7. Diccionarios y datos ordenados

Carlos Zerón Martínez

Universidad Nacional Autónoma de México

zeron@ciencias.unam.mx

Jueves 7 de Enero de 2021

Diccionarios

- Almacenan entradas (clave, objeto) al igual que los mapas
- Las claves pueden ser objetos que redefinan el método *equals*
- Permite duplicación en claves, como un diccionario real en donde se permiten varias definiciones para una misma palabra

Tipos de diccionarios

- **Diccionario ordenado**. Existe una relación de orden total definida entre las claves. En Java equivale a que el tipo de la clave implementa la interfaz *Comparable*
- **Diccionario no ordenado**. No se cuenta con suposición de orden total, sólo se tiene la prueba de igualdad. En Java equivale que el tipo de la clave redefina el método equals.

Especificación del TDA Diccionario

Datos: Pares (*Clave*, *Elemento*)
Operaciones:

- size() : $\mathbb{N} \cup \{0\}$. Devuelve el número de entradas en el diccionario.
- isEmpty() : $\{V, F\}$. Determina si este diccionario es vacío.
- find(k) : *Objeto*. Si el diccionario contiene una entrada con clave k entonces devuelve el elemento, de otro modo, devuelve **null**.
- insert(k, v) : $\varnothing$. Inserta una entrada con clave k y elemento v.
- remove(k) : *Lista* $<$ *Objeto* $>$. Elimina del diccionario las entradas con clave k y devuelve una lista de los objetos que la tienen, de lo contrario devuelve una lista vacía

Las tablas de dispersión pueden servir como implementación para los diccionarios también.

Diccionarios Ordenados

- Si las claves en un diccionario tienen un orden total y se requiere que estén ordenadas en forma no decreciente por claves, podemos usar un arreglo. Nos referimos a esta implementación como una tabla de búsqueda ordenada.

- Ejemplo de un mapa ordenado (caso especial de diccionario ordenado): En un sistema que mantiene información sobre transacciones financieras que han ocurrido, si las marcas temporales fueran únicas, estas podrían servir como llave (identificador) para recuperar rápidamente información sobre la transacción ocurrida.

- Un mapa no ordenado no proporciona una forma fácil de ordenar los eventos por marca temporal de ocurrencia o buscar un evento más cercano a una determinada marca temporal (extendiendo las operaciones de un mapa ordenado)

Eliminación de entradas por llave (mapa ordenado)

A diferencia de un mapa no ordenado, llevar a cabo actualizaciones en una tabla de búsqueda ordenada es más costoso. La eliminación requiere disminuir en una unidad las posiciones de las entradas con clave mayor o igual a la que se elimina, comenzando a recorrer desde la primera posición donde esto se cumple (a la usanza de la implementación de listas con arreglos).

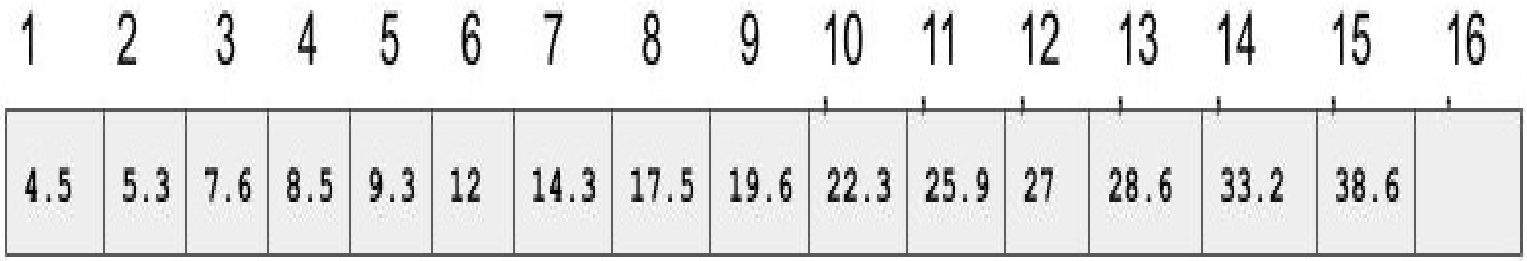
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2.5	4.5	5.3	7.6	8.5	9.3	12	14.3	17.5	19.6	22.3	25.9	27	28.6	33.2	38.6

Eliminación de la clave 2.5, se disminuye en una unidad la posición de todas las entradas siguientes.

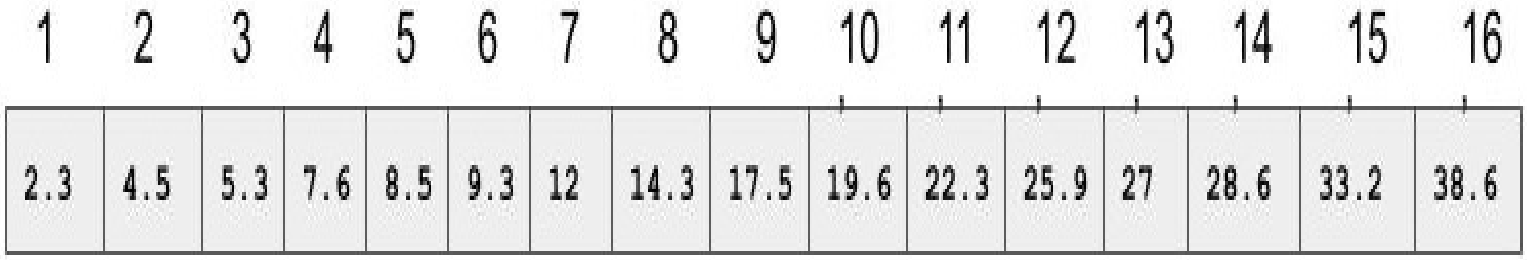
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
4.5	5.3	7.6	8.5	9.3	12	14.3	17.5	19.6	22.3	25.9	27	28.6	33.2	38.6	

Inserción de entrada (mapa ordenado)

La inserción requiere aumentar en una unidad las posiciones de las entradas con clave mayor o igual a la que se inserta, comenzando a recorrer desde la última posición ocupada (a la usanza de la implementación de listas con arreglos).



Inserción de la entrada con clave 2.3, al ser menor que la primera clave 4.5, se aumenta en una unidad la posición de todas las claves existentes.



Complejidad de operaciones fundamentales

La inserción y la eliminación tienen como peor caso $O(n)$, donde n es el número de entradas en el diccionario o mapa (haciendo los recorridos de todas las entradas una posición)

Sin embargo, la operación de búsqueda es más eficiente con un algoritmo denominado búsqueda binaria, que tiene complejidad $O(\log n)$. El análisis quedará fuera del alcance de este curso, pero a continuación veremos la estrategia de este algoritmo.

Búsqueda binaria

Consideramos un arreglo con n entradas. Al estar ordenadas las entradas por clave, la entrada en el índice i, donde $0 \leq i \leq n-1$, tiene una clave mayor o igual que todas las que se encuentran en los índices $0, \ldots, i-1$ y, a su vez, es menor o igual que las claves de los índices $i+1, \ldots, n-1$.

Comparamos k con la entrada ubicada en la posición $mid = \lfloor (low + high)/2 \rfloor$:

- Si $k = e$.getKey(), hemos encontrado una entrada adecuada y termina devolviendo el elemento almacenado.
- Si $k < e$.getKey(), ejecutamos la búsqueda recursivamente en la primera mitad del arreglo, denotada por $A[low \ldots mid - 1]$.
- Si $k > e$.getKey(), ejecutamos la búsqueda recursivamente en la segunda mitad del arreglo, denotada por $A[mid + 1 \ldots high]$.

Búsqueda binaria

El algoritmo (en pseudocódigo) mantiene cuatro parámetros:

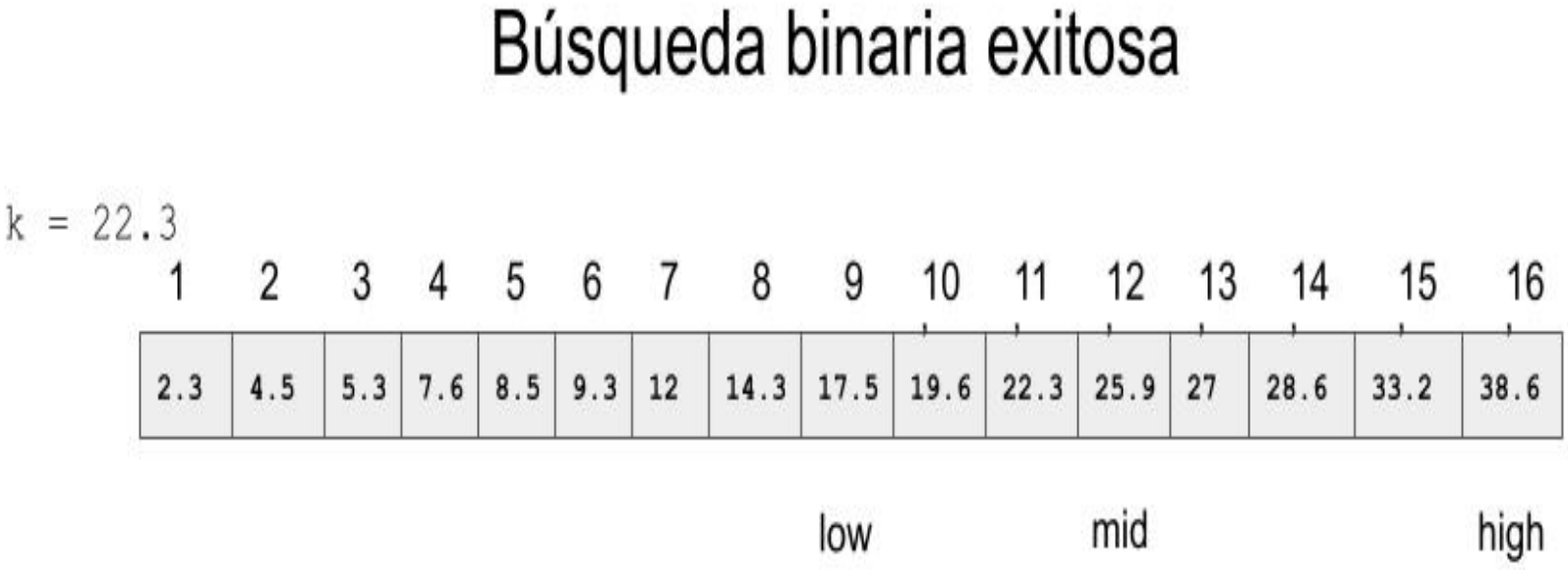
- S tabla de búsqueda ordenada con n entradas en $A[low \ldots high]$
- k clave a buscar
- índices low y $high$, que acotan el espacio de búsqueda

La invocación inicial es con los valores $low = 0$ y $high = n - 1$

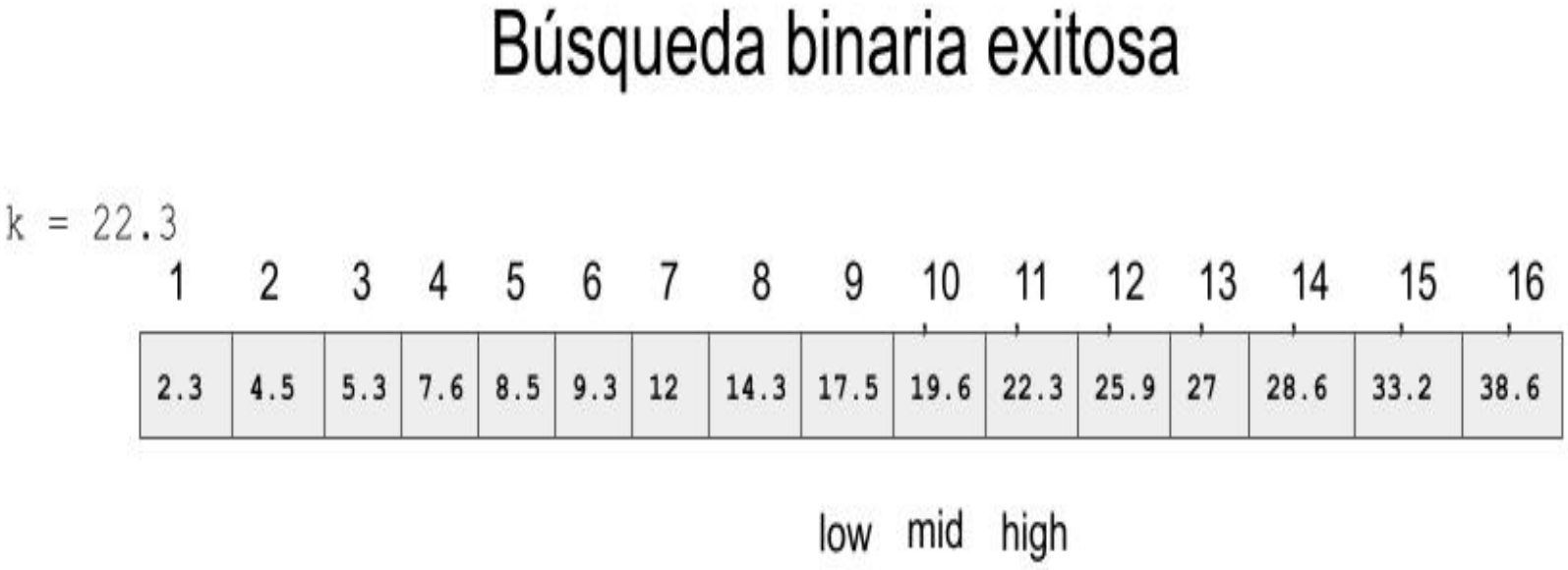
```
binarySearch(S, k, low, high){
 1: if low > high then
 2:     return null;
 3: else
 4:     mid = ⌊(low + high)/2⌋;
 5:     e = S[mid];
 6:     if e.getKey().equals(k) then
 7:         return e.getElement();
 8:     else if e.getKey().compareTo(k) > 0 then
 9:         binarySearch(S, k, low, mid − 1);
10:     else
11:         binarySearch(S, k, mid + 1, high);
12:     end if
13: end if
}
```

Búsqueda binaria

Búsqueda binaria



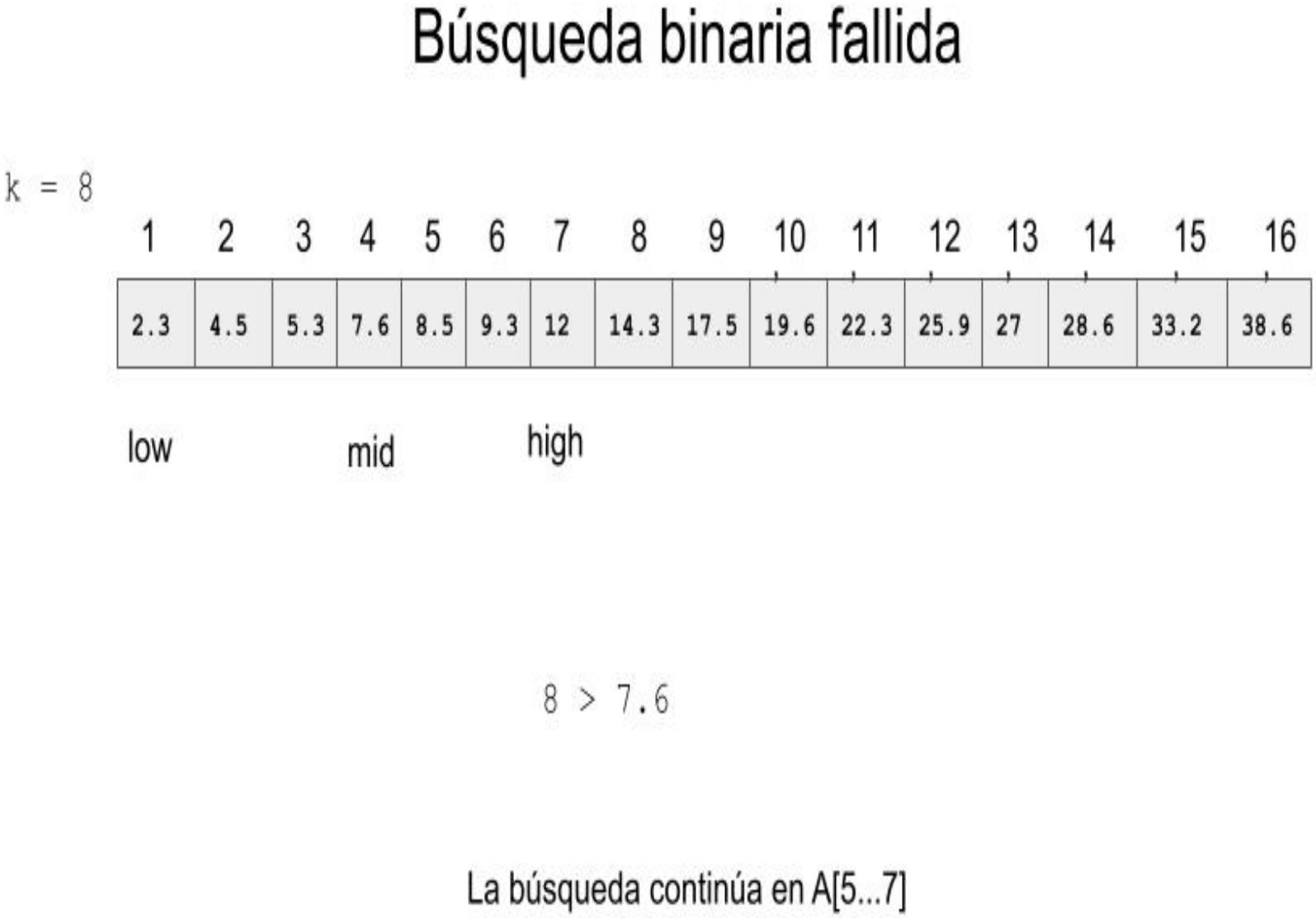
Búsqueda binaria



Búsqueda binaria

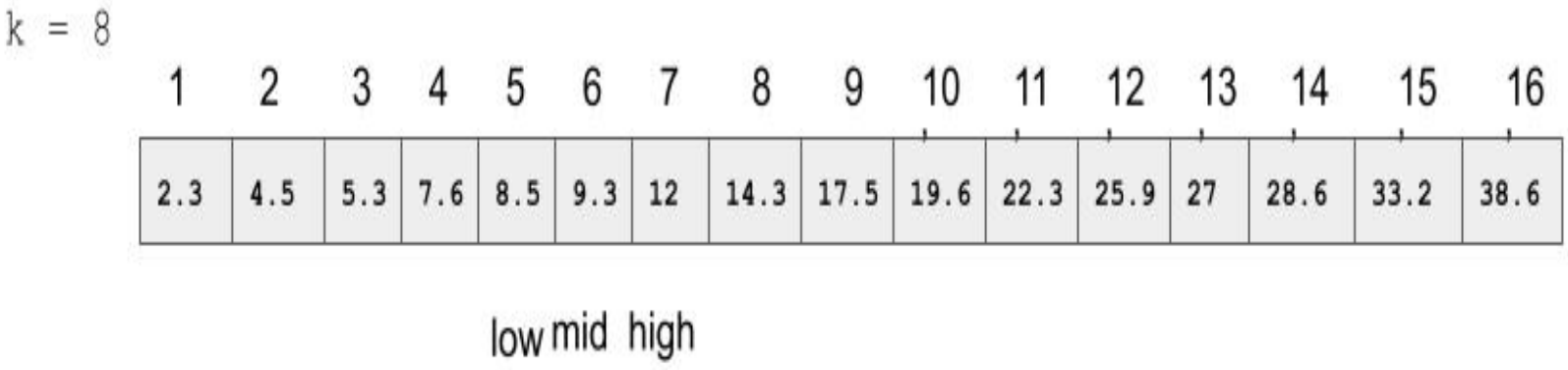
Búsqueda binaria

Búsqueda binaria



Búsqueda binaria

Búsqueda binaria fallida



$8 < 9.3$

La búsqueda continúa en A[5...5]

Búsqueda binaria