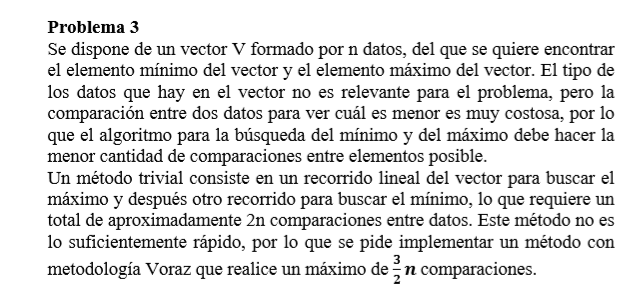
PL2

JAVIER MARTÍN GÓMEZ

47231977M



Para este problema tenemos que realizar un algoritmo que devuelva el elemento máximo y mínimo de una lista con la condición de que el coste máximo sea de 3n/2 por lo que no se puede recorrer la lista dos veces para encontrar ambos valores. Para ello, tenemos que recorrer la mitad de la longitud de la lista (n/2) tres veces, por lo que la suma total del recorrido será 3n/2 (nuestro objetivo). En el algoritmo implementado en Python, se muestra como se leen tres veces la mitad de la longitud de la lista.

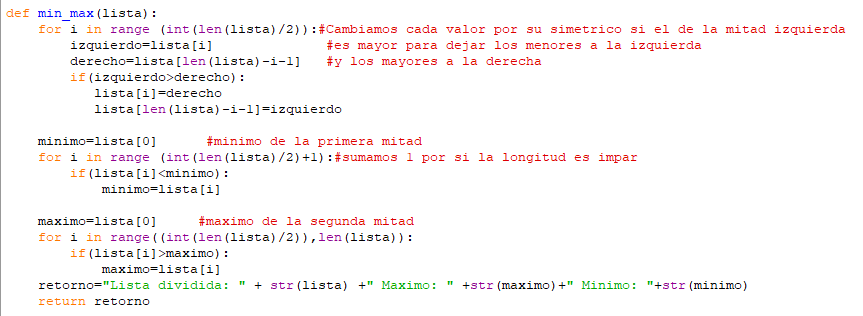
Para conseguir esto, realizamos los siguientes pasos:

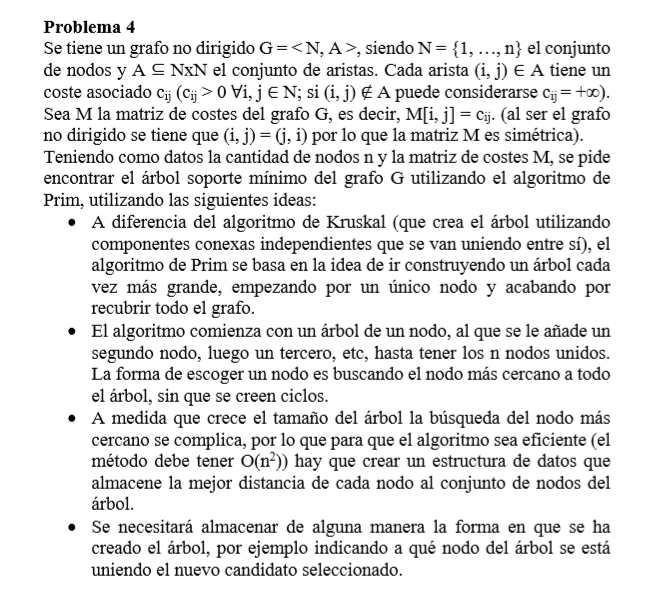
1. Recorremos la primera mitad de la lista y, cada valor, se compara con su simétrico de la lista (el primero con el último, el segundo con el penúltimo, etc). Si el número es mayor que su simétrico se intercambian de posición y si no, se quedan como están. Con esto conseguimos que el valor mínimo esté en la primera mitad y el máximo en la segunda mitad.
2. Una vez dividida la lista, recorremos la primera mitad en busca del mínimo. Y, tras la obtención del mínimo, recorremos la segunda mitad en busca del máximo.

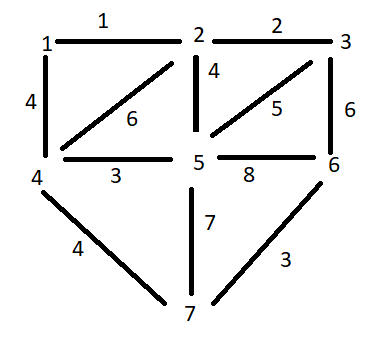
Con estos pasos conseguimos leer tres veces la mitad de la lista por lo que el coste será de 3n/2.

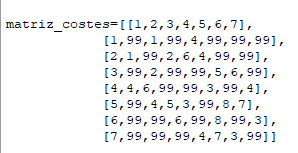
Este es un ejemplo de la salida del programa:



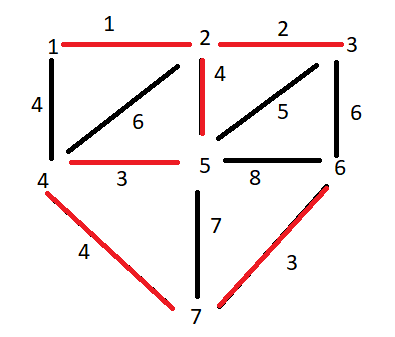
El algoritmo implementado en Python es el siguiente:



Para la realización de este algoritmo primero tenemos que implementar la matriz de costes. En la matriz de costes, la primera columna y fila representan los nodos del árbol y el resto de cada fila representa el coste de ir de un nodo a otro. El infinito (cuando un nodo no puede ir a otro o a sí mismo) se representa con un 99 (suponemos que el coste de los nodos no superará el 99). Este sería un ejemplo de la matriz de costes con su correspondiente árbol:



Donde se aprecia que, por ejemplo, la distancia de 4 a 7 es de 4 y se refleja en la matriz (cuarta fila, última columna). El árbol solución sería el siguiente:



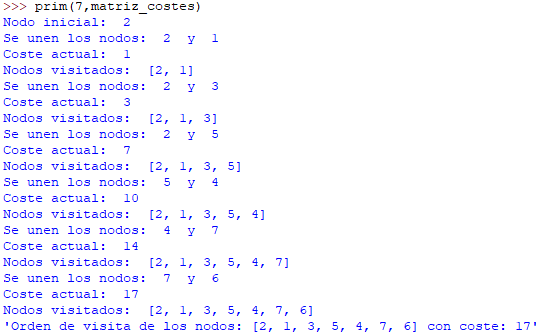
El objetivo de este ejercicio es recorrer el árbol de modo que visite todos los nodos solo una vez (sin ciclos) utilizando el menor coste posible.

Para ello realizamos el programa en Python (con la función principal prim() y la función auxiliar posicionMinimo) que se muestra a continuación:

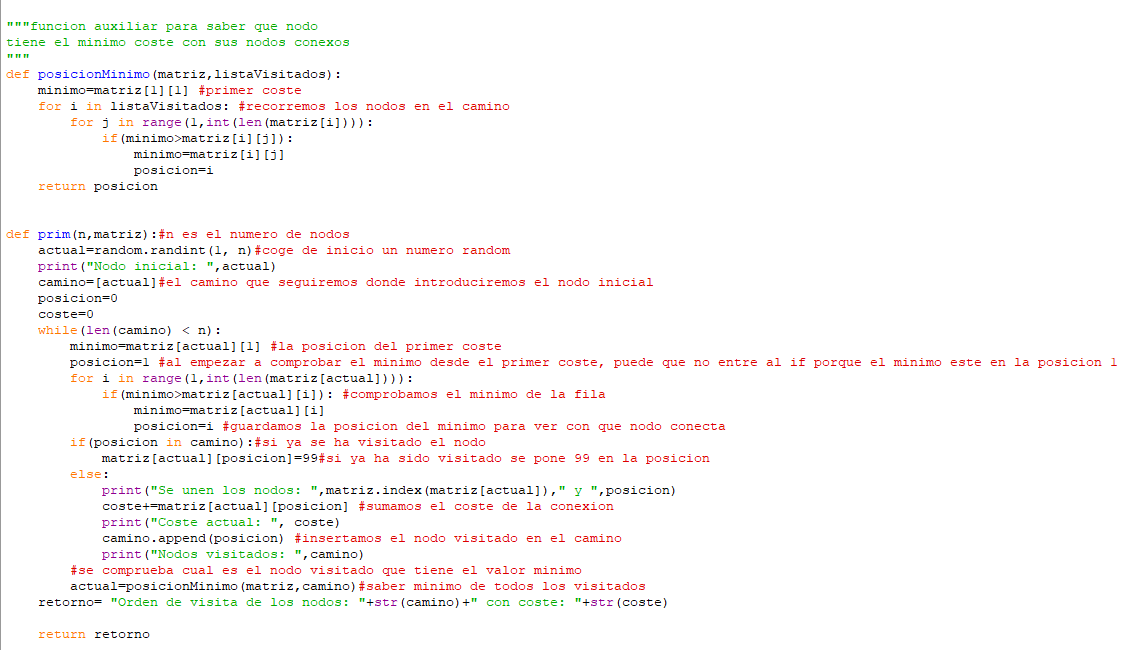
La función llevará por parámetro el número de nodos del árbol y la matriz de coste. Es importante poner correctamente el número de nodos y organizar bien la matriz, ya que sino el programa fallará.

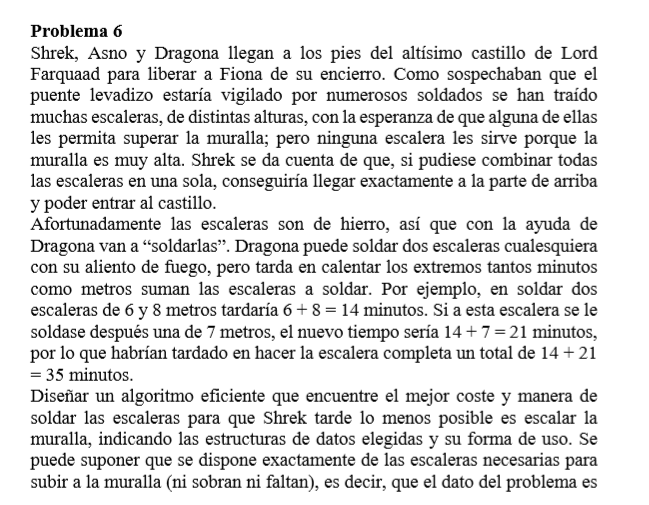
Primero de todo, se asigna un valor aleatorio entre los nodos del árbol para saber cuál es el nodo inicial y lo introducimos en una lista denominada camino donde guardaremos el orden de los nodos que visitamos. Tras esto, buscaremos el mínimo coste de sus nodos conexos (la fila de la matriz) y guardaremos la posición (si el nodo aún no ha sido visitado), que será el nodo al que se conecte y se añadirá a camino. Si el nodo ya se ha visitado, es decir, ya está en la lista camino, el coste entre ambos se pondrá a 99, que quiere decir, que ya no puede acceder a ese nodo al haber sido visitado anteriormente. Una vez que tenemos más de un nodo visitado, tenemos que comprobar qué nodo visitado tiene el valor mínimo, ya que el algoritmo de Prim puede unir un nodo con otro si este tiene el mínimo coste de los posibles sin ser el último que se haya visitado y para ello se realiza la función auxiliar, que recorre las filas de los nodos visitados buscando el nodo con coste mínimo. Estos pasos se realizan hasta que se hayan visitado todos los nodos y el programa devuelve el orden en que se han visitado.

Este sería un ejemplo de la salida del programa con el ejemplo anterior:



Donde se observa que el programa va mostrando qué nodos se unen, el coste actual y la lista con los nodos visitados de cada paso y finalmente muestra el coste final y el orden de visitado de los nodos.

El programa es el siguiente:



\*\*\*\*\*\*\*\*\*\*CORREGIDO\*\*\*\*\*\*\*\*\*

El objetivo del problema consiste en conseguir el mínimo tiempo en soldar las escaleras pudiendo soldar 2 escaleras a la vez.

Para su resolución, tenemos las escaleras primero en una lista. Tenemos una función que toma como parámetros una lista con las escaleras y otra con los costes. El caso base de la función será cuando se hayan usado todas las escaleras y se irá recorriendo la lista con los costes.

Si todavía quedan escaleras (else) se suman las dos menores de la lista escaleras y se eliminan de la lista. Después, se introduce la suma de estas dos escaleras en la lista de costes y, si todavía queda alguna escalera más, se introduce esta unión de las dos escaleras como una escalera “nueva” dentro de la lista escaleras.

Por ejemplo, para el caso de la lista de escaleras [6,8,7,5]:

Se cogen los dos mínimo y se suman -> 5+6 =11

Por lo tanto, la lista escaleras quedan [7,8,11] y la de costes [11]

Se vuelven a coger los dos mínimos y se suman -> 7+8=15

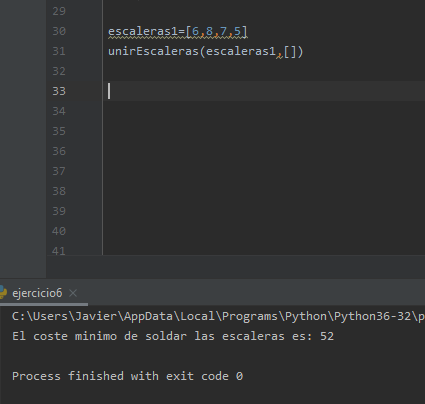
Por lo tanto, la lista escaleras queda [11,15] y la de costes [11,15]

Se vuelven a coger los dos mínimos (los dos únicos que quedan) y se suman -> 11+15=26

Por lo tanto, la lista escaleras queda vacía y la de costes [11,15,26]

Al ser la lista escaleras vacía, se suman los elementos de la lista costes -> 11+15+26=52

Esta sería la salida del programa para este ejemplo:



Vamos a estudiar otro ejemplo:

Para la lista de escaleras [8,9,20,5,3]:

Sumamos los dos mínimos 3+5=8

La lista de escaleras queda [8,9,20,8] y la de costes [8]

Volvemos a sumar los dos mínimos 8+8=16

La lista de escaleras queda [9,20,16] y la de costes [8,16]

Volvemos a sumar los dos mínimos 16+9=25

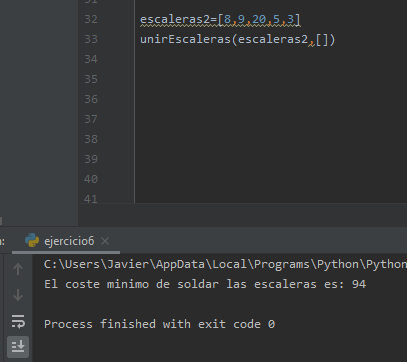
La lista de escaleras queda [20,25] y la de costes [8,16,25]

Volvemos a sumar los dos mínimos 20+25=45

La lista de escaleras queda vacía y la de costes [8,16,25,45]

Al ser la lista de escaleras vacía, se suman los elementos de la lista costes 8+16+25+45=94.

Esta sería la salida para este ejemplo:



El programa es el siguiente: 