Práctica 3 Búsquedas a ciegas

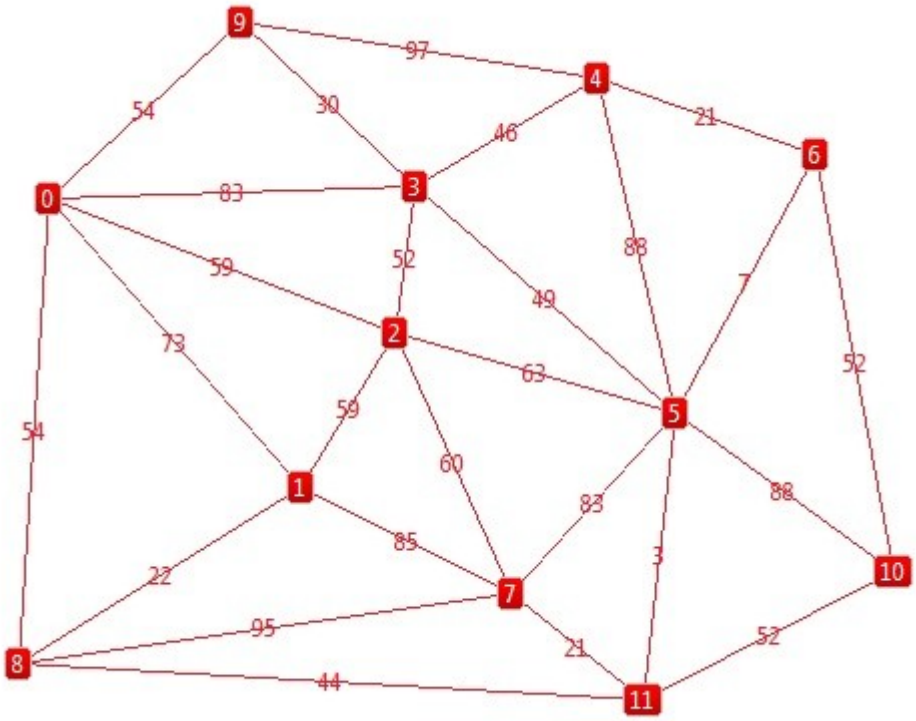
Programar en Python 3 las siguientes técnicas de búsquedas a ciegas:

- Búsqueda primero en profundidad
- Búsqueda de costo uniforme

Para realizar la búsqueda sobre un árbol, en realidad no es necesario contar con el árbol. Se puede implementar la búsqueda mediante una lista, cola o pila. Veamos como realizar la búsqueda primero en anchura:

```
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)
```

El algoritmo trabaja sobre un grafo no dirigido, como el siguiente:



Este tipo de grafos se pueden representar en un archivo de texto de la siguiente manera:

9999 73 59 83 9999 9999 9999 9999 54 54 9999 9999
73 9999 59 9999 9999 9999 9999 85 22 9999 9999 9999
59 59 9999 52 9999 63 9999 60 9999 9999 9999 9999
83 9999 52 9999 46 49 9999 9999 9999 30 9999 9999
9999 9999 9999 46 9999 88 21 9999 9999 97 9999 9999
9999 9999 63 49 88 9999 7 83 9999 9999 88 3
9999 9999 9999 9999 21 7 9999 9999 9999 9999 52
9999 85 60 9999 9999 83 9999 9999 95 9999 9999 21
54 22 9999 9999 9999 9999 9999 95 9999 9999 9999 44
54 9999 9999 30 97 9999 9999 9999 9999 9999 9999 9999
9999 9999 9999 9999 9999 88 52 9999 9999 9999 9999 52
9999 9999 9999 9999 9999 3 9999 21 44 9999 52 9999

Cuando existe una conexión entre el un par de nodos, se escribe el peso de dicha conexión, si no existe conexión escribimos un número grande para indicarlo, en este caso 9999.

Cada línea corresponde a uno de los nodos, por lo tanto tenemos 12 líneas ya que tenemos 12 nodos, del 0 al 11. La primer línea corresponde al nodo 0, la segunda al nodo1, y así sucesivamente hasta la línea 12 que corresponde al nodo 11. El nodo 0 no tiene conexión consigo mismo, por lo tanto el primer dato que aparece es un 9999. Luego tiene conexión con el nodo 1 por lo que aparece el número 73, que corresponde al peso de la conexión entre 0 y 1. Entre 0 y 2 hay una conexión con peso de 59, ese es el siguiente dato en la línea. Y así continúan colocándose los pesos de las conexiones, tenemos 12 con 12

datos cada una. Si el grafo no tiene pesos, entonces se puede representar unicamente con 0 y 1, por ejemplo, el mismo grafo de arriba representado sin tomar en cuenta los pesos:

0 1 1 1 0 0 0 0 1 1 0 0
1 0 1 0 0 0 0 0 1 1 0 0 0
1 1 0 1 0 1 0 1 0 0 0 0
1 0 1 0 1 1 0 0 0 1 0 0
0 0 0 1 0 1 1 0 0 1 0 0
0 0 1 1 1 0 1 1 0 0 1 1
0 0 0 0 1 1 0 0 0 0 1 0
0 1 1 0 0 1 0 0 1 0 0 1
1 1 0 0 0 0 0 1 0 0 0 1
1 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 1
0 0 0 0 0 1 0 1 1 0 1 0

El cero indica que no hay conexión y el 1 indica que si hay conexión

Como la búsqueda primero en anchura no requiere de los pesos (aunque los tuviera no los tomaría en cuenta), vamos a utilizar la representación de 0 y 1 para éste ejemplo. Note que para realizar la búsqueda de costos mínimos si tendrá que usar la representación con los pesos.

Analicemos el código:

- En la línea **24** se abre el archivo Grafo.txt en modo lectura. El archivo Grafo.txt contiene el grafo representado como se muestra arriba, con 0 y 1 (encontrarán el archivo adjunto junto con este PDF).
- En la línea **25** se crea una lista vacía.
- La línea **26** inicia un ciclo for para cada línea leída del archivo de texto
- la línea 27 separa la línea de texto leída en elementos, por ejemplo la primer línea del archivo sería: “0 1 1 1 0 0 0 0 1 1 0 0”, con split() separamos esta cadena de texto en elementos individuales. Como cada elemento ya separado es de tipo string lo convertimos a int. Cada elemento ya convertido en int se agrega a una lista, por eso la instrucción int (x) for x in line.split() esta encerrada en corchetes para crear una lista con cada x. Finalmente agregamos la lista de x a la lista grafo, quedando por ejemplo grafo = [[0,1,1,1,0,0,0,0,1,1,0,0]], al agregar la siguiente línea quedaría: grafo = [[0,1,1,1,0,0,0,0,1,1,0,0],[1,0,1,0,0,0,0,1,1,0,0,0]]. De esta forma podemos representar una matriz en Python, como una lista de listas.
- En la línea **29** pedimos al usuario que ingrese el origen y destino para buscar una ruta.
- La línea **30** ejecuta la función BPA (Búsqueda Primero en Anchura) dándole como parámetros el origen y destino, convertidos en int, y la lista de listas que contiene al grafo. El resultado es una lista con la ruta encontrada que se almacena en la variable ruta.
- En la línea **31** verificamos si obtuvimos una ruta o no, para imprimir la solución.
- La línea **34** invierte el orden de la ruta, la ruta que devuelve BPA comienza con el destino y termina en el origen, entonces la invertimos para presentar la ruta comenzando desde el origen hasta el destino.
- En la línea **5** está la definición de BPA

- Lo primero que hacemos es crear el nodo raíz en la línea **6**. El nodo se crea indicando el padre, el nivel del padre y el origen, como es la raíz, no tiene padre, por eso se indica un None, el nivel del padre sería -1 para que la raíz tenga el nivel 0.
- Como mencionaba arria, no es necesario el árbol, aunque el ejemplo lo usa solo para imprimirlo y que puedan observar cual fue el árbol creado y comprobar que esta correcto el algoritmo.
- La búsqueda la haremos mediante dos listas, frontera y visitados. La lista frontera ira almacenando en orden los siguientes nodos que vamos a explorar y expandir, en la lista visitados se irán agregando los nodos que ya fueron visitados y expandidos, esto para no repetirlos y evitar trabajar de más. Frontera comienza con 1 elemento que es el nodo raíz, y visitados comienza vacío.
- En la línea **9** comenzamos un ciclo while mientras frontera tenga elementos
- En la línea **10** sacamos el primer elemento de frontera en la variable nodo
- En la línea **11** revisamos si nodo es el elemento buscado, es decir el destino, si es así, imprimimos el árbol generado y regresamos la ruta desde el nodo destino al nodo raíz.
- Si no es el nodo buscado, entonces en la línea **15** revisamos si es un nodo que ya fue visitado anteriormente, si ya se visitó antes, no tiene caso volver a expandirlo, sería una ruta más larga y BPA nos da la ruta con menos saltos así que esta opción nunca se exploraría.
- Si el nodo no ha sido visitado entonces en la línea **16** iniciamos un ciclo para cada elemento en la lista correspondiente al nodo para el grafo, por ejemplo si el nodo que estamos revisando es el nodo 6, entonces el ciclo for recorre los indices para cada elemento de la lista número 6 en grafo, en este ejemplo son 12 elementos entonces i va de 0 a 11.
- En la línea **17** revisamos el elemento en el grafo para saber si existe conexión entre el nodo y cada uno del resto de nodos, esto es si el elemento del grafo es 1.
- Para la línea **18**, cuando encontramos una conexión, ésta será un posible camino a seguir en la búsqueda, es uno de los hijos que se van a expandir. Sin embargo, existe la posibilidad de que éste hijo encontrado ya haya sido visitado anteriormente, por lo tanto no tendría caso agregarlo a la frontera, por el mismo razonamiento que en la línea 15. Por lo tanto si el posible hijo no esta en la lista de visitados entonces lo agregamos al final de la frontera para explorarlo más tarde. También se agrega el nodo al arbolo para ir representándolo e imprimirlo al final.
- Cuando terminamos de revisar a todos los posibles hijos del nodo, entonces agregamos el elemento del nodo a la lista de visitados en la línea **21**.
- Finalmente en la línea **22** se regresa un None cuando el ciclo while termina, si el while termina es porque la frontera quedo vacía, es decir ya no hay más nodos que explorar por lo que no hay una ruta entre los nodos origen y destino.

Como pueden notar, para fines de imprimir el árbol resultante se requiere una clase árbol, en la práctica 2 creamos una clase árbol que tenía los datos de hijos [] y elemento. En el caso de la práctica 3 se requiere además almacenar la información del nivel en el que se encuentra el nodo y quien es su padre. El nivel, sirve para saber en que nivel se encuentra el nodo padre cuando queramos agregarle un hijo, esto porque pueden haber 2 nodos con el mismo elemento en diferentes niveles del árbol, pero a nosotros nos interesa uno en particular, así que al agregar el nodo al árbol tenemos que revisar que sera en el nivel que nos indican. La variable padre, nos va a servir para que, una vez encontrado el nodo destino, podamos ir siguiendo hacia arriba la ruta hasta el nodo raíz, esta ruta sera la respuesta encontrada por BPA. Además de la función rutaNodoRaiz para recorrer el árbol de abajo hacía arriba y regresar la ruta obtenida.