UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS
DEPARTAMENTO DE MATEMÁTICA

MATEMÁTICA PARA COMPUTACIÓN 2
Ing. José Alfredo González Díaz
Aux. Byron Estuardo
Sección A

Nombre		Tema
Andres Alejandro Agosto Mendez Pablo Jose Ortiz Linares		Grafos y algoritmos de busqueda
Registro Académico	**Actividad**	**Fecha**
202113580 , 202200231	Proyecto	27 / 04 / 2023

introducción:

En el presente informe se detalla el desarrollo de un programa que permite crear un grafo y aplicar el algoritmo de búsqueda en profundidad (DFS, por sus siglas en inglés). El objetivo de este proyecto es proporcionar una herramienta que permita visualizar y explorar grafos, así como aplicar una técnica de búsqueda que permita encontrar información relevante en ellos. El programa se desarrolló utilizando el lenguaje de programación Python, haciendo uso de la biblioteca Graphviz para la generación y visualización de grafos, y del algoritmo de búsqueda en profundidad para recorrer el grafo y encontrar información relevante. El algoritmo DFS es una técnica de búsqueda utilizada en grafos que permite encontrar todos los nodos accesibles desde un nodo de inicio determinado. A medida que se recorre el grafo, se van marcando los nodos visitados y se profundiza en cada rama del grafo hasta que no hay más nodos por visitar. Este algoritmo es muy utilizado en problemas de búsqueda y exploración de grafos, y es esencial para muchas aplicaciones de inteligencia artificial y análisis de datos. En el presente informe se detallan los aspectos técnicos y funcionales del programa, así como la metodología de desarrollo y las pruebas realizadas para asegurar su correcto funcionamiento. Además, se presentan los resultados obtenidos al aplicar el algoritmo DFS a diferentes grafos de prueba, y se discuten las posibles aplicaciones y mejoras del programa en el futuro. En resumen, el programa desarrollado en este proyecto ofrece una herramienta útil y fácil de usar para la exploración y búsqueda de información en grafos, y puede ser utilizado en una amplia variedad de aplicaciones en diferentes campos de estudio.

Link de GitHub del programa: https://github.com/andrekss/Proyecto-MC2.git

Manual de uso:

Para empezar a usar el programa, primero se debe instalar el programa Graphviz a la computadora, donde se vaya a trabajar. A continuación, se detallan los pasos a seguir para su instalación:

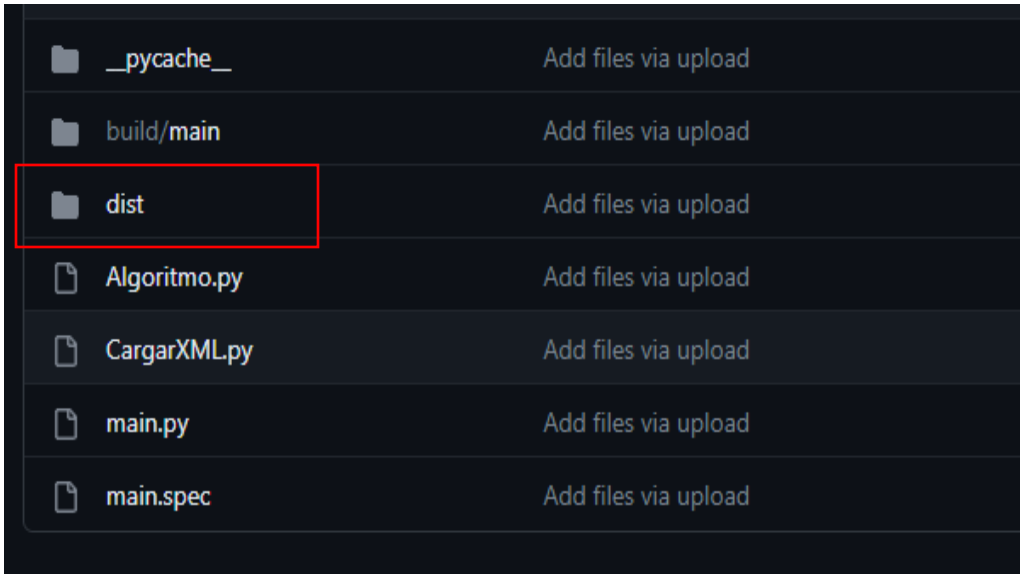
1. Visitar el sitio web oficial de Graphviz (https://graphviz.org/download/) y hacer clic en el botón "Download" para descargar la versión más reciente de Graphviz para Windows.
2. Una vez que se descargue el archivo de instalación, hacer doble clic en él para ejecutarlo. Se abrirá el asistente de instalación.
3. Aceptar los términos de licencia y seguir las instrucciones en pantalla para seleccionar la ubicación de instalación y las características que se desean instalar. Por lo general, se recomienda instalar todos los componentes para asegurarse de que todas las funcionalidades de Graphviz estén disponibles.
4. Hacer clic en "Install" para iniciar la instalación. Esto puede tomar varios minutos.
5. Una vez que se complete la instalación, hacer clic en "Finish" para cerrar el asistente de instalación.
6. Para verificar que Graphviz se ha instalado correctamente, abrir una ventana del símbolo del sistema (CMD) y escribir "dot -V". Esto mostrará la versión de Graphviz que se ha instalado.

Si después de haber seguido los pasos anteriores aun no es posible generar los gráficos puede deberse a un problema con el “PATH” de la computadora que se esta utilizando, si es así por favor tratar los siguientes pasos.

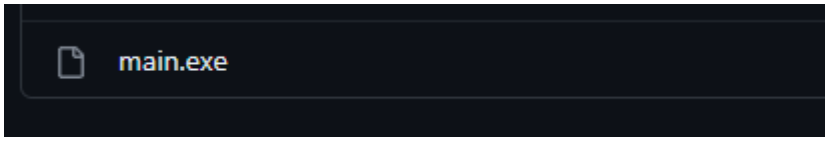
1. Abrir el menú Inicio y buscar "Variables de entorno".
2. Hacer clic en "Editar las variables de entorno del sistema".
3. En la ventana Propiedades del sistema, hacer clic en el botón "Variables de entorno".
4. Buscar la variable "Path" en la lista de variables del sistema y hacer clic en "Editar".
5. Agregar la ruta de instalación de Graphviz al final de la variable "Path", separada por un punto y coma (;). Por ejemplo, si Graphviz se ha instalado en "C:\Program Files (x86)\Graphviz", agregar ";C:\Program Files (x86)\Graphviz\bin" al final de la variable "Path".
6. Hacer clic en "Aceptar" para guardar los cambios y cerrar todas las ventanas.
7. Abrir una nueva ventana de línea de comandos (CMD) y escribir "dot -V" para verificar que Graphviz ahora está disponible en el PATH del sistema.

Una vez instalado graphviz en la computadora se puede abrir el programa el cual se encargará de graficar el grafo.

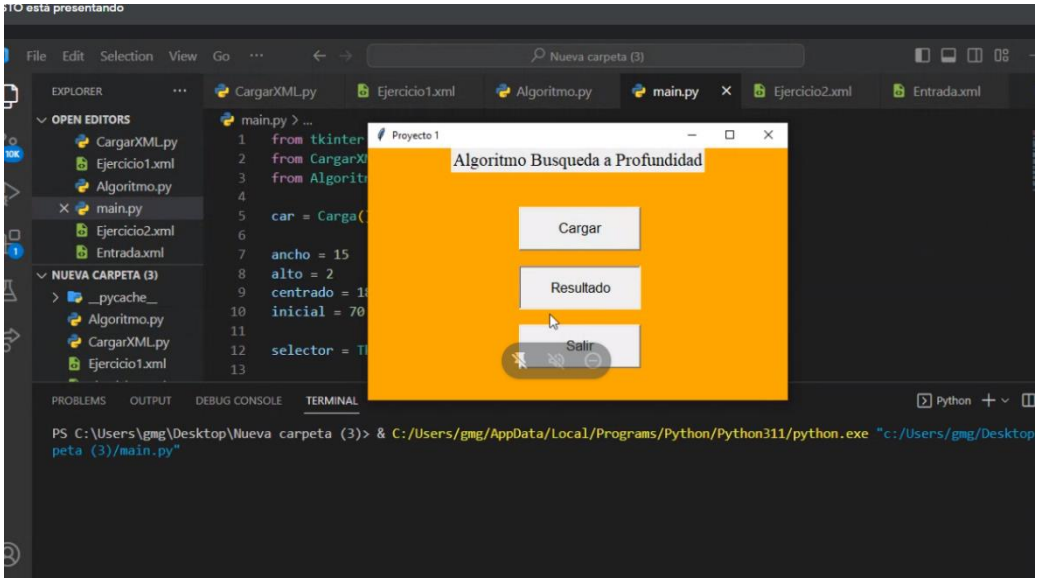
1. Descomprimir el archivo zip dentro de una carpeta.
2. Dentro de la carpeta, abrir la carpeta llamada “dist”.



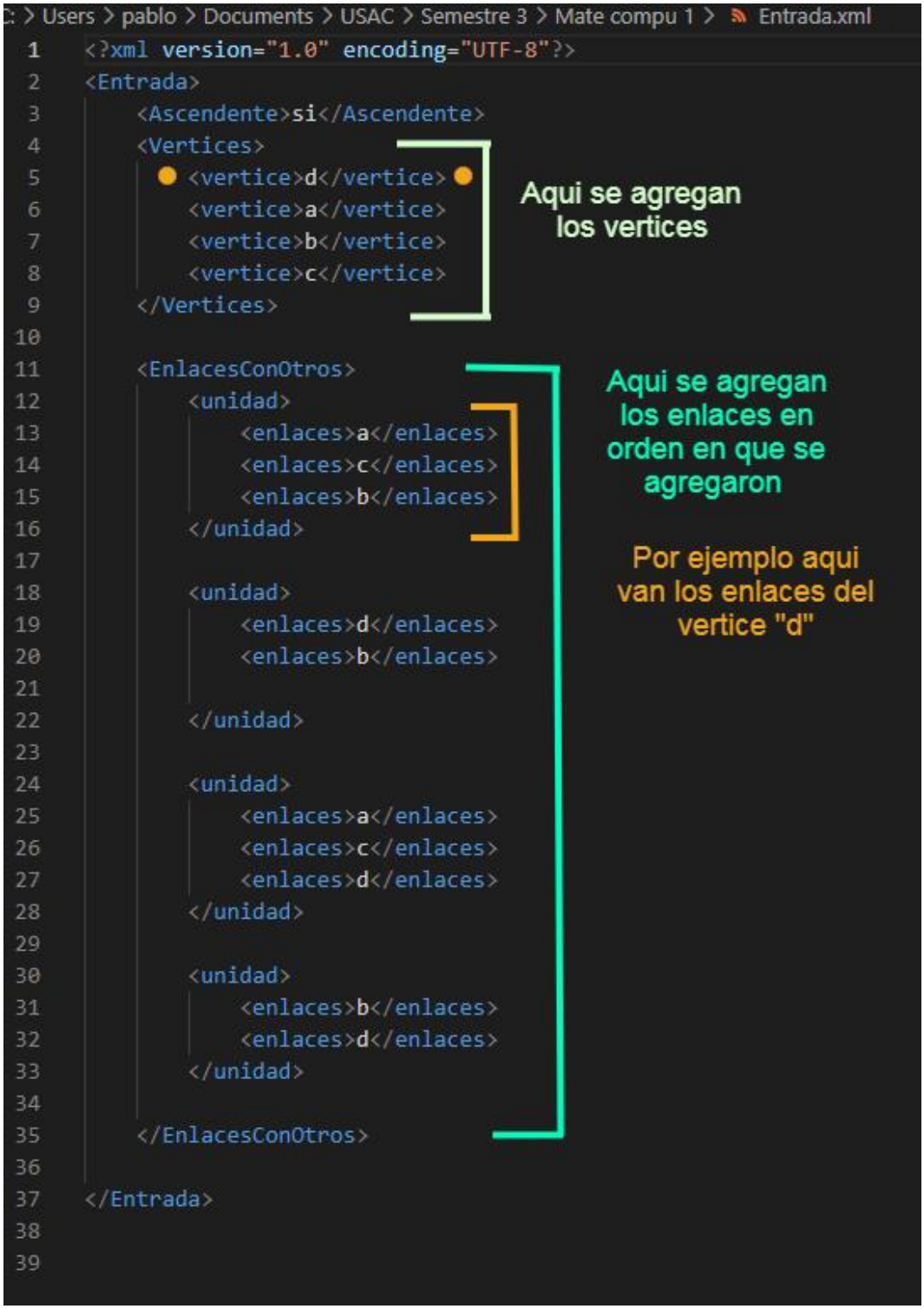
3. Dentro de la carpeta “dist” dar doble clic en el ejecutable “main.exe”



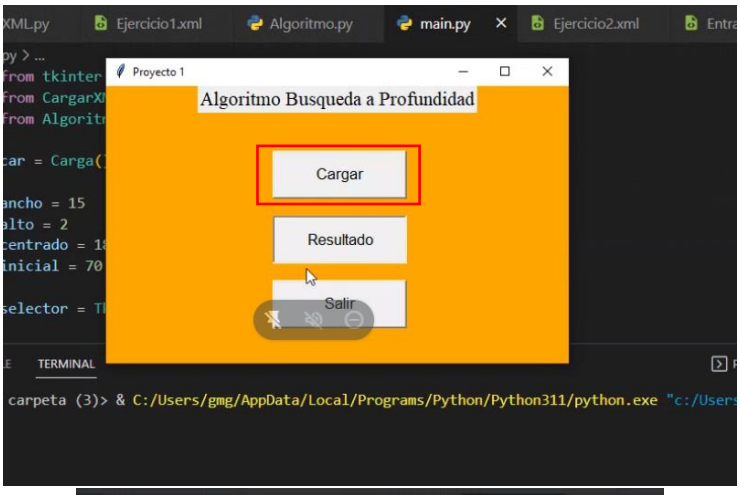
4. Una vez ejecutado se abrirá el siguiente menú.

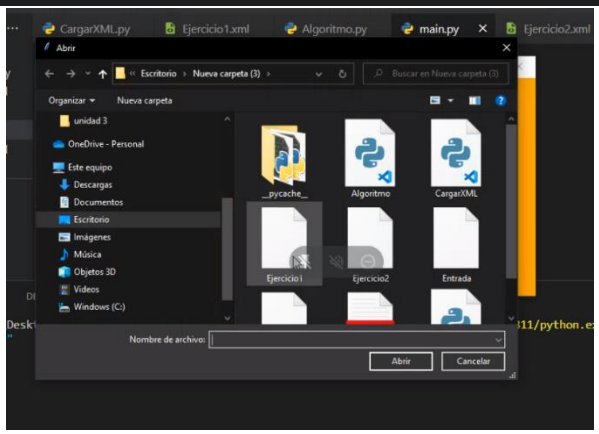


5. A continuación, se debe crear un archivo XML con la siguiente estructura y luego guardarlo, cabe resaltar que únicamente se aceptan letras minúsculas del abecedario.



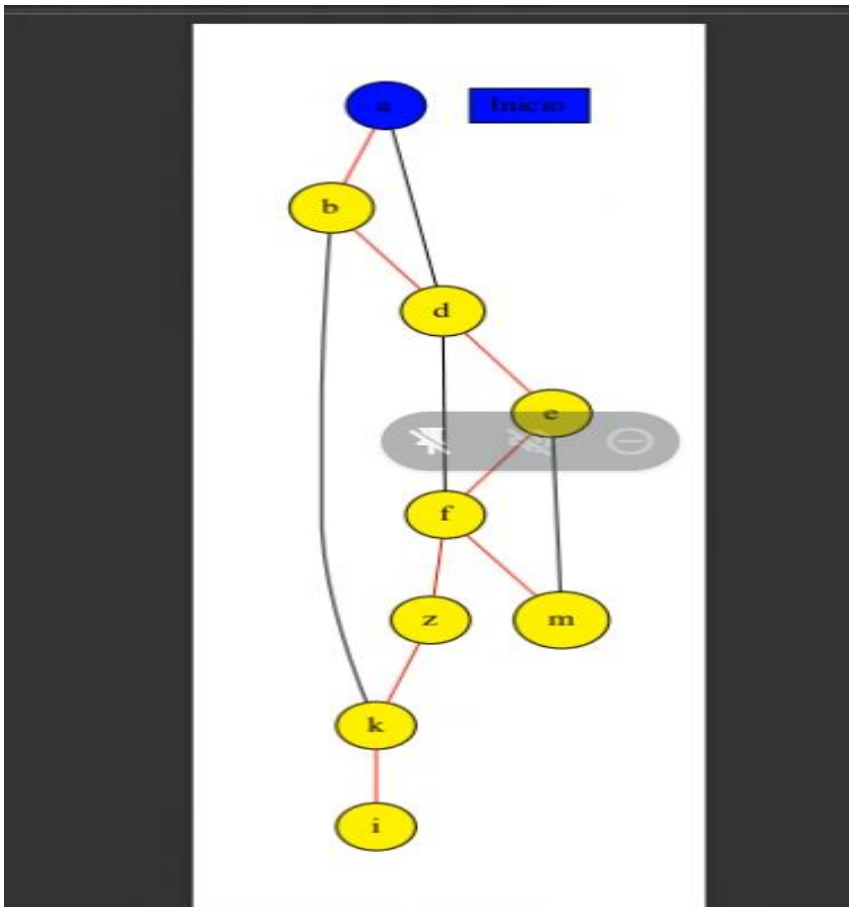
6. Después de haber creado el archivo regresar al menú y dar clic en la opción de “cargar”, se nos dará la opción de seleccionar un archivo el cual deberá ser el XML previamente creado.





7. Luego de cargar el archivo se regresar al menú y dar clic en “resultado”.

8. Al momento de dar clic en la opción de “resultado” se abrirá una ventada mostrando en formato PDF el grafo creado junto con el algoritmo de búsqueda en profundidad aplicado.



9. Finalmente dar clic en la opción “salir” para cerrar el programa.

Explicación general del código:

Para empezar en esta parte se lee el archivo de tipo XML que el usuario debe crear como en el ejemplo.

```python
from xml.etree.ElementTree import *
from Enlace import *

class Carga:
    def __init__(self):
        self.vertices = []
        self.EnlacesV = []

    def LeerXML(self):
      ruta='/Users/gmg/Desktop/Nueva carpeta (3)/Entrada.xml'  # se necesita una ruta del archivo
      if ruta != '':
        try:
         Arbol = parse(ruta) #cargamos el archivo en la variable Arbol
         raiz = Arbol.getroot() #entramos a la raíz de todo el archivo que sería Entrada
         contador = 0
         for lista in raiz: # empezamos a recorrer cada lista del arbol
            if contador == 0:# lista de vértices
                for vertice in lista:
                    self.vertices.append(vertice.text) # agregamos el vertice en la lista
            elif contador ==1: # ciclamos a la siguiente lista
                for unidad in lista:
                    Enlaces = []  # hacemos un arreglo para cada unidad
                    for enlaces in unidad:
                        Enlaces.append(enlaces) # agregamos cada enlace
                    self.EnlacesV.append(Enlaces) #y guardamos ala lista de la unidad respectiva

            contador+=1
        except:
         pass
```

```
C: > Users > pablo > Documents > USAC > Semestre 3 > Mate compu 1 > Entrada.xml
<?xml version="1.0" encoding="UTF-8"?>
<Entrada>
    <Ascendente>si</Ascendente>
    <Vertices>
        <vertice>d</vertice>
        <vertice>a</vertice>
        <vertice>b</vertice>
        <vertice>c</vertice>
    </Vertices>

    <EnlacesConOtros>
        <unidad>
            <enlaces>a</enlaces>
            <enlaces>c</enlaces>
            <enlaces>b</enlaces>
        </unidad>

        <unidad>
            <enlaces>d</enlaces>
            <enlaces>b</enlaces>

        </unidad>

        <unidad>
            <enlaces>a</enlaces>
            <enlaces>c</enlaces>
            <enlaces>d</enlaces>
        </unidad>

        <unidad>
            <enlaces>b</enlaces>
            <enlaces>d</enlaces>
        </unidad>

    </EnlacesConOtros>

</Entrada>
```

Aqui se agregan los vertices

Aqui se agregan los enlaces en orden en que se agregaron

Por ejemplo aqui van los enlaces del vertice "d"

Después, se recorre toda la lista de vértices y ordenan los vértices de menor a mayor por medio de asignarles un valor numérico a cada letra del abecedario. ejemplo si en caso existieran los vértices “f”, “b”, “c” y “a”, el resultado que daría seria “a”, “b”, “c”, “f”.

```
def Ejecutar(self):

   self.LetrasANumeros()
   it = 0
   #Paso 1:  Seleccionamos la etiqueta del vértice con el valor numérico mas pequeño existente
   verticesOrdenados = sorted(self.Vertices,reverse=self.MayorOMenor)# ordenamos de menor a mayor los vértices
   self.pasos.append(verticesOrdenados[0])    # colocamos el vértice del primer paso
   self.Menor = self.Vertices.index(verticesOrdenados[0]) # necesitamos el índice para el vértice self.menors

   while self.Pasos:
    #Paso 2
    self.PasoRecursivo(it,True,0)
    it +=1

   self.NumerosALetras()
   self.Graficar(True)
   print('Pasos')
   p = 1
   for i in self.pasos:
    print(str(p)+')',i)
    p+=1
```

Posteriormente se analizan los enlaces en búsqueda de alguno repetido, es decir, si existe a →b y también b→a entonces solo se tomará en cuenta una de las dos para que no haya conexiones dobles a la hora de graficar.

```
def VerificarRepetidos(self,i,ordenar,Graficar):
      for j in range(len(self.pasos)):
        if j== i and Graficar:
           break
        if self.pasos[j] == ordenar[i]:

            return False   # si se repite
      return True  # no se repite
```

Después empieza la aplicación del algoritmo de búsqueda en profundidad el cual consiste en buscar un camino recibidor yendo de vértice en vértice del menor al mayor, siempre tomando la ruta del menor posible sin repetir vértices, una vez esto ya no es posible, se regresa en búsqueda de otro vértice que tenga un valor mayor, siempre sin repetir un vértice. Por ejemplo de existir a→b y a→d, el camino se ira por a→b y así con todos los vértices.

```
def PasoRecursivo(self,it,mas,actual):
    if it == 0:
     self.ordenar = sorted(self.Enlaces[self.Menor] ,reverse=self.MayorOMenor)

    else:
        #Paso 2 y seguimos la recursividad
        if mas:
         indiceFinal = len(self.pasos)-1
         self.Menor = self.Vertices.index(self.pasos[indiceFinal]) #conseguimios el índice del ultimo paso
         self.ordenar = sorted(self.Enlaces[self.Menor],reverse=self.MayorOMenor)

    for i in range(len(self.Enlaces[self.Menor])):
      if self.VerificarRepetidos(i,self.ordenar,False):
       if mas ==False:
        self.pasos.append(actual)
       self.pasos.append(self.ordenar[i])
       if mas == False:
          self.para = False
          self.Pasos = True
       break
      elif i == (len(self.Enlaces[self.Menor])-1):
       if mas:
        self.Pasos = False
        self.Retroceso()

       break
```

```
def Retroceso(self):
    i = len(self.pasos)-1
    self.para = True
    while self.para:
        self.Menor = self.Vertices.index(self.pasos[i])
        self.ordenar = sorted(self.Enlaces[self.Menor],reverse=self.MayorOMenor)
        self.PasoRecursivo(1,False,self.pasos[i])
        if i == 0:
            break
        i-=1
```