Instituto Politécnico Nacional
Escuela Superior de Cómputo
Ingeniería en
Sistemas Computacionales

Unidad de Aprendizaje:

Inteligencia Artificial

Grupo: 6CV2

Práctica: DFS y BFS

Alumno:

Barros Martínez Luis Enrique
Bautista Ríos Alfredo
Garcia Martinez Angel Yahel

Profesor: Marco Antonio Castillo

Introducción

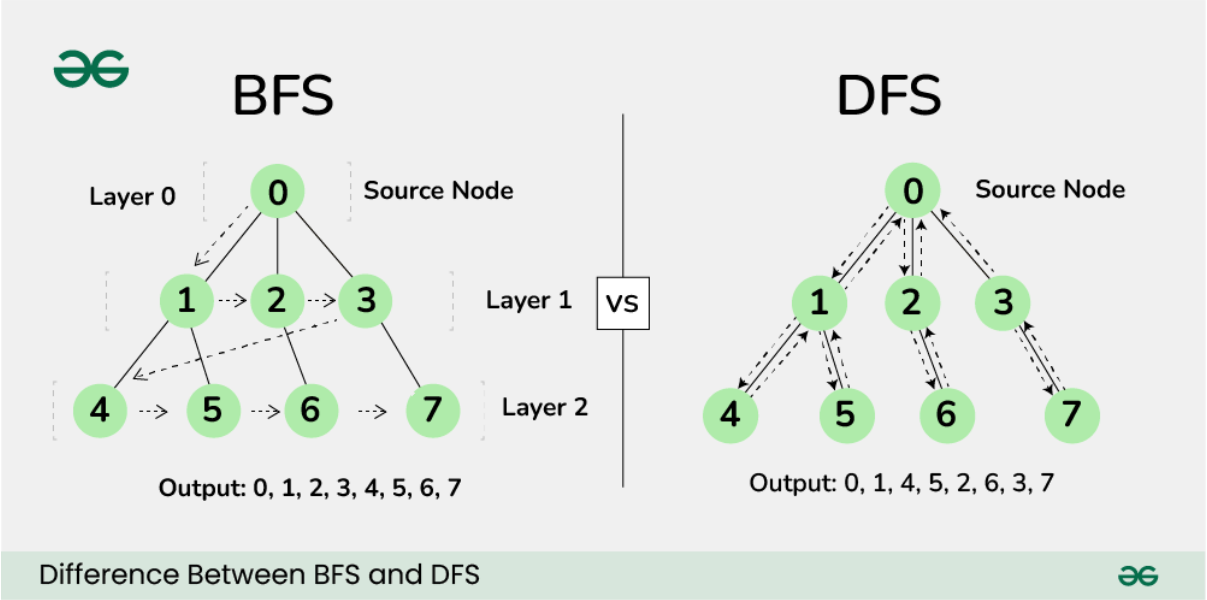
La presente práctica aborda nuestro acercamiento a los algoritmos de búsqueda no informados (DFS y BFS), estos consisten en la selección e implementación de estrategias de búsqueda de un estado solución a partir de un estado inicial sin introducir al algoritmo de solución conocimiento sobre el impacto de las transiciones en la exploración del espacio de estados.

Tras ser implementados dichos algoritmos se obtendrá su análisis respecto al tiempo que les toma llegar a el dato en búsqueda con respecto a los casos ingresados.

Desarrollo
Se realizará la programación del algoritmo de Búsqueda en anchura (BFS) y Búsqueda en profundidad (DFS) en esta ocasión en lenguaje C++

BFS
La búsqueda en anchura provee un algoritmo simple (basado en el recorrido de árboles por niveles) para encontrar un estado de solución desde un estado inicial. Al implementarse mediante el recorrido por niveles, se requiere una estructura cola (FIFO, first-in, first-out).
En inteligencia artificial, el algoritmo Breadth-First Search (BFS) es una herramienta esencial para explorar y navegar por diversos espacios de problemas. Al recorrer sistemáticamente estructuras de grafos o árboles, BFS resuelve tareas como la búsqueda de rutas, el enrutamiento de redes y la resolución de acertijos. Este artículo analiza los conceptos básicos de BFS, sus algoritmos y aplicaciones prácticas en IA.

DFS
Una búsqueda en profundidad (DFS) es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo. Su funcionamiento consiste en ir expandiendo cada uno de los nodos que va localizando, de forma recurrente (desde el nodo padre hacia el nodo hijo). Cuando ya no quedan más nodos que visitar en dicho camino, regresa al nodo predecesor, de modo que repite el mismo proceso con cada uno de los vecinos del nodo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda.
La búsqueda en profundidad surge como un algoritmo crucial en inteligencia artificial que ofrece tanto conocimientos teóricos como aplicaciones prácticas. Los DFS son una herramienta fundamental para atravesar espacios de estados complejos. A medida que la tecnología evoluciona, DFS sigue siendo un algoritmo fundamental para impulsar avances y ampliar los límites de la investigación y las aplicaciones de la IA.



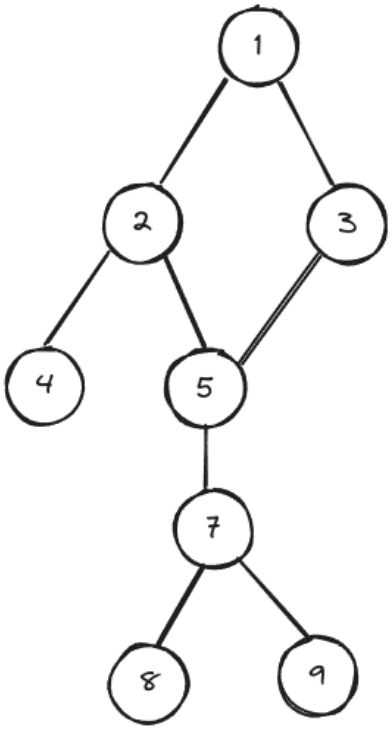
Código:

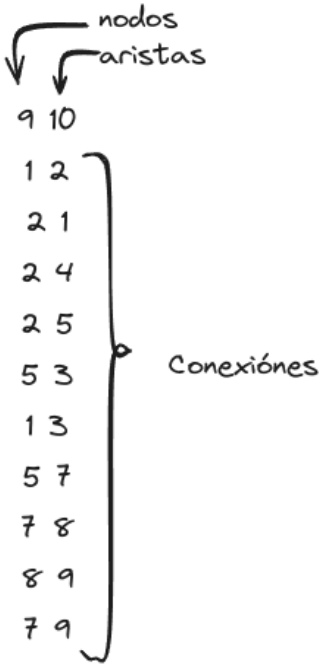
El código mostrado a continuación es capaz de recibir cualquier grafo luego de indicarle (número de nodos, número de aristas, y las conexiones), se genera un número aleatorio dentro del rango del número de nodos, el cual será el nodo a encontrar realizando búsquedas con los algoritmos DFS y BFS, primero imprimirá en pantalla el número aleatorio a buscar, indicará su raíz, y el recorrido que haga según el algoritmo, concluyendo con que se mostrará al usuario el tiempo y memoria empleado en cada búsqueda. Nota: Se detendrá la búsqueda en el momento en el que se encuentre el nodo.

```cpp
#include <bits/stdc++.h>
#include <chrono>
#include <sys/resource.h>  // Para medir el uso de memoria.
using namespace std::chrono;
using namespace std;

const int N = 1e5;   // Tamaño máximo permitido para el número de nodos.
vector<int>* g;      // Puntero a un array de vectores que representará el grafo.
int vis[N];          // Array para llevar un registro de los nodos visitados.
bool encontrado;     // Variable de control para saber si el nodo fue encontrado.

// Función para obtener el uso de memoria en kilobytes.
long getMemoryUsage() {
   struct rusage usage;
   getrusage(RUSAGE_SELF, &usage);
   return usage.ru_maxrss;  // Retorna el máximo tamaño de memoria residente.
}

void dfs(int u, int x) {
   if (encontrado) return;  // Si el nodo ya fue encontrado, detén la búsqueda.

   cout << u << " ";  // Imprime el nodo actual.
   if (u == x) {
       cout << "Me encontraste" << "\n";  // Si se encuentra el nodo, imprime un mensaje.
       encontrado = true;  // Marca como encontrado.
       return;  // Detiene la búsqueda.
   } else {
       cout << "\n";
   }
   vis[u] = 1;  // Marca el nodo como visitado.
   for (auto it : g[u])  // Recorre los vecinos del nodo actual.
       if (!vis[it])  // Si un vecino no ha sido visitado, llama recursivamente a DFS.
           dfs(it, x);
}

void bfs(int u, int x) {
   queue<int> q;
   q.push(u);
   vis[u] = 1;
   while (!q.empty()) {
       int v = q.front(); q.pop();
       cout << v << " ";  // Imprime el nodo actual.
       if (v == x) {
           cout << "Me encontraste " << "\n";  // Si encuentra el nodo, imprime el mensaje.
           return;  // Detén la búsqueda.
       } else {
```

```
            cout << "\n";
        }
        for (auto it : g[v]) {  // Recorre los vecinos del nodo actual.
            if (!vis[it]) {
                q.push(it);
                vis[it] = 1;  // Marca los vecinos como visitados y los añade a la cola.
            }
        }
    }
}

int main() {
    int n, m;
    cin >> n >> m;  // Lee el número de nodos y aristas del grafo.
    g = new vector<int>[n + 1];  // Inicializa la lista de adyacencia.

    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;  // Lee las aristas.
        g[a].push_back(b);  // Agrega la conexión en ambas direcciones (grafo no dirigido).
        g[b].push_back(a);
    }

    srand(time(0));  // Inicializa el generador de números aleatorios.
    int numeroRaiz = 1;  // Establece 1 como raíz.
    int numeroAleatorioEncontrar = rand() % n + 1;  // Elige un nodo aleatorio a buscar.
    cout << "Raiz: "<<numeroRaiz <<" Buscar: " <<numeroAleatorioEncontrar << endl;

    // DFS
    cout << "DFS" << "\n";
    for (int i = 0; i <= n; i++) vis[i] = 0;  // Reinicia el array de nodos visitados.
    encontrado = false;  // Resetea la variable de control.
    auto start = high_resolution_clock::now();  // Captura el tiempo inicial.
    long memoriaInicialDFS = getMemoryUsage();  // Obtén la memoria inicial.
    dfs(numeroRaiz, numeroAleatorioEncontrar);  // Llama a la DFS.
    auto end = high_resolution_clock::now();  // Captura el tiempo final.
    long memoriaFinalDFS = getMemoryUsage();  // Obtén la memoria final.
    auto duration = duration_cast<microseconds>(end - start);  // Calcula la duración.
    cout <<"en "<< duration.count() << "ms";  // Imprime el tiempo de ejecución.
     cout << (memoriaFinalDFS - memoriaInicialDFS) * 1024 << " B\n";   // Imprime la memoria
utilizada en bytes.

    // BFS
    cout << "BFS" << "\n";
    for (int i = 0; i <= n; i++) vis[i] = 0;  // Reinicia el array de nodos visitados.
    start = high_resolution_clock::now();  // Captura el tiempo inicial.
    long memoriaInicialBFS = getMemoryUsage();  // Obtén la memoria inicial.
    bfs(numeroRaiz, numeroAleatorioEncontrar);  // Llama a la BFS.
    end = high_resolution_clock::now();  // Captura el tiempo final.
    long memoriaFinalBFS = getMemoryUsage();  // Obtén la memoria final.
    duration = duration_cast<microseconds>(end - start);  // Calcula la duración.
    cout <<"en "<< duration.count() << "ms y ";  // Imprime el tiempo de ejecución.
     cout << (memoriaFinalBFS - memoriaInicialBFS) * 1024 << " B\n";   // Imprime la memoria
utilizada en bytes.

    return 0;  // Indica que el programa terminó correctamente.
}
```

Ejemplo Caso 1:





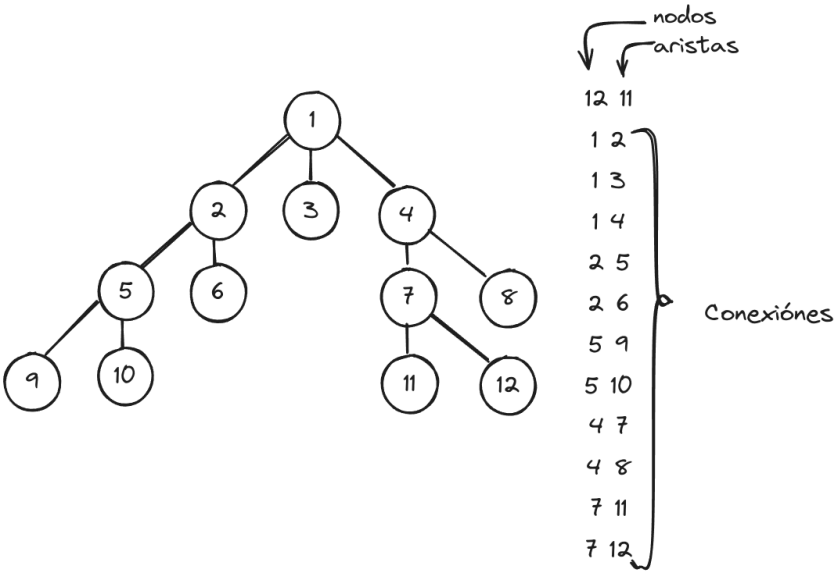

```
alfredo@archlinux ~/Documents/ESCOM-ISC/semester 25-1/06 - Inteligencia_Artificial/Practicas main ± ./P1
9 10
1 2
2 1
2 4
2 5
5 3
1 3
5 7
7 8
8 9
7 9
Raiz: 1 Buscar: 8
DFS
1
2
4
5
3
7
8 Me encontraste
en 49ms0 B
BFS
1
2
3
4
5
7
8 Me encontraste
en 46ms y 0 B
```

```
 alfredo@archlinux  ~/Documents/ESCOM-ISC/semester 25-1/06 - Inteligencia_Artificial/Practi
cas  main ±  ./P1
9 10
1 2
2 1
2 4
2 5
5 3
1 3
5 7
7 8
8 9
7 9
Raiz: 1 Buscar: 7
DFS
1
2
4
5
3
7 Me encontraste
en 51ms0 B
BFS
1
2
3
4
5
7 Me encontraste
en 45ms y 0 B
```

```
 alfredo@archlinux  ~/Documents/ESCOM-ISC/semester 25-1/06 - Inteligencia_Artificial/Practi
cas  main ±  ./P1
9 10
1 2
2 1
2 4
2 5
5 3
1 3
5 7
7 8
8 9
7 9
Raiz: 1 Buscar: 4
DFS
1
2
4 Me encontraste
en 22ms0 B
BFS
1
2
3
4 Me encontraste
en 27ms y 0 B
```

```
 alfredo@archlinux  ~/Documents/ESCOM-ISC/semester 25-1/06 - Inteligencia_Artificial/Practi
cas  main ±  ./P1
9 10
1 2
2 1
2 4
2 5
5 3
1 3
5 7
7 8
8 9
7 9
Raiz: 1 Buscar: 1
DFS
1 Me encontraste
en 14ms0 B
BFS
1 Me encontraste
en 9ms y 0 B
```

Ejemplo Caso 2:




```
alfredo@archlinux ~/Documents/ESCOM-ISC/semester 25-1/06 - Inteligencia_Artificial/Practi
cas  main ± ./P1
12 11
1 2
1 3
1 4
2 5
2 6
5 9
5 10
4 7
4 8
7 11
7 12
Raiz: 1 Buscar: 11
DFS
1
2
5
9
10
6
3
4
7
11 Me encontraste
en 55ms0 B
BFS
1
2
3
4
5
6
7
8
9
10
11 Me encontraste
en 52ms y 0 B
```

```
 alfredo@archlinux  ~/Documents/ESCOM-ISC/semester 25-1/06 - Inteligencia_Artificial/Practi
cas  main ±  ./P1
12 11
1 2
1 3
1 4
2 5
2 6
5 9
5 10
4 7
4 8
7 11
7 12
Raiz: 1 Buscar: 12
DFS
1
2
5
9
10
6
3
4
7
11
12 Me encontraste
en 66ms0 B
BFS
1
2
3
4
5
6
7
8
9
10
11
12 Me encontraste
en 54ms y 0 B
```

```
 alfredo@archlinux  ~/Documents/ESCOM-ISC/semester 25-1/06 - Inteligencia_Artificial/Practi
cas  main ±  ./P1
12 11
1 2
1 3
1 4
2 5
2 6
5 9
5 10
4 7
4 8
7 11
7 12
Raiz: 1 Buscar: 1
DFS
1 Me encontraste
en 15ms0 B
BFS
1 Me encontraste
en 10ms y 0 B
```

```
 alfredo@archlinux  ~/Documents/ESCOM-ISC/semester 25-1/06 - Inteligencia_Artificial/Practi
cas  main ±  ./P1
12 11
1 2
1 3
1 4
2 5
2 6
5 9
5 10
4 7
4 8
7 11
7 12
Raiz: 1 Buscar: 8
DFS
1
2
5
9
10
6
3
4
7
11
12
8 Me encontraste
en 74ms0 B
BFS
1
2
3
4
5
6
7
8 Me encontraste
en 46ms y 0 B
```

Conclusión

Luego de las comparativas se ha podido observar, que el uso de tiempo y memoria usado en grafos con por lo menos una decena de nodos, arrojan una cantidad mínima de recursos usados, con esa cantidad es posible notar que la cantidad de milisegundos usados varía en proporción a en qué momento se detiene la búsqueda, el recorrido aproximadamente entre 5 y 10 ms por nodo, una tiempo muy bueno, sin embargo, de momento, no lo clasificamos como los mejores algoritmos de búsqueda