Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI5651 – Diseño de Algoritmos I
Enero–Marzo 2024

Tarea 6

A continuación encontrará 3 preguntas, cada una dirá cuántos puntos vale en su preámbulo. Sea lo más detallado y preciso posible en sus razonamientos, algoritmos y demostraciones.

Además del informe expresando su solución, debe dar una implementación de su solución en el lenguaje de su elección (solamente como una función; el formato de entrada/salida no es relevante), para todas las preguntas.

La entrega se realizará únicamente por correo electrónico a rmonascal@gmail.com.

Fecha de entrega: Hasta las 11:59pm. VET del **Lunes, 4 de Marzo** (*Semana 8*).

1. (3 puntos) – Considere un arreglo `A[1..N]`, representando una permutación de los números de 1 a N.

   Se desea que ejecute N acciones de la forma `multiswap(a, b)`. Esta acción consite en:

   (a) Intercambiar el valor en la posición a con el valor en la posición b.

   (b) Invocar `multiswap(a+1,b+1)`

   (c) El proceso termina cuando b se sale del rango del arreglo o a alcanza el primer valor de b utilizado.

   A continuación se presenta una implementación en pseudo–Python para `multiswap(a,b)`:

```
def multiswap(A, a, b):
    i, j = a, b
    while i < b and j <= N:
        swap(A, i, j)
        i += 1
        j += 1
```

   Si A inicia como la permutación identidad (números del 1 al N, de menor a mayor) y se ejecutan N operaciones `multiswap(ai, bi)` (donde los valores para ai y bi vienen dados en una lista de tuplas), se desea que imprima el arreglo resultado.

   Diseñe un algoritmo que pueda ejecutar esta acción en tiempo promedio $O(N \log N)$, usando memoria adicional $O(N)$.

   Pista:

   Una estructura de datos que permita *dividir* o *reunir* subarreglos eficientemente con una *alta probabilidad*, puede llevar al camino del bien.

2. (3 puntos) – Sea $A = (N, C)$ un árbol (notemos que $|C| = |N| - 1$) y un predicado $p : C \to \{true, false\}$. Queremos responder consultas que pueden tener una de dos formas:

   - $forall(x, y)$, para $x, y \in N$, que diga si evaluar p para *todas* las conexiones entre entre los nodos x e y resulta en $true$.
   - $exists(x, y)$, para $x, y \in N$, que diga si evaluar p para *alguna* de las conexiones entre entre los nodos x e y resulta en $true$.

   Diseñe un algoritmo que pueda responder Q consultas de cualquiera de estas formas en tiempo $O(|N| + Q \log |N|)$, usando memoria adicional $O(|N|)$

   Pista:

   Realice un precondicionamiento adecuado en $O(|N|)$, que le permita responder cada consulta en $O(\log |N|)$.

3. (3 puntos) – Considere un arreglo `A[1..N]`, representando una permutación de los números de 1 a N.

   Se desea que responda Q consultas de la forma `seleccion(i, j, k)`. Esta consulta pide calcular el k–ésimo elemento del subarreglo `A[i..j]`, si ese subarreglo estuviera ordenado.

   Tomemos, por ejemplo, `A = [2,6,3,1,8,4,7,9,5]`:

   - Al hacer `consulta(2,5,3)`, se refiere al subarreglo comprendido entre las posiciones 2 y 5; es decir: `[6,3,1,4]`. Si ordenáramos este sub–arreglo, el resultado sería `[1,3,4,6]` y el tercero (3–ésimo elemento) sería 4.
   - Al hacer `consulta(3,7,1)`, se refiere al subarreglo comprendido entre las posiciones 3 y 7; es decir: `[3,1,8,4,7]`. Si ordenáramos este sub–arreglo, el resultado sería `[1,3,4,7,8]` y el primero (1–ésimo elemento) sería 1.
   - Al hacer `consulta(1,9,5)`, se refiere al subarreglo comprendido entre las posiciones 1 y 9; es decir: `[2,6,3,1,8,4,7,9,5]`. Si ordenáramos este sub–arreglo, el resultado sería `[1,2,3,4,5,6,7,8,9]` y el quinto (5–ésimo elemento) sería 5.

   Se desea que diseñe un algoritmo que pueda responder todas las consultas usando tiempo $O((N + Q) \log N)$ y memoria $O(N \log N)$.

   Pistas:

   - Consideremos un arreglo de ocurrencias, donde la i–ésima posición representa la ocurrencia del valor i (1 si está y 0 si no está). Consideremos el mismo subarreglo del primer ejemplo: `[6,3,1,4]`. Su arreglo de ocurrencias sería $[1, 0, 1, 1, 0, 1, 0, 0, 0]$.
   - En el arreglo de ocurrencias anterior, ¿cuántas veces aparecen los números del 2 al 5? O, en general, ¿cuántas veces apareces los números del i al j? ¿Hay alguna estructura que permita responder este tipo de consultas *eficientemente*?
   - En algún momento hablamos sobre arreglos cumulativos para resolver consultas estilo `suma(i, j)`. Una idea en particular, que usamos ahí, podría ser de utilidad.
   - Cuando sientan que el problema se vuelve muy difícil, sean *persistentes*.
   - El tiempo y el espacio son uno.

R. Monascal / Febrero 2024