Compresión de datos sin pérdida
Preparación de tareas de programación

1. Objetivos

Esta tarea consiste en implementar algunas herramientas de software que van a ser de utilidad en el desarrollo de futuras tareas específicas sobre compresión de datos. Los objetivos que se persiguen son los siguientes:

1. Refrescar o familiarizarse con el uso de los lenguajes de programación C/C++.
2. Llevar a la práctica técnicas de programación usuales en Compresión de Datos, que requieren manejar con soltura la representación interna de tipos de datos y el uso eficiente de recursos de computación.
3. Generar herramientas que sean de utilidad para el desarrollo de las tareas de programación de esta asignatura.

2. Programación

Una de las tareas que usualmente hay que resolver cuando se desarrolla un compresor o un descompresor es la escritura y lectura de bits en un archivo (o algún otro mecanismo de entrada / salida). En general, la unidad mínima de intercambio de datos en las interfaces estándar de entrada y salida es el byte. Sin embargo, para las aplicaciones que nos interesan, es común que necesitemos escribir y leer bits individualmente, o agrupados en paquetes de un tamaño que no es necesariamente múltiplo de 8. Por este motivo es conveniente desarrollar módulos de entrada y salida de datos que provean interfaces más cómodas para resolver esta tarea.

Las interfaces que se sugieren a continuación son una guía; esta base se puede extender (por ejemplo para manejar situaciones de error) o modificar a criterio de cada estudiante.

2.1. Interfaz de escritura

Para la escritura de bits se sugiere implementar una clase en C++ (o una estructura y funciones equivalentes en C), que representa un archivo para acceso de escritura con métodos similares a los siguientes:

- `Abrir(char *fileName)`: Abre un archivo para escritura; si existe un archivo con el mismo nombre, se reemplaza, en caso contrario, se crea.
- `Escribir(unsigned x, unsigned k)`: Escribe los `k` bits menos significativos de `x` en el archivo.
- `Cerrar()`: Escribe cualquier dato que esté pendiente de escritura (en buffer) y cierra el archivo.

2.2. Interfaz de lectura

Para la lectura de bits los métodos podrían ser similares a los siguientes:

- `Abrir(char *fileName)`: Abre un archivo para lectura.
- `Leer(unsigned *x, unsigned k)`: Lee `k` bits del archivo y los coloca en los `k` bits menos significativos de `x`. Los restantes bits de `x` son puestos en cero.

- `Fin()`: Devuelve `verdadero` si ya fueron leídos todos los bits del archivo.
- `Cerrar()`: Cierra el archivo.

3. Pruebas sugeridas

Antes de usar estas herramientas para la construcción de programas más elaborados, es conveniente probarlas extensamente para convencerse de su correcto funcionamiento. Algunas pruebas sugeridas son las siguientes:

- Preparar un archivo de **texto**, con una secuencia de caracteres '0' y '1'. Escribir un programa que lee este archivo de texto y genera otro archivo **binario** con esa secuencia de ceros y unos escribiendo los bits de a uno. Verificar que el tamaño del archivo resultante es 1/8 del tamaño original a menos de redondeo. Escribir un programa que lee el archivo binario y lo convierte a un archivo de texto; comparar con el archivo de texto original.
- Repetir el procedimiento anterior pero escribiendo y leyendo bloques de bits de largo mayor que 1.
- Usar las interfaces de lectura y escritura construidas para implementar un programa que haga una copia de un archivo arbitrario. A lo largo del proceso de copia, leer y escribir bloques de bits de largos variados. Procure probar casos de borde como por ejemplo leer y escribir una cantidad de bits que coincide con `8*sizeof(unsigned)`.

4. Entrega

No es necesario entregar esta tarea para su evaluación. En función del cronograma previsto para el curso, sin embargo, se sugiere completarla antes del 16 de abril.