

Ejercicio de laboratorio 4

1. El código en Python en mi PC excedió el límite de llamadas recursivas utilizando una matriz de tamaño 100x100

```
Traceback (most recent call last):
  File "C:\Users\marco\OneDrive\Documentos\1ESCOM\Análisis y diseño de algoritmos\Códigos_ADA_2404\maze2.py", line 134, in <module>
    path = solve_maze(maze, start, end)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\marco\OneDrive\Documentos\1ESCOM\Análisis y diseño de algoritmos\Códigos_ADA_2404\maze2.py", line 19, in solve_maze
    return dfs(*start)
    ^^^^^^^^^
  File "C:\Users\marco\OneDrive\Documentos\1ESCOM\Análisis y diseño de algoritmos\Códigos_ADA_2404\maze2.py", line 13, in dfs
    path = dfs(nx, ny)
    ^^^^^^^^^
  File "C:\Users\marco\OneDrive\Documentos\1ESCOM\Análisis y diseño de algoritmos\Códigos_ADA_2404\maze2.py", line 13, in dfs
    path = dfs(nx, ny)
    ^^^^^^^^^
  File "C:\Users\marco\OneDrive\Documentos\1ESCOM\Análisis y diseño de algoritmos\Códigos_ADA_2404\maze2.py", line 13, in dfs
    path = dfs(nx, ny)
    ^^^^^^^^^
[Previous line repeated 993 more times]
  File "C:\Users\marco\OneDrive\Documentos\1ESCOM\Análisis y diseño de algoritmos\Códigos_ADA_2404\maze2.py", line 6, in dfs
    if (x, y) == end:
    ^^^^^^^^^
RecursionError: maximum recursion depth exceeded in comparison
```

El código en c corre en aproximadamente 3mls sin imprimir el camino ni la matriz

```
C:\Users\marco\OneDrive\Documentos\1ESCOM\Análisis y diseño de algoritmos\Códigos_ADA_2404>a
This is the maze
Maze solved! Path:

Tiempo de ejecucion: 0.003000

C:\Users\marco\OneDrive\Documentos\1ESCOM\Análisis y diseño de algoritmos\Códigos_ADA_2404>
```

Y el código en c corre en 9 segundos imprimiendo el camino seguido y la matriz

```
(0, 8)
(6, 9)
(5, 9)
(5, 8)
(5, 7)
(4, 7)
(4, 6)
(4, 5)
(5, 5)
(5, 4)
Pasos: 5360

Tiempo de ejecucion: 9.041000

C:\Users\marco\OneDrive\Documentos\1ESCOM\Análisis y diseño de algoritmos\Códigos_ADA_2404>
```

2. Find_min_max: En cada iteración del algoritmo divide el arreglo en dos partes, siendo la primera el primer elemento del arreglo y la segunda el resto, la cual se resuelve recursivamente.

a. C (Tamaño del arreglo: 10,000)

Éxito #stdin #stdout 0.01s 5392KB

comments (0)

stdin

copy

Standard input is empty

stdout

copy

The minimum number in a given array is : 287073

The maximum number in a given array is : 2147463037

Python

Éxito #stdin #stdout 0.05s 10116KB

comments (0)

stdin

copy

Standard input is empty

stdout

copy

The minimum number in a given array is : 352863072

The maximum number in a given array is : 3745948743

3. Quicksort es un algoritmo basado en la técnica Divide and Conquer, en cada iteración del algoritmo divide el arreglo en 2 partes (sin contar el pivote) la primera de los números menores o iguales al pivote y la segunda con los mayores, y resuelve combinando ambas soluciones. También nos damos cuenta de que el funcionamiento y por tal la complejidad del algoritmo es dependiente del pivote utilizado, por lo que es de importancia elegirlo inteligentemente para no tener una complejidad muy elevada.

a. Tiempo de ejecución con 10,000 elementos en Python (Error en tiempo de ejecución al imprimir los 10,000 elementos del arreglo) :

Éxito #stdin #stdout 0.09s 9988KB

Sorted array:

Error en tiempo de ejecución #stdin #stdout 0.08s 10132KB

Sorted array:

337544 441581 538360 820414 938295 1179500 1452264 1869746 1896025 3259298 3810920
4492566 4503144 5021215 5208362 5271848 5695141 5913920 7285710 7461455 8212394 836
2395 8567389 9430084 9484320 9680992 11382975 11680100 12517857 12685456 12924998 1

b. Tiempo de ejecución con 10,000 elementos en C:

```
input Output clear the output ☒ syntax highlight
Éxito #stdin #stdout 0.01s 5516KB
Sorted array:

Error en tiempo de ejecución #stdin #stdout 0.01s 5516KB
Sorted array:
345065 369647 465442 855694 1047821 1342975 1936364 2249036 2398798 2469477 2524050
2553046 2801121 2813326 2994363 3201728 3583337 3613396 3673880 3741468 4166553 425
8119 4696312 4788844 4988859 5113411 5125126 5135494 5149104 5376103 5767303 602345
```

4. Merge-sort:

Este algoritmo de ordenamiento también utiliza una técnica DaC. En cada iteración se divide el arreglo en dos partes y se resuelve recursivamente cada una de ellas, luego se llena el arreglo original tomando el menor de los elementos restantes de cada arreglo hasta terminar con los subarreglos. Como el algoritmo es recursivo, siempre se llega a un caso base que es “fácil” de resolver y se puede combinar para obtener la solución del arreglo original. A diferencia del algoritmo Quicksort, este método no depende de un pivote por lo que la complejidad no varía de manera significativa respecto dos entradas diferentes.

a. Tiempo de ejecución en C

```
input Output
Éxito #stdin #stdout 0.01s 5540KB
Given array is

Sorted array is
```

b. Tiempo de ejecución en Python

```
input Output
Éxito #stdin #stdout 0.08s 10088KB
Sorted array:
```

En los puntos anteriores los algoritmos utilizados en los códigos de ambos lenguajes son los mismos, solo cambia el lenguaje empleado para ejecutarlos. Los códigos en Python parecen mas simples y tienen menos líneas de código, pero se observa que los códigos toman mayor tiempo de ejecución.