Patrones Algoritmicos

Vectores, Archivos, Estructuras enlazadas

4

Algoritmos y estructura de datos se centra en el estudio de Estructuras de Datos (Archivos, arreglos, estructuras enlazadas) y el manejo de las mismas a través de diferentes algoritmos.
Para el manejo de estas estructuras desarrollaremos en este apartado los patrones algorítmicos mas usados con un enfoque conceptual, independiente de la estructura de datos para abordar luego las particularidades de cada una de ellas en un lenguaje de programación.
Como ya dijimos en la materia se trata de abordar problemas de información para lo que debemos seleccionar la estructura de dato mas adecuada y los patrones que nos permitan abordarlas con eficiencia, ya sea para cargarlas, recorrerlas, modificarlas u ordenarlas. Este apunte aborda ese tema en particular, se toma como base los patrones en arreglos ya vistos en otro documento.

Estructuras Primitivas

Formato	Efecto
SI Condición **ENTONCES** **S** **FIN_SI**	**Si la expresión toma el valor verdadero entonces se ejecuta la secuencia S si es falso ejecuta la acción siguiente a la instrucción en caso de existir.**

Formato	Efecto
SI Condición **ENTONCES** **S** **SI_NO** **R** **FIN_SI**	**Si la expresión toma el valor verdadero entonces se ejecuta la secuencia S si es falso se ejecuta la secuencia R.**

Formato	Efecto
SEGÚN expr **V1 : S1** **V2 : S2** **FIN_SEGÚN**	**Según el valor Vi de la expresión ordinal expr, se ejecuta la secuencia Si(i = 1 ..n). Cualquier valor no especificado no tiene acción asociada**

Formato	Efecto
SEGÚN expr **V1 : S1** **V2 : S2** **EN_OTRO_CASO : Sn** **FIN_SEGÚN**	**Según el valor Vi de la expresión ordinal expr, se ejecuta la secuencia Si(i = 1 ..n). Cualquier valor no especificado realiza la secuencia Sn.**

Formato	Efecto
Mientras Cond Hacer **S** **FIN_MIENTRAS**	**Ejecutar la secuencia S mientras la condición Cond tenga el valor de verdadero. Se ejecuta de 0 .. N**

Formato	Efecto
HACER **S** **MIENTRAS Cond**	**Ejecutar la secuencia S mientras que la expresión Cond sea verdadera. Se ejecuta de 1 .. N**

Formato	Efecto
PARA i [Vi..Vf] HACER **S** FIN_PARA	**Ejecutar la secuencia S (Vf – Vi + 1) veces.**

Equivalencias entre notación algorítmica y lenguajes de programación

CONDICIONAL

Formato	C-C++
SI Condicion **ENTONCES** **S** **[SI_NO** **R** **FIN_SI]**	**If (expresion)** **S;** **[else** **R;]**

Formato	C
SEGÚN expr **V1 : S1** **V2 : S2** **EN_OTRO_CASO : Sn** **FIN_SEGÚN**	**switch (selector) {** **case etiqueta:S; break;** **.....** **default: R;** **}**

ITERACION

Formato	C
Mientras Cond. Hacer **S** **FIN_MIENTRAS**	**while(expresion)** **S;**

Formato	
HACER **S** **MIENTRAS Cond**	**do** **S;** **while(expresion)**

Formto	C
PARA i [Vi..Vf] HACER **S** **FIN_PARA**	**for(i=0;i<vf;i++)** **S;**

Estilos de Indentación

A los efectos de mejorar la presentación del código se muestran diferentes estilis de indentacion, si bien no producen efecto sobre el código pueden facilitar la lectura, se recomienda definirse por un estilo y utilizar siempre el mismo

```
while( SeaVerdad() ) {
   HacerUnaCosa();
   HacerOtraCosa();
}
HacerUnaUltimaCosaMas();
```

```
while( SeaVerdad() ) {
HacerUnaCosa();
HacerOtraCosa();
}
HacerUnaUltimaCosaMas();
```

```
while( SeaVerdad() )
{
HacerUnaCosa();
HacerOtraCosa();
}
HacerUnaUltimaCosaMas();
```

```
while( SeaVerdad() )
{
   HacerUnaCosa();
   HacerOtraCosa();
}
HacerUnaUltimaCosaMas();
```

Patrones de recorrido
Recorrido de una estructura completa

Recorrer en orden natural
Patron conceptual

```
Ubicarse al comienzo de la estructura
MIENTRAS (Haya datos) HACER
   Procesar el dato actual
   Ir al siguiente
FIN_MIENTRAS
```

Según la estructura

Archivo	Arreglo	Lista

fseek(f, 0, SEEK_SET);
fread(&r,sizeof(r),1,f);
while(!feof(f)){
  //procesar r
  fread(&r, sizeof(r),1,f);
};
	i = 0;
	
	while(i<N) {
	  //procesar V[i]
	  I++;
	};
		Nodo* aux = lista;
		
		while(aux) {
		  procesar aux->info
		  aux = aux->sgte;
		};

Recorrer en orden inverso (se supone N cantidad de registros conocidos)
Patron conceptual

```
Ubicarse al final de la estructura
MIENTRAS (Haya datos) HACER
   Procesar el dato actual
   Ir al anterior
FIN_MIENTRAS
```

Según la estructura

Archivo	Arreglo	Lista

for(i=N;i>0;i--){
 fseek(f, sizeof(r)*(n-1), SEEK_SET);
  fread(&r, sizeof(r),1,f);
  //procesar r
};
	for(i=N;i>0; i--){
	
	//procesar V[i-1]
	};
		Con esta estructura no es posible si es simplemente enlazada, se requeriría puntero al anterior y eso es en listas doblemente enlazadas

Recorrido con Corte de control

La utilización de este algoritmo puntual permite resolver el análisis de una secuencia de datos que cumplen con la precondición de tener una clave que se repite, los datos están ordenados o agrupados por esta clave y se requiere información de cada subconjunto formado por todos los valores de la misma clave y además información sobre la totalidad de los datos. Debe garantizarse que se evaluaran todos los datos y que los que corresponden al mismo subgrupo serán evaluados agrupados.

El concepto de resolución es simple, se puede resolver, por ejemplo, mediante dos cicllos de repetición, uno externo que garantice que haya datos y otro interno que cumpla con la conjunción de que haya datos y que los mismos pertenezcan al mismo subgrupo

Patron conceptual

```
AGORITMO
        Leer un registro
        // hace una lectura anticipada del dato de la expresión lógica//
        MIENTRAS (Haya datos) HACER
        //garantiza la secuencia de lectura de todos los datos//
                Anterior = Registro.clave; //guarda e valor a controlar//
                MIENTRAS (haya datos y Anterior = NumeroCliente;) HACER
                //ciclo que garantiza estar en el mismo subgrupo y que aun haya datos//
                        // procesar el registro
                        Leer proximo//si es igual al anterior Continua, sino sale del ciclo.
                FIN_MIENTRAS // ciclo interno
        //acciones propias de cada subgrupo
        FIN_MIENTRAS //ciclo externo
        Acciones propias del conjunto total
FIN.
```

Archivo	Arreglo	Lista

fread(&r, sizeof(r),1,f);
while(!feof(f)){
ant = r.clave);
while(!feof(f) && ant ==r.clave){
// procesar registro
fread(&r, sizeof(r),1,f);
}
// informar por cada grupo
}
// informar sobre la totalidad
	i=0;
	while(i<N){
	ant = V[i].clave;
	while(i<N&&V[i].clave==ant)&
	//procesar V[i]
	I++;
	}
	//
	}
	//
		Nodo*aux = l;
		while(aux){
		ant = aux->info.clave
		while(aux&&aux_info.clave==ant){
		//procesar aux->info
		Aux = aux->sgte;
		}
		//
		}
		//

Como puede observarse el concepto algorítmico es el mismo, solo cambia el acceso al dato en particular se gun la estructura. En caso de ser un archivo, como en memoria hay un solo registro, primero debe leerse de modo de llevar al registro a memoria. En el caso de un arreglo, al estar TODOS los registros en memoria se debe actualizar el índice para acceder al correcto, en caso de una lista, si bien están todos en memoria debe accederse con la referencia al siguiente ya que no es una estructura indexada como el vector.

Otro patrón de recorrido interesante es el apareo.

Recorrido con apareo

Este patrón, para ser utilizado requiere, como precondición dos o mas estructuras con al menos un campo en común y ordenadas por ese campo. Obtiene como poscondicion, o el resultado del mismo es el procesamiento intercalando los datos según el campo comun

Utilizar este procedimiento si se tiene mas de una estructura con un campo clave por el que se los debe procesar intercalado y esas estructuras están ORDENADAS por ese campo común.

PRE: Estructuras a Recorrer mezclados o intercalados, deben ordenadas.

POS: Procesa la totalidad de los datos con el orden de las estructuras intercaladas

Patron conceptual

Modelo 1

```
ALGORITMO
        Procesar primer dato de uma estructura;
        Procesar primer dato de la outra estructura;

    MIENTRAS Haya datos en ambas estructuras HACER
        SI(El de la estructura 1 < que El de la estructura 2)
        ENTONCES
                Procesar Estructura 1;
                Avanzar Estructura 1;
        SINO
                Procesar Estructura 2;
                Avanzar Estructura 2;
        FIN_SI;
    FIN_MIENTRAS
    Agotar la estructura que no termino
FIN
```

Modelo 2

```
ALGORITMO
        Procesar primer dato de uma estructura;
        Procesar primer dato de la outra estructura;

    MIENTRAS Haya datos em alguna de lãs estructuras HACER
        SI(Estr. 2 termino || hay datos en ambas estructuras El de la 1<que El de la  2)
        ENTONCES
                Procesar Estructura 1;
                Avanzar Estructura 1;
        SINO
                Procesar Estructura 2;
                Avanzar Estructura 2;
        FIN_SI;
    FIN_MIENTRAS
FIN
```

Archivo	Arreglos	Listas

Modelo 1
fread(&r1, filesize(r1),1,a);
fread(&r2, filesize(r2),1,b);
while(!feof(a)&&!feof(b)){
if(r1.c < r2.c){
//procesar r1
fread(&r1, filesize(r1),1,a);
} else {
//procesar r2
fread(&r2, filesize(r2),1,b);
};
};
// agotar el que no se agoto
while (!feof(a)){
//procesar r1;
fread(&r1, filesize(r1),1,a);
};
while (!feof(b)){
//procesar r2;
fread(&r2, filesize(r2),1,b);
}; //Fin Modelo 1
 	
Modelo 2
fread(&r1, filesize(r1),1,a);
fread(&r2, filesize(r2),1,b);
while(!feof(a)||!feof(b)){
if(feof(b) || (!feof(a)&&r1.c < r2.c)){
//procesar r1
fread(&r1, filesize(r1),1,a);
} else {
//procesar r2
fread(&r2, filesize(r2),1,b);
};
};
	Modelo 1
	i= 0;
	j = 0;
	while(i<N&&j<M){
	if(V1[i].c < V2[j].c){
	//procesar Vi[i]
	i++;
	} else {
	//procesar V2[j]
	j++;
	};
	};
	// agotar el que no se agoto
	while (i<N){
	//procesar V1[i];
	i++;
	};
	while (!feof(b)){
	//procesar V2[j];
	j++;
	}; //Fin Modelo 1
	
	Modelo 2
	i=0;
	j=0;
	while(i<N||j<M){
	if(j==M || (i<N&&V1[i].c < V2[j].c)){
	//procesar V1[i]
	i++;
	} else {
	//procesar V2[j]
	j++;
	};
	};
		Modelo 1
		Nodo* p = l1;
		Nodo* q = l2;
		while(!p&&!q){
		if(p->.c < q->.c){
		//procesar p
		p=p->sgte;
		} else {
		//procesar q
		q=q->sgte;
		};
		};
		// agotar el que no se agoto
		while (!p){
		//procesar p;
		P=p->sgte;
		};
		while (!q){
		//procesar q;
		q=q_sgte;
		}; //Fin Modelo 1
		
		Modelo 2
		Nodo* p = l1;
		Nodo* q = l2;
		while(!p||!q){
		if(q || (!p&&p->.c < q->.c)){
		//procesar p
		p = p->sgte;
		} else {
		//procesar q
		q = q->sgte ;
		};
		};

Puede hacerse cualquier combinacion de estructuras, dos archivos, o dos vectores., o dos listas, o un archivo con un vecrtor, o un vector con una lista o un archivo con una lista, el concepto no cambia solo cambia el acceso a dacada dato según la estructura de la que se trate.

En el presente documento analizaremos los patrones básicos de arreglos, muchos de los cuales, como hemos visto, son conceptualmente idénticos para archivos y listas.

Acciones y funciones para vectores

BusqSecEnVector(Dato V: Tvector; Dato N: Entero; Dato Clave:Tinfo; Dato_resultado Posic: Entero): una accion

Usar este algoritmo si alguna de las otras búsquedas en vectores mas eficientes no son posibles, recordando que búsqueda directa tiene eficiencia 1, búsqueda binaria es logarítmica y búsqueda secuencial es de orden N

```
 PRE:    V: Vector en el que se debe buscar
         Clave : Valor Buscado
         N: Tamaño lógico del vector
POS:     Posic: Posición donde se encuentra la clave, -1 si no esta.
LEXICO
j : Entero;
ALGORITMO
         Posic = -1;
         j = 0; //Pone el indice en la primera posición para recorrer el vector//
         MIENTRAS (j  <  MAX_FIL  &&  j < N  &&  V[j] <> Clave) HACER
                  J++ //Incrementa el indice para avanzar en la estructura//
         FIN_MIENTRAS;
         SI (j == N)
         ENTONCES //supero el fin lógico por lo que lo buscado
            Posic = -1 // No encontró la clave buscada
         SI_NO
            Posic = j // Encontró la clave en la posición de índice j
         FIN_SI;
         Retornar Posic;
FIN. // Búsqueda secuencial En Vector
```

Controla No superar el tamaño fisico del vector j<= MAX_FIL
No leer mas alla del ultimo elemento logico cargado j <= N
Salir si es que encuentra la clave buscada V[j] <> clave

BusqMaxEnVector(Dato V: Tvector; Dato N: Entero; Dato_resultado Maximo :Tinfo; Dato_resultado Posic: Entero): una acccion

```
PRE:     V: Vector en el que se debe buscar (sin orden)
         N : Tamaño lógico del vector
POS:     Posic: Posición donde se encuentra el máximo
Maximo : Valor máximo del vector.
LEXICO
j : Entero;
ALGORITMO
         Posic = 0;
         Maximo = V[0];
         PARA j [1, N-1] HACER
                  SI (V[j] > Maximo)
                  ENTONCES
                  Posic = j;
                  Maximo = V[j];
                  FIN_SI;
         FIN_PARA;
         Retorna Posic;
FIN. // Búsqueda máximo En Vector
```

Supone que el maximo es el primer valor del vector por lo que le asigna ese valor a maximo y la posición 1 a la posición del maximo. Al haber leido solo un elemento supone ese como maximo

Recorre ahora las restantes posiciones del vector, a partir de la segunda y lcada vez que el valor leido supera al maximo contiene ese valor como maximo y el indice actual como posición del máximo, en caso que lo encontrado sea mayor, entonces reemplaza al Maximo

BusqMinDistCeroEnVector(Dato V: Tvector; Dato N: Entero; Dato_resultado Minimo :Tinfo; Dato_resultado Posic: Entero): una acccion

PRE: V: Vector en el que se debe buscar (sin orden)
N : Tamaño lógico del vector, existe al menos un valor <> de cero

POS: Posic: Posición donde se encuentra el minimo distinto de cero
Minimo : Valor minimo distinto de cero del vector.

```
LEXICO
i,j  : Entero;
ALGORITMO
        //
        J = 0;
MIENTRAS (J<N) && (V[j] == 0) HACER
        Incrementar[j];
Posic = J;
Minimo = V[j];
PARA j [Posic+1 , N-1] HACER
        SI (V[j]<> 0 && V[j] < Minimo)
           ENTONCES
           Posic = j;
           Minimo = v[j];
        FIN_SI;
Retorna Posic;
FIN_PARA;

FIN. // Búsqueda minimo distinto de cero En Vector
```

Recorre el vector hasta encontrar el primero distinto de cero. Al encontrarlo supone ese valor como minimo y el valor del indice como posición del minimo. Es decir, en esta primera parte recorre mientras que haya datos y el valor sea cero. Sale al encontrar uno distinto.

Recorre el vector desde la posición inmediata siguiente hasta la ultima desplazando el minimo solo si el valor es distinto de cero y, ademas, menor que el minimo

BusqMaxySiguienteEnVector(Dato V: Tvector; Dato N: Entero; Dato_resultado Maximo :Tinfo; Dato_resultado Posic: Entero, Dato_resultado Segundo :Tinfo; Dato_resultado PosicSegundo: Entero): una accion

PRE: V: Vector en el que se debe buscar
N : Tamaño lógico del vector mayor o gual a 2

POS: Posic: Posición donde se encuentra el máximo, PosicSegundo: Posición donde se encuentra el siguiente al máximo

Maximo : Valor máximo del vector. Segundo : Valor del siguiente al máximo del vector

LEXICO

j : Entero;

ALGORITMO

```
SI V[0] > V[1]
ENTONCES
   Posic = 0;
   Maximo = V[0];
   PosicSegund = 1;
   Segundo = V[1];
SINO
   Posic = 1;
   Maximo = V[1];
   PosicSegund =0;
   Segundo = V[0];
FIN_SI

PARA j [2, N-1] HACER
   SI (v[j] > Maximo)
   ENTONCES
      Segundo = Maximo;
      PosicSegundo = Posic;
      Posic = j;
      Maximo = v[j];
   SINO
      SI Maximo>Segundo
      ENTONCES
            Segundo = V[j];
            PosicSegundo = j
      FIN_SI
   FIN_SI;
FIN_PARA;
Retorna; // tal como se plantea los valores se pasan por parámetro variable
FIN. // Búsqueda máximo En Vector
```

Se tiene como precondicion que al menos hay dos valores. Se verifica el valor que esta en la primera posición y se lo compara con el que esta en la segunda posición, en el caso de ser mayor, el maximo es ese valor, posición del máximo es uno, el segundo el valor que esta en segundo lugar y posición del segundo es 2. En caso contrario se establece como maximo el valor de la segunda posición y segundo el de la primera.

Se verifica luego desde la tercera posición hasta el final. En el caso que el nuevo valor sea mayor que el maximo, se debe contener el anterior maximo en el segundo y al maximo se le asigna el nuevo valor. Cosa similar hay que hacer con las posiciones. Si esto no ocurriera se debe verificar si el nuevo valor supera al segundo, en ese caso debera desplazarlo

CargaSinRepetirEnVectorV1(Dato_Resultado V: Tvector; Dato_Resultado N: Entero; Dato Clave:Tinfo; Dato_resultado Posic: Entero; Dato_resultado Enc : Booleano): una accion

Utilizar este algoritmo si la cantidad de claves diferentes es fija, se dispone de memoria suficiente como para almacenar el vector y la clave no es posicional(es decir clave e índice no se corresponden directamente con posición única y predecible.Si la posición fuera unica y predecible la busqueda debe ser directa

PRE: V: Vector en el que se debe buscar
Clave : Valor Buscado
N : Tamaño lógico del vector

POS: Posic: Posición donde se encuentra la clave, o donde lo inserta si no esta. Retorna 0 (cero) en caso que el vector esta completo y no lo encuentra
Enc : Retorna True si estaba y False si lo inserto con esta invocación
Carga vector sin orden

```
LEXICO
j : Entero;
ALGORITMO/
  Posic = 0;
  J = 1;
  MIENTRAS (j  <  MAX_FIL  y  j < N  y  V[j] <> Clave) HACER
        J++
  FIN_MIENTRAS;

  SI( j == MAX_FIL)
  ENTONCES
        Retorna -1
  SI_NO
        Posic = j:


        SI (j== N)
        ENTONCES
              Enc =FALSE; // No encontró la clave buscada
              Inc(N);
              V[N] = Clave;
        SI_NO
              Enc = True // Encontró la clave en la posición de índice j
        FIN_SI;
FIN_SI
FIN. // Carga sin repetir en vector
```

Controla No superar el tamaño fisico del vector j<= MAX_FIL
No leer mas alla del ultimo elemento logico cargado j <= N
Salir si es que encuentra la clave buscada V[j] <> clave

Si debio superar el tamaño fisico maximo del vector no pudo cargarlo y retorna -1 como señal de error

Si encontro un dato o lo debe cargar esto es en el indice j por lo que a posic se asigna ese valor. En el caso que j sea igual a n significa que recorrio los n elemntos cargados del vector, tiene estpacio por lo que debe cargar el nuevo en la posición j. En este caso, y al haber un elemento nuevo debe incrementar n que es el identificador que controla el tamaño logico del vector

BusquedaBinariaEnVectorV1(Dato V: Tvector; Dato N: Entero; Dato Clave:Tinfo; Dato_resultado Posic: Entero; Dato_resultado Pri : Entero): una accion

Utilizar este algoritmo si los datos en el vector están ordenados por un campo clave y se busca por ese campo. Debe tenerse en cuenta que si la clave es posicional se deberá utilizar búsqueda directa ya que la diferencia en eficiencia esta dada entre 1, para la búsqueda directa y log_2N para la binaria

PRE: V: Vector en el que se debe buscar con clave sin repetir
Clave : Valor Buscado
N : Tamaño lógico del vector

POS: Posic: Posición donde se encuentra la clave, o -1 si no esta, no carga ordenado
Pri : Retorna la posición del limite inferior

```
LEXICO
j : Entero;
u,m : Entero;
ALGORITMO
        Posic = -1;
        Pri = 0;
        U = N-1;
        MIENTRAS (Pri < = U  y  Pos == -1) HACER
                M = (Pri + U ) div 2
                SI V[M] = Clave
                ENTONCES
                        Retorna M;
                SI_NO
                        SI Clave > V[M]
                        ENTONCES
                                Pri = M+1
                        SI_NO
                                U = M – 1
                        FIN_SI
                FIN_SI
        FIN_MIENTRAS;
Retorna Pos;
FIN. // Búsqueda binaria en vector
```

Establece valores para las posiciones de los elementos del vector, Pri contiene el indice del primero, es decir el valor 0, U el indice del ultimo elemento logico, es decir N-1. Ademas se coloca en Posic en el valor -1, utilizando este valor como bandera para salir del ciclo cuando encuentar el valor buscado

Permanece en el ciclo mientras no encuentre lo buscado, al encontrarlo le asigna a pos el indice donde lo encontro, como es un valor > que cero hace false la expresión logica y sale del ciclo. Si no lo encuentra, y para evirtar ciclo infinito verifica que el primero no tome un valor mayor que el ultimo. Si eso ocurre es que el dato buscado no esta y se debe salir

Si el dato buscado lo encuentra le asigna a posición el indice para salir. Si no lo encuentra verifica si esmayor el dato buscabo a lo que se encuentra revisa en la mitad de los mayores por lo que le asigna al primero el indice siguiente al de la mitad dado que alli no estab y vuelve a dividir el conjunto de datos en la mitas, de ser menor pone como tome ultimo el anterior al de la mitad actual

BusquedaBinariaEnVectorV2(Dato V: Tvector; Dato N: Entero; Dato Clave:Tinfo; Dato_resultado Posic: Entero; Dato_resultado Pri : Entero): una acccion

PRE: V: Vector en el que se debe buscar clave puede estar repetida
Clave : Valor Buscado
N : Tamaño lógico del vector

POS: Posic: Posición donde se encuentra la primera ocurrencia de la clave.
0 (cero) si no esta.
Pri : Retorna la posición del limite inferior

```
LEXICO
j : Entero;
u,m : Entero;
ALGORITMO
        Posic = -1;
        Pri = 0;
        U = N-1;
        MIENTRAS (Pri <  U ) HACER
                M = (Pri + U ) div 2
                SI V[M] = Clave
                ENTONCES
                        Posic = M;
                        Pri = M;
                SI_NO
                        SI Clave > V[M]
                        ENTONCES
                                Pri = M+1
                        SI_NO
                                U = M – 1
                        FIN_SI
                FIN_SI
        FIN_MIENTRAS;
        Retorna Pos;
FIN. // Búsqueda binaria en vector
```

La busqueda es bastante parecida a lo desarrollado anteriormente, pero en pos debe tener la primera aparicion de la clave buscada que puede repetirse.

En la busqueda anterior utilizabamos esta pos como bandera, para saber cuando Sali si lo encontro. En este caso si lo utilizamos con el mismo proposito saldria cuando encuentra un valor oincidente con la clave que no necesariamente es el primero, por lo que esa condicion se elimina. Al encontrarlo en m se le asigna ese valoe a pos, alli seguro esta. No sabemos si mas arriba vuelve a estar por lo que se asigna tambien esa posición al ultimo para seguir iterando y ver si lo vuelve a encontrar. Debe modificarse el operador de relacion que compara primero con ultimo para evitar un ciclo infinito, esto se hace eliminando la relacion por igual. Insisto en el concepto de los valores de retorno de la busqueda binaria. Una particularidad de los datos es que si lo que se busca no esta puede retornal en el primero la posición donde esa clave deberia estar

CargaSinRepetirEnVectorV2(Dato_Resultado V: Tvector; Dato_Resultado N: Entero; Dato Clave:Tinfo; Dato_resultado Posic: Entero; Dato_resultado Enc : Booleano): una acccion

PRE: V: Vector en el que se debe buscar ordenado por clave
Clave : Valor Buscado
N : Tamaño lógico del vector

Una particularidad de los datos en la búsqueda binaria es que si lo que se busca no esta, el valor del primero (pri) es donde debería estar en caso que desearamos mantener el orden.

POS: Posic: Posición donde se encuentra la clave, o donde lo inserta si no esta. Retorna -1 en caso que el vector esta completo y no lo encuentra
Enc : Retorna True si estaba y False si lo inserto con esta invocación
Carga vector Ordenado

```
LEXICO
j : Entero;
ALGORITMO
        Enc = True;
        BusquedaBinariaEnVectorV(V; N; Clave; Posic; Pri)
        SI (Posic == -1)
        ENTONCES
                Enc = False ;
                Posic = Pri;
                PARA j [N-1, Pri](-) HACER
                        V[j+1] = V[j];
                FIN_PARA;
                V[Pri] = Clave;
                N++;
        FIN_SI
FIN. // Carga sin repetir en vector Versión 2. con vector ordenado
```

Al estar el vector ordenadola busqueda puede ser binaria, si lo encuentra retorna en posición un valor mayor a cero. Si no lo encuentra el valor de posición sera cero. En este caso, se conoce que en pri es en la posición donde este valor debe estar

Se produce un desplazamiento de los valores desde el ultimo hasta el valor de pri corriendolos un lugar para poder insertar en la posición pri el nuevo valos. Al pasar por aquí se inserto un nuevo elemento por lo que n, que contiene la cantidad de elementos del vector debe incrementarse en uno

OrdenarVectorBurbuja(Dato_Resultado V: Tvector; Dato N: Entero): una acccion

Pre: V: Vector en el que se debe ordenar, se supone dato simple
N : Tamaño lógico del vector

POS: Vector ordenado por clave creciente

Usar este algoritmo cuando los datos contenidos en un vector deben ser ordenados. Se podría por ejemplo cargar los datos de un archivo al vector, ordenar el vector recorrerlo y generar la estructura ordenada. Para esto la cantidad de elementos del archivo debe ser conocida y se debe disponer de memoria suficiente como para almacenar los datos. Una alternativa, si la memoria no alcanza para almacenar todos los datos podría ser guardar la clave de ordenamiento y la referencia donde encontrar los datos, por ejemplo, la posición en el archivo.

```
LEXICO
I,J, : Entero;
Aux : Tinfo;
ALGORITMO
        PARA i [0, N - 1] HACER
                PARA j [1, N - i] HACER
                  SI (v[j-1] > v[j])
                    ENTONCES
                      Aux = v[j-1];
                      V[j-1] = v[j];
                      V[j ] = Aux;
                 FIN_SI;
                FIN_PARA;
        FIN_PARA;
FIN
```

La idea general es ir desarrollando pasos sucesivos en cada uno de los cuales ir dejando el mayor de los elementos en el último lugar. En el primer paso se coloca el mayor en la ultima posición, en el paso siguiente se coloca el que le sigue sobre ese y asi hasta que queden dos elemntos. En ese caso al acomodar el segundo el otro queda acomodado el primer ciclo cuenta los pasos, son uno menos que la cantidad de elementos porque el ultimo paso permite acomodar 2 elementos, por eso el ciclo se hace entre 1 y N –1 siendo N la cantidad de elemento y teniendo en cuenta que en C la pos del ultimo es N-1

Para poder colocar el mayor al final es necesario hacer comparaciones. Se compara el primero con el segundo y si corresponde se intercambian. Asi hasta llegar al ante ultimo eleemento que se lo compara con el últim.

Al ir recorriendo los distintos pasos, y dado que en cada uno se acomoda un nuevo elemento corresponde hacer una comparación menos en cada avance, como los pasos los recorremos con i, las comparaciones seran n – i. disminuye en 1 en cada paso

CorteDeControlEnVector(Dato V:Tvector; Dato N: Entero): una acccion
Usar este procedimiento solo si se tienen los datos agrupados por una clave común y se requiere procesar emitiendo información por cada subconjunto correspondiente a cada clave.
PRE: V: Vector en el que se debe Recorrer con corte de control
Debe tener un elemento que se repite y estar agrupado por el.
N : Tamaño lógico del vector
POS: Recorre agrupando por una clave

```
LEXICO
I : Entero;

ALGORITMO
        I = 0;
        Anterior = TipoInfo;
        // Inicializar contadores generales
      MIENTRAS (I<N) HACER
          //inicializar contadores de cada sublote
         Anterior = V[i]
          MIENTRAS (I<N Y Anterior == V[i] HACER
                // Ejecutar acciones del ciclo

                I = I+1 // avanza a la siguiente posición
          FIN_MIENTRAS
         // Mostrar resultados del sublote
      FIN_MIENTRAS
      // Mostrar resultados generales
FIN
```

CargaNMejoresEnVector(Dato_Resultado V: Tvector; Dato_Resultado N: Entero; Dato Clave:Tinfo): una acccion

PRE: V: Vector en el que se debe buscar e insertar los mejores
Clave: Valor Buscado
N: Tamaño lógico del vector

POS: Vector con los N mejores sin orden

LEXICO

```
j : Entero;
Minimo: Tinfo;
Posic: Entero;
ALGORITMO
        SI (N < MAX-FIL)
        ENTONCES
                N++;
                V[n] = Clave
        SI_NO
        BusqMinEnVector(V;  N; Minimo :Tinfo; Posic: Entero);
                SI (Clave > Minimo)
                ENTONCES
                        V[Posic] = Clave;
                FIN_SI;
        FIN_SI;
FIN. // Carga los N mejores en vector
```

En caso de no haberse completado el vector el valor nuevo se carga. De lo contrario se busca el minimo en el vector y su posición, si el nuevo valor es mayor que el minimo en la posición de este se agrega el valor nuevo desplazando ese minimo. En caso que el nuevo valor no sea mayor que el minimo este no se cambia

Implementaciones en C++

```
#include <iostream>
using namespace std;

//*********************************************************************/
/*                          Busqueda Secuencial                        */
/*********************************************************************/

//retorna el índice (-1 si no esta)
int BusquedaSec(int Vector[],int Valor){
        for (int i=0; i<max;i++)
         if (*Vector++==Valor)
                return i;
        return -1;
}

/*********************************************************************/
/*                          Busqueda de un Máximo                      */
/*********************************************************************/

int BusquedaMax(int Vector [],int& max, int N){
        int Posicion=0;             //indice del máximo
        max = Vector[0];
        for (int i=1; i<N;i++)
                if (Vector[i]> max){
                   Posicion=i;
                   max = Vector[i];
                };
        return Posicion;
}
//*********************************************************************/
/*                          Busqueda Máxima y siguiente (al menos hay 2 valores)
*/
/*********************************************************************/

void BusquedaMaxSig(int Vector [],int& posMax,int& posSig, int N ){
        int Max=Vector[0], Sig=Vector[1];          //Supongo que el primero es mayor que el segundo
        posMax = 0; posSig = 1;
        if (Sig > Max){                            //Si no fuera los cambio
           Max=Vector[1]; Sig=Vector[0];posMax = 1; posSig =01;
        }
        for (int i=1; i<N;i++){
                if (Vector[i]>max){
                        posSig = posMax; posMax = I;
                        Sig = Max; Max = Vextor[i];
                } else if(Vector[i] > Sig){
                                Sig = Vector[i]; posSig = I;}

        }

        return ;
}
```

```
/*********************************************************************/
/*                          Busqueda Minima distinto de cero                                 */
/*********************************************************************/

int BusquedaMinDig(int Vector[],int max){
        int min, posMin=0;                          //posicion del minimo distinto de 0
        int j=0;
        while (j<max && Vector[j]==0) // recorrer los eventuales ceros
                j++;
        posMin=j;
        min=Vector[posMin];
        for (j=posMin+1;j<max;j++)
                if(Vector[j]<min){posMin=j;, min = Vector[j];
        return PosMin;
}
```

Patrones según las estructuras

Dos conceptos: estructuras indexadas y memoria. Memoria, ya vimos implica almacenamiento electrónico, lo que significa procesamiento rápido, por lo que, salvo que se requiera que el dato persista mas alla de la aplicación, su uso es el recomentado. Indexada permite acceder directamente a un miembro en particular mediante un índice, esto posibilita el acceso directo, el cual significa Efciencia de orden 1, el acceso más eficiente. En memoria las estructuras con las que trabajamos son Arreglos y Estructutas enlazadas, por otro lado, estructuras que permiten acceso directo son arreglos y archivos binarios.
Modificacion de los patrones según las estructuras.

Modificar un dato

Haciendo análisis comparativo de estructuras que manejan conjunto de datos homogéneos, veremos la ventaja de la utilización de estructuras indexadas.
En el caso de modificar un datodebe cumplirse, conceptualmente, con las siguientes acciones:

1. **Apuntar**: llegar hasta donde esta el dato en cuestión
2. **Leer**: Llevarlo a memoria en caso que no este allí
3. **Modificar**: realizar los cambios que se requieran en el dato que esta en memoria.
4. **Volver a apuntar**: en caso que la lectura haya producido un desplazamiento del puntero
5. **Grabar**: en caso que el dato en cuestión este en un dispositivo externo.

Veamos la diferencia según las etructuras. Supongamos tener eu archivo, un array y una lista, todoas de registros y el propósito es modidicar el calmpo C1 del cuarto registro.

Acción	Array	Archivo	Lista
1	V[3].C1 = XX;	fseek(F, sizeof(r)*4, SEEK_SET)	Aux = L; for(i=0;i<3;i++) uux=aux->sgte;
2		fread(&R, sizeof(R),1,F)	Aux->info.C1 = XX;
3		R.C1 = XX	
4		fseek(F,-sizeof(R), SEEK_CUR)	
5		Fwrite(&R,sizeo(R),1,F)	

Se puede observer que, en el caso del archivo se deben hacer todos los pasos por separados, Apuntar al registro pedido, llevar lo que esta en el disco a memoria, modificar el campo del refistro que esta en memoria, retroceder un registro el puntero y grabar el dato en memoria en le disco. En el caso de la losta, si bien ya están todos los registros en memoria, es necesario recorrer secuencialmente la estructura hasta alcanzar el requerido, una vez alcanzado se lo debe modificar a ese que esta en memoria y no se requiere retroceder el puntero ni llevar el registro a otro dispositivo. En el caso del array todas las operaciones se hacen simultaneamenta, ya que están todos los registros en memoria y se accede a uno en particular tan solo con el índice, apuntando a este es posible modificarlo y en este caso tampoco el puntero se desplaza.

Recorrer la estructura completa

En orden natural

Archivo	Arreglo	Lista

fseek(f, 0, SEEK_SET);
fread(&r,sizeof(r),1,f);
while(!feof(f)){
  //procesar r
  fread(&r, sizeof(r),1,f);};
	i = 0;
	
	while(i<N) {
	  //procesar V[i]
	  I++;};
		Nodo* aux = lista;
		
		while(aux) {
		  procesar aux->info
		  aux = aux->sgte;};

Recorrer en orden inverso (se supone N cantidad de registros conocidos)

Archivo	Arreglo	Lista

for(i=N;i>0;i--){
 fseek(f, sizeof(r)*(n-1), SEEK_SET);
  fread(&r, sizeof(r),1,f);
  //procesar r
};
	for(i=N;i>0; i--){
	
	//procesar V[i-1]
	};
		Con esta estructura no es posible si es simplemente enlazada, se requeriría listas doblemente enlazadas

Recorrido con Corte de control

Archivo	Arreglo	Lista

fread(&r, sizeof(r),1,f);
while(!feof(f)){
  ant = r.clave);
  while(!feof(f) && ant ==r.clave){
  // procesar registro
  fread(&r, sizeof(r),1,f);
  }
  // informar por cada grupo
}
// informar sobre la totalidad
	i=0;
	while(i<N){
	  ant = V[i].clave;
	  while(i<N&&V[i].clave==ant)&
	  //procesar V[i]
	  I++;
	  }
	//
	}
	//
		Nodo*aux = l;
		while(aux){
		  ant = aux->info.clave
		  while(aux&&aux_info.clave==ant){
		  //procesar aux->info
		  Aux = aux->sgte;
		  }
		 //
		}
		//

Recorrido con Apareo

Archivo	Arreglos	Listas

Modelo 1
fread(&r1, filesize(r1),1,a);
fread(&r2, filesize(r2),1,b);
while(!feof(a)&&!feof(b)){
 if(r1.c < r2.c){
  //procesar r1
  fread(&r1, filesize(r1),1,a);
  } else {
  //procesar r2
  fread(&r2, filesize(r2),1,b);
  };
};
// agotar el que no se agoto
while (!feof(a)){
  //procesar r1;
  fread(&r1, filesize(r1),1,a);
};
while (!feof(b)){
  //procesar r2;
  fread(&r2, filesize(r2),1,b);
}; //Fin Modelo 1
 	
Modelo 2
fread(&r1, filesize(r1),1,a);
fread(&r2, filesize(r2),1,b);
while(!feof(a)||!feof(b)){
 if(feof(b) || (!feof(a)&&r1.c < r2.c)){
  //procesar r1
  fread(&r1, filesize(r1),1,a);
  } else {
  //procesar r2
  fread(&r2, filesize(r2),1,b);
  };
};
	Modelo 1
	i= 0;
	j = 0;
	while(i<N&&j<M){
	 if(V1[i].c < V2[j].c){
	  //procesar Vi[i]
	  i++;
	  } else {
	  //procesar V2[j]
	  j++;
	  };
	};
	// agotar el que no se agoto
	while (i<N){
	  //procesar V1[i];
	  i++;
	};
	while (!feof(b)){
	  //procesar V2[j];
	  j++;
	}; //Fin Modelo 1
	
	Modelo 2
	i=0;
	j=0;
	while(i<N||j<M){
	 if(j==M || (i<N&&V1[i].c < V2[j].c)){
	  //procesar V1[i]
	  i++;
	  } else {
	  //procesar V2[j]
	  j++;
	  };
	};
		Modelo 1
		Nodo* p = l1;
		Nodo* q = l2;
		while(!p&&!q){
		 if(p->.c < q->.c){
		  //procesar p
		  p=p->sgte;
		  } else {
		  //procesar q
		  q=q->sgte;
		  };
		};
		// agotar el que no se agoto
		while (!p){
		  //procesar p;
		  P=p->sgte;
		};
		while (!q){
		  //procesar q;
		  q=q_sgte;
		}; //Fin Modelo 1
		
		Modelo 2
		Nodo* p = l1;
		Nodo* q = l2;
		while(!p||!q){
		 if(q || (!p&&p->.c < q->.c)){
		  //procesar p
		  p = p->sgte;
		  } else {
		  //procesar q
		  q = q->sgte ;
		  };
		};

Puede hacerse cualquier combinacion de estructuras, dos archivos, o dos vectores., o dos listas, o un archivo con un vecrtor, o un vector con una lista o un archivo con una lista, el concepto no cambia solo cambia el acceso a dacada dato según la estructura de la que se trate.

En el presente documento analizaremos los patrones básicos de arreglos, muchos de los cuales, como hemos visto, son conceptualmente idénticos para archivos y listas.

Cantidad de registros

Conocer la cantidad de elementos que tiene una estructura requiere un recorrido en arreglos y listas pero no requiere recorrido en caso de un archivo.

En el caso de los archivos de acceso directo con registros de tamaño fijo esto puede resolverse utilizando adecuadamente fseek y ftell.

Archivo	Arreglo	Lista

fseek(f, 0, SEEK_END);
 	
// el flujo es F
// y el registro R
 	
i = ftell(F/sizeof(R);
	i = 0;
	while(V[i] != marca) {
	I++;};
	//se utilize marca como centinela
	// i contiene cantidad de registros
		Nodo* aux = lista;
		I = 0;
		while(aux) {
		i++;
		aux = aux->sgte;};
		//i contiene cantidad de nodos

Busqueda secuencial

Esta búsqueda no es recomendable en archivos por el almacenamiento físico de los datos, lo que la hace altamente ineficiente. Si es posible en arreglos también en listas, en este caso es la única búsqueda posible. Esta búsqueda tiene una eficiencia de orden N, siendo N la cantiad de registros.

Archivo	Arreglo	Lista

fseek(f, 0, SEEK_SET);
fread(&r,sizeof(r),1,f);
enc = false;
while(!feof(F)&&!enc){
if(R.c1==buscado)
enc = true;
else
fread(&r, sizeof(r),1,f);};
	i = 0;
	
	enc=false
	while(i<N&&!enc) {
	if(V[i].c1 == buscado
	enc = true;
	else
	I++;};
		Nodo* aux = lista;
		
		enc = false;
		while(aux&&!enc) {
		if(aux->info == buscado)
		enc = true;
		else
		aux = aux->sgte;};

Busqueda directa

Es posible en arreglos y archivos siempre se la posición del registro a buscar sea única y predecible. Esta búsqueda no es posile el estructuras enlazadas, ya que para ello debería conocerse la dirección de memoria, conociendo esto si podría accederse directamente para trabajar con el nodo. Supongamos querer encontrar el registro de la posición X. la efciencia de esta búsqueda es de orden 1 y es la mas recomendada.

Archivo	Arreglo

fseek(f, (X-1)*sizeof(R), SEEK_SET);
 	
fread(&r,sizeof(r),1,f);
	Vector[X-1]

Busqueda Binaria

La eficiencia es logarítmica y no es posible en lastas, algo equivalente se puede hacer con arboles.

Archivo	Arreglo

Pos = -1;
Pri = 0; U = *Cantidad registros*-1;
while(Pri<=U &&pos==-1){
M = (Pri + U) / 2
fseek(F,(M-1)*sizeof(R),SEEK_SET);
fread(&R, sizeof(R),F);
if (R.C1 == buscado) Return M;
else
if(buscado > R.C1)
Pri = M+1
else
U = M – 1
}
}
return -1;
}
	Pos = -1;
	Pri = 0; U = N-1;
	while(Pri<=U &&pos==-1){
	M = (Pri + U) / 2
	
	if (V[M].C1 == buscado) return M;
	else
	if(buscado > V[M].C1)
	Pri = M+1
	else
	U = M – 1
	}
	}
	return -1;
	}

Puede observarse que los patrones que son comunes son conceptualmente idénticos y solo cambia el acceso al dato particilar, algunos patrones no son aplicable a determinadas estructuras y no todos están resueltos en este documento.
Señalo a continuación los patrones aplicables a cada estructura, y dejo como tarea que implementen TODOS en C, o C++.

Patron	Archivo	Array	Lista
Recorrido en orden natural	SI	SI	SI
Recorrido en orden inverso	SI	SI	NO
Recorrido con corte de control	SI	SI	SI
Recorrido con apareo	SI	SI	SI
Insertar delante	NO	SI	SI
Insertar en medio	NO	SI	SI
Insertar al final	SI	SI	SI
Insertar sin repetir clave	NO	SI	SI
Cargar los N mejores	NO	SI	SI
Ordenar con PUP	SI	SI	NO
Metodo de ordenamiento	NO	SI	SI
Modificar un registro	SI	SI	SI
Busqueda directa	SI	SI	NO
Búsqueda binaria	SI	SI	NO
Búsqueda secuencial	SI	SI	SI
Búsqueda de máximo/minimo	SI	SI	SI
Busqueda de max/min y el sgte	SI	SI	SI
Busqueda de minimo distinto de cero	SI	SI	SI

1. RECORRIDO EN ORDEN INVERSO EN LISTAS SOLO ES POSIBLE SI SON DOBLEMENTE ENLAZADAS
2. INSERTAR AL FINAL EN ARREGLO SOLO SI NO SE SUPERA EL TAMAÑO FISICO DEL MISMO
3. EL PROCEDIMIENTO DE INSERTAR EN UNA LISTA LO HACE CON ORDEN
4. LA BUSQUEDA DIRECTA EN UNA LISTA SOLO ES POSIBLE CONOCIENDO LA DIRECCION DE MEMORIA
5. BUSQUEDA BINARIA NO APLICABLE A LISTAS, EL CONCEPTO ES APLICABLE A ARBOL DE BUSQUEDA
6. BUSQUEDA SEC. EN ARCHIVOS SI BIEN PUEDE HACERSE SU USO NO ES RECOMENDABLE
7. BUSQUEDA SECUENCIAL EN LISTAS ES LA UNICA BUSQUEDA QUE ESTA ESTRUCTURA SOPORTA.

NOTA: para el análisis de las particularidades de cada estructura y la profundización de TODOS los patrones para las mismas ver el documento correspondiente a cada una de ellas