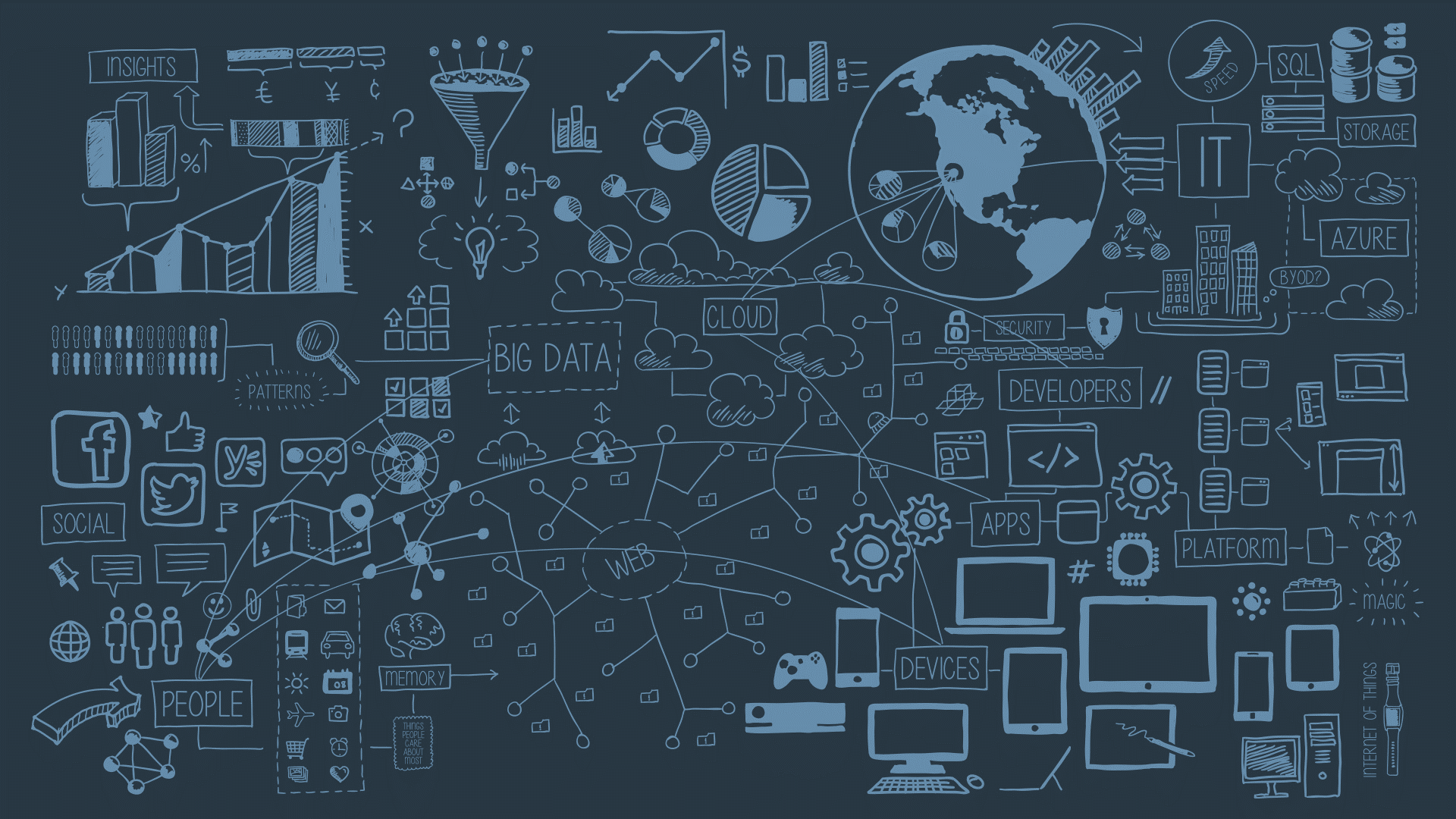
Imagen que contiene Calendario

Descripción generada automáticamente

**21 DE ABRIL DE 2022**

**INTEGRANTES: ROMERO RIBERA GEOVANNI**

**GUZMÁN RAMÍREZ ALDO YAEL**

**PROFESOR:** DR. JOSE ABEL HERRERA CAMACHO

**PROYECTO 1: MÉTODO UNIFORME**

**INTELIGENCIA ARTIFICIAL**

**FACULTAD DE INGENIERÍA**

**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

Método uniforme

Es un algoritmo de búsqueda no informada utilizado para recorrer sobre grafos el camino de costo mínimo entre un nodo raíz y un nodo destino. La búsqueda comienza por el nodo raíz y continúa visitando el siguiente nodo que tiene menor costo total desde la raíz.

Los nodos son visitados de esta manera hasta que el nodo destino es alcanzado.

Puntos importantes

* Se caracteriza por buscar el camino con el costo más pequeño.
* Debido a su forma de trabajo, no se preocupa por el numero de pasos, si no por el costo final.

Algoritmo

El siguiente grafo obtendremos la ruta usando el método uniforme cuyo punto de partida es el nodo A y el destino el nodo F.

3

3

3

1

1

1

2

10

Primeramente, iniciamos colocando el nodo padre, el cual es el nodo de inicio.

Enseguida desplegamos sus nodos hijos. Vemos que una de las rutas ya nos lleva al destino, sin embargo, es muy pronto para decir que esta es la ruta de menor costo por lo que:

1

10

Nos fijamos en la arista con menor peso y desplegamos los hijos de ese nodo, es decir, los hijos del nodo B (D y C).

1

1

10

3

**1**

Realizamos la suma de las trayectorias para comprobar nuevamente cual es la que tiene menor peso.

2=1+ 1

1

10

4=1+3

Y desplegamos las hijas de ese nodo, en este caso es el nodo ‘D’. Volvemos a encontrar ‘F’ (el destino) pero no estamos seguros aun si es la ruta mínima.

1

2

1

10

4

2

**2**

Realizamos la suma nuevamente para buscar el de menor peso.

3=2+1

2

1

10

4

4=2+2

El de menor costo es el nodo E, así que nos movemos hacia el y desplegamos sus nodos hijos (C y F).

**3**

3

3

2

1

10

4

4

3

Realizamos la suma de la trayectoria para observar su costo.

6 = 3 + 3

3

2

1

10

4

4

6 = 3 + 3

Ambas rutas tienen un costo total de 6, y dado que no hay más hacia donde avanzar comparamos las rutas que ya nos llevaban al destino. Concluyendo que de nuestras tres rutas hacia ‘F’ la de menor costo es de 4 que sigue la trayectoria: A – B – D – F.

**4**

6

3

2

1

10

4

4

6

Y así concluye este método.

El recorrido que trazaría en su forma gráfica sería el siguiente:

3

3

3

1

1

1

2

10

Por qué de D no se fue a F si la ruta más corta es 1 y no 2. Esto se debe a que el recorrido que le sigue seria un 3 haciendo que el costo se haga más costoso.

Una vez teniendo el método grafico se realizará el método paso a paso quedaría de la siguiente forma:

|  |  |
| --- | --- |
|  | Nodo extendido |
| Paso 0: A |  |
| Paso 1: | A |
| Paso 2: | B |
| Paso 3: | D |
| Paso 4: | E |

Analizando los dos modelos de análisis podemos concluir lo siguiente:

Recorrido: A – B – D – F

Costo mínimo: 4

Manual de usuario

Para hacer uso de este programa solo tendrá que abrir el archivo Uniforme.py el cual nos desplegará una interfaz en donde se nos mostrará menú donde veremos las ciudades a las que podemos ir, aparte nos solicitará lo siguiente:

1. Colocar la ciudad de inicio.
2. Colocar la ciudad destino.

Una vez ingresado estos dos datos nos desplegara la trayectoria recorrida, trayectoria mínima y el costo mínimo.

Código:

|  |
| --- |
| Método uniforme |
| """  UNIVERSIDAD NACIONAL AUTONAMA DE MÉXICO  FACULTAD DE INGENIERIA  INTELIGENCIA ARTIFICIAL  PROYECTO 1: METODO DE BUSQUEDA UNIFORME  INTEGRANTES: ROMERO RIBERA GEOVANNI  GUZMÁN RAMÍREZ ALDO YAEL  """  # Importando bibliotecas  import pandas as pd  # VARIABLES GLOBALES (Accesibles para cualquier funcion)  info\_ciudades = None # Utilizado en DataFrame para vertices entre ciudades: ciudad1 - ciudad2 - costo  lista\_prioridad\_frontera = [] # Lista FIFO cuyo costo de ruta más bajo sera el Head    # AGREGANDO LAS CIUDADES DE ROMANIA  def agregar\_ciudades():  global info\_ciudades  data = [{'ciudad1': 'Timisoara', 'ciudad2': 'Lugoj', 'costo\_ruta': 111},  {'ciudad1': 'Lugoj', 'ciudad2': 'Mehadia', 'costo\_ruta': 70},  {'ciudad1': 'Mehadia', 'ciudad2': 'Dobreta', 'costo\_ruta': 75},  {'ciudad1': 'Dobreta', 'ciudad2': 'Craiova', 'costo\_ruta': 120},  {'ciudad1': 'Oradea', 'ciudad2': 'Zerind', 'costo\_ruta': 71},  {'ciudad1': 'Oradea', 'ciudad2': 'Sibiu', 'costo\_ruta': 151},  {'ciudad1': 'Zerind', 'ciudad2': 'Arad', 'costo\_ruta': 75},  {'ciudad1': 'Arad', 'ciudad2': 'Sibiu', 'costo\_ruta': 140},  {'ciudad1': 'Arad', 'ciudad2': 'Timisoara', 'costo\_ruta': 118},  {'ciudad1': 'Sibiu', 'ciudad2': 'Fagaras', 'costo\_ruta': 99},  {'ciudad1': 'Sibiu', 'ciudad2': 'Rimnicu Vilcea', 'costo\_ruta': 80},  {'ciudad1': 'Rimnicu Vilcea', 'ciudad2': 'Craiova', 'costo\_ruta': 146},  {'ciudad1': 'Rimnicu Vilcea', 'ciudad2': 'Pitesti', 'costo\_ruta': 97},  {'ciudad1': 'Craiova', 'ciudad2': 'Pitesti', 'costo\_ruta': 138},  {'ciudad1': 'Fagaras', 'ciudad2': 'Bucharest', 'costo\_ruta': 211},  {'ciudad1': 'Pitesti', 'ciudad2': 'Bucharest', 'costo\_ruta': 101},  {'ciudad1': 'Neamt', 'ciudad2': 'Iasi', 'costo\_ruta': 87},  {'ciudad1': 'Iasi', 'ciudad2': 'Vaslui', 'costo\_ruta': 92},  {'ciudad1': 'Bucharest', 'ciudad2': 'Giurgiu', 'costo\_ruta': 90},  {'ciudad1': 'Bucharest', 'ciudad2': 'Urziceni', 'costo\_ruta': 85},  {'ciudad1': 'Urziceni', 'ciudad2': 'Vaslui', 'costo\_ruta': 142},  {'ciudad1': 'Urziceni', 'ciudad2': 'Hirsova', 'costo\_ruta': 98},  {'ciudad1': 'Hirsova', 'ciudad2': 'Eforie', 'costo\_ruta': 86}]  info\_ciudades = pd.DataFrame(data, columns=['ciudad1', 'ciudad2', 'costo\_ruta'])  # print(info\_ciudades) # despliega el dataFrame info\_ciudades  # CLASE NODO Y SUS ATRIBUTOS  class Nodo:  def \_\_init\_\_(self,ciudad,vecino,costo\_ruta):  self.ciudad = ciudad  self.vecino = vecino  self.costo\_ruta = costo\_ruta  # FUNCION MAIN, EJECUTA EL PROGRAMA  def main():  global info\_ciudades # Variable global a modificar  agregar\_ciudades() # Agregamos los caminos de las ciudades  menu() # Desplegamos el menu()  ciudad\_origen = input('Ingrese Ciudad de Inicio: \n')  ciudad\_destino = input('Ingrese Ciudad Destino:\n')  hacer\_recorrido(ciudad\_origen,ciudad\_destino)  def menu():  print('\*--------------------------------------------\*')  print('| METODO UNIFORME |')  print('\*--------------------------------------------\*')  print('| MAPA DE CIUDADES DISPONIBLES PARA VIAJE |')  print('| |')  print('| -Arad -Bucharest -Craiova |')  print('| -Dobreta -Eforie -Fagaras |')  print('| -Giurgiu -Hirsova -Iasi |')  print('| -Lugoj -Mehadia -Neamt |')  print('| -Oradea -Pitesti -Sibiu |')  print('| -Rimnicu Vilcea -Timisoara -Urziceni |')  print('| -Vaslui -Zerind |')  print('\*--------------------------------------------\*')  def hacer\_recorrido(ciudad\_origen, ciudad\_destino):  camino, costo\_total = busqueda\_uniforme(ciudad\_origen, ciudad\_destino)  if not camino: # Si no se pudo llegar al destino  print('ERROR: No se pudo recorrer de ciudad {} a la ciudad {}.\nVerifique sus entradas o las ciudades incluidas en el mapa\n' +  ' y vuelva a ejecutar el programa.'.format(ciudad\_origen, ciudad\_destino))  else:  print('\n\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*')  print('Se ha completado el recorrido de {} a {}, La ruta es: '.format(ciudad\_origen,ciudad\_destino))  ciudades\_visitadas = []  while True:  ciudades\_visitadas.append(camino.ciudad) # Agregando a la lista las ciudades visitadas  if camino.vecino is None: # Cuando camino.vecino no tenga mas valores  break  camino = camino.vecino  size = len(ciudades\_visitadas) # Obtenemos la longitud antes de iterar  for i in range(size): # IMPRIMIENDO LA TRAYECTORIA DE CIUDADES  if i < size - 1:  print(ciudades\_visitadas.pop(), end='' + ", ")  else:  print(ciudades\_visitadas.pop())  print('Costo total: {} [km]'.format(costo\_total))  print('\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*')  # Funcion que comprueba si la ciudad del nodo esta en la lista de prioridad\_frontera  def nodo\_en\_frontera(frontera, nodo):  for i in frontera:  if nodo.ciudad == i.ciudad:  return True  return False # Si no lo encuentra, retorna falso    # Funcion para la busqueda uniforme  def busqueda\_uniforme(c\_origen, c\_destino):  #print(type(c\_origen))  global info\_ciudades, lista\_prioridad\_frontera # Variables que afectan a las globales  nodo = Nodo(c\_origen, None,0) # Creamos un nodo con la ciudad de origen  agregar\_frontera(nodo) # Lo agregamos a la frontera  explorado = []  j = 0 # Para guardar las ciudades ya exploradas    while True:  if len(lista\_prioridad\_frontera) == 0: # Si ya no quedan elementos en la lista de prioridades  return False # Cortamos el flujo del programa  nodo = lista\_prioridad\_frontera.pop(0) # Quitamos el primer elemento  explorado.append(nodo.ciudad)  j += 1  print("Paso {}:" .format(j))  print(explorado) # Prueba de objetivo  if nodo.ciudad == c\_destino: # Si ya completamos el recorrido  return nodo, nodo.costo\_ruta # Terminamos  # Sino, recorremos los nodos secundarios  for i in range(len(info\_ciudades)): # Total de Ciudades  ciudad\_destino = '' #Inicializamos variable ciudad\_destino  if info\_ciudades['ciudad1'][i] == nodo.ciudad: # Si la ciudad de mapa corresponde a la de nodo  ciudad\_destino = info\_ciudades['ciudad2'][i] # Actualizamos ciudad\_destino con el nuevo recorrido  elif info\_ciudades['ciudad2'][i] == nodo.ciudad: # Comparando el nodo de la ciudad con el valor ciudad2 del DaFr  ciudad\_destino = info\_ciudades['ciudad1'][i] # Actualizamos ciudad\_destino con el valor anterior  if ciudad\_destino == '': # Si no hubo un match en el DaFr, continuamos  continue  nodo\_hijo = Nodo(ciudad\_destino, nodo, nodo.costo\_ruta + info\_ciudades['costo\_ruta'][i])    # El siguiente conjunto de lineas nos ayuda a definir los nodos frontera del nodo actual  # Si el nodo no ha sido explorado y no esta mapeado en frontera, se mapea  # Si esta en la frontera, se actualiza con el nodo de menor consumo de ruta  if nodo\_hijo.ciudad not in explorado and not nodo\_en\_frontera(lista\_prioridad\_frontera, nodo\_hijo):  agregar\_frontera(nodo\_hijo) # Agregamos al nodo frontera  elif nodo\_en\_frontera(lista\_prioridad\_frontera, nodo\_hijo):  reemplaza\_frontera(nodo\_hijo) # Corrige para una frontera con menor costo    # Funcion que expande y actualiza las fronteras en base al costo de ruta  def agregar\_frontera(nodo):  global lista\_prioridad\_frontera  size = len(lista\_prioridad\_frontera)  for i in range(size):  if nodo.costo\_ruta < lista\_prioridad\_frontera[i].costo\_ruta: # Si el costo del nodo nuevo es menor al de lista de priodad, se inserta  lista\_prioridad\_frontera.insert(i, nodo)  return  lista\_prioridad\_frontera.append(nodo) # Caso contrario, el costo es mayor y lo agregamos al FINAL de la cola    # Funcion que reemplaza las fronteras en base al costo de ruta  def reemplaza\_frontera(nodo):  global lista\_prioridad\_frontera  size = len(lista\_prioridad\_frontera)  for i in range(size): # Por cada elemento de la lista\_prioridad\_frontera  # Si la ciudad de frontera es la misma que el nodo y el costo de frontera es mayor al del nodo  if lista\_prioridad\_frontera[i].ciudad == nodo.ciudad and lista\_prioridad\_frontera[i].costo\_ruta > nodo.costo\_ruta:  lista\_prioridad\_frontera[i] = nodo # Se actualiza la frontera con el nodo para tener el costo mas bajo  return    #Inicia el programa  main()  # In[ ]: |

Glosario

**Algoritmo de búsqueda no informada:** es un algoritmo cuya estrategia de búsqueda ignora si el siguiente estado es mejor o peor dependiendo su criterio.

**Grafo:** en el campo de la computación, los grafos son la representación de objetos unidos entre si para representar relaciones binarias. Estos objetos se conocen como “vértices” y las uniones se conocen como “aristas”.