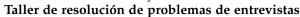
(75.04/95.12) Algoritmos y Programación II 24 de mayo de 2018





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 (?)

Se tiene un arreglo A[1...n] de números enteros positivos y nos interesa ordenarlo de la siguiente manera:

- Primero, por orden ascendente en la cantidad de unos en la representación binaria de los números. Por ejemplo, $7_2 = 111$ y $8_2 = 1000$, por lo que 8 aparecería antes que 7 en el arreglo ordenado.
- A igual cantidad de unos, por orden ascendente en el valor del número. Por ejemplo, 5₂ = 101 y 6₂ = 110 contienen dos unos en su representación binaria. Como 5 < 6, 5 aparecería antes que 6 en el arreglo ordenado.

Proponer un algoritmo para solucionar este problema y programarlo a través de la función void sort_array(std::vector<int>&).
Tener en cuenta que el código debe pasar los siguientes tests:

```
void test_empty()
{
    vector<int> A;
    sort_array(A);
    assert(A.size() == 0);
}
```

```
void test_singleton()
{
   vector<int> A;
   A.push_back(5);

   sort_array(A);

   assert(A.size() == 1);
   assert(A[0] == 5);
}
```

```
void test_complex()
{
    vector<int> A;
    A.push_back(5);
    A.push_back(1);
    A.push_back(8);
    A.push_back(7);
    A.push_back(6);
    sort_array(A);
    assert(A.size() == 5);
    assert(A[0] == 1);
    assert(A[1] == 8);
    assert(A[2] == 5);
    assert(A[3] == 6);
    assert(A[4] == 7);
}
```

Nota: asumir que $1 \le n \le 10^5$ y que $1 \le A[i] \le 10^9$, $1 \le i \le n$.

Segundo problema (Facebook)

Dado un arreglo A[1...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...10] pueden encontrarse los números 4, 2 y -6, cuyos casilleros aparecen sombreados:

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

Tener en cuenta las siguientes consideraciones:

- 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 (similar al problema anterior).
- Documentar todo.

Tercer problema (Google)

Dados dos arreglos A[1...n] y B[1...m] con dígitos decimales, proponer un algoritmo para computar el número más grande de $k \le n + m$ dígitos combinando dígitos de ambos arreglos y preservando el orden relativo de los dígitos de un mismo arreglo. Por ejemplo, dados $A = \langle 3, 4, 6, 5 \rangle$, $B = \langle 9, 1, 2, 5, 8, 3 \rangle$ y k = 5, el algoritmo debe devolver el número 98653.

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!).
- 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).
- Dado un arreglo de números y una ventana deslizante de cierto tamaño *k* que se mueve hacia la derecha de a una posición por vez, encontrar el máximo valor de cada ventana.
- 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.