Algoritmo Depth First Search (DFS) en C++

El algoritmo Depth First Search (DFS) es un algoritmo fundamental para recorrer o buscar en estructuras de datos como grafos o árboles. DFS explora tan lejos como sea posible a lo largo de cada rama antes de retroceder.

Conceptos Clave

- **Pila o Recursión**: DFS puede implementarse utilizando una pila (enfoque iterativo) o mediante recursión (enfoque recursivo).
- **Visitar Nodos**: Los nodos se marcan como visitados para evitar procesarlos más de una vez.

Implementación Básica

La implementación básica de DFS utiliza una lista de adyacencia para representar el grafo y un vector para marcar los nodos visitados.

Código de la Implementación Básica

```cpp
#include <iostream>
#include <vector>

void dfs(int v, std::vector<bool>& visited, const std::vector<std::vector<int>>& adj)
{
    visited[v] = true;
    std::cout << v << " ";

    for (int u : adj[v]) {
        if (!visited[u]) {
            dfs(u, visited, adj);
        }
    }
}

int main() {
    int n = 5;
    std::vector<std::vector<int>> adj(n);
    adj[0] = {1, 2};
    adj[1] = {0, 3, 4};
    adj[2] = {0};
    adj[3] = {1};
    adj[4] = {1};

    std::vector<bool> visited(n, false);
    std::cout << "DFS a partir del nodo 0: ";
    dfs(0, visited, adj);

    return 0;
}
```

Salida esperada: `DFS a partir del nodo 0: 0 1 3 4 2`

Aplicaciones del Algoritmo DFS

1. Detección de Ciclos en un Grafo Dirigido

DFS puede ser utilizado para detectar ciclos en un grafo dirigido. Si durante la búsqueda DFS encontramos un nodo que ya está en el camino actual, entonces existe un ciclo.

```
#include <iostream>
#include <vector>

bool dfs_cycle(int v, std::vector<bool>& visited, std::vector<bool>& recStack, const
std::vector<std::vector<int>>& adj) {
    visited[v] = true;
    recStack[v] = true;

    for (int u : adj[v]) {
        if (!visited[u] && dfs_cycle(u, visited, recStack, adj)) {
            return true;
        } else if (recStack[u]) {
            return true;
        }
    }

    recStack[v] = false;
    return false;
}

bool hasCycle(int n, const std::vector<std::vector<int>>& adj) {
    std::vector<bool> visited(n, false);
    std::vector<bool> recStack(n, false);

    for (int i = 0; i < n; ++i) {
        if (!visited[i] && dfs_cycle(i, visited, recStack, adj)) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<std::vector<int>> adj = {
        {1}, {2}, {0}, {4}, {5}, {3}
    };

    if (hasCycle(6, adj)) {
        std::cout << "El grafo tiene un ciclo." << std::endl;
    } else {
        std::cout << "El grafo no tiene un ciclo." << std::endl;
    }
```

```
    return 0;
}
```

Salida esperada: `El grafo tiene un ciclo.`

2. Componente Fuertemente Conectado (SCC)

DFS se puede usar para encontrar componentes fuertemente conectados (SCC) en un grafo dirigido. Un SCC es un subgrafo en el cual cada nodo es alcanzable desde cualquier otro nodo dentro del subgrafo.

```
#include <iostream>
#include <vector>
#include <stack>

void dfs_scc(int v, std::vector<bool>& visited, const std::vector<std::vector<int>>&
adj, std::stack<int>& Stack) {
    visited[v] = true;

    for (int u : adj[v]) {
        if (!visited[u]) {
            dfs_scc(u, visited, adj, Stack);
        }
    }

    Stack.push(v);
}

void reverse_dfs(int v, std::vector<bool>& visited, const
std::vector<std::vector<int>>& adj) {
    visited[v] = true;
    std::cout << v << " ";

    for (int u : adj[v]) {
        if (!visited[u]) {
            reverse_dfs(u, visited, adj);
        }
    }
}

void findSCCs(int n, const std::vector<std::vector<int>>& adj) {
    std::stack<int> Stack;
    std::vector<bool> visited(n, false);

    for (int i = 0; i < n; ++i) {
        if (!visited[i]) {
            dfs_scc(i, visited, adj, Stack);
        }
    }

    std::vector<std::vector<int>> adjT(n);
    for (int v = 0; v < n; ++v) {
        for (int u : adj[v]) {
```

```
            adjT[u].push_back(v);
        }
    }

    std::fill(visited.begin(), visited.end(), false);
    while (!Stack.empty()) {
        int v = Stack.top();
        Stack.pop();

        if (!visited[v]) {
            reverse_dfs(v, visited, adjT);
            std::cout << std::endl;
        }
    }
}

int main() {
    std::vector<std::vector<int>> adj = {
        {1}, {2}, {0, 3}, {4}, {5}, {3}
    };

    std::cout << "Componentes Fuertemente Conectados:" << std::endl;
    findSCCs(6, adj);

    return 0;
}
```

Salida esperada:

```
Componentes Fuertemente Conectados:
0 2 1
3 5 4
```

3. Búsqueda de Caminos en un Laberinto

DFS es ideal para resolver laberintos o para encontrar todos los caminos posibles entre dos puntos en un grafo.

```
#include <iostream>
#include <vector>

bool dfs_maze(int x, int y, std::vector<std::vector<int>>& maze,
std::vector<std::vector<bool>>& visited) {
    if (x < 0 || y < 0 || x >= maze.size() || y >= maze[0].size() || maze[x][y] == 1
|| visited[x][y]) {
        return false;
    }

    if (x == maze.size() - 1 && y == maze[0].size() - 1) {
        std::cout << "Camino encontrado!" << std::endl;
        return true;
    }
```

```
    visited[x][y] = true;

    if (dfs_maze(x + 1, y, maze, visited) || dfs_maze(x, y + 1, maze, visited) ||
        dfs_maze(x - 1, y, maze, visited) || dfs_maze(x, y - 1, maze, visited)) {
        return true;
    }

    visited[x][y] = false;
    return false;
}

int main() {
    std::vector<std::vector<int>> maze = {
        {0, 1, 0, 0},
        {0, 0, 0, 1},
        {1, 0, 1, 0},
        {0, 0, 0, 0}
    };

    std::vector<std::vector<bool>> visited(maze.size(), std::vector<bool>
(maze[0].size(), false));

    if (!dfs_maze(0, 0, maze, visited)) {
        std::cout << "No se encontró un camino." << std::endl;
    }

    return 0;
}
```

Salida esperada: `Camino encontrado!`