INFORME PROYECTO FINAL HABITABILIDAD EN EDIFICIO RESIDENCIAL

Faider Camilo Trujillo

cod:20192020136

Hanna Valentina Sarmiento Marques

cod:20192020105

Juan Esteban Forero Rodríguez

cod:20192020125

Introducción

El siguiente informe pretende exponer como un lugar es habitable o no dependiendo de ciertas variables que permiten la habitabilidad en una zona residencial, más específicamente en un edificio de apartamentos. Para esto se toma al ruido y su propagación como objeto de estudio y con base a un programa, algoritmos y estructuras de datos arrojar un resultado que permita determinar y recomendar si un apartamento es apto o no para que sea habitado.

Objetivo

Crear un programa y usar estructuras de datos permitir evaluar si un apartamento situado en un edifico puede ser habitable o no basándose en la transmisión del ruido.

Metodología

Haciendo uso de los aprendido en clase se emplearon distintas estructuras de datos capaces de resolver el problema en cuestión. Empleando grafos, árboles y métodos de busca en arboles junto con Librerías de Python y métodos propios se logra dar resolución al problema, a continuación, se explicarán las estructuras empleadas:

1. Se usaron librerías de Python que permitieran leer archivos de tipo Excel, pues en esta se encontraba una tabla con los datos de las resistividades de materiales por frecuencias

2. Se creo un directorio con clave valor, y se le asigno los nombres de los nodos tal cual se muestra en la figura.

EdificioFinal.py G:\... EdificioFinal.py Proyecto_Ciencias2 EdificioFinal.py .\

EdificioFinal.py > determinar_transmision

```
import networkx as nx

datos = pd.read_excel('Materiales.xlsx', index_col=False)
df = pd.DataFrame(datos)

G={'a':['b','c'] ,'b':['d'],'c':['e'],'d':['f'],'e':['f'],'f':[]  }
visitado=set()
visitado2=set()
```

3. Se empleo un grafo y se creo una clase por cada parte que lo compone, así se pueden observar clases como nodo, arista, grafo dirigido.

```
class Grafo_Dirigido:
    def __init__(self):
        self.dicc_grafo = {}

    def agregar_nodo(self,nodo):
        self.dicc_grafo[nodo] = []
        if nodo in self.dicc_grafo:
            return "El Nodo ya existe en el grafo"
        self.dicc_grafo[nodo] = []

    def agregar_arista(self,arista):
        nodo_origen = arista.get_nodo_origen()
        nodo_destino = arista.get_nodo_destino()
        if nodo_origen not in self.dicc_grafo:
            raise ValueError(f' El Nodo{nodo_origen.get_nombre()} no esta en el grafo')
        if nodo_destino not in self.dicc_grafo:
            raise ValueError(f' El Nodo{nodo_destino.get_nombre()} no esta en el grafo')
        self.dicc_grafo[nodo_origen].append(nodo_destino)

    def comprobar_nodo(self,nodo):
        return nodo in self.dicc_grafo

    def get_nodo(self,nombre_nodo):
        for n in self.dicc_grafo:
            if nombre_nodo == n.get_nombre() : return n
        print(f'El Nodo {nombre_nodo} no existe')

    def get_vecinos(self,nodo):
        return self.dicc_grafo[nodo]

    def __str__(self):
        todos_vertices = ''
        for nodo_origen in self.dicc_grafo:
            for nodo_destino in self.dicc_grafo[nodo_origen]:
                todos_vertices += nodo_origen.get_nombre() + '---->' + nodo_destino.get_nombre() + '\n'
        return todos_vertices
```

```python
class Grafo_No_Dirigido (Grafo_Dirigido):
    def agregar_arista(self, arista):
        Grafo_Dirigido.agregar_arista(self, arista)
        arista_viceversa = Arista(arista.get_nodo_destino(), arista.get_nodo_origen())
        Grafo_Dirigido.agregar_arista(self, arista_viceversa)


class Arista:
    def __init__(self,nodo_origen,nodo_destino):
        self.nodo_origen = nodo_origen
        self.nodo_destino = nodo_destino

    def get_nodo_origen(self):
        return self.nodo_origen
    def get_nodo_destino(self):
        return self.nodo_destino
    def __str__(self):
        return self.nodo_origen.get_nombre() + '---->' + self.nodo_destino.get_nombre()

class Nodo:
    def __init__(self,nombre, ruido, resistividad, piso ,transmision, mensaje):
        self.nombre = nombre
        self.ruido = ruido
        self.resistividad = resistividad
        self.piso = piso
        self.transmision=transmision
        self.mensaje = mensaje
    def set_transmision(self, ruido):
        self.ruido = ruido
    def get_nombre(self):
        return self.nombre
    def get_ruido(self):
        return self.ruido
    def get_mensaje(self):
        return self.mensaje

    def set_mensaje(self,mensaje):
```

4.Se crea el grafo, se inicializan variables y se empiezan a crear las funciones que ayudan calcular la habitabilidad

```python

def crear_grafo(grafo):
    g=grafo()
    for n in ( 'a','b','c','d','e','f'):
        g.agregar_nodo(Nodo(n,0,0,0,0,''))

    g.agregar_arista(Arista(g.get_nodo('a'),g.get_nodo('b')))
    g.agregar_arista(Arista(g.get_nodo('a'),g.get_nodo('f')))
    g.agregar_arista(Arista(g.get_nodo('b'),g.get_nodo('c')))
    g.agregar_arista(Arista(g.get_nodo('c'),g.get_nodo('d')))
    g.agregar_arista(Arista(g.get_nodo('c'),g.get_nodo('f')))
    g.agregar_arista(Arista(g.get_nodo('d'),g.get_nodo('e')))
    g.agregar_arista(Arista(g.get_nodo('e'),g.get_nodo('f')))

    return g

def calcular_resistividad(material:str, frecuencia:str):
    return (df[df['Material'] == material] [frecuencia]).values[0]


def busqueda_prof_imp_transmision(visitado,grafo:Grafo_No_Dirigido,diccionario,raiz):
    if raiz not in visitado:
        print(raiz)
        visitado.add(raiz)
        for vecinos in diccionario[raiz]:
            nodo_vecino:Nodo = grafo.get_nodo(vecinos)
            print(nodo_vecino.get_transmision())
            busqueda_prof_imp_transmision(visitado,grafo,diccionario,vecinos)



def busqueda_prof_habitabilidad(visitado2,grafo:Grafo_No_Dirigido,diccionario,raiz):
    habitabilidad = 0
    if raiz not in visitado2:
        print(raiz)
        visitado2.add(raiz)
        nodo_raiz:Nodo = grafo.get_nodo(raiz)
        habitabilidad = nodo_raiz.get_resistividad() - nodo_raiz.get_transmision()
        nodo_raiz.set_mensaje(von_habitabilidad(habitabilidad))
```

5.Se realiza un apartado grafico para mostrar los datos en pantalla al usuario

```
216
217 G2=nx.Graph()
218 G2.add_nodes_from(['a', 'b', 'c', 'd','e','f'])
219 G2.add_edges_from([('a', 'b'), ('a', 'c'), ('b', 'd'), ('c', 'e'), ('c', 'd') ,('d', 'f'),('e','f')])
220 pos = nx.spring_layout(G2)  # Obtener la posición de los nodos
221 pos = {'a': (0, 0), 'b':  (1, 0) , 'c': (0, 1), 'd': (1, 1),'e': (0, 2),'f': (1, 2)}
222 pos_lbl={'a': (0.01, 0.2), 'b':  (1, 0.2) , 'c': (0, 1.1), 'd': (1, 1.1),'e': (0, 2.1),'f': (1, 2.1)}
223 nx.draw_networkx(G2, pos)
224 labels = {'a': 'a', 'b': 'b', 'c': 'c', 'd': 'd','e': 'e','f': 'f'}
225 labels2 = {}
226 print("########################################")
227 for n in ( 'a','b','c','d','e','f'):
228     labels2[n] = G1.get_nodo(n).get_mensaje()
229     print(G1.get_nodo(n).get_mensaje())
230 nx.draw_networkx_labels(G2, pos_lbl, labels2, font_size=8, font_color='black', horizontalalignment='left')
231 nx.draw_networkx_labels(G2, pos, labels)
232 plt.subplots_adjust(left=0.01, right=0.98, bottom=0.1, top=0.9)
233 plt.title("Habitabilidad Edificio")
234 plt.show()
```

Resultados

Se muestran en consola tanto en pantalla los datos obtenidos del calculo de la resistividad del ruido y posterior calculo de la habitabilidad de los apartamentos:

```
Grafo No dirigido
a---->b
a---->f
b---->a
b---->c
c---->b
c---->d
c---->f
d---->c
d---->e
e---->d
e---->f
f---->a
f---->c
f---->e

60
b
0
d
50
f
20
c
0
e
50


habitabilidad
a
b
d
f
c
e
########################################
```

```
habitabilidad
a
b
d
f
c
e
##########################################
No habitable, sólo para usos comerciales cómo gimnasios, bares, discotecas
No habitable, sólo para uso social o comercial como restaurantes, tiendas
Habitable
Habitable
Habitable
Habitable pero se aconseja material aislante para actividades cómo dormir o estudiar
```

Conclusión

Se obtiene satisfactoriamente un resultado positivo en cuanto a que se ven reflejados los datos y los cálculos en los estándares oficiales para determinar si un lugar es apto para la habitabilidad analizando el ruido, además se logró con satisfacción dar una recomendación acerca de la afectación en cada apartamento al usuario. Sé concluye que las estructuras de datos son unas herramientas poderosas que pueden ser usadas en conjuntos para plantear, resolver y modelar sistemas o situaciones complejas del mundo cotidiano y crear programas y software que usen dichos datos para obtener un beneficio cualquiera.