# Parcial - Doble Map

Primer semestre 2024

Un diccionario doble es un diccionario que tiene 2 claves pero que permite obtener todos los valores asociados a 1 sola de dichas claves por completo. Así, se puede pedir todos los valores asociados a un caso particular de la primera clave o todos los valores asociados a un caso particular de la segunda clave, pero no de ambos al mismo tiempo. Se podría pensar como una matriz en la que queremos acceder a todos los valores de una fila o de una columna de manera eficiente.

Se puede modelar con un TAD `DoubleMap k v` cuya interfaz es la que se enuncia a continuación. Se utilizan los valores *K1* para la cantidad de claves que aparecen como primera clave, *K2* para la cantidad de claves que aparecen como segunda clave y *K12* para la cantidad máxima de primeras claves asociadas a una segunda clave.

- `emptyDM :: DoubleMap k v`, que describe un doble map vacío
  **Eficiencia:** *O(1)*
- `assocDM :: Eq k => k -> k -> v -> DoubleMap k v -> DoubleMap k v`, que describe el resultado de modificar el map dado, agregando el valor a las dos claves dadas, en el orden dado. Esto es, en `assocDM k1 k2 v m` se asocia como valor `v` a la clave `k1` como primera clave y a la clave `k2` como segunda clave, reemplazando el valor anterior asociado a ambas claves, si lo hubiese.
  **Eficiencia:** *O(log K1 * log K2)*
- `lookupDM :: Eq k => Either k k -> DoubleMap k v -> [(k,v)]` que describe la búsqueda por clave en un doble map
  NOTA: el tipo `Either a b` se define como
  ```
  data Either a b = Left a | Right b
  ```
  y se utiliza para distinguir el origen de los datos desde dos fuentes diferentes. En este caso los valores de la forma `(Left k1)` indican que la clave a buscar es la primera, y deben devolverse todos los valores asociados, organizados según la segunda clave, mientras que los valores de la forma `(Right k2)` indican que la clave a buscar es la segunda y deben devolverse todos los valores asociados según la primera clave.
  **Eficiencia:** *O(min (log K1 * K2 log K2) (log K2 * K1 log K1))*
- `keysDM :: DoubleMap k v -> ([k], [k])` que describe el conjunto de claves que son primera clave, y el conjunto de claves que son segunda clave. Los elementos de la primera lista son todas

## Ejercicios

**Ejercicio 1)** En este ejercicio se crea una instancia de DoubleMap para representar los datos de docentes y estudiantes calificando recíprocamente su desempeño. La primera componente es la calificación del estudiante sobre el docente y la segunda la del docente sobre el estudiante. Se define

```
type Nombre = String
type Calificaciones = DoubleMap Nombre (Int, Int)
```

donde los strings representan los nombres de docentes y estudiantes y el par de enteros tiene primero la calificación del docente al estudiante y luego la del estudiante al docente. Implementar como vimos en clase las siguientes operaciones:

a) `docentesCalificados :: Calificaciones -> [Nombre]` que describe la lista de los docentes que recibieron una calificación.
**Eficiencia:** $O(K^2 * K^2)$

b) `topNAfinidad :: Calificaciones -> Nombre -> Int -> [(Int, Nombre)]` que dado un estudiante y un número n, describe la lista de los n docentes que tienen más afinidad con él junto con el valor de esas afinidades. Existe una función

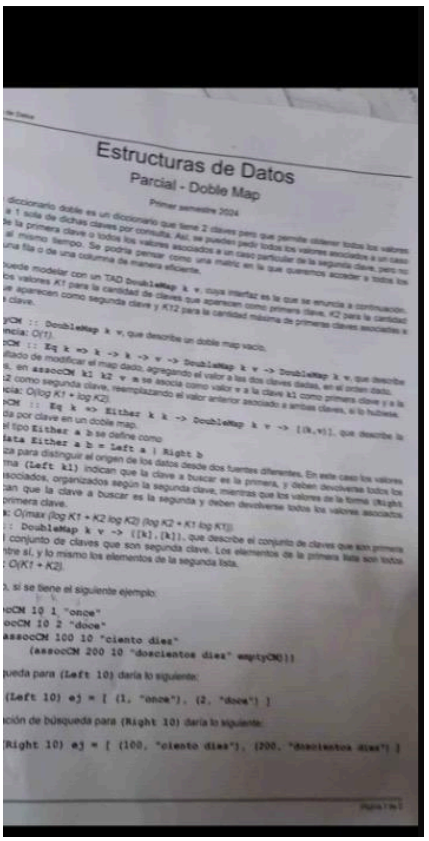
```
afinidad :: (Int, Int) -> Int
```

que calcula la afinidad entre dos calificaciones recíprocas (la que el docente dio al estudiante y la que el estudiante dio al docente).
**AYUDA:** Utilizar una heap para almacenar los nombres de los docentes calificados por el estudiante dado ordenados por afinidad.
**Eficiencia:** $O(N * K^2 * \log K^2 + K^2 * K' * \log K')$, siendo $N$ el valor del número pedido.

Para implementar este TAD, se decide el siguiente tipo de representación:

```
data DoubleMap k v =
  DM (Map k (Map k v)) -- Calif. por clave 1, organizadas por clave 2
     (Map k (Map k v)) -- Calif. por clave 2, organizadas por clave 1
```

donde se utilizan 2 maps simples para guardar en el primero los valores asociados a la 1ra clave dentro de un



# Estructuras de Datos

## Parcial - Doble Map

Primer semestre 2024

diccionario doble es un diccionario que tiene 2 claves pero que permite obtener todos los valores
a 1 sola de dichas claves por consulta. Así, se pueden pedir todos los valores asociados a un caso
le la primera clave o todos los valores asociados a un caso particular de la segunda clave, pero no
al mismo tiempo. Se podría pensar como una matriz en la que queremos acceder a todos los
una fila o de una columna de manera eficiente.

uede modelar con un TAD **DobleMap k v**, cuya interfaz es la que se enuncia a continuación.
os valores *K1* para la cantidad de claves que aparecen como primera clave, *K2* para la cantidad
ue aparecen como segunda clave y *K12* para la cantidad máxima de primeras claves asociadas a
clave.

**yCM :: DobleMap k v**, que describe un doble map vacío.
**ncia:** *O(1)*.

**oCM :: Eq k => k -> k -> v -> DobleMap k v -> DobleMap k v**, que describe
ilidad de modificar el map dado, agregando el valor a los dos claves dadas, en el orden dado.
k, en **asocCM k1 k2 v m** se asocia como valor v a la clave k1 como primera clave y a la
2 como segunda clave, reemplazando el valor anterior asociado a ambas claves, si lo hubiese.
**cia:** *O(log K1 + log K2)*.

**oCM :: Eq k => Either k k -> DobleMap k v -> [(k,v)]**, que describe la
da por clave en un doble map.
el tipo **Either a b** se define como

```
data Either a b = Left a | Right b
```

za para distinguir el origen de los datos desde dos fuentes diferentes. En este caso los valores
rna (**Left k1**) indican que la clave a buscar es la primera, y deben devolverse todos los
sociados, organizados según la segunda clave, mientras que los valores de la forma (**Right
can que la clave a buscar es la segunda y deben devolverse todos los valores asociados
primera clave.

**a:** *O(max (log K1 + K2 log K2) (log K2 + K1 log K1))*

**:: DobleMap k v -> [[k],[k]]**, que describe el conjunto de claves que son primera
l conjunto de claves que son segunda clave. Los elementos de la primera lista son todos
tre sí, y lo mismo los elementos de la segunda lista.
: *O(K1 + K2)*.

), si se tiene el siguiente ejemplo:

```
oCM 10 1 "once"
oCM 10 2 "doce"
assocCM 100 10 "ciento diez"
    (assocCM 200 10 "doscientos diez" emptyCM))
```

ueda para (**Left 10**) daría lo siguiente:

```
(Left 10) ej = [ (1, "once"), (2, "doce") ]
```

ción de búsqueda para (**Right 10**) daría lo siguiente:

```
(Right 10) ej = [ (100, "ciento diez"), (200, "doscientos diez") ]
```