UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2014

10 de octubre de 2014

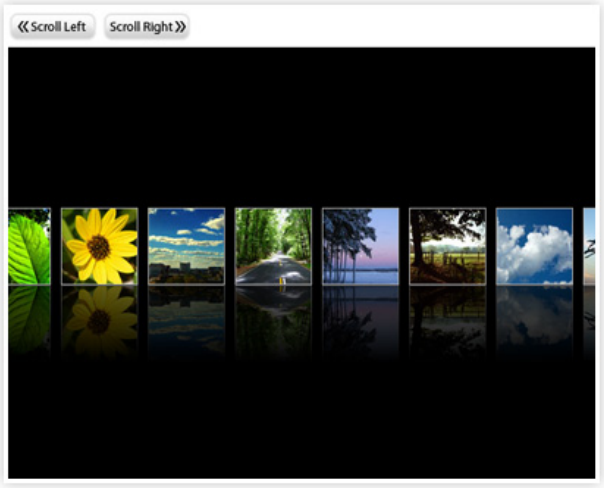
Galerazo

Aclaraciones

- Para aprobar la totalidad del TP es necesario tener aprobado cada uno de sus módulos.
- **No** está permitido el uso de **acum** para la resolución.
- **Fecha de entrega:** 31 de Octubre de 2014

El problema **Galerazo** es mundialmente conocido también como **Photo Gallery** o **Image Gallery**.

Básicamente, nos permite tener una lista de imágenes o **Galería** y sobre ellas obtener valoraciones o **votos**, obtener algunas imágenes específicas que cumplan cierta propiedad, etc.



Se pide realizar la especificación de los problemas mencionados a continuación, así como también implementarlos en `C++`.

Las clases implementadas deberán utilizar los archivos `.h` que se entregan como anexo al presente documento. En dichos archivos se pueden agregar los métodos auxiliares que sean necesarios. Dichos métodos deben estar *en todos los casos* en la parte **privada** de la clase. No se permite modificar la parte **pública** de las clases ni agregar atributos adicionales, ya sean públicos o privados.

1. Tipos

```
tipo Pixel {
  observador red (p: Pixel) : ℤ;
  observador green (p: Pixel) : ℤ;
  observador blue (p: Pixel) : ℤ;
  invariante enRango : 0 ≤ red(p) ≤ 255 ∧
     0 ≤ green(p) ≤ 255 ∧ 0 ≤ blue(p) ≤ 255;
}
```

```
tipo Imagen {
  observador ancho (i: Imagen) : ℤ;
  observador alto (i: Imagen) : ℤ;
  observador color (i: Imagen, x, y: ℤ) : Pixel;
       requiere 0 ≤ x < ancho(i) ∧ 0 ≤ y < alto(i);
  invariante tieneTamano : ancho(i) > 0 ∧ alto(i) > 0;
}
```

```
tipo Galeria {
  observador imagenes (g: Galeria) : [Imagen];
  observador votos (g: Galeria, i: Imagen) : ℤ;
       requiere i ∈ imagenes(g);
  invariante sinRepetidos : (∀i, j ← [0..|imagenes(g)|), i ≠ j)imagenes(g)_i ≠ imagenes(g)_j;
  invariante votosNoNegativos : (∀i ← imagenes(g))votos(g, i) ≥ 0;
}
```

2. Especificación

Especificar los siguientes problemas:

1. `problema posicionesMasOscuras` (i: Imagen) = `result` : [($\mathbb{Z}$, $\mathbb{Z}$)]
   Devuelve una lista con todas las posiciones más oscuras de la imagen. Las posiciones más oscuras son aquellas con menor color, es decir aquellas que tienen red+green+blue más cercano a 0.
2. `problema top10` (g: Galeria) = `result` : [Imagen]
   Devuelve las 10 imágenes con más votos (en orden decreciente de votos).
3. `problema laMasChiquitaConPuntoBlanco` (g: Galeria) = `result` : Imagen
   Devuelve alguna de las imágenes de la galería que tenga un pixel blanco y que tenga la menor cantidad de píxeles en total (entre todas las que tienen al menos un pixel blanco).
4. `problema agregarImagen` (g:Galeria, i: Imagen)
   Modifica la galería agregando la imagen i.
5. `problema votar` (g:Galeria, i: Imagen)
   Modifica la galería sumándole un voto a la imagen i.
6. `problema eliminarMasVotada` (g:Galeria)
   Modifica la galería eliminando alguna de las imagenes que cuentan con mayor cantidad de votos.

3. Implementación

Utilizando los archivos `.h` provistos por la cátedra, implementar los problemas especificados a continuación. Implementar además los problemas `posicionesMasOscuras`, `top10`, `laMasChiquitaConPuntoBlanco` `agregarImagen`, `votar` y `eliminarMasVotada`, respetando las especificaciones dadas previamente.

3.1. Pixel

`problema Pixel` $(r, g, b : \mathbb{Z})$ = `result : Pixel` {
  `requiere` $0 \leq r \leq 255 \land 0 \leq g \leq 255 \land 0 \leq b \leq 255$;
  `asegura` $red(result) == r \land$
    $green(result) == g \land blue(result) == b$;
}

`problema cambiarPixel` $(this : \texttt{Pixel}, r, g, b : \mathbb{Z})$ {
  `requiere` $0 \leq r \leq 255 \land 0 \leq g \leq 255 \land 0 \leq b \leq 255$;
  `modifica` $this$;
  `asegura` $red(this) == r \land$
    $green(this) == g \land blue(this) == b$;
}

`problema red` $(this : \texttt{Pixel})$ = `result` : $\mathbb{Z}$ {
  `asegura` $result == red(this)$;
}

`problema green` $(this : \texttt{Pixel})$ = `result` : $\mathbb{Z}$ {
  `asegura` $result == green(this)$;
}

`problema blue` $(this : \texttt{Pixel})$ = `result` : $\mathbb{Z}$ {
  `asegura` $result == blue(this)$;
}

3.2. Imagen

`problema Imagen` $(alto, ancho : \mathbb{Z})$ = `result : Imagen` {
  `requiere` $alto > 0 \land alto > 0$;
  `asegura dimensionesOK` : $alto(result) == alto \land ancho(result) == ancho$;
  `asegura todaNegra` : $(\forall i \leftarrow [0..ancho), j \leftarrow [0..alto))\ esNegro(color(result, i, j))$;
}

`problema obtenerPixel` $(this : \texttt{Imagen}, f, c : \mathbb{Z})$ = `result : Pixel` {
  `requiere` $0 \leq f < ancho(this) \land 0 \leq c < alto(this)$;
  `asegura` $result == color(this, f, c)$;
}

`problema modificarPixel` $(this : \texttt{Imagen}, f, c : \mathbb{Z}, p : \texttt{Pixel})$ {
  `requiere` $0 \leq f < ancho(this) \land 0 \leq c < alto(this)$;
  `modifica` $this$;
  `asegura dimensionesOK` : $alto(this) == alto(pre(this)) \land ancho(this) == ancho(pre(this))$;
  `asegura demasPosicionesIguales` :
    $(\forall x \leftarrow [0..ancho(this)), y \leftarrow [0..alto(this)), x \neq f \lor y \neq c) color(this, x, y) == color(pre(this), x, y)$;
  `asegura cambiaPosicion` : $color(this, f, c) == p$;
}

```
problema ancho (this : Imagen) = result : ℤ {
  asegura result == ancho(this);
}

problema alto (this : Imagen) = result : ℤ {
  asegura result == alto(this);
}

problema blur (this : Imagen, k : ℤ) {
  requiere k > 0;
  modifica this;
  asegura mismasDimensiones : alto(this) == alto(pre(this)) ∧ ancho(this) == ancho(pre(this));
  asegura blurKVecinosIncompletos : (∀x ← [0..ancho(this)), y ← [0..alto(this)))
      ¬KVecinosCompletos(this, x, y, k) ⇒ esNegro(color(this, x, y));
  asegura blurKVecinosCompletos : (∀x ← [0..ancho(this)), y ← [0..alto(this)))
      KVecinosCompletos(this, x, y, k) ⇒ esPromedio(color(this, x, y), Kvecinos(pre(this), x, y, k));
  aux esPromedio (p: Pixel, kv: [Pixel]) : Bool = red(p) == promedio([red(x)|x ← kv]) ∧
      green(p) == promedio([green(x)|x ← kv]) ∧ blue(p) == promedio([blue(x)|x ← kv]);
}

problema acuarela (this : Imagen, k : ℤ) {
  requiere k > 0;
  modifica this;
  asegura mismasDimensiones : alto(this) == alto(pre(this)) ∧ ancho(this) == ancho(pre(this));
  asegura acuarelaKVecinosIncompletos : (∀x ← [0..ancho(this)), y ← [0..alto(this)))
      ¬KVecinosCompletos(this, x, y, k) ⇒ esNegro(color(this, x, y));
  asegura acuarelaKVecinosCompletos : (∀x ← [0..ancho(this)), y ← [0..alto(this)))
      KVecinosCompletos(this, x, y, k) ⇒ esMediana(color(this, x, y), Kvecinos(pre(this), x, y, k));
  aux esMediana (p: Pixel, kv: [Pixel]) : Bool = red(p) == mediana([red(x)|x ← kv]) ∧
      green(p) == mediana([green(x)|x ← kv]) ∧ blue(p) == mediana([blue(x)|x ← kv]);
}
```

3.3. Galería

Para la implementación de esta clase, asumir que el entero almacenado en la *i-ésima* posición del vector `votos` corresponde a la cantidad de votos de la imagen almacenada en la *i-ésima* posición del vector `imagenes`. Tanto las imágenes como los votos correspondientes a cada imagen debe almacenarse de forma tal que el vector `votos` se encuentre siempre ordenado de manera creciente.

Dar el predicado de abstracción y el invariante de representación de la clase GaleriaImagenes.

```
problema dividirYAgregar (this : Galeria, i : Imagen, n, m : ℤ) {
  requiere i ∈ imagenes(this);
  requiere n > 0 ∧ m > 0;
  requiere divideBien : ancho(i) mod n == 0 ∧ alto(i) mod m == 0;
  requiere sinPartesIguales : (∀x ← [0..n), y ← [0..m)) esDistintoDeTodos(i, x, y, n, m);

  modifica this;

  asegura |imagenes(this)| == |imagenes(pre(this))| + n * m;
  asegura lasAnteriores SiguenEstando :
      (∀x ← imagenes(pre(this))) x ∈ imagenes(this) ∧ votos(this, x) == votos(pre(this), x);
  asegura lasDivisionesEstan :
      (∀x ← [0..n), y ← [0..m)) estaEsaParte(i, x, y, ancho(i)/n, alto(i)/m, this);

  aux esDistintoDeTodos (imagen : Imagen, x, y, n, m : ℤ) : Bool =
      (∀i ← [0..n), j ← [0..m), i ≠ x ∨ j ≠ y)
      alMenosUnPixelDistinto(imagen, x, y, i, j, ancho(imagen)/n, alto(imagen)/m);
  aux alMenosUnPixelDistinto (imagen : Imagen, x, y, i, j, ancho, alto : ℤ) : Bool = (∃a ← [0..ancho), b ← [0..alto))
      color(imagen, (x × ancho) + a, (y × alto) + b) ≠ color(imagen((i × ancho) + a, (j × alto) + b));
  aux estaEsaParte (i : Imagen, x, y, ancho, alto : ℤ, g : Galeria) : Bool = (∃ imagen ← imagenes(g))
      ancho(imagen) == ancho ∧ alto(imagen) == alto ∧ mismosPixeles(imagen, i, x, y, ancho, alto) ∧ votos(g, i) == 0;
  aux mismosPixeles (im, imCompleta : Imagen, x, y, ancho, alto : ℤ) : Bool = (∀i ← [0..ancho), j ← [0..alto))
      color(im, i, j) == color(imCompleta, (x × ancho) + i, (y × alto) + j);
```

}

Auxiliares

`aux esNegro` $(p : \texttt{Pixel}) : \texttt{Bool} = red(p) == 0 \land green(p) == 0 \land blue(p) == 0$;
`aux KVecinos` $(im : \texttt{Imagen}, x, y, k : \mathbb{Z}) : [\texttt{Pixel}] = [color(im, i, j) | i \leftarrow [0..ancho(im)), j \leftarrow [0..alto(im)), sonKVecinos(k, x, y, i, j)]$
`aux KVecinosCompletos` $(im : \texttt{Imagen}, x, y, k : \mathbb{Z}) : \texttt{Bool} = |KVecinos(im, x, y)| == (2k-1)^2 - 1$;
`aux promedio` $(xs : [\mathbb{Z}]) : \mathbb{Z} = \sum xs / |xs|$;
`aux mediana` $(xs : [\mathbb{Z}]) : \mathbb{Z} = ordenar(xs)_{|xs| \ div \ 2}$;
`aux ordenar` $(xs : [\mathbb{Z}]) : [\mathbb{Z}] = [\ x \mid i \leftarrow [0..|xs|),\ x \leftarrow xs,\ cuentaMenores(xs, x) == i]$;
`aux cuentaMenores` $(xs : [\mathbb{Z}], x : \mathbb{Z}) : \mathbb{Z} = |[1 \mid y \leftarrow xs,\ y < x]|$;
`aux sonKVecinos` $(k, xa, ya, xb, yb : \mathbb{Z}) : \texttt{Bool} = |xa - xb| < k \land |ya - yb| < k$;

4. Entrada - Salida

El sistema se conectará con el exterior mediante un ciclo en el `main` que se ejecuta hasta que se presione alguna tecla distinguida como ser "x".

Deberá mostrar por pantalla un menú con las opciones posibles. Luego de seleccionar la opción deseada, tomará los parámetros por teclado.

Cuando la salida sea de tipo `Imagen` o `Galeria` (ver Sección 5.1), deberán salir a un archivo, cuyo nombre se deberá pedir por pantalla.

Las mismas serán (aunque pueden renombrarlas a gusto):

- **blur**. Toma un entero K y dos nombres de archivos. Uno será el archivo de entrada y el otro el de salida.
- **acuarela**. Toma un entero K y dos nombres de archivos. Uno será el archivo de entrada y el otro el de salida.
- **Cargar galeria**. Tomará como parámetro el nombre del archivo desde donde levantar la galería.[5.1.3]. No hará nada, sólo las cargará en memoria para trabajar luego.
- **dividirYAgregar**. Tomará primero el archivo con la galería, luego el archivo con la imagen y luego el m y el n (éstos directo por teclado). Generará una nueva imagen en la
- **posicionesMasOscuras**. Tomará el archivo con la imagen y devuelve una lista de tuplas de posiciones (Esta lista por pantalla). A la galería se le agregan las nuevas imágenes.
- **Top10**. Utilizando la galería ya cargada, devolverá el resultado en un archivo[5.1.2] con la lista de las imágenes.
- **laMasChiquitaConPuntoBlanco**. Utilizando la galería ya cargada, devolverá el resultado en un archivo de imagen.
- **agregarImagen**. Tomará un arhivo de imagen y agregará dicha imagen a la galería.
- **votar**. Tomará un archivo de imagen y le agregará un voto a dicha imagen.
- **eliminarMasVotada**. No toma ningún parámetro.
- **Guardar galería**. Guarda la galería con los cambios hechos.

El sistema luego esperará una orden numérica de acuerdo al problema a especificar

5. Anexo Formatos

Definiremos los formatos de los distintos componentes (su entrada y salida) como un `string`. Cada uno de la siguiente manera:

- **Pixel:** `(Z;Z;Z)`
  Ej: `(123;1;23)`
  Es decir que sólo es una tripla de enteros.

5.1. Formato de los archivos

5.1.1. Formato de archivo de imagen

El archivo con imagen, deberá contener `ALTO ANCHO (Z;Z;Z),(Z;Z;Z),(Z;Z;Z),(Z;Z;Z)`

Ejemplo, (imagen de 2×2):

```
2 2 [(1;2;3),(123;121;312),(2;1;231),(167;161;173)]
```

Es decir que primero se dice la cantidad de elementos de alto, luego la cantidad de elementos de ancho y finalmente la lista de esos elementos.

5.1.2. Formato de archivo de lista de imágenes

Este archivo contendrá simplemente una lista de imágenes.

Ejemplo, (2 imágenes, una de 2×2 y una de 2×1):

```
[2 2 [(1;2;3),(123;121;312),(2;1;231),(167;161;173)],2 1 [(1;2;3),(123;121;312)]]
```

5.1.3. Formato de archivo de galería

Para el archivo de tipo galería se tomará una lista de imágenes. Para esto se deberá tomar del archivo una imagen por línea.

Ejemplo, (galería de 2 imágenes):

```
2 2 [(1;2;3),(123;121;312),(2;1;231),(167;161;173)]
2 1 [(1;2;3),(123;121;312)]
```

6. Demostraciones

Se pide demostrar la correctitud de la funcón `eliminarMasVotada` respecto de la especificación dada. Se pide también demostrar que la implementación de dicha función preserva el invariante de representación propuesto. La especificaciones de los métodos de la clase `vector` necesarias para las demostraciones se encuentran en la siguiente sección.

7. Vector

A continuación se presenta la especificación de algunos de los métodos de la clase `vector`.

```
problema CVector (n : ℤ, e : T) = result : [T] {
  requiere n > 0;
  asegura |result| == n;
  asegura (∀x ← result)x == e;
}

problema size (a : [T]) = result : ℤ {
  asegura result == |a|;
}

problema at (a : [T], i : ℤ) = result : T {
  requiere 0 ≤ i < |a|;
  asegura result == a_i;
}
```

```
problema push_back (a : [T], e : T) {
  modifica a;
  asegura |a| == |pre(a)| + 1;
  asegura (∀i ← [0..|pre(a)|))a_i == pre(a)_i;
  asegura a_{|a|-1} == e;
}

problema pop_back (a : [T]) {
  requiere |a| > 0;
  modifica a;
  asegura |a| == |pre(a)| - 1;
  asegura (∀i ← [0..|pre(a)| - 1))a_i == pre(a)_i;
}
```