



## Unidad 09 Patrones algorítmicos

9

### Contenidos analíticos

#### Parte III: Estructura de datos estáticos y almacenamiento lógico

Patrones de carga: en la definición. Secuencial, directa, ordenada  
Patrones de recorrido: total, parcial, en ambas direcciones. Con corte de control, apareando dos o mas estructuras  
Patrones de búsqueda: directa, binaria, lineal. Análisis de eficiencia  
Patrones de ordenamiento: Con posición Única predecible, mediante métodos de ordenamiento.  
Análisis de eficiencia

Se desarrollan algunos patrones algorítmicos para datos simples, secuencias, estructuras lógicas y físicas.

El desarrollo es conceptual quedando para cada uno de ustedes adaptarlos según las características propias de cada caso.

A los efectos de una auto evaluación se recomienda en caso que sea posible generalizar, según corresponda según tipo de dato o criterio de seleccion

Los patrones de estructuras enlazadas se profundizan cuando abordemos esa unidad

Datos simples	
Dados dos identificadores intercambiar valores	Dados tres valores mostrarlos ordenados
<pre>void intercambio (int &amp;a, int &amp;b) {     // se asumen valores diferentes      int c = a;     a = b;     b = c;     return; }</pre> <p>Generalice para escalares</p>	<pre>void ordenar3(int &amp;mr, int &amp;md, int &amp;mn) {     /* poner en mr el mayor, mn el menor y en     md el medio. Los tres valores son diferentes*/     if (mr&lt;md) intercambio(mr,md);     if(mr&lt;mn) intercambio(mr, mn);     if(md&lt;mn) intercambio(md, mn);     return; }</pre> <p>Generalice para escalares</p>
Dados dos valores retornar el mayor	datos tres valores retornar el mayor
<pre>int mayorDeDos(int a, int b ) {     return a&gt;b?a:b; }</pre> <p>//plantee otra solución</p>	<pre>Int mayorDeTres(int a; int b; int c) {     int aux = mayorDeDos(a,b);     return mayorDeDos(aux, c); }</pre> <p>//plantee alternativas</p>



buscar el máximo en un conjunto de 10 numeros enteros		
Alternativa 1 valores > 0	Alternativa 2 sin restricción	Cual se le ocurre?
<pre>int maximo() {     int v, myr = 0, i;     for(i=0; i&lt;10; i++) {         cin&gt;&gt;v;         if(v&gt;myr) myr= v;     }     return myr; }</pre>	<pre>int máximo() {     Int v, myr, i;     for(i=0, i&lt;10; i++){         cin&gt;&gt;v;         if(i==0   v&gt;myr) myr= v;     }     return myr; }</pre>	Como imagina, por ejemplo una lectura anticipada asignándole ese primer valor al mayor y luego avanzar con el resto de las lecturas haciendo las evaluaciones? Obtendría el resultado correctamente?
<p>En la propuesta anterior que cambios propone si:</p> <p>El tipo de dato es float. Se puede generalizar por tipo de dato?</p> <p>El lote evalua N valores y N es un parámetro</p> <p>Para buscar un mínimo se puede generalizar por tipo de dato y criterio de selección?</p> <p>El lote, para cualquiera de las opciones no es conocido a priori</p> <p>Buscar el máximo o mínimo y la posición relativa del mismo dentro del conjunto de valores</p> <p>Buscar el máximo o mínimo y el siguiente</p> <p>Idem indicando posición relativa de ambos</p> <p>Buscar máximo o mínimo y los dos siguientes</p> <p>Idem indicando posición relativa de los tres valores</p> <p>Dado un conjunto de float&lt;&gt; de cero buscar el máximo de los negativos y el mínimo positivo</p>		
Piense situaciones problemáticas que requieran utilización de estos patrones de datos simples		
Con struct → suponga struct tr { int c1, char c2[10]}		
Función que cargue los datos de un registro	Función que muestre una struct	
<pre>tr crearReg(int a, char s[]) {     tr r;     r.c1 = a;     strcpy(r.c2,s);     return r; }</pre>	<pre>void mostrarReg(tr r) {     cout&lt;&lt;r.c1&lt;&lt;" "&lt;&lt;r.c2&lt;&lt;endl; }  /* la asignación externa de entrada y salida en una struct en miembro a miembro</pre>	
Estructuras de colección de datos homogéneos		
Vector	Archivo	Lista
Carga con asignación interna de estructuras con struct		
<pre>void cargar(tr v[], int N) {     int i=0;     int a;     char s[N];     tr r;     while(i&lt;N &amp;&amp; la que sugiera){         cin&gt;&gt;a;         cin &gt;&gt; s;         v[i]=creaeReg(a,s); i++;     } }</pre>	<pre>void cargar( FILE* f) {     int a;     char s[N];     tr r;     while( la que sugiera){         cin&gt;&gt;a;         cin &gt;&gt; s;         fwrite(&amp;r, sizeof(r),1, f);     } }</pre>	<pre>void cargar( Nodo* l) {     int a;     char s[N];     tr r;     while( la que sugiera){         cin&gt;&gt;a;         cin &gt;&gt; s;         insertarOrdenado(l,r);     } }</pre>



Mostrar todos los registros por el dispositivo externo de salida recorriendo del primero al ultimo		
<pre>void mostrarNat(tr v[], int N) {     int i=0;     while(i&lt;N ){         mostrarReg(v[i]);         i++;     } }</pre>	<pre>void mostrarNat(FILE * f) {     tr r;     while(fread(&amp;r, sizeof(r),1,f ){         mostrarReg(r);     } }</pre>	<pre>void mostrarNat(Nodo *&amp; l) {     tr r;     while(l!=NULL){         r = pop(l);         mostrarReg®;     } }</pre>
Mostrar todos los registros en orden inverso		
<pre>void mostrarInv(tr v[], int N) {     int u = N-1;      for( int i = u; i &gt;= 0; i--){          mostrarReg(v[i]);     } }</pre>	<pre>void mostrarInv(FILE* f) {     int u = cantReg(f);     tr r;     for( int i = u; i &gt;= 0; i--){         fseek(f, sizeo(r)*i, SEEK_SET);         fread(&amp;r, sizeof( r), 1, f );         mostrarReg(r);     } }</pre>	<pre>void mostrarInv(Nodo* l) {     Nodo* p = NULL;     tr r;     while (l)         push(p, pop(l));     while (p){         r = pop(p);         mostrarReg(r);     } }</pre>
Recorrido con corte de control		
Vector	Archivo	
<pre>void corteControl(tr v[], int N) {      int i = 0;     int control;     while(i&lt;N){         control = v[i].c1;         cout&lt;&lt; control&lt;&lt;endl;         while(l&lt;N &amp;&amp; control == v[i].c1){             cout&lt;&lt; v[i].c2;             .....             l++;         }         Mostrar datos del subconjunto     }     Mostrar datos totales }</pre>	<pre>void corteControl(FILE * F) {     tr r     fread(&amp;r, sizeof(r), 1,f);     int control ;     while(!feof(f)){         control = r.c1;         cout&lt;&lt; control&lt;&lt;endl;         while(!feof(f) &amp;&amp; control == v[i].c1){             cout&lt;&lt; r.c2;             .....             Fread(&amp;r, sizeof(r), 1,f);         }         Mostrar datos del subconjunto     }     Mostrar datos totales }</pre>	
<p>El corte de control requiere al menos un campo que se repite y arma subconjuntos con los registros que pertenecen al mismo. Se produce un corte al leer un valor distinto y se muestran los datos del nuevo sub grupo. Este patron NO ordena, muestra sin repetir el campo que agrupa. Cree que se puede aplicar este modelo con listas? Puede generalizar el patron?</p> <p>Que pasa si cada subconjunto tiene en el mismo otro campo que se repite. Se puede hacer un doble corte de control? Se puede resolver con un modelo diferente al propuesto?</p>		



Apareo (enumere las precondiciones que deben tenerse en cuenta)	
Vector	Archivo
<pre>void apareo(tr v1[], int N, tr v2[], int M) {     int i = 0;     int j = 0;     while(i&lt;N&amp;&amp;j&lt;M){         if(v1[i].c1&lt;v2[j].c1){             procesar v1[i];             i ++;         }         else{             procesar v2[j];             j++;         }     }     while(i&lt;N){         procesar v1[i];         i++;     }     while(j&lt;M){         procesar v2[j];         j++;     }     return; }</pre> <p>Otro modelo</p> <pre>void apareo(tr v1[], int N, tr v2[], int M) {     int i = 0;     int j = 0;     while(i&lt;N    j&lt;M){         if(j==N    i&lt;N&amp;&amp;v1[i].c1&lt;v2[j].c1){             procesar v1[i];             i ++;         }         else{             procesar v2[j];             j++;         }     }     return; }</pre>	<pre>void apareo(FILE* f1, FILE * f2) {     tr r1,r2;     fread(&amp;r1, sizeof(tr),1, f1);     fread(&amp;r2, sizeof(tr),1,f2);     while(! feof(f1)&amp;&amp;!feof(f2){         if(r1.c1&lt;r2.c1){             procesar r1;             fread(&amp;r1, sizeof(tr),1, f1);         }         else{             procesar r2;             fread(&amp;r2, sizeof(tr),1, f2);         }     }     while(!feof(f1)){         procesar r1;         fread(&amp;r1, sizeof(tr),1, f1);     }     while(!feof(f2)){         procesar r2;         fread(&amp;r2, sizeof(tr),1, f2);     }     return; }</pre> <p>Puede adaptar el otro modelo para estructura tipo archivo Señale semejanzas y-o diferencias entre los modelos Cual cree mas eficiente ( defina las eficiencias que considere) Que modelo le parece mas adecuado</p>
<p>Los modelos precedentes evalúan con un criterio creciente y por un solo campo, cree que es posible aparear por mas de un campo? Que precondiciones deberían cumplirse? Es posible cambiar el criterio de selección? Se puede establecer la propiedad de generalidad para el tipo de dato? Para el criterio de selección? Justifique sus respuestas</p> <p>El modelo presentado no evalúa la posibilidad de la igualdad en forma explícita. Como esta planteado, en caso de haber igualdades, que estructura se evalúa primero? El valor igual cree que</p>	



se procesa en la próxima iteración? En caso que así sea como puede evitarse procesar ese registro? Si hubiera repeticiones en la misma estructura indexada, como se puede procesar solo un representante de ese subconjunto? Habiendo ejemplificado con estructuras del mismo tipo, preguntamos: Puede hacerlo con listas? Si el propósito es aparear dos y generar una tercera, que estructura generaría para optimizar eficiencia? Es posible hacerlo con estructuras de características diferentes? Se pueden aparear tres estructuras? Justifique. Se pueden aparear N estructuras? Los registros de las estructuras a aparear, tienen que ser del mismo tipo? Justifique. Puede pensar un modelo que solo utilice un ciclo de repetición? Se puede aparear por más de un campo?. Podría conservar la algoritmia? Justifique.

#### Busqueda secuencial

Vector	Lista
<pre>int buscarSec(tr v[], int bus, int N){     i = 0;     while(i &lt; N &amp;&amp; v[i].c1 != bus){         i++;     }     return i &lt; N? i: -1; }</pre>	<pre>Nodo * buscarSec(Nodo * l, int bus){     while(l &amp;&amp; l-&gt;info.c1 != bus){         l = l-&gt;sgte;     }     return l != NULL &amp;&amp; l-&gt;info != bus? l: NULL; }</pre>

Se puede buscar por coincidencia de más de un campo? Puede generalizar por tipo de dato y criterio? Justifique. Puede optimizar el modelo? Justifique. Que opina de la aplicación de este patrón en archivos? Que condición deben cumplir los datos para que el patrón se pueda aplicar? Las expresiones dentro del while se pueden invertir sin alterar la selección y resultado?

#### Busqueda directa

Vector	Archivo
<pre>tr búsquedaDir(tr v[], int indice){     return v[indice]; }</pre>	<pre>tr búsquedaDir(FILE * f, int indice){     tr r;     fseek(f, sizeof(tr)*índice, SEEK_SET);     fread(&amp;r, sizeof(tr), 1 f);     return r; }</pre>

Puede utilizar un patrón similar en estructuras enlazadas

Que condición debe cumplirse, en la estructura y la clave para aplicar este patrón?

#### Busqueda binaria

Vector	Archivo
<pre>Int busBin( tr v[], int aBuscar, int N, int &amp;pri){     int ult = N-1, pri = 0, m;      while(pri &gt;= ult)         m = (pri + ult)/2;      if (v[m]. ci == aBuscar) return m;     if( aBuscar &gt; v[m].c1) pri = m+1;     else ult = m-1; } return -1; }</pre>	<pre>Int busBin( FILE*f, int aBuscar){     int ult = CantReg(f)-1, pri = 0, m;     tr r;     while(pri &gt;= ult)         m = (pri + ult)/2;         fseek(f, m*sizeof(r), SEEK_SET);         fread(&amp;r, sizeof(r), 1 f);         if (r. ci == aBuscar) return m;         if( aBuscar &gt; v[m].c1) pri = m+1;         else ult = m-1;     }     return -1; }</pre>



Es aplicable el modelo en estructuras enlazadas? Justifique. Es posible buscar por dos campos? Que condición debe darse? Cambiaría lógica, algoritmia o ambas cosas? Justifique. Que criterio de ordenamiento supone la búsqueda desarrollada, puede generalizar criterios? Y tipo de dato? Que cambios haría para combinar búsqueda binaria por un campo con repetición con directa, binaria o secuencial por otro campo? Que condiciones deberían cumplirse para poder efectivizar esto, que cambios propondría? Como se puede implementar la búsqueda binaria con una función recursiva? Que piensa de la eficiencia en este modelo?

#### Metodo de ordenamiento

#### Vector

```
void ordenarVector(tr v[], int N) { // v el vector a ordenar N cantidad de componentes
    int i,j;
    tr aux;
    for(i = 1; i < N; i++){ // pasos → 1..N-1
        for( j = 1; j <= N - i; j++){ // comparaciones en cada paso → 1..N-i
            if(v[j-1].c1 > v[j].c1{
                aux = v[j];
                v[j] = v[j - 1];
                v[j - 1] = aux;
            }
        } // fin ciclo interno
    } // fin ciclo externo
}
return;
} // fin de la función
```

Ejemplo del concepto de modificar la lógica de selección manteniendo la algoritmia ordenando por dos campos

```
void ordenarVector(tr v[], int N) { // v el vector a ordenar N cantidad de componentes
    int i,j; tr aux;
    for(i = 1; i < N; i++){ // pasos → 1..N-1
        for( j = 1; j <= N - i; j++){ // comparaciones en cada paso → 1..N-i
            if((v[j-1].c1 > v[j].c1) | (v[j-1].c1 == v[j].c1 && v[j-1].c2 > v[j].c2){
                aux = v[j];
                v[j] = v[j - 1];
                v[j - 1] = aux;
            }
        } // fin ciclo interno
    } // fin ciclo externo
}
return;
} // fin de la función
```

Ejemplo con plantilla

```
Template <typename T> void ordenarVector(tr v[], int N, int(criterio(T,T) {
    int i,j; T aux;
    for(i = 1; i < N; i++){
        for( j = 1; j <= N - i; j++){
            if((criterio(v[j-1], v[j]) == 1{
                aux = v[j];
                v[j] = v[j - 1];
                v[j - 1] = aux;
            }
        }
    } // fin ciclo interno
}
```



<pre>                 }// fin ciclo externo return; función de ordenamiento creciente del campo 1 int campo1creciente(tr a, tr b){     return a.c1&gt;b.c1?1:0; } Ejemplo invocacion ordenarVector&lt;tr&gt;(v, n. campo1creciente) </pre>	
<p>Es aplicable este método para archivos o estructuras enlazadas?          Que entiende por carga directa con PUP posición única predecible?          En que tipo de estructura puede ser aplicable?          Que características debe tener la clave de búsqueda          Que método de ordenamiento es aplicable en listas ordenadas?          Es aplicable la combinación Búsqueda binaria – binaria; búsqueda directa – directa; búsqueda binaria – directa; búsqueda binaria – secuencial?          Si en lo anterior algún modelo es factible ejemplifique.</p>	
Ejemplo de plantillas y punteros a funciones	
Recorrer sin plantilla y criterio	Recorrer con plantilla y puntero
<pre> void mostrar(int v[], int n){      for(int i=0; i&lt;n; i++){         cout &lt;&lt;(v[i]&lt;&lt;endl;     }      return; } </pre>	<pre> template &lt;typename T&gt; void mostrar(T v[], int n),void (*ver(T))){     for(int i=0; i&lt;n; i++){         ver(v[i];     }     return; }  void ver_reg(TipoReg r){     cout &lt;&lt; r.c1.....&lt;&lt;endl;     return; }  mostrar&lt;tipoReg&gt;(v, 10,ver_reg); </pre>
Cargar sin repetir la clave	
Vector sin orden	Vector ordenado
<pre> void cargarSinRepetir(tr v[], tr aCargar, int &amp;N) {      Int pos = buscarSec(v, aCargar, N);     If (pos == -1){          v[N] = aCargar;         N++;     }     return; } </pre>	<pre> void cargarSinRepetir(tr v[], tr aCargar, int &amp;N) {     int pri = 0, i = N;     int pos = busBin(v, aCargar, N, pri);     If (pos == -1){         for( ;i&gt;= pri; i--) v[i+1] = v[i];         v[pri]= a Cargar;         N++;     }     return; } </pre>
<p>Es aplicable este modelo en archivos?          De tener necesidad de hacerlo, que estrategia utilizaría?</p>	



Recuerde que en listas ordenadas existen las funciones insertarOrdenado y cagar sin repetir. Como diferencia tamaño lógico de físico, si el parámetro N es el tamaño lógico, que precondition debe establecer para no superar el tamaño físico? justifique

```
void mostrar(int v[], int n)
{
    for(int i=0; i<n; i++){
        cout << (v[i]<<endl;
    }

    return;
}
```

```
template <typename T> void
mostrar(T v[], int n),void
(*ver(T))
{
    for(int i=0; i<n; i++){
        ver(v[i];
    }

    return;
}

void ver_reg(TipoReg r){
    cout << r.cl.....<<endl;
    return;
}

mostrar<tipoReg>(v, 10, ver_reg);
```

Observe la función ver dice `void (*ver(T))` retorna el valor ausente (void) y es un puntero a una función que evalúa un tipo de dato, genérico llamado aquí T. Aquí generaliza el tipo de dato, observe en la invocación que entre corchetes angulares se establece el tipo de dato particular a evaluar en este caso y `ver_reg` invoca a la función particular donde se indica que mostrar en este caso particular. Aquí se puede generalizar el criterio de selección de lo que busca mostrar. Aquí formulamos algunas preguntas porque la declaración es `void (* ver( T))` ? podría ser `void*ver(T)`? que significado tienen los paréntesis? Defina la semántica de las dos declaraciones. Observe además la invocación, cuál es la semántica de ese identificador? Es correcto que no tenga parámetros? Justifique su respuesta

Los patrones para estructuras enlazadas serán desarrollados luego de la exposición teórica y conceptual de los temas