Quick Select. Programación Dinámica

Algoritmos y Estructuras de Datos Avanzadas 2023-2024

Practica 3

Fecha de entrega: 15 de diciembre de 2023

ÍNDICE

I. El Problema de Selección

En las funciones inferiores, cuando la tabla tenga cinco o menos elementos, resolver el problema de selección ordenando dicha tabla con el método `np.sort` de Numpy y devolviendo el elemento que corresponda.

I-A. QuickSelect básico

Vamos a implementar el método QuickSelect para la determinación del elemento que ocupa la posición del índice k en una tabla, primero usando una función `split` que parta una tabla usando como pivote el primer elemento de la tabla.

1. Escribir una función
   `split(t: np.ndarray)-> Tuple[np.ndarray, int, np.ndarray]:`
   que reparta los elementos de `t` entre dos arrays con los elementos menores y mayores que `t[0]` y devuelva una tupla con los elementos menores, el elemento `t[0]` y los elementos mayores.
2. Escribir una función
   `qsel(t: np.ndarray, k: int)-> Union[int, None]`
   que aplique de manera recursiva el algoritmo QuickSelect usando la función `split` anterior y devuelva el valor del elemento que ocuparía el índice `k` en una ordenación de `t` si ese elemento existe y `None` si no.
3. Escribir una función no recursiva
   `qsel_nr(t: np.ndarray, k: int)-> Union[int, None]`
   que elimine la recursión de cola de la función anterior.

I-B. QuickSelect_5

Vamos a modificar la implementación anterior de QuickSelect con una selección de pivote mediante el procedimiento "mediana de medianas de cinco elementos".

1. Escribir una función
   `split_pivot(t: np.ndarray, mid: int)-> Tuple[np.ndarray, int, np.ndarray]`
   que modifique la funcion `split` anterior de manera que use el valor `mid` para dividir `t` .
2. Escribir una función
   `pivot5(t: np.ndarray)-> int`
   que devuelve el "pivote 5"del array `t` de acuerdo al procedimiento "mediana de medianas de 5 elementos" y **llamando a la función `qsel5_nr` que se define a continuación.**
3. Escribir una función **no recursiva**
   `qsel5_nr(t: np.ndarray, k: int)-> Union[int, None]`
   que devuelve el elemento en el índice `k` de una ordenación de `t` utilizando la funciones `pivot5, split_pivot` anteriores.

I-C. *QuickSort_5*

Finalmente, vamos a aplicar lo anterior a intentar obtener una versión de QuickSort de coste $O(N \log N)$ en el caso peor.

1. Escribir una función
   `qsort_5(t: np.ndarray)-> np.ndarray`
   que utilice las funciones anteriores `split_pivot, pivot_5` para devolver una ordenación de la tabla `t`.

I-D. *Cuestiones sobre QuickSelect y QuickSort*

Contestar razonadamente a las siguientes cuestiones incluyendo gráficas si fuera preciso.

1. Argumentar en primer lugar que MergeSort ordena una tabla de 5 elementos en a lo sumo 8 comparaciones de clave. Pero, en realidad, en `qsel_5` solo queremos encontrar la mediana de una tabla de 5 elementos, pero no ordenarla. ¿Podríamos reducir así el número de comparaciones de clave necesarias? ¿Cómo?
2. ¿Qué tipo de crecimiento cabría esperar en el caso peor para los tiempos de ejecución de nuestra función `qsort_5`? Intenta justificar tu respuesta experimentalmente.

II. Programación dinámica

II-A. *Dando cambio*

Vamos a implementar el algoritmo de programación dinámica (PD) para hallar el número mínimo de monedas para dar cambio de una cierta cantidad para, a continuación, dar también la combinación óptima de monedas.

1. Escribir una función
   `change_pd(c: int, l_coins: List[int])-> np.ndarray`
   que devuelva la matriz generada por el algoritmo PD para obtener el número mínimo de monedas para dar cambio de una cantidad `c` con las monedas de la lista `l_coins` .
2. Escribir una función
   `optimal_change_pd(c: int, l_coins: List[int])-> Dict`
   que devuelva un dict con claves las monedas de la lista `l_coins` y valores el número de dichas monedas a usar para dar cambio óptimo de la cantidad `c` , haciendo un backtracking adecuado sobre la matriz PD para este problema a partir de la posición del valor óptimo.
   El diccionario devuelto debe estar ordenado por sus claves (esto es, `0, 1, ...`). Para ello, tras efectuar primero el import
   ```
   from collections import OrderedDict
   ```
   en su momento se puede usar el siguiente código
   ```
   d = dict(OrderedDict(sorted(d.items())))
   ```

II-B. *El problema de la mochila*

Vamos a implementar tanto el algoritmo codicioso como el de programación dinámica (PD) para resolver el problema de la mochila 0-1.

1. Escribir una función
   `knapsack_fract_greedy(l_weights: List[int], l_values: List[int], bound: int)-> Dict`
   que devuelva un `dict` con los pesos a tomar de cada elemento en la solución greedy del problema de la mochila fraccionaria. (Usar para ello los enteros `0, 1, ...` , como claves del `dict` .)
2. Escribir una función
   `knapsack_01_pd(l_weights: List[int], l_values: List[int], bound: int)-> int`
   que devuelva el valor óptimo de la mochila 0-1 obtenida mediante PD.

II-C. *Cuestiones sobre las funciones de programación dinámica*

Contestar razonadamente a las siguientes cuestiones.

1. ¿Cuál es el coste en espacio de los algoritmos PD para el problema del cambio y de la mochila 0-1? Si en dichos problemas sólo queremos conocer los valores óptimos y no la composición de las soluciones, ¿hay alguna manera de reducir el coste en memoria? ¿Cómo?
2. Una variante del problema de la mochila 0-1 consiste en suponer que de cada elemento i hay cualquier número de copias. Desarrollar en detalle las fórmulas de una solución PD para esta variante.

III. Material a entregar y corrección

III-A. Material a entregar

Crear una carpeta con el nombre `p3NN` donde `NN` indica el número de pareja y añadir en ella **sólo** los siguientes archivos:

1. Un archivo Python `p3NN.py` con el código de las funciones desarrolladas en la práctica (y NO elementos como scripts de medida de tiempo o dibujo de curvas) así como los `imports` estrictamente necesarios, a saber:

   ```
   import numpy as np
   import itertools

   from typing import List, Tuple, Dict, Callable, Iterable
   ```

   Los nombres y parámetros de las funciones definidas en ellos deben ajustarse EXACTAMENTE a los utilizados en este documento.
   Además, todas las definiciones de funciones deben incorporar los type hints adecuados.
2. Un fichero `p3NN.html` con el resultado de aplicar el comando `pdoc` del paquete `pdoc3` al módulo Python `p3NN.py` .
3. Un archivo `p3NN.pdf` con una breve memoria que contenga las respuestas a las cuestiones de la práctica en formato pdf. **En la memoria se identificará claramente el nombre de los estudiantes y el número de pareja**. **Si se añaden figuras o gráficos, DEBEN ESTAR SOBRE UN FONDO BLANCO.**

Una vez que se hayan puesto estos archivos, comprimir esta carpeta en un archivo llamado `p3NN.zip` o `p3NN.7z` . **No añadir ninguna estructura de subdirectorios a la carpeta.**

La práctica no se corregirá hasta que el envío siga esta estructura y se penalizarán segundas entregas debidas a esta causa.

III-B. Corrección

La corrección se hará en base a los siguientes elementos:

- La ejecución de un script que importará el módulo `p3NN.py` y comprobará la corrección de su código.
  La práctica no se corregirá mientras este script no se ejecute correctamente, penalizando las segundas presentaciones por esta causa.
- La revisión de la documentación del código contenida en los ficheros html generados por `pdoc`. **En particular, las docstrings deben ser escritas muy cuidadosamente.**
  Además, se recomienda que el código Python esté formateado según el estándar PEP-8. Utilizar para ello un formateador como `autopep8` o `black`.
- La aplicación de la herramienta `pylint` de análisis de código, que comprueba un archivo Python de acuerdo a un archivo de configuración (en nuestro caso el archivo `my_pylintrc` acccesible en Moodle y que copiaremos al directorio de la práctica) y proporciona un listado de los errores encontrados así como un score entre 0 y 10 mediante la orden
  `pylint p2XX.py --rc-file my_pylint`
  .
- La revisión de una selección de las funciones de Python contenidas en el módulo `p3NN.py` .
- La revisión de la memoria con las respuestas a las preguntas anteriores.