Práctica 4 Encontrar la salida de un laberinto

Escribir el código para encontrar la salida de un laberinto, pueden usar la búsqueda primero en profundidad o primero en anchura.

El laberinto puede ser de cualquier tamaño, con un X se marcan los muros por donde no se puede pasar, el inicio se marca con I y la salida con S. Por ejemplo:

	0	1	2	3	4	5	6	7	8	9
0	X				X			S	X	
1	X	X		X	X		X	X	X	
2	X									
3	X	X	X	X	X		X	X	X	X
4										
5		X	X		X	X	X	X		X
6						X		X		
7		X	X	X				X		X
8						X		X		
9	X	X	X	X	I					X

Así entonces el inicio es (9,4) y la salida está en (0,7). en un archivo de texto para representar el laberinto Recomiendo que los espacios en blanco, que es por donde si se puede avanzar, tengan una letra también, por ejemplo la C de camino. El laberinto de arriba en un archivo de texto lo pueden escribir:

X C C C X C C S X C
X X C X X C X X X C
X C C C C C C C C C
X X X X X C X X X X
C C C C C C C C C C
C X X C X X X X C X
C C C C C X C X C C
C X X X C C C X C X
C C C C C X C X C C
X X X X I C C C C X

De esta forma pueden cargar el archivo de texto como en la práctica anterior, salvo que ahora no hay que convertir cada elemento a int

Pueden partir de la práctica anterior, haciendo las adecuaciones para el laberinto. Por ejemplo, a continuación está el código que les envié para la práctica 3

```
1       #!/usr/bin/python3
2
3       from Arbol import Arbol
4
5       def BPA(origen, destino, grafo):
6           raiz = Arbol(None,-1,origen)
7           frontera = [raiz]
8           visitados = []
9           while frontera:
10              nodo = frontera.pop(0)
11              if nodo.elemento == destino:
12                  print("Arbol generado")
13                  print (raiz)
14                  return nodo.rutaNodoRaiz()
15              if not (nodo.elemento in visitados):
16                  for i in range(0,len(grafo[nodo.elemento])):
17                      if grafo[nodo.elemento][i] == 1:
18                          if not (i in visitados):
19                              raiz.agregar(nodo.elemento,nodo.nivel,i)
20                              frontera.append(Arbol(nodo,nodo.nivel,i))
21                  visitados.append(nodo.elemento)
22          return None
23
24      archivo = open("Grafo.txt", "r")
25      grafo = []
26      for line in archivo.readlines():
27          grafo.append( [ int (x) for x in line.split() ] )
28
29      origen, destino = input("Ingrese el origen y destino: ").split()
30      ruta = BPA(int(origen), int(destino), grafo)
31      if ruta == None:
32          print ("No se encontro una ruta")
33      else:
34          ruta.reverse()
35          print ("Ruta encontrada")
36          print (ruta)
```

- En la línea **27** se agregan los elementos de cada línea leída, convirtiéndolos a int, como en este caso estamos leyendo caracteres, X, C, I o S no hay que convertir a int. Esa linea puede quedar solo **laberinto.append(line.split())** (cambié el nombre de grafo a laberinto solo para ser más precisos)
- En **29** se pide el origen y destino, eso era para el grafo, en el laberinto la información del inicio y salida están en el archivo de texto, por lo tanto no hay que pedir esta información
- En **30** se hace la llamada a la función BPA, con los parámetros origen, destino y grafo (el archivo de texto leído), ya no se debe enviar origen y destino solo grafo
- En la línea **5** se define la función BPA, también habría que quitar a origen y destino

- En la línea **6** se crea la raíz del arbol con el origen, pero no podemos hacer eso porque no tenemos el origen, primero hay que buscar la posición de el punto inicial, es decir, la posición de **I**.
- Recuerde que el archivo de texto está almacenado en una lista de listas, del ejemplo del laberinto de arriba sería: [[X,C,C,C,X,C,C,S,X,C], [X,X,C,X,X,C,X,X,X,C], [X,C,C,C,C,C,C,C,C,C], [X,X,X,X,X,C,X,X,X,X], [C,C,C,C,C,C,C,C,C,C], [C,X,X,C,X,X,X,X,C,X], [C,C,C,C,C,X,C,X,C,C], [C,X,X,X,C,C,C,X,C,X], [C,C,C,C,C,X,C,X,C,C], [X,X,X,X,I,C,C,C,C,X]]
- la posición del renglón donde se encuentra la I es la posición de la lista [X,X,X,X,I,C,C,C,C,X] dentro la lista de listas para este ejemplo tenemos una lista de 10 listas, cada una de estas 10 representa un renglón del laberinto, en este caso la I se encuentra en la lista con posición 9. La posición de la columna de I es la posición de la I dentro de la lista [X,X,X,X,I,C,C,C,C,X], en este caso 4. por lo tanto nuestra posición inicial sería (9,4)
- La posición puede ser el elemento de nuestros nodos, pueden convertirla a string o dejarla como una tupla, recomiendo la tupla. En python existen las variables tupla que contiene 2 elementos de cualquier típo separados por coma. Por ejemplo si la posición del renglon está en posR y la posición de la columna está en posC podemos asignar a la variable origen la tupla de posR y posC como: origen = (posC,posR)
- Ya con el origen ahora si creamos el nodo raíz
- En la línea **11** se revisa si el nodo que estamos revisando es el destino, en este caso lo que tenemos que hacer es revisar si la posición del nodo (que esta guardada en elemento) es una S en el grafo. Por ejemplo:

```
posR, posC = nodo.elemento
if laberinto[posR][posC] == 'S':
```

- En este caso encontraríamos la salida
- Si no encontramos la salida revisamos que el nodo no lo hayamos visitado antes, linea **15**, esto sería igual, necesitamos revisar si no hemos pasado antes por esa casilla.
- En la línea **16** se hace un for para recorrer todo el grafo y ver quienes son vecinos, con quien está conectado el nodo que estamos revisando. En este caso no se debe recorrer todo el archivo de texto entonces ese for ya no aplica
- En la línea **17** se revisa si el nodo tiene conexión con el vecino i. Para el caso del laberinto lo que tenemos que revisar es si podemos avanzar hacia arriba, izquierda, abajo o derecha. Para esto entonces deben revisar si laberinto[posR-1][posC] != 'X' esto nos indica que si en posR-1 y posC hay un carácter diferente de X entonces agregamos esa posición como siguiente nodo a explorar. Si en esa posición hubiera una X entonces no hay paso por lo que no debería agregarse esa posición a los posibles nodos a explorar. posR-1 nos indica que dimos un paso a la izquierda, si fuera +1 sería a la derecha y lo mismo para posC con -1sería un movimiento hacia arriba y con +1 hacia abajo

Básicamente solo se debe adaptar el elemento que se le da a los nodos del árbol, que como menciono arriba recomiendo usar una tupla. Modificar la forma en la que revisamos que ya llegamos al destino, y la forma en la que revisamos quienes son los hijos del nodo que estamos explorando, es decir sus cuatro posibles movimientos.