Manejo de Registros de longitud variable en colecciones dinámicas.

Organización de Datos 75.06
Curso: Ing Saubidet.

Table of Contents

Manejo de Registros de longitud variable en colecciones dinámicas.

En este apunte vamos a describir los detalles de implementación que deben tenerse en cuenta al trabajar con registros de longitud variable en archivos directos o indexados. Estos conceptos se pueden aplicar tambien a otras organizaciones o bien directamente a estructuras de archivos planos (streams) en los cuales se manejen datos de longitud variable.
Se dice que la colección de datos es dinámica cuando la ocurrencia de altas, bajas y modificaciones en los datos es posible. Cuando esto no ocurre se dice que la colección es estática.

Lineamientos generales

Vamos a suponer que lo unico que nos preocupa en cuanto a la implementación del archivo es el manejo de los datos de longitud variable. Estos datos estaran almacenados en un espacio común, a veces llamado área de datos y se accederá a los mismos mediante una referencia que en general será un “offset” o desplazamiento que indique la posición (número de byte) en donde se encuentra el registro a buscar.

Este espacio común o area de datos puede ser un archivo independiente o bien parte de otro archivo segun la organización que se maneje.

La forma en que se determina el offset para acceder a un registro no es tema de este apunte, puede ser mediante una función de hashing, mediante un índice u otra técnica.

Registros de longitud variable

A efectos prácticos vamos a suponer que trabajamos con una organización indexada muy simple que permite almacenar datos sobre aeropuertos. Cada aeropuerto se identifica por una clave internacional de 3 caracteres alfanuméricos. El único dato que se guarda es el nombre del aeropuerto. Esta organización se va a implementar usando un archivo con registros de longitud fija para guardar los pares clave-offset y un archivo con registros de longitud variable en donde se van a guardar los datos. Para buscar una clave se realiza una búsqueda binaria sobre el índice. En el archivo de datos se almacenará, en principio, un registro a continuación del otro indicando la longitud de cada registro. Las longitudes se almacenaran en 2 bytes.

Ejemplo (pares clave-valor) :

R1 = (“MIA”, “Miami International”)

R2 = (“EZE”, “Ezeiza, Ministro Pistarini”)
R3 = (“LAX”, “Los Angeles International”)
R4 = (“ORL”, “Paris, Orly”)

claves.dat (clave-offset)

EZE	21
LAX	48
MIA	0
ORL	75

datos.dat (la primera columna indica el offset y no es parte del archivo)

0	1	9	M	i	a	m	i		I	n	t	e	r	n	a	t	i	o	n	a
20	l	2	5	E	z	e	i	z	a		M	i	n	i	s	t	r	o		P
40	i	s	t	a	r	i	n	i	2	6	L	o	s		A	n	g	e	l	e
60	s		I	n	t	e	r	n	a	t	i	o	n	a	l	1	0	P	a	r
80	i	s		O	r	l	y	////	////	////	////	////	////	////	////	////	////	////	////	////	////

Veamos entonces como se procede a hacer una búsqueda: En primer lugar se hace una búsqueda binaria sobre el indice recuperandose el offset del registro buscado, luego se accede al archivo de datos mediante dicho offset, se lee la longitud del registro en bytes y luego se lee la cantidad de bytes indicada del archivo recuperando el registro completo.

Como hemos visto en el archivo de datos es necesario indicar la longitud del registro para saber cuantos bytes leer, estos datos que no forman parte de los datos en si son datos administrativos y se definen como “overhead administrativo”

> **Definición**:
> *Overhead*: Espacio ocupado en un archivo por datos administrativos.

Como vemos el tamaño real de un registro es igual al tamaño de los datos mas el tamaño de los datos administrativos, en este caso todo registro mide su longitud mas 2 (dos) bytes administrativos en donde se almacena la longitud de los datos propiamente dichos.

Altas bajas y modificaciones con registros de longitud variable

Como sabemos es posible sobre un archivo indexado realizar altas, bajas y modificaciones por lo que debemos analizar como se implementaría cada una de estas

operaciones en el esquema que propusimos. Como el manejo de registros de longitud variable es independiente del tipo de archivo y acceso utilizado vamos a trabajar a partir de aqui unicamente con el área de datos olvidándonos de las claves y de su manejo.

Vamos a estudiar si con los datos administrativos que hemos agregado hasta el momento se pueden realizar las tres operaciones basicas sobre registros (Alta, Baja y Modificación)

Bajas:

Las bajas las vamos a representar como

D(offset).

Indicando el offset del registro que debemos eliminar, obviamente este offset se obtiene a partir del índice o de alguna otra estructura que ya no es relevante para nosotros.
Al dar de baja un registro es imposible, fisicamente, eliminar datos del medio de un archivo, por lo tanto la única opción es marcar al registro como libre. Tenemos que tener alguna forma, pues, de marcar si un registro esta libre u ocupado.

De aqui se desprende que debemos agregar a los datos administrativos algun flag que indique el “estado” del registro que puede ser “libre” u “ocupado”.

Como vemos en nuestra área de datos van a co-existir registros ocupados y registros libres a medida que se borren registros de datos.

A los registros libres, por no contener datos los vamos a denominar “fragmentación externa”

Al liberarse un registro debe observarse si los registros “vecinos” estan libres en cuyo caso es posible “fundir” dos o tres registros libres en uno solo de mayor tamaño.

Definicion:
Fragmentacion Externa: Registros libres dentro del area de datos.

La fragmentacion externa se da unicamente cuando los registros libres tienen longitud variable ya que el problema es precisamente que los registros libres chicos pueden quedar si usarse en forma indefinida si no sirven para guardar datos.

Algoritmo Baja:

```
        registro.status <- libre
```

```
        if(next_record.status == libre && prev_record.status ==
libre) {
          join next, prev and current record in 1 record
        } elseif (next_record.status == libre) {
          join next and current record in 1 record
        } elseif (prev_record.status == libre) {
          join prev and current record in 1 record
        }
        update list of free records
```

Altas:

Las altas se representan aqui como:

A(“datos”)

Indicando los datos del registro que debe agregarse al área de datos.
Al dar de alta un registro debemos contemplar si existen registros “libres” que puedan ser reusados para reducir la fragmentación externa. En caso de existir debemos seleccionar a alguno de estos registros y existen diferentes algoritmos para ello. Por eso en general es necesario manejar a los espacios libres como si fueran una lista de forma tal de poder, rapidamente, conocer cuales son los registros libres y operar sobre ellos.

Los datos administrativos deben poder contemplar el encadenamiento de registros libres a fin de que los mismos se estructuren como una lista.

Cuando no existan registros libres o no pueda usarse ninguno de ellos para el registro que quiere darse de alta se agrega un nuevo registro al final del área de datos. Se dice entonces que el area de datos se “extiende”

Algoritmo alta:

```
           if (r <- select_free_record()) {
               use record r to store data
               update list of free records
           } else {
               add new record at the end
           }
```

Modificaciones:

Una modificacion se representa de la forma:

U(offset,datos)

Indicando los datos que reemplazan a los datos del registro cuyo offset se indica.
Al realizar una modificación (Update) de un registro puede ocurrir que la longitud de los datos sea igual, mayor o menor a los datos anteriores.

Cuando la longitud es la misma los nuevos datos sobre-escriben a los datos viejos sin que se produza ningun problema.

Cuando la longitud de los nuevos datos es inferior a los datos anteriores lo primero que podemos pensar es en crear un registro libre con el espacio liberado, pero esto no siempre es posible ya que el espacio liberado debe ser suficiente como para poder contener al menos 1 byte de datos y ademas los datos adminstrativos de un registro libre. Cuando esto no es posible la unica opción que nos queda es marcar que el registro tiene una cierta cantidad de bytes libres. Por lo tanto los datos administrativos deben poder almacenar la cantidad de bytes libres en un registro. A estos bytes, insuficientes para crear un nuevo registro libre los llamaremos “fragmentacion interna”.

Definicion:
Fragmentacion interna: Bytes libres dentro de un registro.

Cuando la longitud del nuevo registro es mayor a la longitud original debe analizarse si existen bytes libres en el registro que permitan almacenar los datos, si esto no ocurre se puede analizar si el registro siguiente al que se quiere actualizar esta libre, en cuyo caso se puede usar su espacio para agrandar el registro, quedando un registro libre de menor tamaño a continuación o bien dejando bytes libres en el nuevo registro formado.

Si ninguna de las opciones mencionadas es posible se debe marcar el registro como libre haciendo una operación de borrado y proceder como si fuera una operación de alta insertando el registro en algun espacio libre o bien al final del archivo.

Algoritmo Update:

```
if (new_size == old_size) {
  update record
 } elseif(new_size <= old_size) {
   determine if a free record can be created
   if(space enough for new free record) {
     create new free record
     update record
     update list of free records
   } else {
     update record (mark free bytes)
   }
 } else {
```

```
            if (record has enough free space) {
              update record free space
              update record
            } else {
              delete record
              if(nex_sibling || prev_sibling = free) {
                create new free record
              }
              insert (new_record)
          }
        }
```

Datos administrativos necesarios para este método

En base a lo anterior vemos que los datos administrativos deben ser capaces de almacenar la siguiente informacion:

- La longitud del registro
- La cantidad de bytes libres en el mismo
- El estado del registro (libre u ocupado)
- Encadenamiento de registros libres

Veamos como podemos implementar estos datos administrativos en nuestro ejemplo:

- Longitud del registro (2 bytes, ya lo teniamos)
- Cantidad de bytes libres en el mismo (2 bytes)
- Estado del registro (1 byte)
- Encadenamiento de libres (si el reg esta libre los 2 bytes usados para indicar la cantidad de bytes libres se usan para indicar el proximo registro libre, se usa 0 para indicar que el registro es el ultimo libre). En el header del archivo se guarda el primer registro libre.

Analisis de los datos administrativos

El tamaño que se destina a los datos administrativos no es trivial ya que la cantidad de registros, longitud maxima de los mismos y otros parámetros importantes dependen de estos factores. Veamos de que forma:

Longitud del registro:

La longitud del registro en bytes es un entero sin signo que indica la cantidad de bytes

almacenados en el registro. Este dato limita efectivamente la longitud máxima de un registro en el archivo.

Cantidad de bytes libres/Encadenamiento de libres:

La cantidad de bytes libres es un entero sin signo de la misma longitud que el utilizado para la longitud del registro ya que la máxima cantidad de bytes libres en un registro es igual a la longitud máxima del mismo menos 1.
Además, como vimos, este dato se utiliza también para encadenar los registros libres ya que cuando el registro esta libre todos sus bytes estan libres y este campo no se utilizaría. Debido a esto este campo limita también el maximo offset alcanzable dentro del archivo limitando pues el tamaño máximo del archivo.

Estado del registro:

El estado del registro puede ser como mínimo un bit indicando si el registro está libre u ocupado.

La lista de registros libres

Para que el manejo de la lista de libres sea eficiente en general se trabaja la lista de registros libres en memoria levantando la lista al momento de abrir el archivo o bien en el primer momento en que deba usarse la lista (lazy load).

Algoritmos para la selección de registros libres

Hemos mencionado que al realizar un “insert” lo primero que debemos verificar es la posibilidad de usar un registro de la lista de libres para reducir la fragmentación externa, obviamente solo consideramos como “candidatos” a aquellos registros libres que tengan espacio suficiente para el nuevo registro. Los algoritmos que se pueden usar para seleccionar el candidato son:

First Fit: Elegir el primer registro que se pueda usar
Best Fit: Elegir el registro de tamaño mas parecido al que se quiere guardar
Worst Fit: Elegir siempre el registro libre mas grande

Best Fit tiende a crear fragmentacion interna mientras que Worst fit reduce la fragmentacion interna pero crea fragmentacion externa. Estos algoritmos pueden funcionar mejor o peor segun el caso.

Un ejemplo práctico

Vamos a trabajar realizando distintas operaciones sobre el area de datos usando como datos administrativos los de nuestro ejemplo, vamos a usar 1 byte para indicar el estado del registro (O=ocupado, L=libre), 2 bytes para la longitud del mismo y 2 bytes para la cantidad de bytes libres y encadenamiento de registros libres. De esta forma cada registro ocupa su longitud mas 5 bytes de datos administrativos. Se utilizará el algoritmo “Worst Fit” para seleccionar un registro libre al hacer un alta.

I(“hola”)

Offset	***Datos***
0	O,04,00,hola (9 bytes)

I(“123”)

Offset	***Datos***
0	O,04,00,hola (9 bytes)
9	O,03,00,123 (8 bytes)

I(“saludos”)

Offset	***Datos***
0	O,04,00,hola (9 bytes)
9	O,03,00,123 (8 bytes)
17	O,07,00,saludos (12 bytes)

U(9,”pepe”)

En el offset 9 observamos que el registro tiene solo 3 bytes, al no haber un vecino libre tenemos que dar de baja el registro y agregar uno nuevo al final.

Offset	***Datos***
0	O,04,00,hola (9 bytes)
9	L,03,00,XXX (8 bytes)

Offset	***Datos***
17	O,07,00,saludos (12 bytes)
29	O,04,00,pepe (9bytes)

U(0,"A")
Esta operación implica reducir el registro que contiene "hola" a "A", como "hola" solo ocupa 4 bytes los bytes liberados serían insuficientes para generar un nuevo registro libre por lo que debemos generar fragmentación interna.

Offset	***Datos***
0	O,01,03,A (9 bytes)
9	L,03,00,XXX (8 bytes)
17	O,07,00,saludos (12 bytes)
29	O,04,00,pepe (9 bytes)

I("juan")

Como no hay registros libres con al menos 4 bytes debemos agregarlo al final.

Offset	***Datos***
0	O,01,03,A (9 bytes)
9	L,03,00,XXX (8 bytes)
17	O,07,00,saludos (12 bytes)
29	O,04,00,pepe (9 bytes)
38	O,04,00,juan (9 bytes)

D(17)

Se marca el registro como libre y como el anterior tambien esta libre se unen ambos registros en un unico registro libre. Notar que la longitud del nuevo registro libre se suman los bytes administrativos que se usaban en el registro vecino.

Offset	***Datos***
0	O,01,03,A (9 bytes)
9	L,15,00(15 bytes)
29	O,04,00,pepe (9 bytes)
38	O,04,00,juan (9 bytes)

Caracteristicas

- Las operaciones de Alta, Baja y Modificacion son complejas
- Existe fragmentacion externa que puede ser un problema
- La fragmentación interna es mínima
- Muy eficiente para realizar lecturas

Registros de longitud variable: resumen

Parametrización	Longitud (bits) para almacenar la longitud de un reg. Longitud (bits) para indicar bytes libres en un registro (tambien usada para encadenar registros) Longitud en bits para indicar el estado de un registro
Altas	1 acceso
Bajas	1 acceso
Modificaciones	1 o 2 accesos
Búsquedas	1 acceso
Fragmentación interna	Escasa
Fragmentación externa	Abundante, puede requerir reorganizar el archivo frecuentemente cuando hay demasiados “huecos” inusables.

El efecto queso gruyere, reorganizando archivos

El efecto “queso gruyere” se da cuando en un archivo de datos existen muchos registros “libres”, en condiciones ideales el espacio libre no deberia existir o de existir debería estar ubicado al final del archivo, de lo contrario se corre el riesgo de que el espacio libre

ocupe tanto o mas espacio que los datos reales especialmente si los datos son muy dinámicos. En casi todas las implementaciones debe existir una forma de “reorganizar” el archivo eliminando del mismo los registros libres y generando un nuevo archivo que contiene unicamente datos, a esta operación se la denomina “packing” o “reorganización”.
Dependiendo de la implementación esta operación se puede hacer automaticamente cuando se detecta una determinada condición o bien a pedido del usuario o administrador del sistema.

Usando registros de longitud fija para almacenar datos de longitud variable

En el método anterior describimos las técnicas y algoritmos de alta, baja y modificación genéricos para manejar registros de longitud variable. Como veremos a continuación existen otros métodos para almacenar datos de longitud variable, curiosamente uno de estos métodos esta basado en la utilización de registros de longitud fija, es decir que se podria implementar sobre un archivo relativo.

En este método cada registro tiene una longitud fija en bytes. Como datos administrativos se indican por cada registro el estado del mismo (libre/ocupado), la cantidad de bytes libres en el mismo y ademas cual es el registro en el cual los datos continuan para los casos en los cuales un dato no entre en un registro.

En esta estructura las referencias a los datos pueden ser simplemente un numero de registro, no hace falta un offset ya que el mismo se calcula multiplicando el número de registro por la longitud de cada registro. La lista de registros libres puede ser una lista de numeros de registro libres sin datos adicionales ya que todos los registros libres tienen el mismo tamaño.

Ejemplo: (con registros de 4 bytes, 1 byte de estado, 2 bytes para bytes libres, 2 bytes para indicar el proximo registro. El numero de proximo registro se indica por numero de registro, no por offset numerando desde 0)

Estado	***Bytes libres***	***Proximo***	***Datos***
O	1	0	123
O	0	3	salu
L	4	0	
O	1	0	dos

Aqui vemos dos registros almacenados, uno con “123” y otro con “saludos”, este segundo registro ocupa dos registros en el archivo ya que en uno no entraría.

Altas bajas y modificaciones usando registros de longitud fija

Veamos ahora como funcionarian las distintas operaciones utilizando esta estructura.

Bajas

Para dar de baja un registro simplemente se lo marca como libre y si los datos ocupaban mas de un registro se marcan como libres todos los bloques que se usaban.

Altas

Para dar de alta un registro simplemente se usan uno o mas bloques libres en caso de ser necesario, si no hay suficiente cantidad de bloques libres se agregan bloques libres al final del archivo.

Modificaciones

Para realizar una operacion de tipo update el procedimiento es muy sencillo. Si el nuevo dato es menor en tamaño que el anterior se eliminan los registros que antes se usaban y ya no hacen falta marcandolos como libres y/o se ajusta la cantidad de bytes libres en el primer registro. Si el dato es mas grande segun entre o no en un registro se procede a agregar nuevos registros para contener el dato.

Caracteristicas

- Las operaciones de Alta, Baja y Modificacion son muy simples
- Se elimina, practicamente, la fragmentacion externa
- La fragmentacion interna es enorme
- Pueden requerirse varios accesos para leer un solo registro

Registros de longitud fija: resumen

Parametrización	Longitud (bits) para indicar bytes libres en un registro Longitud en bits para indicar el estado de un registro Longitud (bits) para almacenar el próximo registro en caso de que los datos no entren en un registro.
Altas	Varios accesos
Bajas	Varios accesos
Modificaciones	Varios accesos
Búsquedas	Varios accesos
Fragmentación interna	Abundante
Fragmentación externa	Nula

El método de Zobel, Moffat y Sack-Davis

Hasta el momento hemos visto dos técnicas para almacenar datos de longitud variable, el primer método estaba basado en registros de longitud variable mientras que el segundo estaba basado en registros de longitud fija. Existe una tercera aproximación que fue sugerida por Zobel, Moffat y Sack-Davis que se basa en una combinación de las dos técnicas que vimos hasta el momento. A continuación describimos el método.

El área de datos se divide en “n” “bloques” de una longitud dada. En general esta longitud debe ser mucho mayor al tamaño promedio de un registro.
Cada registro se identifica con un número, existe un mapeo que indica en que bloque se encuentra un registro de acuerdo a su número.

Estructura

Cada bloque tiene la siguiente estructura:

En primer lugar un numero indicando la cantidad de registros en el bloque.
Luego una tabla de longitud variable donde cada entrada de la tabla tiene longitud fija e indica: número de registro + offset.
Luego el espacio libre dentro del bloque.
Y por ultimo los registros almacenados.

Ejemplo:

Usando 4 bytes para cada entrada de la tabla, 2 para el número de registro y 2 para el offset del mismo en el bloque, la cantidad de registros en el bloque ocupa 4 bytes,):

```
00: 0003 (cantidad de registros)
04: 2329 (aqui empieza la tabla reg29 = offset)
08: 1534 (reg 15 = offset 34)
12: 0138 (reg 01 = offset 38)
16: XXXX (espacio libre)
20: XXXX (...)
24: XXXX (...)
28: X123 (en el byte 29 empieza el reg 29)
32: 45ho (en el byte 34 empieza el reg 15)
36: laca (en el byte 38 empieza el reg 01)
37: sita
```

En el ejemplo vemos un bloque con 3 registros almacenados, el registro 23 en el offset 29 del bloque tiene 5 bytes y contiene “12345”. El registro 15 en el offset 34 del bloque tiene 4 bytes y contiene “hola” mientras que el registro 01 en el offset 38 del bloque contiene “casita”. Las “X” simbolizan el espacio libre en el bloque.
Notese que no hace falta indicar la longitud de cada registro ya que la misma se obtiene de la tabla buscando el offset del registro próximo, en donde comienza un registro termina el anterior.

Observemos que la cantidad de registros del bloque indica la longitud de la tabla que especifica el offset de cada registro.

La lista de espacios libres indica numero de bloque y cantida de espacio libre para aquellos bloques en los cuales el espacio libre es mayor a una cierta tolerancia fijada (de lo contrario todos los bloques estarían en la lista de libres). Cuando esta lista es demasiado grande se procede a reorganizar el archivo.

Intencionalmente se deja el espacio libre en el medio de forma tal que la tabla que indica la ubicación de los registros crezca hacia abajo y los datos crezcan hacia arriba en el bloque para minimizar el movimiento de datos dentro del bloque.

Altas bajas y modificaciones

En general todas las operaciones se hacen leyendo/escribiendo bloques enteros a

memoria, por lo tanto las operaciones dentro de un mismo bloque no son costosas ya que se hacen en memoria. Esta es una de las claves del metodo presentado.

Veamos como se realizan las operaciones de alta, baja y modificacion con esta estructura:

Bajas

Para dar de baja un registro se procede primero a localizar el bloque al cual pertenece el registro, una vez dentro del bloque se identifica el offset del registro dentro del bloque y se libera el espacio reacomodando los registros y reescribiendo la tabla del bloque. Se debe ademas dar de baja el registro en el mapeo registro-bloque.

Ejemplo:

```
00: 0003
04: 2329
08: 1534
12: 0138
16: XXXX
20: XXXX
24: XXXX
28: X123
32: 45ho
36: laca
37: sita
```

Luego de dar de baja el registro 15:

```
00: 0003
04: 2333
08: 0138
12: XXXX
16: XXXX
20: XXXX
24: XXXX
28: XXXX
32: X123
36: 45ca
37: sita
```

Observemos como queda reorganizado el espacio libre y el espacio de datos con los datos de los registros que permanecen en el bloque.

Altas

Si existe un bloque en la lista de bloques con espacio libre suficiente entonces buscar un bloque que tenga espacio libre para contener al dato (se puede usar Worst Fit o Best Fit), almacenar el dato en el bloque y actualizar la lista de libres. En caso de no haber un bloque con suficiente espacio se crea un nuevo bloque al final del archivo y se agrega el registro al mismo.
Para agregar un registro ya dentro del bloque se agrega una entrada en la tabla y se agrega el registro en el offset correspondiente creciendo hacia arriba..

Ejemplo: Agregar el registro 28 que contiene “z”

Antes:

```
00: 0003
04: 2333
08: 0138
12: XXXX
16: XXXX
20: XXXX
24: XXXX
28: XXXX
32: X123
36: 45ca
37: sita
```

Despues

```
00: 0003
04: 2333
08: 0138
12: 2832
16: XXXX
20: XXXX
24: XXXX
28: XXXX
32: z123
36: 45ca
37: sita
```

Notar que no hizo falta reorganizar nada, se agrego una entrada en la tabla haciendo crecer la misma hacia abajo y se agregaron los datos en el área de datos. Para que un registro pueda agregarse deben existir entonces tantos bytes libres como los necesarios para guardar los datos del registro mas una entrada de la tabla.

Ademas debe actualizarse el mapeo indicando que el bloque en el cual se puede encontrar ahora al registro numero 28.

Modificaciones

Para realizar modificaciones se procede de forma similar, si el dato se achica se reorganiza el bloque, eventualmente agregandolo a la lista de libres si antes no estaba y el nuevo espacio libre es suficiente. Si el dato es mas grande y hay espacio libre suficiente en el bloque entonces se procede a reorganizar el bloque, si no hay espacio libre suficiente se elimina el registro del bloque y se procede como si fuera un alta.

¿Que pasa cuando un registro es mas largo que un bloque?

En primer lugar es meritorio analizar si tal cosa se permite, en caso de permitirse los autores sugieren agregar un elemento extra a cada bloque indicando número_registro + bloque_siguiente, de esta forma se permite unicamente un bloque en “overflow” por registro pero esto es mas que suficiente si el tamaño del bloque se elige cuidadosamente. Las operaciones de borrado y update deben tener en cuenta esta variante si.

Caracteristicas

- Altas bajas y modificaciones simples
- Fragmentacion externa casi nula (bloques libres son raros)
- La fragmentación interna es abundante (espacio libre en cada bloque)
- Muy eficiente para realizar lecturas

Registros de longitud variable: resumen

Parametrización	Tamaño del bloque. Longitud (bits) para indicar la cantidad de registros en cada bloque. Longitud (bits) para las entradas de la tabla de cada bloque indicando número de registro y offset. Cantidad de bytes libres que debe tener un bloque para ser inlcuido en la lista de libres. Posibilidad de que un registro ocupe mas de un bloque (longitud máxima del registro)
Altas	2 accesos
Bajas	2 accesos
Modificaciones	2 accesos
Búsquedas	2 accesos
Fragmentación interna	Abundante
Fragmentación externa	Escasa