Universidad Nacional Autónoma de México

Facultad de Estudios Superiores Aragón

Proyecto Segundo Parcial

Equipo 2

Integrantes:

Monzón Lucero Miguel Ángel

Robles Leon Cristopher Ruben

Grupo: 1509

Materia:

Diseño y Análisis de Algoritmos

Profesor:

Miguel Angel Sanchez Hernandez

Índice

Introducción

En este trabajo se presenta la documentación de los algoritmos solicitados y revisados durante la clase de Diseño y Análisis de Algoritmos, se harán de una forma ordenada y tratando de seguir el mismo formato y procedimiento en cada uno de ellos. Esta documentación incluye aspectos como el objetivo, el modelo, el pseudocódigo, el algoritmo utilizado, las entradas y salidas del algoritmo y formas en las que creemos que se puede aplicar estos algoritmos a diferentes situaciones; en esta ocasión no se toma en cuenta la función de complejidad de los algoritmos, ya que el propósito de este análisis es comprender los algoritmos solicitados y exprimirlos; además de este análisis se incluyen “simulaciones” de los algoritmos y los códigos de cada algoritmo como evidencia de su implementación. Al final del documento también se agrega una conclusión donde agregamos de manera crítica en qué aspectos tuvimos deficiencias y en general un análisis de nuestro desempeño a lo largo de la realización del proyecto.

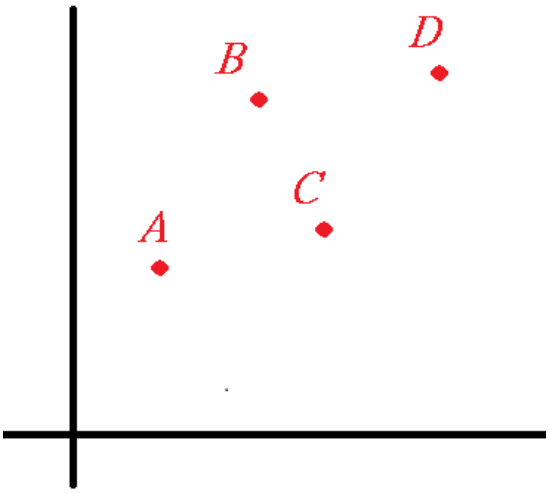
Objetivos

- Como principal objetivo queremos resolver de manera satisfactoria todos y cada uno de los algoritmos, ya que consideramos que en especial el algoritmo de rangos, es complejo y podríamos tener problemas en este aspecto.
- Nuestro siguiente objetivo es poder comprender cada uno de los algoritmos que analizaremos y entender cómo es que podemos exprimir estos algoritmos para que se puedan aplicar de otras maneras.
- Como tercer objetivo queremos poder llegar a comprender el funcionamiento de los 3 algoritmos y el conocimiento que hay detrás de estos, ya que consideramos que no solo es saber que, si no saber cómo es que podemos implementarlos para distintos fines a los solicitados.
- Como último objetivo lo que buscamos es poder programar estos algoritmos, ya que buscamos exprimirlos y sacar información a través de estos, además que, para simularlos debe haber un programa, entonces aunque sea nuestro último objetivo, es de los más importantes para nosotros.

Análisis Algoritmo “Rangos 2 dimensiones”

Objetivo

Desarrollar y analizar un algoritmo que resuelva la problemática de encontrar los rangos de cada punto en un conjunto finito de puntos.



Se denomina rango a la serie de puntos que cumplan las siguientes condiciones
dado una serie de puntos (x 1 ,y 1),(x 2 ,y 2)
(x 1 ,y 1) tiene como rango a (x 2 ,y 2) si y sólo si **x1>x2** y **y1>y2**
Si tenemos que A no domina a B ni B domina A, entonces A y B no se pueden comparar

- B domina A
- D domina A,B,C
- C domina A

Restricciones

- Los rangos se calculan tomando un punto y de ahí hacia la izquierda
- La lista debe de tener al menos dos puntos para que tengan rangos.
- dado una serie de puntos (x 1 ,y 1),(x 2 ,y 2), (x 1 ,y 1) tiene como rango a (x 2 ,y 2) si y sólo si **x1>x2** y **y1>y2**
- Si tenemos que A no domina a B ni B domina A, entonces A y B no se pueden comparar
- no se pueden ingresar letras a la lista

Modelo

Entradas:

-se ingresa una serie de puntos los cuales son con el formato [x , y]
-se selecciona un punto con ayuda de la gráfica

Proceso:

1. se recorren los puntos y a la par se hacen las comparaciones
2. se muestra una gráfica con el resultado, con ayuda del método *text* de la librería *matplotlib*

Restricción:

no puede ser una lista vacía

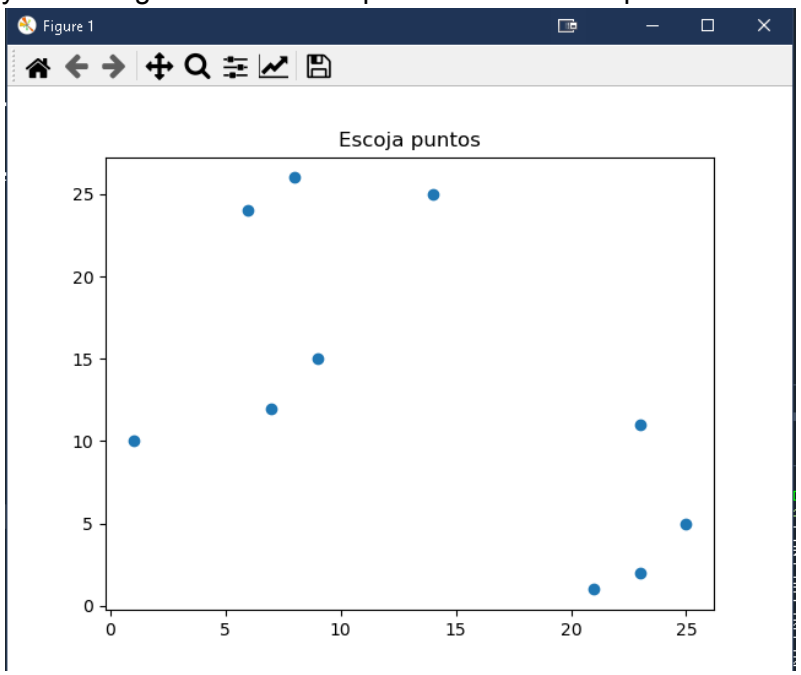
no se pueden ingresar letras a la lista

Descripción del Algoritmo

el algoritmo lo que hace es agarrar el punto seleccionado por el usuario y recorrer la lista desde el elemento 0 hasta el elemento de la lista seleccionado, haciendo las respectivas comparaciones para ver si son menores los puntos por lo que pasa con ayuda de dos for que son los que hacen el recorrido y con if hacen las comparaciones.

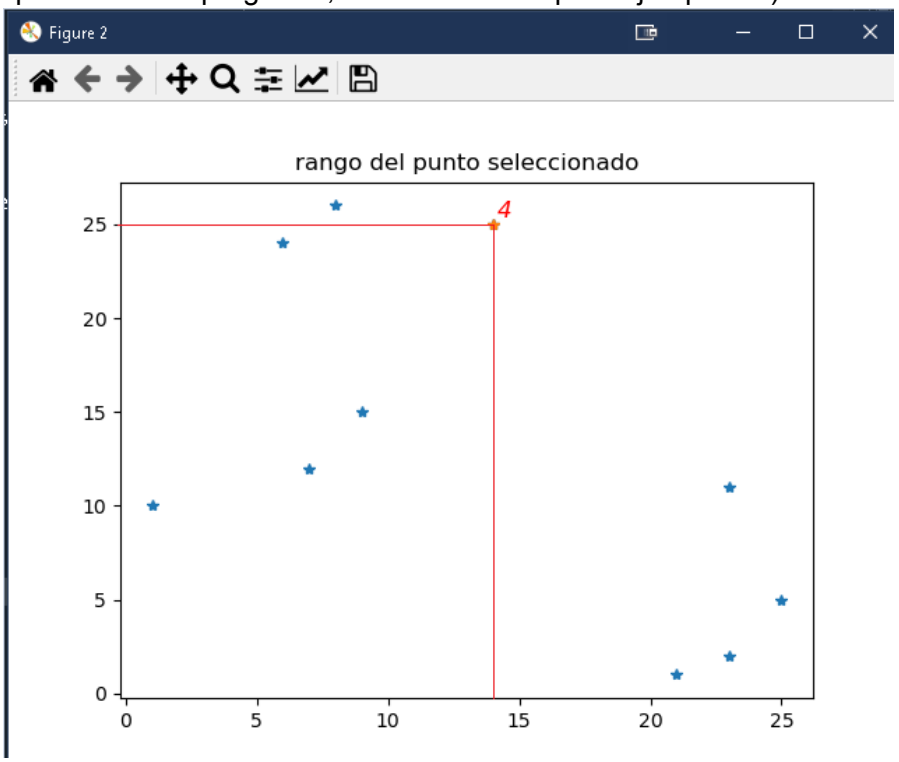
Entrada

se toman como entradas la lista lista=[[21, 1], [8, 26], [6, 24], [25, 5], [14, 25], [9, 15], [23, 2], [24, 11], [1, 10], [7, 12]]
y como segunda entrada el punto seleccionado por el usuario con ayuda de la gráfica



Salida

una vez seleccionado un punto automáticamente nos aparece una segunda gráfica en la cual nos remarca el punto seleccionado teniendo a su lado su rango
ayudándonos de líneas para mostrar el proceso de cómo calcula los rangos(dichas líneas no aparecen en el programa, son meramente para ejemplificar)



de igual manera se adjunta una tabla con los rangos para este ejemplo.

x	y	rango
1	10	**0**
6	24	**1**
7	12	**1**
8	26	**3**
9	15	**2**
14	25	**4**
21	1	**0**
23	2	**1**
24	11	**3**
25	5	**2**

Código

Se utilizó el método de ordenamiento de *quicksort* para ordenar la lista de puntos con respecto a 'x' y facilitar el cálculo de los rangos.

```
def metodo_quicksort(lista):
   quicksort_auxiliar(lista,0,len(lista)-1)

def quicksort_auxiliar(lista,primero,ultimo):
   if primero<ultimo:
       puntoDivision = particion(lista,primero,ultimo)
       quicksort_auxiliar(lista,primero,puntoDivision-1)
       quicksort_auxiliar(lista,puntoDivision+1,ultimo)

def particion(lista,primero,ultimo):
   valorPivote = lista[primero]
   marcaIzq = primero+1
   marcaDer = ultimo
   hecho = False
   while not hecho:
       while marcaIzq <= marcaDer and lista[marcaIzq] <= valorPivote:
           marcaIzq = marcaIzq + 1
       while lista[marcaDer] >= valorPivote and marcaDer >= marcaIzq:
           marcaDer = marcaDer -1
       if marcaDer < marcaIzq:
           hecho = True
       else:
           temp = lista[marcaIzq]
           lista[marcaIzq] = lista[marcaDer]
           lista[marcaDer] = temp
   temp = lista[primero]
   lista[primero] = lista[marcaDer]
   lista[marcaDer] = temp

   return marcaDer
```

También se usó la librería *matplotlib* para la representación de los puntos en una gráfica y de igual forma mostrar el rango de cada punto según se le de click.

```
def graficas(puntos):
    for indice in range(len(puntos)):
        x.append(int(puntos[indice][0]))
        y.append(int(puntos[indice][1]))
    line, =ax.plot(x,y,'o',picker=5)

    puntosSeleccionado=fig.canvas.mpl_connect('pick_event',onpick)
    plt.show()
def onpick(event):
    thisline=event.artist
    xdata=thisline.get_xdata()
    ydata=thisline.get_ydata()
    ind=event.ind
    listaPunto=[]
    puntoSeleccionado=tuple(zip(xdata[ind],ydata[ind]))
    listaPunto.append(puntoSeleccionado[0][0])
    listaPunto.append(puntoSeleccionado[0][1])
    print(listaPunto)
    segundaVentana(listaPunto)

def segundaVentana(listaPunto):
    figDos=plt.figure()
    a=figDos.add_subplot(111)
    a.set_title('rango del punto seleccionado')
    puntos=encontrar_rangos(listaPunto[0])
    a.text(listaPunto[0]+.7, listaPunto[1]+.7, puntos, fontsize=12, color = "r", style = "italic", weight = "light", vertic
    print(puntos)
    line, =a.plot(x,y,'*',picker=5)
    line, =a.plot((listaPunto[0]),(listaPunto[1]),'*', picker=5)
```

función para encontrar el rango dependiendo de un punto de la lista

```
def encontrar_rangos(punto):
    rango=0
    for i in range(len(lista)):
        if punto == lista[i][0]:
            j=punto
            for j in range(i):

                if lista[j][0]<lista[i][0]:
                    if lista[j][1]<lista[i][1]:
                        rango+=1
    return rango
```

Pseudocódigo

encontrar_rangos(punto_seleccionado)
rango=0 esta variable es a la que se le irá sumando con cada comparación
for(i=0; i<longitud de lista; i++){
if(punto selecciona == a lista [i][0])
j=punto seleccionado
for(j ; j<i; i++){
if (lista[j][0]<lista[i][0] and lista[j][0]!=lista[i][0]:){ x
if (lista[j][1]<lista[i][1]){ y
rango+=1
}
}
}
return rango

Conclusiones

para el objetivo planteado, podemos concluir que este se logró, ya que se pudo encontrar la solución para que el algoritmo nos diera el rango de los puntos.
Sabiendo que este tipo de algoritmos tiene muchas posibilidades de adaptación para poder utilizarse desde un enfoque diferente, tal vez para hacer gráficas de ventas y saber cuáles fechas es donde le fue mejor a dicha empresa comparando las fechas anteriores y tener un aproximado de cuales son las mejores.

Análisis Algoritmo “Juego Gallina, Maíz y Lobo”

Objetivo

El objetivo de este algoritmo es crear un sistema experto que contenga las reglas de esta especie de juego/acertijo, y que pueda determinar cuándo se puede ganar o perder en un juego. Finalmente se busca poder “jugar” y que el sistema experto determine con qué movimientos ganamos o perdemos.

Restricciones

Este algoritmo su única restricción es que solo se pueden elegir las opciones con los números que se marcan en el sistema experto

Modelo

Entradas:

Cualquier número entre 0 y 3, que corresponden a los movimientos del juego.

Proceso:

1. El algoritmo recibirá el número de entrada del usuario.
2. Esta decisión se evaluará en las funciones del sistema experto
3. Devolverá el estado de los personajes, y una lista de las nuevas opciones que tiene el usuario

Descripción del Algoritmo

Nuestro algoritmo se basa en funciones que corresponde a cada acción que se puede realizar en el juego, dentro de estas funciones se especifican las “reglas” que sigue el algoritmo para determinar cuándo es que 2 personajes pueden estar juntos o no, y así se determina cuando la gallina se come el maíz o el lobo a la gallina. El algoritmo funciona hasta que algún personaje muera, o todos se encuentren al lado derecho del río, esto nos da la posibilidad de que el usuario pueda hacer cuantos movimientos sean necesarios para conseguir ganar o perder, aunque podría perderse en el camino y dar vueltas sin sentido.

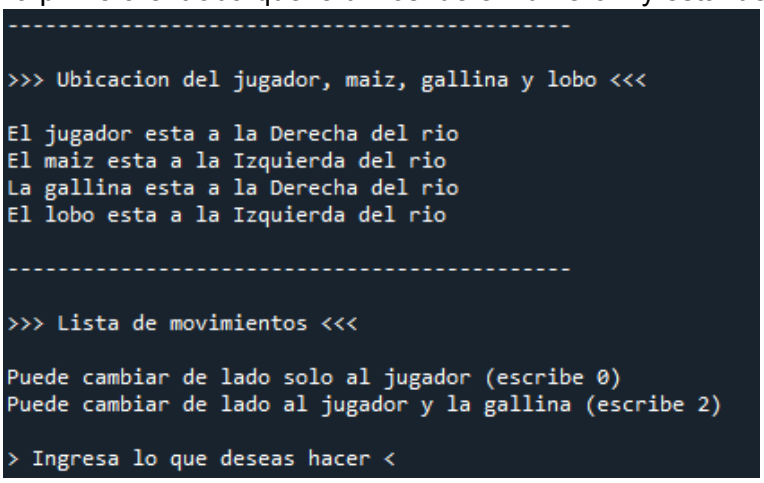
Entrada

Al ejecutar el algoritmo obtenemos esto en pantalla:

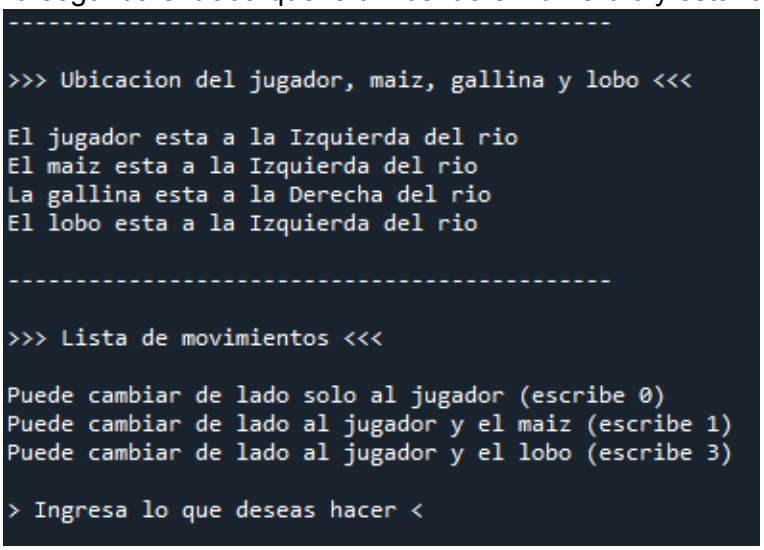
El orden de las instrucciones que hicimos fue el siguiente:
[2,0,3,2,1,0,2]

Salida

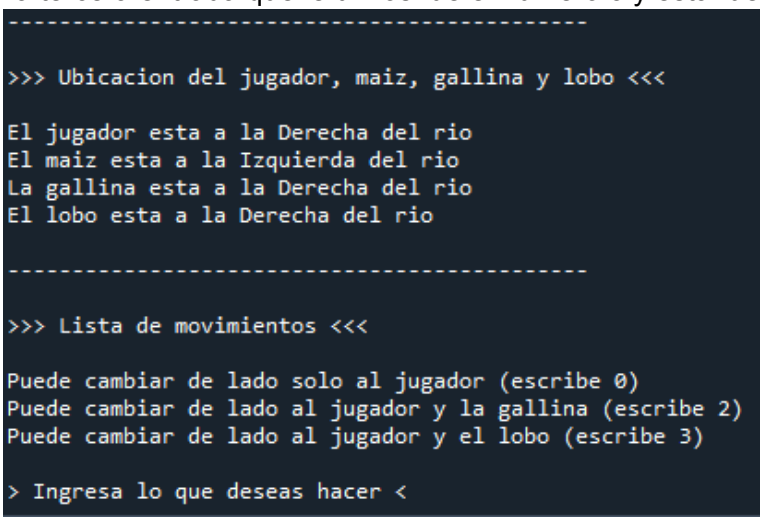
La primera entrada que le dimos fue el número 2 y esta fue la salida:



La segunda entrada que le dimos fue el número 0 y esta fue la salida:



La tercera entrada que le dimos fue el número 3 y esta fue la salida:



La cuarta entrada que le dimos fue el número 2:

```
------------------------------------------

>>> Ubicacion del jugador, maiz, gallina y lobo <<<

El jugador esta a la Izquierda del rio
El maiz esta a la Izquierda del rio
La gallina esta a la Izquierda del rio
El lobo esta a la Derecha del rio

------------------------------------------

>>> Lista de movimientos <<<

Puede cambiar de lado solo al jugador (escribe 0)
Puede cambiar de lado al jugador y el maiz (escribe 1)
Puede cambiar de lado al jugador y la gallina (escribe 2)

> Ingresa lo que deseas hacer <
```

La quinta entrada que le dimos fue el número 1:

```
------------------------------------------

>>> Ubicacion del jugador, maiz, gallina y lobo <<<

El jugador esta a la Derecha del rio
El maiz esta a la Derecha del rio
La gallina esta a la Izquierda del rio
El lobo esta a la Derecha del rio

------------------------------------------

>>> Lista de movimientos <<<

Puede cambiar de lado solo al jugador (escribe 0)
Puede cambiar de lado al jugador y el maiz (escribe 1)
Puede cambiar de lado al jugador y el lobo (escribe 3)

> Ingresa lo que deseas hacer <
```

La sexta entrada que le dimos fue el número 0:

```
------------------------------------------

>>> Ubicacion del jugador, maiz, gallina y lobo <<<

El jugador esta a la Izquierda del rio
El maiz esta a la Derecha del rio
La gallina esta a la Izquierda del rio
El lobo esta a la Derecha del rio

------------------------------------------

>>> Lista de movimientos <<<

Puede cambiar de lado solo al jugador (escribe 0)
Puede cambiar de lado al jugador y la gallina (escribe 2)

> Ingresa lo que deseas hacer <
```

La última entrada fue el número 2:

```
-----------------------------------------

>>> Ubicacion del jugador, maiz, gallina y lobo <<<

El jugador esta a la Derecha del rio
El maiz esta a la Derecha del rio
La gallina esta a la Derecha del rio
El lobo esta a la Derecha del rio

Ganaste!!!
Lograste pasar a todos al otro lado con exito
```

Como se puede observar esta combinación nos lleva a la victoria, si alteramos este orden de entradas, perderemos.

Código

Esta función nos muestra en pantalla en que lado del río se encuentran los personajes.

```
def posicion(persona, maiz, gallina, lobo):
    print("\n-----------------------------------------\n")
    print(">>> Ubicacion del jugador, maiz, gallina y lobo <<<\n")
    print("El jugador esta a la " + ubicacion(persona) + " del rio")
    print("El maiz esta a la " + ubicacion(maiz) + " del rio")
    print("La gallina esta a la " + ubicacion(gallina) + " del rio")
    print("El lobo esta a la " + ubicacion(lobo) + " del rio")
```

Esta función nos indica de qué lado se encuentra cada personaje.

```
def ubicacion(entidad):
    if(entidad == True):
        return "Izquierda"
    else:
        return "Derecha"
```

Esta función es la que se encarga de recibir la decisión del usuario y subirá a un personaje o a otro al bote.

```
def sube_al_bote(selector, persona, maiz, gallina, lobo, estado):
    if(selector == 1):
        maiz = cambia_estado(maiz)
        persona = cambia_estado(persona)
        estado = cambia_estado(estado)
    elif(selector == 2):
        gallina = cambia_estado(gallina)
        persona = cambia_estado(persona)
        estado = cambia_estado(estado)
    elif(selector == 3):
        lobo = cambia_estado(lobo)
        persona = cambia_estado(persona)
        estado = cambia_estado(estado)
    elif(selector == 0):
        persona = cambia_estado(persona)
        estado = cambia_estado(estado)
    return persona, maiz, gallina, lobo, estado
```

Esta función es la que se encarga de cambiar a los personajes de lado, ya sea a la izquierda o a la derecha.

```
def cambia_estado(entidad):
    if(entidad == True):
        return False
    else:
        return True
```

Esta es la parte más importante del código pues esta función es la que nos dice que reglas se tienen que cumplir para que un personaje esté vivo o muerto, como tal compara de que lado se encuentran los personajes y determina si alguno de ellos muere o todos viven.

```
def vivo_o_muerto(persona, maiz, gallina, lobo, estado):
    vivo = True
    if((maiz == gallina) and (estado != (maiz and gallina))):
        vivo = False
        print("\n-----------\n")
        print("Perdiste!!!")
        print("La gallina se comio el maiz")
    elif((lobo == gallina) and (estado != (lobo and gallina))):
        vivo = False
        print("\n-----------\n")
        print("Perdiste!!!")
        print("El lobo se comio a la gallina")
    return vivo
```

Esta función es la encargada de mostrarnos los posibles movimientos que podemos hacer cuando estamos de un lado u otro, claro está que esto solo funciona mientras no hayamos perdido.

```
def movimiento(persona, maiz, gallina, lobo, estado):
    a = []
    a.append(0)
    print("\n-----------------------------------------------\n")
    print(">>> Lista de movimientos <<<\n")
    print("Puede cambiar de lado solo al jugador (escribe 0)")
    if(estado == maiz):
        print("Puede cambiar de lado al jugador y el maiz (escribe 1)")
        a.append(1)
    if(estado == gallina):
        print("Puede cambiar de lado al jugador y la gallina (escribe 2)")
        a.append(2)
    if(estado == lobo):
        print("Puede cambiar de lado al jugador y el lobo (escribe 3)")
        a.append(3)
    return a
```

Esta función solo sirve para imprimir las reglas del juego.

```
def reglas():
    print("\n-----------------------------------------------\n")
    print(">>> Objetivo <<<\n")
    print("El objetivo principal de este juego es conseguir cruzar a todos los personajes al otro lado del rio.")
    print("Inicialmente comenzamos en el lado izquierdo de un rio virtual siendo el jugador, un maiz, una gallina y un lobo,")
    print("El juego consiste en pasar de un lado a otro mediante un barco, pero en este unicamente caben 2 personajes, el chiste
    print("la combinacion correcta de movimientos para que todos los personajes se mantengan con vida, en el momento que uno muer
          + "acaba el juego")
    print("\n>>> Observaciones <<<\n")
    print("1.- En el barco unicamente caben 2 personajes, uno es el jugador por default y el otro cualquier personaje")
    print("2.- El jugador es el unico que puede moverse en el barco solo")
    print("3.- En todo momento se mencionara al jugador las posiciones de todos los personajes y los movimientos que puede realiz
          + "estar de un lado o del otro lado del rio")
    print("Divierte!!")
```

Este es el corazón del juego pues es la función que llama al algoritmo principal.

```python
def juegoGallina():
    persona = True
    lobo = True
    maiz = True
    gallina = True
    vivo = True
    estado = True
    posibles = []
    pierde = True
    print("Juego de la Gallina, el maíz y el lobo")
    reglas()
    while(persona|maiz|gallina|lobo):
        posicion(persona, maiz, gallina, lobo)
        posibles = movimiento(persona, maiz, gallina, lobo, estado)
        vivo = vivo_o_muerto(persona, maiz, gallina, lobo, estado)
        if(vivo == True):
            selector = int(input("> Ingresa lo que deseas hacer < "))
            try:
                indice = posibles.index(int(selector))
                persona,maiz,gallina,lobo,estado = sube_al_bote(selector, persona, maiz, gallina, lobo, estado)
            except:
                print("No puede realizar esta acción")
        else:
            pierde = False
            break
    if(pierde == True):
        posicion(persona, maiz, gallina, lobo)
        print("\nGanaste!!!")
        print("Lograste pasar a todos al otro lado con exito")
```

Pseudocódigo

```
juegoGallina():{
        persona = True
        lobo = True
        maiz = True
        gallina = True
        vivo = True
        estado = True
        posibles = []
        pierde = True
        print(“El juego de la Gallina, el maíz y el lobo”)
        reglas()
        while(persona|maiz|gallina|lobo){
                posicion(persona,maiz,gallina,lobo)
                posibles = movimiento(persona, maiz, gallina, lobo, estado)
                vivo = vivo_o_muerto(persona, maiz, gallina, lobo, estado)
                if(vivo == True){
                        selector = int(input("> Ingresa lo que deseas hacer < "))
                        try{
                                indice = posibles.index(int(selector))
                                persona,maiz,gallina,lobo,estado = sube_al_bote(selector,
                                persona, maiz, gallina, lobo, estado)
                        }
                        except{
                                print("No puede realizar esta acción")
                        }
                }
                else{
                        pierde = False
                        break
                }
        }
```

```
        if(pierde == True){
                posicion(persona, maiz, gallina, lobo)
                print("\nGanaste!!!")
                print("Lograste pasar a todos al otro lado con éxito")

        }
}
```

Conclusiones

Para concluir con este algoritmo, creemos que hacer sistemas expertos, puede automatizar muchas tareas diarias no solo hacer juegos, como este.

Nuestro diseño se basa de un video de YouTube donde se hace un sistema experto que puede “diagnosticar” a un paciente por medio de un cuestionario que realiza el usuario, entonces creemos que la utilidad principal de los sistemas expertos es automatizar ciertas actividades que pueden ser cansadas de hacer constantemente por una persona, y que mejor lo haga un programa para optimizar tiempos de espera, simplemente nos vuela la cabeza, ya que cuando estábamos empezando hacer el algoritmo del juego, no veíamos claro como para que otras cosas podria ser util, después de mirar este video nos quedó más claro sus usos y se nos abrió un mundo infinito de posibilidades.

Análisis Algoritmo “Rutas”

Objetivo

Este algoritmo tiene como objetivo armar un grafo con los datos de un archivo con extensión csv, que nos mostrará cómo se comportan nuestras “rutas”, de qué nodo a que nodo va una ruta y las múltiples formas de llegar de un punto a otro. Un segundo archivo csv nos mostrará más datos sobre estas rutas como el precio, la aerolínea, duración del vuelo o estancia. Finalmente como si se tratara de un catálogo de vuelos, de forma teórica, nos dirá que ruta es mejor tomar para llegar del origen al destino, tomando en cuenta el menor precio para llegar a ese destino.

Restricciones

El algoritmo tiene que recibir la ruta del grafo que creamos con la ayuda de otros algoritmos. Esta entrada debe incluir la ruta del grafo, el nodo del que partimos, el nodo objetivo, y el peso, que en este caso es el que menor precio cueste.

NOTA: Esta ruta tiene que tener una estructura específica para poder generar una lista con la ruta más corta de un punto a otro utilizando el algoritmo de dijkstra y la estructura que este solicita para su ejecución.

Modelo

Entradas:

Una lista que se forma en base al algoritmo “dijkstra” donde se incluye la ruta de un grafo, el nodo inicial, el nodo objetivo, y el peso que queremos tomar en cuenta.

Proceso:

1. Crear el grafo con los datos del archivo csv.
2. Crear la entrada para el algoritmo principal.

NOTA: Este algoritmo puede modificarse para mostrar la información que sea necesaria o que se requiera obtener, modificando los archivos csv.

Descripción del Algoritmo

El algoritmo se divide en 2.

1. La primera parte de este, es la creación de un grafo en base a información contenida en archivos csv.
    a. En uno de ellos se debe contener la información sobre los nodos, para este caso incluimos el nombre del país, la ciudad del que parte el vuelo y una letra identificadora. *Utilizando pandas leeremos el archivo para que se cree una tabla y para posteriormente asignarle a una columna el valor de índice y poder acceder a su información.*
    b. En el archivo csv que utilizaremos para crear el grafo, incluimos el origen, el destino y el precio de un boleto de avión. *También leemos el csv con pandas, pero esta vez no asignamos un índice ya que con este archivo crearemos el grafo utilizando networkx y por defecto ya toma un índice*
    c. En un tercer archivo csv, incluimos características sobre el vuelo y el viaje, como la duración del vuelo, la duración de la estancia, la aerolínea en la que se volara, y nuevamente el precio, esto con el fin de poder relacionar esta información posteriormente en la segunda parte del algoritmo. *Después de leer el csv con pandas, asignamos el índice a la columna del precio, con el fin de poder relacionar el csv del grafo.*
2. La siguiente parte del algoritmo se encarga de darle formato a la salida que queremos, que es la ruta más corta de un punto a otro, en esta parte organizamos y relacionamos la información de los archivos csv. Una vez que diseñamos nuestra salida, el algoritmo al trabajar arrojará los datos de vuelo de nodo en nodo, dependiendo de la cantidad de nodos que se deben recorrer para poder llegar al

destino. Si se pasa por 3 nodos, arrojará información de 2 vuelos, si pasa por 4 nodos arrojará información de 3 vuelos.

Entrada

1. El archivo que se usó para la creación del grafo fue el siguiente:

Lista_vuelos.csv

```
origen,destino,precio
A,B,8313
A,D,16046
D,E,2591
D,L,2504
D,F,13996
I,A,5668
J,A,28261
K,J,5028
A,G,32578
A,I,4160
B,A,11523
A,N,23213
D,M,12641
B,C,6342
M,A,20711
M,E,4527
M,F,14661
B,I,22421
I,J,12912
N,B,46377
B,H,20063
J,I,11890
J,K,6505
J,D,33529
```

2. Los archivos que se usaron para obtener la información para la salida del algoritmo de ruta fueron los siguientes.

Lista_paises.csv

```
indice,pais,ciudad
A,México,Ciudad de México
B,Estados Unidos,Nueva York
C,Canada,Toronto
D,España,Madrid
E,Francia,Paris
F,Emiratos Arabes Unidos,Dubai
G,Rusia,Moscú
H,Brazil,San Pablo
I,Colombia,Medellín
J,Argentina,Buenos Aires
K,Chile,Santiago
L,Italia,Roma
M,Reino Unido,Londres
N,Japón,Tokio
```

Lista_caracteristicas.csv

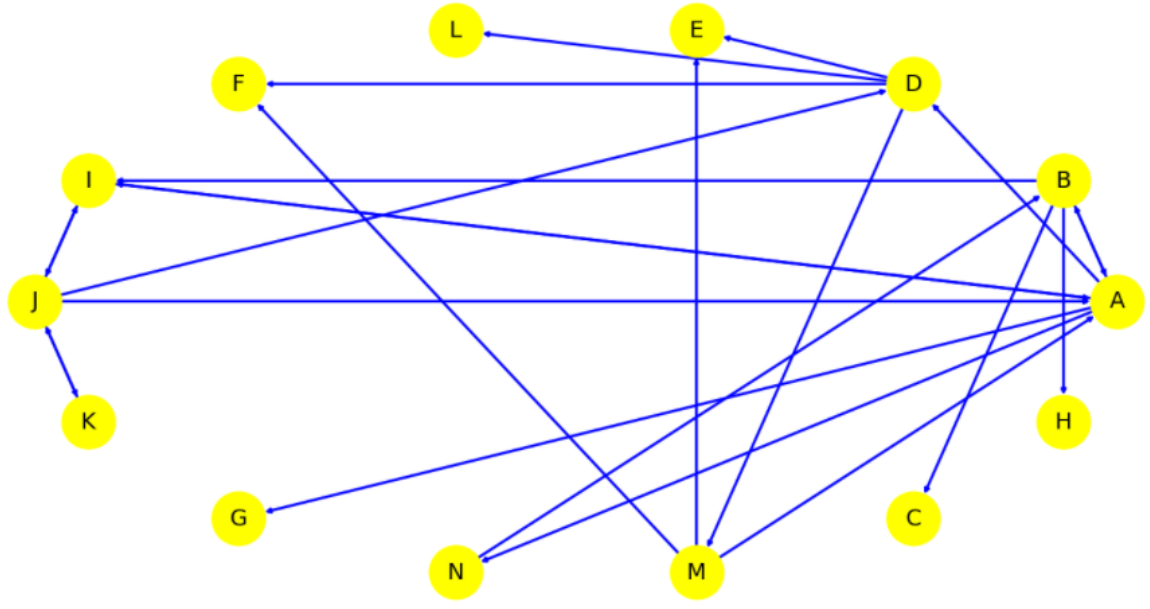
```
precio,duracion vuelo,aerolinea,duracion estancia
8313,5h 00m,Aeroméxico,4 dias
16046,10h 25m,Aeroméxico,10 dias
2591,2h 55m,Air Europa,10 dias
2504,2h 30m,Air Europa,3 dias
13996,7h 00m,Emirates,7 dias
5668,4h 30m,Aeroméxico,8 dias
28261,6h 35m,Aeroméxico,15 dias
5028,2h 00m,Aerolíneas Argentinas,24 dias
32578,10h 10m,KLM,14 dias
4160,4h 30m,Viva Air,7 dias
11523,5h 35m,Aeroméxico,13 dias
23213,14h 55m,Aeroméxico,17 dias
12641,2h 25m,Air Europa,4 dias
6342,1h 37m,Air Canada,8 dias
20711,12h 00m,British Airways,4 dias
4527,1h 20m,Air France,9 dias
14661,6h 50m,Emirates,7 dias
22421,3h 18m,American Airlines,18 dias
12912,1h 09m,Copa Airlines,7 dias
46377,12h 20m,United Airlines,N/A dias
20063,3h 22m,American Airlines,26 dias
11890,6h 20m,Avianca Airlines,9 dias
6505,2h 20m,Aerolineas Argentinas,12 dias
33529,12h 00m,Air Europa,16 dias
```

3. La entrada que se le dio al algoritmo de ruta fue la siguiente:

```
camino = list(nx.dijkstra_path(DG, source = 'B', target = 'E', weight = 'peso'))
ruta(camino)
```

Salida

1. La salida al crear el grafo fue la siguiente:



2. La salida que arroja el algoritmo de ruta fue la siguiente, y se puede comprobar con la información del grafo.

```
Estados Unidos -> México
Ciudad de origen: Nueva York
Ciudad de llegada: Ciudad de México
Aerolinea: Aeroméxico
Duracion del vuelo: 5h 35m
Estancia en la ciudad: 13 dias
Precio individual: 11523 MXN
-----------------------------------------------
México -> España
Ciudad de origen: Ciudad de México
Ciudad de llegada: Madrid
Aerolinea: Aeroméxico
Duracion del vuelo: 10h 25m
Estancia en la ciudad: 10 dias
Precio individual: 16046 MXN
-----------------------------------------------
España -> Francia
Ciudad de origen: Madrid
Ciudad de llegada: Paris
Aerolinea: Air Europa
Duracion del vuelo: 2h 55m
Estancia en la ciudad: 10 dias
Precio individual: 2591 MXN
-----------------------------------------------

Costo total del viaje: 30160 MXN
```

Código

1. Para crear el grafo.

Librerías importadas.
Pandas: Es una librería de Python especializada en el manejo y análisis de estructuras de datos.
networkx: Es un paquete de Python para la creación, manipulación y estudio de la estructura, dinámica y funciones de redes complejas.

```
import pandas as pd
import networkx as nx
from IPython.display import HTML
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (20.0, 10.0)
```

Lectura del archivo csv para crear el grafo.

```
rutas = pd.read_csv("Lista_vuelos.csv")
rutas.head()
```

Creación del grafo.
Se diseña el formato del grafo, se asignan cuáles serán los nodos de origen y de destino en el grafo y asignamos el peso que se usará para determinar cuál es la ruta más barata.

```
DG = nx.DiGraph()
for filas in rutas.iterrows():
    DG.add_edge(filas[1]['origen'], filas[1]['destino'], peso = filas[1]['precio'])

DG.nodes(data = True)
nx.draw_circular(DG,
                 node_color = 'yellow',
                 edge_color = 'blue',
                 font_size = 25,
                 width = 3, with_labels = True, node_size = 3500)
```

2. Algoritmo de ruta.

Lectura del archivo csv con la información de los países y asignación del índice identificador.

```
ciudades = pd.read_csv("Lista_de_paises.csv")
ciudades.set_index(['indice'], inplace=True)
ciudades.head()
```

Lectura del archivo csv con la información sobre las características de los vuelos.

```
caracteristicas = pd.read_csv("Lista_caracteristicas.csv")
caracteristicas.set_index(['precio'], inplace=True)
caracteristicas.head()
```

Algoritmo ruta

```
def ruta(path):
    total_precio = 0
    for i in range(len(path)-1):
        origen = path[i]
        destino = path[i+1]
        precio = DG[origen][destino]['peso']
        total_precio = total_precio + precio
        print(" %s -> %s \n Ciudad de origen: %s \n Ciudad de llegada: %s \n Aerolinea: %s \n Duracion del vuelo: %s \n Estancia
             ciudades.loc[origen]['pais'],
             ciudades.loc[destino]['pais'],
             ciudades.loc[origen]['ciudad'],
             ciudades.loc[destino]['ciudad'],
             caracteristicas.loc[precio]['aerolinea'],
             caracteristicas.loc[precio]['duracion vuelo'],
             caracteristicas.loc[precio]['duracion estancia'],
             precio)
             )
        print("-----------------------------------------------")
    print("\n Total de precio: %s MXN" %(total_precio))

camino = list(nx.dijkstra_path(DG, source = 'B', target = 'E', weight = 'peso'))
ruta(camino)
```

Pseudocódigo

```
ruta(path){
        total_precio = 0
        for(i=0; i<path-1; i++){
                origen = path[i]
                destino = path[i+1]
                precio = DG[origen][destino]['precio']
                total_precio = total_precio + precio
                print("Aquí se acomoda toda la información, se le dará el formato como
                queremos que se nos muestre a la salida")
        }
        print("Precio total")
}
```

Conclusiones

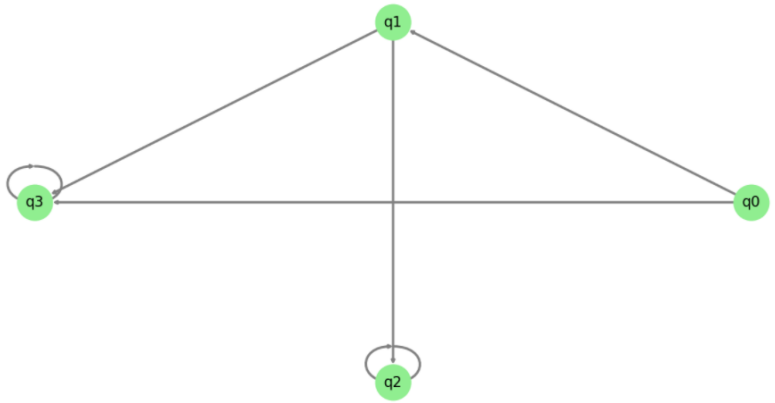
Lo que podemos concluir sobre este algoritmo es que podemos darle muchas aplicaciones ya que de forma específica para las rutas podemos agregar la información que queramos y dar una salida creativa para que se vea como una página de internet que analiza vuelos o viajes entre distintas aerolíneas y precios como lo puede ser despegar.com, que fue de donde tomamos inspiración para obtener la información sobre los viajes y las características de los vuelos.

En cuanto a la creación de los grafos podemos determinar que su uso no es tan claro, pero de manera personal como equipo nos resultó útil para representar sistemas autómatas, un ejemplo de cómo lo hicimos fue el siguiente:

La información de nuestro autómata fue la siguiente:

```
origen,destino
q0,q1
q1,q3
q2,q2
q3,q3
q0,q3
q1,q2
q2,q2
q3,q3
```

y el grafo del autómata que nos arrojó fue el siguiente:



Esta forma de hacer grafos nos ayudó a que pudiéramos representar nuestros autómatas para tareas de la materia de Lenguajes Formales y Autómatas.

Así que de manera personal como equipo nos resultó demasiado útil esta parte del algoritmo, aunque siempre hay áreas de mejora.

Referencias

- Alberca, A. S. (2021, 14 mayo). La librería Pandas. Aprende con Alf. Recuperado 16 de noviembre de 2021, de https://aprendeconalf.es/docencia/python/manual/pandas/
- NetworkX — NetworkX documentation. (s. f.). NetworkX. Recuperado 16 de noviembre de 2021, de https://networkx.org/
- Corredor Velandia, J. C. [Juan Camilo Corredor Velandia]. (2020, 23 octubre). Sistemas Expertos - Sistema Experto Médico Python (Julián Camilo Corredor) [Vídeo]. YouTube. https://www.youtube.com/watch?v=Py5_Ks7HDXg&t=122s