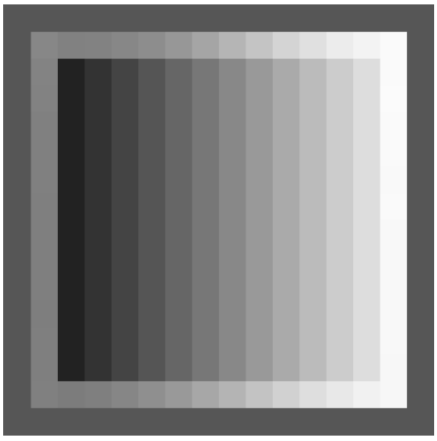
Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Proyecto del curso, 2019-10

# 1. Descripción del problema

Una imagen es la representación visual de la apariencia de un objeto o escena. Mucha de la información que utilizamos y procesamos diariamente se presenta en forma de imágenes, como ilustraciones, diagramas, mapas, pinturas, fotografías, y otras representaciones pictóricas. En la actualidad, existen diferentes formas de generar imágenes digitales (para ser representadas por el computador), como cámaras fotográficas, escáneres, equipos especializados, programas de dibujo y diseño, entre otros.

Computacionalmente hablando, una imagen corresponde a un arreglo bidimensional de pixeles (*picture elements*, mínima unidad de información de una imagen), los cuales tienen valores de intensidad de color asociados y juntos conforman la correspondiente representación visual. Para imágenes en escalas de gris, comúnmente se utiliza un byte para representar la intensidad de cada píxel, así se obtienen 256 valores diferentes de intensidad, desde 0 (negro) hasta 255 (blanco), como se muestra en la Figura 1.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 |
| 86 | 135 | 129 | 130 | 135 | 141 | 151 | 165 | 181 | 196 | 212 | 224 | 236 | 243 | 250 | 86 |
| 86 | 131 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 250 | 86 |
| 86 | 130 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 251 | 86 |
| 86 | 128 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 250 | 86 |
| 86 | 128 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 250 | 86 |
| 86 | 128 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 249 | 86 |
| 86 | 127 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 250 | 86 |
| 86 | 127 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 248 | 86 |
| 86 | 127 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 248 | 86 |
| 86 | 127 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 248 | 86 |
| 86 | 126 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 248 | 86 |
| 86 | 127 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 248 | 86 |
| 86 | 127 | 34 | 51 | 68 | 85 | 102 | 119 | 136 | 153 | 170 | 187 | 204 | 221 | 247 | 86 |
| 86 | 128 | 124 | 127 | 134 | 143 | 153 | 167 | 180 | 195 | 209 | 222 | 232 | 241 | 247 | 86 |
| 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 |

Figura 1: Representación computacional de imágenes en escala de grises

En el caso de imágenes en color (conocidas como imágenes RGB), se utilizan 3 de estos arreglos bidimensionales, cada uno representando la intensidad de los 3 colores base: rojo (R), verde (G) y azul (B). La combinación de estas tres intensidades genera para cada píxel un color entre 16 millones posibles ($256^3$), aunque el ojo humano es capaz apenas de discriminar hasta 10 millones de colores diferentes[1]. Un ejemplo de la representación computacional de estas imágenes se presenta en la Figura 2.

Una vez generadas con diferentes técnicas y tecnologías, estas imágenes pueden ser manipuladas en el computador con algoritmos específicos para resaltar, extraer o generar nueva información que pueda llegar a ser de interés. Estas herramientas se conocen con el nombre de algoritmos de procesamiento de imágenes. El área de procesamiento de imágenes abarca entonces todo el conjunto de técnicas computacionales especializadas que permiten extraer y analizar la información visual contenida en las imágenes.

# 2. Descripción del proyecto

El objetivo del presente proyecto es construir un sistema que incluya algunos componentes para el procesamiento de imágenes en escala de grises, utilizando algunos conceptos de Estructuras de Datos vistos a lo largo del semestre.

[1] http://en.wikipedia.org/wiki/Color_depth

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 255 | 255 | 255 | 252 | 177 | 116 | 110 | 122 | 146 | 178 | 211 | 245 | 255 | 255 | 255 | 255 |
| 255 | 255 | 222 | 86 | 61 | 124 | 170 | 196 | 211 | 219 | 224 | 230 | 243 | 254 | 255 | 255 |
| 255 | 223 | 59 | 88 | 153 | 166 | 158 | 174 | 199 | 227 | 250 | 251 | 245 | 251 | 255 | 255 |
| 249 | 85 | 87 | 157 | 110 | 118 | 151 | 182 | 209 | 236 | 254 | 255 | 252 | 251 | 250 | 255 |
| 173 | 56 | 151 | 108 | 122 | 141 | 163 | 196 | 223 | 248 | 255 | 255 | 255 | 252 | 244 | 252 |
| 87 | 103 | 155 | 111 | 140 | 161 | 179 | 207 | 238 | 255 | 255 | 255 | 255 | 255 | 248 | 239 |
| 48 | 130 | 111 | 121 | 149 | 175 | 199 | 224 | 250 | 255 | 255 | 255 | 255 | 255 | 251 | 230 |
| 24 | 146 | 100 | 123 | 152 | 181 | 211 | 239 | 255 | 255 | 255 | 255 | 255 | 255 | 251 | 222 |
| 27 | 144 | 98 | 118 | 147 | 176 | 202 | 229 | 253 | 255 | 255 | 255 | 255 | 255 | 246 | 219 |
| 50 | 128 | 108 | 107 | 133 | 158 | 179 | 208 | 241 | 255 | 255 | 255 | 255 | 255 | 236 | 221 |
| 97 | 96 | 156 | 91 | 113 | 132 | 155 | 191 | 224 | 251 | 255 | 255 | 255 | 253 | 221 | 231 |
| 193 | 15 | 142 | 106 | 91 | 107 | 137 | 173 | 205 | 237 | 255 | 255 | 255 | 236 | 208 | 248 |
| 255 | 109 | 68 | 146 | 111 | 88 | 121 | 155 | 188 | 218 | 247 | 255 | 241 | 207 | 228 | 255 |
| 255 | 241 | 81 | 64 | 132 | 159 | 145 | 154 | 183 | 216 | 239 | 234 | 204 | 217 | 254 | 255 |
| 255 | 255 | 243 | 122 | 25 | 89 | 132 | 161 | 178 | 182 | 180 | 187 | 225 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 210 | 143 | 108 | 99 | 119 | 159 | 203 | 245 | 255 | 255 | 255 | 255 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 255 | 255 | 255 | 255 | 240 | 223 | 217 | 214 | 216 | 223 | 232 | 248 | 255 | 255 | 255 | 255 |
| 255 | 255 | 249 | 223 | 215 | 230 | 243 | 249 | 251 | 247 | 239 | 231 | 230 | 250 | 255 | 255 |
| 255 | 250 | 221 | 225 | 246 | 255 | 255 | 255 | 255 | 255 | 255 | 244 | 216 | 204 | 247 | 255 |
| 254 | 229 | 226 | 249 | 255 | 255 | 255 | 255 | 255 | 255 | 251 | 232 | 223 | 186 | 185 | 255 |
| 245 | 225 | 246 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 239 | 211 | 197 | 197 | 115 | 221 |
| 238 | 235 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 248 | 222 | 194 | 163 | 189 | 131 | 146 |
| 237 | 245 | 255 | 255 | 255 | 255 | 255 | 255 | 254 | 233 | 202 | 170 | 139 | 142 | 134 | 90 |
| 237 | 250 | 253 | 253 | 253 | 253 | 254 | 254 | 243 | 213 | 179 | 145 | 111 | 105 | 131 | 47 |
| 220 | 242 | 233 | 232 | 231 | 231 | 230 | 229 | 222 | 198 | 166 | 132 | 102 | 100 | 123 | 46 |
| 202 | 221 | 209 | 199 | 199 | 196 | 189 | 191 | 191 | 176 | 150 | 121 | 94 | 119 | 101 | 75 |
| 200 | 190 | 208 | 165 | 162 | 154 | 151 | 159 | 159 | 146 | 124 | 102 | 79 | 149 | 64 | 134 |
| 225 | 126 | 185 | 151 | 121 | 113 | 118 | 124 | 123 | 113 | 99 | 82 | 124 | 115 | 26 | 221 |
| 255 | 158 | 119 | 166 | 123 | 81 | 90 | 94 | 92 | 87 | 76 | 123 | 133 | 52 | 144 | 255 |
| 255 | 245 | 110 | 82 | 133 | 150 | 116 | 96 | 96 | 120 | 146 | 113 | 50 | 110 | 250 | 255 |
| 255 | 255 | 244 | 124 | 18 | 64 | 99 | 115 | 113 | 90 | 54 | 27 | 145 | 253 | 255 | 255 |
| 255 | 255 | 255 | 255 | 209 | 129 | 73 | 43 | 45 | 81 | 141 | 222 | 255 | 255 | 255 | 255 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 255 | 255 | 255 | 252 | 172 | 88 | 55 | 34 | 37 | 59 | 101 | 188 | 255 | 255 | 255 | 255 |
| 255 | 255 | 229 | 102 | 58 | 106 | 136 | 150 | 151 | 135 | 104 | 30 | 109 | 237 | 255 | 255 |
| 255 | 236 | 110 | 116 | 161 | 158 | 123 | 112 | 112 | 124 | 165 | 152 | 85 | 82 | 238 | 255 |
| 252 | 161 | 148 | 187 | 138 | 121 | 130 | 137 | 135 | 126 | 109 | 118 | 155 | 84 | 117 | 255 |
| 221 | 160 | 203 | 174 | 164 | 157 | 155 | 164 | 162 | 151 | 133 | 110 | 120 | 146 | 28 | 205 |
| 203 | 198 | 221 | 197 | 195 | 190 | 185 | 189 | 189 | 176 | 153 | 126 | 99 | 158 | 89 | 119 |
| 212 | 228 | 229 | 224 | 224 | 223 | 220 | 220 | 217 | 197 | 167 | 135 | 105 | 120 | 115 | 71 |
| 235 | 249 | 250 | 250 | 249 | 250 | 250 | 250 | 239 | 208 | 175 | 141 | 108 | 103 | 130 | 44 |
| 242 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 252 | 225 | 194 | 161 | 126 | 118 | 136 | 61 |
| 247 | 254 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 244 | 214 | 182 | 149 | 153 | 130 | 102 |
| 253 | 253 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 254 | 231 | 201 | 168 | 190 | 114 | 160 |
| 254 | 248 | 251 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 247 | 217 | 207 | 179 | 102 | 230 |
| 255 | 246 | 240 | 252 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 241 | 214 | 151 | 186 | 255 |
| 255 | 254 | 235 | 233 | 248 | 255 | 255 | 255 | 255 | 255 | 252 | 232 | 182 | 188 | 252 | 255 |
| 255 | 255 | 254 | 232 | 214 | 225 | 238 | 243 | 240 | 228 | 210 | 196 | 220 | 254 | 255 | 255 |
| 255 | 255 | 255 | 255 | 247 | 225 | 209 | 201 | 199 | 207 | 223 | 247 | 255 | 255 | 255 | 255 |

Figura 2: Representación computacional de imágenes en color (RGB)

## 2.1. Formato de imágenes

Las únicas entradas requeridas por el sistema son las imágenes a procesar. De acuerdo al componente, las entradas y salidas pueden ser una única imagen, o pueden ser varias imágenes. En cualquier caso, se utilizará siempre el formato PPM (Netpbm color image format)[2] para almacenar las imágenes. El archivo de imagen corresponde a un archivo de texto (caracteres en formato ASCII), con la siguiente estructura:

```
P3
# Algún comentario acerca del archivo...
W H
M
rI(1,1) gI(1,1) bI(1,1) rI(2,1) gI(2,1) bI(2,1) ... rI(W,1) gI(W,1) bI(W,1)
rI(1,2) gI(1,2) bI(1,2) rI(2,2) gI(2,2) bI(2,2) ... rI(W,2) gI(W,2) bI(W,2)
...
rI(1,H) gI(1,H) bI(1,H) rI(2,H) gI(2,H) bI(2,H) ... rI(W,H) gI(W,H) bI(W,H)
```

Donde:

- `P3` es un código (número mágico) que identifica el formato de la imagen (PPM en ASCII). En nuestro caso siempre utilizaremos el mismo código.
- Las líneas que empiezan con el caracter '#' son comentarios.
- `W` y `H` son números enteros positivos. Representan respectivamente el ancho y el alto de la imagen en pixeles.

[2] http://netpbm.sourceforge.net/doc/ppm.html

- `M` es un número entero positivo. Representa el valor de píxel más grande de la imagen, es decir, la intensidad máxima presente en la imagen. Este valor no debería ser mayor a 255.
- `rI(i,j)` es el valor de rojo del píxel de la imagen ubicado en la coordenada bidimensional `(i,j)`, donde $1 \leq i \leq W$ y $1 \leq j \leq H$. Este valor debe estar en el rango 0 a 255.
- `gI(i,j)` es el valor de verde del píxel de la imagen ubicado en la coordenada bidimensional `(i,j)`, donde $1 \leq i \leq W$ y $1 \leq j \leq H$. Este valor debe estar en el rango 0 a 255.
- `bI(i,j)` es el valor de azul del píxel de la imagen ubicado en la coordenada bidimensional `(i,j)`, donde $1 \leq i \leq W$ y $1 \leq j \leq H$. Este valor debe estar en el rango 0 a 255.

## 2.2. Componentes del sistema

A continuación se describen los componentes individuales que conforman el presente proyecto:

### 2.2.1. Componente 1: Proyección de imágenes

**Objetivo:** A partir de una serie ordenada de imágenes que conforman un volumen 3D, construir la proyección del volumen hacia un plano de acuerdo a una dirección y a un criterio de proyección.

La proyección de imágenes tridimensionales hace referencia a la generación de una representación planar (2D) de un objeto o escena 3D. Un tipo particular de proyección es comúnmente utilizado para la producción de radiografías. En este caso, el objeto a visualizar es bombardeado por una serie de rayos X, paralelos entre sí y perpendiculares a la superficie de captura. La cantidad de energía del rayo que alcanza a atravesar el objeto se refleja en la superficie de captura, donde queda plasmada la imagen de la proyección. Este proceso se ilustra en la Figura 3.

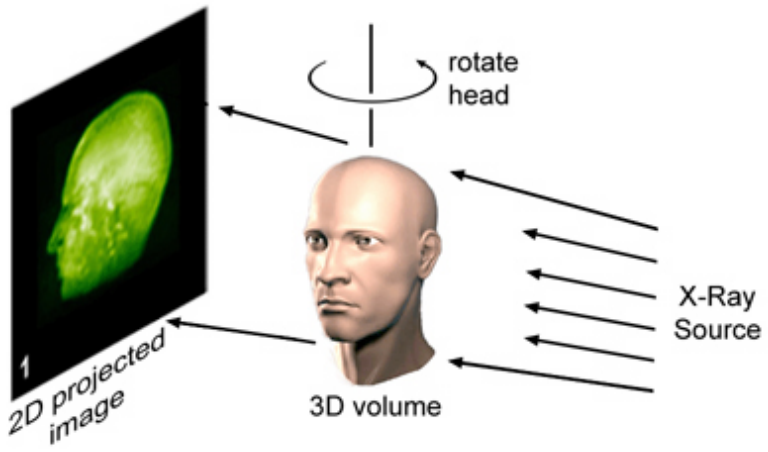



Figura 3: Ilustración del proceso de generación de una proyección planar por medio de rayos X (Tomado de `http://www.sv.vt.edu/classes/ESM4714/methods/FuncExt.html`)

De esta forma, el componente 1 del sistema corresponderá a una simulación del proceso de captura de radiografías. Para esto, se recibirá una serie ordenada de imágenes 2D, es decir, una conjunto de imágenes que juntas (en un orden específico) representan una escena 3D. Luego, se simulará el proceso de proyección analizando linealmente la información del volumen en una dirección específica, es decir, colapsando la información contenida en cada una de las filas del volumen en un sólo píxel. Esto generará una proyección 2D de la imagen volumétrica en la dirección especificada. El componente se implementará en los siguientes comandos:

- **comando:** `cargar_volumen <nombre_base> <n_im>`
  **salida en pantalla:**
  (proceso satisfactorio) `El volumen <nombre_base> ha sido cargado`
  (mensaje de error) `El volumen <nombre_base> no ha podido ser cargado`
  **descripción:** El comando debe cargar en memoria (en la estructura más adecuada) la serie ordenada de imágenes identificada con el `<nombre_base>` y cuyo tamaño corresponde a `<n_im>` imágenes (la serie podrá tener máximo 99 imágenes). Todas las imágenes de la serie deben estar nombradas como `<nombre_base>xx.ppm`, donde `xx` corresponde a dos dígitos de identificación de la posición de la imagen

en la serie (varía en el rango 01 - <n_im> ). Una vez cargada toda la información en memoria, el comando debe mostrar el mensaje de carga satisfactoria. Si por alguna razón no es posible cargar completamente la serie ordenada de imágenes (nombre de base erróneo, cantidad de imagénes no corresponde, error en alguna imagen), el comando debe mostrar el mensaje de error.

- **comando:** `proyeccion2D <dirección> <criterio> <nombre_archivo.ppm>`
  **salida en pantalla:**
  (proceso satisfactorio) `La proyección 2D del volumen en memoria ha sido generada`
  (mensajes de error)
  `El volumen aún no ha sido cargado en memoria`
  `La proyección 2D del volumen en memoria no ha podido ser generada`
  **descripción:** El comando debe tomar la serie ordenada de imágenes (ya cargada en memoria), y de acuerdo a la dirección especificada por el usuario, debe recorrer cada posición en el plano perpendicular a la dirección dada, y para cada una debe colapsar toda la información existente en la dirección dada utilizando el criterio especificado. Esto genera un sólo valor de píxel para cada posición del plano perpendicular, generando así una imagen 2D con la proyección de la información en el volumen. La dirección puede ser una entre `x` (en dirección de las columnas), `y` (en dirección de las filas) o `z` (en dirección de la profundidad). El criterio puede ser uno entre `minimo` (el valor mínimo de intensidad de cada canal de color), `maximo` (el valor máximo de intensidad de cada canal de color), `promedio` (el valor promedio de intensidad de cada canal de color) o `mediana` (el valor mediana de intensidad de cada canal de color). Una vez generada la proyección, debe guardarse como imagen en formato PPM como <nombre_archivo.ppm>. La Figura 4 ilustra el proceso de proyección en cada una de las posibles direcciones.

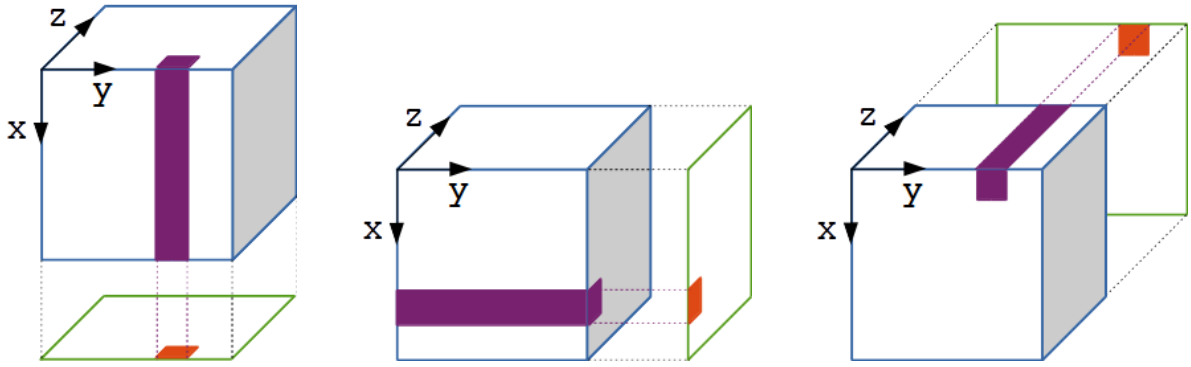



Figura 4: Ilustración del proceso de proyección 2D. La información del vector (en morado) se colapsa en un único pixel (en naranja). Izquierda: proyección en x. Centro: proyección en y. Derecha: proyección en z.

### 2.2.2. Componente 2: Codificación de imágenes

**Objetivo:** Utilizar el algoritmo de codificación de Huffman para comprimir y descomprimir imágenes.

Un principio muy importante alrededor de la compresión de datos es codificar cada símbolo de un mensaje usando la cantidad mínima posible de bits. Por ejemplo, si un mensaje estuviera escrito en lenguaje ASCII, el cual tiene 256 símbolos diferentes, la cantidad mínima de bits por símbolo requerida sería de 8. Otro principio esencial es que, aquellos símbolos que aparecen más frecuentemente en un mensaje, sería útil codificarlos con menos bits que aquellos menos frecuentes, de tal forma que el mensaje comprimido ocupe el menor espacio posible.

La codificación de Huffman[3] tiene en cuenta los dos principios anteriores. Provee una forma de representar cada símbolo de un mensaje con la menor cantidad posible de bits, y al mismo tiempo permite codificar cada símbolo con una cantidad variable de bits, dependiendo de su frecuencia de ocurrencia en el mensaje. Para realizar el proceso de codificación y decodificación, se utiliza un árbol de Huffman. Éste es un árbol binario que representa una codificación de un conjunto de símbolos óptima: el símbolo que tenga una frecuencia más alta (i.e. el que más se repita) se representa con un número pequeño de bits. Un símbolo poco frecuente se representa con más bits. Un ejemplo de un árbol de Huffman se muestra en la Figura 5.

[3] http://en.wikipedia.org/wiki/Huffman_coding

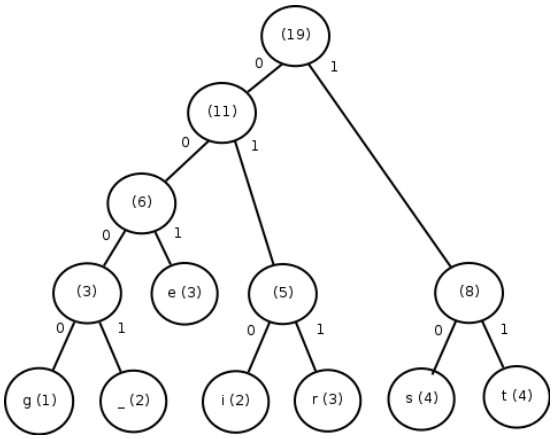



Figura 5: Árbol de Huffman.

En este árbol, cada nodo **hoja** almacena un símbolo del lenguaje utilizado en un mensaje y la cantidad de veces que se encuentra dicho símbolo en el mensaje (valor entre paréntesis), al cual llamaremos "frecuencia". Los **nodos intermedios y la raíz** no contienen símbolos, sino sólo un valor correspondiente a la suma de las frecuencias de sus nodos hijos. Note que los símbolos se encuentran exclusivamente en las hojas y cada símbolo aparece una sola vez en todo el árbol (no se repiten en el árbol).

Este árbol se utiliza para codificar y decodificar los símbolos de un mensaje. Cada arista entre un nodo padre y sus hijos se etiqueta con un "0" para el hijo izquierdo y con un "1" para el hijo derecho. Cada camino entre el nodo raíz y las hojas se puede representar como la concatenación de las etiquetas de las aristas que lo conforman. Esta representación corresponde a la codificación para cada uno de los símbolos. En este caso, por ejemplo, el símbolo que más se repite es la "t" (4 veces), y su codificación es "11", el cual corresponde a las etiquetas de las aristas del camino desde la raíz hasta el nodo con el símbolo "t"

Otros ejemplos de codificación son:

- La "s" se codifica como "10".
- El "_" se codifica como "0001".
- La "r" se codifica como "011".

Por ejemplo, la siguiente matriz de datos (imagen donde cada fila empieza y termina con el símbolo '+'):

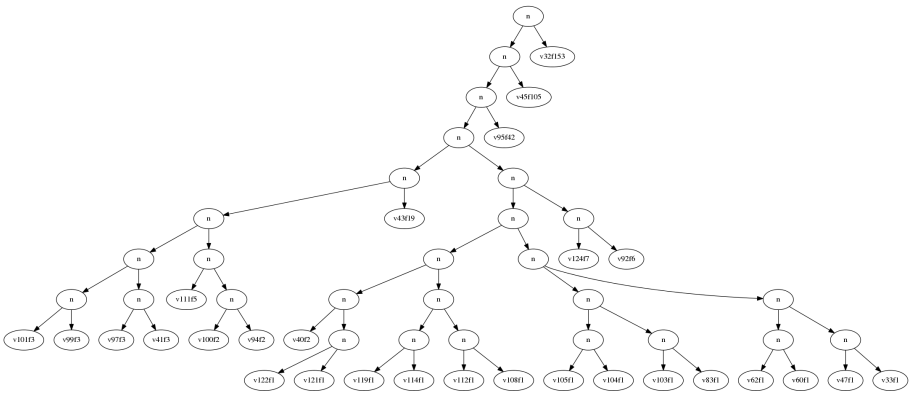
```
+-----------------------------------+
+ ______________________________   +
+ < Soy capaz de hacer el codigo! > +
+ ------------------------------   +
+         \   ^__^                  +
+          \  (oo)\_______          +
+             (__)\       )\/\      +
+                 ||----w |         +
+                 ||     ||         +
+-----------------------------------+
```

Tiene las siguientes características:

- Ancho de la imagen (W) = 36.
- Alto de la imagen (H) = 10.
- Valor (intensidad) máximo posible de la imagen (M) = 255.
- Los caracteres (que hacen las veces de intensidades) y sus frecuencias son:
  - `ASCII = 32, caracter = ' ', frecuencia = 153`
  - `ASCII = 33, caracter = '!', frecuencia = 1`
  - `ASCII = 40, caracter = '(', frecuencia = 2`

- ASCII = 41, caracter = ')', frecuencia = 3
- ASCII = 43, caracter = '+', frecuencia = 20
- ASCII = 45, caracter = '-', frecuencia = 105
- ASCII = 47, caracter = '/', frecuencia = 1
- ASCII = 60, caracter = '<', frecuencia = 1
- ASCII = 62, caracter = '>', frecuencia = 1
- ASCII = 83, caracter = 'S', frecuencia = 1
- ASCII = 92, caracter = '\', frecuencia = 6
- ASCII = 94, caracter = '^', frecuencia = 2
- ASCII = 95, caracter = '_', frecuencia = 42
- ASCII = 97, caracter = 'a', frecuencia = 3
- ASCII = 99, caracter = 'c', frecuencia = 3
- ASCII = 100, caracter = 'd', frecuencia = 2
- ASCII = 101, caracter = 'e', frecuencia = 3
- ASCII = 103, caracter = 'g', frecuencia = 1
- ASCII = 104, caracter = 'h', frecuencia = 1
- ASCII = 105, caracter = 'i', frecuencia = 1
- ASCII = 108, caracter = 'l', frecuencia = 1
- ASCII = 111, caracter = 'o', frecuencia = 5
- ASCII = 112, caracter = 'p', frecuencia = 1
- ASCII = 114, caracter = 'r', frecuencia = 1
- ASCII = 119, caracter = 'w', frecuencia = 1
- ASCII = 121, caracter = 'y', frecuencia = 1
- ASCII = 122, caracter = 'z', frecuencia = 1
- ASCII = 124, caracter = '|', frecuencia = 7

- El árbol asociado es (las hojas tiene asociado un valor "vXfY", donde "X" es el código del símbolo y "Y" es su frecuencia):



El objetivo del componente 2 es implementar un árbol de Huffman para codificar y decodificar imágenes en escala de grises. En este contexto, el "mensaje" sería la imagen y los "símbolos" del mensaje serían los diferentes píxeles de la imagen, cada uno con un color (intensidad) específico. En el proceso de codificación, se tomará una imagen en formato PPM y se generará un archivo binario con la correspondiente codificación. El archivo binario tiene la siguiente estructura:

`W H M F0 F1 ... F255` *bits...*

Donde:

- "`W`" y "`H`" son números enteros positivos de 2 bytes (`unsigned short`). Representan respectivamente el ancho y el alto de la imagen.
- "`M`" es un número entero positivo de 1 byte (`unsigned char`). Representa el valor de píxel más grande de la imagen, es decir, la intensidad máxima de la imagen.
- Los valores "`Fi`" son las frecuencias de cada dato posible en la imagen sin importar a cuál canal (color básico) pertenecen. Cada valor se representa con un entero positivo de 8 bytes (`unsigned long`).
- La secuencia "*`bits`*" son los bits asociados a la compresión de la imagen: los píxeles están ordenados de la misma forma que en el formato del archivo PPM (fila-columna rojo-verde-azul).

En el proceso de decodificación, se tomará un archivo binario con la imagen codificada (que sigue la estructura presentada anteriormente) y se decodificará, produciendo la correspondiente imagen PPM. De esta forma, el componente completo se implementará en los siguientes comandos:

- **comando:** `codificar_imagen <nombre_imagen.ppm> <nombre_archivo.huffman>`
  **salida en pantalla:**
  (proceso satisfactorio) `La imagen <nombre_imagen.ppm> ha sido codificada exitosamente`
  (mensaje de error) `La imagen <nombre_imagen.ppm> no ha podido ser codificada`
  **descripción:** El comando debe cargar en memoria (en la estructura más adecuada) la imagen identificada con `<nombre_imagen.ppm>` y luego debe generar el archivo de texto con la correspondiente codificación de Huffman, almacenándolo en disco bajo el nombre `<nombre_archivo.huffman>`.
- **comando:** `decodificar_archivo <nombre_archivo.huffman> <nombre_imagen.ppm>`
  **salida en pantalla:**
  (proceso satisfactorio) `El archivo <nombre_archivo.huffman> ha sido decodificado exitosamente`
  (mensaje de error) `El archivo <nombre_archivo.huffman> no ha podido ser decodificado`
  **descripción:** El comando debe cargar en memoria (en la estructura más adecuada) la información de codificación contenida en el archivo `<nombre_archivo.huffman>` y luego debe generar la correspondiente imagen decodificada en formato PPM, almacenándola en disco bajo el nombre `<nombre_imagen.ppm>`.

### 2.2.3. Componente 3: Componentes conectados en imágenes

**Objetivo:** Segmentar una imagen a partir de unas semillas dadas por el usuario.

Un tipo particular de imágenes se conocen como etiquetadas, es decir, imágenes con diferentes regiones de color que representan información de interés, como los departamentos en un mapa geográfico o las regiones funcionales del cerebro.

La segmentación se realiza de la siguiente forma:

1. El usuario provee una imagen y una secuencia de semillas. Cada semilla se compone de tres datos: "`x y l`", donde "`x`" y "`y`" es la coordenada de un píxel en la imagen y "`l`" es un valor de etiqueta que tiene un valor entre 1 y 255.
2. La imagen se considera un grafo, donde cada píxel es un nodo del grafo; y cada nodo está conectado con sus vecinos de "arriba", "abajo", "izquierda" y "derecha".
3. A partir de cada semilla, se lanza una instancia del algoritmo de Dijkstra. El costo de cada arista está dado por la distancia euclidiana entre los colores. Por ejemplo, los píxeles [10,11] y [10,10] son vecinos, entonces el costo de su conexión es:

$$C([10,11],[10,10]) = \sqrt{[rI(10,11) - rI(10,10)]^2 + [gI(10,11) - gI(10,10)]^2 + [bI(10,11) - bI(10,10)]^2}$$

4. La etiqueta final de un píxel es aquella asociada a la instancia del algoritmo de Dijkstra que lo alcance primero.

El componente 3 se encargará entonces de segmentar una imagen PPM a partir de un conjunto de semillas dadas por el usuario, esto se implementará con los siguientes comandos:

- **comando:** `segmentar <entrada_imagen.ppm> <salida_imagen.ppm> sx1 sxy1 sl1 sx2 sxy2 sl2 ...`
  **salida en pantalla:**
  (proceso satisfactorio) La imagen en `<entrada_imagen.ppm>` fue segmentada correctamente y guardada en `<salida_imagen.ppm>`.

(mensaje de error) `La imagen <entrada_imagen.ppm> no pudo ser segmentada.`
**descripción:** El comando debe cargar en memoria (en la estructura más adecuada) la imagen identificada con `<entrada_imagen.ppm>` y el conjunto de semillas correspondiente. La imagen con las etiquetas debe quedar guardada en `<salida_imagen.ppm>`.

### 2.3. Interacción con el sistema

La interfaz del sistema a construir debe ser una consola interactiva donde los comandos correspondientes a los componentes serán tecleados por el usuario, de la misma forma que se trabaja en la terminal o consola del sistema operativo. El indicador de línea de comando debe ser el caracter `$`. Se debe incluir el comando `help` para indicar una lista de los comandos disponibles en el momento. Así mismo, para cada comando se debe incluir una ayuda de uso que indique la forma correcta de hacer el llamado, es decir, el comando `help comando` debe existir.

Cada comando debe presentar en pantalla mensajes de éxito o error que permitan al usuario saber, por un lado, cuando terminó el comando su procesamiento, y por el otro lado, el resultado de ese procesamiento. Los comandos de los diferentes componentes deben ensamblarse en un único sistema (es decir, funcionan todos dentro de un mismo programa, no como programas independientes por componentes).

## 3. Evaluación

Las entregas se harán en la correspondiente actividad de Uvirtual, hasta la media noche del día anterior al indicado para la entrega. Se debe entregar un archivo comprimido (únicos formatos aceptados: .zip, .tar, .tar.gz, .tar.bz2, .tgz) que contenga dentro de un mismo directorio (sin estructura de carpetas interna) los documentos (único formato aceptado: .pdf) y el código fuente (.h, .hxx, .hpp, .c, .cxx, .cpp). Si la entrega contiene archivos en cualquier otro formato, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de 0 (cero) sobre 5 (cinco).

### 3.1. Entrega 1 (jueves 28 de febrero o viernes 1 de marzo de 2019)

Componente 1 completo y funcional. Esta entrega tendrá una sustentación durante la correspondiente sesión de clase, y se compone de:

- (30 %) Documento de diseño. El documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: Diagrama de clases (si ya vio ADOO o POO) o diseño precondiciones, poscondiciones e invariantes (si no ha visto ADOO). Además, se exigirá un(os) esquemático(s) que resuma(n) el(los) problema(s) y su(s) solución(ones) gráficamente.
- (55 %) Código fuente compilable en el compilador gnu-g++ v4.0.0 (mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (15 %) Sustentación con participación de todos los miembros del grupo.

### 3.2. Entrega 2 (jueves 11 o viernes 12 de abril de 2019)

Componentes 1 y 2 completos y funcionales. Esta entrega tendrá una sustentación durante la correspondiente sesión de clase, y se compone de:

- (15 %) Completar la funcionalidad que aún no haya sido desarrollada de la primera entrega. Se debe generar un acta de evaluación de la entrega anterior que detalle los elementos faltantes y cómo se completaron para la segunda entrega.
- (25 %) Documento de diseño. El documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares. Para la descripción de los TADs utilizados, debe seguirse la plantilla definida en clase. Además, se exigirán esquemáticos (diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones principales.
- (45 %) Código fuente compilable en el compilador gnu-g++ v4.0.0 (mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (15 %) Sustentación con participación de todos los miembros del grupo.

### 3.3. Entrega 3 (jueves 30 o viernes 31 de mayo de 2019)

Componentes 1, 2 y 3 completos y funcionales. Esta entrega tendrá una sustentación (del proyecto completo) durante el jueves 30 o el viernes 31 de mayo de 2019 entre las 8am y las 6pm, y se compone de:

- (15 %) Completar la funcionalidad que aún no haya sido desarrollada de la primera y segunda entregas. Se debe generar un acta de evaluación de la entrega anterior que detalle los elementos faltantes y cómo se completaron para la segunda entrega.
- (25 %) Documento de diseño. El documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares. Para la descripción de los TADs utilizados, debe seguirse la plantilla definida en clase. Además, se exigirán esquemáticos (diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones principales.
- (45 %) Código fuente compilable en el compilador gnu-g++ v4.0.0 (mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (15 %) Sustentación con participación de todos los miembros del grupo.