UNIVERSIDAD NACIONAL DEL ALTIPLANO

Facultad de Ingeniería Estadística e Informática

Estructuras de Datos

Autor: Condori Calapuja Andre

Docente: Ing. Torres Cruz Fred

2025
Puno - Perú

31 de mayo de 2025

Índice

1. Introducción a Estructuras de Datos

1.1. Conceptos

Las estructuras de datos son modelos fundamentales para organizar y almacenar información en la memoria de una computadora. Proporcionan mecanismos eficientes para manipular datos, permitiendo operaciones como inserción, búsqueda, eliminación y ordenamiento de manera óptima.

- **Por organización:**
  - **Lineales:** Elementos dispuestos secuencialmente (arrays, listas, pilas, colas)
  - **No lineales:** Elementos con relaciones jerárquicas o de red (árboles, grafos)
- **Por tamaño:**
  - **Estáticas:** Tamaño fijo definido en tiempo de compilación (arrays tradicionales)
  - **Dinámicas:** Tamaño variable que puede crecer o reducirse en tiempo de ejecución (listas enlazadas, árboles)

1.2. Ejemplo en C++

```cpp
#include <iostream>
#include <vector>  // Estructura din mica
#include <array>   // Estructura est tica

int main() {
    // Array est tico (tama o fijo)
    std::array<int, 5> arrEstatico = {1, 2, 3, 4, 5};

    // Vector din mico (tama o variable)
    std::vector<int> vecDinamico = {1, 2, 3};
    vecDinamico.push_back(4);  // A adir elemento

    std::cout << "Array est tico: ";
    for(int num : arrEstatico) {
        std::cout << num << " ";
    }

    std::cout << "\nVector din mico: ";
    for(int num : vecDinamico) {
        std::cout << num << " ";
    }

    return 0;
```

```
}
```

Listing 1: Ejemplo de estructuras en C++

2. Arrays o Arreglos

2.1. Conceptos

Los arrays son estructuras de datos lineales y estáticas que almacenan elementos del mismo tipo en posiciones contiguas de memoria. Cada elemento se accede mediante un índice numérico, generalmente comenzando desde 0.

- **Ventajas:**
  - Acceso aleatorio rápido (O(1))
  - Eficiencia en memoria (solo almacena datos)
- **Desventajas:**
  - Tamaño fijo
  - Inserción/eliminación costosas (O(n))

2.2. Ejemplo en C++

```cpp
#include <iostream>
using namespace std;

void ordenamientoBurbuja(int arr[], int tama o) {
    for(int i = 0; i < tama o -1; i++) {
        for(int j = 0; j < tama o -i-1; j++) {
            if(arr[j] > arr[j+1]) {
                swap(arr[j], arr[j+1]);
            }
        }
    }
}

int main() {
    const int TAMA O = 6;
    int numeros[TAMA O] = {30, 10, 50, 20, 40, 60};

    cout << "Array original: ";
    for(int i = 0; i < TAMA O; i++) {
        cout << numeros[i] << " ";
    }

    ordenamientoBurbuja(numeros, TAMA O);
```

```
24
25      cout << "\nArray ordenado: ";
26      for(int i = 0; i < TAMA O; i++) {
27          cout << numeros[i] << " ";
28      }
29
30      // B squeda binaria
31      int objetivo = 40;
32      int izquierda = 0, derecha = TAMA O - 1;
33      while(izquierda <= derecha) {
34          int medio = izquierda + (derecha - izquierda) / 2;
35          if(numeros[medio] == objetivo) {
36              cout << "\nElemento " << objetivo << "
                    encontrado en posici n " << medio;
37              break;
38          }
39          if(numeros[medio] < objetivo) {
40              izquierda = medio + 1;
41          } else {
42              derecha = medio - 1;
43          }
44      }
45
46      return 0;
47  }
```

Listing 2: Operaciones con arrays en C++

3. Registros (Structs)

3.1. Conceptos

Los registros (structs en C++) son estructuras de datos compuestas que permiten agrupar variables de diferentes tipos bajo un mismo nombre. Cada variable dentro del struct se denomina miembro o campo.

- **Características:**
  - Agrupación lógica de datos relacionados
  - Acceso a miembros mediante operador punto (.) o flecha (-¿)
  - Pueden contener otros structs

3.2. Ejemplo en C++

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
```

```cpp

struct Direccion {
    string calle;
    int numero;
    string ciudad;
};

struct Estudiante {
    string nombre;
    int edad;
    float promedio;
    Direccion dir;
};

void mostrarEstudiante(const Estudiante& est) {
    cout << "Nombre: " << est.nombre << "\n"
         << "Edad: " << est.edad << "\n"
         << "Promedio: " << est.promedio << "\n"
         << "Direcci n: " << est.dir.calle << " "
         << est.dir.numero << ", " << est.dir.ciudad <<
             "\n";
}

int main() {
    Estudiante estudiante1 = {
        "Ana Garc a",
        20,
        8.5,
        {"Calle Principal", 123, "Ciudad Ejemplo"}
    };

    Estudiante grupo[3] = {
        {"Juan P rez", 21, 7.8, {"Calle 1", 45, "Ciudad
            A"}},
        {"Luisa M ndez", 19, 9.1, {"Calle 2", 67, "Ciudad
            B"}},
        estudiante1
    };

    grupo[0].promedio = 8.0;

    for(int i = 0; i < 3; i++) {
        cout << "\nEstudiante " << i+1 << ":\n";
        mostrarEstudiante(grupo[i]);
    }

    return 0;
}
```

Listing 3: Uso de structs en C++

4. Listas Enlazadas

4.1. Conceptos

Una lista enlazada es una estructura de datos lineal y dinámica compuesta por nodos, donde cada nodo contiene datos y un puntero al siguiente nodo en la secuencia.

- **Ventajas:**
  - Tamaño dinámico
  - Inserción/eliminación eficientes
- **Desventajas:**
  - Acceso secuencial (O(n))
  - Mayor uso de memoria por punteros

4.2. Ejemplo en C++

```cpp
#include <iostream>
using namespace std;

class Nodo {
public:
    int dato;
    Nodo* siguiente;

    Nodo(int valor) : dato(valor), siguiente(nullptr) {}
};

class ListaEnlazada {
private:
    Nodo* cabeza;
    Nodo* cola;

public:
    ListaEnlazada() : cabeza(nullptr), cola(nullptr) {}

    ~ListaEnlazada() {
        while(cabeza != nullptr) {
            Nodo* temp = cabeza;
            cabeza = cabeza->siguiente;
            delete temp;
        }
    }

    void insertarFinal(int valor) {
        Nodo* nuevo = new Nodo(valor);
```

```
30          if(cola == nullptr) {
31              cabeza = cola = nuevo;
32          } else {
33              cola->siguiente = nuevo;
34              cola = nuevo;
35          }
36      }
37
38      void eliminar(int valor) {
39          Nodo* actual = cabeza;
40          Nodo* previo = nullptr;
41
42          while(actual != nullptr && actual->dato != valor) {
43              previo = actual;
44              actual = actual->siguiente;
45          }
46
47          if(actual == nullptr) return;
48
49          if(previo == nullptr) {
50              cabeza = actual->siguiente;
51              if(cabeza == nullptr) cola = nullptr;
52          } else {
53              previo->siguiente = actual->siguiente;
54              if(actual == cola) cola = previo;
55          }
56
57          delete actual;
58      }
59
60      void mostrar() const {
61          Nodo* actual = cabeza;
62          while(actual != nullptr) {
63              cout << actual->dato;
64              if(actual->siguiente != nullptr) cout << " -> ";
65              actual = actual->siguiente;
66          }
67          cout << " -> NULL\n";
68      }
69  };
70
71  int main() {
72      ListaEnlazada lista;
73
74      lista.insertarFinal(10);
75      lista.insertarFinal(20);
76      lista.insertarFinal(30);
77      lista.insertarFinal(40);
78
79      cout << "Lista original: ";
```

```
    lista.mostrar();

    lista.eliminar(20);
    cout << "Despu s de eliminar 20: ";
    lista.mostrar();

    lista.insertarFinal(50);
    cout << "Despu s de insertar 50: ";
    lista.mostrar();

    return 0;
}
```

Listing 4: Implementación de lista enlazada en C++

5. Listas Dobles y Circulares

5.1. Conceptos

- **Listas doblemente enlazadas:** Cada nodo tiene punteros al anterior y siguiente.
- **Listas circulares:** El último nodo apunta al primero.
- **Ventajas:**
  - Recorrido bidireccional (dobles)
  - Acceso circular sin verificar final

5.2. Ejemplo en C++

```
#include <iostream>
using namespace std;

class NodoDoble {
public:
    int dato;
    NodoDoble* anterior;
    NodoDoble* siguiente;

    NodoDoble(int valor) : dato(valor), anterior(nullptr),
        siguiente(nullptr) {}
};

class ListaDobleCircular {
private:
    NodoDoble* cabeza;

```

```
17 public:
18     ListaDobleCircular() : cabeza(nullptr) {}
19
20     ~ListaDobleCircular() {
21         if(cabeza == nullptr) return;
22
23         NodoDoble* actual = cabeza->siguiente;
24         while(actual != cabeza) {
25             NodoDoble* temp = actual;
26             actual = actual->siguiente;
27             delete temp;
28         }
29         delete cabeza;
30     }
31
32     void insertarFinal(int valor) {
33         NodoDoble* nuevo = new NodoDoble(valor);
34
35         if(cabeza == nullptr) {
36             cabeza = nuevo;
37             cabeza->anterior = cabeza;
38             cabeza->siguiente = cabeza;
39         } else {
40             NodoDoble* ultimo = cabeza->anterior;
41
42             nuevo->siguiente = cabeza;
43             nuevo->anterior = ultimo;
44             ultimo->siguiente = nuevo;
45             cabeza->anterior = nuevo;
46         }
47     }
48
49     void mostrarAdelante() const {
50         if(cabeza == nullptr) {
51             cout << "Lista vac a\n";
52             return;
53         }
54
55         NodoDoble* actual = cabeza;
56         do {
57             cout << actual->dato;
58             actual = actual->siguiente;
59             if(actual != cabeza) cout << " <-> ";
60         } while(actual != cabeza);
61         cout << " (Circular)\n";
62     }
63 };
64
65 int main() {
66     ListaDobleCircular lista;
```

```
67
68      lista.insertarFinal(10);
69      lista.insertarFinal(20);
70      lista.insertarFinal(30);
71      lista.insertarFinal(40);
72
73      cout << "Recorrido hacia adelante: ";
74      lista.mostrarAdelante();
75
76      return 0;
77  }
```

Listing 5: Lista doble circular en C++

6. Pilas (Stacks)

6.1. Conceptos

Estructura LIFO (Last In, First Out) donde los elementos se insertan y eliminan por el mismo extremo (tope).

- **Operaciones:**
  - Push: Insertar en el tope
  - Pop: Eliminar del tope
  - Peek/Top: Ver el tope sin eliminar
- **Aplicaciones:**
  - Evaluación de expresiones
  - Gestión de llamadas a funciones
  - Algoritmos de backtracking

6.2. Ejemplo en C++

```
1  #include <iostream>
2  #include <stack>
3  #include <string>
4  using namespace std;
5
6  bool verificarParentesis(const string& expresion) {
7      stack<char> pila;
8
9      for(char c : expresion) {
10         switch(c) {
11             case '(': case '[': case '{':
12                 pila.push(c);
```

```
13                  break;
14
15              case ')':
16                  if(pila.empty() || pila.top() != '(')
                        return false;
17                  pila.pop();
18                  break;
19
20              case ']':
21                  if(pila.empty() || pila.top() != '[')
                        return false;
22                  pila.pop();
23                  break;
24
25              case '}':
26                  if(pila.empty() || pila.top() != '{')
                        return false;
27                  pila.pop();
28                  break;
29          }
30      }
31
32      return pila.empty();
33  }
34
35  int main() {
36      stack<int> pila;
37
38      pila.push(10);
39      pila.push(20);
40      pila.push(30);
41
42      cout << "Tope de la pila: " << pila.top() << endl;
43      pila.pop();
44      cout << "Tope despu s de pop: " << pila.top() << endl;
45
46      string expresion1 = "{(a + b) * [c - d]}";
47      string expresion2 = "{(a + b) * [c - d}}";
48
49      cout << "\nExpresi n 1: " << expresion1 << " - ";
50      cout << (verificarParentesis(expresion1) ? "Balanceado"
            : "No balanceado") << endl;
51
52      cout << "Expresi n 2: " << expresion2 << " - ";
53      cout << (verificarParentesis(expresion2) ? "Balanceado"
            : "No balanceado") << endl;
54
55      return 0;
56  }
```

Listing 6: Implementación de pila en C++

7. Colas (Queues)

7.1. Conceptos

Estructura FIFO (First In, First Out) donde los elementos se insertan por un extremo (final) y se eliminan por el otro (frente).

- **Variantes:**
  - Cola circular
  - Cola doble (deque)
  - Cola de prioridad
- **Aplicaciones:**
  - Gestión de procesos
  - Buffers de impresión
  - Simulación de líneas de espera

7.2. Ejemplo en C++

```cpp
#include <iostream>
#include <queue>
#include <deque>
using namespace std;

void simulacionAtencionClientes() {
    queue<string> colaClientes;

    colaClientes.push("Cliente 1");
    colaClientes.push("Cliente 2");
    colaClientes.push("Cliente 3");

    cout << "Proceso de atenci n:\n";
    while(!colaClientes.empty()) {
        string clienteActual = colaClientes.front();
        colaClientes.pop();

        cout << "Atendiendo a " << clienteActual << endl;
    }
    cout << "Todos los clientes atendidos\n";
}

```

```cpp
int main() {
    queue<int> cola;

    cola.push(10);
    cola.push(20);
    cola.push(30);

    cout << "Frente de la cola: " << cola.front() << endl;
    cola.pop();
    cout << "Nuevo frente despu s de dequeue: " <<
        cola.front() << endl;

    deque<int> colaDoble;
    colaDoble.push_back(10);
    colaDoble.push_front(5);
    colaDoble.push_back(20);

    cout << "\nCola doble: ";
    for(int num : colaDoble) {
        cout << num << " ";
    }
    cout << endl;

    cout << "\nSimulaci n de atenci n:\n";
    simulacionAtencionClientes();

    return 0;
}
```

Listing 7: Implementación de cola en C++

8. Recursión

8.1. Conceptos

Técnica donde una función se llama a sí misma para resolver problemas que pueden dividirse en subproblemas más pequeños.

- **Componentes:**
  - Caso base: Condición de terminación
  - Caso recursivo: Llamada con subproblema más pequeño
- **Ventajas:**
  - Código más claro para ciertos problemas
  - Natural para algoritmos divide y vencerás
- **Desventajas:**

- Posible desbordamiento de pila
- Ineficiencia en algunos casos

8.2. Ejemplo en C++

```cpp
#include <iostream>
#include <vector>
using namespace std;

unsigned long long factorial(int n) {
    if(n <= 1) return 1;
    return n * factorial(n - 1);
}

int busquedaBinaria(const vector<int>& arr, int objetivo,
    int izquierda, int derecha) {
    if(izquierda > derecha) return -1;

    int medio = izquierda + (derecha - izquierda) / 2;

    if(arr[medio] == objetivo) {
        return medio;
    } else if(arr[medio] < objetivo) {
        return busquedaBinaria(arr, objetivo, medio + 1,
            derecha);
    } else {
        return busquedaBinaria(arr, objetivo, izquierda,
            medio - 1);
    }
}

void recorridoPreorden(int nodo, int nivel) {
    if(nodo > 15) return;

    cout << string(nivel*2, ' ') << nodo << "\n";
    recorridoPreorden(2*nodo, nivel+1);
    recorridoPreorden(2*nodo+1, nivel+1);
}

int main() {
    int num = 5;
    cout << "Factorial de " << num << " es " <<
        factorial(num) << "\n\n";

    vector<int> arr = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int objetivo = 23;
    int resultado = busquedaBinaria(arr, objetivo, 0,
        arr.size() - 1);

```

```
40      cout << "B squeda binaria: " << objetivo << " est  en
            ndice  " << resultado << "\n\n";
41
42      cout << "Recorrido preorden de  rbol  binario:\n";
43      recorridoPreorden(1, 0);
44
45      return 0;
46  }
```

Listing 8: Ejemplos de recursión en C++