3,6,8],     # del vertice 5
          [3,5,7,8,9],   # del vertice 6
          [4,6,9],       # del vertice 7
          [5,6,9],       # del vertice 8
          [6,7,8]]       # del vertice 9
```

**Solución:**

```
def is_feasible(self, s):
  return s[0] == s[-1]

def branch(self, s):
  return [s+[v] for v in self.G[s[-1]]]

def is_promising(self, s):
  return self.is_complete(s) or s[-1] not in s[:-1]
```

**2** **1 punto**

Responde brevemente a las siguientes cuestiones:

1) ¿Cuáles son las **diferencias** fundamentales entre los esquemas de Vuelta Atrás y Ramificación y Poda?
   Tal y como se han visto los algoritmos en clase, RP resuelve problemas de optimización, mientras que VA no. Ambos esquemas se basan en la exploración del árbol de estados, aunque VA lo hace por primero en profundidad y RP por primero el mejor (con una función de puntuación que suele ser la cota optimista). Además, en VA sólo se mantiene activo el estado que se va construyendo hacia la solución, mientras que en RP hay un conjunto de estados activos, del que hay que elegir el mejor para explorar. Por último, RP proporciona criterios de poda diferentes a la poda por factibilidad que propone VA como, por ejemplo, poda por cota optimista.
2) ¿Puedes utilizar el esquema de Vuelta Atrás para resolver un **problema de optimización**?
   Sí, únicamente hay que explorar el árbol de estados en su totalidad, para encontrar todas las soluciones y devolver la mejor. En su forma habitual, en cuanto se encuentra una solución, el algoritmo de VA la devuelve y finaliza. Para saber cual es la óptima, hay que explorarlas todas manteniendo la mejor.
3) ¿Puedes utilizar el esquema de Ramificación y Poda para resolver un **problema de no optimización**?
   Sí, se puede emular el esquema de VA utilizando RP, cambiando el concepto de función de puntuación y cota optimista. Habría que puntuar los estados según su profundidad en el árbol, para seleccionar primero los más cercanos a las hojas. La función de cota sólo podría servir para podar estados no factibles. Y el algoritmo terminaría en cuanto se encontrara la primera solución.

**3** **3 puntos**

En una fábrica, hay $N$ productos que pueden ser fabricados por $M$ máquinas diferentes, si bien tardan un tiempo distinto en fabricarlo, y cada máquina puede fabricar $p_i$ productos como máximo. Una asignación de $N$ productos a las $M$ máquinas se puede representar mediante un vector $x$ de $N$componentes, $(x_1, x_2, \ldots, x_N)$ donde $x_i = j$ indica que el producto $i$-ésimo se ha asignado a la máquina $j$, $1 \leq j \leq M$. Nos interesa minimizar el coste total en tiempo. Por ejemplo, la siguiente asignación de 5 productos a 3 máquinas (2, 2, 1, 3, 1) con la siguiente tabla de tiempos $t_{i,j}$ que indica el tiempo que la máquina $i$ tarda en finalizar el trabajo $j$, y asumiendo que cada máquina no puede fabricar más de 2 productos:

| | T1 | T2 | T3 | T4 | T5 |
|---|---|---|---|---|---|
| M1 | 4 | 4 | 3 | 5 | 6 |
| M2 | 2 | 3 | 4 | 3 | 4 |
| M3 | 5 | 2 | 4 | 4 | 3 |

genera un coste de $2 + 3 + 3 + 4 + 6 = 18$. Realiza una traza de un algoritmo de Ramificación y Poda sobre la instancia presentada basada en *poda explícita* que calcule la asignación de productos a máquinas que minimice el tiempo total. Explica la cota que utilizas. La traza se debe mostrar como el *conjunto de estados activos* de cada iteración del algoritmo indicando la cota optimista de cada estado (ej: como superíndice) y subrayando el estado que se selecciona para la siguiente iteración. Hay que indicar también si se actualiza la variable mejor solución y la poda explícita. En caso de inicializar la mejor solución inicial a una solución arbitraria, debes indicar el algoritmo que usas y su coste.

**Solución:** Se puede utilizar como cota optimista el tiempo acumulado en el estado más una estimación optimista de lo que falta: el mínimo tiempo que tardan las tareas aún no asignadas, independientemente de que la máquina correspondiente ya tenga el máximo número de tareas que puede procesar. Se puede hacer un preproceso para obtener ese valor y calcularse incrementalmente con respecto al valor del estado padre en tiempo constante.

Supongamos que no utilizamos una solución arbitraria para inicializar la variable $x$ y $fx$ que contiene la mejor solución y su coste alcanzada hasta el momento. En ese caso:

- $A_0 = \{\underline{(?)^{13}}\}$
  $x = None, fx = +\text{ínf}$
- $A_1 = \{(1,?)^{15}, \underline{(2,?)^{13}}, (3,?)^{16}\}$
- $A_2 = \{(1,?)^{15}, (2,1,?)^{15}, (2,2,?)^{14}, \underline{(2,3,?)^{13}}, (3,?)^{16}\}$
- $A_3 = \{(1,?)^{15}, (2,1,?)^{15}, (2,2,?)^{14}, \underline{(2,3,1,?)^{13}}, (2,3,2,?)^{14}, (2,3,3,?)^{14}, (3,?)^{16}\ \}$

- $A_4 = \{(1,?)^{15}, (2,1,?)^{15}, (2,2,?)^{14}, (2,3,1,1,?)^{15}, \underline{(2,3,1,2,?)^{13}}, (2,3,1,3,?)^{14}, (2,3,2,?)^{14}, (2,3,3,?)^{14}, (3,?)^{16}$
- En la siguiente iteración se obtienen las primeras soluciones: $(2,3,1,2,1)$ con un coste de 16 y $(2,3,1,2,3)$ con un coste de 13. Se desecha la de peor coste. (Adicionalmente se genera la solución no factible $(2,3,1,2,2)$.) Se actualizan las variables: $x = (2,3,1,2,3)$ y $fx = 13$. Se entra en el proceso de poda explícita, podando aquellos estados con un valor de cota mayor o igual a 13:

  $A_5 = \{\cancel{(1,?)^{15}}, \cancel{(2,1,?)^{15}}, \cancel{(2,2,?)^{14}}, \cancel{(2,3,1,1,?)^{15}}, \cancel{(2,3,1,3,?)^{14}}, \cancel{(2,3,2,?)^{14}}, \cancel{(2,3,3,?)^{14}}, \cancel{(3,?)^{16}}\ \}$

  Es decir, se eliminan todos los estados, así que el algoritmo termina, la mejor solución tiene valor 13.

Se podría utilizar para inicializar el solución inicial varias ideas basadas en algoritmos voraces:

1. Asignar las tareas a la primera máquina que esté libre, coste $O(N)$. Para la instancia presentada: $x = (1,1,2,2,3)$, $fx = 18$.

2. Asignar las tareas a las máquinas en orden, hasta que se alcance el máximo, coste $O(N)$. Para la instancia presentada: $x = (1,2,3,1,2)$, $fx = 20$.

3. Asignar las tareas a la máquina en que menor tiempo se procese, siempre que esa máquina no tenga el máximo número de tareas asignadas. Si eso ocurre, se pasa a la máquina libre cuyo coste sea menor, coste $O(NM)$. Para la instancia presentada: $x = (2,3,1,2,3)$, $fx = 13$.

Las trazas utilizando las soluciones iniciales dadas por los dos primeros algoritmos son muy similares a la ya dada, excepto que la inicialización de $x$ y $fx$ cambia.

Hagamos la traza para el supuesto de usar el tercer algoritmo con la solución inicial $x = (2,3,1,2,3)$, $fx = 13$.

- $A_0 = \{\underline{(?)^{13}}\}$
  $x = (2,3,1,2,3)$, $fx = 13$.

  El estado inicial, que contiene todas las soluciones factibles, tiene un valor asociado de cota optimista igual a 13, y ya tenemos una solución cuyo valor es 13. Por tanto, el estado inicial se puede podar, y el algoritmo termina devolviendo la solución óptima, que ha resultado ser para esta instancia la que se ha obtenido con un algoritmo voraz.

**4** **4 puntos**

Tenemos un talonario de millas aéreas acumuladas que va a caducar próximamente y queremos planificar un viaje para utilizarlo en su totalidad. El talonario nos permite realizar $N$ vuelos, independientemente de la distancia de este vuelo. Las restricciones que nos impone la compañía son:

- sólo podemos utilizar vuelos directos (nos han proporcionado el grafo de conexiones de la compañía, $conexiones[i,j] = conexiones[j,i] = m$ indica que las ciudades $i$ y $j$ están conectadas por vuelo directo a una distancia de $m$ millas; si $conexiones[i,j] = -1$, indica que las ciudades no están conectadas),
- debemos partir de nuestra ciudad origen, Valencia, y volver de nuevo a Valencia, y
- no podemos volar a una ciudad más de una vez (excepto a Valencia, de la que debemos partir y llegar al final).

Las ciudades a las que vuela la compañía están numeradas de 0 a $C$ (la ciudad 0 es Valencia). Nuestro objetivo es, con las restricciones impuestas por la compañía, volar el máximo de millas. Diseña un algoritmo de Ramificación y Poda para maximizar las millas. Para ello:

a) Expresa formalmente el conjunto de soluciones factibles, la función objetivo a maximizar y la solución óptima buscada.
b) Describe los siguientes conceptos sobre los estados que serán necesarios para el algoritmo:
   1) Representación de un estado (no terminal) y su coste espacial.
   2) Condición para que un estado sea solución.
   3) Identifica el estado inicial que representa todo el conjunto de soluciones factibles.

c) Define una función de ramificación. Analiza su coste temporal.
d) Diseña una cota optimista no trivial. Estudia cómo se puede realizar el cálculo de la cota de forma eficiente. ¿Cuál sería su coste temporal? Justifícalo.

**Solución:** El conjunto de soluciones factibles serán todos los trayectos que comienzan en Valencia, vuelan a $N-1$ ciudades conectadas entre sí y se regresa a Valencia y que se pueden modelar como una secuencia de ciudades de la manera siguiente:

$$X = \{(x_0, x_1, \ldots, x_N) \in \{0, \ldots, C\}^N \mid x_0 = x_N = 0, x_i \neq x_j \ \forall 0 \leq i < j \leq N-1, \\ conexiones[x_i, x_{i+1}] \neq -1 \ \forall 0 \leq i \leq N-1\}$$

La función objetivo a maximizar es:

$$f((x_0, \ldots, x_N)) = \sum_{i=0}^{N-1} conexiones(x_i, x_{i+1})$$

Y la solución óptima buscada es:

$$\hat{x} = \underset{x \in X}{\operatorname{argmax}} f(x)$$

Una solución parcial puede representarse como un prefijo de una solución completa. Es decir, mediante una tupla $(x_0, x_1, \ldots, x_k, ?)$ con $k < N$, su coste espacial es lineal con la longitud de la misma (una cota superior puede ser $O(N)$), cumpliéndose las restricciones de $X$ para la parte del trayecto que llevamos hasta el momento.

Para que un estado $(x_0, x_1, \ldots, x_k, ?)$ sea solución basta con que $k = N$ y, evidentemente, que se cumplan las restricciones.

El estado inicial que representa implícitamente todo el conjunto de soluciones factibles se puede representar con la secuencia $(0, ?)$, pues es una condición que todos los trayectos comiencen en Valencia.

La función de ramificación, para los estados $(x_0, x_1, \ldots, x_k, ?)$, con $k < N-1$, se puede expresar como:

$$branch((x_0, x_1, \ldots, x_k, ?)) =$$

$$\{(x_0, x_1, \ldots, x_k, x_{k+1}, ?) \,|\, x_{k+1} \in \{1, \ldots, C\}, x_{k+1} \neq x_i \forall 0 \leq i \leq k, conexiones[x_k, x_{k+1}] \neq -1\}$$

Es decir, basta con añadir una nueva ciudad adyacente a la última en la tupla de modo que esa ciudad no coincida con ninguna ciudad visitada previamente.

Si estamos ramificando un estado $(x_0, x_1, \ldots, x_k, ?)$, con $k = N-1$ se obtendrá la solución:

$$branch((x_0, x_1, \ldots, x_k, ?)) = \{(x_0, x_1, \ldots, x_k, 0) \,|\, conexiones[x_k, 0] \neq -1\}$$

El coste temporal de generar un estado hijo es, como poco, el coste de generarlo (escribirlo), si bien hay un coste asociado a los estados no prometedores que finalmente ni se generan. El coste de ramificar todos los hijos se puede acotar por $O(C \times N)$.

Una cota optimista no trivial (es decir, que depende de la solución parcial particular) sería sumar a las millas ya recorridas de la parte conocida una estimación optimista de lo que queda por recorrer. Para un estado de la forma $(x_0, x_1, \ldots, x_k, ?)$ quedan por usar $N-k$ bonos y podemos considerar que se seleccionan los vuelos de más millas, independientemente de que sean a ciudades ya visitadas o formen un trayecto total conectado:

$$opt((x_0, x_1, \ldots, x_k)) = \sum_{i=0}^{k-1} conexiones(x_i, x_{i+1}) + \sum_{i=1}^{N-k} millas[i]$$

El vector $millas[1]$ contendrá las millas del vuelo con más millas, así $millas[i]$ contendrá correspondientes a las millas del $i$-ésimo vuelo de más millas. La manera más eficiente de calcular esta cota es hacerla incremental:

$$opt((x_0, x_1, \ldots, x_{k+1})) = opt((x_0, x_1, \ldots, x_k)) - millas(k) + conexiones(x_k, x_{k+1})$$

De esta manera, el coste temporal pasaría a constante.