INSTITUTO POLITECNICO NACIONAL

Escuela Superior de Cómputo

PRACTICA 3

Autor:
Gomez Hernandez Alan Javier

12 octubre, 2021—

PROGRAMA COMBINATORIO

RESUMEN

Elaborar un programa para realizar movimientos ortogonales y diagonales en un tablero de ajedrez de 4x4 con una pieza. Los movimientos y las reglas están explicadas en las láminas del curso de Stanford.Adicionalmente, el programa debe de contar con las siguientes características: 6. Una vez definida la cadena de movimientos para una pieza, se deben generar los archivos de todos los movimientos posibles y generar otro archivo con todos los movimientos ganadores. 7. Dibujar el tablero y mostrar los movimientos de dos jugadas seleccionadas aleatoriamente del archivo de movimientos ganadoras. Para el caso de la animación pueden intentar poner la pieza del rey con bitmap o dibujar un circulo dentro del cuadrado, para posteriormente despintar y pintar el círculo, de manera que parezca que se mueve. Sugerencia: para dibujar el tablero utilizar la función de la librería gráfica que refiera un cuadrado y aplicar un fill para ponerle el color.

INTRODUCCION

a través de Microsoft Store utiliza el intérprete básico de Python3, pero maneja la configuración de su PATH para el usuario actual (evitando la necesidad de acceso de administrador), además de proporcionar actualizaciones automáticas. Python es un lenguaje de programación sumamente utilizado en el análisis, interpretación y visualización de datos, diferentes tipos de datos financieros y el Machine learning. Además, una de las ventajas más significativas es que su curva de aprendizaje es corta por lo que se pueden realizar diferentes tipos de tareas nuevas en solamente unos meses. Es posible ejecutar Python de diferentes maneras, por lo que cada usuario deberá realizarlo según las necesidades que posea. Sin embargo, es importante destacar que la mayoría de los sistemas de Python ya vienen instalados, por lo que no requiere de acción alguna. Es posible ejecutar Python desde la terminal o línea de comando IDE o bien emplear opciones en la nube que incluya: Cuadernos Jupyter Google Colab Cada una de las opciones mencionadas proporcionan una experiencia sumamente fácil, ideal para que quienes no tengan experiencia puedan aprender y probar elementos de código de una manera más sencilla. Emplear Python puede traer aparejado una gran diversidad de beneficios de diferentes índoles para cualquier clase de compañía, debido a los diferentes tipos de prestaciones que se pueden llevar a cabo con el mismo. Ibstalar Python es una excelente manera de automatizar los procesos de manera simple, rápida y sencilla. Todas las tareas que están programadas en Python podrán llevarse a cabo de manera precisa y sin errores posibles, ahorrando tiempo y dinero. Si bien los software de avanzada tecnología

permiten que la compañía pueda desempeñar tareas en masa a gran escala, es fundamental que todo esté controlado por personal idóneo para que el éxito esté asegurado. El recurso humano de cualquier empresa es una de las valoraciones más altas que posee, debido a que no se puede replicar en masa y lleva años de formación intelectual. Entonces aprovechando esto crearemos nuestro ajedrez para obtener el automata y el grafo.

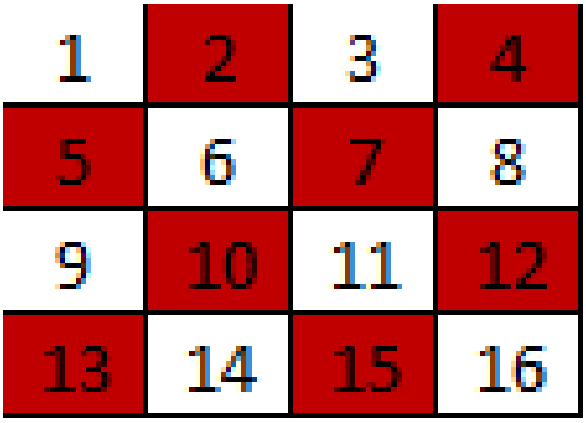

Figure 1: 4x4

Numero	Rojo	Blanco
1	2,5	6
2	5,7	1,3,6
3	4,2,7	6,8
4	7	3,8
5	2,10	1,6,9
6	2,5,7,10	1,3,9,11
7	2,4,10,12	3,11
8	4,7,12	3,11
9	5,10,13	6,14
10	5,7,13,15	6,9,11,14
11	7,10,12,15	6,8,14,16-
12	7,15	8,11,16-
13	10	9,14
14	10,13,15	9,11
15	10,12	11,14,16-
16	15,12	11

DESARROLLO

0.1 CODIGO

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 12 04:41:54 2021

@author: user
"""

import random
import string
from graphviz import Digraph
import chess
import chess.svg
from IPython.display import SVG, display

class NFA(object):
  def __init__(self, Q, sigma, delt, q_0, F):
    """
        Q: Conjunto finito llamado "de estados" #Numero de estados (Int)
        sigma : Conjunto finito llamado Alfabeto #Lista
        delta : Funcion de transicion tal que Q x sigma => P(Q x Sigma) #Diccionario de list
        q_0 : Estado inicial #String
        F:  Estados de aceptacion #Lista de strings
    """
    '''
        delta={
              "q1":{"0":None, "1": ["q2","q3"]},
              "q2":{"0":"q5", "1": None},
              "q3":{"0":"q4", "1": None},
              "q4":{"0":"q3", "1": None},
              "q5":{"0":["q5",q6"], "1" : None},
              "q6":{"0":"q2", "1": None}
              }
        automata_finito_no_det = NDFA(Q = 6,
                    sigma=[0],
                    delt = delta,
                    q_0 = ["q1"],
                    F = ["q2","q3"])
    '''
    Q_list = []
    for i in range(Q):
        Q_list.append("q"+str(i+1))
    self.Q = Q_list
```

```
  self.sigma = sigma
  self.delta = delt
  self.completar_diccionario()
  self.q_0 = q_0
  self.F = F
  self.estado_actual = [q_0]
  self.camino = []
def get_estado_actual(self):
    return self.estado_actual

def set_estado_actual(self, estado_actual):
  self.estado_actual = estado_actual

def get_F(self):
  return self.F

def completar_diccionario(self):
  sigmaTemp = self.sigma
  sigmaTemp.append(chr(1013))
  for i in self.delta.keys():
    if "rest" in self.delta[i].keys():
      for j in self.sigma:
        if j == chr(1013) and not (j in self.delta[i].keys()):
          self.delta[i][j] = None
          continue
        if not (j in self.delta[i].keys()):
          self.delta[i][j] = self.delta[i]["rest"]
      self.delta[i].pop("rest")
    else:
      for j in sigmaTemp:
        if not j in self.delta[i].keys():
          self.delta[i][j] = None

def prueba(self, cadena):
  indice = 0
  for i in cadena:
    #print("Bit leido de la cadena: " + i)
    estado_actual = self.get_estado_actual()
    print("Estado actual: " + str(estado_actual))
    estado_siguiente = []

    for estados in estado_actual:
      if estados == None:
        continue
      conexionesTemp = self.delta[estados][i]
      if conexionesTemp != None:
```

```
        if isinstance(conexionesTemp, list):
          for sig_estados in conexionesTemp:
            estado_siguiente.append(sig_estados)
        else:
          estado_siguiente.append(conexionesTemp)
      if conexionesTemp == None:
        estado_siguiente.append(None)
      episilon = conexionesTemp = self.delta[estados][chr(1013)]
      if episilon == None:
        continue
      if isinstance(episilon ,list):
        for estados in episilon:
            estado_siguiente.append(estados)
      else:
        estado_siguiente.append(episilon)

    #print("Los siguientes estados " + str(estado_siguiente))
    self.set_estado_actual(estado_siguiente)
    indice +=1
  for i in self.get_estado_actual():
    if i in self.get_F():
      return True
  return False


def pruebaRecursiva(self, cadena, estadoActual):
  if cadena == "":
    if estadoActual in self.get_F():
      return True
    else:
      return False
  siguientesEstados = self.delta[estadoActual][cadena[0]]
  if isinstance(siguientesEstados, list):
    for estado in siguientesEstados:
      if self.pruebaRecursiva(cadena[1:],estado):
        self.camino.append(estado)
        return True

  else:
    if self.pruebaRecursiva(cadena[1:], siguientesEstados):
      self.camino.append(siguientesEstados)
      return True
    else:
      return False

def drawn(self):
```

```
    # inicializa el diagrama
    f = Digraph('finite_state_machine', filename='fsm.gv', format='png')
    f.attr(rankdir='LR', size='8,5')
    # Dibuja los nodos finales con doblecirculo
    f.attr('node', shape='doublecircle')
    for i in self.F:
        f.node(i)
    f.attr('node', shape='circle')
    f.attr('edge', )
    # Agrega todos los nodos de todos los estados
    for key in self.delta.keys():
        f.node(key)
    #Agrega todas las conexiones
    #Cuando una misma conexion tiene varios inputs los une en una sola flecha
    for estado, conexiones in self.delta.items():
      for nombre, conex in conexiones.items():
        if conex == None:
          continue
        if isinstance(conex,list):
          for i in conex:
            f.edge(estado,i,nombre)
        else:
          f.edge(estado,conex,nombre)


    #Se crea un nodo transparente para poner la flecha del estado inicial
    f.attr('node', style='filled')
    f.attr('node', color='white')
    f.edge('', self.q_0)
    f.view()


print("Bienvenido al ajedrez!")
print("Elija una opcion:\n1.-Desea generar cadenas aleatorias\n2.-Desea ingresar las cadenas
opc = input()
if opc == "1":
  elegir=False
else:
  elegir = True


def generarCadena(ingresar = False, automata = None, estado = None):
  if not ingresar:
    cad = ""
    tam = random.randint(2,10)
    for i in range(tam):
```

```
      if random.randint(0,1):
        cad+="r"
      else:
        cad+="b"
    cad+="b"
    return cad
  while 1:
    cad = input("Ingresa tu cadena: ")
    if automata.pruebaRecursiva(cad, estado):
      #automata.camino.clear()
      automata.camino.reverse()
      return cad
    else:
      print("Cadena no valida")
      automata.camino.clear()

q = 9
sigma = ["r", "b"]
delta = {
    "q1" : {"r": ["q2","q4"], "b": "q5"},
    "q2" : {"r": ["q4","q6"], "b": ["q1","q3","q5"]},
    "q3" : {"r": ["q2","q6"], "b": "q5"},
    "q4" : {"r": ["q2","q8"], "b": ["q1","q5","q7"]},
    "q5" : {"r": ["q2","q4","q6","q8"], "b": ["q1","q3","q7","q9"]},
    "q6" : {"r": ["q2","q8"], "b": ["q3","q5","q9"]},
    "q7" : {"r": ["q4","q8"], "b": "q5"},
    "q8" : {"r": ["q4","q6"], "b": ["q5","q7","q9"]},
    "q9" : {"r": ["q6","q8"], "b": "q5"},
}
q_0 = "q1"
F = ["q9"]
automata1 = NFA(q, sigma, delta, q_0, F)
q_0N = "q9"



'''if elegir:
  cadenaBlanco=generarCadena(True, automata1,automata1.q_0)
  cadenaNegro=generarCadena(True, automata2, automata2.q_0)
'''

def recalcularRuta(automata, estadoA):
  print("Recualculando Ruta...")
  while 1:
    automata.camino.clear()
    if elegir:
```

```
      nuevaRuta = generarCadena(True, automata, estadoA)
      break
    else:
        nuevaRuta = generarCadena()
    if automata.pruebaRecursiva(nuevaRuta, estadoA):
      automata.camino.reverse()
      return
recalcularRuta(automata1, automata1.q_0)
print("El camino que tomara el blanco es: "+ str(automata1.camino))


casillas = {
  "q1": "a8",
  "q2": "b8",
  "q3": "c8",
  "q4": "a7",
  "q5": "b7",
  "q6": "c7",
  "q7": "a6",
  "q8": "b6",
  "q9": "c6",
}
board = chess.Board()
board.clear()
board.set_piece_at(chess.A8, chess.Piece.from_symbol("K"))
board.set_piece_at(chess.C6, chess.Piece.from_symbol("k"))
display(SVG(chess.svg.board(board=board)))

#Vamos a jugar!!!!!!!!

turno = 0
i=0 #indice de la lista del automata1

estadoActual1 = automata1.q_0

while True:
    #Tira el blanco
  if turno == 0:
    siguienteJugada = automata1.camino[i]
    print("Turno del blanco")
    print("Esta en la casilla: " + estadoActual1+ " La siguiente jugada es: " + siguienteJug
    i+=1
    movimiento = chess.Move.from_uci(casillas[estadoActual1]+casillas[siguienteJugada])
    board.push(movimiento)
    display(SVG(chess.svg.board(board=board)))
    estadoActual1 = siguienteJugada
```

```
    if siguienteJugada in automata1.F:

        break
turno = 1 - turno
```

0.2 FUNCIONAMIENTO Y PRUEBAS

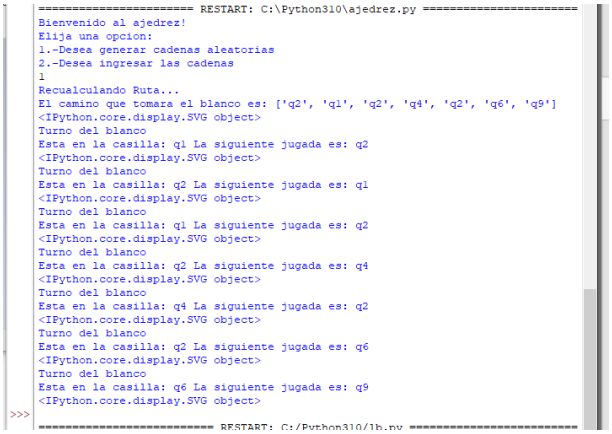

Figure 2: Se realiza la prueba y guarda el grafo

CONCLUSION

Para esta practica se tuvieron varias dificultades, las cuales fue realizar y crear el autómata que se acoplara correctamente a las especificaciones, la primera fue que dentro del programa pudiera generar pasos de manera aleatoria para llegar al final, porque llegaba a entrar a ciclos finitos por no obtener un valor de salida y quedara registrado, eso lo solucionamos con una lista de cadenas para llevar el control de estados, como también de donde inicia y donde sale, con esto mismo se crean funciones cada ves que termina un proceso. Segundo el movimiento de piezas que eran diagonales se tomaban como un mismo arreglo total, pero se empalmaban con otras, se separaron y quedo esta parte. Finalmente, para este autómata podemos ver trabajar y como se movía la pieza según los datos ingresados. Para esta práctica hizo falta un mejor funcionamiento y complemento para finalizarla de forma correcta.