Estructura de Datos y Base de Datos Relacionales

Balcon Edwart, Cardenas Silva, Espeche Exequiel, Tycona Alex, Velasquez Bryam

14 de octubre de 2020

Resumen

Este articulo pretende dar a conocer un panorama general de la importancia de la aplicación de estructura de datos buscando detallar de manera comprensiva y la utilización de forma correcta de las base de datos relacionales aplicando en estas ultimas las muchas variedad de operaciones que se definirán en el articulo

I. Introduccion

Las tecnologías de la información obligan a modificar la organización de la educación, porque crean entornos educativos que amplían considerablemente las posibilidades del sistema, no sólo de tipo organizativo, sino también de transmisión de conocimientos y desarrollo de destrezas, habilidades y actitudes. La clave está en transformar la información en conocimiento y éste, en educación y aprendizaje significativo

II. Objetivos

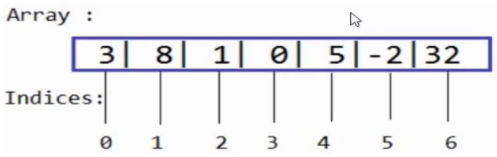
- Entender qué son las Estructura de Datos.
- Entender que son las Base de Datos Relacionales.
- Entender las Ventajas y Desventajas de las Estructura de Datos.
- Entender las Ventajas y Desventajas de los Datos Relacionales.
- Dar ejemplos entendibles de Estructuras de Datos.

III. Desarrollo

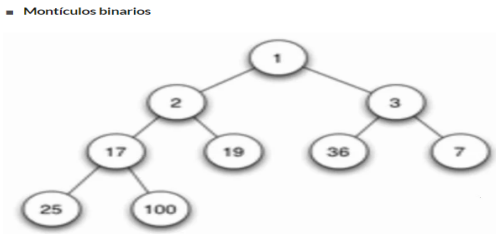
1. ¿Qué son las Estructura de Datos?

Las estructuras de datos son una forma de organizar los datos en la computadora, de tal manera que nos permita realizar operaciones de forma mas eficiente.Depende que algoritmo queramos ejecutar, habrá veces que sea mejor utilizar una estructura de datos u otra estructura que nos permita más velocidad.

- Array: Constan de un índice para acceder a una posición concreta y del valor que el mismo almacena.



- Montículos binarios: Es una forma de guardar los datos de tal manera que aunque no estén ordenados, se pueda retirar de ese conjunto, datos de forma ordenada.

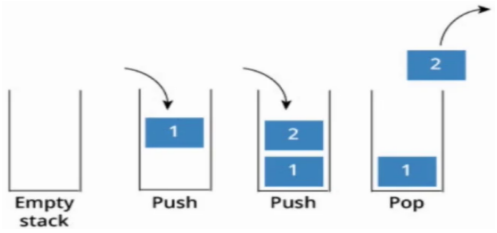


- Pilas: Es una forma de guardar los datos de tal manera que aunque no estén ordenados, se pueda retirar de ese conjunto, datos de forma ordenada.

  Para el manejo de los datos se cuenta con dos operaciones básicas:

- Apilar (push), que coloca un objeto en la pila.
- Retirar (o desapilar, pop), que retira el último elemento apilado.

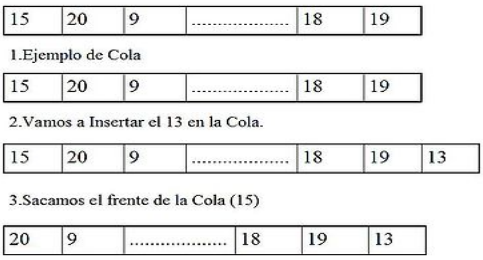
En cada momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado. La operación retirar permite la obtención de este elemento, que es retirado de la pila permitiendo el acceso al siguiente.



Las pilas suelen emplearse en los siguientes contextos:

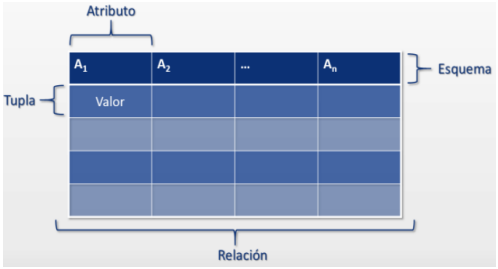
- Evaluación de expresiones en notación postfija (notación polaca inversa).
- Reconocedores sintácticos de lenguajes independientes del contexto.
- Implementación de recursividad.

- Colas: Es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción push se realiza por un extremo y la operación de extracción pop por el otro. También se le llama estructura FIFO (del inglés First In First Out), debido a que el primer elemento en entrar será también el primero en salir.



II. ¿Que son las Base de Datos Relacionales?

Es una relación que representa un conjunto de entidades con las mismas propiedades. Cada relación se compone de una serie de filas o registros (las llamadas tuplas), cuyos valores dependen de ciertos atributos (columnas).
La tabla es el formato que utiliza el modelo relacional para explicar de un modo visual la ordenación de los valores de una tupla en función de los atributos definidos en la relación. Una base de datos relacional no es otra cosa, entonces, que un conjunto de tablas interrelacionadas.



Todo gestor de bases de datos relacionales soporta al menos un lenguaje formal que permite ejecutar las siguientes operaciones:

- Definir la estructura de datos: en la definición de los datos se guarda una descripción con metadatos de la estructura de datos en el diccionario del sistema. Cuando un usuario crea una tabla nueva, en el diccionario de datos se almacena su correspondiente esquema. El vocabulario de un len-

guaje de bases de datos que se utiliza para definir los datos se denomina Data Definition Language (DDL), lenguaje de definición de datos.

- Definir derechos: todos los lenguajes de bases de datos proporcionan una sintaxis que permite otorgar o retirar permisos. En este contexto se habla de Data Control Language (DCL) o lenguaje de control de datos, un vocabulario integrado en el lenguaje de bases de datos.

- Definir condiciones de integridad: por condiciones de integridad se entienden los requisitos de estado que se exigen a un banco de datos. Si se definen condiciones para su integridad, la BD garantiza que se cumplan en todo momento. Se habla entonces de un estado consistente. Una condición básica de integridad en una base de datos relacional es, por ejemplo, que cada registro (tupla) pueda identificarse de forma inequívoca.

- Definir transacciones: Es cuando se lleva a una BD de un estado consistente a otro diferente. Estas transacciones contienen una serie de instrucciones que deben ejecutarse siempre de forma íntegra. Si una se interrumpe, la BD vuelve a su estado original (Rollback). Cada transacción comienza con una orden para crear una conexión con la BD a la que siguen otras que inician las operaciones de datos en sí, así como un paso de comprobación (Commit) que asegura la integridad de la BD. Las operaciones que pongan en peligro la integridad de la tabla no se consignan (committed), es decir, no se escriben en la base de datos de forma permanente. Por último, se cierra la conexión con la BD. Al vocabulario del lenguaje de bases de datos con el que se manipulan los datos se le conoce como Data Manipulation Language (DML).

- Definir vistas: las llamadas views son vistas virtuales de un subconjunto de los datos de una tabla. Para crear una vista, el SGBD genera una tabla virtual (relación lógica) sobre la base de las tablas físicas. En estas vistas pueden emplearse las mismas operaciones que se utilizarían en tablas físicas. Según la función de la vista de datos pueden distinguirse distintos tipos de vista. Las más habituales son aquellas que filtran determinadas filas (consulta de selección) o columnas (vista de columnas) de una tabla, así como las que conectan diversas tablas entre sí (vista de conjunto).

III. Entender las Ventajas y Desventajas de las Estructura de Datos

Ventajas

- Permite modificar globalmente las variables sin tener que recorrer el código buscando cada aparición.
- Define variables y evita que cambien entre rutinas.
- Separa desde el inicio del programa el espacio en memoria.

IV. Entender las Ventajas y Desventajas de los Datos Relacionales

Ventajas

- No se tiene la necesidad de utilizar el papel como fuente de almacenamiento primario.
- La actualización de la información en una base de datos es mucha más rápida.
- Disposición de información precisa y actualizada en momentos justos.
- Reducción de información repetida durante el ingreso de datos.
- Esta arquitectura lleva más de 50 años en el mercado, por lo que, hoy

en día, existen gran cantidad de productos diseñados específicamente para gestionarla.
- Permite tener la información muy bien organizada y estructurada.
- La atomicidad es una de sus principales propiedades. Esta se refiere a la condición de que cada operación se realiza en su totalidad o se revierte, lo cual evita que las operaciones queden a medias.

Desventajas

- La manera de cómo se relacionan las diferentes entidades de la base de datos es más compleja de entender.
- Los datos abstractos o no estructurados como los del big data no son admitidos.
- El mantenimiento es muy costoso y complicado cuando la base de datos crece a un gran tamaño.
- Los tiempos de respuesta suelen ralentizarse a medida que la base de datos crece.

- La manera de cómo se relacionan las diferentes entidades de la base de datos es más compleja de entender.
- Los datos abstractos o no estructurados como los del big data no son admitidos.
- El mantenimiento es muy costoso y complicado cuando la base de datos crece a un gran tamaño.
- Los tiempos de respuesta suelen ralentizarse a medida que la base de datos crece.
- El inconveniente de esta implementación es que es necesario fijar de ante mano el número máximo de elementos que se puede contener la pila,y por lo tanto al apilar un elemento es necesario controlar que no se inserte un elemento si la pila está llena.

V. Ejemplos entendibles de Estructura de Datos

Estructura de Datos Pila

```
#include<iostream>
#include<stdlib.h>
using namespace std;

int pila[3] , cima=-1; //declarando mi pila

//funciones
void meterdatos(){
 if(cima<2) {
  cout<<"introduzca el valor a meter a la pila:"<<endl<<"-->";
  cin>>pila[cima+1];
  cima++;
    cout<<"el valor se ha introducido correctamente"<<endl<<endl;
     system("PAUSE");
 }
 else{
    cout<<"Imposible meter datos,la pila esta llena"<<endl<<endl;
    system("PAUSE");
 }
}
void sacardatos(){
    if(cima>-1){//Si la pila esta vacia
        cima--;
        cout<<"Se ha eliminado un valor de la pila"<<endl<<"el valor eliminado es"<<pila[cima+1]<<endl<<endl;
        system("PAUSE");
    }
    else{
        cout<<"imposible sacar datos,la pila esta vacia"<<endl<<endl;
        system("PAUSE");
    }
}
void pilavacia(){
    if(cima==-1){
        cout<<"la pila esta vacia"<<endl<<endl;
        system("PAUSE");
    }
    else{
        cout<<"la pila NO esta vacia"<<endl<<endl;
        system("PAUSE");
    }

}
void mostrarpila(){
    if(cima>-1){ //[1,2,3,END]
        cout<<"\t[ ";
        for(int a=cima; a>=-1;a--){
            cout<<pila[a]<<",";

        }

    }
    else{
        cout<<"la pila esta vacia"<<endl<<endl;
        system("PAUSE");
    }
}

 //main
int main(){
    int opcion;
 do{
 system("cls");
 cout<<"Bienvenida al ejemplo de una pila"<<endl<<endl
 <<"MENU"<<endl<<endl
 <<"[1] Meter datos"<<endl
 <<"[2] Sacar datos"<<endl
 <<"[3] ¿pila vacia?"<<endl
 <<"[4] Mostrar pila"<<endl
 <<"[5] Salir"<<endl<<endl<<"->";
 int opcion; cin>>opcion;

   switch(opcion){
    case 1:
        meterdatos();
        break;
    case 2:
        sacardatos();
        break;
    case 3:
        pilavacia();
        break;
    case 4:
        mostrarpila();
        break;
    case 5:
        break;
    default:
     cout<<"opcion no valida"<<endl<<endl;
     system("PAUSE");
        }

    }while(opcion!=5 );
}
```

Estructura de Datos Lista

```
#include <iostream>
#include <stdlib.h>
using namespace std;

struct nodo{
       int nro;        // en este caso es un numero entero
       struct nodo *sgte;
};

typedef struct nodo *Tlista;

void insertarInicio(Tlista &lista, int valor)
{
    Tlista q;
    q = new(struct nodo);
    q->nro = valor;
    q->sgte = lista;
    lista  = q;
}
void insertarFinal(Tlista &lista, int valor)
{
    Tlista t, q = new(struct nodo);

    q->nro  = valor;
    q->sgte = NULL;

    if(lista==NULL)
    {
        lista = q;
    }
    else
    {
        t = lista;
        while(t->sgte!=NULL)
        {
            t = t->sgte;
        }
        t->sgte = q;
    }

}
int insertarAntesDespues()
{
    int _op, band;
    cout<<endl;
    cout<<"\t 1. Antes de la posicion           "<<endl;
    cout<<"\t 2. Despues de la posicion         "<<endl;

    cout<<"\n\t Opcion : "; cin>> _op;

    if(_op==1)
        band = -1;
    else
        band = 0;
    return band;
}
void insertarElemento(Tlista &lista, int valor, int pos)
{
    Tlista q, t;
    int i;
    q = new(struct nodo);
    q->nro = valor;

    if(pos==1)
    {
        q->sgte = lista;
        lista = q;
    }
    else
    {
        int x = insertarAntesDespues();
        t = lista;

        for(i=1; t!=NULL; i++)
        {
            if(i==pos+x)
            {
                q->sgte = t->sgte;
                t->sgte = q;
                return;
            }
            t = t->sgte;
        }
    }
    cout<<"   Error...Posicion no encontrada..!"<<endl;
}
```

```
}
void buscarElemento(Tlista lista, int valor)
{
    Tlista q = lista;
    int i = 1, band = 0;
    while(q!=NULL)
    {
        if(q->nro==valor)
        {
            cout<<endl<<" Encontrada en posicion "<< i <<endl;
            band = 1;
        }
        q = q->sgte;
        i++;
    }
    if(band==0)
        cout<<"\n\n Numero no encontrado..!"<< endl;
}
void reportarLista(Tlista lista)
{
    int i = 0;
    while(lista != NULL)
    {
        cout <<' '<< i+1 <<") " << lista->nro << endl;
        lista = lista->sgte;
        i++;
    }
}


void eliminarElemento(Tlista &lista, int valor)
{
    Tlista p, ant;
    p = lista;

    if(lista!=NULL)
    {
        while(p!=NULL)
        {
            if(p->nro==valor)
            {
                if(p==lista)
                    lista = lista->sgte;
                else
                    ant->sgte = p->sgte;

                delete(p);
                return;
            }
            ant = p;
            p = p->sgte;
        }
    }
    else
        cout<<" Lista vacia..!";
}
```

```
void eliminaRepetidos(Tlista &lista, int valor)
{
    Tlista q, ant;
    q = lista;
    ant = lista;

    while(q!=NULL)
    {
        if(q->nro==valor)
        {
            if(q==lista) // primero elemento
            {
                lista = lista->sgte;
                delete(q);
                q = lista;
            }
            else
            {
                ant->sgte = q->sgte;
                delete(q);
                q = ant->sgte;
            }
        }
        else
        {
            ant = q;
            q = q->sgte;
        }

    }// fin del while

    cout<<"\n\n Valores eliminados..!"<<endl;
}

void menu1()
{
    cout<<"\n\t\tLISTA ENLAZADA SIMPLE\n\n";
    cout<<" 1. INSERTAR AL INICIO                "<<endl;
    cout<<" 2. INSERTAR AL FINAL                 "<<endl;
    cout<<" 3. INSERTAR EN UNA POSICION          "<<endl;
    cout<<" 4. REPORTAR LISTA                    "<<endl;
    cout<<" 5. BUSCAR ELEMENTO                   "<<endl;
    cout<<" 6. ELIMINAR ELEMENTO 'V'             "<<endl;
    cout<<" 7. ELIMINAR ELEMENTOS CON VALOR 'V'  "<<endl;
    cout<<" 8. SALIR                             "<<endl;

    cout<<"\n INGRESE OPCION: ";
}
```

```
/*                          Funcion Principal
-------------------------------------------------------------------*/

int main()
{
    Tlista lista = NULL;
    int op;     // opcion del menu
    int _dato;  // elemenento a ingresar
    int pos;    // posicion a insertar

    system("color 0b");

    do
    {
        menu1();  cin>> op;

        switch(op)
        {
            case 1:

                cout<< "\n NUMERO A INSERTAR: "; cin>> _dato;
                insertarInicio(lista, _dato);
            break;


            case 2:

                cout<< "\n NUMERO A INSERTAR: "; cin>> _dato;
                insertarFinal(lista, _dato );
            break;


            case 3:

                cout<< "\n NUMERO A INSERTAR: ";cin>> _dato;
                cout<< " Posicion : ";      cin>> pos;
                insertarElemento(lista, _dato, pos);
            break;


            case 4:

                cout << "\n\n MOSTRANDO LISTA\n\n";
                reportarLista(lista);
            break;


            case 5:

                cout<<"\n Valor a buscar: "; cin>> _dato;
                buscarElemento(lista, _dato);
            break;

            case 6:
```

Estructura de Datos Colas

```
#include <iostream>
using namespace std;
/* ------------------- Estructura de los nodos de la cola  ----------------*/
struct nodo
{
    int nro;
    struct nodo *sgte;
};
/* ------------------------- Estructura de la cola -------------------*/
struct cola
{
    nodo *delante;
    nodo *atras  ;
};
/* ---------------------- Encolar elemento-------------------------------*/
void encolar( struct cola &q, int valor )
{
    struct nodo *aux = new(struct nodo);

    aux->nro = valor;
    aux->sgte = NULL;

    if( q.delante == NULL)
        q.delante = aux;   // encola el primero elemento
    else
        (q.atras)->sgte = aux;

    q.atras = aux;        // puntero que siempre apunta al ultimo elemento
}
/* ------------------------ Desencolar elemento-------------------------*/
int desencolar( struct cola &q )
{
    int num ;
    struct nodo *aux ;

    aux = q.delante;      // aux apunta al inicio de la cola
    num = aux->nro;
    q.delante = (q.delante)->sgte;
    delete(aux);          // libera memoria a donde apuntaba aux

    return num;
}
/* -------------------------Mostrar Cola-------------------------------*/
void muestraCola( struct cola q )
{
    struct nodo *aux;

    aux = q.delante;

    while( aux != NULL )
    {
        cout<<"   "<< aux->nro ;
        aux = aux->sgte;
    }
}
```

```
/*-------------------Eliminar todos los elementos de la Cola -------------*/
void vaciaCola( struct cola &q)
{
    struct nodo *aux;
    while( q.delante != NULL)
    {
        aux = q.delante;
        q.delante = aux->sgte;
        delete(aux);
    }
    q.delante = NULL;
    q.atras   = NULL;
}
/*--------------------Menu de opciones -------------------------------*/
void menu()
{
    cout<<"\n\t IMPLEMENTACION DE COLAS EN C++\n\n";
    cout<<" 1. ENCOLAR                          "<<endl;
    cout<<" 2. DESENCOLAR                       "<<endl;
    cout<<" 3. MOSTRAR COLA                     "<<endl;
    cout<<" 4. VACIAR COLA                      "<<endl;
    cout<<" 5. SALIR                            "<<endl;
    cout<<"\n INGRESE OPCION: ";
}
/* ---------------------Funcion Principal-------------------------*/
int main()
{   struct cola q;
    q.delante = NULL;
    q.atras   = NULL;
    int dato;  // numero a encolar
    int op;    // opcion del menu
    int x ;    // numero que devuelve la funcon pop
    system("color 0b");
    do
    {
        menu();  cin>> op;
        switch(op)
        {   case 1:
                cout<< "\n NUMERO A ENCOLAR: "; cin>> dato;
                encolar( q, dato );
                cout<<"\n\n\t\tNumero " << dato << " encolado...\n\n";
            break;
            case 2:
                x = desencolar( q );
                cout<<"\n\n\t\tNumero "<< x <<" desencolado...\n\n";
            break;
            case 3:
                cout << "\n\n MOSTRANDO COLA\n\n";
                if(q.delante!=NULL) muestraCola( q );
                else   cout<<"\n\n\tCola vacia...!"<<endl;
            break;
            case 4:

                vaciaCola( q );
                cout<<"\n\n\t\tHecho...\n\n";
            break;
            default;
        }
        cout<<endl<<endl;
        system("pause");  system("cls");
    }while(op!=5);
    return 0;
}
```

Estructura de Datos Arbol

```java
package javaapplication1;
import javaapplication1.Nodo;
/**
 *
 * @author Alex
 */
public class Pruebaarbol {
  public static void main(String[] args) {

    //Se asignan los valores a los nodos = 1,2,3,4,5 y 6 dentro del arbol
    //Nodo raiz = 1
    Nodo raiz = new Nodo(1);

    //Nodo2 Izquierdo = 2
    Nodo nodo2 = new Nodo(2);

    //Nodo3 Derecho = 3
    Nodo nodo3 = new Nodo(3);


    //Se asigna el valor 6 al nodo que sera hijo del nodo 3 a la derecha
    nodo3.setNodoDerecho(new Nodo(6));

    //Se asigna el valor 5 al nodo que sera hijo del nodo 3 a la izquierda
    nodo3.setNodoIzquierdo(new Nodo(5));

    //Se asigna el valor 4 al nodo que sera hijo del nodo 4 a la izquierda
    nodo2.setNodoIzquierdo(new Nodo(4));

    //Se crean el Nodo 2 a la izquierda y el Nodo 3 a la derecha de la raiz
    raiz.setNodoIzquierdo(nodo2);
    raiz.setNodoDerecho(nodo3);

    //Resultado en pantalla
    System.out.println("Recorrido Preorden: ");
    preOrden(raiz);
    System.out.println("Recorrido Inorden: ");
    inorden(raiz);
    System.out.println("Recorrido PostOrden: ");
    posOrden(raiz);
  }
```

```java
package javaapplication1;

public class Nodo {
  //Variables
  private int dato;
  private Nodo izq;
  private Nodo der;

  //Constructor
  public Nodo(int dato){
    this.dato = dato;
  }
  //nodo izquierdo
  public Nodo getNodoIzquierdo(){
    return izq;
  }
  //nodo derecho
  public Nodo getNodoDerecho(){
    return der;
  }
  //crear nodo derecho
  public void setNodoDerecho(Nodo nodo){
    der = nodo;
  }
  //crear nodo derecho
  public void setNodoIzquierdo(Nodo nodo){
    izq = nodo;
  }

  public int getDato(){
    return dato;
  }
}
```

```
Output - JavaApplication1 (run)
run:
Recorrido Preorden:
1 - 2 - 4 - 3 - 5 - 6 - Recorrido Inorden:
4 - 2 - 1 - 5 - 3 - 6 - Recorrido PostOrden:
4 - 2 - 5 - 6 - 3 - 1 - BUILD SUCCESSFUL (total time: 0 seconds)
```

IV. Recomendaciones

Se recomienda aprender practicando ya que la experiencia es la única forma hábil de detectar donde te pueden ayudar las Estructura de Datos y en Bases De Datos Relacionales

V. Bibliográfia

- Telles Brunelli Lazzarin, Ivo Barbi (2012, 19 agosto).Analisis de Bases De Datos Relacionales.

  Recuperado: https://dspace.ups.edu.ec/bitstream/123456789/69 CT003639.pdf

- Fernando Quevedo (2011).Construcción de una base de datos.

  Recuperado: https://blog.enzymeadvisinggroup.com/ejemplos-de-base-de-datos

- Guglielmo Trentin (1992).Estructura y organización de una base de datos

  Recuperado: https://prezi.com/4vycdtsiigbo/estructura-de-datos/

- Vera Koester (2015).Estructura de datos

  Recuperado: http://www.calcifer.org/documentos/librognome/g lists-queues.html

- Vera Koester (2019).Técnica de extracción de información de bases de datos

  Recuperado: https://www.upo.es/revistas/index.php/RevMetC

- Juan Esteban Díaz Montejo (2015).Técnica de extracción de información de bases de datos

Recuperado: https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/bases-de-datos-relacionales/

- Mervat Chouman (2013).Tecnicas de Monticulos

  Recuperado: https://www.infor.uva.es/ cva-ca/asigs/monticulos.pdf

- Susan Goldman (2007).Importancia de las Estructuras de Datos

  Recuperado: https://openwebinars.net/blog/que-son-las-estructuras-de-datos-y-por-que-son-tan-utiles/