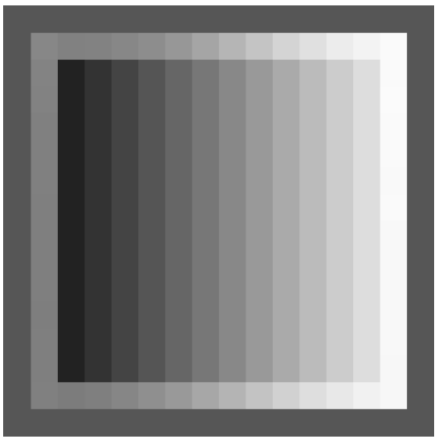
Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Proyecto del curso, 2025-10

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

Figure 1: Representación computacional de imágenes en escala de grises. A la izquierda, una imagen de 16x16 pixeles tal como se visualiza en la pantalla del computador. A la derecha, la información numérica de los pixeles que corresponden a la imagen; se evidencia que los grises oscuros tienen valores más bajos, mientras que los grises claros tienen valores más altos.

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.

1.1 Formato de imágenes

La información de la imagen (intensidades numéricas de los pixeles) se almacena normalmente en un archivo de texto (caracteres en formato ASCII). En este caso, se utilizará siempre el formato PGM (Netpbm grayscale image format)[1], el cual cuenta con la siguiente estructura:

```
P2
# Algún comentario acerca del archivo...
W H
M
```

[1] http://netpbm.sourceforge.net/doc/pgm.html

```
I(1,1) I(2,1) ... I(W,1)
I(1,2) I(2,2) ... I(W,2)
...
I(1,H) I(2,H) ... I(W,H)
```

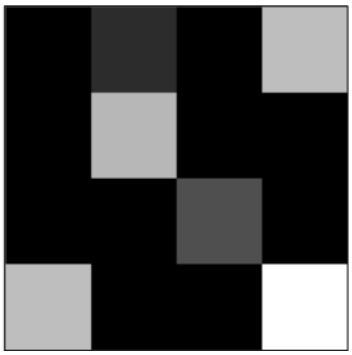
Donde:

- `P2` es un código (conocido también como número mágico) que identifica el formato de la imagen (PGM en ASCII).
- Las líneas que empiezan con el caracter '#' son comentarios, que no tienen influencia en el contenido de la imagen.
- `W` y `H` son números enteros positivos. Representan el ancho y el alto de la imagen en pixeles, respectivamente.
- `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. En general, este valor no debería ser mayor a 255.
- `I(i,j)` es el valor de gris 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.

Nota: dentro del archivo, la información de intensidad de los pixeles no necesariamente está siempre almacenada por líneas; es decir, no necesariamente una línea de texto de intensidades corresponde a una fila de pixeles de la imagen.

A continuación se presenta un ejemplo de archivo en formato PGM, junto con la visualización (magnificada) de la imagen que representa:

```
P2
# imagen ejemplo
4 4
255
0 33 0 170
0 143 0 0
0 0 63 0
170 0 0 255
```



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. El sistema se implementará como una aplicación que recibe comandos textuales, agrupados en componentes con funcionalidades específicas. A continuación se describen los componentes individuales que conforman el proyecto.

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, utilizando estructuras lineales.

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 2.

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

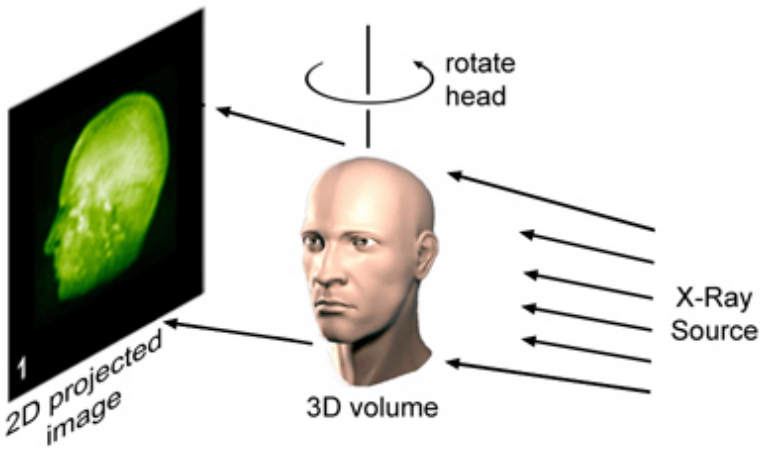


Figure 2: Ilustración del proceso de generación de una proyección planar por medio de rayos X (Tomado de `http://www.sv.vt.edu/classes/ESM4714/methods/FuncExt.html`)

la información contenida en cada una de las filas del volumen en un único píxel. Esto generará una proyección 2D de la imagen volumétrica en la dirección especificada. El componente se implementará con los siguientes comandos:

- **comando:** `cargar_imagen` *nombre_imagen.pgm*
  posibles salidas en pantalla:
  (proceso satisfactorio) `La imagen` *nombre_imagen.pgm* `ha sido cargada.`
  (mensaje de error) `La imagen` *nombre_imagen.pgm* `no ha podido ser cargada.`
  descripción: El comando debe cargar en memoria (en la estructura más adecuada) la imagen identificada con el *nombre_imagen.pgm*. Una vez cargada la información en memoria, el comando debe mostrar el mensaje de carga satisfactoria. Si por alguna razón no es posible cargar la imagen (nombre de archivo erróneo o no existe), el comando debe mostrar el mensaje de error. Solo es posible cargar una única imagen por sesión, de tal forma que si el comando es llamado nuevamente con otro nombre de archivo, la nueva imagen sobreescribe en memoria a la que ya estaba cargada anteriormente.

- **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.pgm`, 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 imágenes no corresponde, error en alguna imagen), el comando debe mostrar el mensaje de error. Solo es posible cargar un único volumen por sesión, de tal forma que si el comando es llamado nuevamente con otro nombre base, el nuevo volumen sobreescribe en memoria al que ya estaba cargado anteriormente.

- **comando:** `info_imagen`
  salida en pantalla:
  (proceso satisfactorio) `Imagen cargada en memoria:` *nombre_imagen.pgm*`, ancho:` *W*`, alto:` *H*.
  (mensaje de error) `No hay una imagen cargada en memoria.`
  descripción: El comando debe mostrar en pantalla la información básica de la imagen actualmente cargada en memoria: nombre de archivo, ancho en pixeles y alto en pixeles. Si no se ha cargado aún una imagen en memoria, el comando debe mostrar el mensaje de error.

- **comando:** `info_volumen`
  salida en pantalla:

(proceso satisfactorio) `Volumen cargado en memoria: nombre_base, tamaño: n_im, ancho: W, alto: H.`
(mensaje de error) `No hay un volumen cargado en memoria.`
descripción: El comando debe mostrar en pantalla la información básica del volumen (serie de imágenes) cargado actualmente en memoria: nombre base, cantidad de imágenes, ancho en pixeles y alto en pixeles. Si no se ha cargado aún un volumen en memoria, el comando debe mostrar el mensaje de error.

- **comando:** `proyeccion2D` *dirección criterio nombre_archivo.pgm*
  salida en pantalla:
  (proceso satisfactorio) `La proyección 2D del volumen en memoria ha sido generada y almacenada en el archivo` *nombre_archivo.pgm*.
  (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 de esas posiciones debe colapsar toda la información existente en la dirección dada utilizando el criterio especificado. Esto genera un único 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), `maximo` (el valor máximo de intensidad), `promedio` (el valor promedio de intensidad) o `mediana` (el valor mediana de intensidad). Una vez generada la proyección, debe guardarse como imagen en formato PGM como *nombre_archivo.pgm*. Es importante anotar que este comando solo puede funcionar sobre volúmenes (series de imágenes). La Figura 3 ilustra el proceso de proyección en cada una de las posibles direcciones.

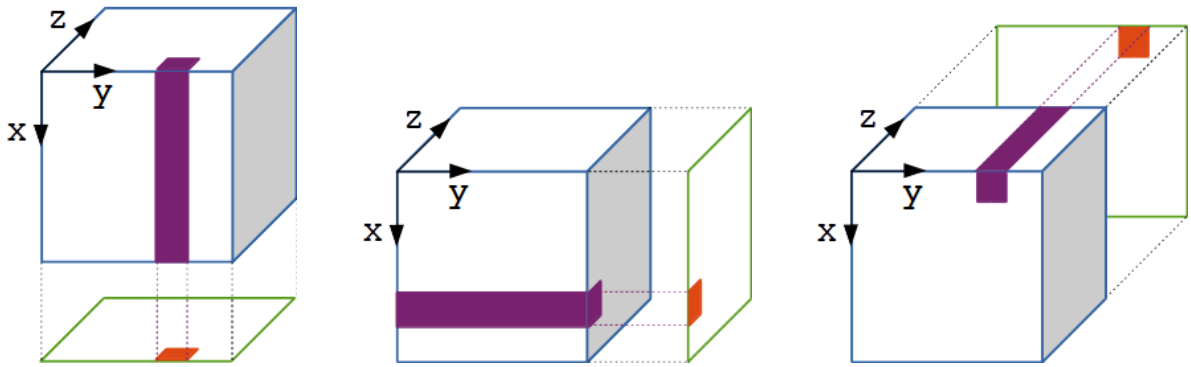


Figure 3: Ilustración del proceso de proyección 2D. La información del vector (en morado) se colapsa en un único pixel (en naranja). Izquierda: proyección en `x`. Centro: proyección en `y`. Derecha: proyección en `z`.

2.2 Componente 2: Codificación de imágenes

Objetivo: Utilizar el algoritmo de codificación de Huffman, y su implementación basada en árboles, 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[2] 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

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

representa una codificación de un conjunto de símbolos óptima: el símbolo que tenga una frecuencia más alta (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 4.

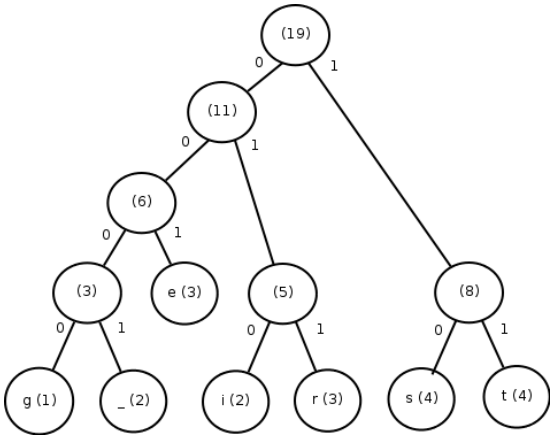


Figure 4: Árbol de Huffman.

En este árbol, cada nodo **hoja** almacena un símbolo del lenguaje utilizado en un mensaje, junto con 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 solo 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 conexión 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 conexiones 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 conexiones 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, asumiendo la siguiente matriz de datos como una imagen, donde cada fila empieza y termina con el símbolo '+':

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

Se pueden identificar las siguientes características:

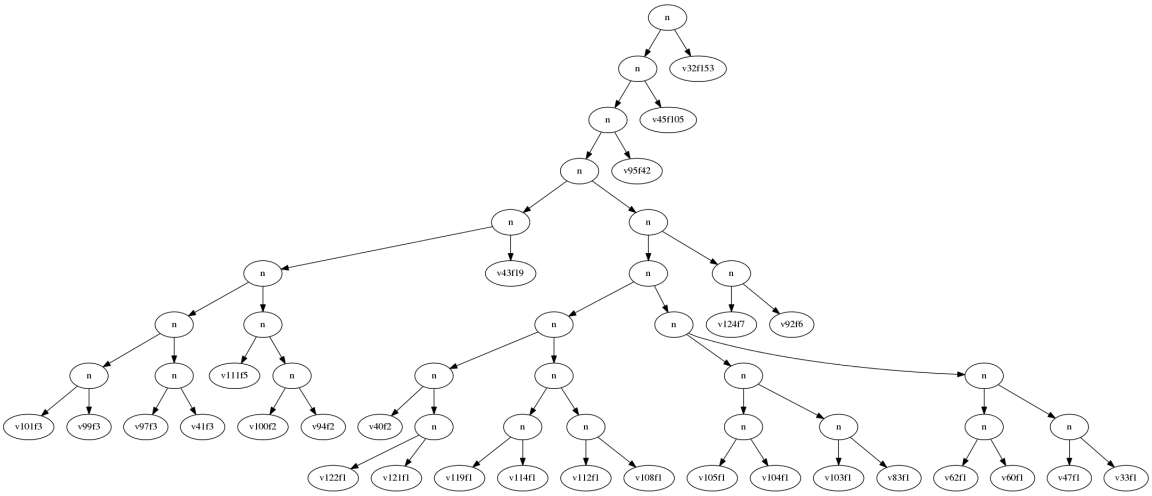
- Ancho de la imagen (`W`) = 36.
- Alto de la imagen (`H`) = 10.
- Valor (intensidad) máximo posible de la imagen (`M`) = 255.

- Los caracteres (que hacen las veces de intensidades) y sus frecuencias son:

ASCII	carácter	frecuencia
32		153
33	!	1
40	(	22
41	)	3
43	+	20
45	-	105
47	/	1
60	<	1
62	>	1
83	S	1
92	\	6
94	^	2
95	_	42
97	a	3

ASCII	carácter	frecuencia
99	c	3
100	d	2
101	e	3
103	g	1
104	h	1
105	i	1
108	l	1
111	o	5
112	p	1
114	r	1
119	w	1
121	y	1
122	z	1
124	|	7

- El árbol de codificación 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 PGM y se generará un archivo binario con la correspondiente codificación. El archivo binario tiene la siguiente estructura:

```
W H M F0 F1 ... FM 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 (intensidad) posible en la imagen, tomadas en orden ascendente (desde la intensidad `0` hasta la `M`). 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, es decir, la secuencia de códigos que corresponden a las intensidades de pixeles presentes en la imagen, leída y concatenada por filas.

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 PGM. Es importante resaltar que se trabaja con una única imagen en el proceso de codificación y decodificación. De esta forma, el componente completo se implementará con los siguientes comandos:

- **comando:** `codificar_imagen` *`nombre_archivo.huf`*
  salida en pantalla:
  (proceso satisfactorio) `La imagen en memoria ha sido codificada exitosamente y almacenada en el archivo` *`nombre_archivo.huf`* `.`
  (mensaje de error) `No hay una imagen cargada en memoria.`
  descripción: El comando debe generar el archivo de texto con la correspondiente codificación de Huffman para la imagen que se encuentre actualmente cargada en memoria, almacenándolo en disco bajo el nombre *`nombre_archivo.huf`*. Si no se ha cargado aún una imagen en memoria, el comando debe mostrar el mensaje de error.

- **comando:** `decodificar_archivo` *`nombre_archivo.huf nombre_imagen.pgm`*
  salida en pantalla:
  (proceso satisfactorio) `El archivo` *`nombre_archivo.huf`* `ha sido decodificado exitosamente, y la imagen correspondiente se ha almacenado en el archivo` *`nombre_imagen.pgm`* `.`
  (mensaje de error) `El archivo` *`nombre_archivo.huf`* `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.huf`* y luego debe generar la correspondiente imagen decodificada en formato PGM, almacenándola en disco bajo el nombre *`nombre_imagen.pgm`*. Si por alguna razón no es posible cargar la información de codificación (nombre de archivo erróneo o no existe), o no es posible realizar el proceso de decodificación (mal formato del archivo), el comando debe mostrar el mensaje de error.

2.3 Componente 3: Componentes conectados en imágenes

Objetivo: Segmentar una imagen a partir de unas semillas dadas por el usuario, utilizando representaciones basadas en grafos.

Un tipo particular de imágenes se conocen como etiquetadas o segmentadas, 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 de una imagen puede realizarse 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. Cada etiqueta representa el color de una región diferente que se espera encontrar en la imagen.

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". Nodos en los bordes de la imagen estarán conectados solo a los vecinos que estén disponibles en cualquiera de esas 4 direcciones.

3. Dentro del grafo, el costo de cada arista está dado por la diferencia (o distancia) entre los colores. Por ejemplo, los píxeles [10,11] y [10,10] son vecinos, entonces el costo de su conexión es:

$$C([10,11],[10,10]) = I(10,11) - I(10,10)$$

4. A partir de cada semilla dada por el usuario, se lanza una instancia del algoritmo de Dijkstra. La etiqueta final de cada píxel en la imagen resultado es aquella asociada a la instancia del algoritmo de Dijkstra que lo alcance primero.

El componente 3 se encargará entonces de segmentar una única imagen PGM a partir de un conjunto de semillas dadas por el usuario. Es importante resaltar que se trabaja con una única imagen en el proceso de segmentación. De esta forma, el componente se implementará con el siguiente comando:

- **comando:** `segmentar` *`salida_imagen.pgm sx1 sy1 sl1 sx2 sy2 sl2 ...`*
  salida en pantalla:
  (proceso satisfactorio) `La imagen en memoria fue segmentada correctamente y almacenada en el archivo` *`salida_imagen.pgm`*.

(mensaje de error)
`No hay una imagen cargada en memoria.`
`La imagen en memoria no pudo ser segmentada.`
descripción: El comando debe cargar la información del conjunto de semillas correspondiente a la imagen cargada en memoria, para luego proceder a su segmentación de acuerdo al algoritmo presentado anteriormente. El usuario puede ingresar un máximo de 5 semillas en el comando. La imagen con las etiquetas debe quedar guardada en *salida_imagen.pgm*. Si no se ha cargado aún una imagen en memoria, o por alguna razón no es posible realizar el proceso de segmentación (semillas mal ubicadas, problemas en la construcción del grafo), el comando debe mostrar el mensaje de error.

2.4 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 ayuda 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 `ayuda` *comando* debe existir.

Cada comando debe presentar en pantalla los mensajes de resultado (éxito o error) especificados antes, además de otros mensajes necesarios 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 a través de la correspondiente asignación de BrightSpace, hasta la media noche del día anterior al indicado para la sustentación de la entrega. Se debe entregar un archivo comprimido (único formato aceptado: .zip) que contenga dentro de un mismo directorio (**sin estructura de carpetas interna**) los documentos (único formato aceptado: .pdf) y el código fuente (.h, .hxx, .cxx, .cpp). Si la entrega contiene archivos en cualquier otro formato, o no cumple con las indicaciones anteriores, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de **0 (cero) sobre 5 (cinco)**.

3.1 (10%) Entrega 0: semana 3

La entrega inicial corresponderá únicamente a la interfaz de usuario necesaria para interactuar con el sistema. De esta forma, se verificará el indicador de línea del comando, y que el sistema realice la validación de los comandos permitidos y sus parámetros (en cantidad, y formato de ser necesario). (Revisar en particular el numeral 2.4).

3.2 (30%) Entrega 1: semana 6

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

- (40%) 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 (comandos). 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 (comandos) principales.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `proyeccion2D`.
- (30%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0, como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (10%) Sustentación (individual) con participación de todos los miembros del grupo.

3.3 (30%) Entrega 2: semana 12

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

- (10%) Completar la funcionalidad que aún no haya sido desarrollada de la primera entrega. Se debe generar un acta de evaluación de la entrega anterior (incluirla al principio del documento de diseño) que detalle los comentarios textuales (literales) hechos a la entrega y la forma en la que se corrigieron, arreglaron o completaron para la segunda entrega.
- (30%) 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 (comandos). 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 (comandos) principales.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `decodificar`.
- (30%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0, como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (10%) Sustentación (individual) con participación de todos los miembros del grupo.

3.4 (30%) Entrega 3: semana 18

Componentes 1, 2 y 3 completos y funcionales. Esta entrega tendrá una sustentación (del proyecto completo) entre las 8am y las 12m del último día de clase de la semana 18, y se compone de:

- (10%) Completar la funcionalidad que aún no haya sido desarrollada de la primera y segunda entregas. Se debe generar un acta de evaluación de las entregas anteriores (incluirla al principio del documento de diseño) que detalle los comentarios textuales (literales) hechos a las entregas y la forma en la que se corrigieron, arreglaron o completaron para la tercera entrega.
- (30%) 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 (comandos). 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 (comandos) principales.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `segmentar`.
- (30%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0, como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (10%) Sustentación (individual) con participación de todos los miembros del grupo.