



**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 (Amazon)

Supongamos la siguiente declaración de listas enlazadas:

```
struct list
{
    list(int data) : data(data), next(NULL) {}
    list(int data, list &next) : data(data), next(&next) {}

    int data;
    list *next;
};
```

Una lista es *palíndromo* si coincide elemento a elemento con su reverso. Programar la función `bool is_palindrome(const list*)` para determinar si una lista arbitraria es palíndromo, sujeta a una complejidad espacial  $\Theta(1)$ . Validar la solución a través de los siguientes casos de prueba:

```
void test_singleton()
{
    list l(1);

    assert(is_palindrome(l));
}
```

```
void test_even_palindrome()
{
    list l4(1), l3(5, l4), l2(5, l3), l1(1, l2);

    assert(is_palindrome(l));
}
```

```
void test_odd_palindrome()
{
    list l5(1), l4(5, l5), l3(7, l4),
        l2(5, l3), l1(1, l2);

    assert(is_palindrome(l));
}
```

```
void test_non_palindrome()
{
    list l5(1), l4(5, l5), l3(7, l4),
        l2(5, l3), l1(2, l2);

    assert(!is_palindrome(l));
}
```

## Segundo problema (Amazon)

- Se tiene una escalera de  $n$  escalones que puede subirse de a uno o dos escalones por vez únicamente. Programar una función `climbing_ways` que calcule la cantidad total de formas de subir la escalera.
- Imaginemos que ahora es posible subir de a  $\{e_1, \dots, e_k\}$  escalones por vez. Reformular el algoritmo para resolver este nuevo problema.

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)

Se tienen dos arreglos  $A[1 \dots n]$  y  $B[1 \dots n]$  de números arbitrarios y un valor objetivo  $k$  y se desea determinar la mínima distancia a  $k$  de cualquier posible suma entre un valor de  $A$  y otro de  $B$ . A modo de ejemplo, considerar los arreglos  $A[1 \dots 4]$  y  $B[1 \dots 4]$  de la figura. Cuando  $k = 10$ , la mínima distancia posible es 1, que viene dada por sumar  $A[4] = 7$  y  $B[1] = 4$ :

A	2	3	-1	7
B	4	-2	1	-3

Programar una función `closest_sum_to_target` para resolver este problema.

## 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.
- Dado un arreglo  $A[1 \dots n]$  de números arbitrarios, determinar si  $A$  contiene tres números que sumen cero.

## 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.