ICOM2021 – Examen Final

13 de diciembre de 2021

Notas:

1. Uso de prácticos: **se pueden utilizar los trabajos prácticos propios realizados.**

Problema 1: Imágenes monocromáticas y codificación RLE

Una imagen monocromática es una imagen en donde solo pueden aparecer dos colores. Una manera de almacenar en memoria imágenes bidimensionales de este tipo es utilizando mapas de bits para cada fila, es decir, representar los colores de los pixeles de una fila de la imagen como una secuencia de bits, interpretando que un bit en 0 (cero) representa un color y un bit en 1 el color restante.

Suponiendo, para simplificar, que se está trabajando en una plataforma en donde un entero sin signo ocupa 4 bytes (32 bits), para una imagen de **`n`** filas por **`m`** columnas, el mapa de bits de cada fila se implementa como un arreglo de **`(m+31)/32`** enteros sin signo.

Así, el color del pixel en la posición **`X`** de la fila **`Y`**-ésima de la imagen está representado por el bit **`(X % 32)`**-ésimo del entero que está en el índice **`(X / 32)`** del vector de enteros que representa la fila **`Y`**-ésima de pixeles.

Dadas estas definiciones, se solicita diseñar el UDT **`MonoImg`** para representar una imagen monocromática completando el esqueleto presente en **`mono_img.cpp`**.

Por otro lado, existen diferentes técnicas de compresión de datos que apuntan a reducir más el volumen de información que describe a una imagen cuando esta es almacenada en un archivo. Una de tales técnicas es la **RLE** (**R**un-**L**ength-**E**ncoding) que se basa en representar a secuencias de pixeles consecutivos con el mismo color (repetido), como un único valor seguido por el número de repeticiones presentes en esa secuencia.

Suponga que se cuenta con una imagen comprimida, almacenada en un archivo, cuyo formato es:

```
ancho alto nruns
rl1 rl2 rl3 rl4 …
```

en donde **`ancho`** y **`alto`** definen las dimensiones de la imagen, **nruns** es la cantidad de runs que definen la imagen, y los **`rlx`** son los run-length codes. Estos se encuentran codificados como caracteres sin signo (8 bits sin signo) de tal manera que el bit 7 representa el color (**`color = (rlx >> 7)`**) y el número de repeticiones de ese color está codificado en los 7 bits menos significativos (**`nrep = rlx & 0x7F`**).

En la compresión, los runs se generaron recorriendo la imagen fila por fila, de izquierda a derecha, de arriba hacia abajo y se define que un run solo codifica píxeles de una fila.

Se solicita implementar la función:

```
MonoImg *decompress(const string &filename);
```

que descomprima la imagen codificada con **RLE** en el archivo dado.

Probar con el **`main`** y las funciones auxiliares implementadas en **`mono_img.cpp`**.

Problema 2: Incapacidad Artificial

El perceptrón multicapa es una red neuronal artificial (**RNA**) que tiene capacidad para resolver problemas que no son linealmente separables. Las capas pueden clasificarse en tres tipos:

- Capa de entrada: Constituida por aquellos nodos que introducen los patrones de entrada en la red. En estos nodos no se produce procesamiento.
- Capas ocultas: Formada por aquellos nodos cuyas entradas provienen de capas anteriores y cuyas salidas pasan a nodos de capas posteriores.
- Capa de salida: nodos cuyos valores de salida se corresponden con las salidas de toda la red, sus entradas provienen de la última capa oculta.

Un perceptrón multicapa puede estar totalmente o localmente conectado.

En el primer caso, cada salida de un nodo (`j`) de una capa `i` (`out_i,j`) es entrada (`in_i+1,k`) de todos los nodos (`k`) de la capa `i+1`, pesada por un arreglo de pesos `W_(k,j)`. Más allá de la arquitectura particular de la **RNA** en cuestión, para poder cumplir alguna función se requiere de un aprendizaje previo, donde se compara la predicción de la red, frente al valor esperado. El aprendizaje se logra modificando los pesos `W_(k,j)` entre las distintas conexiones, mediante un algoritmo llamado "**backpropagation**". Una vez que la red ha sido suficientemente entrenada, los pesos obtenidos se pueden utilizar para calcular un resultado en base a datos de entrada que presenten un formato adecuado, por más que nunca hayan sido utilizados durante el proceso de aprendizaje.

En el archivo `perceptron_tester.cpp` encontrará el esqueleto de un programa que permite comprobar el correcto funcionamiento de **RNA**s multicapa totalmente conectadas, con arquitecturas arbitrarias*.

Deberá completar todos los **TODO** para que el mismo funcione correctamente.

Para que pueda probar esto, se le facilitará un archivo de texto con una **RNA** entrenada previamente para llevar a cabo una tarea extremadamente simple: decidir cuál es el mayor entre dos números cuyos valores se encuentren entre 0 y 1.

Tanto la arquitectura como los pesos entrenados se encuentran guardados en este archivo: **RNA.txt**. Esta red en particular consta de 4 capas totalmente conectadas: entrada (2 nodos), primera capa oculta (3 nodos), segunda capa oculta (3 nodos), salida (1 nodo).

*tenga en cuenta que para que esta afirmación fuera realmente cierta, el método `run()` no debería ser tan específico, pero para no complicar el código se decidió dejarlo así.

Problema 3: Guardería de Animales

Se está diseñando un conjunto de clases para manejar en forma básica el inventario de animales que posee la guardería, así como un soporte mínimo para calcular los requerimientos de alimentos para las distintas especies.

Se solicita, utilizando el esqueleto en `guarderia.cpp`, completar la implementación, probándolo con el `main` dado.