Universidad Nacional Autónoma de México
Facultad de Ciencias
Algoritmos Paralelos

Proyecto final: Implementación de Run-length enconding (RLE)

Autor: Medina Peralta Joaquín

Problema

Run-Length encoding es una forma de compresión de datos sin pérdida, donde la secuencia de símbolos de la entrada se almacenan como una simple ocurrencia con el número simbolos consecutivos.

Definimos, para una cadena $w \in \Sigma^*$, una serie de pares (r, l, s) de datos consecutivos dadas por:

- El carácter de control $r \notin \Sigma$.
- La longitud de la secuencia $l \in \mathbb{N}$
- El símbolo del alfabeto $s \in \Sigma$

Por ejemplo, tenemos una cadena perteneciente al alfabeto en inglés, `ABABBBC` y la lista de tuplas con el caracter de control α, se tiene:

$$
\begin{aligned}
&(\alpha, 1, \mathtt{A}), (\alpha, 1, \mathtt{B}), (\alpha, 1, \mathtt{A}) \\
&(\alpha, 3, \mathtt{B}), (\alpha, 1, \mathtt{C})
\end{aligned}
$$

Para poder representarlo en un archivo, los símbolos con un solo elemento consecutivo, es decir que no se repiten, se representarán como `s`. Con el ejemplo anterior, la cadena comprimida sería `ABA(`α`,3,B)C`.

Por lo tanto, el algoritmo es eficiente para cadenas o contenido donde hay muchas secuencias consecutivas, es decir, archivos de texto, binarios representación de una imagen por medio de pixeles o componentes de bloques largos de archivos de sonido. (Pu, 2005).

Para este trabajo, se consideran únicamente archivos de texto para comprimir que estén en `utf-8`, debido a que, se usará una combinación para representar las tuplas. Para los símbolos sin repetición o secuencia muy cortas, , como se menciona en (Pu, 2005), se implementará un modo literal donde los bytes se escriben directamente sin tupla. Esto evita la expansión del archivo, permitiendo que la compresión se reserve únicamente para las secuencias que si generen un ahorro significativo.

Algoritmo secuencial

Como se menciona en (Pu, 2005), se define caracteres de control de repetición $r_3, r_4, \ldots$, los cuales para r_i representa la cantidad i de símbolos consecutivos. De esta manera, se usará el caracter de control únicamente cuando la longitud sea mayor a 3.

Asi, definimos el algoritmo de encriptación como:

- Repetir hasta llegar al final del archivo:
  - Leer el símbolo S de la secuencia:
    - Contar el número de símbolos consecutivos iguales i hasta encontrar uno diferente.
    - Guardar en el archivo de salida el par r_iS solo si $i > 2$, en oto caso guardar el símbolo i

Para la implementación, se debe de considerar tres apuntadores i, j donde $j = i + 1$ donde los usaremos para recorrer el arreglo de símbolos S, para lo cual, mientras que $S[i] == S[j]$ entonces podemos incrementar el contador, de esta manera verificamos si es mayor al umbral, de esta manera podemos representarlo en forma de tupla y en otro caso, se escribe el caracter $S[q]$ y $S[p]$ en la cadena comprimida. (El pseudocódigo se encuentra en el anexo 1)

De esta manera, definimos el algoritmo de desencriptación como:

- Repetir hasta llegar al final del archivo:
  - Leer el símbolo S de la secuencia:
    - Si el símbolo $S = r_i$, se debe de leer el símbolo consecutivo y escribir i veces en la salida.
    - En otro caso, escribir el símbolo actual.

De la misma manera, es necesario un apuntador p para recorrer el archivo comprimido, donde si $S[p]$ es el caracter de control definido r, se toma el valor de $S[p+1] = n$ como un número y $S[p+2]$ como el símbolo a repetir n veces, en otro caso se guarda $S[p]$ en la salida. (El pseudocódigo se encuentra en el anexo 2)

A su vez, se considera el uso de `FLAG_RLE` para iniciar la tupla definida en la sección anterior y en dado caso que el símbolo s sea igual a `FLAG_RLE`, se considera a `FLAG_LITERAL` para escapar y escribir el valor de `FLAG_RLE`.

Algoritmo en paralelo

Para esto, se toma en inspiración del artículo (Manchev, 2006), donde se propone el dividir la secuencia de entrada en bloques procesados en paralelo donde cada proceso identifica las corridas locales dentro de su bloque, para posteriormente realizar una fase de combinación donde se revisa si los límites entre bloques contiguos se cruzan con corridas de símbolos, asegurando que las secuencias iguales se representen correctamente.

De esta manera, se debe de utilizar el uso de memoria compartidad y distribuida para poder ejecutar el algoritmo y por lo tanto, nos esta danto la pista de usar `MPI`.

Antes de pasar a la implementación, debemos mencionar que esta estrategía de paralelizar el algoritmo secuencial `RLE` sigue el principio de `Fork Join` donde se divide el trabajo entre los distintos procesadores y al final, juntar los resultados de cada procesador en uno global.

Se puede observar la técnica de `Fork Join` de manera explicita al dividir la secuencia de entrada en bloques para que cada procesador realice `RLE` de manera local y al final se junta cada bloque revisando si no hay cruzes entre cada uno.

De esta manera, proponemos el pseudocódigo (3) donde se puede observar la compresión en `RLE` en paralelo y en (4) la manera de descomprimir en paralelo.

Experimentación

Para esto, trabajaremos con 3 archivos de 100 MB con una alta, media y baja repetitividad los cuales se calculara el tiempo de comprimir de manera secuencial y paralela con $2, 4, 6, 20, 50$ procesadores.

De este modo, se realizará la prueba en el siguiente equipo:

```
---------------------
Sistema operativo: Darwin MacBook-Pro-****.local 22.6.0 Darwin
Kernel Version 22.6.0: Tue Jul 15 08:22:28 PDT 2025; root:xnu
   -8796.141.3.713.2~2/
RELEASE_X86_64 x86_64
CPU: Intel(R) Core(TM) i5-7360U CPU @ 2.30GHz
Nucleos fisicos: 2
Nucleos logicos: 4
Memoria RAM total: 8,00 GB
-------------------------------------------
```

Resultados

Conclusión

Referencias

Manchev, N. (2006). Parallel Algorithm for Run Length Encoding. *Proceedings of the Third International Conference on Information Technology: New Generations (ITNG 2006).* https://doi.org/10.1109/ITNG.2006.106

Pu, I. M. (2005). *Fundamental Data Compression* [pp. 56–64]. Elsevier Butterworth–Heinemann.

Anexo

Algoritmo 1 Algoritmo para comprimir en `RLE` con modo literal (Umbral 3)

```
procedure COMPRIMIR(S)                                    ▷ Cadena de símbolos S
    N ← longitud(S)
    0 ← []                                                ▷ Cadena de salida comprimida
    p ← 0
    while not EOF do
        i ← 1
        q ← p + 1                                         ▷ Fase 1: Contar la secuencia
        while q < N y S[q] = S[p] y i < 255 do            ▷ Máximo conteo en 1 byte
            i ← i + 1
            q ← q + 1
        end while
                                                          ▷ Fase 2: Aplicar el umbral y codificar
        if i ≥ 3 then                                     ▷ Guardar como tupla RLE: ⟨FLAG_RLE, i, S_p⟩
            O ← O ‖ FLAG_RLE ‖ i ‖ S[p]
        else                        ▷ Guardar como secuencia literal: S[p] se repite i veces
            for k ← 0 hasta i − 1 do
                O ← O ‖ S[p]
            end for
        end if
        p ← p + i          ▷ Mover el apuntador principal al inicio de la siguiente secuencia
    end while
    Salida: 0
end procedure
```

Figura 1: Algoritmo secuencial para comprimir

Algoritmo 2 Algoritmo para Descomprimir en `RLE` (Modo Extendido)

```
procedure DESCOMPRIMIR(C)                         ▷ Cadena de símbolos comprimidos C
    N ← longitud(C)
    0 ← []                                        ▷ Cadena de salida descomprimida
    p ← 0
    while p < N_C do
        S ← C[p]                                  ▷ Leer el símbolo actual (byte de control o literal)
        if S = FLAG_RLE then
            if p + 2 ≥ N_C then
                break                             ▷ Error: Buffer incompleto
            end if
            i ← C[p + 1]                          ▷ Leer el conteo (longitud)
            V ← C[p + 2]                          ▷ Leer el valor del símbolo
            for k ← 1 hasta i do
                O ← O ∥ V                         ▷ Escribir el valor V el número de veces i
            end for
            p ← p + 3                             ▷ Avanzar el apuntador 3 posiciones (FLAG, Conteo, Valor)
        else if S = FLAG_LITERAL then             ▷ Símbolo FLAG escapado
            if p + 1 ≥ N_C then
                break                             ▷ Error: Buffer incompleto
            end if
            V ← C[p + 1]                          ▷ Leer el siguiente byte (que es un FLAG real)
            O ← O ∥ V                             ▷ Escribir el byte escapado
            p ← p + 2                             ▷ Avanzar 2 posiciones (FLAG_LITERAL, Valor)
        else                                      ▷ Símbolo sin repetición
            O ← O ∥ S                             ▷ Escribir el símbolo literal actual S
            p ← p + 1                             ▷ Avanzar 1 posición
        end if
    end while
    Salida: O
end procedure
```

Figura 2: Algoritmo secuencial para descomprimir

Algoritmo 3 Algoritmo para comprimir en `RLE` en paralelo

procedure COMPRIMIR_PARALELO(S,N) ▷ S entrada y N número de procesos
▷ Lógica del Coordinador (Pre-procesamiento)
    $Global\_Size \leftarrow longitud(S)$
    for $id \leftarrow 0 \dots N-1$ **do**
        OffsetStart
        $SizeChunk \leftarrow calculate\_chunk(id, Global\_Size, S)$
        $Chunkid \leftarrow S[OffsetStart \dots OffsetStart + SizeChunk + 1]$
▷ Incluye 1 byte de frontera
        Enviar($Chunkid$) a N_id
    end for
    $Sincronizar()$
▷ Lógica del Trabajador
    $\text{Input}_{\text{T}} \leftarrow$ **Recibir**(Chunk_{id})
    $\text{Input}_{\text{Main}} \leftarrow \text{Input}_{\text{T}}$[datos principales]
    $\text{Byte}_{\text{Next}} \leftarrow \text{Input}_{\text{T}}$[byte de frontera]
    $\text{Output}_{\text{local}} \leftarrow$ Comprimir_Secuencial($\text{Input}_{\text{Main}}$) ▷ Usa RLE Extendido
▷ Corrección de Fronteras (Comunicación/Sincronización)
    if $id < N - 1$ **then**
        Enviar_Front($\text{Output}_{\text{local}}$.último_bloque) a $\text{Trabajador}_{\text{id+1}}$
        Recibir_Front($\text{Señal}_{\text{Fus}}$) de $\text{Trabajador}_{\text{id+1}}$
        if $\text{Señal}_{\text{Fus}}$ = FUSIONADO **then**
            $\text{Output}_{\text{local}}$.Eliminar_último_bloque()
        end if
    end if
    if $id > 0$ **then**
        Recibir_Front($\text{Bloque}_{\text{Prev}}$) de $\text{Trabajador}_{\text{id}-1}$
        Intentar_Fusion($\text{Output}_{\text{local}}$.primero, $\text{Bloque}_{\text{Prev}}$)
        Enviar_Front($\text{Señal}_{\text{Fus}}$) a $\text{Trabajador}_{\text{id}-1}$
    end if
    Enviar($\text{Output}_{\text{local}}$) al Coordinador
    Sincronizar()
▷ Lógica del Coordinador (Post-procesamiento)
    Recibir_Buffers(Todos los trabajadores)
    $\text{Len}_{\text{Acumulada}} \leftarrow$ Calcular_Suma_Prefijo(Longitudes_locales)
    $O_{\text{comp}} \leftarrow$ Concatenar(Buffers recibidos, $\text{Len}_{\text{Acumulada}}$)
    Salida: O_{comp}
end procedure

Figura 3: Algoritmo paralelo para comprimir

Algoritmo 4 Algoritmo para descomprimir en `RLE` en paralelo

procedure DESCOMPRIMIR_PARALELO(C,N) ▷ C entrada comprimida y N número de procesos
▷ Lógica del Coordinador (Pre-procesamiento)
  $Par_{count} \leftarrow read\_Metadato(C)$ ▷ Número total de pares/bloques
  for $id \leftarrow 0 \dots N-1$ **do**
    $Offset_{par}, Size_{par} \leftarrow Calcular\_Particion\_Pares(id, Par_{count})$
    $Chunk_{id} \leftarrow C[Offset_{par} \dots Offset_{par} + Size_{par}]$ ▷ Lectura de porción de pares
    Enviar($Chunkid$) a N_id
  end for
  $Sincronizar()$
▷ Lógica del Trabajador
  $\mathrm{Input_T} \leftarrow$ **Recibir**($\mathrm{Chunk_{id}}$)
  $\mathrm{Output_{local}} \leftarrow$ Descomprimir_Secuencial($\mathrm{Input_T}$) ▷ Genera bytes sin comprimir
  $\mathrm{Len_{local}} \leftarrow$ longitud($\mathrm{Output_{local}}$)
▷ Cálculo del Offset Global (Suma Prefijo)
  Enviar($\mathrm{Len_{local}}$) a Concentrador
  Sincronizar()
  Recibir(Todas_$\mathrm{Len_{local}}$)
  $\mathrm{Offset_{Final}} \leftarrow$ Calcular_Suma_Prefijo(Todas_$\mathrm{Len_{local}}$)[id]
▷ Cada trabajador conoce dónde escribir
  Escribir_en_Posicion_Global($O_{\mathrm{descomp}}, \mathrm{Offset_{Final}}, \mathrm{Output_{local}}$)
▷ Escritura dispersa
  Sincronizar()
▷ Lógica del Coordinador (Post-procesamiento)
  Finalizar_Archivo(O_{descomp})
  Salida: O_{descomp}
end procedure

Figura 4: Algoritmo paralelo para descomprimir