Algoritmos de búsqueda

Razonamiento y planificación automática

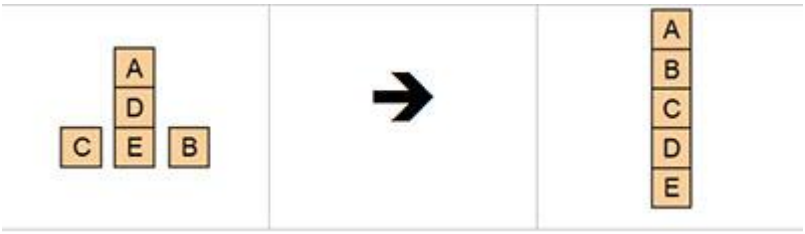
Federico Damián Estebanez

5-18-2019

Enunciado

El objetivo es desarrollar en lenguaje de programación Python una implementación del algoritmo para afianzar la mecánica de implementación de un primer algoritmo de búsqueda.

Deseamos resolver el problema: dada una configuración inicial del mundo de los bloques en la cual los cubos se encuentran como se indica en la figura, deseamos alcanzar una ordenación correcta de los mismos.



Se deberán crear tantas clases o estructuras de datos como sean necesarias para representar el espacio de estados y los nodos de exploración del árbol.

Práctica

Cargamos la labrería numpy, ya que sus funciones nos facilitarán el trabajo con arrays.

Importación de librerías

In [1]:
```
1 #Libraria numpy
2 import numpy as np #Incuye funciones de array
```

El algoritmo utilizado irá recorriendo el array desde la primera posición hasta la última y haciendo intercambio con las posiciones escritas anteriormente si estas tienen un valor mayor.

Algoritmo

In [143]:

```
def sorter(data):
    #Muestreo del array introducido
    print("Array introducido: ", data, "\n")
    #Copiar datos
    sortdata=data.copy()
    #Logitud
    datashape=data.shape[0]
    #Comienzo de la organización de elemento por el segundo elemento
    for i in range(1,datashape):
        #valor de la posición de evalución copiada en variable auxiliar
        aux=sortdata[i]
        #C queda definida por el valor de i
        c=i
        counter=0
        #Evaluación de la posición del array con todas las anteriores
        for j in range(1,i+1):
            #El valor actual evaluado debe ser mayor al anterior
            #En caso de que asi no se de en los datos de entrada cambiará su posición
            #con la posición que guarda un menor valor en el array
            if (sortdata[c]<sortdata[i-j]):
                counter=counter+1
                if counter==1:
                    print ("Valor", sortdata[c], "en la posición", c+1, "desordenado. Procedemos a ordenarlo.")
                print(sortdata[c], "cambia su posición con", sortdata[i-j] )
                sortdata[c]=sortdata[c-1]
                sortdata[c-1]=aux
                c=i-j
                print(sortdata,  "\n")
    #Sort array
    return sortdata
```

Datos introducidos y llamada a la función

In [148]:

```
data=np.random.randint(100, size=5)
print( "Datos ordenados: ", sorter(data))
```

Resultados obtenidos:

```
Array introducido:  [67 51 28  0 99]

Valor 51 en la posición 2 desordenado. Procedemos a ordenarlo.
51 cambia su posición con 67
[51 67 28  0 99]

Valor 28 en la posición 3 desordenado. Procedemos a ordenarlo.
28 cambia su posición con 67
[51 28 67  0 99]

28 cambia su posición con 51
[28 51 67  0 99]

Valor 0 en la posición 4 desordenado. Procedemos a ordenarlo.
0 cambia su posición con 67
[28 51  0 67 99]

0 cambia su posición con 51
[28  0 51 67 99]

0 cambia su posición con 28
[ 0 28 51 67 99]

Datos ordenados:  [ 0 28 51 67 99]
```

Bibliografía

(n.d.). Retrieved from https://nintil.com/2014/01/21/deduccion-induccion-y-abduccion/

UNIR - Actividad. (n.d.). Retrieved from https://campusvirtual.unir.net

UNIR - Clase. (n.d.). Retrieved from https://unir.adobeconnect.com/p4vrbwyeyhxl?session=em2breezecmqe6p4x9rwk7o2&proto=true

Anexo – Código fuente

```
#Libraria numpy

import numpy as np #Incuye funciones de array


def sorter(data):

   #Muestreo del array introducido

   print("Array introducido: ", data, "\n")

   #Copiar datos

   sortdata=data.copy()

   #Logitud

   datashape=data.shape[0]

   #Comienzo de la organización de elemento por el segundo elemento

   for i in range(1,datashape):

      #valor de la posición de evalución copiada en variable auxiliar

      aux=sortdata[i]

      #C queda definida por el valor de i
```

```
        c=i
        counter=0
        #Evaluación de la posición del array con todas las anteriores
        for j in range(1,i+1):
            #El valor actual evaluado debe ser mayor al anterior
            #En caso de que asi no se de en los datos de entrada cambiará su posición
            #con la posición que guarda un menor valor en el array
            if (sortdata[c]<sortdata[i-j]):
                counter=counter+1
                if counter==1:
                    print ("Valor", sortdata[c], "en la posición", c+1, "desordenado. Procedemos a
ordenarlo.")
                print(sortdata[c], "cambia su posición con", sortdata[i-j] )
                sortdata[c]=sortdata[c-1]
                sortdata[c-1]=aux
                c=i-j
                print(sortdata,  "\n")
    #Sort array
    return sortdata


data=np.random.randint(100, size=5)
print( "Datos ordenados: ", sorter(data))
```