Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2022.

Proyecto 1: Gestor de Pisos Artesanales S.A.

202010044 – Ramiro Agustín Télles Carcuz

Resumen

El programa gestor de pisos artesanales puede leer distintas listas de pisos por medio de un archivo xml y después poder gestionar de buena manera los pisos ingresados.
El programa ordena los pisos y sus patrones alfabéticamente y también incluye otras funciones para poder hacer una mejor gestión de los pisos que tiene que manejar la empresa Pisos Artesanales S.A.

Este programa incluye la opción de poder graficar los patrones de los pisos para poder visualizar el trabajo terminado, así como la habilidad de calcular la forma mas optima y barata en la que un patrón de piso puede ser intercambiado por otro.

Esto sumándole una interfaz amigable con el usuario para que sea mucho más simple e interactivo su uso para los trabajadores.

Estas y otras funciones implementadas con una gestión de memoria dinámica para lograr la mayor eficiencia en el programa y que sea de gran utilidad para Pisos Artesanales S.A.

Palabras clave

Pisos, Artesanales,Software,TDA,memoria dinamica

Abstract

The artisanal tiles manager can read different list of tiles by a xml file and it can manage the tiles entered in a good way.

The program can sort the tiles and its patterns alphabetically and the program includes others functions to be able to manage better the tiles that Pisos artesanales S.A. need to manage.

This program includes the option to graphic the patterns of the tiles and be able to visualize the final work, as well as the ability to calculate the most efficient and cheap way to change the patterns.

It has a friendly interface to make the program more simpler and interactive for workers.

This and another functions implemented with a dynamic memory manager to make the program more efficient and more useful to Pisos Artesanales S.A.

Keywords

Tiles, artesanal, Software,TDA,Dynamic Memory

Introducción

Aquí se presenta la solución de software pedida por Pisos Artesanales S.A., la cual pidió una solución de software en la que este pueda llevar el control de una mejor manera de su producto.

Este pide un software el cual le pueda manejar los pisos artesanales que ellos venden, así como poder enlistarlos de manera alfabética, el poder ver los patrones de los pisos con una grafica hecha en graphviz y para poder calcular cual seria la mejor manera para poder intercambiar patrones para poder darle el menor precio a sus clientes al momento de que quieran intercambiar pisos.

Para poder llevarlo a cabo se implemento una lista doblemente enlazada para poder llevar una gestión de memoria más eficiente y controlada de los pisos que se agregarán al sistema.

Así como las opciones de poder graficar un patrón y el poder calcular el costo del cambio de cada patrón.

Desarrollo del tema

A. Lista doblemete enlazada

Este programa implementa una lista doblemente enlazada para poder llevar a cabo la buena gestión y los cálculos que se tengan que hacer al momento de manipular los pisos y sus patrones.

Se implementaron los nodos que serán los elementos que tendrá la lista los cuales se les asignaran los datos que el programa necesite para poder hacer una buena gestión de memoria.

Los nodos tienen 3 atributos que son el dato que contendrán, el apuntador al siguiente nodo, y el apuntador al nodo anterior.

La clase lista se encargará de poder gestionar todos los nodos para poder ingresar a cada nodo de la lista y poder obtener la información que tienen.

Tienen como atributos una variable booleana que es vacio y esta dice si la lista está vacia o no, otra variable entera llamada cant que lleva la cantidad de elementos que tiene la lista y un apuntador llamado inicio que apunta al primer nodo de la lista.

Esta lista tiene los siguientes métodos

- Imprimir: Imprime los elementos de la lista.
- Remplazar: remplaza el dato de un nodo en la posición deseada.
- EliminarPos: Elimina un nodo de la lista según la posición deseada
- EliminarDato: Elimina un nodo de la lista según el dato dado
- Agregar_inicio. Agrega un nodo al inicio
- Agregar_Final: Agrega un nodo al final de la lista.
- Buscar: busca si existe un dato en la lista.
- getPos: devuelve el dato de la lista según la posición dada.

B. Pisos y patrones

El programa lleva la clase pisos que es una clase en la que guardará todos los datos que sean necesarios para el manejo de los pisos en el programa.

El programa también lleva la clase patrones que es una clase en la que guardará todos los datos que sean necesarios para el manejo de los patrones en el programa.

La clase patrones lleva como atributos:

- Codigo: código que el patrón llevara
- Cadena: Una cadena que describe el patrón que podrá llevar el piso

La clase piso tiene los atributos:

- Nombre: guardará el nombre del piso
- R: cantidad de filas del piso
- C: cantidad de columnas del piso
- F: costo que tendrá el voltear un piso
- S: Costo que tendrá el intercambiar un piso con el otro
- Patrones: Una lista de objetos tipo patron que el piso podrá tener.

Para poder llevar una buena gestión de los pisos y patrones que se lleven en el programa, se hizo uso de la lista doblemente enlazada para poder almacenar los pisos y patrones para hacer un uso correcto y eficiente dentro del sistema.

C. Ingreso de los pisos al programa

Para poder ingresar los pisos al sistema se hizo uso del lenguaje xml y de una estructura simple en la que almacenaba la información del nombre del piso, el valor de sus columnas(C), el valor de sus filas(R), el costo de voltear(F), el costo de intercambiar(S) y la lista de patrones que el piso podía tener. Y la lista de patrones contenían la información del código(codigo) con el que identificaba el patron y la cadena(cadena) que definia como iba a hacer el patron.

Para poder leer el xml se utilizó una librería llamada element tree para poder leer todo el archivo xml y después poder leerlo y extraer la información que el archivo xml traía.

D. Clase metodos

Se tiene a la clase métodos, la cual contiene todos los métodos útiles que se podrán utilizar para poder manipular lista, pisos y patrones para el correcto funcionamiento del programa.

La clase métodos no contiene ningún atributo, pero contiene métodos importantes para el correcto funcionamiento del programa.

Métodos de la clase métodos:

- ordenarPisos: Este toma una lista de pisos y lo ordena alfabéticamente
- ordenarPatrones: Este toma una lista de patrones y lo ordena alfabéticamente.
- cambiarPatron: realiza el calculo de cuales serán los movimientos mas eficientes y menos costosos para poder cambiar un patron a otro. Este método retorna el precio que costará realizar el cambio del patron, los pasos a realizar para poder realizar el cambio del patron y el patron resultante después del cambio hecho
- graficarPatron: Este método crea un archivo .dot el cual contiene las instrucciones que graphviz tendrá que ejecutar para poder graficar el patrón.

E. Interfaz y uso de la aplicación

En este programa se implementó una interfaz amigable e interactiva para que fuera mucho mas fácil y simple su uso.

El programa lanza primero un menú en el que la primera opción es ingresar los pisos que se quieren gestionar por medio de un archivo xml, al seleccionar esta opción el programa pedirá la ruta del archivo xml para proceder a hacer la carga de los pisos al sistema.

La opción 2 es lista de pisos y al seleccionar está se le mostrarán todos los pisos cargados al sistema ordenados alfabéticamente para poder ubicar de mejor manera estos mismos.

Si se selecciona la opción 3, el programa terminará de ejecutarse.

F. Dentro de la lista pisos

Una vez cargados los pisos al programa, se puede entrar a la lista de los pisos, en este menú se le pedirá seleccionar un piso de todos los que están cargados en memoria, si se llega a equivocar, habrá una opción volver a la cual podrá volver a la lista de piso.

Una vez seleccionado un piso se le pedirá si desea la opción 1, graficar patron o opción 2, cambiar patron de un piso.

Si selecciona la opción 1, graficar patron, se le pedirá que seleccione el patron a graficar, después de seleccionar el patron, el programa creará la grafica y la mostrará en pantalla y luego volverá al menú de selección de graficar o cambiar patrón.

Si se selecciona la opción cambiar patron, el programa pedirá que se seleccione el patron de inicio, que es el patron que esta al principio, y el patron final que será el patron al cual cambiará el piso. Al momento de seleccionar el patron final, el programa calculará que movimientos se deben hacer para poder cambiar el patron y el costo mínimo que tendrá el cambio del patron, luego imprimirá en pantalla el patron que quedó al cambiar patrones.

unm asdasdlenguaje técnico preciso, organizado de preferencia en párrafos cortos.

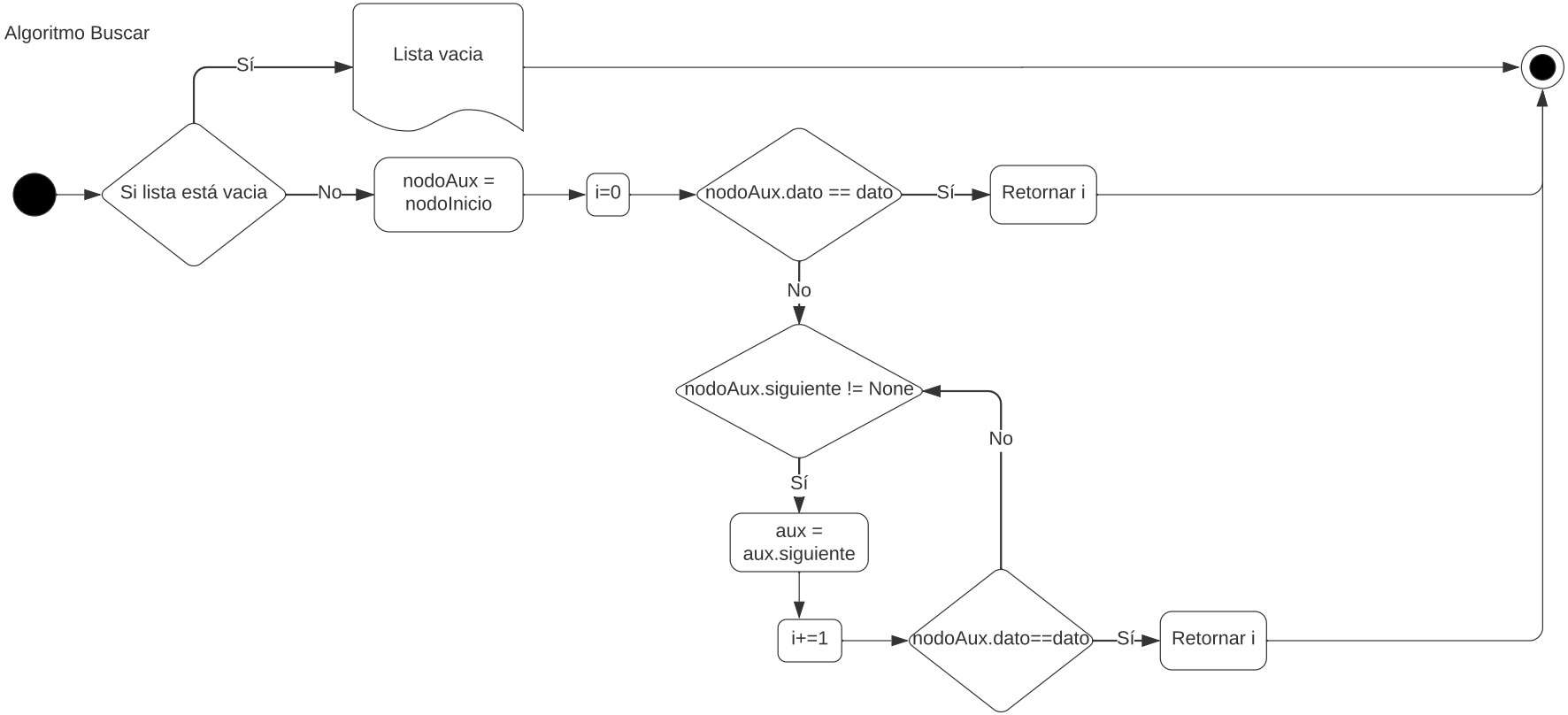
Conclusiones

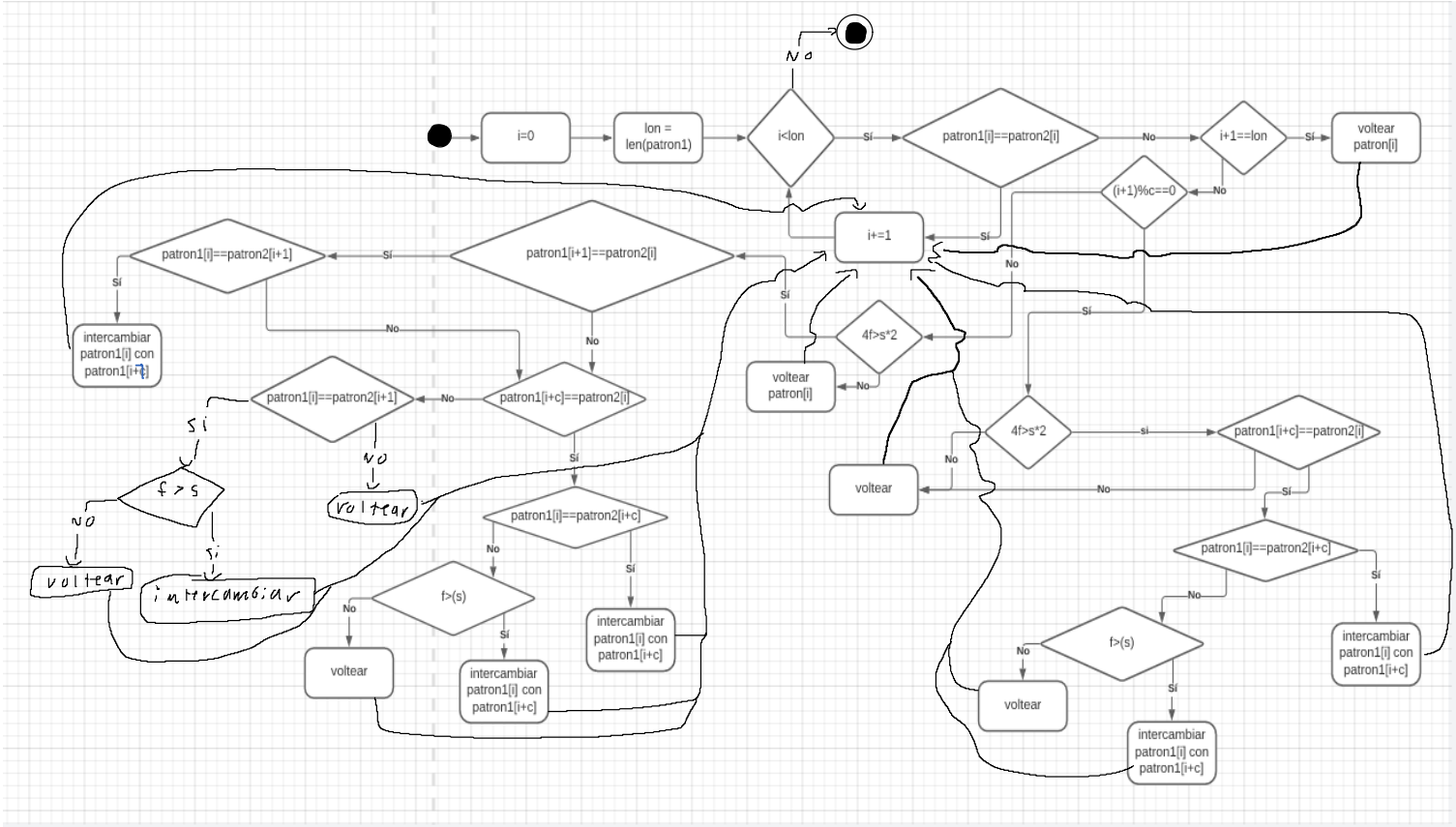
Se logró realizar un programa que utiliza una lista doblemente enlazada para usar la memoria de manera eficiente

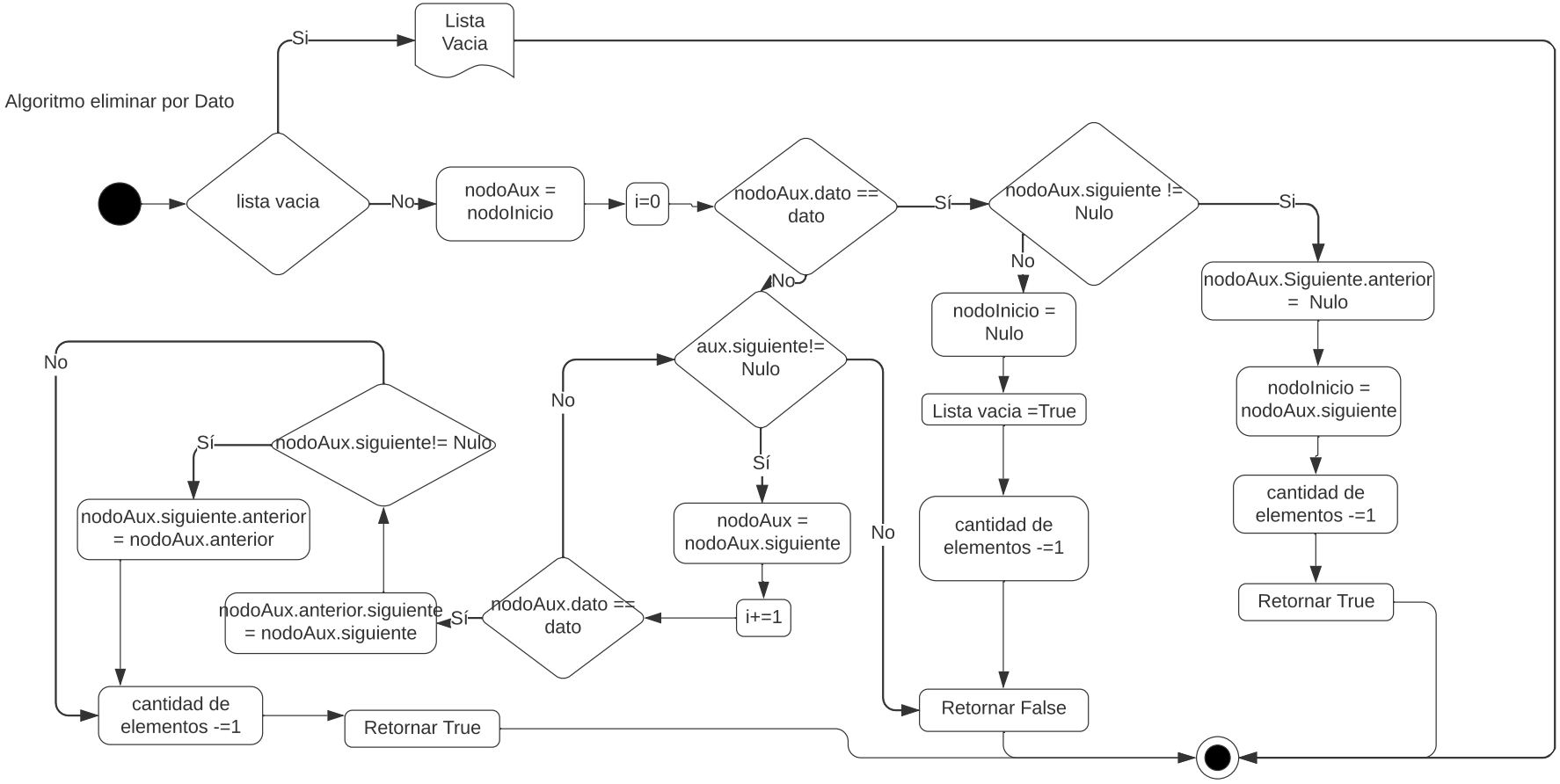
Se logró realizar un programa que calcula la manera más barata de poder hacer el cambio de un patrón a otro y que pasos realizar para poder llegar a esa respuesta

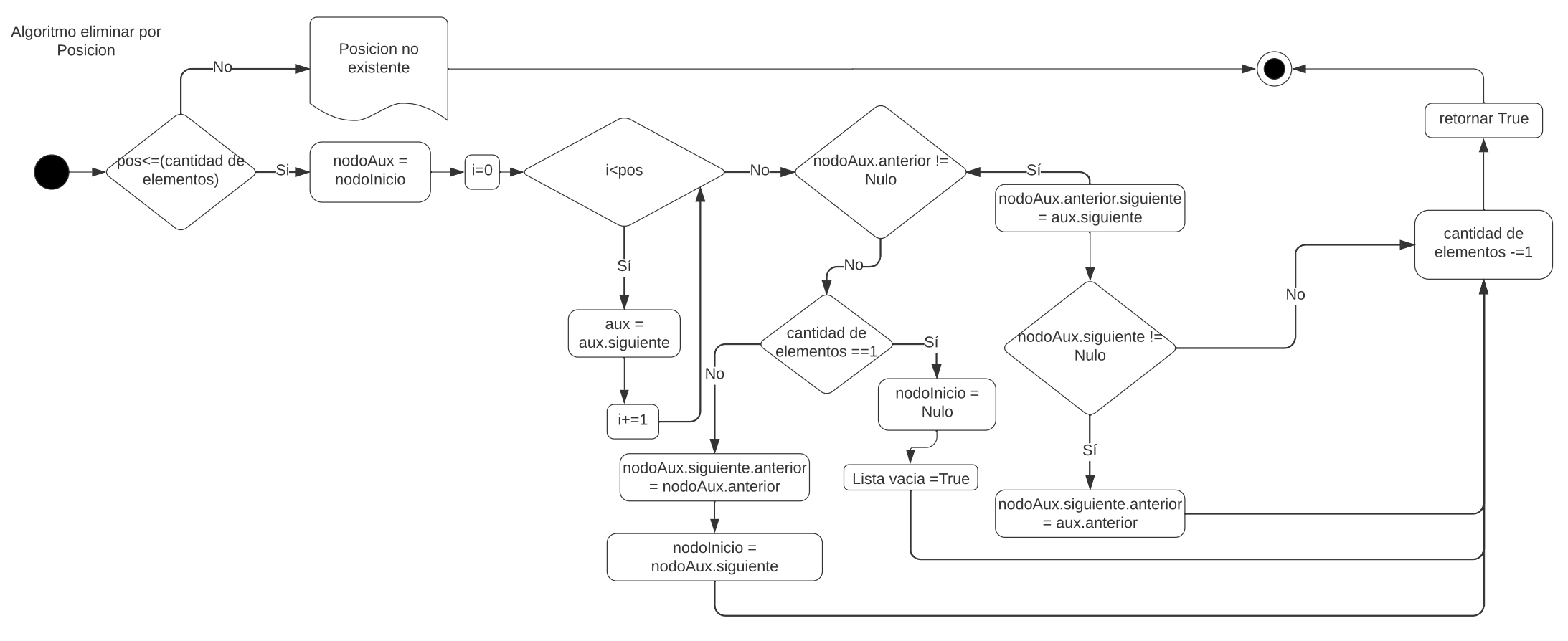
Se realizó un programa que es capaz de llevar el control de todos los pisos y patrones de una forma amigable y interectiva para el usuario.

Anexos

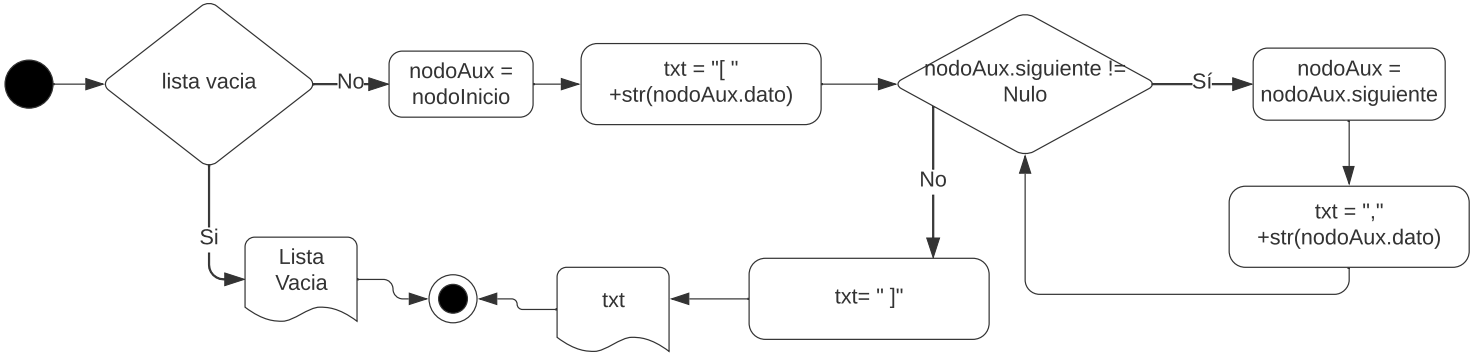

Algoritmo Buscar
Sí
Lista vacia
Si lista está vacia
No
nodoAux = nodoInicio
i=0
nodoAux.dato == dato
Sí
Retornar i
No
nodoAux.siguiente != None
No
Sí
aux = aux.siguiente
i+=1
nodoAux.dato==dato
Sí
Retornar i


No
i=0
lon = len(patron1)
i<lon
Sí
patron1[i]==patron2[i]
No
i+1==lon
Sí
voltear patron[i]
(i+1)%c==0
No
i+=1
Sí
No
patron1[i]==patron2[i+1]
Sí
patron1[i+1]==patron2[i]
Sí
intercambiar patron1[i] con patron1[i+1]
No
No
patron1[i]==patron2[i+1]
No
patron1[i+c]==patron2[i]
Sí
Si
f>s
NO
voltear
Si
intercambiar
NO
voltear
Sí
patron1[i]==patron2[i+c]
No
f>(s)
No
voltear
Sí
intercambiar patron1[i] con patron1[i+c]
Sí
intercambiar patron1[i] con patron1[i+c]
Sí
4f>s*2
No
voltear patron[i]
Sí
4f>s*2
si
patron1[i+c]==patron2[i]
No
voltear
No
Sí
patron1[i]==patron2[i+c]
No
f>(s)
No
voltear
Sí
intercambiar patron1[i] con patron1[i+c]
Sí
intercambiar patron1[i] con patron1[i+c]

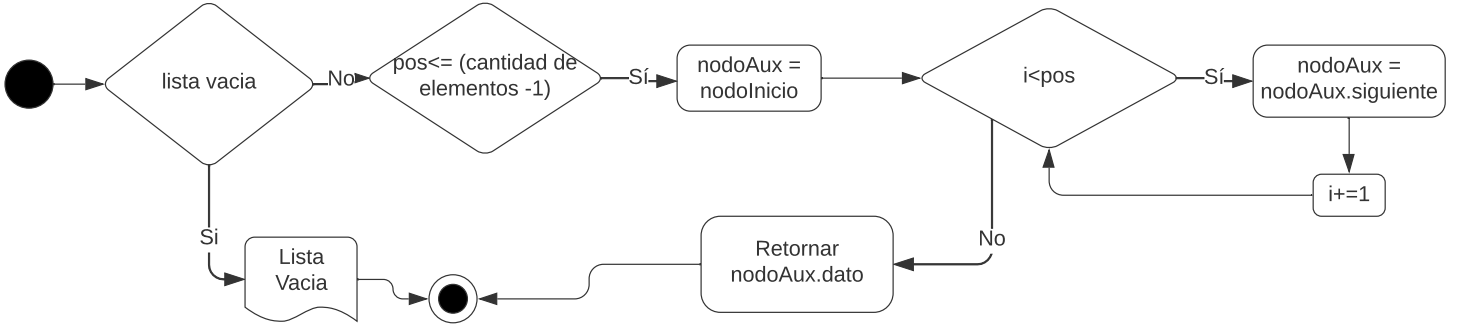

Algoritmo eliminar por Dato
lista vacia
Si
Lista Vacia
No
nodoAux = nodoInicio
i=0
nodoAux.dato == dato
Sí
nodoAux.siguiente != Nulo
Si
nodoAux.Siguiente.anterior = Nulo
nodoInicio = nodoAux.siguiente
cantidad de elementos -=1
Retornar True
No
nodoInicio = Nulo
Lista vacia =True
cantidad de elementos -=1
Retornar False
No
aux.siguiente!= Nulo
Sí
nodoAux = nodoAux.siguiente
i+=1
nodoAux.dato == dato
Sí
nodoAux.anterior.siguiente = nodoAux.siguiente
No
nodoAux.siguiente!= Nulo
Sí
nodoAux.siguiente.anterior = nodoAux.anterior
No
cantidad de elementos -=1
Retornar True

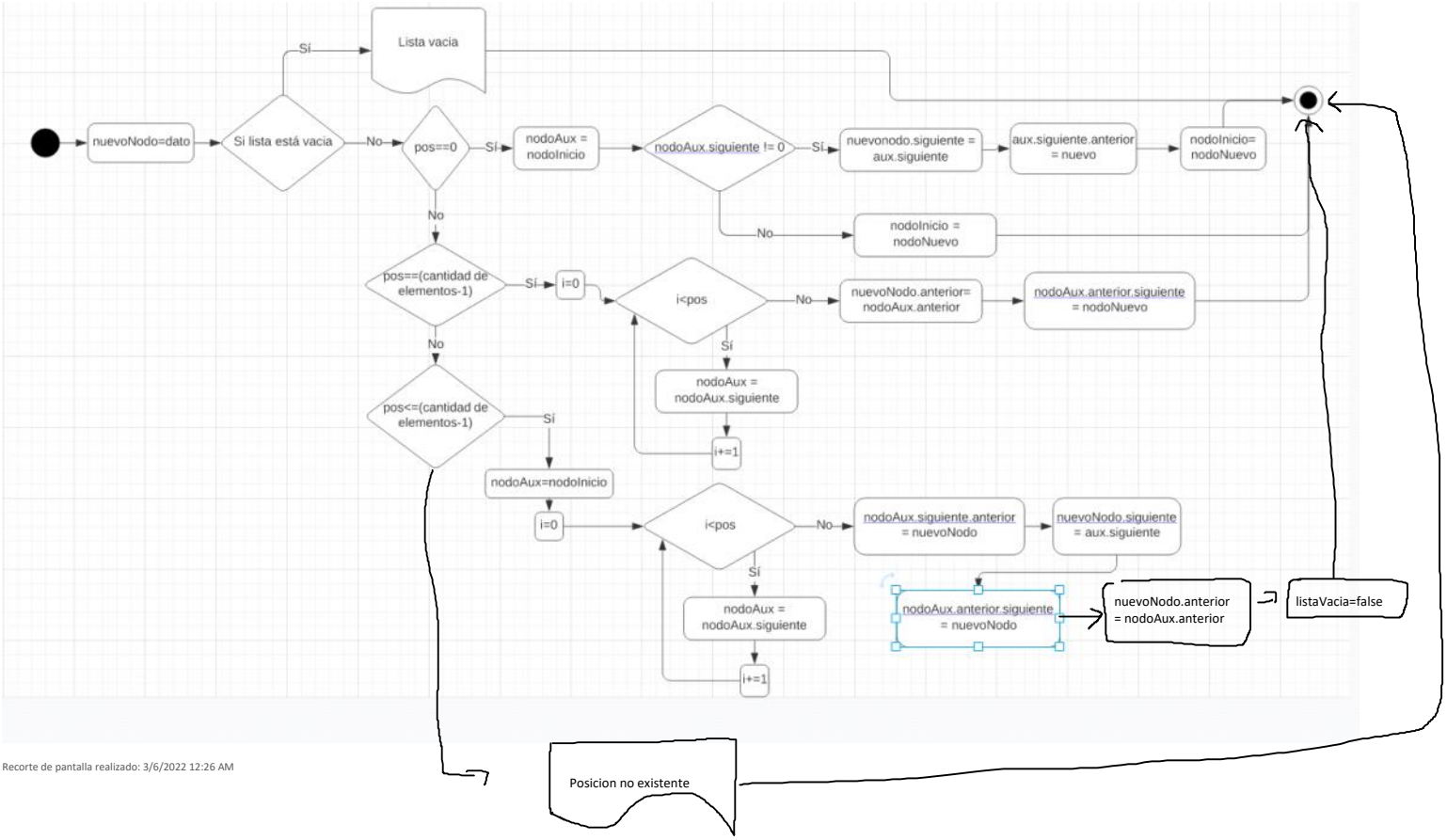

Algoritmo eliminar por Posicion
pos<=(cantidad de elementos)
No
Posicion no existente
Si
nodoAux = nodoInicio
i=0
i<pos
No
Sí
aux = aux.siguiente
i+=1
nodoAux.anterior != Nulo
Sí
nodoAux.anterior.siguiente = aux.siguiente
No
cantidad de elementos ==1
Sí
nodoInicio = Nulo
Lista vacia =True
No
nodoAux.siguiente.anterior = nodoAux.anterior
nodoInicio = nodoAux.siguiente
nodoAux.siguiente != Nulo
No
Sí
nodoAux.siguiente.anterior = aux.anterior
cantidad de elementos -=1
retornar True

algoritmo Imprimir

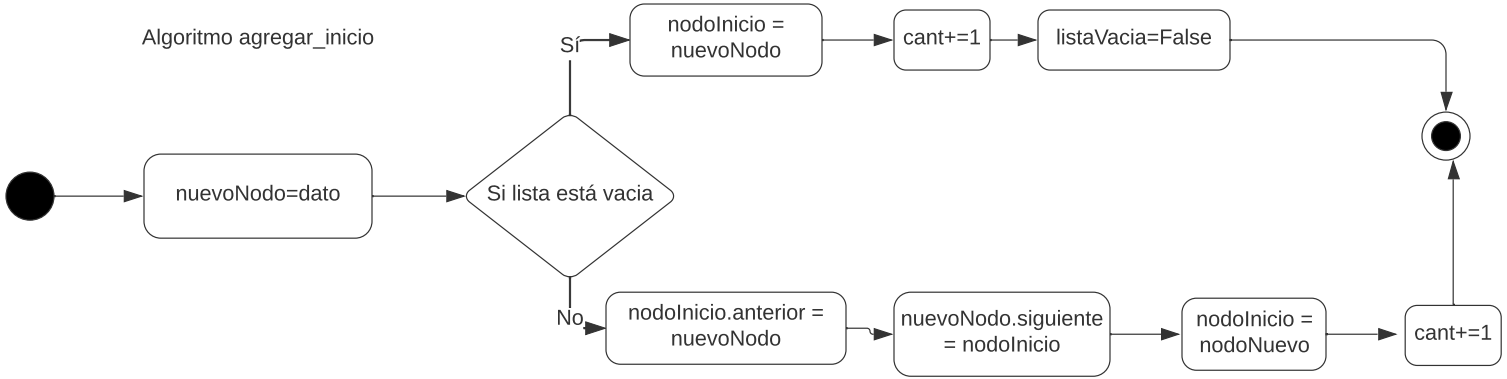


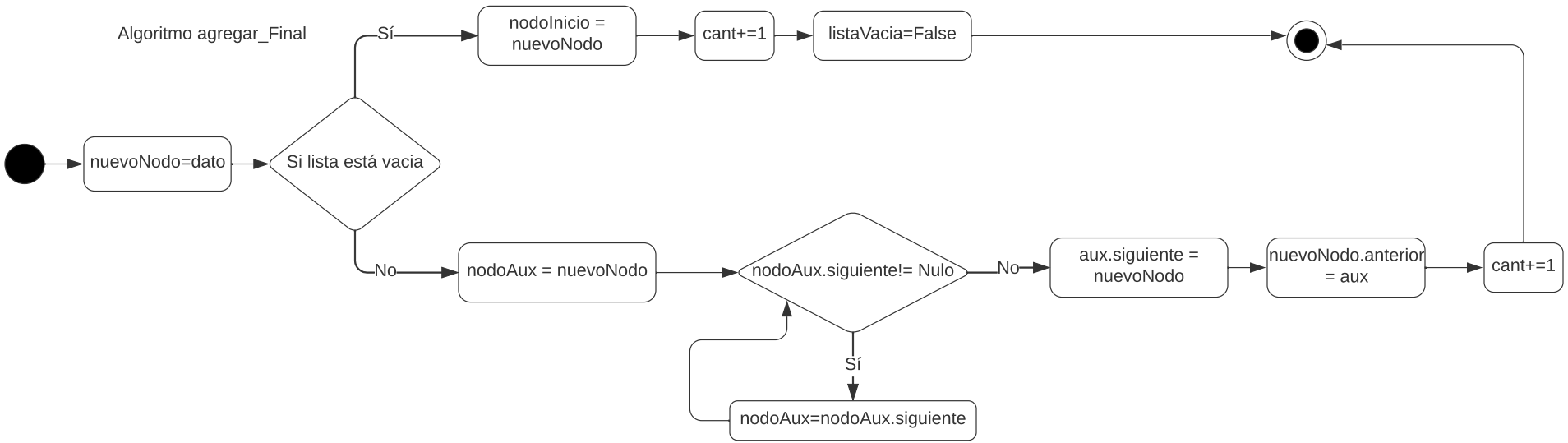
algoritmo obtener dato por posicion

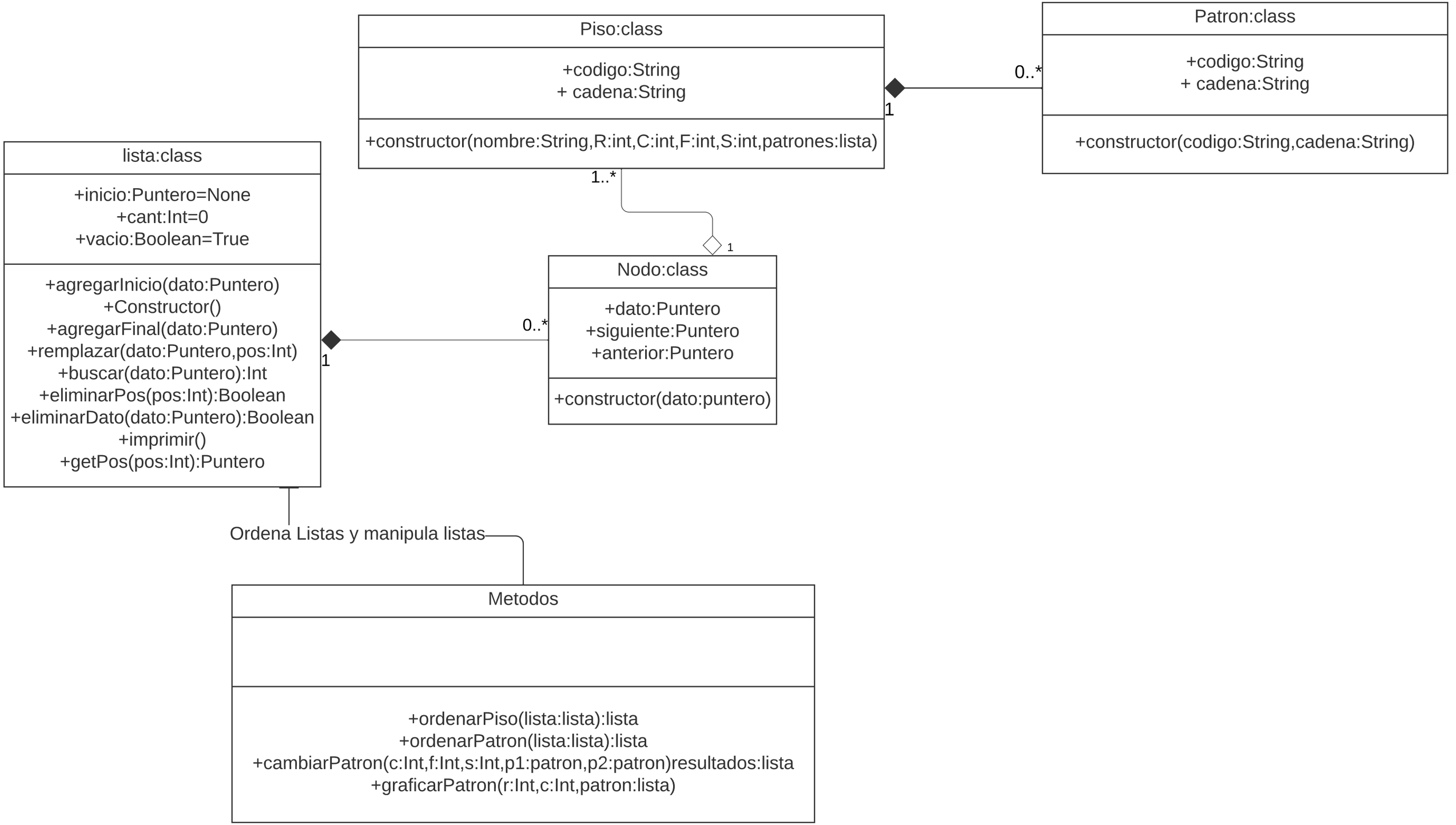




Recorte de pantalla realizado: 3/6/2022 12:26 AM


Algoritmo agregar_inicio
nuevoNodo=dato
Si lista está vacia
Sí
nodoInicio = nuevoNodo
cant+=1
listaVacia=False
No
nodoInicio.anterior = nuevoNodo
nuevoNodo.siguiente = nodoInicio
nodoInicio = nodoNuevo
cant+=1


Algoritmo agregar_Final
nuevoNodo=dato
Si lista está vacia
Sí
nodoInicio = nuevoNodo
cant+=1
listaVacia=False
No
nodoAux = nuevoNodo
nodoAux.siguiente!= Nulo
No
aux.siguiente = nuevoNodo
nuevoNodo.anterior = aux
cant+=1
Sí
nodoAux=nodoAux.siguiente


Piso:class
+codigo:String
+ cadena:String
+constructor(nombre:String,R:int,C:int,F:int,S:int,patrones:lista)
Patron:class
+codigo:String
+ cadena:String
+constructor(codigo:String,cadena:String)
0..*
1
1..*
1
lista:class
+inicio:Puntero=None
+cant:Int=0
+vacio:Boolean=True
+agregarInicio(dato:Puntero)
+Constructor()
+agregarFinal(dato:Puntero)
+remplazar(dato:Puntero,pos:Int)
+buscar(dato:Puntero):Int
+eliminarPos(pos:Int):Boolean
+eliminarDato(dato:Puntero):Boolean
+imprimir()
+getPos(pos:Int):Puntero
1
0..*
Nodo:class
+dato:Puntero
+siguiente:Puntero
+anterior:Puntero
+constructor(dato:puntero)
Ordena Listas y manipula listas
Metodos
+ordenarPiso(lista:lista):lista
+ordenarPatron(lista:lista):lista
+cambiarPatron(c:Int,f:Int,s:Int,p1:patron,p2:patron)resultados:lista
+graficarPatron(r:Int,c:Int,patron:lista)

Referencias bibliográficas

M. seidl, M. Scholz, C. Huemer, G. Kappel, (2012).

An introduction to Object-Oriented Modeling. Springer International Publishing AG.