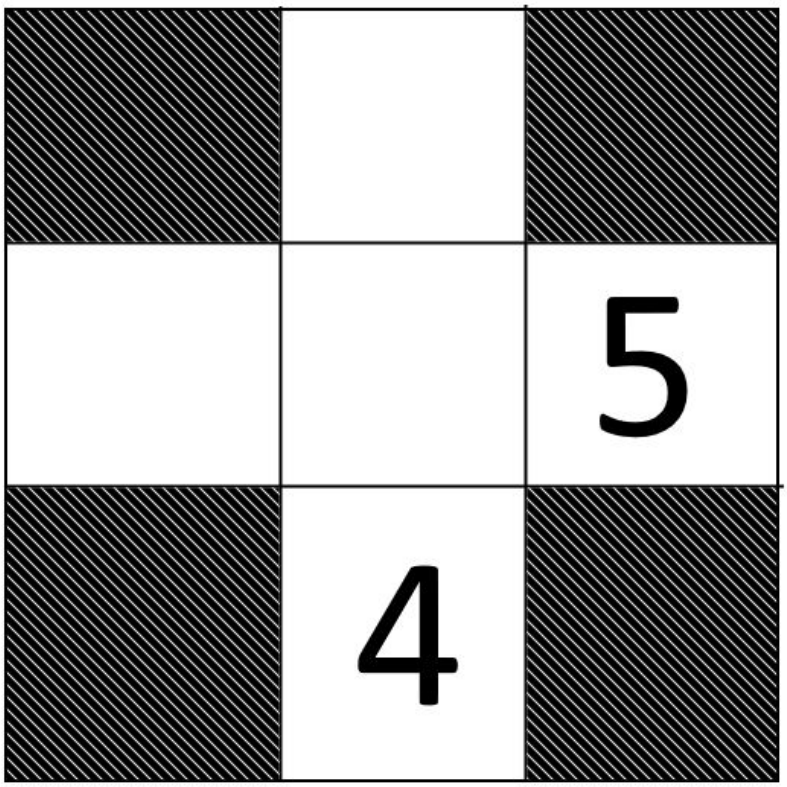
Inteligencia Artificial

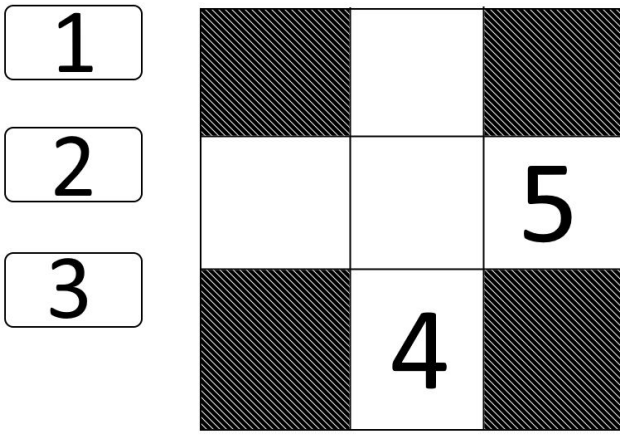
ROMPECABEZAS NUMÉRICO



Integrante:	Narváez Marqueda Ricardo André Sebastián		
Número de cuenta	41705242-5		
Grupo 5		Fecha de ejecución:	12/11/2020
		Fecha de entrega	26/11/2020
Profesora	Dra. MARIA DEL CARMEN EDNA MARQUEZ MARQUEZ		
Semestre	2021-1		

Objetivos

Resolver el siguiente rompecabezas numérico por medio de una búsqueda ciega por amplitud.



La suma Horizontal y Vertical debe resultar en 9

- La búsqueda será realizada por un programa que ejecute los operadores para generarlos estados.
- Utilizar un lenguaje procedural (Python).
- El orden en que se crean los estados y el orden en que se revisan.
- Y al final dar la ruta de solución.

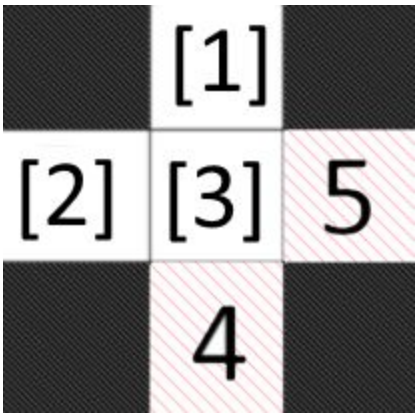
Para el manejo del problema realizaremos una representación en forma de lista de la siguiente manera (a motivo de no manejar matrices con espacios vacíos).

Representación por A,B,C para los espacios en las columnas.

Esto en forma de lista sería de la siguiente manera:

[A,B,C,5,4]

O bien, indexando la lista anterior:



1

2

3

[<1>,<2>,<3>,5,4]

O bien en python:

nodo.data[0],nodo.data[1],nodo.data[2]...

Ahora bien una vez definido esto, podemos concretar que el caso de éxito es cuando las siguientes ecuaciones se cumplan de manera satisfactoria:

$$A + C + 4 = 9 \quad : Suma\ Vertical$$

$$B + C + 5 = 9\ Suma\ Horizontal$$

Como sabemos que tenemos 3 elementos (3 números en este caso), tendremos los siguientes operadores:

Colocar **1** en [**A**]

Colocar **1** en [**B**]

Colocar **1** en [**C**]

Colocar **2** en [**A**]

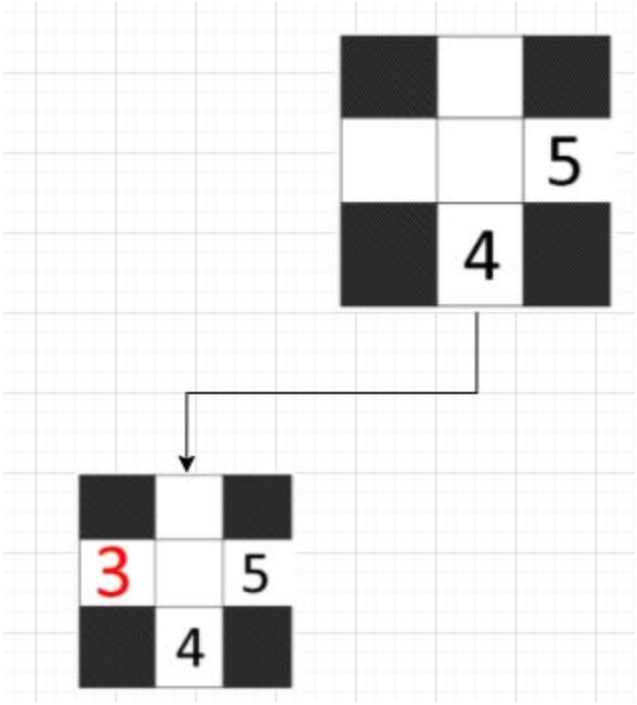
Colocar **2** en [**B**]

...

Colocar **3** en [**C**]

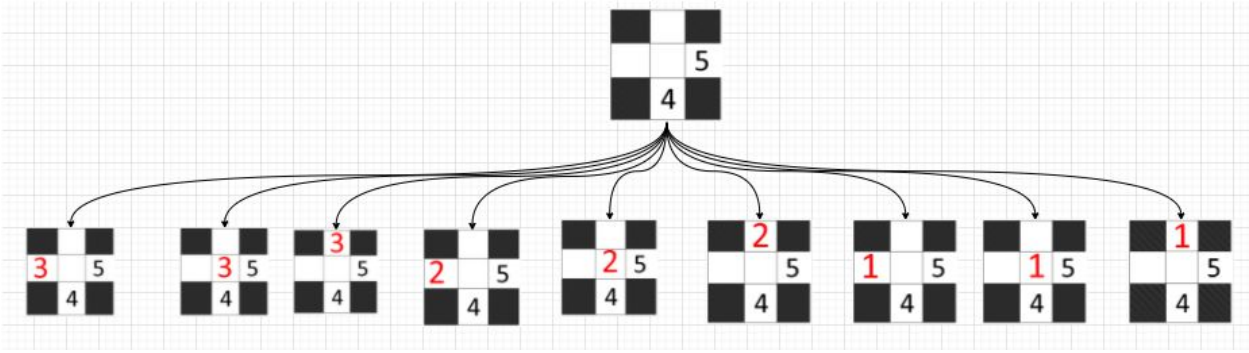
Tal como se definió en la clase, el orden de los operadores puede ser tanto aleatorio, como fijo para el desarrollo de la estructura árbol, y se tiene en consideración que una vez aplicado un operador, éste no puede ser repetido la sucesión de nodos hijos, por mencionar un ejemplo sencillo:

Operador Colocar 3 en [**B**]



Tanto la casilla B como el número 3 no podrán ser utilizados para los hijos, por lo que solo nos quedarán los operadores: Colocar [1,2] en [A,C].
Al ser una búsqueda en amplitud los siguientes operadores a aplicar será regresar un nivel anterior y aplicar el operador Colocar [1,2] en [A,C]. hasta terminar la capa actual del árbol.

Solo por mostrar gráficamente lo que se debe de obtener teóricamente:



Primer nivel del árbol al desarrollar la búsqueda ciega en amplitud.

Representación en forma de lista:

[_,_,_,5,4] → *Raíz*

- *[3,_,_,5,4]* → *Hijos*
- *[_,3,_,5,4]*
- *[_,_,3,5,4]*
- *[1,_,_,5,4]*
- *[_,1,_,5,4]*

...

Implementación:

Realicé un programa cuasi escalable (debido a que el tamaño del arreglo y las constantes pueden ser modificadas):

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 12 22:28:14 2020
PÝTHON 3.X.X
No dependencies needed
@author: Narváez Marqueda Ricardo André Sebast (RAM)
"""

INITIAL_STATE=[0,0,0,5,4] # Los ceros representan un espacio aún no ocupado
CONSTANTES=[1,2,3]# Números a colocar en las casillas desocupadas
```

La clase nodo la cual contendrá la bitácora de la aplicación de los operadores:

```
class Node:

    def __init__(self, data):#Declaración del nodo
        self.data = data #Data será la lista que contendrá el estatus actual de las casillas
        self.hijos=[] #Hijos será una lista dado que es un arbol no binario

    #Imprimir valor del nodo actual
    def printValue(self):
        print("Node Value",self.data)

    #Insertar un hijo
    def insertNode(self,node):
        self.hijos.append(node)
        node.padre=self
    #Imprime el estatus general dado un nodo
```

En ella encontraremos los métodos básicos de una estructura de tipo árbol tal como:

La inicialización de la estructura__init__: El cual dado que es un arbol no binario (puede contener una o más ramas), por lo que tendrá una lista de referencia a memoria respectivamente a los hijos.

PrintValue: Imprimirá en pantalla el estado actual del nodo que estemos recorriendo

InsertNode: El cual inserta un nodo por llamada

Más adelante tenemos la función operadorRaiz, el cual es el encargado de realizar la operación de tomar las constantes disponibles y ubicarlas en las casillas. En este caso opté por una implementación ordenada y bien definida, aunque si cambiamos *operadoresDisponibles[i] por operadoresDisponibles[randomSel(0,2)]* donde *randomSel(0,2)* tomará una constante aleatoria aún no utilizada en dicha capa, aleatorizando el proceso.

```
#Función que aplica los operadores de manera recursiva
def operadorRaiz(nodo):
    estadoActual=nodo.data[:]#Se obtiene el estado actual del nodo
    operadoresDisponibles=CONSTANTES[:]#Se mantiene una bitácora de los operadores
    for i in range(3):#El rango es a tres dado que por la implementación de la lista solo los primeros 3 lugares son las casillas a utilizar
        if nodo.data[i]!=0:#Si ya se utilizó un operador en alguno de las casillas disponibles
            operadoresDisponibles.remove(nodo.data[i])#Se remueve de los operadores disponibles a utilizar

    if len(operadoresDisponibles)!=0:#Si ya no quedan operadores, el nodo en materia es una hoja
        for i in range(len(operadoresDisponibles)):#Operadores disponibles
            estadoActual=nodo.data[:]
            for j in range(estadoActual.count(0)): #Contando el número de espacios vacios lo cual será equivalente al número de hijos
                estadoActual=nodo.data[:]
                if estadoActual[j]==0:#Si la casilla está vacía
                    estadoActual[j]=operadoresDisponibles[i]#Entonces se aplica el operador
                    hijo=Node(estadoActual)#Se crea el nuevo nodo, con el operador aplicado
                    nodo.insertNode(hijo)#Se inserta el nodo como hijo
                else:
                    pass
        for i in range(len(nodo.hijos)):
            operadorRaiz(nodo.hijos[i])#Repetir este procedimiento para cada uno de los hijos
```

Una vez generada y bitacorizada la creación del arbol, podemos comenzar con el recorrido a través de BFS.

```
def BFS(nodo,flag):
    if flag==False: #Si aún no se ha encontrado la solución, continuar iterando
        sumaVertical=nodo.data[0]+nodo.data[2]+nodo.data[4]#Comprobar que se cumpla la suma vertical del nodo actual
        sumaHorizontal=nodo.data[1]+nodo.data[2]+nodo.data[3]#Comprobar que se cumpla la suma horizontal del nodo actual
        if sumaHorizontal==9 and sumaVertical==9:#Si se cumplen ambas:
            print("SOLUCIÓN ENCONTRADA: ",nodo.data)#Se imprime la solución
            flag=True#No se itera más y se realiza la impresión recursiva de la ruta solución
        else:#En caso contrario
            for i in range (len(nodo.hijos)):#Seguir recorriendo los hijos
                if len(nodo.hijos)!=0:#Si el nodo no es una hoja
                    nodo.hijos[i].printValue()#imprimir su valor
            for i in range(len(nodo.hijos)):
                    if flag==True:#Si aun no se ha encontrado una solución
                        pass
                    else:
                        BFS(nodo.hijos[i],flag=False) #Continuar recursivamente analizando los hijos siempre y cuando no sea una hoja, en ese caso
                                                      #Regresar un nivel y analaizar los nodos hermanos
```

Salida de ejecución del programa:

```
In [26]: runfile('C:/Users/andre/Desktop/AI Puzzle/AI_Puzzle.py', wdir='C:/Users/andre/
Desktop/AI Puzzle')
Iniciando creación del árbol
Aplicando operadores y realizando bitácora de desarrollo:
Buscando ruta solución:
Node Value [1, 0, 0, 5, 4]
Node Value [0, 1, 0, 5, 4]
Node Value [0, 0, 1, 5, 4]
Node Value [2, 0, 0, 5, 4]
Node Value [0, 2, 0, 5, 4]
Node Value [0, 0, 2, 5, 4]
Node Value [3, 0, 0, 5, 4]
Node Value [0, 3, 0, 5, 4]
Node Value [0, 0, 3, 5, 4]
Node Value [1, 2, 0, 5, 4]
Node Value [1, 3, 0, 5, 4]
Node Value [2, 1, 0, 5, 4]
Node Value [3, 1, 0, 5, 4]
Node Value [2, 0, 1, 5, 4]
Node Value [0, 2, 1, 5, 4]
Node Value [3, 0, 1, 5, 4]
Node Value [0, 3, 1, 5, 4]
Node Value [3, 2, 1, 5, 4]
Node Value [2, 3, 1, 5, 4]
Node Value [2, 1, 0, 5, 4]
Node Value [2, 3, 0, 5, 4]
Node Value [1, 2, 0, 5, 4]
Node Value [3, 2, 0, 5, 4]
Node Value [1, 0, 2, 5, 4]
Node Value [0, 1, 2, 5, 4]
Node Value [3, 0, 2, 5, 4]
Node Value [0, 3, 2, 5, 4]
Node Value [3, 1, 2, 5, 4]
Node Value [1, 3, 2, 5, 4]
Node Value [3, 1, 0, 5, 4]
Node Value [3, 2, 0, 5, 4]
Node Value [1, 3, 0, 5, 4]
Node Value [2, 3, 0, 5, 4]
Node Value [1, 0, 3, 5, 4]
Node Value [0, 1, 3, 5, 4]
Node Value [2, 0, 3, 5, 4]
Node Value [0, 2, 3, 5, 4]
Node Value [2, 1, 3, 5, 4]
Node Value [1, 2, 3, 5, 4]
SOLUCIÓN ENCONTRADA, DETENIENDO PROCESO:
 [2, 1, 3, 5, 4]
Colocar  2  En la casilla 0
Colocar  1  En la casilla 1
Colocar  3  En la casilla 2
```

De acuerdo a la manera en que asigné las casillas y operadores, la solución gráficamente se visualizará de la siguiente manera.

Solución descrita por mi programa