Taller de Herramientas Computacionales

Brenda Paola García Rivas

24.01.19

Jueves 24 de Enero. Clase 14

Hoy fue una clase muy interesante, se inició con el problema de fibonacci, el cuál es un problema de recursividad; ayer se vió como resolverlo, sin embargo, esta vez el profesor nos explicó una forma más fácil de hacer los llamados para cada número. Este método implica guardar los resultados en una lista, en este se modificará que: el programa observe que esté un número en la lista para después guardarlo y posteriormente usarlo, pero ¿cómo se sabe que el décimo, onceavo, o el número que sea está en la lista?, pues fácil, teniendo en cuenta la longitud de la lista.

Además de esto, también se resolvieron algunas dudas de la tarea, se inició con el problema de la creación de un laberinto, para realizar éste, debemos tener en mente varias cosas:

1. Primero, se identificó en el pizarrón cómo se iba a hacer el laberinto, se intentó hacerlo lo más facil posible dado que sería únicamente un ejmplo. El laberinto tenía forma de 3x3, en dónde únicamente se podía pasar horizantalmente por la fila de enmedio.

2. Ahora bien, lo siguiente en cuestionarse es: ¿Puedo avanzar hacia enfrente?, si la respuesta es "sí", entonces, se debe mover una columna, y en la coordenada 'y' se le aumentaría un lugar.

3. Lo siguiente que hay que ver es que si se ya no se puede avanzar, entonces se regrese un mensaje que diga "Ya no es posible avanzar".

   El código queda así:

```
# -*- coding: utf-8 -*-
L=[[True, True, True],    #Se define el laberinto
[False, False, False],
[True, True, True]]
def resolver(L,e):
print (e)
n=len(L[0])  #Columna 1
x=e[0]
y=e[1]
if y==n-1: salida
return e[0]+1,e[1]+1  #Ya llegué
else: #Si no se ha llegado...
if L[x][y+1] == False: #Se puede mover una columna más, por eso se aumenta a y
e=[x,y+1]
return resolver(L,e)
else:
print 'Ya no se puede avanzar'
```

Lo siguiente que se hizo fue, complicar ligeramente el laberinto, esta vez sería de 3 filas y cuatro columnas; este nuevo laberinto tiene un bloqueo, si

no se puede caminar hacia enfrente, entonces deberemos preguntar si hay otro camino disponible. El segundo camino que se eligió fu hacia abajo.

El programa con esta actualización quedó así:

```
# -*- coding: utf-8 -*-
L=[[True, True, True, True],
[False, False, False, True],  #Forma del laberinto
[True, True, False, True]]
def resolver(L,e):
print (e)
m=len(L)
n=len(L[0])  #Columna 1
x=e[0]
y=e[1]
if y==n-1 or x==m-1 : #casos de salida
return e[0]+1,e[1]+1  #Ya llegué
else: #Si no se ha llegado...
if L[x][y+1] == False: #Se puede mover una columna más, por eso se aumenta a y
e=[x,y+1]
return resolver(L,e)
elif L[x+1][y]== False:
e=[x+1, y]
return resolver(L,e)
else:
print 'Ya no se puede avanzar'

type(L)
e=[1,0]
r=resolver(L,e)
import numpy as np
print(np.matrix(L))
```

Además de esto, casi al final de la clase se alcanzó a ver un problema más; este último consistía en una cadena de ADN,lo que se intentaba hacer era un conteo de su contenido, se logró hacer de 4 formas distintas:

1. ```
   def contar_v1(adn,base):
   adn=list(adn)
   i=0
   for c in adn:
   if c == base:
   i += 1
   return i
   ```

2. ```
   def contar_v2(adn,base):
   i=0
   for c in adn:
   ```

```
if c == base:
i += 1
return i
```

3. ```
   def contar_v3(adn,base):
   i=0
   for j in range(len(adn)):
   if adn[j] == base:
   i += 1
   return i
   ```

4. ```
   def contar_v4(adn,base):
   i=0
   j=0
   while j < len(adn):
   if adn[j] == base:
   i += 1
   j +=1
   return i
   ```