(75.04/95.12) Algoritmos y Programación II

1 de noviembre de 2018

Taller de resolución de problemas de entrevistas

Facultad de Ingeniería
Universidad de Buenos Aires

Aclaración: Algunos enunciados fueron intencionalmente modificados y/o extendidos con el objeto de hacerlos más claros, más completos y/o más cercanos al lenguaje utilizado en la materia.

Primer problema (Facebook)

Un mensaje formado por letras mayúsculas del alfabeto se codifica utilizando la siguiente tabla:

$$\begin{aligned} \texttt{A} &\rightarrow 1 \\ \texttt{B} &\rightarrow 2 \\ &\vdots \\ \texttt{Z} &\rightarrow 26 \end{aligned}$$

Dado un string no vacío s de dígitos decimales, nos interesa determinar la cantidad total de maneras posibles de decodificarlo. Por ejemplo,

- Si $s = \texttt{12}$, existen exactamente dos formas de decodificar el mensaje: `AB` (1 2) y `L` (12).
- Si $s = \texttt{226}$, los mensajes posibles son tres: `BZ` (2 26), `VF` (22 6) y `BBF` (2 2 6).
- Si $s = \texttt{01}$, la respuesta es cero (observar que ningún código empieza en 0).

Programar en C++ la función `int num_decodings(const std::string&)` para resolver este problema. Tener en cuenta que el código debe satisfacer como mínimo el siguiente conjunto de pruebas:

```
void test_undecodable()
{
    int n1 = num_decodings("0");
    int n2 = num_decodings("01");
    int n3 = num_decodings("100");
    int n4 = num_decodings("1001");

    assert(n1 == 0);
    assert(n2 == 0);
    assert(n3 == 0);
    assert(n4 == 0);
}
```

```
void test_one_way()
{
    int n1 = num_decodings("1");
    int n2 = num_decodings("10");
    int n3 = num_decodings("2735");

    assert(n1 == 1);
    assert(n2 == 1);
    assert(n3 == 1);
}
```

```
void test_several_ways()
{
    int n1 = num_decodings("12");
    int n2 = num_decodings("226");
    int n3 = num_decodings("11111");

    assert(n1 == 2);
    assert(n2 == 3);
    assert(n3 == 8);
}
```

Segundo problema (Google)

Se tiene un árbol binario de búsqueda t formado por números enteros y queremos encontrar los valores de los nodos de t con la menor distancia posible a un número dado n. Por ejemplo, cuando $n = 3$,

- Si t contiene un nodo con valor 3, la respuesta es $\langle 3 \rangle$.
- Si t contiene nodos con valores $1, 2, 5, 6$ y 9, la respuesta es $\langle 2 \rangle$.
- Si t contiene nodos con valores $1, 2, 4, 6$ y 9, la respuesta es $\langle 2, 4 \rangle$ (el orden no es importante).

Diseñar y programar un algoritmo para resolver este problema. Tener en cuenta las siguientes consideraciones:

- Comenzar enunciando las propiedades de esta clase de árboles y decidir cómo aprovecharlas para resolver el problema.
- Diagramar una solución, estimar la complejidad asintótica y discutir si es eficiente.
- En caso de existir otras alternativas, decidir cuál implementar en función del *trade-off* entre complejidad de codificación y complejidad algorítmica.
- Escribir dos o tres casos de prueba que cubran las distintas posibilidades (similar al problema anterior).
- **Documentar todo**.

Tercer problema (Facebook)

Dado un arreglo $A[1 \ldots n]$ de números arbitrarios, programar una función `has_zero_sum_elems` que determine si A contiene tres números que sumen cero. A modo de ejemplo, en el siguiente arreglo $A[1 \ldots 10]$ pueden encontrarse los números 4, 2 y -6, cuyos casilleros aparecen sombreados:

8	9	4	-1	5	-6	5	2	0	7

Problemas adicionales

- Invertir una cadena de caracteres in-place (i.e., sin usar estructuras auxiliares).
- Calcular todas las permutaciones de un arreglo (lo tenemos en la guía de ejercicios de recursividad!).
- Decidir si un árbol binario es de búsqueda (también lo tenemos en la guía de árboles).
- Determinar si una lista enlazada contiene un ciclo en tiempo lineal y usando memoria constante.
- Dado un arreglo y un elemento x, eliminar todas las ocurrencias de x in-place y devolver la nueva longitud (no importa lo que quede al final de arreglo después de dicha longitud).
- Programar una función `anagrams` que, dada una secuencia de palabras, devuelva una secuencia de listas de palabras tales que cada lista contenga todas las palabras que son anagramas.
- Se tiene una secuencia de n monedas $c_1, \ldots, c_n$, donde n es par, con las que jugamos el siguiente juego por turnos contra cierto oponente: en cada turno, el jugador elige la primera moneda o la última, la elimina de la secuencia e incrementa su puntaje con el valor de dicha moneda. Proponer un algoritmo para determinar el máximo puntaje que podríamos conseguir haciendo el primer movimiento del juego.
- Decidir si una lista enlazada es palíndromo con una complejidad espacial constante.

Referencias útiles

[1] J. Mongan, N. Kindler, and E. Giguere, *Programming Interviews Exposed: Secrets to Landing Your Next Job*. Programmer to Programmer, Wiley, 2008.

[2] S. Nakariakov, *Cracking Programming Interviews: 350 Questions with Solutions*. USA: CreateSpace Independent Publishing Platform, 2013.

[3] G. L. McDowell, *Cracking the coding interview: 150 programming interview questions and solutions; 5th ed.* Palo Alto, CA: CarrerCup, 2011.