(75.04/95.12) Algoritmos y Programación II

31 de octubre de 2019

Taller de resolución de problemas de entrevistas

Facultad de Ingeniería
Universidad de Buenos Aires

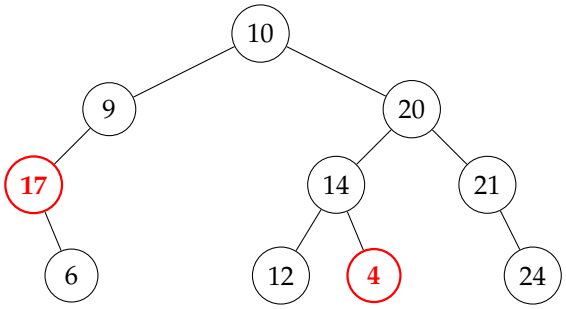
Aclaración: Algunos enunciados fueron intencionalmente modificados y/o extendidos con el objeto de hacerlos más claros, más completos y/o más cercanos al lenguaje utilizado en la materia.

Consideraciones generales

- Comenzar esbozando la solución trivial y estimar su complejidad temporal asintótica.
- A partir de ésta, diagramar una o más soluciones más sofisticadas cuyas complejidades temporales sean inferiores.
- 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.
- **Documentar todo**.

Primer problema (Microsoft/Amazon)

Se tiene un árbol binario de búsqueda en el que, por error, exactamente dos de sus claves han sido intercambiadas, tal como se muestra en la siguiente figura:



Programar la función `find_swapped_keys` que permita encontrar las dos claves intercambiadas en el árbol. Validar la solución a través de los siguientes casos de prueba (el código será provisto por la cátedra):

```
void test_basic_tree()
{
    int k1, k2;
    bst l(2), t(1,&l,NULL);
    bst r(1), t1(2,NULL,&r);

    find_swapped_keys(&t, k1, k2);

    assert(KEYS_OK(k1,k2,1,2));

    find_swapped_keys(&t1, k1, k2);

    assert(KEYS_OK(k1,k2,1,2));
}
```

```
void test_keys_in_left_subtree()
{
    int k1, k2;
    bst ll(3), lr(1), r(7), l(2,&ll,&lr),
        t(5, &l, &r);

    find_swapped_keys(&t, k1, k2);

    assert(KEYS_OK(k1,k2,1,3));
}
```

```
void test_keys_in_right_subtree()
{
    int k1, k2;
    bst rlll(6), rll(5,&rlll,NULL),
        rl(7,&rll,NULL), rr(9),
        r(8,&rl,&rr), t(4, NULL, &r);

    find_swapped_keys(&t, k1, k2);

    assert(KEYS_OK(k1,k2,5,6));
}
```

```
void test_keys_in_both_subtrees()
{
    int k1, k2;
    bst lr(10), l(3,NULL,&lr), rr(4),
        r(9,NULL,&rr), t(7,&l,&r);

    find_swapped_keys(&t, k1, k2);

    assert(KEYS_OK(k1,k2,10,4));
}
```

Segundo problema (Google/Microsoft)

Supongamos una hipotética computadora cuyo teclado sólo tiene las siguientes cuatro teclas:

- `A`, que imprime una letra A en la pantalla,
- ↑, que selecciona todo lo escrito en la pantalla,
- `C`, que copia la selección al buffer, y
- `V`, que pega el contenido del buffer en la pantalla a continuación de lo que ya estaba escrito.

Programar la función `max_number_of_As` que, dado un número natural n, calcula la máxima cantidad de letras A que pueden generarse presionando n teclas en la computadora.

Tercer problema (Facebook)

Dado un arreglo $A[1 \dots n]$ 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 $A[1 \dots 10]$ 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 $c_1, \dots, c_n$, 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.