Algoritmos y Estructuras de Datos (CB100, 95.15) – Curso Buchwald

1.er parcialito (1R) – 19/06/2025

Resolvé los siguientes problemas en forma clara y legible. Podés incluir tantas funciones auxiliares como creas necesarias. **Los algoritmos a implementar deben ser de la mejor complejidad posible dadas las características del problema**.

1. Se tiene una cadena que contiene `()` y ningún otro caracter (considerar que un único caracter es de tipo rune). Un ejercicio típico es dada una cadena averiguar si está balanceada (es decir, todos los símbolos de apertura se cierran, y además respetan el orden en el que se abrieron. Ejemplos balanceados: `"()()()"`, o `"(())()"`. No balanceados: `"(()"`, o `")("`.

   Teniendo en cuenta esto, se tiene una cadena que se **asegura** que en caso de borrar algunos paréntesis la cadena será balanceada, se pide implementar una función `func cantBorradosBalanceada(cadena string) int` que dada una cadena de este tipo, devuelva la cantidad mínima de paréntesis que se deben borrar para que la cadena esté balanceada.

   Indicar y justificar la complejidad del algoritmo.

   Ejemplos:

   ```
   cadena: '()' -> 0
   cadena: ')(' -> 2
   cadena: '(()' -> 1
   cadena: ')(()' -> 2
   ```

2. Implementar una función `func esCuadradoPerfecto(n int) bool` que por División y Conquista determine si el número n (un positivo entero) es un cuadrado perfecto. Un número es cuadrado perfecto si existe un número entero x tal que $x^2 = n$. Indicar y justificar la complejidad del algoritmo utilizando el Teorema Maestro.

3. Realizar el **seguimiento** de aplicar RadixSort para ordenar capítulos de las series favoritas de una persona en particular. Se quiere que quede ordenado primero por nombre de la serie, luego por temporada (a igualdad de nombre de serie) y finalmente por número de capítulo (a igualdad de los anteriores). Considerar que el nombre de la serie está representado con un enum, el cual está ordenado según su orden alfabético (ver dorso). Indicar que algoritmo de ordenamiento interno se utilizará y justificar por qué se puede aplicar. Indicar y justificar la complejidad del algoritmo.

Algoritmos y Estructuras de Datos (CB100, 95.15) – Curso Buchwald

1.er parcialito (1R) – 19/06/2025

Resolvé los siguientes problemas en forma clara y legible. Podés incluir tantas funciones auxiliares como creas necesarias. **Los algoritmos a implementar deben ser de la mejor complejidad posible dadas las características del problema**.

1. Se tiene una cadena que contiene `()` y ningún otro caracter (considerar que un único caracter es de tipo rune). Un ejercicio típico es dada una cadena averiguar si está balanceada (es decir, todos los símbolos de apertura se cierran, y además respetan el orden en el que se abrieron. Ejemplos balanceados: `"()()()"`, o `"(())()"`. No balanceados: `"(()"`, o `")("`.

   Teniendo en cuenta esto, se tiene una cadena que se **asegura** que en caso de borrar algunos paréntesis la cadena será balanceada, se pide implementar una función `func cantBorradosBalanceada(cadena string) int` que dada una cadena de este tipo, devuelva la cantidad mínima de paréntesis que se deben borrar para que la cadena esté balanceada.

   Indicar y justificar la complejidad del algoritmo.

   Ejemplos:

   ```
   cadena: '()' -> 0
   cadena: ')(' -> 2
   cadena: '(()' -> 1
   cadena: ')(()' -> 2
   ```

2. Implementar una función `func esCuadradoPerfecto(n int) bool` que por División y Conquista determine si el número n (un positivo entero) es un cuadrado perfecto. Un número es cuadrado perfecto si existe un número entero x tal que $x^2 = n$. Indicar y justificar la complejidad del algoritmo utilizando el Teorema Maestro.

3. Realizar el **seguimiento** de aplicar RadixSort para ordenar capítulos de las series favoritas de una persona en particular. Se quiere que quede ordenado primero por nombre de la serie, luego por temporada (a igualdad de nombre de serie) y finalmente por número de capítulo (a igualdad de los anteriores). Considerar que el nombre de la serie está representado con un enum, el cual está ordenado según su orden alfabético (ver dorso). Indicar que algoritmo de ordenamiento interno se utilizará y justificar por qué se puede aplicar. Indicar y justificar la complejidad del algoritmo.

```
type Capítulo struct {
    serie Serie
    temporada int
    capitulo int
}

const (
    BetterCallSaul Serie = iota
    BreakingBad
    Dexter
    TheOffice
)
```

```
type Serie int

[]Capítulo{
  {serie: BreakingBad, temporada: 1, capitulo: 1},
  {serie: Dexter, temporada: 2, capitulo: 3},
  {serie: BetterCallSaul, temporada: 1, capitulo: 2},
  {serie: TheOffice, temporada: 3, capitulo: 5},
  {serie: BetterCallSaul, temporada: 1, capitulo: 1},
  {serie: BreakingBad, temporada: 2, capitulo: 1},
  {serie: Dexter, temporada: 1, capitulo: 9},
  {serie: TheOffice, temporada: 2, capitulo: 8},
}
```


```
type Capítulo struct {
    serie Serie
    temporada int
    capitulo int
}

const (
    BetterCallSaul Serie = iota
    BreakingBad
    Dexter
    TheOffice
)
```

```
type Serie int

[]Capítulo{
  {serie: BreakingBad, temporada: 1, capitulo: 1},
  {serie: Dexter, temporada: 2, capitulo: 3},
  {serie: BetterCallSaul, temporada: 1, capitulo: 2},
  {serie: TheOffice, temporada: 3, capitulo: 5},
  {serie: BetterCallSaul, temporada: 1, capitulo: 1},
  {serie: BreakingBad, temporada: 2, capitulo: 1},
  {serie: Dexter, temporada: 1, capitulo: 9},
  {serie: TheOffice, temporada: 2, capitulo: 8},
}
```