Informe N° 6: DFS

Gianfranco Pupiales

2024-06-08

Tabla de Contenidos

1. Objetivos

- Distinguir los conceptos fundamentales del algoritmo de búsqueda en profundidad (DFS).
- Implementar el algoritmo de búsqueda en profundidad (DFS) utilizando Python.

2. Introducción

El algoritmo de búsqueda en profundidad (DFS) es un enfoque empleado para explorar grafos y árboles. Este método inicia en un nodo raíz y profundiza lo máximo posible a lo largo de cada rama antes de retroceder, facilitando así el recorrido completo de los nodos en un grafo. Su implementación puede ser recursiva o iterativa utilizando una pila. DFS es valioso para detectar ciclos en grafos, identificar componentes fuertemente conexos y resolver problemas que requieren la exploración exhaustiva de todas las posibles rutas o combinaciones, como puzzles o laberintos. No obstante, en grafos cíclicos, DFS puede quedar atrapado en bucles infinitos si no se lleva un registro de los nodos visitados («Depth First Search or DFS for a Graph - GeeksforGeeks — geeksforgeeks.org»).

3. Desarrollo

3.1 Enunciado ejercicio 1

Genere el árbol DFS (DFS Tree) para el grafo G, sabiendo que el nodo de inicio es Start. Ese sería el componente conectado que contiene al nodo Start. Implemente el algoritmo DFS (recursivo o con uso de pila) y use las estructuras auxiliares necesarias.

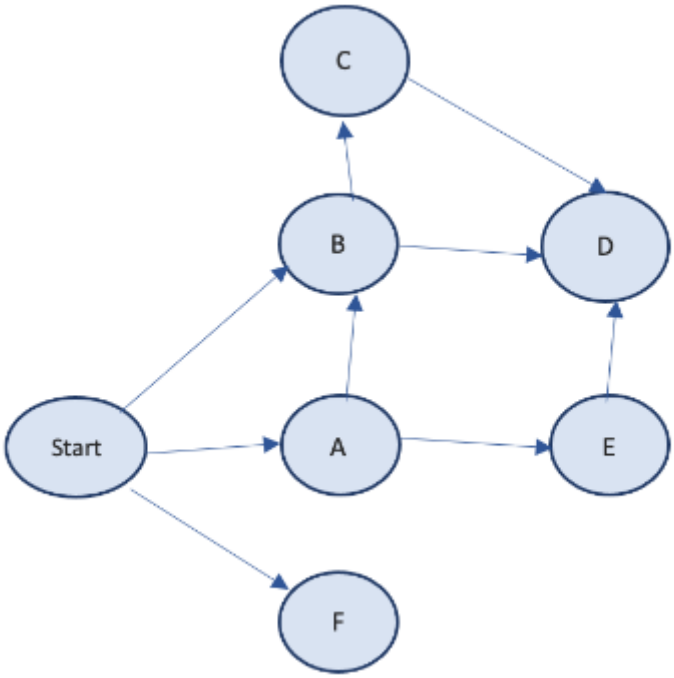

Figura 1: Grafo G

```
def dfs(nodo, grafo, searched, componentR, dfs_tree, parent=None):
     # Ir agregando los nodos que pertenecen al componente conectado
    componentR.append(nodo)
    searched[nodo] = True  # marcar al nodo como visitado

    if parent is not None:
        # Agregar la relación padre-hijo en el árbol DFS
        dfs_tree[parent].append(nodo)

    print(f"Nodo actual: {nodo}, Padre: {parent}")
    print(searched)
```

```
    print('Vecinos de', nodo, ':', grafo[nodo])

    # Recorrer los nodos adyacentes del nodo dado
    for vecino in grafo[nodo]:
        # Verificar que el nodo adyacente no está marcado como searched
        if not searched[vecino]:
            dfs(vecino, grafo, searched, componentR, dfs_tree, nodo)
            print('Finaliza', vecino)
            print('Vuelve a', nodo)
            print()
```

```
grafo_G = {
    "Start": ["B", "A", "F"],
    "B": ["C", "D"],
    "A": ["B", "E"],
    "F": [],
    "C": ["D"],
    "D": [],
    "E": ["D"]
}

nodoS = "Start"  # nodo start
# Al iniciar ningún nodo del grafo ha sido visitado
searched = {nodo: False for nodo in grafo_G}
# Lista para almacenar los nodos que pertenecen al componente conectado
componentR = []
# Diccionario para representar el árbol DFS
dfs_tree = {nodo: [] for nodo in grafo_G}

dfs(nodoS, grafo_G, searched, componentR, dfs_tree)
# Imprimir los nodos que pertenecen al componente conectado
print(f"La búsqueda DFS es: {componentR}")
print(f"El Árbol DFS es: {dfs_tree}")  # Imprimir el árbol DFS
```

```
Nodo actual: Start, Padre: None
{'Start': True, 'B': False, 'A': False, 'F': False, 'C': False, 'D': False, 'E': False}
Vecinos de Start : ['B', 'A', 'F']
Nodo actual: B, Padre: Start
{'Start': True, 'B': True, 'A': False, 'F': False, 'C': False, 'D': False, 'E': False}
Vecinos de B : ['C', 'D']
Nodo actual: C, Padre: B
```

```
{'Start': True, 'B': True, 'A': False, 'F': False, 'C': True, 'D': False, 'E': False}
Vecinos de C : ['D']
Nodo actual: D, Padre: C
{'Start': True, 'B': True, 'A': False, 'F': False, 'C': True, 'D': True, 'E': False}
Vecinos de D : []
Finaliza D
Vuelve a C

Finaliza C
Vuelve a B

Finaliza B
Vuelve a Start

Nodo actual: A, Padre: Start
{'Start': True, 'B': True, 'A': True, 'F': False, 'C': True, 'D': True, 'E': False}
Vecinos de A : ['B', 'E']
Nodo actual: E, Padre: A
{'Start': True, 'B': True, 'A': True, 'F': False, 'C': True, 'D': True, 'E': True}
Vecinos de E : ['D']
Finaliza E
Vuelve a A

Finaliza A
Vuelve a Start

Nodo actual: F, Padre: Start
{'Start': True, 'B': True, 'A': True, 'F': True, 'C': True, 'D': True, 'E': True}
Vecinos de F : []
Finaliza F
Vuelve a Start

La búsqueda DFS es: ['Start', 'B', 'C', 'D', 'A', 'E', 'F']
El Árbol DFS es: {'Start': ['B', 'A', 'F'], 'B': ['C'], 'A': ['E'], 'F': [], 'C': ['D'], 'D'
```

3.2 Enunciado ejercicio 2

Puede probar su algoritmo con el ejemplo visto en clase para probar el recorrido correcto de las ejecuciones DFS

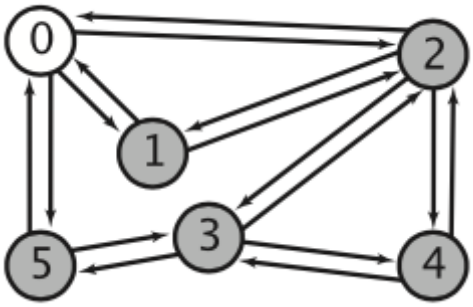

Figura 2: Grafo clase

```
graph = {
    0: [2, 1, 5],
    1: [0, 2],
    2: [0, 1, 3, 4],
    3: [5, 4, 2],
    4: [3, 2],
    5: [3, 0]
}

nodoS = 0  # nodo start
# Al iniciar ningún nodo del grafo ha sido visitado
searched = [False] * len(graph)
# Lista para almacenar los nodos que pertenecen al componente conectado
componentR = []
# Diccionario para representar el árbol DFS
dfs_tree = {nodo: [] for nodo in graph}

dfs(nodoS, graph, searched, componentR, dfs_tree)
print(f"La búsqueda DFS es: {componentR}")
print(f"El Árbol DFS es: {dfs_tree}")
```

```
Nodo actual: 0, Padre: None
[True, False, False, False, False, False]
Vecinos de 0 : [2, 1, 5]
Nodo actual: 2, Padre: 0
[True, False, True, False, False, False]
Vecinos de 2 : [0, 1, 3, 4]
Nodo actual: 1, Padre: 2
[True, True, True, False, False, False]
```

```
Vecinos de 1 : [0, 2]
Finaliza 1
Vuelve a 2

Nodo actual: 3, Padre: 2
[True, True, True, True, False, False]
Vecinos de 3 : [5, 4, 2]
Nodo actual: 5, Padre: 3
[True, True, True, True, False, True]
Vecinos de 5 : [3, 0]
Finaliza 5
Vuelve a 3

Nodo actual: 4, Padre: 3
[True, True, True, True, True, True]
Vecinos de 4 : [3, 2]
Finaliza 4
Vuelve a 3

Finaliza 3
Vuelve a 2

Finaliza 2
Vuelve a 0

La búsqueda DFS es: [0, 2, 1, 3, 5, 4]
El Árbol DFS es: {0: [2], 1: [], 2: [1, 3], 3: [5, 4], 4: [], 5: []}
```

4. Conclusiones

- El algoritmo DFS destaca por su método de profundización en cada rama de un grafo antes de retroceder. Este enfoque permite una exploración exhaustiva de los caminos desde el nodo inicial hasta los nodos terminales, siendo particularmente útil en aplicaciones que requieren una búsqueda completa de rutas. La implementación del ejercicio ilustra cómo DFS avanza hasta el final de una rama antes de retroceder para investigar otras opciones.
- Al implementar DFS, es crucial marcar los nodos visitados para prevenir ciclos infinitos. En grafos cíclicos, si no se registra qué nodos han sido visitados, el algoritmo puede quedar atrapado en un bucle interminable.

5. Declaración del Uso de IA

En este informe se utilizó IA para mejorar la coherencia, cohesión y sintaxis de las conclusiones.

6. Referencias Bibliográficas

«Depth First Search or DFS for a Graph - GeeksforGeeks — geeksforgeeks.org». https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/.