Códigos para Corrección de Errores
Proyecto de implementación

Consideraciones generales.

- El trabajo consiste en la implementación, en software, de algoritmos de codificación y decodificación de códigos Reed-Solomon o relacionados. Junto al desarrollo del software, se resolverán algunos problemas teóricos relacionados a los códigos y los algoritmos, y se realizarán experimentos que permitan por un lado verificar que el software funciona como se espera, y por otro entender el desempeño práctico de los códigos.

- Los programas deberán cumplir con los requisitos siguientes:

  - El lenguaje de programación debe ser C o C++.
  - Los experimentos solicitados en el trabajo deben ser compilables, ejecutables, y reproducibles por los docentes en plataformas de computación genéricas (PC con Windows o Linux, Mac, etc.), sin requerir paquetes especiales. La entrega debe incluir instrucciones claras de cómo compilar y ejecutar los programas (incluyendo archivos makefile, proyectos de MS Visual Studio, etc., apropiados a la plataforma).
  - Los programas deben ser eficientes y permitir la realización de los experimentos requeridos, procesando millones de bloques de código en tiempo razonable (segundos, no horas). Una vez depurado el programa, y para realizar los experimentos, recuerde compilar con máxima optimización del tiempo: opción -O3 con gcc/g++en Linux/Mac, opción /Ot o versión Release con Visual en Windows. No utilice la versión Debug (u opción -g en Linux/Mac) en los experimentos.

- En la letra del proyecto se especifica la funcionalidad mínima requerida. Decisiones de diseño que no estén definidas explícitamente quedan a criterio cada estudiante. Mejoras en funcionalidad, si son interesantes y efectivas, recibirán puntaje extra.

- El trabajo de evaluación debe ser concebido e implementado en su totalidad por el grupo de estudiantes evaluado.

  - **Está prohibido** bajar soluciones de cualquier tipo, totales o parciales, de internet, o copiarlas de cualquier otra fuente, incluyendo ejercicios similares en ediciones anteriores del curso.

  Tenga en cuenta que transgresiones a esta prohibición son relativamente fáciles de identificar (existen herramientas sofisticadas de dominio público que permiten detectar con gran confiabilidad la copia, con o sin alteraciones, de trabajos existentes, sea de internet o de otras fuentes).

- Se permite utilizar el resultado de un inciso teórico en un inciso posterior, aunque no se haya podido demostrar el resultado.

- El funcionamiento correcto de los programas es condición necesaria para la aprobación del proyecto.
- La entrega del proyecto se hará en forma electrónica, y consistirá en:

    1. Una copia del enunciado (este archivo).
    2. Los archivos con el código fuente de los programas, de manera que puedan ser compilados y ejecutados independientemente. Incluya instrucciones detalladas de compilación y ejecución, mencionando el sistema operativo y compilador con que se desarrolló el programa.
    3. Un informe conteniendo documentación del código entregado, los resultados de los experimentos, incluyendo tablas y gráficos, así como las respuestas y desarrollos correspondientes a las preguntas teóricas.
    4. Junto con los resultados, incluir en el documento copias exactas de las líneas de comando de los utilitarios empleados. En el caso de utilitarios que utilicen números seudo-aleatorios (p.ej. simuladores de canal), incluir en el informe las semillas utilizadas por los generadores. Todos los experimentos deben ser reproducibles.

Códigos para Corrección de Errores
Proyecto de implementación

Códigos Reed-Solomon en canales con errores y borraduras y errores aleatorios

Sea $m = 8$, y $q = 2^m$. Se implementará un decodificador para códigos Reed-Solomon (acortados) $[n, k, n-k+1]$ en GF(q), de largo n y redundancia ($r = n-k$) variables. También se investigará la capacidad correctora de estos códigos en canales que introducen borraduras y errores aleatorios.

Formato de los datos.

Los archivos de prueba utilizados en este proyecto serán binarios, interpretados como secuencias de símbolos de m bits c/u. Cada símbolo se almacena en un octeto. Las secuencias de símbolos se particionan a su vez en bloques, que pueden ser mensajes de largo k símbolos (entrada al codificador) o palabras de código (a veces corruptas) de largo n símbolos (salida del codificador, entrada y salida del canal, entrada al decodificador). En todos los casos se debe utilizar un número entero de bloques.

1. Representación de GF(q)

Se representa el cuerpo GF(q) como extensión de GF(2) utilizando el polinomio irreducible y primitivo $x^8 + x^4 + x^3 + x^2 + 1$. Sea α una raíz de este polinomio. Los elementos de GF(q) se representan en binario según la base $\{1, \alpha, \alpha^2, \ldots, \alpha^7\}$. A la vez, al ser α primitivo, se utilizará para generar tablas de logaritmos y antilogaritmos discretos, que facilitarán la implementación de la multiplicación, inversión, y división en GF(q). Estas tablas (`gflog` y `gfalog`) estarán disponibles en el sitio EVA del curso junto a macros implementando las operaciones aritméticas, en formato compilable en C/C++. Por ejemplo, `gfalog[i]` contiene la representación binaria de α^i, y si interpretamos dicha representación binaria como un número entero `a`, `gflog[a]` contiene al entero i (el logaritmo de α^i en base α).

2. Codificador y canal

1. Se utilizará un codificador sistemático de códigos RS convencionales, de largo n y redundancia r especificados como parámetros del programa. El programa toma un archivo arbitrario de entrada, en el formato arriba descrito, y produce un archivo codificado de salida. El codificador obedece las convenciones siguientes:

   - El codificador implementado es el descrito en los slides del curso titulados "Systematic encoding of RS codes" y "Systematic encoding circuit".
   - El polinomio generador del código RS es

$$g(x) = \sum_{j=0}^{r} g_j x^j = (x - \alpha^b) \cdot (x - \alpha^{b+1}) \cdots (x - \alpha^{b+r-1}),$$

donde α es el elemento primitivo utilizado para generar las tablas de logaritmos/antilogaritmos mencionadas arriba. Se elige $b = 1$ (código "narrow-sense").

- Palabras de código
$$\mathbf{c} = [c_1\, c_2\, \ldots\, c_n],$$
que se interpretan como polinomios $c(x) = \sum_{i=0}^{n-1} c_{i-1} x^i$, se almacenan en el archivo codificado en orden *decreciente* de índice, o sea c_n primero, seguido de c_{n-1}, etc., terminando en c_1 (la misma convención se utiliza para los bloques de mensaje en la entrada al codificador).

Un utilitario `rsencode` implementando el codificador estará disponible en el sitio EVA del curso. Corra `rsencode` sin argumentos para obtener un listado de las opciones disponibles.

2. Se utilizará un simulador de canal ruidoso, que produce borraduras y errores de símbolo aleatorios. El canal recibe como entrada una secuencia de símbolos de GF(q), y saca como salida una secuencia de símbolos en GF(q) $\cup$ {?}, donde ? indica un símbolo borrado. Sean δ y ρ parámetros reales que satisfacen $0 \le \delta < 1$ y $0 \le \rho < 1-\delta$. El canal actúa sobre los símbolos de entrada en forma independiente según las probabilidades de transición siguientes:
$$P(\text{salida} = b \,|\, \text{entrada} = a) = \begin{cases} 1-\delta-\rho & b = a \ \text{(sin corrupción)} \\ \rho & b = ? \ \text{(borradura)} \\ \delta/(q-1) & b \neq a,\ b \neq ? \ \text{(error)} \end{cases}$$
Note que la probabilidad de que un símbolo no se borre y salga del canal con error es δ, y esta probabilidad se divide uniformemente entre los símbolos erróneos posibles. Nos referimos a este canal como REEC(δ, ρ) (random errors and erasures channel). Los parámetros δ, ρ se especifican al correr el simulador. La salida del simulador consiste en dos archivos:

   - un archivo de símbolos posiblemente corruptos que resulta del original al pasar por el canal, donde a los símbolos borrados se les asigna el valor 0,
   - un archivo de bytes que caracteriza los lugares de símbolos borrados; este archivo contiene un byte por cada símbolo original, con valor 1 si el símbolo fue borrado, o 0 si no lo fue. Este archivo le permite al decodificador identificar los símbolos borrados.

   Un utilitario `reechannel` implementando el simulador estará disponible en el sitio EVA del curso.

   Ejemplo. Supongamos que tenemos un archivo `codificados.rse` de datos ya codificados con un código RS sobre GF(256), y queremos simular su pasaje por el canal REEC, con parámetros $\delta = 0{,}01$, y $\rho = 0{,}02$. La línea de comando para el utilitario sería

```
reechannel --fieldsize 256 --delta 0.01 --rho 0.02 codificados.rse
```

El utilitario produce los archivos de salida `codificados.ech` (símbolos corruptos) y `codificados.eras` (indicadores de borradura), cuyos nombres por defecto son derivados del nombre del archivo de entrada. Corra '`reechannel -h`' para obtener un listado completo de las opciones disponibles.

Sugerencia. El utilitario tiene un modo adicional de operación que permite generar archivos de símbolos aleatorios de GF(q). Ejemplo:

```
reechannel -q 256 -n 42 -R 1000000 misdatos.dat
```

genera un archivo de 1 millón de bloques de largo 42 con símbolos uniformemente distribuidos en GF(256). Este archivo puede usarse como entrada al codificador.

3. Se utiliza un código RS de parámetros $[n, n-r, r+1]$. Demuestre que la probabilidad de que un bloque codificado que pasó por el canal REEC(n, δ, ρ) *no sea* decodificado correctamente por un decodificador convencional para errores y borraduras está dada por

$$p_{\text{block}} = 1 - \sum_{s=0}^{r} \binom{n}{s} \rho^s \sum_{t=0}^{\lfloor (r-s)/2 \rfloor} \binom{n-s}{t} \delta^t (1-\rho-\delta)^{n-s-t}$$

Un utilitario `reepblock`, que calcula p_{block} en función de n, r, δ, ρ estará disponible en el sitio EVA del curso.

3. Decodificador

4. Implemente un decodificador correspondiente al codificador de la sección 1, para un código RS $[n, n-r, r+1]$ con n y r especificados como parámetros del programa (siendo $0 < r < n < q$). El decodificador debe corregir todos lo patrones de borraduras y errores posibles para el valor de redundancia r elegido, con un algoritmo de decodificación convencional para códigos RS.

   El programa recibe como entrada los archivos de salida del simulador de canal (que a su vez recibe como entrada un archivo codificado con el codificador de la sección 1), y produce un archivo decodificado. Más detalles:

   - La línea de comando para el programa deberá ser similar a

     ```
     rsdecode n r symbolfile erasfile
     ```

     donde `n` es el largo de código, `r` es la redundancia, y `symbolfile` y `erasfile` son los archivos de entrada (salidas del simulador de canal). El nombre del archivo de salida puede agregarse a la línea de comando, o derivarse del nombre de la entrada, según su preferencia. El archivo decodificado *debe contener solamente los símbolos de mensaje*. Si se prefiere, se pueden usar banderas p.ej. `-n, -r` para los parámetros del código, como lo hace `rsencode`.
   - El programa debe leer la entrada y escribir la salida bloque por bloque. *No leer todo el archivo de entrada de una vez.*

- Si se llama sin argumentos, o los argumentos de entrada son incorrectos, el programa debe imprimir un mensaje de error y breves instrucciones de uso, y terminar sin estrellarse. Se debe verificar que los valores de los argumentos numéricos estén en rangos correctos.
- En el programa, siempre que sea posible, verifique que las hipótesis de trabajo se cumplen (p.ej., verifique que el número de raíces del polinomio localizador de errores, Λ, es igual a su grado). Cuando se detecta una inconsistencia, se da el bloque por incorregible, y se copia el mensaje de entrada sin modificar. Nos referimos a este caso como una *falla de decodificación*.
- El programa debe acumular, e imprimir al terminar, estadísticas de bloques clasificados en tres categorías: sin errores, con errores que se corrigieron, y fallidos (todo desde el punto de vista del decodificador).

4. Experimentos de decodificación

5. Fijamos los parámetros del código RS: $n = 48$, $r = n - k = 6$. Utilizando los utilitarios provistos, y su decodificador, corra experimentos de codificación/corrupción/decodificación en el canal REEC, con parámetros

$$\rho \in \{0{,}02, 0{,}03\}$$
$$\delta \in \{1e{-}5, 1e{-}4, 1e{-}3, 1e{-}2, 1e{-}1\}\,.$$

   El archivo de entrada en los experimentos debe ser suficientemente grande para minimizar efectos significativos de fluctuaciones estadísticas (se recomienda por lo menos un millón de bloques—el largo del archivo de entrada al codificador debe ser múltiplo de k).

   Se utilizará un programa utilitario que permite comparar archivos originales con archivos decodificados, y generar estadísticas de errores de bloques, símbolos, y bits. Este utilitario (`diffblock`) también estará disponible en el sitio EVA del curso. Corra `diffblock` sin argumentos para obtener un listado de las opciones disponibles. Utilizando este utilitario `diffblock`, calcule las probabilidades p_{block} *empíricas* en sus experimentos. Grafique p_{block} (teórico y empírico) en función de δ, para cada uno de los valores de ρ, todas las curvas en la misma figura. Utilice escala logarítmica en los dos ejes. Discuta los resultados. En particular,

   - Comente sobre la coincidencia (o no) de los valores empíricos con los teóricos.
   - Discuta el comportamiento general de p_{block} vs. δ para ρ fijo.
   - En las gráficas, p_{block} se nivela cuando δ tiende a cero. Explique este comportamiento.
   - Compare el conteo de bloque fallidos reportado por su decodificador con el reportado por `diffblock`. Comente si nota diferencias, y explíquelas. En la comparación, se recomienda utilizar *diferencias relativas*. P.ej., si estamos comparando N_1 con N_2, se recomienda tabular N_1, N_2, y $(N_1 - N_2)/N_2$.