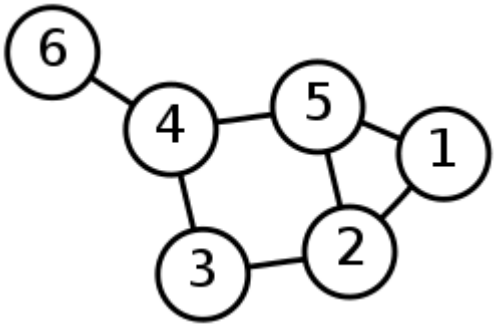


PROYECTO ALGORITMO DE BUSQUEDA: BUSQUEDA EN HAZ

DESCRIPCIÓN BREVE

Documentación del código que implementa el algoritmo de búsqueda en haz.

Autor: Daniel Salazar - Matricula: 17289193

Materia: Inteligencia Artificial – Facultad de sistemas – Universidad Autónoma de Coahuila

¿Cómo funciona la búsqueda en haz?

La búsqueda en haz es uno de los métodos de búsqueda heurística, la búsqueda heurística quiere decir que es una búsqueda ciega, no importa si da el camino más corto o el mejor, lo importante es llegar, mientras menor sea el nodo en teoría deberíamos estar más cerca de la meta.

La búsqueda en haz funciona en árboles, esto quiere decir que si se quiere utilizar búsqueda en haz en un grafo primero se debe realizar su árbol para poder aplicar el algoritmo en él, después el árbol que se tiene se va a separar por niveles, y en cada nivel se va a seleccionar un numero de nodos, el número de nodos que se seleccionan será igual a la mayor cantidad de ramificaciones que tiene un nodo en el grafo, estos nodos seleccionados deben ser los nodos con menos peso de su nivel y su padre debe pertenecer a los nodos seleccionados, en cada uno de los niveles vamos a tachar el nodo menos pesado, esto quiere decir que ya fue revisado, esto se hace por que si se termina de revisar todos los niveles del árbol y el nodo meta no fue encontrado se debe regresar al primer nivel y comenzar la búsqueda de nuevo, pero ahora ya no se puede seleccionar los nodos que ya fueron revisados, esto hace que en cada corrida se vaya marcando como visitado un nodo de cada nivel, así hasta llegar al nodo meta, para poder seleccionar un nodo meta su padre debe pertenecer a los nodos seleccionados y el peso de los nodos es acumulable.

Referencia externa a la clase: https://www.youtube.com/watch?v=KP3mYkp2UIs

Funcionamiento del código:

Primera Línea:

busquedahaz.py > ...

```
1   from operator import itemgetter
2
```

En esta línea se importa una librería que vamos a utilizar más adelante para conseguir los nodos menos pesados de cada nivel.

Función leergrafo:

```
3   def leergrafo(grafo):
4       grafosinseparar = grafo.split(",")
5
6       grafoseparado = []
7
8       for nodoaristaypeso in grafosinseparar:
9           grafoseparado.append(nodoaristaypeso.split(" "))
10
11      grafoinamovibleseparado = tuple(grafoseparado)
12
13      return grafoinamovibleseparado
14
```

El propósito de esta función es recibir un grafo y darle formato para poder usarlo.

En esta función recibimos como parámetro un grafo, el grafo debe estar escrito en el siguiente formato:

1 2 13,1 3 15,1 4 12,2 5 4,2 7 4,3 6 8,3 2 6,4 2 9,4 7 8,5 8 4,5 10 8,6 5 2,6 9 7,6 8 6,7 5 10,7 10 15,8 10 8,9 10 10

Cada nodo esta separado por una coma, el primer carácter es el nodo padre, el segundo el nodo hijo y el tercero es el peso de la conexión entre ellos.

En la línea 4 tenemos una variable que recibe grafo.split(“,”) esto separa la información en nodo por comas, lo que hace que cada nodo padre, nodo hijo y peso se vayan guardando por separado en una lista, dando como resultado que “grafo sin separar” sea una lista de listas.

En la línea 6 se declara “grafoseparado” el cual es una lista vacía, y la utilizaremos porque en la línea 8 comienza un ciclo en el cual cada valor dentro de una lista en la lista de listas se va a guardar por separado, separándolos por un espacio en blanco, así el nodo que introdujimos anteriormente se verá así:

[['1', '2', '13'], ['1', '3', '15'], ['1', '4', '12'], ['2', '5', '4'], ['2', '7', '4'], ['3', '6', '8'], ['3', '2', '6'], ['4', '2', '9'], ['4', '7', '8'], ['5', '8', '4'], ['5', '10', '8'], ['6', '5', '2'], ['6', '9', '7'], ['6', '8', '6'], ['7', '5', '10'], ['7', '10', '15'], ['8', '10', '8'], ['9', '10', '10']]

Siendo que cada lista tendrá por separado los valores de padre, hijo y el peso de la conexión entre ellos, al final de una función esta información se transforma en una tupla de listas, ya que el contenido de una tupla no puede ser modificado y esto la hace más segura, después solo se regresa la tupla de listas.

Función valor_de_k:

```
15  def valor_de_k(grafo):
16      k = 0
17      valores_de_k = []
18      auxiliar2 = grafo[0][0]
19
20      for nodo in grafo:
21          auxiliar = nodo[0]
22
23          if auxiliar == auxiliar2:
24              k += 1
25          else:
26              valores_de_k.append(k)
27              k = 1
28
29          auxiliar2 = auxiliar
30
31      for kauxiliar  in valores_de_k:
32          if kauxiliar >= k:
33              k = kauxiliar
34
35      return k
```

El propósito de esta función es retornar el valor de k, el cual es la mayor cantidad de ramificaciones que tiene un nodo en el grafo.

Se recibe como parámetro un grafo que ya haya sido formateado por la función leergrafo, después en las primeras tres líneas dentro de la función se establecen variables de las que estaremos haciendo uso, la variable k es donde se guardara el valor encontrado, valores_de_k es una lista que sirve para guardar la cantidad de veces que es encontrado un mismo padre en el grafo, y auxiliar2 es el valor del primer padre en el grafo.

En la línea 20 se comienza un ciclo que va a analizar cada nodo en el grafo, utilizaremos un auxiliar que recibe el valor del padre en el nodo, después si ese nodo padre se vuelve a encontrar el valor de k aumenta, y si ya no se encuentra más el valor de ese nodo padre se mete como una de las veces que se encontró k y luego se reestablece k a 1, después el valor del auxiliar2 será igual al nodo padre del nodo anterior que fue revisado y esto se va repitiendo.

Después en la línea 31 comienza un nodo para ver cual fue la mayor cantidad de veces que se encontró un padre, y al final se retorna ese valor para k.

Para ejemplificar y entender mejor la función utilizaremos el nodo que se utilizo como ejemplo en la función anterior, en ella se repite 3 veces el 1 como padre, 2 veces el 2, 2 veces el 3, 2 veces el 4, 2 veces el 5, 3 veces el 6, 2 veces el 7, 1 vez el 8 y 1 vez el 9, esto hace que después de procesar este grafo en la función valor_de_k la lista valores_de_k será igual a: ['3', '2', '2', '2', '2', '3', '2', '2', '1', '1'], y la mayor cantidad de veces que se encontró el mismo padre en un nodo es 3.

Función formararbolporniveles:

```
37  def formararbolporniveles(grafoescrito, inicio):
38      nodospadres = [[inicio, 0]]
39      nivel = []
40      arbolporniveles = {}
41      contadorniveles = 1
42
43      while len(nodospadres) > 0:
44
45          for nodopadre in nodospadres:
46
47              grafo = leergrafo(grafoescrito)
48
49              for nodo in grafo:
50
51                  nodotemporal = nodo
52
53                  if nodotemporal[0] == nodopadre[0]:
54
55                      pesonodo = int(nodotemporal[2])
56
57                      pesopadre = int(nodopadre[1])
58
59                      pesoacumulado = pesonodo + pesopadre
60
61                      nodotemporal[2] = str(pesoacumulado)
62
63                      nivel.append(nodotemporal)
64
65          nodospadres = []
66
67          for nodopadreaañadir in nivel:
68              nodospadres.append([nodopadreaañadir[1],nodopadreaañadir[2]])
69
70          arbolporniveles[f'nivel{contadorniveles}'] = nivel
71          contadorniveles+=1
72          nivel = []
73
74      contadorniveles-=1
75
76      del arbolporniveles[f'nivel{contadorniveles}']
77
78      return arbolporniveles
```

Esta función tiene como objetivo generar el árbol separado por niveles con sus pesos acumulados del grafo que fue ingresado.

Se recibe como parámetro el grafo no formateado con la función “leergrafo” y el nodo inicio.

Para comenzar se declara una lista de listas llamada “nodospadres” que contiene el nodo padre y el peso de ese nodo, se inicializa con el valor del nodo que ingresemos como inicio y con su peso en 0, después se declara “nivel” que servirá para guardar los nodos de cada nivel, después se declara un diccionario llamado “arbolporniveles”, en este se van a ingresar cada nivel, y al final declaramos un int llamado “contadorniveles”, llevara la cuenta de en qué nivel estamos.

En la línea 43 se comienza un ciclo, este ciclo va a seguir hasta que la lista de “nodospadres” este vacía, dentro de el tenemos un ciclo que hace uso de cada ”nodopadre” en la lista de “nodospadres”, después se obtiene el grafo formateado con la función “leergrafo”, después la siguiente indicación es otro ciclo que hace uso de cada nodo en el grafo, dentro de este ciclo se declara un “nodotemporal” que contiene el valor del nodo, para así no editar el contenido del nodo original.

Dentro de estos ciclos en la línea 53 se establece un if, solo se puede acceder a su contenido si el nodo padre del nodo que se esta usando es igual a alguno de los nodos padres actuales en la lista de “nodospadres”, dentro de este if se suma el peso del nodo padre y del nodo hijo para ir acumulando los pesos y se añade a los nodos que serán parte del nivel.

Para entender mejor estas líneas de código tenemos el siguiente ejemplo: si al principio de la función la lista “nodospadres” = [[1,0]]

Al entrar a los ciclos solo serán añadidos al nivel los nodos que tienen como padre el nodo 1, estos nodos según el grafo que hemos usado como ejemplo serían: ['1', '2', '13'], ['1', '3', '15'] y ['1', '4', '12'], entonces el nivel 1 del árbol serían estos nodos.

Al salir del ciclo establecido en la línea 45 se sigue estando dentro del ciclo de la línea 43 que solo se detendrá cuando los nodos padres de los siguientes nodos sean 0 o menores a 0, entonces lo que se hace a continuación en el ciclo es establecer los nuevos valores de padres, primero la lista “nodospadres” se declarara como una lista vacía, para a continuación en el siguiente ciclo en la línea 67 añadir los nodos padres y sus pesos de los nodos que fueron encontrados en el nivel, siguiendo con el ejemplo anterior el contenido de la lista “nodospadres” sería igual a: [['2', '13'], ['3', '15'], ['4', '12']], así para que un nodo sea agregado en la siguiente corrida del ciclo en la línea 45 su nodo padre debe ser igual a 2, 3 o 4.

Para finalizar el ciclo lo que se hace es que se agrega al diccionario “arbolporniveles” una clave como “nivel1” y su valor el cual sería : ['1', '2', '13'], ['1', '3', '15'] y ['1', '4', '12'], después la lista “nivel” se establece como vacía para que en la siguiente corrida se puedan añadir solo los nodos que sean hijos de los nodos padres del nivel que se acaba de analizar, y la variable “contadorniveles” se va aumentando para llevar la contabilización de en que nivel estamos.

Al salir de todos los ciclos en la línea 74 se reduce en uno el valor de “contadorniveles”, esto por que después de todos los ciclos anteriores el diccionario “arbolporniveles” contiene un nivel de más, el cual se genera intentando buscar hijos del nodo final, pero como este nodo final ya no

tiene hijos termina siendo un nivel vacío, así que utilizamos esta variable y la siguiente línea para borrar ese ultimo nivel vacío generado del diccionario “arbolporniveles”.

Al final en la línea 78 solo se retorna el diccionario “arbolporniveles” generado.

Función busquedahaz:

```
80  def busquedahaz(k, inicio, fin, arbol):
81
82      valoresdelarbol = arbol.values()
83      padresdenodosseleccionados = [inicio]
84      nodosseleccionados = []
85      nodosseleccionadosdelnivel = []
86      caminoalreves = []
87      contadornivel = 1
88      nodofinalencontrado = False
89
90      for nivel in valoresdelarbol:
91
92          contador = 0
93
94          if len(nivel) <= k:
95
96              for nodopadre in padresdenodosseleccionados:
97
98                  if nodofinalencontrado == True:
99                      break
100
101                 for nodo in nivel:
102
103                     if nodo[1] == fin and nodo[0] == nodopadre:
104                         meta = nodo
105                         nodofinalencontrado = True
106                         break
107
108                 for nodo in nivel:
109                     if nodo[0] == nodopadre:
110                         nodosseleccionadosdelnivel.append(nodo)
111         else:
112             salio = False
113             while contador < k:
114                 nodoseleccionado = min(enumerate(map(itemgetter(-1), nivel)),key=itemgetter(1))
115                 indice = nodoseleccionado[0]
116
117                 contadorpadre = 0
118
119                 while contadorpadre < len(padresdenodosseleccionados):
120
121                     if contadorpadre == 0:
122
123                         for nodopadre in padresdenodosseleccionados:
124
125                             if nodofinalencontrado == True:
126                                 break
```

```
127
128                         for nodo in nivel:
129                             if nodo[1] == fin and nodo[0] == nodopadre:
130                                 meta = nodo
131                                 nodofinalencontrado = True
132                                 break
133
134                 if nivel[indice][0] == padresdenodosseleccionados[contadorpadre]:
135                     nodosseleccionadosdelnivel.append(nivel.pop(indice))
136                     contador +=1
137                     salio = True
138                     break
139
140                 contadorpadre+=1
141
142             if salio == False:
143                 nivel.pop(indice)
144             salio = False
145
146         padresdenodosseleccionados = []
147
148         for nodopadreaañadir in nodosseleccionadosdelnivel:
149             padresdenodosseleccionados.append(nodopadreaañadir[1])
```

```
150
151     indiceyvalor = min(enumerate(map(itemgetter(-1), nodosseleccionadosdelnivel)),key=itemgetter(1))
152     nodomenordelosseleccionados = nodosseleccionadosdelnivel[indiceyvalor[0]]
153
154     print(f'Los nodos seleccionados del nivel {contadornivel} son: {nodosseleccionadosdelnivel} y el menor de esos nodos es: {nodomenordelosseleccionados}')
155     contadornivel+=1
156
157     nodosseleccionados.append(nodosseleccionadosdelnivel)
158
159     nodosseleccionadosdelnivel = []
160
161     if nodofinalencontrado == True:
162         print('¡SE ENCONTRO LA META!')
163         break
```

```
164
165     print(f'El nodo meta fue: {meta}')
166     costo = meta[2]
167     meta = meta[1]
168     caminoalreves.append(meta)
169
170     for nivel in reversed(nodosseleccionados):
171             for nodo in nivel:
172                 if meta == nodo[1]:
173                     meta = nodo[0]
174                     caminoalreves.append(meta)
175                     break
176
177     camino = []
178
179     for nodo in reversed(caminoalreves):
180         camino.append(nodo)
181
182     print(f'El camino es: {camino}, con un costo de: {costo}')
```

Esta función tiene como objetivo implementar el algoritmo de búsqueda haz sobre el árbol antes generado.

La función recibe como parámetros el valor de k, el nodo inicio, el nodo final y el árbol generado en la función formararbolporniveles.

Lo que se declara en las primeras líneas es: una lista llamada “valoresdelarbol” la cual contiene los nodos separados por niveles del árbol transformada en una lista de listas para manejarla más fácilmente, después tenemos la lista “padresdenodosseleccionados” aquí se van a ir guardando los padres de los nodos que se seleccionen de cada nivel, después otras dos listas, la lista “nodosseleccionados” va a guardar todos los nodos seleccionados del árbol y la lista “nodosseleccionadosdelnivel” solo ira guardando los nodos que se seleccionen en cada nivel, la lista “caminoalreves” contendrá el camino que se encuentre de el nodo inicio al nodo meta al revés, la variable “contadornivel” nos ayudará a saber en que nivel vamos y la variable “nodofinalencontrado” se usará para saber cuando se encuentre el nodo final.

En la línea 90 comenzamos con un ciclo que recorrerá todos los niveles que tenga el árbol, al comenzar tendremos un contador inicializado en 0, el cual próximamente nos ayudara a saber cuantos nodos han sido seleccionados y no sean más del valor establecido en k, después comenzamos con un condicional en la línea 94, y accedemos a su contenido solo si el numero de nodos que tiene el nivel actual es menor o igual al valor de k, esto por que si el numero de nodos

que tiene el nivel es menor o igual a k no se tiene que hacer el proceso de escoger los que pesen menos, solo revisar que su nodo padre sea parte de los nodos padres seleccionados, entonces si accedemos nos encontramos con un ciclo el cual analiza cada nodo padre en la lista de "padresdenodosseleccionados", en este ciclo tenemos un if, en el cual si el nodo meta ha sido encontrado hace que el ciclo se pare, después tenemos un ciclo el cual analiza cada nodo en el nivel para saber si entre ellos esta el nodo meta, en este condicional si el nodo hijo es igual a el nodo final y el nodo padre es parte de los padres de los nodos seleccionados entonces se establece que se ha encontrado el nodo meta y se sale de el ciclo, aquí por ejemplo con el grafo que hemos llevado para que un nodo fuera seleccionado tendría que ser algo como : ['sea alguno de los nodos padres', '10', 'algún peso'], esto por que el nodo final de el grafo que hemos utilizado como ejemplo es 10, después de este proceso se busca que los nodos en el nivel sean hijos de algún nodo que pertenezca a la lista "padresdenodosseleccionados", y si es así se ingresa ese nodo a la lista "nodosseleccionadosdelnivel". Recordemos que este proceso es solo si el número de nodos en el nivel es igual o menor a k.

Si el nivel tiene más nodos que el valor de k accedemos a el contenido de la línea 111, primero nos encontramos con una variable que nos ayuda a saber si un nodo se ha eliminado del nivel, después accedemos a un ciclo, el cual se va a seguir corriendo hasta que el numero de nodos seleccionados sea igual a k, el código en la línea 114 nos dará el índice y peso del nodo con menor peso en el nivel, la variable "índice" en la siguiente línea guarda el índice del nodo con menor peso en el nivel, el siguiente contador "contadorpadre" nos ayuda a indicar que padre en "padresdenodosseleccionados" esta siendo utilizado, después se accede a un ciclo el cual se repite hasta que todos los valores de la lista "padresdenodosseleccionados" sean utilizados, al entrar a este ciclo tenemos un condicional que nos ayuda a que el proceso dentro de este solo se haga una vez dentro del ciclo, y el proceso dentro del condicional es el mismo que se hace de la línea 96 a la línea 106 para saber si el nodo meta esta en los nodos dentro de ese nivel, y en la línea 134 tenemos una condicional para saber si el nodo que fue encontrado anteriormente como el menos pesado en el nivel puede ser seleccionado, para ser seleccionado su padre debe formar parte de los "padresdenodosseleccionados", si puede ser seleccionado se añade a la lista "nodosseleccionadosdelnivel" y se elimina del nivel para que no vuelva a ser seleccionado como el menos pesado, se aumenta el valor de "contador" y se indica como True el valor de "salio", indicando que ya fue encontrado un nodo que cumple las condiciones para ser seleccionado en el nivel, después si el nodo con menor peso encontrado no cumple con las condiciones para ser seleccionado se saca del nivel sin añadirse a ninguna lista, para que la próxima vez que se repita el ciclo no se seleccione el mismo nodo con menor peso.

Después de realizar el proceso de seleccionar los nodos con menor peso del nivel y checar si el nodo meta se encuentra en los nodos del nivel pasamos a la línea 146, en la cual se establece como vacía la lista "padresdenodosseleccionados" para que en el siguiente ciclo en la línea 148 se añadan los padres de los nodos que fueron seleccionados en el nivel, este es un proceso que también se realizaba en la función "formararbolporniveles" luego, en la línea 151 se busca el índice y valor del nodo con menor peso de los nodos que ya fueron seleccionados, y a continuación en la línea 154 se imprimen los nodos seleccionados del nivel y el nodo con menor peso de los nodos seleccionados del nivel.

Luego le sigue en la línea 155 un aumento al contador de nivel, para llevar la cuenta de en que nivel vamos, luego los nodos seleccionados del nivel se añade a la lista “nodosseleccionados” ya que como se había indicado en ella se guardaran los nodos seleccionados de todos los niveles, y después se establece como vacía la lista “nodosseleccionadosdelnivel” para que en la siguiente repetición del ciclo se ingresen los nodos que se seleccionen en ese nivel y no se junten con los nodos seleccionados del nivel anterior, para finalizar el ciclo principal tenemos un condicional en la línea 161, en el cual accedemos si el nodo meta ha sido encontrado, y si es así se imprime que el nodo meta fue encontrado y se sale del ciclo.

Después, ya afuera del ciclo donde se seleccionaron los nodos con menos peso en cada nivel, el nodo con menos peso de los nodos seleccionados y se busco el nodo meta tenemos una impresión que nos dice cual fue el nodo meta, ha esto le sigue una variable que guarda el costo del camino que se recorre para llegar a ese nodo, después la variable meta será igual al hijo del nodo meta y esta se añade a la lista “caminoalreves”, después tenemos un ciclo en la línea 170 que se encarga de buscar el camino del nodo meta al nodo inicial, luego creamos una lista para guardar el camino en orden normal y luego tenemos un ciclo en la línea 179 que recorre al revés la lista “caminoalreves” y va añadiendo sus valores a la lista “camino”, para así tener el camino guardado de forma correcta.

Al final de la función tenemos la línea 182 que nos imprime el camino al nodo meta y el peso del camino.

Líneas de código Finales:

```
187  grafoescrito = input("Introduce tu grafo: ")
188  inicio = input("¿Cual es tu nodo inicial?: ")
189  fin = input("¿cual es el nodo final?: ")
190
191  grafoformateado = leergrafo(grafoescrito)
192
193  k = valor_de_k(grafoformateado)
194
195  arbolporniveles = formararbolporniveles(grafoescrito,inicio)
196
197  print(f'Valor de k: {k}')
198  print('Arbol por niveles con sus pesos acumulados: ')
199
200  for key in arbolporniveles:
201      print(key, ":", arbolporniveles[key])
202
203  print('------------------------------------------------------------------------')
204  print('Comienzo de la busqueda del nodo meta por el algoritmo busqueda haz')
205  print('------------------------------------------------------------------------')
206
207  busquedahaz(k, inicio, fin, arbolporniveles)
```

De las líneas 187 a 189 se leen los datos necesarios para realizar los procesos, en la línea 191 se hace uso de la función “leergrafo” para tener el grafo formateado, en la línea 193 se hace uso de la función “valor_de_k” para obtener el valor de k y en la línea 195 se hace uso de la función “formararbolporniveles” para obtener el árbol por niveles, de las líneas 197 a 201 se imprimen los valores de k y el árbol que se genera, de las líneas 203 a la 205 se imprimen unos separadores y se indica que ya comienza el proceso del algoritmo búsqueda haz, y en la línea 207 se hace uso de la función “busquedahaz”.

Pruebas del algoritmo:

Prueba 1:

Grafo ingresado: 1 2 13,1 3 15,1 4 12,2 5 4,2 7 4,3 6 8,3 2 6,4 2 9,4 7 8,5 8 4,5 10 8,6 5 2,6 9 7,6 8 6,7 5 10,7 10 15,8 10 8,9 10 10

Inicio: 1

Final: 10

Resultado:

Prueba 2:

Grafo ingresado: A B 308,B C 221,B D 510,C E 513,D E 185,D H 657,E F 657,H K 36,F H 45,K M 169,F G 234,G I 97,I J 13,J L 62,L M 30

Inicio: A

Final: M

Resultado:

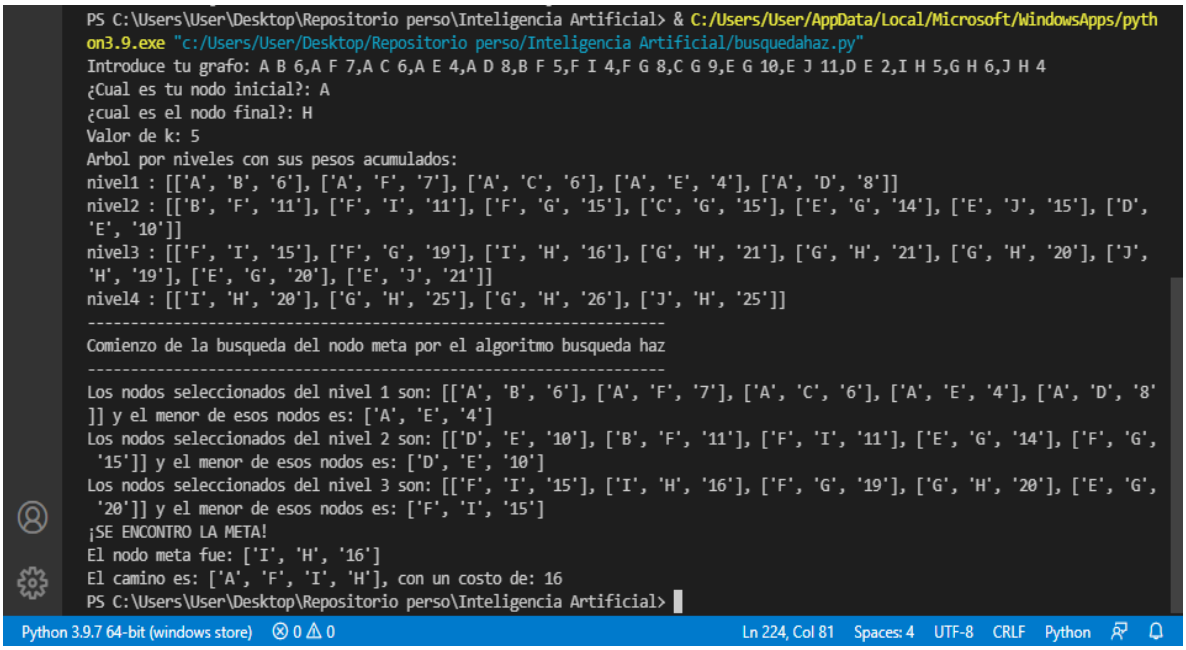
```
PS C:\Users\User\Desktop\Repositorio perso\Inteligencia Artificial> & C:/Users/User/AppData/Local/Microsoft/WindowsApps/python3.9.exe "c:/Users/User/Desktop/Repositorio perso/Inteligencia Artificial/busquedahaz.py"
Introduce tu grafo: A B 308,B C 221,B D 510,C E 513,D E 185,D H 657,E F 657,H K 36,F H 45,K M 169,F G 234,G I 97,I J 13,J L 62,L M 30
¿Cual es tu nodo inicial?: A
¿cual es el nodo final?: M
Valor de k: 2
Arbol por niveles con sus pesos acumulados:
nivel1 : [['A', 'B', '308']]
nivel2 : [['B', 'C', '529'], ['B', 'D', '818']]
nivel3 : [['C', 'E', '1042'], ['D', 'E', '1003'], ['D', 'H', '1475']]
nivel4 : [['E', 'F', '1699'], ['E', 'F', '1660'], ['H', 'K', '1511']]
nivel5 : [['F', 'H', '1744'], ['F', 'G', '1933'], ['F', 'H', '1705'], ['F', 'G', '1894'], ['K', 'M', '1680']]
nivel6 : [['H', 'K', '1780'], ['G', 'I', '2030'], ['H', 'K', '1741'], ['G', 'I', '1991']]
nivel7 : [['K', 'M', '1949'], ['I', 'J', '2043'], ['K', 'M', '1910'], ['I', 'J', '2004']]
nivel8 : [['J', 'L', '2105'], ['J', 'L', '2066']]
nivel9 : [['L', 'M', '2135'], ['L', 'M', '2096']]
---------------------------------------------------------------
Comienzo de la busqueda del nodo meta por el algoritmo busqueda haz
---------------------------------------------------------------
Los nodos seleccionados del nivel 1 son: [['A', 'B', '308']] y el menor de esos nodos es: ['A', 'B', '308']
Los nodos seleccionados del nivel 2 son: [['B', 'C', '529'], ['B', 'D', '818']] y el menor de esos nodos es: ['B', 'C', '529']
Los nodos seleccionados del nivel 3 son: [['D', 'E', '1003'], ['C', 'E', '1042']] y el menor de esos nodos es: ['D', 'E', '1003']
Los nodos seleccionados del nivel 4 son: [['E', 'F', '1660'], ['E', 'F', '1699']] y el menor de esos nodos es: ['E', 'F', '1660']
Los nodos seleccionados del nivel 5 son: [['F', 'H', '1705'], ['F', 'H', '1744']] y el menor de esos nodos es: ['F', 'H', '1705']
Los nodos seleccionados del nivel 6 son: [['H', 'K', '1741'], ['H', 'K', '1780']] y el menor de esos nodos es: ['H', 'K', '1741']
Los nodos seleccionados del nivel 7 son: [['K', 'M', '1910'], ['K', 'M', '1949']] y el menor de esos nodos es: ['K', 'M', '1910']
¡SE ENCONTRO LA META!
El nodo meta fue: ['K', 'M', '1949']
El camino es: ['A', 'B', 'D', 'E', 'F', 'H', 'K', 'M'], con un costo de: 1949
PS C:\Users\User\Desktop\Repositorio perso\Inteligencia Artificial>
Python 3.9.7 64-bit (windows store)  ⊗ 0 ⚠ 0   Ln 224, Col 81  Spaces: 4  UTF-8  CRLF  Python
```

Prueba 3:

Grafo ingresado: A B 6,A F 7,A C 6,A E 4,A D 8,B F 5,F I 4,F G 8,C G 9,E G 10,E J 11,D E 2,I H 5,G H 6,J H 4

Inicio: A

Final: H



FALTANTES EN EL CÓDIGO:

1. Si en el recorrido del árbol no se llega al nodo meta en la función “búsqueda haz” no se encontrará la meta de ninguna forma, esto porque no tiene una manera en la cual recorrer las veces que sean necesarias el árbol hasta que encuentre el nodo meta, creo que si se añadiera un while el cual deje de recorrerse hasta que se encuentre el nodo meta funcionaria, pero haciendo pruebas no logre encontrar la forma en que esto funcionara, así que lo deje como está, que es la mayor funcionalidad posible a la que llegue.
2. Es posible que en algún grafo el nodo meta sea uno que no es hijo de nodos padres que hayan sido seleccionados, esto por que si digamos el numero 5 aparece como nodo padre varias veces en el mismo nivel pero una de esas veces si es seleccionado y otras no el código solo toma como criterio para seleccionar el nodo meta que su nodo padre sea 5, pero no puede saber si fue seleccionado o no, para saber esto se habría que añadir otro identificador único a cada uno de los nodos, así digamos si tenemos dos nodos cinco siendo: [5, 10, 15, identificador1] y [5, 10, 13, identificador2] y solo el segundo es de los nodos seleccionados el nodo meta tomaría como criterio que el padre sea 5, y que con su identificador si sea un nodo seleccionado, entonces solo podría tomar uno de estos nodos como nodo meta, ya que sin este identificador puede tomar cualquiera de los dos, pues los dos tienen como padre 5. Pero este nuevo identificador adicional tampoco logre implementarlo.

PD: El código está comentado por completo, las capturas en este documento fueron tomadas antes de comentar el código.