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

2.do parcialito (1R) – 23/06/2025

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 menor complejidad posible dadas las características del problema planteado.**

1. Implementar una **primitiva** `eliminarColisiones(clave K) []K` para el hash, que elimine del hash todas las claves que colisionen con la clave pasada por parámetro **en el estado actual** (sin eliminar dicha clave del diccionario, si se encuentra), y devuelva dichas claves. Implementar tanto para el hash abierto como para el hash cerrado. Si no se implementa para alguno, el ejercicio no estará aprobable. Indicar y justificar la complejidad de la primitiva para ambos casos.

2. Sobre un AVL cuyo estado inicial puede reconstruirse a partir del preorder: `40 - 10 - 3 - 17 - 15 - 64 - 47 - 74 - 92`, realizar un seguimiento de insertar los siguientes elementos (incluyendo rotaciones intermedias): `20, 23, 13, 14, 16, 12`.

3. Implementar una función `mejoresPromedios(alumnos []Alumno, k int) Lista[Alumno]` que, dado un arreglo de `Alumnos` y un valor entero k, nos devuelva una lista de los k alumnos de mayor promedio (ordenada de mayor a menor). Indicar y justificar la complejidad del algoritmo implementado.

   Considerar que la estructura del alumno es:

```
type Alumno struct {
    nombre  string
    padron  int
    notas   []int
}
```

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

2.do parcialito (1R) – 23/06/2025

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 menor complejidad posible dadas las características del problema planteado.**

1. Implementar una **primitiva** `eliminarColisiones(clave K) []K` para el hash, que elimine del hash todas las claves que colisionen con la clave pasada por parámetro **en el estado actual** (sin eliminar dicha clave del diccionario, si se encuentra), y devuelva dichas claves. Implementar tanto para el hash abierto como para el hash cerrado. Si no se implementa para alguno, el ejercicio no estará aprobable. Indicar y justificar la complejidad de la primitiva para ambos casos.

2. Sobre un AVL cuyo estado inicial puede reconstruirse a partir del preorder: `40 - 10 - 3 - 17 - 15 - 64 - 47 - 74 - 92`, realizar un seguimiento de insertar los siguientes elementos (incluyendo rotaciones intermedias): `20, 23, 13, 14, 16, 12`.

3. Implementar una función `mejoresPromedios(alumnos []Alumno, k int) Lista[Alumno]` que, dado un arreglo de `Alumnos` y un valor entero k, nos devuelva una lista de los k alumnos de mayor promedio (ordenada de mayor a menor). Indicar y justificar la complejidad del algoritmo implementado.

   Considerar que la estructura del alumno es:

```
type Alumno struct {
    nombre  string
    padron  int
    notas   []int
}
```

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

2.do parcialito (1R) – 23/06/2025

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 menor complejidad posible dadas las características del problema planteado.**

1. Implementar una **primitiva** `eliminarColisiones(clave K) []K` para el hash, que elimine del hash todas las claves que colisionen con la clave pasada por parámetro **en el estado actual** (sin eliminar dicha clave del diccionario, si se encuentra), y devuelva dichas claves. Implementar tanto para el hash abierto como para el hash cerrado. Si no se implementa para alguno, el ejercicio no estará aprobable. Indicar y justificar la complejidad de la primitiva para ambos casos.

2. Sobre un AVL cuyo estado inicial puede reconstruirse a partir del preorder: `40 - 10 - 3 - 17 - 15 - 64 - 47 - 74 - 92`, realizar un seguimiento de insertar los siguientes elementos (incluyendo rotaciones intermedias): `20, 23, 13, 14, 16, 12`.

3. Implementar una función `mejoresPromedios(alumnos []Alumno, k int) Lista[Alumno]` que, dado un arreglo de `Alumnos` y un valor entero k, nos devuelva una lista de los k alumnos de mayor promedio (ordenada de mayor a menor). Indicar y justificar la complejidad del algoritmo implementado.

   Considerar que la estructura del alumno es:

```
type Alumno struct {
    nombre  string
    padron  int
    notas   []int
}
```