BFS / DFS

Algo 알고😆

진홍엽

목차

- 눈으로 살펴보기
- DFS
  - 재귀 함수
- BFS
- 비교 정리

눈으로 살펴보기

https://visualgo.net/en/dfsbfs

DFS 깊이우선탐색

- 스택

```
visited=[False]*9

def dfs(graph,v,visited):
  #현재 노드 방문 처리
  visited[v]=[True]
  print(v, end=' ')
  for i in graph[v]:
    if not visited[i]:
      dfs(graph,i,visited)
```

- 재귀함수

```
# DFS 메서드 정의
def dfs(graph, v, visited):
    # 현재 노드를 방문 처리
    visited[v] = True
    print(v, end=' ')
    # 현재 노드와 연결된 다른 노드를 재귀적으로 방문
    for i in graph[v]:
        if not visited[i]:
            dfs(graph, i, visited)

# 각 노드가 방문된 정보를 표현 (1차원 리스트)
visited = [False] * 9
```

재귀 함수

- 재귀 함수 (Recursive Function)란 자기 자신을 다시 호출하는 함수를 의미
- 재귀 함수의 종료 조건을 반드시 명시해야함

- 팩토리얼 구현 예제

- $n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$
- 수학적으로 0!과 1!의 값은 1입니다.

```
# 반복적으로 구현한 n!
def factorial_iterative(n):
    result = 1
    # 1부터 n까지의 수를 차례대로 곱하기
    for i in range(1, n + 1):
       result *= i
    return result

# 재귀적으로 구현한 n!
def factorial_recursive(n):
    if n <= 1: # n이 1 이하인 경우 1을 반환
        return 1
    # n! = n * (n - 1)!를 그대로 코드로 작성하기
    return n * factorial_recursive(n - 1)

# 각각의 방식으로 구현한 n! 출력(n = 5)
print('반복적으로 구현:', factorial_iterative(5))
print('재귀적으로 구현:', factorial_recursive(5))
```

BFS 너비우선탐색

```python
from collections import deque

# BFS 메서드 정의
def bfs(graph, start, visited):
    # 큐(Queue) 구현을 위해 deque 라이브러리 사용
    queue = deque([start])
    # 현재 노드를 방문 처리
    visited[start] = True
    # 큐가 빌 때까지 반복
    while queue:
        # 큐에서 하나의 원소를 뽑아 출력하기
        v = queue.popleft()
        print(v, end=' ')
        # 아직 방문하지 않은 인접한 원소들을 큐에 삽입
        for i in graph[v]:
            if not visited[i]:
                queue.append(i)
                visited[i] = True
```

```python
# 각 노드가 연결된 정보를 표현 (2차원 리스트)
graph = [
    [],
    [2, 3, 8],
    [1, 7],
    [1, 4, 5],
    [3, 5],
    [3, 4],
    [7],
    [2, 6, 8],
    [1, 7]
]

# 각 노드가 방문된 정보를 표현 (1차원 리스트)
visited = [False] * 9

# 정의된 BFS 함수 호출
bfs(graph, 1, visited)
```

```
1 2 3 8 7 4 5 6
```

정리

	BFS	DFS
탐색방법	루트 노드(시작 지점)에서 시작해 다음 분기 (branch)로 넘어가기 전에 해당 분기를 완벽하게 탐색	루트 노드(시작 지점)에서 시작에 인접한 노드 먼저 탐색. 시작 노드부터 가까운 노드를 먼저 탐색하고 멀리 떨어져 있는 노드를 나중에 방문
구현방법	Queue	Stack, Recursive function
장점	탐색 지점이 시작 시점과 같을 때 빠르게 탐색	BFS에 비해 메모리 소모 적음
단점	어떤 층을 검색할 때 그 층에 있는 모든 노드의 자식 노드를 저장해야해서 메모리 소모 큼 탐색 지점이 멀다면 짱 느림	검색속도가 빠를 수도 있고 느릴 수도 있다
사용시기	가중치가 없는 경로의 최단 거리	각 경로의 특징을 저장해야 할 때