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 2021.

APLICACIÓN DE MEMORIA DINÁMICA Y REDUCCIÓN DE UNA COLECCIÓN DE DATOS CON ARQUITECTURA EN MATRIZ

201709020 – Jaime Efraín Chiroy Chavez

Resumen

El proyecto consiste en el análisis de archivos de extensión xml, el cual posee los registros de matrices y los datos de sus elementos en sus respectivas posiciones, los datos leídos se almacenan en memoria con el uso de un TDA de tipo lista circular simplemente enlazada, estas matrices son procesadas mediante un algoritmo que registra el patrón de cada tupla de datos (fila de matriz) y valida si hay patrones similares, si se encuentran coincidencias los valores de estas tuplas se sumarán, obteniendo de esta manera la reducción de matrices, la aplicación podrá mostrar los resultados en un archivo de salida de formato xml, los registros de la matriz (todos sus atributos y su matriz original) podrán visualizarse al generar un grafo a través de la aplicación.

Palabras clave

TDA – Tipo de Dato Abstracto, Matriz, Nodo, Lista enlazada, Tuplas.

Abstract

This project consists of an analysis of files with an xml extension, where the matrix record and its element data are in their respective locations, and the data read is stored in memory using a linked circular list type TDA. Each of these matrices is processed by an algorithm that creates a pattern for each data tuple (matrix row) and verifies for similar patterns. If a match is found, the values of these tuples are added, thus allowing the application to show the results in an output file in xml format, and the matrix record (all attributes and the original matrix). You can view it when generate the graph through the application.

Keywords

ADT – Abstract Data Type, Matrix, Node, Linked list, Tuples.

.

Introducción

Con la implementación de TDA's para el almacenamiento de matrices de tamaño desconocido, el uso de memoria dinámica resulta ser muy conveniente, una desventaja es la lógica del manejo de éstas, ya que, pueden llegar a ser confusas como parte de la solución al planteamiento de la reducción para tuplas de datos; para el análisis del patrón binario de los valores de las tuplas, las tuplas que coincidan en sus patrones se sumarán para generar una matriz reducida, de estos datos se almacenarán los grupos y la frecuencia de cada uno, la solución proveerá validaciones que controlarán errores de ejecución o lectura obteniendo así una aplicación altamente confiable; una vez obtenida la solución (matriz reducida), el programa deberá ser capaz de generar archivos de salida (grafo y xml) para visualizar los datos almacenados y procesados.

Desarrollo del tema

Los requerimientos del software son:

-Implementación de POO y TDA
-Lectura de archivos de entrada (Formato XML)
-Creación de archivos de salida (Formato XML)
-Operaciones con TDA
-Generar un grafo de una matriz seleccionada

MODULO – Listas_Nodos.py
Especificaciones de TDA

Ante la restricción de no uso de array's, se considera factible el uso de una lista circular enlazada superior que contenga los atributos simples, matrices y grupos en forma de sub listas enlazadas circulares.

Comenzando con la definición del TDA para la "lista matriz" se considera:

```
Nodo
|   Fila
|   Columna
|   Valor
|   Siguiente: Nodo
Fin Nodo
```

La clase "**Nodo**" contendrá todos los atributos que conforman a la matriz, tanto el valor como sus índices dentro de la matriz, así como el puntero a un siguiente objeto de la misma clase, estos atributos únicamente tendrán los métodos *getters* y *setters*.

```
ListaEnlazadaCircular
|   inicio: Nodo
Fin ListaEnlazadaCircular
```

La clase "**Lista enlazada circular**" se definirá con un solo atributo de tipo Nodo, sus posibles operaciones son:

Inicializar: Colocar a null el apuntador inicio de la lista.

estaVacía: Retornará True si *inicio* es null o de lo contrario retornará False.

Agregar:

Si está vacía(inicio/final): Se crea un nodo que se asigna al inicio y se autorreferencia a sí mismo.

Al Inicio(No vacía): Se crea un nuevo nodo, el puntero de este enlazará al inicio, con un while se recorrerá la lista hasta encontrar el nodo (último) que apunta al inicio, al encontrar el último nodo de la lista, el puntero de este cambiara hacia el nuevo nodo, y el nuevo nodo será asignado como el nuevo inicio.

Al Final(No vacía): Se crea un nuevo nodo, el puntero de este enlazará al inicio, con un while se recorrerá la lista hasta encontrar el nodo (último) que apunta al inicio, al encontrar el último nodo de la lista, el puntero de este cambiara hacia el nuevo nodo, *sin cambiar el inicio*.

getInicio: Retorna el nodo inicio.

generaListaDeDimension: Recibe dos valores enteros (fila y columna), a través de dos ciclos for, se añaden nodos por la operación agregar al Final dándole al nodo un valor de relleno con la fila y columna del ciclado.

evaluaLista: Analizará si todos los nodos de la lista poseen un valor distinto al valor de "relleno" dado por la operación generaListaDeDimension, retornará True si los valores de relleno fueron modificados, o False, si hay valores de relleno en alguno de los nodos.

ReemplazaDatos: Recibe tres valores (fila, columna, valor), se recorrerán todos los nodos en busca de coincidencias para los atributos fila y columna de los nodos y al hallar una coincidencia, el atributo valor (dado por la operación generaListaDD) de ese nodo se reemplazará con el valor recibido.

dameDatos: Es una operación de búsqueda que recibirá dos valores (fila y columna), se recorrerá la lista y al hallar una coincidencia el atributo valor será retornado.

dameMatrizEnFormato: Este generará y retornará un formato de la lista matriz, el cual será útil para mostrar en consola el estado de la matriz.

Se consideró también la definición del TDA para la "lista grupo" se considera:

NodoGrupo
| Fila
| Grupo
| Siguiente: Nodo
Fin NodoGrupo

La clase "**NodoGrupo**" contendrá todos los atributos que conforman a los grupos generados por los patrones, tanto el valor como sus índices dentro de la matriz, así como el puntero a un siguiente objeto de la misma clase, estos atributos únicamente tendrán los métodos *getters* y *setters*.

ListaGrupo
| inicio: NodoGrupo
| gruposTot
Fin ListaGrupo

La clase "**Lista grupo**" se definirá con un solo atributo de tipo NodoGrupo y un atributo que indicará la cantidad de grupos dentro de la lista, sus posibles operaciones son:

Inicializar: Colocar a null el apuntador inicio de la lista.

estaVacía: Retornará True si *inicio* es null o de lo contrario retornará False.

getGruposTot: Devolverá el contador de grupos totales en la lista.

setGruposTot: Recibirá un valor entero y lo asignará a la variable gruposTot.

Agregar:

Si está vacía(inicio/final): Se crea un nodo que se asigna al inicio y se autorreferencia a sí mismo.

Al Inicio(No vacía): Se crea un nuevo nodo, el puntero de este enlazará al inicio, con un while se recorrerá la lista hasta encontrar el nodo (último) que apunta al inicio, al encontrar el último nodo de la lista, el puntero de este cambiara hacia el nuevo nodo, y el nuevo nodo será asignado como el nuevo inicio.

Al Final(No vacía): Se crea un nuevo nodo, el puntero de este enlazará al inicio, con un while se recorrerá la lista hasta encontrar el nodo (último) que apunta al inicio, al encontrar el último nodo de la lista, el puntero de este cambiara hacia el nuevo nodo, *sin cambiar el inicio*.

getInicio: Retorna el nodo inicio.

buscaEnGrupo: Este método recibirá un parámetro (fila) y verificará si hay un registro con el mismo dato en la lista, al hallar una coincidencia retornará True o en su defecto False.

getInfo: Retorna un valor str con toda la información de la lista concatenada.

Para la definición del TDA para la "lista principal" se considera:

NodoPrincipal
| Nombre
| Filas
| Columnas
| Grupos
| Matriz: ListaEnlazadaCircular
| MatrizRedu: ListaEnlazadaCircular
| Siguiente: NodoPrincipal
Fin NodoPrincipal

La clase "**NodoPrincipal**" contendrá todos los atributos que conforman todos los datos pertenecientes a la matriz, nombre, cantidad de filas y columnas, los grupos que comparten un patron, la matriz original y la matriz reducida, así como el puntero a un siguiente objeto de la misma clase, estos atributos únicamente tendrán los métodos *getters* y *setters*.

Operación para uso de comprobaciones.

getInfo: Única operación especializada, que retorna un string, con todos los datos concatenados y las matrices con formato (con el uso del método especializado de las sublistas de matrices) para la visualización del estado completo de la matriz.

ListaPrincipal
| inicio: NodoPrincipal
| procesado:Boolean
Fin ListaPrincipal

La clase "**Lista principal**" (también de tipo lista circular) se definirá con un atributo de tipo NodoPrincipal y un booleano (procesado) para indicar si los nodos ya han sido analizadas y generado sus matrices reducidas, sus posibles operaciones son:

Inicializar: Colocar a null el apuntador inicio de la lista, y asigna False al atributo "procesado".

yaProcesado: Cambia el valor del atributo procesado a True.

estaProcesado: Retornara el valor booleano que posea el atributo "procesado".

estaVacía: Retornará True si *inicio* es null o de lo contrario retornará False.

Agregar:

Al Inicio(No vacía): Se crea un nuevo nodo, el puntero de este enlazará al inicio, con un while se recorrerá la lista hasta encontrar el nodo (último) que apunta al inicio, al encontrar el último nodo de la lista, el puntero de este cambiara hacia el nuevo nodo, y el nuevo nodo será asignado como el nuevo inicio.

Al Final(No vacía): Se crea un nuevo nodo, el puntero de este enlazará al inicio, con un while se recorrerá la lista hasta encontrar el nodo (último) que apunta al inicio, al encontrar el último nodo de la lista, el puntero de este cambiara hacia el nuevo nodo, *sin cambiar el inicio*.

Si está vacía(inicio/final): Se crea un nodo que se asigna al inicio y se autorreferencia a sí mismo.

existeNombre: Recorre la lista y evalúa si un nombre de entrada ya está registrado, si hay una coincidencia retornará True de lo contrario retorna False.

dameNombres: Retorna un listado de los nombres registrados en los nodos.

dameNodo: Retorna el nodo que coincida con un nombre recibido.

getInicio: Retorna el inicio.

muestraLista: Recorre la lista y muestra los datos en consola a través de la función *getInfo* de cada nodo.

MODULO – LectorXML.py

Librerias utilizadas:
- minidom de xml.dom: Permitirá la lectura de archivos xml
- time: Permitirá crear pausas para visualizar el proceso de lectura.
- *manejardor*: Módulo general de manejo, donde se aloja la lista global.
- *ListaSimpleCircular* de *Listas_Nodos*: Permitirá crear un TAD temporal.

Función – **leerxml(ruta, nArchivo):**

Recibe como parámetro la ruta y nombre de archivo, en el cual con uso de *minidom.parse(ruta)* el archivo es cargado al programa, y mediante *getElementsByTag*, se separan los datos según las etiquetas indicadas según los requerimientos de estructura del archivo xml (ver anexo).

En este método se valida el nombre de la nueva matriz, con la función auxiliar *validaNombre(nombre)* que obtiene la lista enlazada, ésta valida si la lista está vacía y si contiene nodos evalúa si el nombre ya pertenece a algún registro dentro de la lista, si hay coincidencias retorna True de lo contrario retorna False.

Obtiene los atributos de tamaño de la matriz y evalúa si los valores son numéricos, de lo contrario el sistema lanzará un mensaje de error y omitirá la lectura de "esa" matriz en la lista de matrices.

Se recorren la lista de etiquetas "*dato*" que contienen los valores y posiciones de las matrices, se obtienen y se evalúan si son numéricos y si los índices obtenidos no exceden a tamaño especificado al inicio de lectura, de lo contrario se indicarán los errores (*taBien = False*) y se omitirá el almacenamiento de dicha matriz.

Al finalizar el análisis de una matriz, si el booleano *taBien* es True, significa que la lectura ha sido completada sin errores; con los datos ya validados, se crea una matriz (sublista) temporal, y con el método *generaListaDeDimension(fila,col)* ***(ver definición de TDA lista circular)*** donde se recorrerá nuevamente los elementos de etiqueta "*dato*" para usar el método *reemplazaDatos(fil,col,valor)* ***(ver definición de TDA lista circular)*** sobre la lista matriz y luego de reemplazar cada dato, usar el método *evaluaLista()* ***(ver definición de TDA lista circular)***, para luego agregar en la lista global (TDA implementada).

MODULO – procesadorMatriz.py

Librerias utilizadas:
- time: Permitirá crear pausas para visualizar el proceso de lectura.
- *ListaSimpleCircular* de *Listas_Nodos*: Permitirá crear un TAD temporal.
- *ListaGrupo* de *Listas_Nodos*: Permitirá crear un TAD temporal.

El procesamiento de la matriz se divide en tres funciones o fases:
- Creación de la matriz patrón (con 0's y 1's)
- Análisis de patrones (Se generan los grupos)
- Reducción de la matriz (Se genera la matriz reducida con base a los grupos)

Función(1er-Fase) – **procesarMatriz(filas, columnas, matriz):**

Esta función genera un TDA lista simple circular temporal (lista patrón) que con ayuda de la función *generaListaDeDimension(filas,cols)* generará una lista de igual magnitud al de la matriz original, el algoritmo recorrerá la lista matriz original y evaluará los valores en cada nodo, sí este, es mayor que cero (0) en la lista patrón se reemplazará el valor con la función *reemplazaDatos(fil, col, valor)* ***(según corresponda a los registros del nodo analizado)***, el valor será 1; si el valor del nodo es igual a cero (0) el valor a reemplazar es 0.

De esta manera se tendrá la lista (matriz) con los patrones de la lista (matriz) original, una vez finalizado el "mapeo" de la matriz original, se evaluará la lista patrón con la función *analizaLista()* de ser válida retornará la siguiente fase con los parámetros requeridos.

Función(2da-Fase) – **analizaPatrones(filas, columnas, matriz, mapaPatron):**

Como su nombre lo indica este algoritmo evaluará los patrones de la matriz, creada en la *1ra Fase* del análisis, generando una lista que contendrá los grupos que poseen patrones idénticos.

Se creará un TDA *ListaGrupo*, el algoritmo comparará cada elemento de una fila con los elementos de las restantes, si éste es igual la variable booleana será True e indicará que la fila pertenece al grupo, al final de cada ciclo superior el contador de grupo aumentará; cabe mencionar que, el algoritmo está pensado para que en cada ciclo se disminuya la cantidad de comparaciones realizadas, también se evaluará en cada ciclo superior si la fila que se va a analizar ya está registrado en un grupo para indicar si se evaluará con las restantes o no, como última instancia se evaluará si la última fila (o índice de fila) de la matriz se encuentra registrado en la lista de grupo, de no estar registrada se almacenará como un nuevo grupo.

Esta función retornará a la última fase de análisis con los parámetros requeridos.

Función(3ra-Fase) – **reduceMatriz(filas, columnas, matriz, groupsTemps):**

Con base a los requerimientos de funcionalidad, las filas que conformen grupos deberán ser sumados y almacenados en una sola tupla (fila).

El algoritmo evalúa si la cantidad de grupos es igual a la cantidad de filas de la matriz original, si esta es igual se entiende que la matriz no posee reducción, de lo contrario se creará una nueva lista circular que se dimensionará (*generaListaDeDimension(fil,col)*) correspondiente al número de grupos (groupsTemps.getGruposTot()) que será el tamaño para las filas (fil) y las columnas obtenidas de los parámetros, ésta será nuestra lista reducida.

Se distribuye en dos ciclos for, el exterior que recorrerá las filas (0 hasta #Grupos) y el interior que recorrerá las columnas, se definirá una variable suma que como su nombre lo indica contendrá la suma de los valores para cada columna, se recorrerá la lista de grupos evaluando según el índice del ciclo exterior, si este es igual, se obtendrá el valor de la posición en la matriz (*matriz.dameDatos(fil, col)*) y se sumará a la variable suma, cuando la lista de grupos haya finalizado, en la lista reducida se reemplazaran los datos correspondientes a la fila (#grupo) y columna que se éste analizando.

Una vez finalizada la reducción y reemplazo de datos la función retornará por fin la lista temporal (lista reducida) y la lista groupsTemps (lista de grupos).

MODULO – generarXML.py

Librerias utilizadas:
- *ET* de *xml.etree.ElementTree*: Permitirá manejar un árbol para el manejo de la estructura de archivos de extensión XML.
- *minidom* de *xml.dom*: Permitirá manejar archivos de extensión XML.

Función – **generaResultado(name, cols, groups, matrizReduc):**

Permitirá generar la estructura xml (etiquetado) de los registros de las matrices, con los parámetros requeridos. Creará la raíz, con el parámetro matrizReduc (un TDA), se recorrerá la lista

obteniendo así los datos (fila, columna, valor) para ser agregado al esquema xml, se realizará un proceso similar con la lista de grupos, obteniendo el número de grupo y la suma de las repeticiones en los nodos (frecuencia).

Esta función retornará otra función para darle formato a la estructura generada para los registros.

MODULO – generarGrafo.py

Librerias utilizadas:
- *system* de *os*: Permitirá usar funciones del sistema.
- *path* de *os*: Permitirá trabajar funciones con rutas de archivo.

Función – **getsource():**

A través de la librería *path* este método retornará la ruta del script que se está ejecutando.

Función – **mimetodo(di):**

Esta función recibirá un parámetro de tipo string, que contendrá un comando a ejecutar, a través de la librería *os* con la función system.

Función – **grafo(nombreM, filas, columnas, matriz):**

Esta función permitirá la escritura de un archivo de extensión *.dot*, en donde se manejará la estructura del archivo; se recorrerá la lista matriz para obtener los atributos (fila, columna y valor).

Una vez obtenidos los nodos y enlaces se escribirán a través de un método, con el uso de la función *open(route),* escribirá el archivo *.dot.*

MODULO – manejador.py

Librerias utilizadas:
- *path* de *os*: Permitirá trabajar funciones con rutas de archivo.
- *sleep* de *time*: Permitirá crear pausas en tiempo de ejecución.
- *leerxml* de *LectorXML*: Permitirá realizar la lectura del archivo de carga.
- *ListaPrincipal* de *Listas_Nodos*: Permitirá crear un objeto de tipo lista principal.
- *procesaMatriz* de *procesadorMatriz*: Permitirá evaluar la matriz a través de la función de procesamiento.
- *generaResultado* de *generarXML*: Permitirá generar un archivo de salida de extensión xml, con los datos de la lista procesada.
- *grafo* de *generarGrafo*: Permitirá crear el grafo del registro de matriz.

Variable global (única): listaPrueba.

Función – **getLista():**
Retornará la lista global.

Función – **agregaEnLista(nombre, fil, col, matriz):**
Utiliza la función de la lista global para almacenar nuevos registros de matrices.

Función – **cargarArchivo():**
Al llamar a esta función se validará si la lista posee registros para ser limpiada y así almacenar los nuevos registros, se evaluará la ruta del archivo (si existe y si es de extensión xml) para luego llamar a la función *leerxml.*

Función – **procesarArchivo():**
Si la lista global no esta vacía, se procesarán los registros, llamando a la función *procesaMatriz*, se recorrerá la lista global, enviando sus parámetros solicitados por la función (*procesaMatriz)* y capturando los valores retornados (matriz reducida y lista de grupos), luego estos se almacenarán o actualizarán para su nodo perteneciente.

Función – **escribeSalida():**
Esta función solicitará al usuario una ruta para la escritura del archivo xml, con los resultados de las listas procesadas, también evaluará si la extensión ingresada es xml, se recorrerá la lista global iterando para cada matriz que luego llamará a la función *generaResultado()* el cual retornará la matriz con formato xml, que será capturado en una variable, para que con el método *write* (para función *open*), el archivo sea creado.

Función – **generaGrafoMatriz():**
Evalúa si la lista esta vacía, esta función despliega un menú con los nombres de las matrices registradas, el usuario deberá seleccionar una para generar su respectivo grafo, una vez el usuario haya seleccionado un **numero** valido, se llamará a la función *grafo*, el cual generará un grafo en la carpeta de los scripts en ejecución y lo abrirá automáticamente.

Función – **datosEstudiante():**
Mostrará la información del programador.

Conclusiones

El uso de TDA puede sonar un tanto complicado y aún mas si se le aplica la lógica de matrices, el aprovechamiento del manejo de memoria dinámica es tan vital para crear un buen programa, generar estructuras robustas y entendibles, para que en su momento le sea posible refactorizar el código, el esquematizar correctamente un TDA es de gran utilidad al momento del manejo de los datos o análisis para los registros dentro de lo que en este caso es una lista enlazada circular.

Referencias bibliográficas

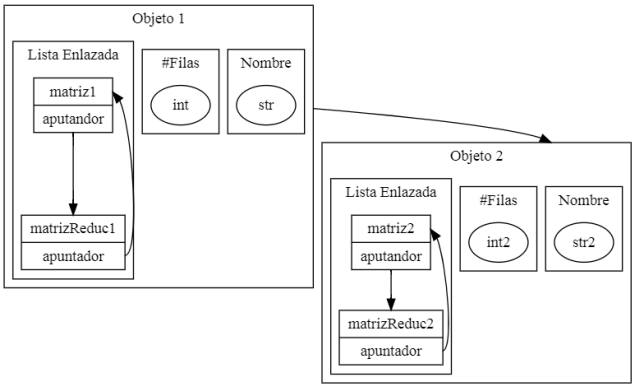
Documentation. (s. f.). graphviz.org. Recuperado 7 de marzo de 2021, de https://graphviz.org/documentation/

Robinson, S. (s. f.). Reading and Writing XML Files in Python. Stack Abuse. Recuperado 7 de marzo de 2021, de https://stackabuse.com/reading-and-writing-xml-files-in-python/

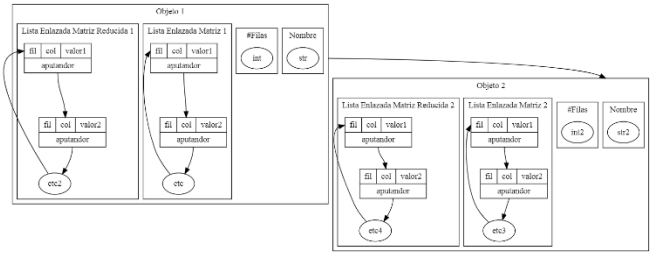
Anexos

Primeros modelos de estructura de almacenamiento.

Modelo-1

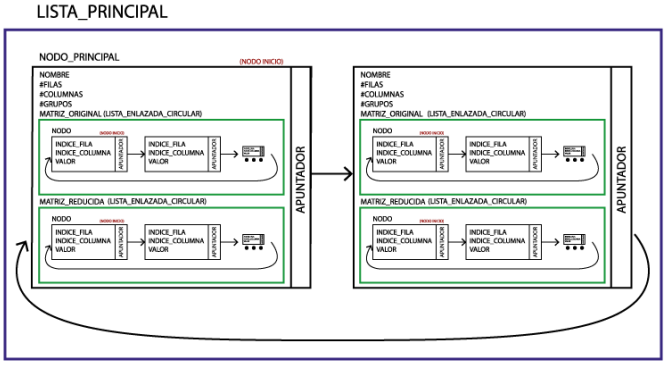


Modelo-2



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 2021.

Modelo-3 (Modelo-2 formalizado)



Modelo-4 (Modelo Final)

Implementación completa de memoria dinámica.