Universidad de San Carlos de Guatemala

Facultad de Ingeniería Escuela de Ciencias y Sistemas

Estructura de datos P Vacaciones de junio 2019

Catedrático: Ing. Luis Espino

Auxiliar: Javier Navarro

Drawing canvas

Manual Técnico

Nombre: César Alejandro Chinchilla González

Carné: 201612132

INTRODUCCIÓN

En el siguiente trabajo trata el contenido para una guía y muestra de la implementación que se lleva a cabo para el desarrollo de una aplicación en el lenguaje c++ que está basado en la generación de imágenes a través de la simulación de pixeles.

Este proyecto se requiere la utilización de estructuras de datos dinámicas avanzadas como los árboles binarios de búsqueda, avl y matrices dispersas. También el uso de listas dobles y simples para el manejo óptimo de la aplicación.

Objetivos

Objetivo General:

Realizar una aplicación de escritorio basado en la generación de imágenes con simulación de pixeles amigable y simple, implementado estructuras como árboles, listas enlazadas y matrices dispersas para el manejo de la información en memoria.

Objetivos Específicos:

1. Implementar una matriz dispersa para guardar la imagen de pixeles a nivel lógico y de aplicación.
2. Desarrollar listas enlazadas y arboles binarios de búsqueda y equilibrados para el manejo de capas, imágenes y usuarios de la aplicación.
3. Programar la carga y lectura de archivos para obtener la información que se almacenará en el sistema.

Descripción

El sistema consiste en un generador de imágenes por capas, la aplicación contara con un conjunto de capas cargadas previamente y almacenadas en memoria para ser utilizadas, estas capas contendrán imágenes hechas con pixeles. El sistema es capaz de generar imagen seleccionando las capas deseadas.

Métodos Principales

Metodos para la gestión de Menus:

```
void menuPrincipal(){
    int opcionmenu=0;
    bool cargahecha = false;
    string linea;

    while(opcionmenu!=3){
        cout << "_________________Drawing Canvas_________________" << endl;
        cout << "Seleccione una opcion:" << endl;
        cout << "1. Carga masiva de archivos" << endl;
        cout << "2. Funcionalidades" << endl;
        cout << "3. Salir" << endl;
        getline(cin,linea);
        if(esNumero(linea)){
            opcionmenu=atoi(linea.c_str());
            if(opcionmenu==1 && cargahecha==false){
                lecturaCapas();
                lecturaImagenes();
                lecturaUsuarios();
                cargahecha = true;
                cout << "Cargas masivas realizadas exitosamente." << endl;
                funcionalidades();
            } else if(opcionmenu==1 && cargahecha==true){
                 cout << "Las cargas masivas ya fueron realizadas con exito." << endl;
            } else if(opcionmenu==2 && cargahecha==false){
                 cout << "Primero realice la carga masiva de archivos por favor." << endl;
            } else if(opcionmenu==2 && cargahecha==true){
```

```
            funcionalidades();
        } else if(opcionmenu==3){

        } else{
            cout << "Ingrese un numero de las opciones por favor." << endl;
        }

    } else {
        cout << "Entrada escrita invalida, intentelo nuevamente." << endl;
    }
  }

}

void funcionalidades(){
  int opcionfuncion = 0;
  string linea;
  while(opcionfuncion!=5){
    cout << "*Funcionalidades" << endl;
    cout << "¿Que desea hacer?" << endl;
    cout << "1. Generar imagen" << endl;
    cout << "2. Gestionar ABC" << endl;
    cout << "3. Ver estado de la memoria" << endl;
    cout << "4. Ver otros reportes" << endl;
    cout << "5. Salir de funcionalidades" << endl;
    getline(cin,linea);

    if(esNumero(linea)){
      opcionfuncion = atoi(linea.c_str());
      if(opcionfuncion==1){
        generacionImagenes();
```

```
        } else if(opcionfuncion==2){
            gestionarABC();
        } else if(opcionfuncion==3){
            estadoMemoria();
        } else if(opcionfuncion==4){
            reportes();
        } else if(opcionfuncion==5){
            cout << "Saliendo de funcionalidades ... " << endl;
        } else{
            cout<< "Ingrese un numero de las opciones por favor." << endl;
        }
    } else{
        cout << "Entrada escrita invalida, intentelo nuevamente." << endl;
    }
  }


}

void generacionImagenes(){
  string linea;
  int opcionimg = 0;
  while(opcionimg!=5){
    cout << "+Generacion de imagenes" << endl;
    cout << "Seleccione una opcion:" << endl;
    cout << "1. Por recorrido limitado" << endl;
    cout << "2. Por lista de imagenes" << endl;
    cout << "3. Por capa" << endl;
    cout << "4. Por usuario" << endl;
    cout << "5. Salir de generacion de imagenes" << endl;
    getline(cin,linea);
```

```
        if(esNumero(linea)){
            opcionimg = atoi(linea.c_str());
            if(opcionimg==1){
                porRecorridoLimitado();
            } else if(opcionimg==2){
                porListaImagenes();
            }  else if(opcionimg==3){
                porCapa();
            }  else if(opcionimg==4){
                porUsuario();
            }  else if(opcionimg==5){
                cout << "Saliendo de generacion de imagenes ..." << endl;
            } else {
                cout << "Ingrese un numero de las opciones por favor." << endl;
            }
        } else{
            cout << "Entrada escrita invalida, intentelo nuevamente." << endl;
        }
    }

}
```

- ## Métodos para Recorrido Limitado:

```
void porRecorridoLimitado(){
    string linea;
    int numcapas = 0;
    int tiporecorrido = 0;
    int idimagen = 0;

    NodoImagen* imagenactual;
```

```
    bool idcorrecto = false;
    bool esnumcapa = false;

    while(idcorrecto!=true){
        cout << "Ingrese el identificador de la imagen a graficar: " << endl;
        getline(cin,linea);
        if(esNumero(linea)){
            idimagen=atoi(linea.c_str());
            imagenactual = imagenes->buscarImagen(idimagen);
                if(imagenactual==NULL){
                    cout << "No existe esa imagen en el sistema, intentelo nuevamente." <<endl;
                } else{
                    idcorrecto=true;
                    cout << "Imagen encontrada en el sistema." << endl;
                }
        } else{
            cout << "Entrada invalida para el identificador de la imagen, intentelo nuevamente con un
numero por favor." << endl;
        }
    }

    while(esnumcapa!=true){
        cout << "Ingrese el numero de capas a utilizar: " << endl;
        getline(cin,linea);
        if(esNumero(linea)){
            numcapas = atoi(linea.c_str());
            esnumcapa = true;
        } else{
            cout << "Entrada invalida para el numero de capas, intentelo nuevamente." << endl;
        }
    }
```

```
do{
   cout << "Seleccione el tipo de recorrido:" << endl;
   cout << "1.Preorden" << endl;
   cout << "2.Inorden" << endl;
   cout << "3.Postorden" << endl;
   getline(cin,linea);
   if(esNumero(linea)){
      tiporecorrido = atoi(linea.c_str());
      if(tiporecorrido == 1 || tiporecorrido==2 || tiporecorrido==3){

      } else{
         cout << "Ingrese un numero de las opciones por favor." << endl;
      }
   } else{
      cout << "Entrada invalida, intentelo nuevamente." << endl;
   }
} while(tiporecorrido != 1 && tiporecorrido!=2 && tiporecorrido!=3);


if(imagenactual->listacapas->primero!=NULL && numcapas!=0){

   if(tiporecorrido==1){
      recorridoPreordenCapas(arbolcapas->Oraiz(),imagenactual,&numcapas);
      contador = 0;
      auxmatriz->generarGrafica();
      auxmatriz->generarLienzo();
      auxmatriz = new MatrizCapa();
   } else if(tiporecorrido==2){
      recorridoInordenCapas(arbolcapas->Oraiz(),imagenactual,&numcapas);
      contador = 0;
```

```
            auxmatriz->generarGrafica();
            auxmatriz->generarLienzo();
            auxmatriz = new MatrizCapa();
        } else if(tiporecorrido==3){
            recorridoPostordenCapas(arbolcapas->Oraiz(),imagenactual,&numcapas);
            contador = 0;
            auxmatriz->generarGrafica();
            auxmatriz->generarLienzo();
            auxmatriz = new MatrizCapa();
        }
    } else {
        auxmatriz->agregarPixel(0,0,"#000000");
        auxmatriz->generarGrafica();
        auxmatriz->generarLienzo();
        auxmatriz = new MatrizCapa();
    }

}

void recorridoPreordenCapas(NodoCapa* r,NodoImagen* imagen,int* numcapas){
    if (r != NULL && contador!=*numcapas){
        Capa* actual = imagen->listacapas->primero;
        while(actual!=NULL){
            if(r->capa==atoi(actual->idcapa.c_str())){
                NodoLateral* lateral = r->matrizcapa->laterales->primero;
                while(lateral!=NULL){
                    NodoDispersa* nodocolor = lateral->fila->primero;
                    while(nodocolor!=NULL){
                        auxmatriz->agregarPixel(nodocolor->posx,nodocolor->posy,nodocolor->color);
                        nodocolor = nodocolor->derecha;
                    }
```

```
            lateral = lateral->siguiente;
          }
          ++contador;
          break;
        }
        actual = actual->siguiente;
      }
      recorridoPreordenCapas(r->subarbolIzdo(),imagen,numcapas);
      recorridoPreordenCapas(r->subarbolDcho(),imagen,numcapas);
    }
}

void recorridoInordenCapas(NodoCapa* r,NodoImagen* imagen,int* numcapas){
    if (r != NULL && contador!=*numcapas){
      recorridoInordenCapas(r->subarbolIzdo(),imagen,numcapas);
      Capa* actual = imagen->listacapas->primero;
      while(actual!=NULL){
        if(r->capa==atoi(actual->idcapa.c_str())){
          NodoLateral* lateral = r->matrizcapa->laterales->primero;
          while(lateral!=NULL){
            NodoDispersa* nodocolor = lateral->fila->primero;
            while(nodocolor!=NULL){
              auxmatriz->agregarPixel(nodocolor->posx,nodocolor->posy,nodocolor->color);
              nodocolor = nodocolor->derecha;
            }
            lateral = lateral->siguiente;
          }
          ++contador;
          break;
        }
        actual = actual->siguiente;
```

```
        }
        recorridoInordenCapas(r->subarbolDcho(),imagen,numcapas);
    }
}

void recorridoPostordenCapas(NodoCapa* r,NodoImagen* imagen,int* numcapas){
    if (r != NULL && contador!=*numcapas){
        recorridoPostordenCapas(r->subarbolIzdo(),imagen,numcapas);
        recorridoPostordenCapas(r->subarbolDcho(),imagen,numcapas);
        Capa* actual = imagen->listacapas->primero;
        while(actual!=NULL){
            if(r->capa==atoi(actual->idcapa.c_str())){
                NodoLateral* lateral = r->matrizcapa->laterales->primero;
                while(lateral!=NULL){
                    NodoDispersa* nodocolor = lateral->fila->primero;
                    while(nodocolor!=NULL){
                        auxmatriz->agregarPixel(nodocolor->posx,nodocolor->posy,nodocolor->color);
                        nodocolor = nodocolor->derecha;
                    }
                    lateral = lateral->siguiente;
                }
                ++contador;
                break;
            }
            actual = actual->siguiente;
        }
    }
}
```

- Metodos para generar imagen por id:

```
void porListaImagenes(){
    string linea;
    NodoImagen* imagen;
    bool esnumeroexistente = false;
    while(esnumeroexistente!=true){
        cout << "Ingrese el id de la imagen que desea graficar:" << endl;
        getline(cin,linea);
        if(esNumero(linea)){
            imagen = imagenes->buscarImagen(atoi(linea.c_str()));
            if(imagen!=NULL){
                esnumeroexistente = true;
                Capa* capaactual = imagen->listacapas->primero;
                if(capaactual!=NULL){
                    while(capaactual!=NULL){
                        generarImagenporId(arbolcapas->Oraiz(),capaactual);
                        capaactual = capaactual->siguiente;
                    }
                auxmatriz->generarGrafica();
                auxmatriz->generarLienzo();
                auxmatriz = new MatrizCapa();
                } else{
                    auxmatriz->agregarPixel(0,0,"#000000");
```

```
                auxmatriz->generarGrafica();
                auxmatriz->generarLienzo();
                auxmatriz = new MatrizCapa();
            }
        } else{
            cout << "Id no existente en la lista de imagenes, intentelo de nuevo por favor." << endl;
        }
    } else {
        cout << "Id invalido, intentelo con un numero por favor." << endl;
    }
  }

}

void generarImagenporId(NodoCapa* r,Capa* capaactual){
  if (r != NULL){
    if(r->capa==atoi(capaactual->idcapa.c_str())){
        NodoLateral* lateral = r->matrizcapa->laterales->primero;
        while(lateral!=NULL){
            NodoDispersa* nodocolor = lateral->fila->primero;
            while(nodocolor!=NULL){
                auxmatriz->agregarPixel(nodocolor->posx,nodocolor->posy,nodocolor->color);
                nodocolor = nodocolor->derecha;
            }
                lateral = lateral->siguiente;
            }
    } else{
        generarImagenporId(r->subarbolIzdo(),capaactual);
        generarImagenporId(r->subarbolDcho(),capaactual);
    }
  }
```

```
}
```

- ## Metodo para graficar por capa:

```
void porCapa(){
   string linea;
   bool capacorrecta = false;
   while(capacorrecta!=true){
      cout << "Ingrese id de capa a graficar por favor:" << endl;
      getline(cin,linea);
      if(esNumero(linea)){
         NodoCapa* capanula = NULL;
         NodoCapa* nodocapa = arbolcapas->buscarCapa(arbolcapas-
>Oraiz(),capanula,atoi(linea.c_str()));
         if(nodocapa!=NULL){
            capacorrecta=true;
            nodocapa->matrizcapa->generarGrafica();
            nodocapa->matrizcapa->generarLienzo();
         } else{
            cout << "Id de capa no existente en el sistema, intentelo con otro id por favor." << endl;
         }
      } else{
         cout << "Entrada invalida, por favor ingrese un numero." << endl;
      }
   }

}
```

- Método para graficar por usuario:

```
void porUsuario(){
    string linea;
    bool usuarioexiste = false;

    while(usuarioexiste!=true){
        cout << "Usuarios: " << endl;
        usuarios->preorden();
        cout << "Ingrese el nombre del usuario que desea buscar:" << endl;
        getline(cin,linea);
        NodoAvlUsuario* usuarionulo = NULL;
        NodoAvlUsuario* usuario = usuarios->buscarUsuario(usuarios->Oraiz(),usuarionulo,linea);
        if(usuario!=NULL){
            Img* imagen = usuario->listaimagenes->primero;
            if(imagen!=NULL){
                usuarioexiste=true;
                bool imagencorrecta = false;
                while(imagencorrecta!=true){
                    cout << "Seleccione una imagen de la lista de imagenes:" << endl;
                    while(imagen!=NULL){
                        cout << "-" << imagen->idimagen << endl;
                        imagen = imagen->siguiente;
                    }
                    imagen = usuario->listaimagenes->primero;
                    getline(cin,linea);
                    if(esNumero(linea)){
                        NodoImagen* nodoimagen = imagenes->buscarImagen(atoi(linea.c_str()));
                        if(nodoimagen!=NULL){
```

```
                    imagencorrecta=true;
                    Capa* capaactual = nodoimagen->listacapas->primero;
                    if(capaactual!=NULL){
                       while(capaactual!=NULL){
                          generarImagenporId(arbolcapas->Oraiz(),capaactual);
                          capaactual = capaactual->siguiente;
                       }
                       auxmatriz->generarGrafica();
                       auxmatriz->generarLienzo();
                       auxmatriz = new MatrizCapa();
                    } else{
                       auxmatriz->agregarPixel(0,0,"#000000");
                       auxmatriz->generarGrafica();
                       auxmatriz->generarLienzo();
                       auxmatriz = new MatrizCapa();
                    }
                 } else{
                    cout << "Imagen no existe en memoria pero si en carga de archivos, corrige el
archivo para una posterior carga por favor." << endl;
                 }
              } else{
                 cout << "Entrada invalida, seleccione un numero de la lista de imagenes por favor." <<
endl;
              }
           }
        } else{
           cout << "Este usuario no tiene registrado ninguna imagen, busque imagenes en otro
usuario." << endl;
        }

     } else{
        cout << "Usuario no existente en el sistema, ingrese uno de las opciones por favor." << endl;
```

```
    }
  }
}
```

- Métodos para gestión ABC:

```
void gestionarABC(){
  int opcionabc = 0;
  string linea;
  while(opcionabc!=3){
    cout << "+ABC:" << endl;
    cout << "Seleccione una opcion a gestionar:" << endl;
    cout << "1. Usuarios" << endl;
    cout << "2. Imagenes" << endl;
    cout << "3. Salir de ABC" << endl;
    getline(cin,linea);
    if(esNumero(linea)){
      opcionabc = atoi(linea.c_str());
      switch(opcionabc){
        case 1:
          gestionarUsuarios();
          break;
        case 2:
          gestionarImagenes();
          break;
        case 3:
          cout << "Saliendo de gestionar ABC..." << endl;
          break;
        default:
          cout << "Entrada incorrecta, ingrese un numero de las opciones por favor." << endl;
      }
```

```
        } else{
            cout << "Entrada invalida, ingrese un numero de las opciones por favor." << endl;
        }



    }


}

void gestionarUsuarios(){
    string linea;
    int opgestion = 0;
    bool usuariocorrecto = false;
    while(opgestion!=2){
        usuariocorrecto= false;
        cout << "*Gestion de Usuarios:" << endl;
        cout << "1. Agregar usuario:" << endl;
        cout << "2. Salir de gestion de usuarios." << endl;
        getline(cin,linea);
        if(esNumero(linea)){
            opgestion = atoi(linea.c_str());
            switch(opgestion){
                case 1:
                    while(usuariocorrecto!=true){
                        cout << "Ingrese el nombre del usuario a agregar:" << endl;
                        getline(cin,linea);
                        NodoAvlUsuario* usuarionulo = NULL;
                        NodoAvlUsuario* user = usuarios->buscarUsuario(usuarios->Oraiz(),usuarionulo,linea);
```

```
                if(user==NULL){
                    usuariocorrecto=true;
                    NodoAvlUsuario* nuevo = new NodoAvlUsuario(linea);
                    usuarios->insertarAvl(nuevo);
                    cout << "Usuario agregado al sistema." << endl;
                } else{
                    cout << "Usuario ya existente, intentelo con otro nombre por favor." << endl;
                }
            }
            break;
        case 2:
            cout << "Saliendo de gestion de usuarios..." << endl;
            break;
        default:
            cout << "Opcion incorrecta, ingrese un numero de las opciones por favor." << endl;
      }
    } else {
      cout << "Entrada invalida, ingrese un numero de las opciones por favor." << endl;
    }
  }
}



void gestionarImagenes(){
  int opcionimg = 0;
  string linea;
  while(opcionimg!=3){
    cout << "*Gestion de imagenes:" << endl;
    cout << "1. Agregar imagen:" << endl;
    cout << "2. Eliminar imagen " << endl;
```

```
        cout << "3. Salir de gestion de imagenes." << endl;
        getline(cin,linea);
        if(esNumero(linea)){
           opcionimg = atoi(linea.c_str());
           switch(opcionimg){
              case 1:
                 agregarImagenABC();
                 break;
              case 2:
                 eliminarImagenABC();
                 break;
              case 3:
                 cout << "Saliendo de gestion de imagenes..." << endl;
                 break;
              default:
                 cout << "Opcion incorrecta, ingrese un numero de las opciones por favor." << endl;
           }
        } else {
           cout << "Entrada invalida, ingrese un numero de las opciones por favor." << endl;
        }



     }




}

void agregarImagenABC(){
    string linea;
    bool usuariocorrecto = false;
```

```
    bool idcorrecto = false;
    while(usuariocorrecto!=true){
        cout << "Lista de usuarios:" << endl;
        usuarios->preorden();
        cout << "Seleccione un usuario ingresando su nombre por favor." << endl;
        getline(cin,linea);
        NodoAvlUsuario* usuarionulo = NULL;
        NodoAvlUsuario* user = usuarios->buscarUsuario(usuarios->Oraiz(),usuarionulo,linea);
        if(user!=NULL){
            usuariocorrecto = true;
            while(idcorrecto!=true){
                cout << "Ingrese un Id para la imagen por favor." << endl;
                getline(cin,linea);
                if(esNumero(linea)){
                    Img* image = user->listaimagenes->primero;
                    while(image!=NULL){
                        if(image->idimagen==linea){
                            break;
                        }
                        image = image->siguiente;
                    }
                    if(image==NULL){
                        NodoImagen* nodoimagen = imagenes->buscarImagen(atoi(linea.c_str()));
                        if(nodoimagen!=NULL){
                            idcorrecto=true;
                            user->listaimagenes->ingresarImagen(castear(nodoimagen->imagen));
                            cout << "Imagen agregada a la lista de imagenes del usuario exitosamente." << endl;
                        } else{
                            cout << "Esta imagen no se encuentra en el sistema, por lo tanto no se puede
agregar al usuario." << endl;
                        }
```

```
                } else{
                    cout << "Este id de imagen ya esta registrado. intente otro por favor" << endl;
                }
            } else {
                cout << "Entrada invalida, ingrese un numero por favor." << endl;
            }
        }


    } else{
        cout << "Usuario incorrecto, ingrese un usuario de la lista de usuarios por favor." << endl;
    }
  }

}

void eliminarImagenABC(){
  string linea;
  bool usuariocorrecto = false;
  bool idcorrecto = false;
  while(usuariocorrecto!=true){
    cout << "Lista de usuarios:" << endl;
    usuarios->preorden();
    cout << "Seleccione un usuario ingresando el nombre por favor." << endl;
    getline(cin,linea);
    NodoAvlUsuario* usuarionulo = NULL;
    NodoAvlUsuario* user = usuarios->buscarUsuario(usuarios->Oraiz(),usuarionulo,linea);
    if(user!=NULL){
      usuariocorrecto=true;
      while(idcorrecto!=true){
        Img* image = user->listaimagenes->primero;
```

```
            cout << "Lista de imagenes del usuario:" << endl;
            while(image!=NULL){
               cout << "-" << image->idimagen << endl;
               image = image->siguiente;
            }
            cout << "Seleccione el id de la imagen a eliminar por favor." << endl;
            getline(cin,linea);
            if(esNumero(linea)){
               image = user->listaimagenes->buscarImagen(linea);
               if(image!=NULL){
                  idcorrecto=true;
                  string id = image->idimagen;
                  user->listaimagenes->eliminarImagen(id);
                  cout << "Imagen eliminada de la lista de imagenes del usuario." << endl;
                  imagenes->eliminarImagen(atoi(id.c_str()));
               } else{
                  cout << "Id incorrecto, ingrese un id de la lista por favor." << endl;
               }
            } else{
               cout << "Entrada invalida, ingrese un numero de la lista de imagenes por favor." << endl;
            }
         }

      } else{
         cout << "Usuario incorrecto, ingrese un usuario de la lista de usuarios por favor." << endl;
      }
   }
}
```

- Método para el estado de la memoria:

```
void estadoMemoria(){
    string linea;
    int opcionmemoria = 0;
    bool idcorrecto = false;
    while(opcionmemoria!=7){
        cout << "+Estado de la Memoria" << endl;
        cout << "Seleccione una opcion:" << endl;
        cout << "1. Ver lista de imagenes " << endl;
        cout << "2. Ver arbol de capas" << endl;
        cout << "3. Ver arbol de capas espejo" << endl;
        cout << "4. Ver capa" << endl;
        cout << "5. Ver imagen y arbol de capas" << endl;
        cout << "6. Ver arbol de usuarios" << endl;
        cout << "7. Salir del estado de la memoria" << endl;
        getline(cin,linea);

        if(esNumero(linea)){
            opcionmemoria = atoi(linea.c_str());

            switch(opcionmemoria){
                case 1:
                    imagenes->generarGrafica();
                    break;
                case 2:
                    arbolcapas->generarGrafica();
                    break;
                case 3:
                    arbolcapas->generarGraficaEspejo();
```

```
            break;
        case 4:
            while(idcorrecto!=true){
                cout << "Ingrese el id de la capa a mostrar:" << endl;
                getline(cin,linea);
                if(esNumero(linea)){
                    NodoCapa* capanula = NULL;
                    NodoCapa* capa = arbolcapas->buscarCapa(arbolcapas->Oraiz(),capanula,atoi(linea.c_str()));
                    if(capa!=NULL){
                        idcorrecto = true;
                        capa->matrizcapa->generarGrafica();
                    } else{
                        cout << "Capa no registrada en el sistema, intento con otro numero por favor." << endl;
                    }
                } else{
                    cout << "Entrada invalida, ingrese un numero de capa por favor." << endl;
                }
            }
            break;
        case 5:
            verImagenCapas();
            break;
        case 6:
            usuarios->generarGrafica();
            break;
        case 7:
            cout << "Saliendo del estado de la memoria." << endl;
            break;
        default:
            cout << "Opcion incorrecta, ingrese un numero de las opciones por favor." << endl;
```

```
        }

    } else{
        cout << "Entrada invalida, ingrese un numero de las opciones por favor." << endl;
    }



  }
}
```

Clases de estructuras principales

- ## Arbol AVL Usuarios:

```
#ifndef ARBOLAVLUSUARIOS_H
#define ARBOLAVLUSUARIOS_H
#include <iostream>
#include <fstream>
#include <windows.h>
#include "ListaSimpleImagenes.h"

using namespace std;

class NodoAvlUsuario{
public:

  string usuario;
  NodoAvlUsuario* izdo;
  NodoAvlUsuario* dcho;
```

```
int fe;
ListaSimpleImagenes* listaimagenes;

NodoAvlUsuario(string valor){
usuario = valor;
izdo = dcho = NULL;
fe = 0;
listaimagenes = new ListaSimpleImagenes();
}

string valorNodo(){
   return usuario;
}
NodoAvlUsuario* subarbolIzdo(){
   return izdo;
}
NodoAvlUsuario* subarbolDcho(){
   return dcho;
}
void nuevoValor(string d){
   usuario = d;
}
void ramaIzdo(NodoAvlUsuario* n){
   izdo = n;
}
void ramaDcho(NodoAvlUsuario* n){
   dcho = n;
}
void visitar(){
   std::cout << "-" << usuario << std::endl;
}
```

```
    void imprimir(){
        std::cout << usuario << std::endl;
    }

    void Pfe(int vfe){
        fe = vfe;
    }
    int Ofe(){
        return fe;
    }

};


class ArbolAvlUsuarios {
public:
    NodoAvlUsuario* raiz;

    ArbolAvlUsuarios(){
        raiz = NULL;
    }

    NodoAvlUsuario* Oraiz();
    void Praiz(NodoAvlUsuario *r);
    void preorden();
    void arbolespejo();
    void insertarAvl(NodoAvlUsuario* nuevo);
    void inorden();
    void postorden();
    void generarGrafica();
```

```
    NodoAvlUsuario* buscarUsuario(NodoAvlUsuario* r,NodoAvlUsuario* usuarionulo,string user);
    int NumeroUsuarios();
private:
    NodoAvlUsuario* rotacionII(NodoAvlUsuario * n, NodoAvlUsuario* n1);
    NodoAvlUsuario* rotacionDD(NodoAvlUsuario* n, NodoAvlUsuario* n1);
    NodoAvlUsuario* rotacionDI(NodoAvlUsuario* n, NodoAvlUsuario* n1);
    NodoAvlUsuario* rotacionID(NodoAvlUsuario* n, NodoAvlUsuario* n1);
    NodoAvlUsuario* insertarAvl(NodoAvlUsuario* raiz, NodoAvlUsuario* nuevo, bool &hc);
    void preorden(NodoAvlUsuario* r);
    void inorden(NodoAvlUsuario* r);
    void arbolespejo(NodoAvlUsuario* r);
    void escribirDot(NodoAvlUsuario* r);
    void escribirImagenes(NodoAvlUsuario *r);
    void postorden(NodoAvlUsuario* r);
    int NumeroUsuarios(NodoAvlUsuario* r,int numusuarios);
    string dotarbolavl = "";
    string dotimg = "";
};
```

- ## Arbol binario de Búsqueda:

```
#ifndef ARBOLCAPAS_H
#define ARBOLCAPAS_H
#include <iostream>
#include "MatrizCapa.h"

class NodoCapa{
public:

    int capa;
    MatrizCapa* matrizcapa;
```

```
    NodoCapa* izq;
    NodoCapa* der;

    NodoCapa(int capa){
        this->capa = capa;
        izq = der = NULL;
        matrizcapa = new MatrizCapa();
    }

    NodoCapa(NodoCapa* ramaIzdo, int capa, NodoCapa* ramaDcho){
        this->capa = capa;
        izq = ramaIzdo;
        der = ramaDcho;
        matrizcapa = new MatrizCapa();
    }

    //Operaciones de acceso
    int getCapa(){ return capa; }
    NodoCapa* subarbolIzdo(){ return izq; }
    NodoCapa* subarbolDcho(){ return der; }

    // operaciones de modificación
    void ramaIzdo(NodoCapa* n){ izq = n; }
    void ramaDcho(NodoCapa* n){ der = n; }

    void visitar(){
        cout << "Capa: " << castear(capa) << endl;
    }

private:
    string castear(int i){
```

```
        std::string cadenai = static_cast<std::ostringstream*>(&(std::ostringstream() << i))->str();
    return cadenai;
    }

};



class ArbolCapas {
public:
    ArbolCapas(NodoCapa* r){
        raiz=r;
    }
    ArbolCapas(){
        raiz = NULL;
    }

    NodoCapa* buscarCapa(NodoCapa* r,NodoCapa* capanula,int capa);
    void Preorden();
    void inorden();
    void Postorden();
    void Praiz(NodoCapa* r);
    NodoCapa* Oraiz();
    bool esVacio();
    NodoCapa* hijoIzdo();
    NodoCapa* hijoDcho();
    void insertar(NodoCapa* nuevo);
    void generarGrafica();
    void generarGraficaEspejo();
    void CapasHojas(NodoCapa* r);
    int profundidad(NodoCapa* r);
private:
```

```
  NodoCapa* insertar(NodoCapa* raizSub,NodoCapa* nuevo);
  void escribirdot(NodoCapa *r);
  void escribirdotEspejo(NodoCapa *r);
  void Preorden(NodoCapa* r);
  void inorden(NodoCapa* r);
  void Postorden(NodoCapa* r);
  string castear(int i);
  string dotarbol = "";
  string dotarbolespejo = "";
protected:
   NodoCapa* raiz;

};
```

- ## Matriz Dispersa:

```
class NodoDispersa {

public:

   int posx, posy;
   std::string color;
   NodoDispersa* arriba;
   NodoDispersa* abajo;
   NodoDispersa* izquierda;
   NodoDispersa* derecha;
```

```
    NodoDispersa(int x, int y, std::string color){
        posx = x;
        posy = y;
        this->color = color;
        arriba = NULL;
        abajo = NULL;
        izquierda = NULL;
        derecha = NULL;
    }

};


class MatrizCapa{
public:
    ListaCabeceras* cabeceras;
    ListaLaterales* laterales;

    MatrizCapa(){
        cabeceras = new ListaCabeceras();
        laterales = new ListaLaterales();
    }



    void agregarPixel(int x, int y, string color);
    void generarGrafica();
    void generarLienzo();
private:
```

```
    string escribirMatrizCapa();
    string escribirLienzo();
    string castear(int i);

};
```