PECL3[1]

Miguel Ángel Guerrrero y Jorge Guillamón

3 de abril de 2020

[1]**Puede encontrar el fuente aquí**

Problemas 3 (Busca el mínimo) y 6 (Indiana Croft)

Ambos problemas requieren de la búsqueda de un mínimo para resolverse, en consecuencia, se ha hecho una implementación común usando búsqueda binaria.

Se ha añadido una clase **Función** encargada de generar datos para probar el programa. Para generar esta clase se han de especificar 3 puntos: inicio, mínimo y final (en forma de lista de 2 elementos).Se trata de una función lineal por partes que decrece desde inicio a mínimo y crece de mínimo a final.
Al llamar a **Funcion.calcula(x)** se devuelve el resultado de evaluar la función en x.

Finalmente se ha implementado búsqueda binaria para solucionar el problema.

```
def busqueda(inicio, rango, funcion: Funcion, error):
        if(rango <= error):
                print(inicio)
                return

        f1 = funcion.calcula(inicio + rango/4)
        f2 = funcion.calcula(inicio + rango * 3/4)

        if(f1 < f2): #Mitad inferior
                busqueda(inicio, rango/2, funcion, error)
        else:           #Mitad superior
                busqueda(inicio+rango/2, rango/2, funcion, error)
```

4- El robot embotellador

El algoritmo que hemos creado divide sucesivamente las botellas y los tapones en dos subgrupos más pequeños hasta llegar al menor de todos ellos. Su entrada debe ser dos listas de los mismos números enteros, no necesariamente en el mismo orden.
De acuerdo a las restricciones, se han evitado las comparaciones de dos elementos de una misma lista.
El código viene acompañado de una función para rellenar las listas aleatoriamente, la función main contiene un bucle en el que se imprimen los distintos tiempos de ejecución a medida que se aumenta el tamaño de las listas.
La complejidad ciclomática del algoritmo es $\mathcal{O}(n \log n)$ como podrá comprobar.

```
def robot(botellas: list, tapones: list):
        izdaT = []
        izdaB = []
        dchaT = []
        dchaB = []

        tapon = tapones[0]
        botella = 0

        for i in range(len(tapones)):
                if botellas[i] == tapon:
                        botella = botellas[i]

        tapones.remove(tapon)
        botellas.remove(botella)

        for i in range(len(tapones)):
                if tapones[i] < botella:
                        izdaT.append(tapones[i])
                else:
                        dchaT.append(tapones[i])

                if botellas[i] < tapon:
                        izdaB.append(botellas[i])
                else:
                        dchaB.append(botellas[i])

        if(len(izdaB)>0): robot(izdaB, izdaT)
        if(len(dchaB)>0): robot(dchaB, dchaT)
```

5- Abcdlandia

El problema requiere de la trasposición de la matriz asociada al grafo inicial de la ciudad. Se ha implementado un algoritmo que lleva a cabo la tarea dividiendo en 2 submatrices la actual hasta llegar a una matriz fila, que recorrerá linealmente.
Tiene como entrada dos listas de listas de tamaño nxm y mxn (origen y destino).
El argumento fila hace referencia a la posición de la primera fila de la matriz.
El código fuente contiene una demo para probar la función.

```
def transponer(origen, destino, fila, n_filas):
        if(n_filas == 1):
                for i in range(len(origen[0])):
                        destino[i][fila] = origen[fila][i]

        else:
                n = int(n_filas / 2)
                transponer(origen, destino, fila, n)
                transponer(origen, destino, fila + n, n_filas - n)
```