알고리즘

그래프 탐색 DFS, BFS

이영석

Graph

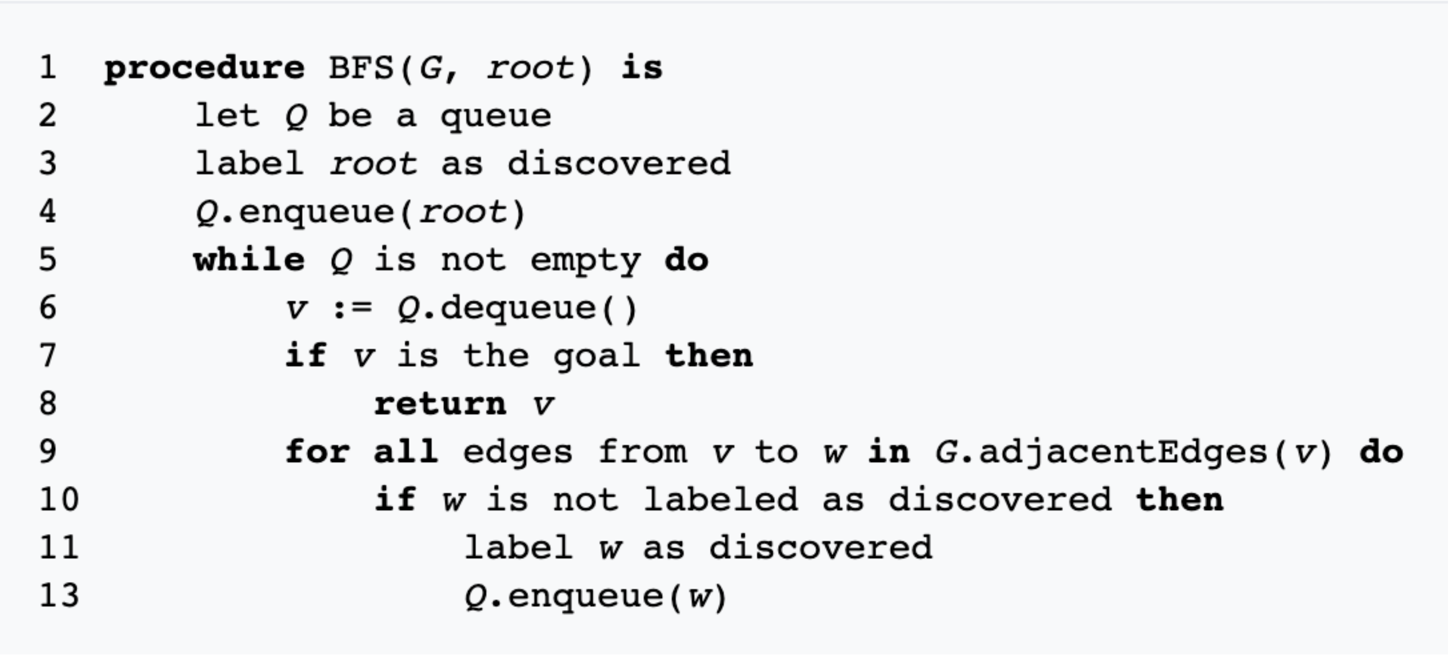
- G = (V, E)
- 자료구조
  - Adjacency matrix/list
  - 실제 구현 방법: 배열, 리스트, 사전

- 문제
  - 탐색/순회
    - BFS, DFS
  - 그래프의 순회 결과 트리: 최소비용신장트리
    - Kruskal, Prim
  - 경로
    - 최단거리: Dijkstra, Bellman-Ford, A-Star
  - Network Flow

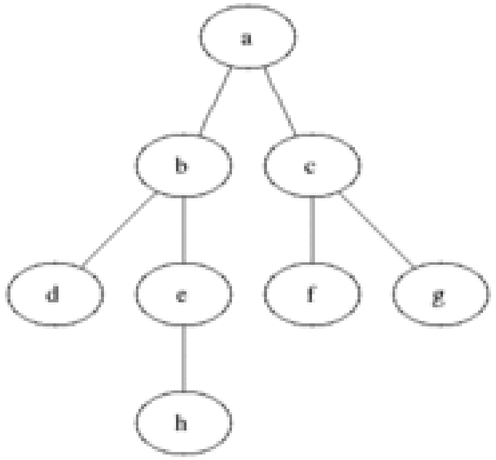
그래프 노드 순회/탐색: BFS

- Breadth First Search
    - 주어진 노드에서 모든 노드를 방문
- 구현
    - Queue 이용

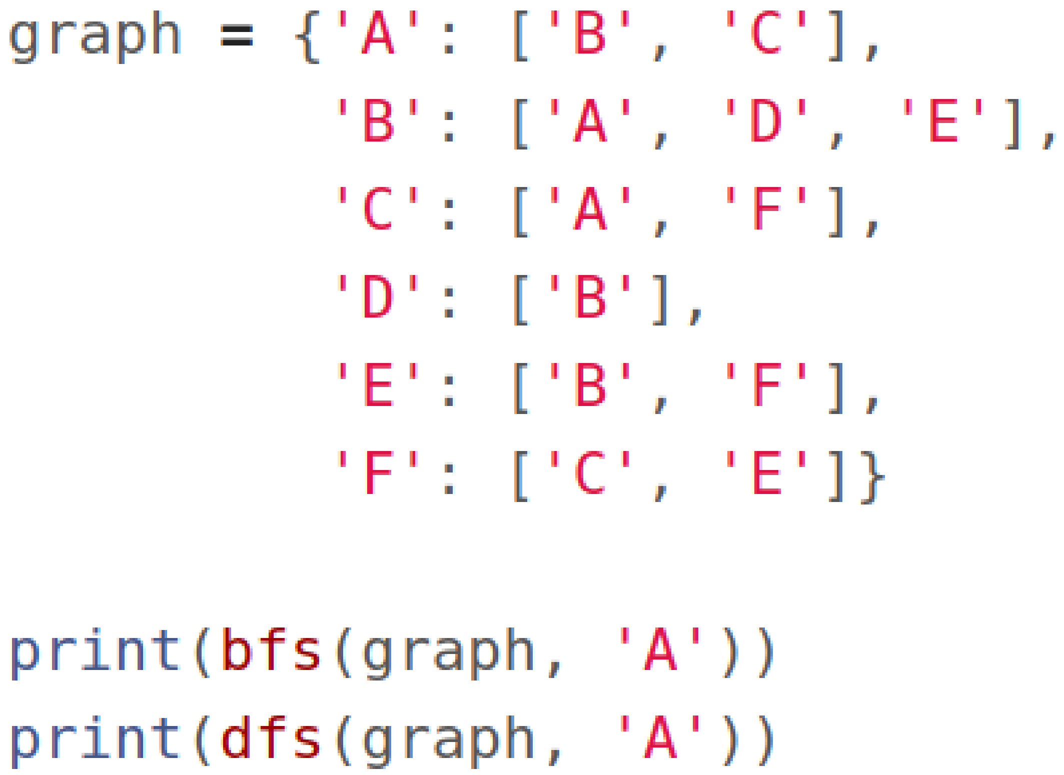
- 응용
    - P2p, SNS
    - 그래프: 최단경로, 경로찾기, 연결 컴포넌트
- 복잡도
    - O(V + E)



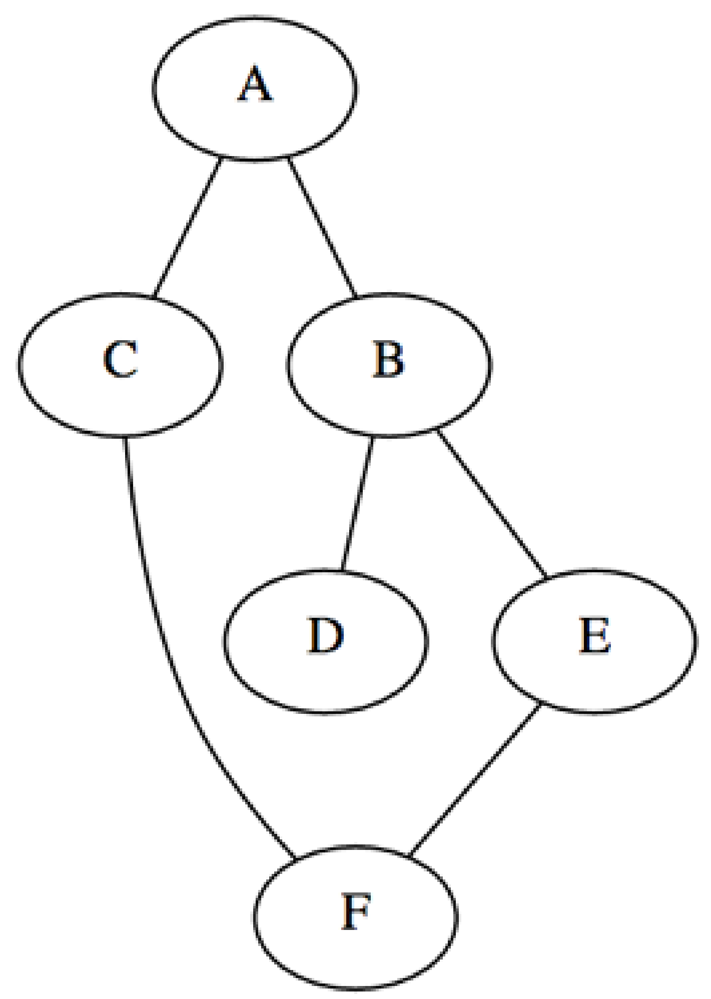
```
procedure BFS(G, root) is
    let Q be a queue
    label root as discovered
    Q.enqueue(root)
    while Q is not empty do
        v := Q.dequeue()
        if v is the goal then
            return v
        for all edges from v to w in G.adjacentEdges(v) do
            if w is not labeled as discovered then
                label w as discovered
                Q.enqueue(w)
```

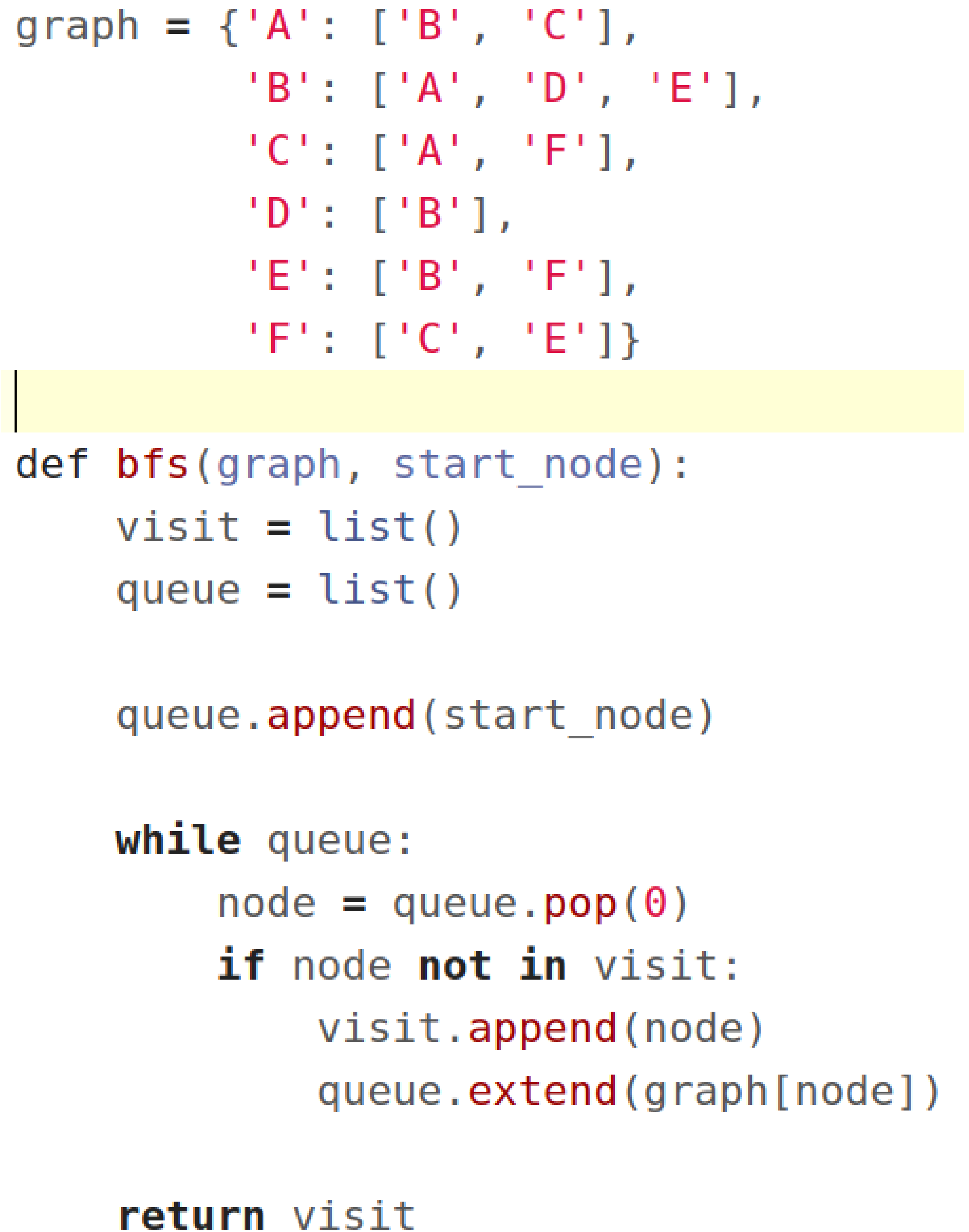


Graph in Python



```
graph = {'A': ['B', 'C'],
         'B': ['A', 'D', 'E'],
         'C': ['A', 'F'],
         'D': ['B'],
         'E': ['B', 'F'],
         'F': ['C', 'E']}

print(bfs(graph, 'A'))
print(dfs(graph, 'A'))
```

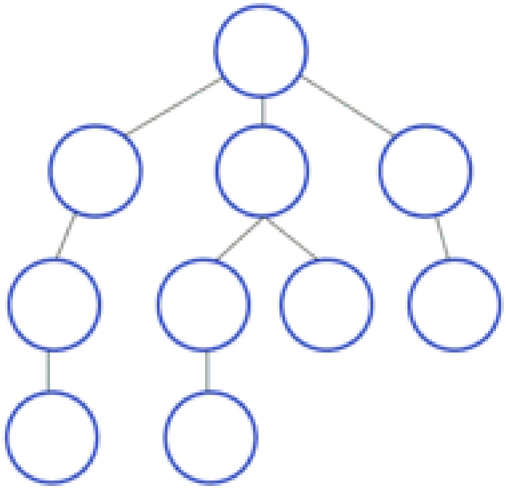


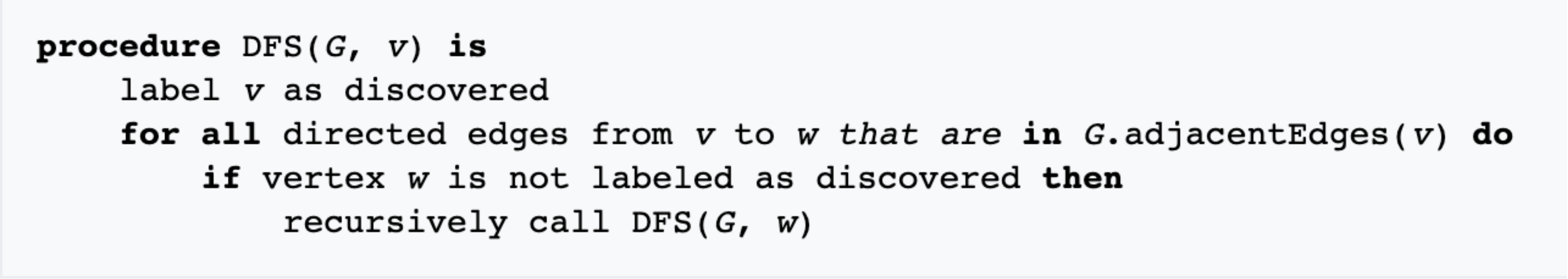


```
graph = {'A': ['B', 'C'],
         'B': ['A', 'D', 'E'],
         'C': ['A', 'F'],
         'D': ['B'],
         'E': ['B', 'F'],
         'F': ['C', 'E']}

def bfs(graph, start_node):
    visit = list()
    queue = list()

    queue.append(start_node)

    while queue:
        node = queue.pop(0)
        if node not in visit:
            visit.append(node)
            queue.extend(graph[node])

    return visit
```

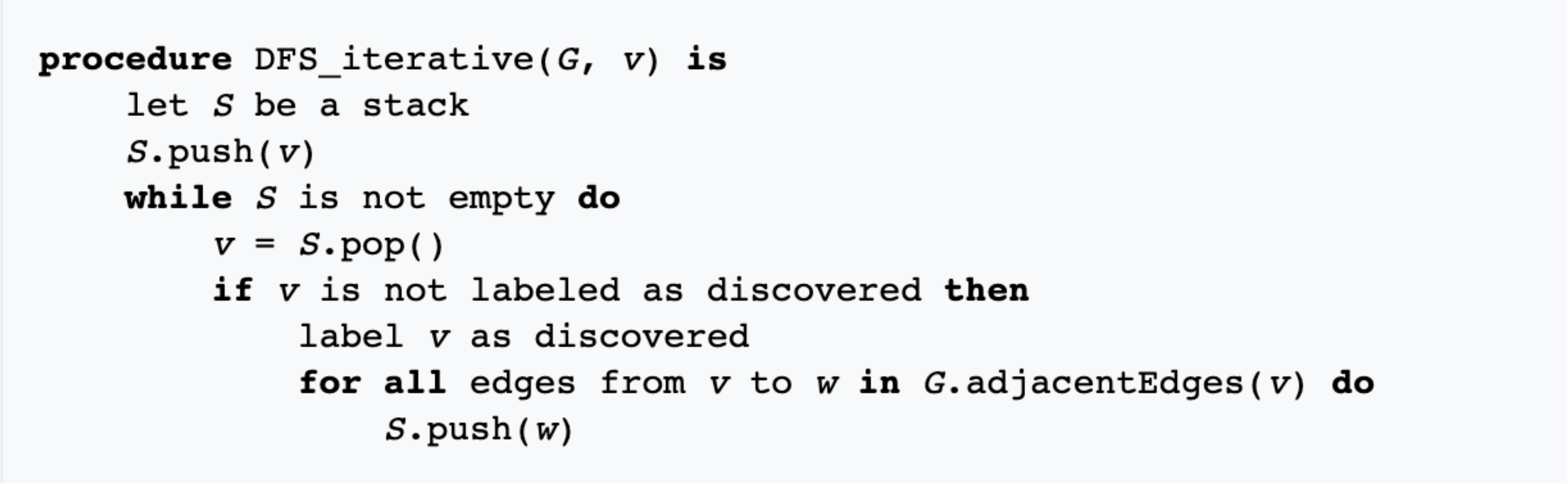
DFS

- Depth first search

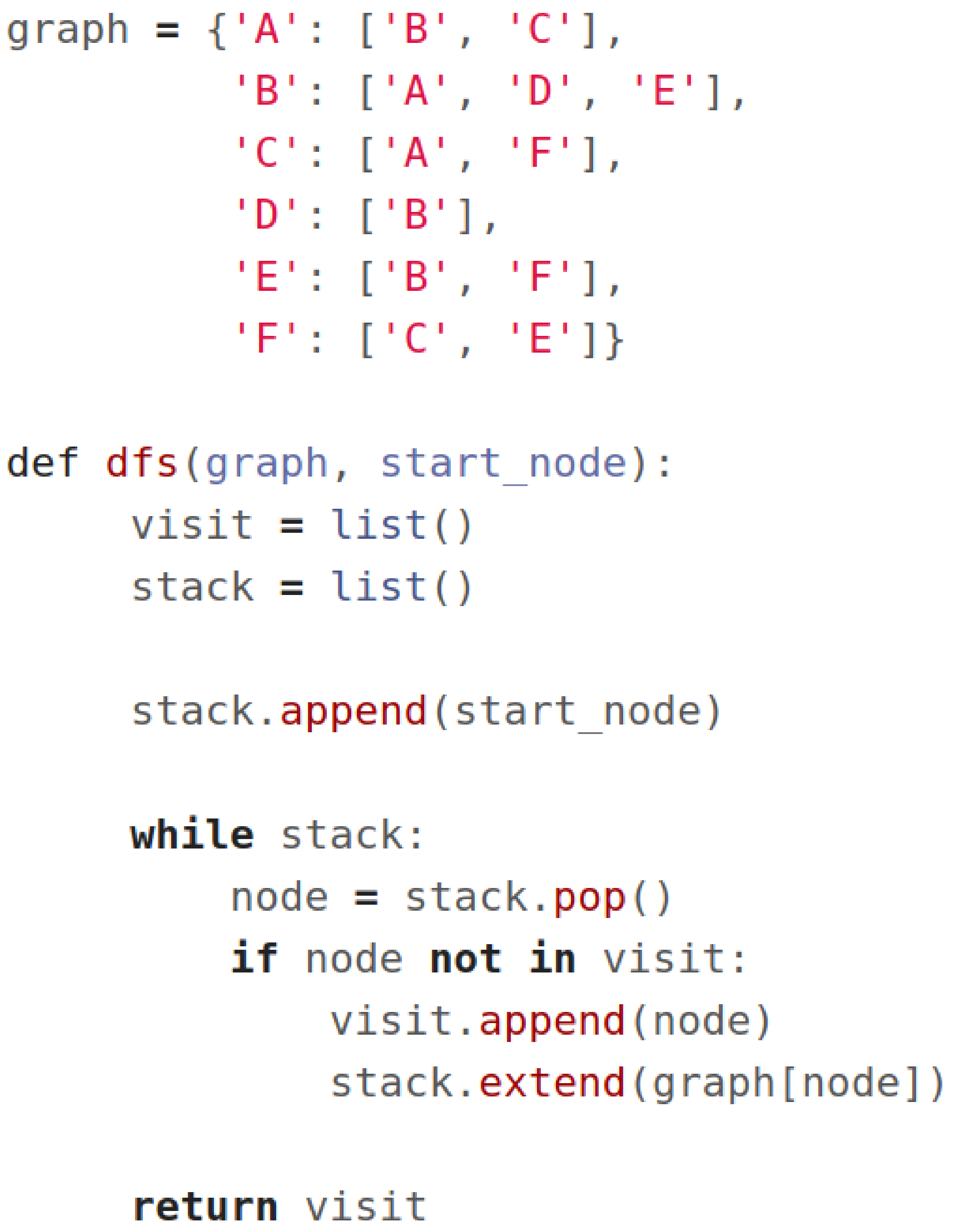




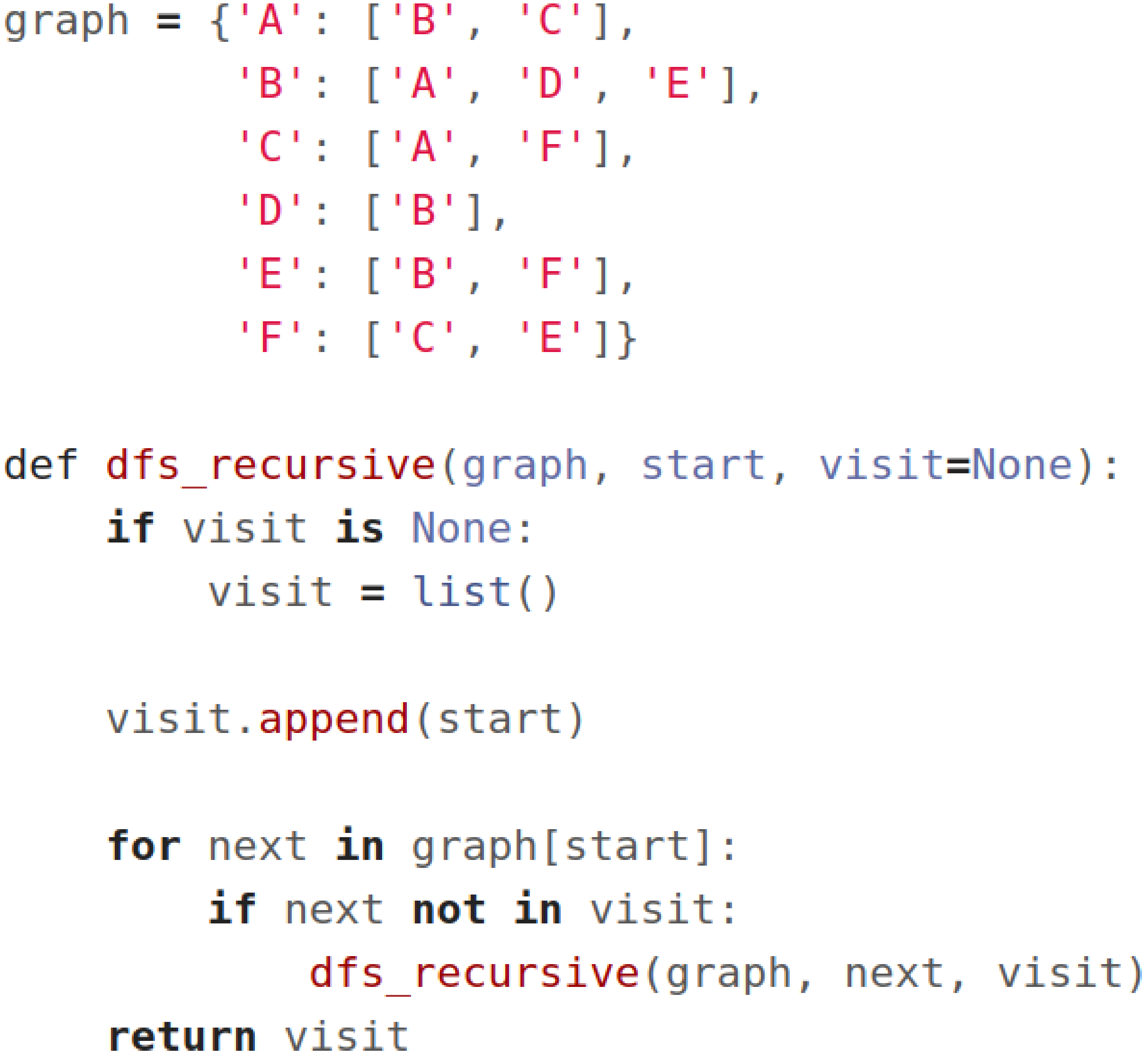
```
procedure DFS(G, v) is
    label v as discovered
    for all directed edges from v to w that are in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G, w)
```



```
procedure DFS_iterative(G, v) is
    let S be a stack
    S.push(v)
    while S is not empty do
        v = S.pop()
        if v is not labeled as discovered then
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```



```python
graph = {'A': ['B', 'C'],
         'B': ['A', 'D', 'E'],
         'C': ['A', 'F'],
         'D': ['B'],
         'E': ['B', 'F'],
         'F': ['C', 'E']}

def dfs(graph, start_node):
    visit = list()
    stack = list()

    stack.append(start_node)

    while stack:
        node = stack.pop()
        if node not in visit:
            visit.append(node)
            stack.extend(graph[node])

    return visit
```



```python
graph = {'A': ['B', 'C'],
         'B': ['A', 'D', 'E'],
         'C': ['A', 'F'],
         'D': ['B'],
         'E': ['B', 'F'],
         'F': ['C', 'E']}

def dfs_recursive(graph, start, visit=None):
    if visit is None:
        visit = list()

    visit.append(start)

    for next in graph[start]:
        if next not in visit:
            dfs_recursive(graph, next, visit)
    return visit
```