Universidad de Talca
Facultad de Ingeniería
Ingeniería Civil en Computación

Algoritmos y estructuras de datos
Informe tarea N° 3
10 de Julio de 2012

1. Introducción

Él problema principal reside en ordenar elementos alojados en la memoria auxiliar (HDD) utilizando la menor cantidad posible de memoria fisica (RAM), dado que el conjunto de datos a ordenar supera enormemente la cantidad de memoria fisica disponible.

2. Solucion del problema

2.1. Análisis del problema y algoritmo de solucion

Dado los antecedentes anteriores, utilizar algoritmos tipicos de ordenamiento en memoria fisica, tales como *Quicksort*. Pero dado que la memoria auxiliar tambien puede ser utilizada de la misma forma que la fisica, estos algoritmos pueden facilmente ser implementados, pero nos enfrentamos al problema de que la escritura y lectura es lenta.

Algorithm 1 lectura en disco

```
mover aguja a posicion
leer contenido
```

Pero el leer el contenido de la memoria auxiliar en si es mucho mas lento comparado con una lectua en memoria fisica sea la cantidad de datos que sean, el mover la aguja en si, toma un tiempo constante. Este tiempo constante, si bien no es mucho, ha de ser multiplicado por la cantidad de lecturas que es nesesario realizar para poder cargar la informacion completa. Debido a esto, los algoritmos que no son estables, o que nececitan de muchas operaciones de insercion, intercambio o eliminacion se ven enormenente desfavorecidas.

Una de las formas de lograr ordenar esto, es dividir el archivo en partes que sea posible ordenarlas en la memoria auxiliar, y luego combinar todas estas soluciones para poder obtener el conjunto final de datos, completamente ordenados. El algoritmo de ordenamiento *MergeSort* funciona de esa manera, combinando soluciones en memoria fisica, tomando en cuenta, que los trozos que ha de mezclar ya estan previamente ordenados, lo cual deja el funcionamiento de nuestro programa asi:

Algorithm 2 $principal(vias)$

```
generacion ← 0
numeroPaginas ← separarArchivo(tamanoPaginaMemoria)
while numeroPaginas > 1 do
  numeroPaginas ← mezclar(generacion, numeroPaginas, vias)
  borrarGeneracion(generacion)
  generacion ← generacion + 1
end while
renombrar(generacion)
```

La primera de las subrutinas que se han de implementar es la de la separacion de archivos. Para ese efecto, se procede a leer el archivo binarizado, leyendo de a un dato a la vez, hasta llegar al final del archivo. El resultado de esta lectua es almacenada en un *buffer* de memoria, el cual ha de ser la cantidad de ram objetivo a utlizar. Para efectos de la tarea este numero esta truncado a 2^16 (65536) bytes (equivalentes a 2^20 (1048576) bits).

Dado esto, el numero de paginas a obtener se puede calcular utlizando la formula:

$$numeroPaginas = \lceil \frac{tamanoArchivo}{paginaMemoria} \rceil$$

Por lo cual es nesesario tener disponible adicionales en el disco, una cantidad equivalente a el doble de el tamaño del archivo original ya que hasta ahora el tamaño que ha de emplear el algoritmo ha de ser:

$$memoriaTotal = memoriaPaginas + memoriaArchivoOriginal$$

Una vez lleno este buffer, es volcado hacia la memoria auxiliar, y luego se sigue llenandose secuencialmente. Esto es para darle algo mas de flexibilidad al momento de encontrar conjuntos de datos cuyo tamaño no esten truncados en el tamaño especificado. Dado que se utiliza un heap para administrar el contenido de la pagina, al momento de volcarse esta, se garantiza que los datos esten ordenados.

Algorithm 3 separarArchivo($tamanoPaginaMemoria$)

```
numeroPaginas ← 0
buf ← newBuffer(tamanoPaginaMemoria)
while !archivo.fin() do
  buf.push(archivo.readByte())
  if buf.estaLleno() then
    buf.volcar()
    numeroPaginas ← numeroPaginas + 1
  end if
end while
return numeroPaginas
```

Luego de ejecutar la separacion del archivo, para poder ordenarlos, dependiendo si es ordenamiento de miltiples vias, o de via doble. Para ello se utiliza un heap, para poder tener un acceso ordenado a los datos. El funcionamiento es relativamente simple asi que no requiere mayor explicacion; para cada uno de los archivos se lee solo un byte. Luego, este byte, es agregado al heap minimo, de modo de obtener el dato menor. Al momento de recibir el dato, se revisa la pagina de donde provino, y se le pide un nuevo dato. Si a esta se le agotan, entonces se llego al fin de la pagina y se omite. Todo el procedimiento anterior termina cuando todas las paginas han sido revisadas (y por ende mezcladas).

Algorithm 4 mezclar($generacion, numeroPaginas, vias$)

```
numeroPaginasProsesadas ← 0
while numeroPaginasProsesadas < numeroPaginas do
  mezclarGeneracion(generacion, vias, numeroPaginasProsesadas)
end while
return numeroPaginasProsesadas
```

Algorithm 5 mezclarGeneracion($generacion, vias, iteracion$)

```
administradorPaginas ← newPageAdmin(generacion, vias, iteracion)
archivoSalida ← newArchivoSalida(generacion + 1, iteracion)
while !administradorPaginas.hasNext() do
  archivoSalida.push(administradorPaginas.nextInt())
  if archivoSalida.estaLleno() then
    archivoSalida.write()
  end if
end while
archivoSalida.write()
archivoSalida.close()
```

2.2. Diagrama de estados

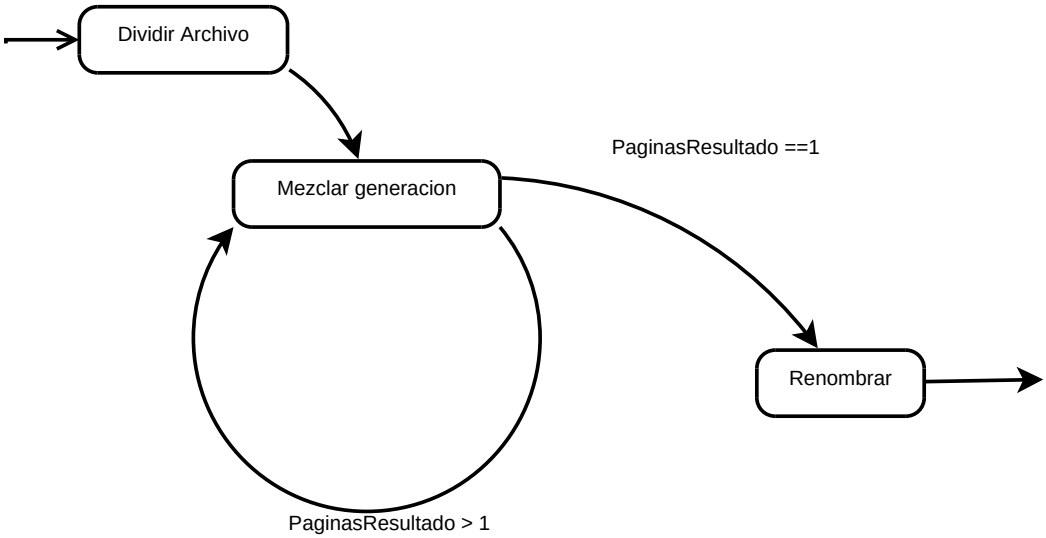

Figura 1: Programa principal, primer estado de desorden

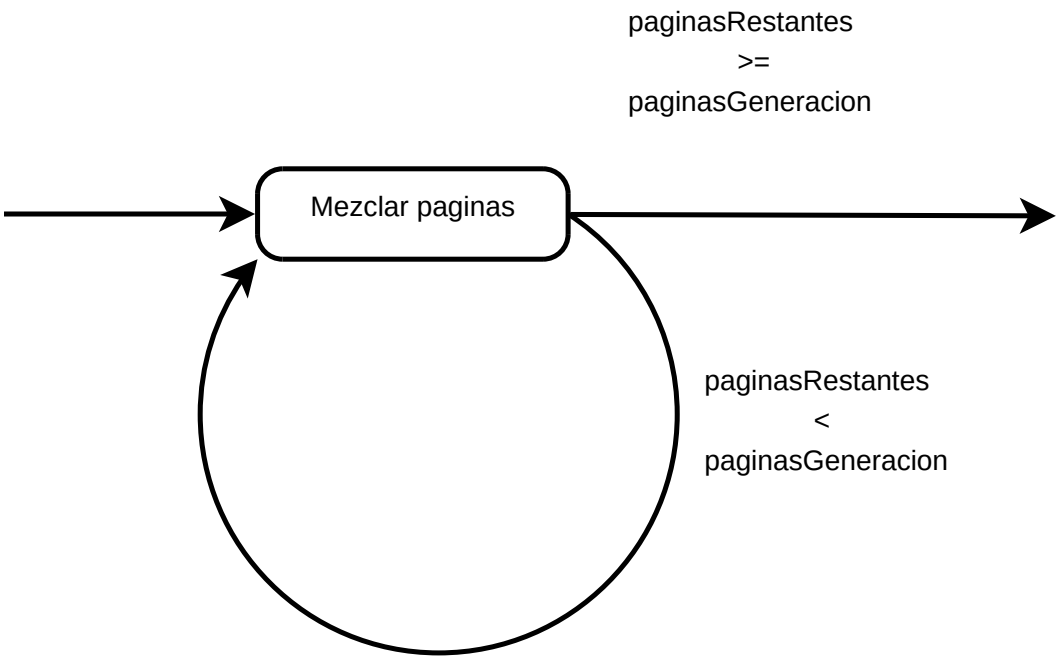

Figura 2: Estado de ordenamiento

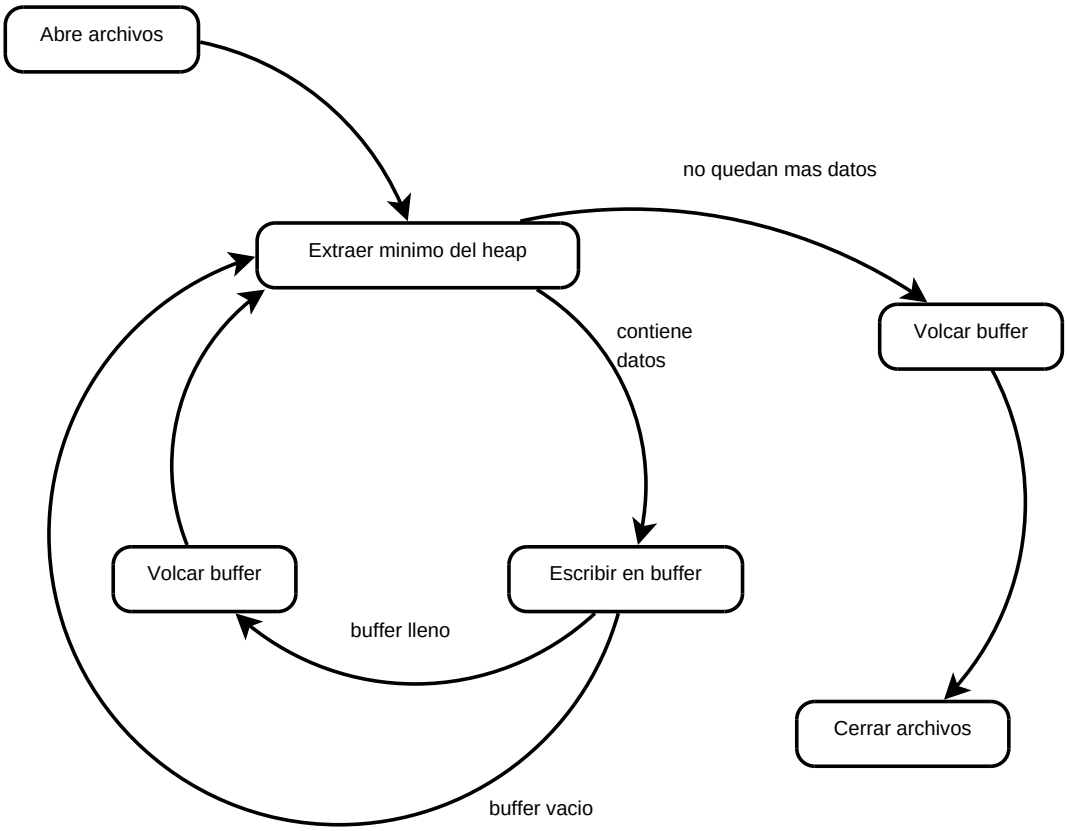

Figura 3: Ordenamiento de pagina y mezclado

2.3. Resultados de las pruebas

Pendiente.

3. Instalacion y uso del programa

Dentro del tar se encuentra un espacio de trabajo de eclipse, en el cual hay dos proyectos nombrados "proyecto3_2wz "proyecto3_8w.esçritos en el lenguaje C++. Estos contienen el mismo programa en si, el cual ejecuta una version del proyecto diferente -en el codigo se encuentra escrita la diferencia-, uno hecho para ejecutarse bajo dos vias (Mergesort comun) y otro para ejecutarse bajo 8 vias (MultiWay-Mergesort). Para poder compilar el programa, se nececita importar el proyecto a eclipse,y luego ejecutar el comando "build". Esto deja en el directorio "Debugün archivo ejecutable, con el mismo nombre del proyecto compilado. Este ejecutable ha de ser dejado en el directorio en donde se encuentra el archivo "data.bin", el cual ha de contener el conjunto de numeros desordenados en formato binario. Para dar inicio, basta con solo ejecutar el progama y este comenzara a ordenar solo el archivo en cuestion, y dejara escrito un archivo de nombre "data.bin_sorted", el cual contiene el archivo mezclado y ordenado.

Nota: El programa fue hecho pensandose que se ejecutara en alguna maquina con un entorno basado en UNIX, como Linux o Mac.