Estructura de datos

Tema 5. Ordenamiento y búsqueda

Presenta: Mtro. David Martínez Torres
Universidad Tecnológica de la Mixteca
Instituto de Computación
Oficina No. 37
dtorres@gs.utm.mx

Contenido

Introducción

Aplicar algoritmos de ordenamientos a arreglos o registros, reduce el tiempo al realizar búsquedas.

Son útiles sobre todo en sistemas de gestión o de aplicación como son: Sistema de puntos de venta, sistema bancario, sistema de biblioteca, sistema escolar, sistema de agendas, diccionarios, etc.

Introducción

Los elementos a ordenar se toman de dos en dos, se comparan y se intercambian si no están en el orden adecuado.

Este proceso se repite hasta que se haya analizado todo el conjunto y ya no se presenten intercambios.

Ejemplo de algoritmos de ordenamiento: burbuja, quick-sort, mergesort, etc.

1. Ordenamiento Burbuja

Realiza una prueba de escritorio del algoritmo para el siguiente arreglo:
a[]={4,3,5,2,1}

```
void burbuja(int a[], int tam)
int i,j,temp;
para i ← 1 hasta tam-1
        para j ← 0 hasta tam-2
                si (a[j] > a[j+1])
                        temp ←  a[j]
                        a[j] ←  a[j+1]
                        a[j+1] ←  temp
                finSi
        finPara
finPara
fin
```

```
void burbuja(int a[], int tam)
int i,j,temp;
para i ← 1 hasta tam-1
      para j ← 0 hasta tam-2
            si (a[j] > a[j+1])
                  temp ←  a[j]
                  a[j] ←  a[j+1]
                  a[j+1] ←  temp
            finSi
      finPara
finPara
fin
```

Ejemplo Algoritmo de la Burbuja: a[]={4,3,5,2,1}

tam	i	i<tam-1	j	j<tam-2	a[j]>a[j+1]	a[]
5	1	✓	0	✓	✓	4,3,5,2,1
			1	✓	x	3,4,5,2,1
			2	✓	✓	3,4,5,2,1
			3	✓	✓	3,4,2,5,1
			4	x		3,4,2,1,5
5	2	✓	0	✓	x	3,4,2,1,5
			1	✓	✓	3,4,2,1,5
			2	✓	✓	3,2,4,1,5
			3	✓	x	3,2,1,4,5
			4	x		3,2,1,4,5
Continua siguiente diapositiva						

```
void burbuja(int a[], int tam)
int i,j,temp;
para i ← 1 hasta tam-1
    para j ← 0 hasta tam-2
        si (a[j] > a[j+1])
            temp ←  a[j]
            a[j] ←  a[j+1]
            a[j+1] ←  temp
        finSi
    finPara
finPara
fin
```

…Ejemplo Algoritmo de la Burbuja: a[]={4,3,5,2,1}

tam	i	i<tam-1	j	j<tam-2	a[j]>a[j+1]	a[]
5	3	✓	0	✓	✓	3,2,1,4,5
			1	✓	✓	2,3,1,4,5
			2	✓	x	2,1,3,4,5
			3	✓	x	2,1,3,4,5
			4	x		2,1,3,4,5
5	4	✓	0	✓	✓	2,1,3,4,5
			1	✓	x	1,2,3,4,5
			2	✓	x	1,2,3,4,5
			3	✓	x	1,2,3,4,5
			4	x		1,2,3,4,5
5	5	x				

1.1 Ordenamiento por método de la Burbuja mejorado

```
void burbujaMejorado(int a[], int tam)
int i,j,temp, band ← 1
para i ← 1 hasta TAM-1 and band=1
    band ← 0
    para j ← 0 hasta TAM-i-1
        si (a[j] > a[j+1])
                temp ←  a[j]
                a[j] ←  a[j+1]
                a[j+1] ←  temp
                band ← 1
        finSi
    finPara
finPara
fin
```

Ahora realiza una prueba de escritorio con esta versión del algoritmo para el mismo arreglo: a[]={4,3,5,2,1}

Termine la prueba de escritorio que se muestra.

```
void burbujaMejorado(int a[], int tam)
int i,j,temp, band ← 1
para i ← 1 hasta TAM-1 and band=1
    band ← 0
    para j ← 0 hasta TAM-i-1
     si (a[j] > a[j+1])
          temp ←  a[j]
          a[j] ←  a[j+1]
          a[j+1] ←  temp
          band ← 1
     finSi
    finPara
finPara
fin
```

tam=5 4 3

i	band	i<=tam-1 y band==1	j	j<=tam-i-1	a[j]>a[j+1]	temp	a				
	1						j				
1	0	v v v	0	v	v	4	4	3	5	2	1
	1		1	v	f		3	4	5	2	1
	1		2	v	v	5	3	4	2	5	1
			3	v	v	5	3	4	2	1	5
			4	f							
2		Termine la prueba de escritorio.									

Ordenamiento por Burbuja mejorado para un arreglo de estructuras

```
typedef struct{
    int matricula;
    char nombre[30];
    float promedio;
}Alumno;

void burbujaMejorado(Alumno lista[],int tam){
int i,j,bandera=1;
Alumno temp;
for(i=1;i<tam && bandera==1;i++){
    bandera=0;
    for(j=0;j<=tam-i-1;j++){
        if(lista[j].matricula>lista[j+1].matricula){
            temp=lista[j];
            lista[j]=lista[j+1];
            lista[j+1]=temp;
            bandera=1;
        }
    }
}
}
```

Realice una prueba de escritorio para el siguiente arreglo {{4,Juan,8.5}{2,Ana,8.2}{3,Juan,6.5}{1,Hector,7.2}{5,Ana,7.1}} por un criterio que decida.

Ordenamiento por Burbuja de una lista enlazada desordenada

Realice una prueba de escritorio para la siguiente lista de manera ascendente:

5->1->10->7->NULL

```
void burbuja(tipoListaPtr listaT, int tam){
int i,j,band=1,temp;
tipoListaPtr aux;

for(i=1;i<tam && band==1 &&listaT!=NULL;i++){
    band=0;
    aux=listaT;
    for(j=0;j<tam-i&&aux!=NULL;j++){
        if(aux->dato > aux->sig->dato){
            temp=aux->dato;
            aux->dato=aux->sig->dato;
            aux->sig->dato=temp;
            band=1;
        }
        aux=aux->sig;
    }
}
}
```

Resuelva el siguiente ejercicio con burbuja mejorado.

```
typedef struct {
        int matricula;
        char nombre[30]
        float promedio;
}tipoAlumno;
```

```
int main(){
tipoAlumno lista[5]={4,"Juan",8.5,
                     2,"Ana",8.2,
                     3,"Juan",6.5,
                     1,"Hector",7.2,
                     5,"Ana",7.1};
...
}
```

* Probar con el arreglo de ejemplo y también proporcionar la opción de generar un arreglo de estructuras de manera aleatoria tanto matrícula, nombre (puede usar un arreglo de cadenas) y promedio.
* Realizar un ordenamiento del arreglo de estructuras anterior por los siguientes criterios: nombre y promedio, ambos de forma **ascendente**. Ejemplo:

Matrícula	Nombre	Promedio
5	Ana	7.1
2	Ana	8.2
1	Héctor	7.1
3	Juan	6.5
4	Juan	8.5

2. Ordenamiento quicksort

Es un método de ordenamiento rápido. Mejor que el método de intercambio directo y fue propuesto por Charles Antony Richard Hoare.

Se basa en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo proporcional a $n \log n$.

Algoritmo quicksort

Ordenar el siguiente arreglo de forma **ascendente**

A: {15 67 8 16 44}

1. Se selecciona un elemento X de una posición cualquiera del arreglo conocido como pivote. Ejemplo A[0]
2. Se recorre el arreglo de derecha a izquierda comparando si los elementos son >= a X. Si un elemento no cumple, se intercambian y se almacena en una variable la posición del elemento intercambiado (se acota el arreglo por la derecha).
3. Ahora se inicia el recorrido pero de izquierda a derecha, comparando si los elementos son <= a X. Si un elemento no cumple, se intercambian los mismos y se almacena en otra variable la posición del elemento intercambiado (se acota el arreglo por la izquierda).
4. Se repiten los pasos anteriores hasta que el elemento X encuentra su posición correcta en el arreglo.

Ejemplo de ordenamiento con quicksort

Ordenar el siguiente arreglo de forma ascendente

A: {15 67 8 16 44}

pivote A[0], X←15

Primera pasada:

Recorrido de derecha a izquierda

A[4]>=x (44>=15) No hay intercambio

A: 15 67 8 16 44

A[3]>=X (16>=15) No hay intercambio

A: 15 67 8 16 44

A[2]>=X (8>=15) Hay intercambio y se acota

A: 15 67 8 16 44

A: 8 67 15 16 44

X=15

Recorrido de izquierda a derecha

A[1]<=X (67<=15) Hay intercambio y se acota

A: 8 67 15 16 44

Segunda pasada:

Recorrido de derecha a izquierda

A[1]>=15 (15>=15) No hay intercambio

A: 8 15 67 16 44

A[0]>=15 (8>=15) No hay intercambio

A: 8 15 67 16 44

Recorrido de izquierda a derecha

A[1]<=15 (15<=15) No hay intercambio

A: 8 15 67 16 44

A[2]<=15 (67<=15) No hay intercambio

A: 8 15 67 16 44

Se obtienen las 2 primeras sublistas, las cuales se verificarán y se procederá en revisar si se aplica el proceso anterior.

A: 8 15 67 16 44

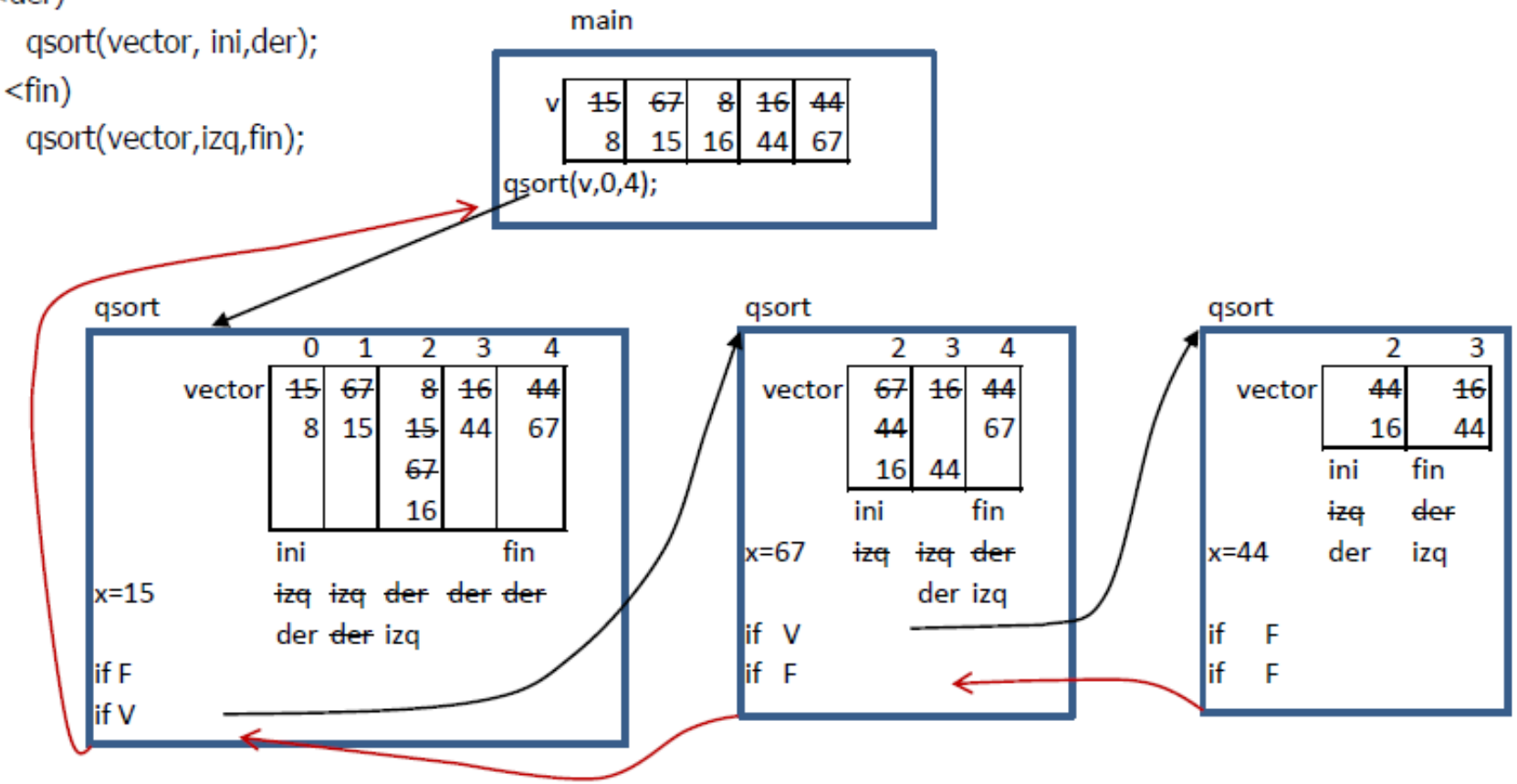
```
void qsort(int vector[],int ini, int fin){
int izq,der,x,aux;
x=vector[ini];
izq=ini; der=fin;
do {
    while(vector[der]>=x && der>ini)
            der--;
    if (izq<=der){
            aux= vector[izq];
            vector[izq]=vector[der];
            vector[der]=aux;
            izq++;
     }
    while(vector[izq]<=x && izq<fin)
            izq++;
    if (izq<=der) {
            aux= vector[izq];
            vector[izq]=vector[der];
            vector[der]=aux;
            der--;
     }
}while(izq<=der);
if(ini<der)
        qsort(vector, ini,der);
if(izq <fin)
        qsort(vector,izq,fin);
}
```

Ahora se presenta el algoritmo quicksort en su forma recursiva y se probará con el mismo arreglo [15 67 8 16 44] para entender su comportamiento.

Ahora realice la prueba de escritorio que a continuación se muestra.

```
void qsort(int vector[],int ini, int fin){
int izq,der,x,aux;
x=vector[ini];
izq=ini; der=fin;
do {
    while(vector[der]>=x && der>ini)
        der--;
    if (izq<=der){
        aux= vector[izq];
        vector[izq]=vector[der];
        vector[der]=aux;
        izq++;
     }
    while(vector[izq]<=x && izq<fin)
        izq++;
    if (izq<=der) {
        aux= vector[izq];
        vector[izq]=vector[der];
        vector[der]=aux;
        der--;
     }
}while(izq<=der);
```

```
if(ini<der)
        qsort(vector, ini,der);
if(izq <fin)
        qsort(vector,izq,fin);
}
```



Para fines de comprensión, en cada función se muestra una copia del arreglo.

Resuelva el siguiente ejercicio con Quicksort.

```
typedef struct {
        int matricula;
        char nombre[30]
        float promedio;
}tipoAlumno;
```

```
int main(){
tipoAlumno lista[5]={4,"Juan",8.5,
                     2,"Ana",8.2,
                     3,"Juan",6.5,
                     1,"Hector",7.2,
                     5,"Ana",7.1};
...
}
```

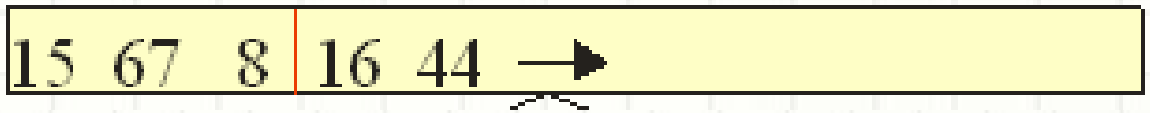
* Probar con el arreglo de ejemplo y también proporcionar la opción de generar un arreglo de estructuras de manera aleatoria tanto matrícula, nombre (puede usar un arreglo de cadenas) y promedio.
* Realizar un ordenamiento del arreglo de estructuras anterior por los siguientes criterios: nombre y promedio, ambos de forma **descendente**. Ejemplo:

Matrícula	Nombre	Promedio
3	Juan	8.5
4	Juan	6.5
1	Héctor	7.1
5	Ana	8.2
2	Ana	7.1

3. Ordenamiento Merge sort

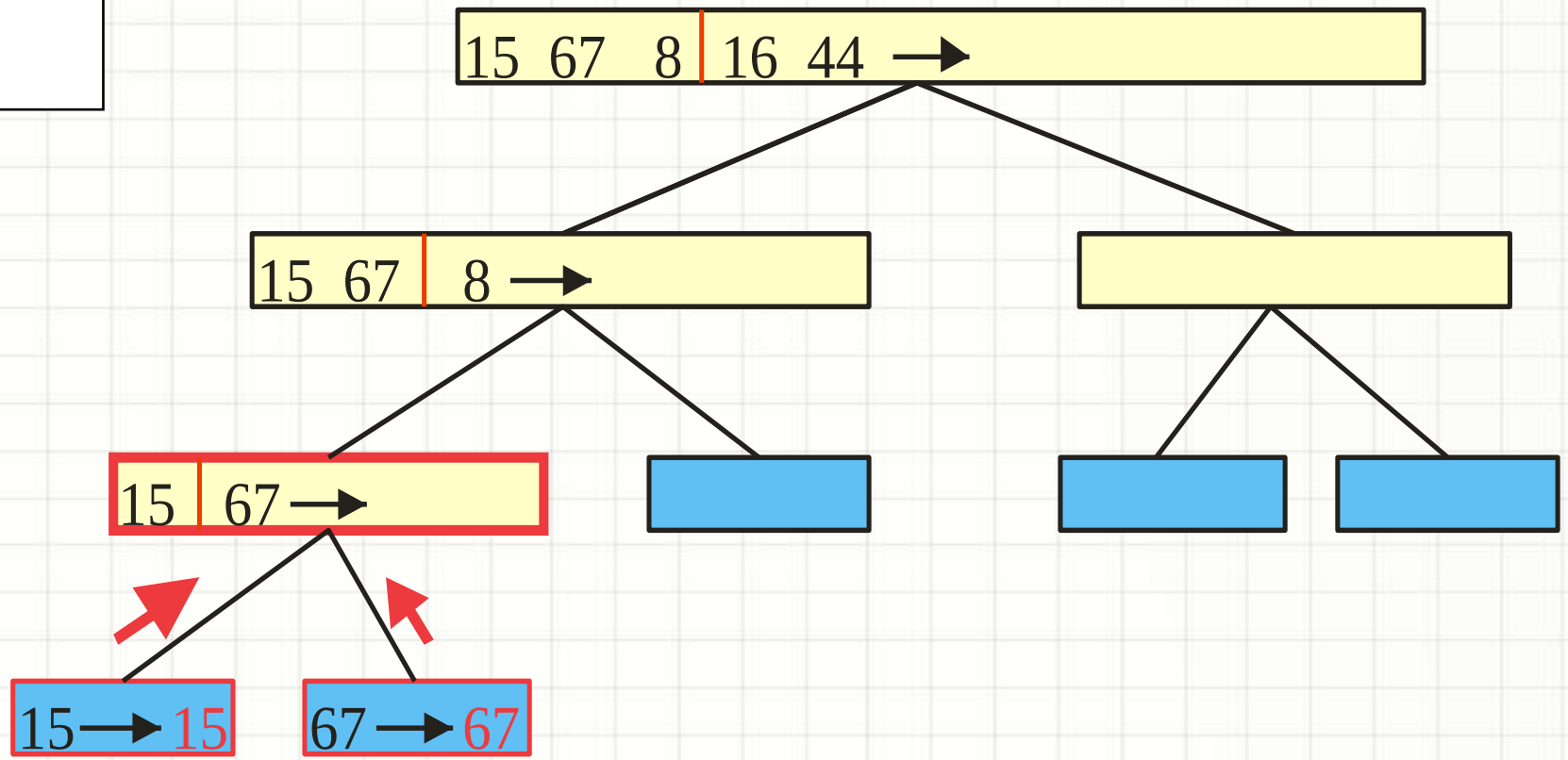
Merge sort utiliza la técnica divide y vencerás para ordenar un arreglo de registros.

El arreglo es dividido de manera recursiva en dos subarreglos de tamaño similar, se detiene en cuanto el tamaño del subarreglo es 1. Después se realiza una mezcla para ordenar los subarreglos, hasta reconstruir el arreglo original de tamaño ***n***

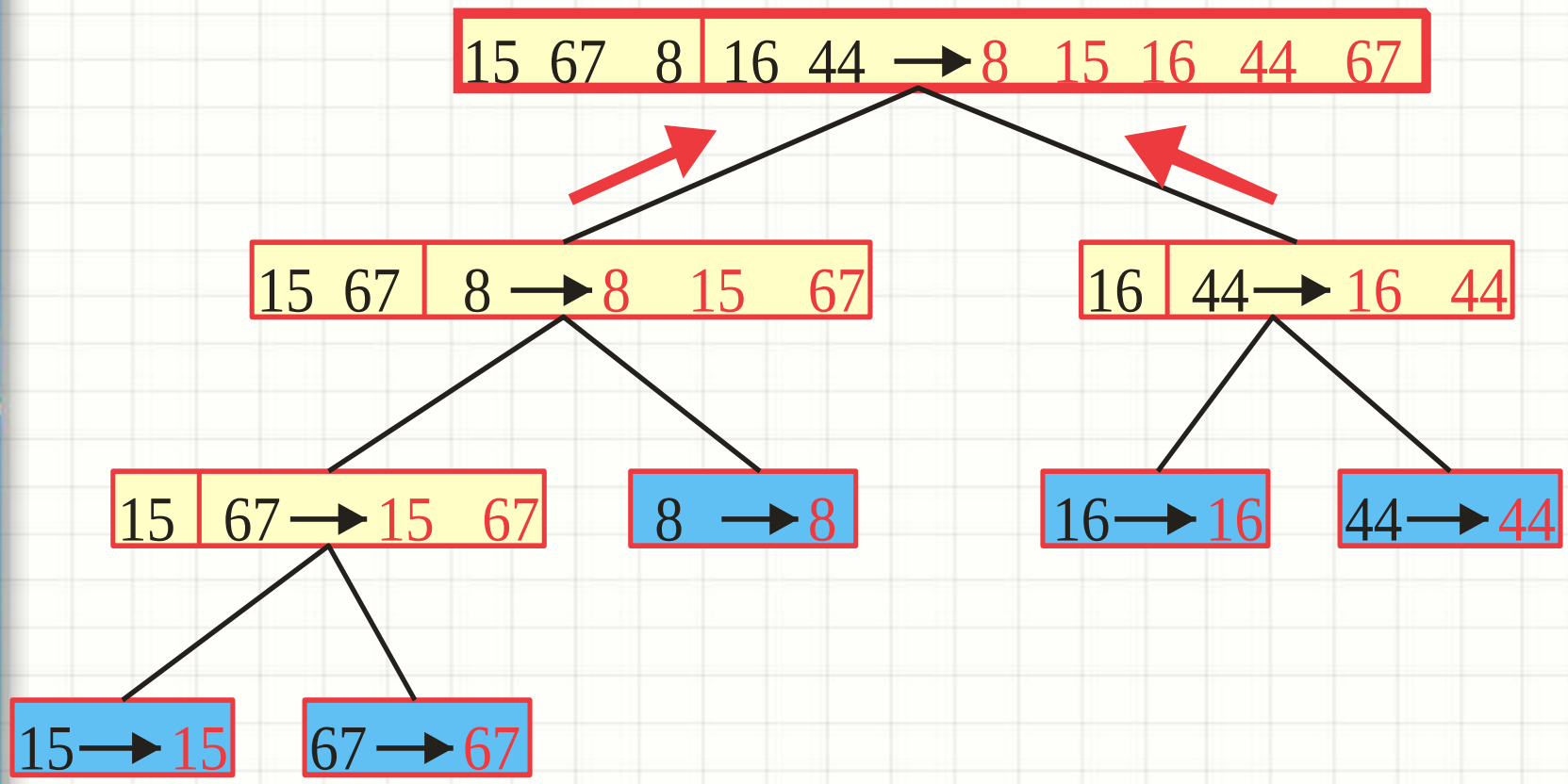


3. Ordenamiento Merge sort: ejemplo

Ordenar el siguiente arreglo
a={15, 67, 8, 16, 44}



3. Ordenamiento Merge sort: ejemplo



Después de particionar recursivamente y mezclar los subarreglos, tenemos el resultado final.

3. Ordenamiento Merge sort

Primera parte del algoritmo

```
void mergeSort(int vector[],int ini, int fin){
int medio;
if(ini<fin){
        medio=(ini+fin)/2;
        mergeSort(vector, ini, medio);
        mergeSort(vector, medio+1, fin);
        merging(vector, ini, medio, fin);
}
}
```

Función que realiza la mezcla de las dos sublistas

```
void merging(int vector[], int ini, int medio, int fin){
int ini1, ini2, i,b[TAM];
for(ini1=ini, ini2=medio+1, i=ini; ini1<=medio && ini2<=fin;i++){
        if(vector[ini1]<=vector[ini2])
                b[i]=vector[ini1++];
        else
                b[i]=vector[ini2++];
}
while(ini1<=medio)
        b[i++]=vector[ini1++];
while(ini2<=fin)
        b[i++]=vector[ini2++];
for(i=ini;i<=fin;i++)
        vector[i]=b[i];
}
```

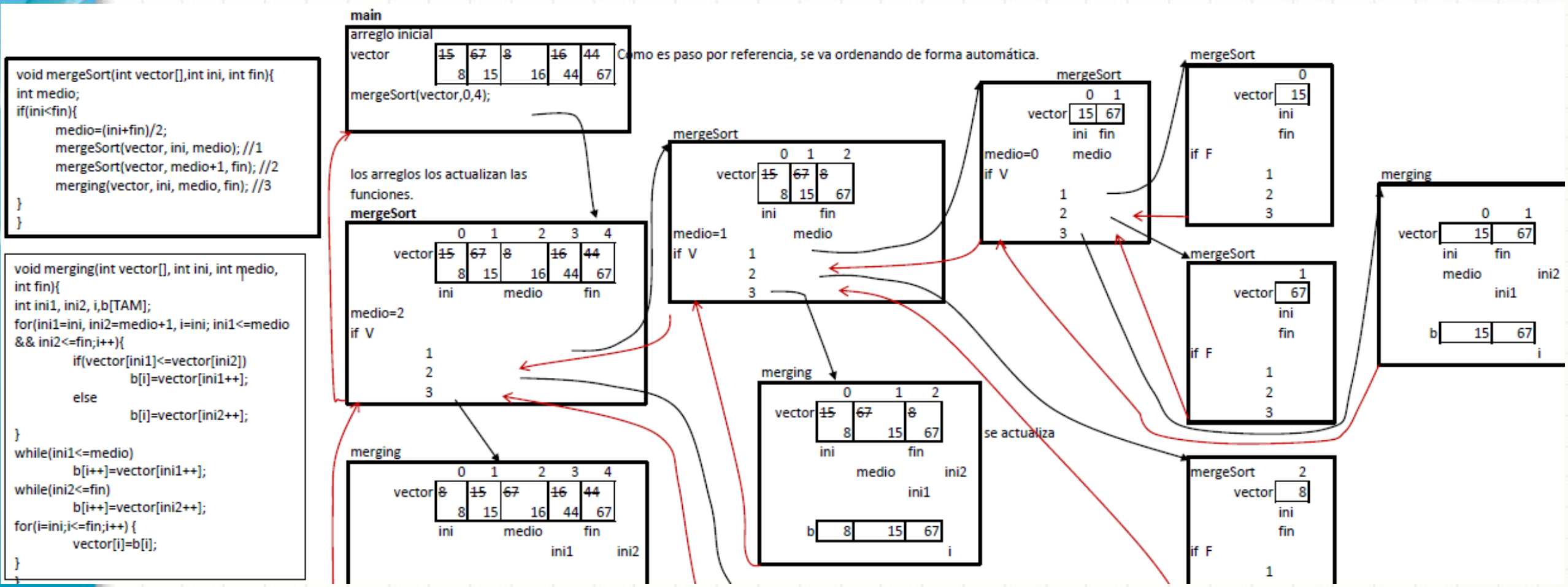
Realice una prueba de escritorio para el arreglo

[15 67 8 16 44], aplicando las funciones correspondientes para tener una mejor comprensión.

Ahora termine de realizar la prueba de escritorio que a continuación se muestra.

```
void mergeSort(int vector[],int ini, int fin){
int medio;
if(ini<fin){
        medio=(ini+fin)/2;
        mergeSort(vector, ini, medio); //1
        mergeSort(vector, medio+1, fin); //2
        merging(vector, ini, medio, fin); //3
}
}
```

```
void merging(int vector[], int ini, int medio,
int fin){
int ini1, ini2, i,b[TAM];
for(ini1=ini, ini2=medio+1, i=ini; ini1<=medio
&& ini2<=fin;i++){
        if(vector[ini1]<=vector[ini2])
                b[i]=vector[ini1++];
        else
                b[i]=vector[ini2++];
}
while(ini1<=medio)
        b[i++]=vector[ini1++];
while(ini2<=fin)
        b[i++]=vector[ini2++];
for(i=ini;i<=fin;i++) {
        vector[i]=b[i];
}
}
```



Resuelva el siguiente ejercicio con Merge Sort.

```
typedef struct {
        int matricula;
        char nombre[30]
        float promedio;
}tipoAlumno;
```

```
int main(){
tipoAlumno lista[5]={4,"Juan",8.5,
                     2,"Ana",8.2,
                     3,"Juan",6.5,
                     1,"Hector",7.2,
                     5,"Ana",7.1};
...
}
```

* Probar con el arreglo de ejemplo y también proporcionar la opción de generar un arreglo de estructuras de manera aleatoria tanto matrícula, nombre (puede usar un arreglo de cadenas) y promedio.
* Realizar un ordenamiento del arreglo de estructuras anterior por los siguientes criterios: **nombre** (descendente) y **promedio** (ascendente). Ejemplo:

Matrícula	Nombre	Promedio
3	Juan	6.5
4	Juan	8.5
1	Héctor	7.1
5	Ana	7.1
2	Ana	8.2

4. Búsqueda secuencial

Este tipo de búsqueda se aplica a un arreglo aunque no este ordenado.

Dependiendo del planteamiento del problema y si haya datos repetidos, podrá encontrar el primer dato, el registro completo y devolver la posición, devolver valor booleano, imprimir que si se encontró el dato, y terminar la búsqueda. Por otro lado, puede devolver el número de veces que se encontró el dato, o las posiciones donde se encontró el dato, mediante un arreglo con todas las posiciones del dato, etc.

4. Búsqueda secuencial

En un arreglo “no ordenado”, se tiene que recorrer todo el arreglo. Para este caso, encuentra la posición en donde está el elemento.

```
typedef struct {
        int matricula;
        char nombre[30]
        float promedio;
}tipoAlumno;
int busquedaSecuencial(Alumno alumnos[], int tam, char nombre[]);
int main(){
tipoAlumno alumnos[5]={{4,”Juan”,8.5},{2,”Ana”,8.2},{3,”Juan”,6.5},
                       {1,”Hector”,7.2},{5,”Ana”,7.1}};
…
}
int busquedaSecuencial(Alumno alumnos[], int tam, char nombre[]){
int i, indice=-1;
for(i=0;i<tam ;i++){
    if(strcmp(alumnos[i].nombre, nombre)==0){
        indice=i;
        i=tam;
    }
}
return indice;
}
```

4. Búsqueda secuencial: ejercicio

Dado un arreglo alumnos[]={{4,”Juan”,8.2},{2, “Ana”, 7.2},{3,”Juan”,6.5},{1,”Hector”,7.2},{5,”Ana”,8.2}};

Escriba las siguientes funciones de búsqueda secuencial con los criterios que se indican:

1. Por promedio, devolviendo los registros correspondientes.
2. Por matrícula, devolviendo el registro correspondiente.
3. Por nombre, devolviendo el número de alumnos encontrados

5. Búsqueda binaria

Este tipo de búsqueda se debe aplicar **solo sí el arreglo está ordenado**.

Por lo tanto, **solo debe encontrar una coincidencia** en caso que se encuentre el dato.

Podría devolver un valor booleano, la posición del dato o un registro

5. Búsqueda binaria

```
int busquedaBinaria(tipoAlumno alumnos[], int tam,
int matricula){
int i=0, j=tam-1,medio;
do{
      medio=((i+j)/2);
      if(matricula>alumnos[medio].matricula)
            i=medio+1;
      else
            j=medio-1;
}while(alumnos[medio].matricula != matricula && i<=j );
if(matricula != alumnos[medio].matricula)
      medio=-1;
return medio;
}
```

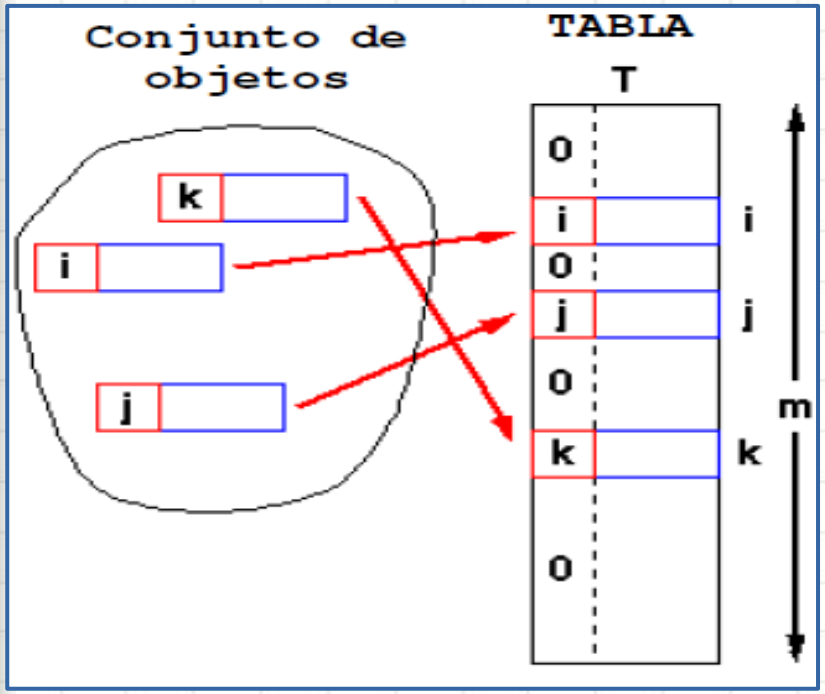
Se aplica a un arreglo ordenado. No recorre todo el arreglo, si encuentra el elemento se detiene. Encuentra la posición o devuelve el elemento.

Realice una prueba para el siguiente arreglo:

alumnos={{1,”Juan”,8.2},{2, “Ana”, 7.2},{3,”Juan”,6.5},{4,”Hector”,7.2},{5,”Ana”,8.2}};

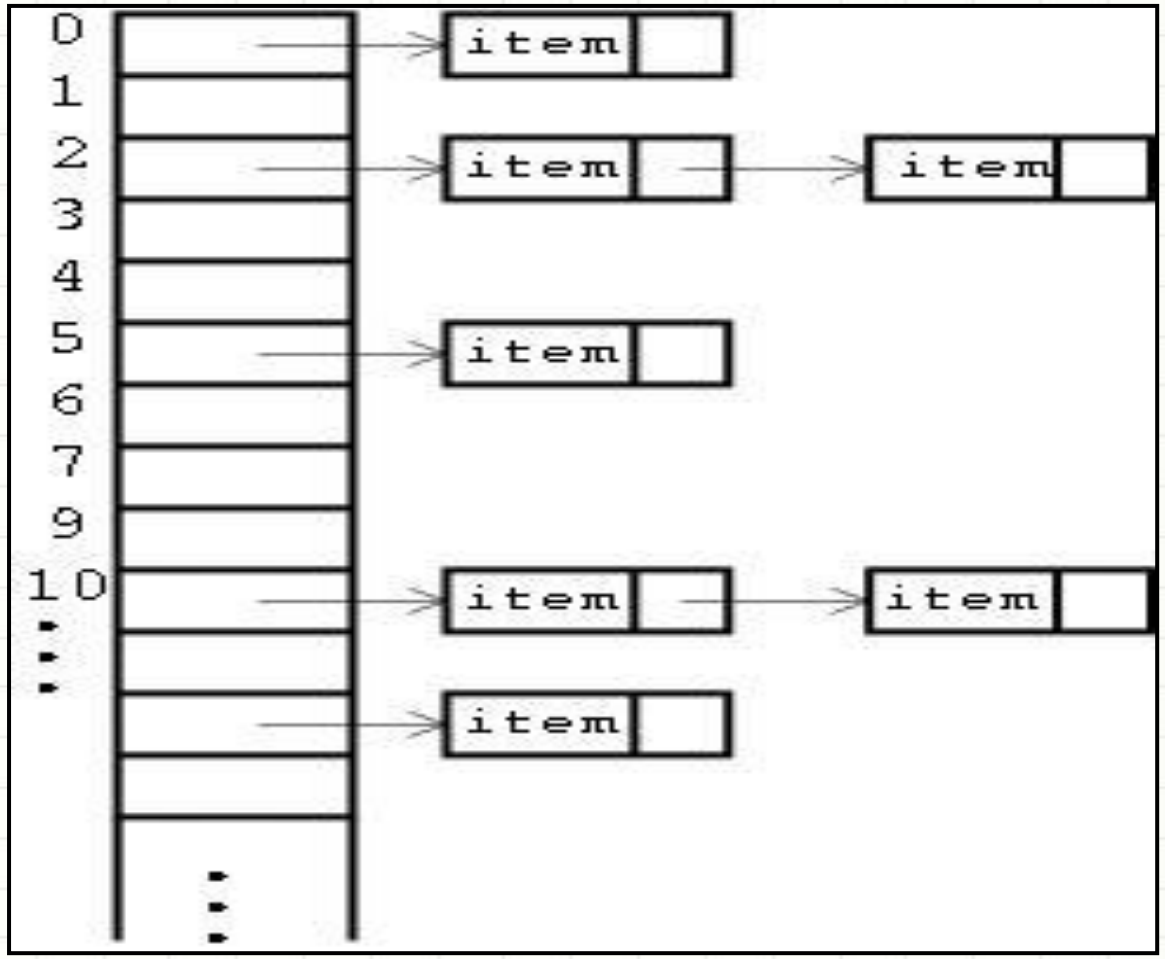
6. Búsqueda en tablas hash

Consiste en utilizar parte de la información del objeto a ordenar como índice de ubicación en una tabla



6. Búsqueda en tablas hash

En caso que los registros se repitan, considere una tabla hash de apuntadores a registros y que se inserten por ejemplo como una pila, cola o lista.

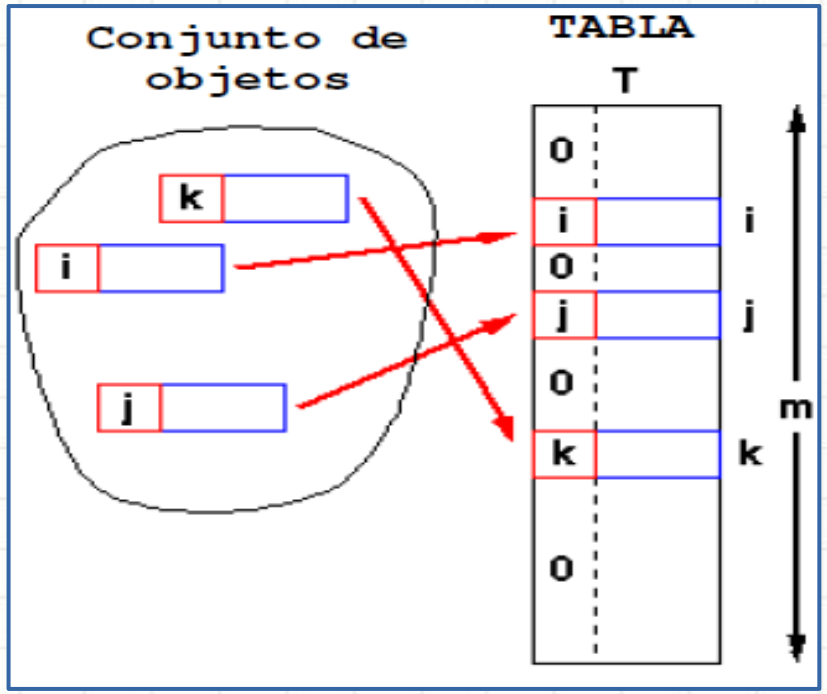


6. Búsqueda en tablas hash

- Para conseguir el índice de la estructura que almacena los objetos, se debe contar con una función hash que, tomando los atributos necesarios de los objetos, devuelva un índice.

  m = hash (attr 1, attr 2, attr 3, ..., attr n)

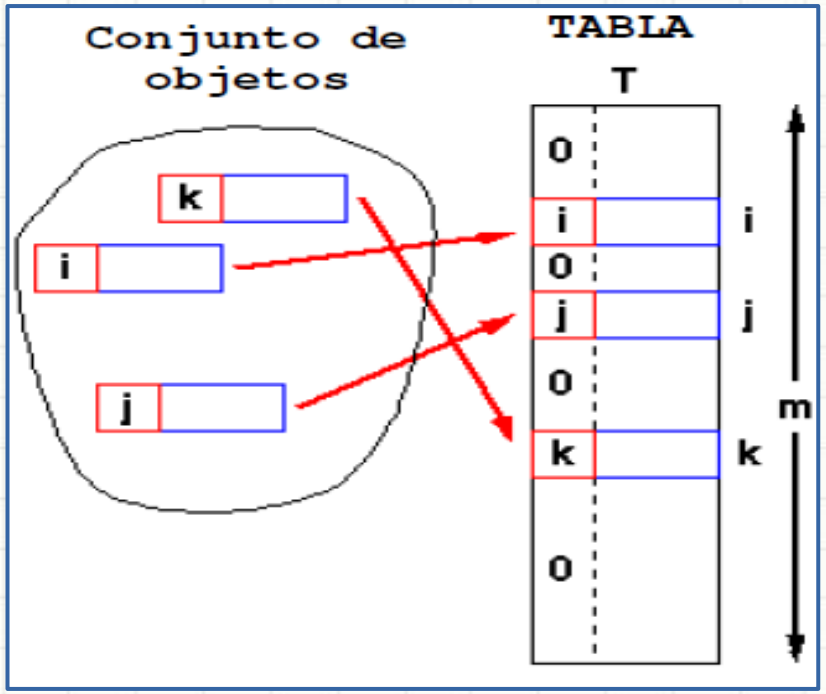
- siendo n = número de atributos que se quieren involucrar en el cálculo del índice “m”. m tiene que ser un entero.



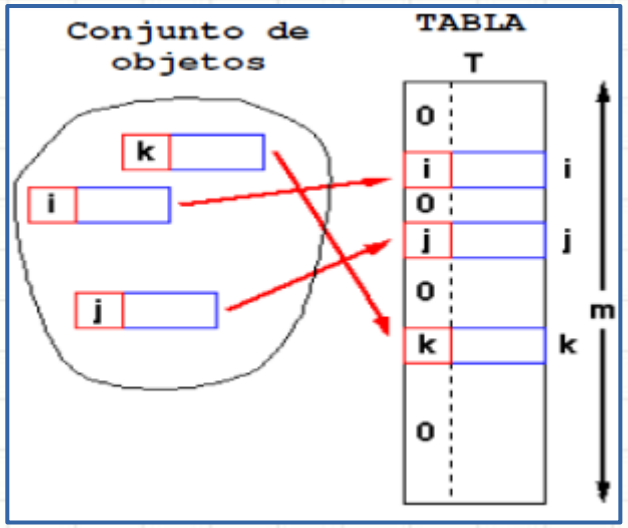
6. Búsqueda en tablas hash

Restricciones y precauciones:

1. El modelo ideal implica que la creación del índice asegure una clave única e irrepetible.
2. En el estudio del diseño de la función hash se debe conseguir un rango adecuado de índices de manera que:
   a) se minimice el número de índices repetidos.
   b) se distribuyan uniformemente los objetos.
   c) no se desperdicie espacio en la estructura de datos.



6. Búsqueda en tablas hash



3. La función tiene que ser de rápido cálculo, de otro modo se estaría elevando el tiempo de acceso.
4. El cálculo del rango tiene que tener estrecha relación con el espacio en memoria que se piensa utilizar.

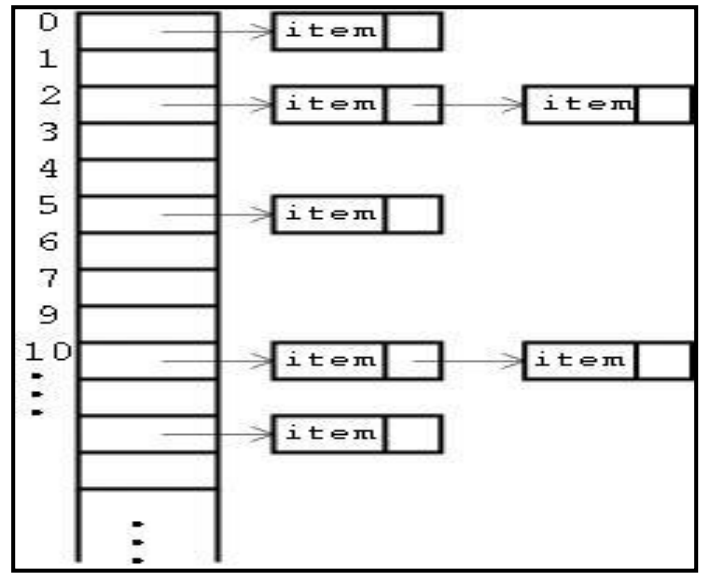
   Dado un rango [1..N] y una estructura con M lugares.

   a) Si N = M, entonces, estaríamos en la situación ideal.
   b) Si N < M, entonces, estaríamos desperdiciando espacio en memoria.
   c) Si N > M, entonces, nos estamos exponiendo a una repetición de claves. Este inconveniente se conoce como *colisión*.

6. Búsqueda en tablas hash: ejercicio

Ejemplo:

Servicios escolares necesita gestionar las listas de los alumnos de forma ordenada. Se sabe que no van a ser más de 50 y, el equipo de desarrollo de software estableció como función hash el siguiente criterio: “el valor entero de los últimos dos dígitos de la matrícula / 2”. Por ejemplo, si la matrícula de “Pedro” es 2017020230, entonces, la función hash devolverá 15.



6. Búsqueda en tablas hash: ejercicio

Considere la siguiente tabla que contiene un valor asociado a cada letra del alfabeto.

A: 1	F: 6	K: 11	P: 16	U: 21	Z: 26
B: 2	G: 7	L: 12	Q: 17	V: 22	
C: 3	H: 8	M: 13	R: 18	W: 23	
D: 4	I: 9	N: 14	S: 19	X: 24	
E: 5	J: 10	O: 15	T: 20	Y: 25	

Se le solicita escribir una función insertarTablaHash de tamaño 50. Para calcular el índice de la tabla de un nombre de una persona, se obtiene sumando valores correspondientes a cada letra de dicho nombre.

Ejemplo: Para (Samuel), el valor hash es: 19 (S) + 1 (A) + 13 (M) + 21 (U) + 5 (E) + 12 (L) = 71. Módulo de 50 = 21.

6. Referencias

1. Tenenbaum, Aaron & Langsam, Yedidyah & Augenstein, Moshe “Estructuras de Datos en C”. Prentice-Hall, México 1997.
2. Wirth, Niklaus “Algoritmos y estructura de Datos”. Prentice-Hall, México.
3. Joyanes Aguilar, Luis (1996) Fundamentos de programación, Algoritmos y Estructura de datos. McGraw-Hill, México.
4. Deitel & Deitel (2001) C++ Como programar en C/C++. Prentice Hall
5. Kerrighan y Ritchie “El lenguaje de programación”. Prentice Hall
6. Gottfried, Byron (1999) “Programación en C” McGrawHill, México.
7. Levine Gutierrez, Guillermo (1990) Introducción a la computación y a la programación estructurada. McGraw-Hill, México.
8. H. Schildt, C++ from the Ground Up, McGraw-Hill, Berkeley, CA, 1998
9. Keller,,AL;Pohl,Ira. A Book on C. 3 ª edición. Edit.Benjamin umnings.1995