RutaEficiente

Tomás quiere viajar de Buenos Aires a Mar del Plata en su flamante Renault 12. Como está preocupado por la autonomía de su vehículo, se tomó el tiempo de anotar las distintas estaciones de servicio que se encuentran en el camino. Modeló el mismo como un segmento de 0 a M, donde Buenos aires está en el kilómetro 0, Mar del Plata en el M, y las distintas estaciones de servicio están ubicadas en los kilómetros $0 = x_1 \leq x_2 \leq \ldots x_n \leq M$. Razonablemente, Tomás quiere minimizar la cantidad de paradas para cargar nafta. Él sabe que su auto es capaz de hacer hasta C kilómetros con el tanque lleno, y que al comenzar el viaje este está vacío.

a) Proponer un algoritmo *greedy* que indique cuál es la cantidad mínima de paradas para cargar nafta que debe hacer Tomás, y que aparte devuelva el conjunto de estaciones en las que hay que detenerse. Probar su correctitud.
b) Dar una implementación de complejidad temporal $O(n)$ del algoritmo del inciso a).

Algoritmo, implementado en Python, con complejidad temporal $O(n)$.

```
def f(estaciones: [int], tanque: int):
  estaciones = [0] + estaciones
  n = len(estaciones)
  s = []
  i = 0
  combustible = tanque
  # Invariante: Puedo llegar a la estación `i`-ésima,
  # terminando con `combustible` en el tanque.
  while i < n - 1:
    delta = estaciones[i+1] - estaciones[i]
```

```
      if delta > combustible: # Si no puedo llegar con el combustible
  actual:
        combustible = tanque # Cargo.
        combustible -= delta # Uso para llegar a la próxima estación.
        if combustible < 0: return None
        s.append(i)
      else:
        combustible -= delta # Uso para llegar a la próxima estación.
      i += 1 # Avanzo a la próxima estación.
    return s
```

Demostración de correctitud

Definición. Una **solución** X es un conjunto de n booleanos booleanos, donde X_j es True si elegimos la j-ésima estación, y False si no.
Definición. Dada una solución X, definimos una **sub-solución** Y de X cuando Y es un prefijo de X. Escribimos eso $Y \sqsubset X$. **Definición**. Una solución o sub-solución es **válida** cuando no nos quedamos sin combustible en ningún momento. Llamamos a una solución válida X **óptima** cuando tiene el mínimo número de Trues, entre todas las soluciones válidas. Decimos que una sub-solución Y de X es **i-extensible** a X cuando $|Y| = n - i$. Es decir, Y se puede convertir en X agregándole los últimos i elementos de X. Escribimos eso $Y \sqsubset_i X$. Vemos que $Y \sqsubset_0 X \iff Y = X$.

El invariante del ciclo es $P(i) =$ "$0 \leq i \leq n - 1$, y notando S_i es la solución representada por `c`, entonces S_i es una i-tupla de booleanos, es una forma válida de llegar a la estación con índice i, y si el problema tiene alguna solución válida, entonces existe una solución óptima $S*$, tal que S_i es $(n - i)$-extensible a S^*".

La variante del ciclo es $v(i) = (n - 1) - i$, y la guarda es $g(i) = i < n - 1$. La precondición es $S_0 = []$. La postcondición es que `s` representa una sub-solución S_{n-1} que es 1-extensible a una solución óptima, o devolvemos `None`.

El teorema del invariante nos pide que probemos que:

1. Vale la precondición.
2. La precondición y $g(0)$ implican $P(0)$.
3. La variante v decrece hasta cero.
4. Cuando $v(i) = 0$, tenemos que $\neg g(i)$.
5. El invariante y la negación de la guarda implican la postcondición.
6. $\forall i \in \mathbb{N}.\ ((g(i) \land P(i))) \Rightarrow P(i+1))$.

Demostremos una por una.

1. Vale la precondición: Trivial.
2. La precondición y $g(0)$ implican $P(0)$: El programa comienza con `s = []`, luego $S_0 = []$. El conjunto de soluciones válidas es finito porque es un subconjunto del conjunto de n-tupla de booleanos, y como $g(0)$, entonces $0 < n - 1 \iff 1 < n \Rightarrow n \geq 1$, luego el conjunto de soluciones no es vacío. Luego si existe al menos una solución válida, existe al menos una solución válida que es óptima. Sea S^* cualquier tal solución óptima. La lista vacía es un prefijo de toda otra lista. Luego, si existe alguna solución válida S^*, entonces $S_0 \sqsubset_n S^*$, que es lo que queríamos demostrar. Finalmente, la primer estación está en la posición 0, luego para cualquier tamaño del tanque en $\mathbb{N}$, podemos llegar cargando en ningún lugar, a la primer estación.
3. En cada iteración, aumentamos i en 1, incondicionalmente. Luego, $v(i) = (n-1) - i$ decrece en cada iteración, y eventualmente se hace cero, porque los naturales son discretos.
4. Si $v(i) = 0$, entonces $(n-1) - i = 0 \iff i = n - 1$, y luego no vale que $i < n - 1$. Luego, vale que $\neg g(i)$.
5. Si $\neg g(i)$, sabemos que $i \geq n - 1$. Por el invariante, sabemos que $i \leq n - 1$. Luego $i = n - 1$. Por el invariante, sabemos que si el problema tiene alguna solución válida, entonces existe una solución óptima S^* tal que $S_i \sqsubset_{n-i} S^*$, pero como $i = n - 1$, entonces $n - (n-1) = 1$, y $S_i \sqsubset_1 S^*$, y por lo tanto S_i es 1-extensible a una solución óptima S^*. Luego `s`, que representa S_i, es 1-extensible a una solución óptima, que es la

postcondición. Si el problema no tiene una solución válida, no podríamos haber creado `s` sin salir del ciclo con `return None`, porque $S_i = S_{n-1}$ es una forma válida de llegar a la estación $n-1$, y S_{n-1} + (False,) es una solución válida.

6. Sabemos que vale $g(i)$ y $P(i)$. Que $0 \leq i+1 \leq n-1$ es fácil de ver, puesto que por la guarda sabemos que $i < n-1$, por el invariante sabemos que $0 \leq i$, y aumentamos i en 1. Por el resto de $P(i)$, si existe alguna solución válida, entonces existe una solución óptima S^*, tal que `s` representa S_i, y $S_i \sqsubset_{n-i} S^*$. En la i-ésima iteración, decimos `delta = estaciones[i+1] - estaciones[i]`. A `s` le agregamos o no `i`, dependiendo de si `delta > combustible` o `delta <= combustible`. Como representamos S_i, esto significa que S_{i+1} es S_i + (False,) o S_i + (True,), respectivamente. Partamos en los casos que parte nuestro algoritmo:

   1. Si `delta > combustible`: De existir una solución válida, teníamos una solución S^* que tiene a S_i como prefijo, luego al llegar a la estación i, S^* tiene el mismo `combustible` que nuestra sub-solución S_i. Como `delta > combustible`, la solución óptima no puede llegar a la estación $i+1$ sin cargar combustible, luego tiene que ser cierto que S_i + (True,) es prefijo de S^*. Pueden pasar dos cosas:

      1. Si `tanque - delta >= 0`, decimos $S_{i+1} = S_i$ + (True,), entonces S^* puede cargar acá, y termina con `combustible = tanque - delta` al llegar a la estación $i+1$. Luego, tenemos que S_i + (True,) $\sqsubset_{n-(i+1)} S^*$ y S_{i+1} es una forma válida de llegar a la estación $i+1$.

      2. Si `tanque - delta < 0`, entonces aún cargando en esta estación, S^* no podría llegar a la próxima. Luego, S^* no existe, y nuestro algoritmo correctamente devuelve `None`.

      En ambos casos, vale el invariante $P(i+1)$.

   2. Si `delta < combustible`, entonces nuestra solución puede llegar a la estación $i+1$ sin cargar más combustible. Nuestro código no agrega nada a `s`, luego esto representa la sub-solución $S_{i+1} = S_i$ + (False,). Mirando a S^*, pueden pasar dos cosas, que el próximo ($i+1$) elemento de S^* sea True, o que sea False. Si es False, entonces tenemos $S_{i+1} \sqsubset_{n-(i+1)} S^*$, y estamos. Si no, el próximo ($i+1$) elemento de S^* es True, que significa que S^* usó a la i-ésima estación para llegar a la estación $i+1$, y nuestra solución, S_{i+1}, no. Tenemos que encontrar *otra* solución óptima, $S^{*\prime}$,

tal que $S_{i+1} \sqsubset_{n-(i+1)} S^{*'}$.

Veamos qué pasa, dependiendo de si S^* cargó alguna vez después que en la estación i, o no.

1. Si existe un primer $j > i+1$ tal que $S^*_j =$ True. Entonces sabemos que `estaciones[j] - estaciones[i] <= tanque`, dado que j es el *primer* tal índice, y que S^* es válida, luego j es la primer estación en la que necesitó cargar, y llegó a la estación j sin cargar . Luego, como `estaciones[i+1] >= estaciones[i]`, tenemos que `estaciones[j] - estaciones[i+1] <= tanque` también. Consideremos entonces $S^{*'}$, definida como S^* en todas partes, excepto que $S^{*'}_{i+1} =$ False, y $S^{*'}_{i+2} =$ True. Notar que $i+2$ existe, puesto que $j > i+1$, luego $j \geq i+2$. Esta modificación significa no cargar en la estación i (llegando a $i+1$) y sí cargar en la estación $i+1$ (llegando a $i+2$). Queremos ver tres cosas, que $S_{i+1} \sqsubset_{n-(i+1)} S^{*'}$, que $S^{*'}$ es válida, y que $S^{*'}$ es óptima.
    1. Claramente $S_i \sqsubset_{n-i} S^{*'}$, porque $S_i \sqsubset_{n-i} S^*$, y $S^{*'}$ tiene los mismos primeros i elementos que S^*. Asimismo, definimos $S^{*'}_{i+1} =$ False, que es igual a $S_{i+1 i+1}$, luego S_{i+1} y $S^{*'}$ son iguales en los primeros $i+1$ elementos, y luego $S_{i+1} \sqsubset_{n-(i+1)} S^*$.
    2. Como `delta < combustible`, podemos llegar a la estación $i+1$ sin cargar más, luego el prefijo de longitud $i+1$ de $S^{*'}$ es válido. Cargando en $i+1$, como `estaciones[j] - estaciones[i+1] <= tanque`, podemos seguir usando j para cargar luego de cargar en $i+1$, y no nos quedamos sin combustible. Luego de cargar en j, como S^* ya había cargado en j, se puede continuar hasta el final. Luego, nuestro $S^{*'}$ puede llegar desde la primer estación hasta la i con S_i, de la i sin cargar a la $i+1$, cargando en $i+1$, y de ahí a la j, y de la j hasta el final usando el sufijo de S^*. Luego $S^{*'}$ es una solución válida.
    3. Notemos que, como sabemos que $S^*_{i+1} =$ True, estamos sacando un True de la i-ésima posición, y a lo sumo estamos poniendo un True en la posición $i+2$ (a lo sumo porque quizás S^*_{i+2} ya era True). Luego, a lo sumo $S^{*'}$ tiene el mismo número de Trues que S^*, y como S^* era óptima (tiene el mínimo número de Trues), luego $S^{*'}$ también lo es.

2. Si $S_j^* =$ False para todo $j > i+1$. Entonces i es la última estación donde cargó S^*, pero podía llegar a la próxima sin cargar, porque `delta < combustible`. Luego, reemplazando cargar en la i-ésima estación, con cargar en la $(i+1)$-ésima estación (o no cargar, si $i+1 = n-1$, dado que habríamos llegado al final del recorrido), tenemos otra solución $S^{*'}$, que también permite llegar al final del recorrido, con a lo sumo el mismo número de cargas, y más aún, S_{i+1} es $(i+1)$-extensible a esa $S^{*'}$, pues ahora comparten un elemento más que entre S_i y S^*.

Luego, si existe S^* tal que $S_i \sqsubset_{n-i} S^*$, entonces existe $S^{*'}$ tal que $S_{i+1} \sqsubset_{n-(i+1)} S^{*'}$. Esto prueba $P(i+1)$.

Luego, por el teorema del invariante, al terminar el ciclo vale la postcondición. Basta ver que una sub-solución 1-extensible de una solución óptima representa en realidad la misma solución óptima, dado que el último elemento de una solución óptima siempre es False - nunca vamos a cargar en la última estación, si ya pudimos llegar a la misma. Luego al devolver la lista `s`, estamos devolviendo precísamente una solución óptima, que es lo que queríamos demostrar.