

def enqueue(a, o):

a.insert(0, o)

def dequeue(a):

return a.pop()

g = { # graph: 被搜尋的網路

'A': {'n':['B'], 'v':0}, # n: neighbor (鄰居), v: visited (是否被訪問過)

'B': {'n':['C'], 'v':0},

'C': {'n':['E'], 'v':0},

'D': {'n':['B'], 'v':0},

'E': {'n':['D','F'], 'v':0},

'F': {'n':[], 'v':0}

}

def init(g): # 初始化、設定 visited 為 0

for i in g:

g[i]['v'] = 0

def dfs(g, node): # 深度優先搜尋

if g[node]['v']!=0: # 如果已訪問過，就不再訪問

return

print(node, '=> ', end = '') # 否則、印出節點

g[node]['v'] = 1 # 並設定為已訪問

neighbors = g[node]['n'] # 取出鄰居節點

for n in neighbors: # 對於每個鄰居

dfs(g, n) # 逐一進行訪問

queue=['A'] # BFS 用的 queue, 起始點為 1。

def bfs(g, q): # 廣度優先搜尋

if len(q)==0: # 如果 queue 已空，則返回。

return

node = dequeue(q) # 否則、取出 queue 的第一個節點。

if g[node]['v'] == 0: # 如果該節點尚未拜訪過。

g[node]['v'] = 1 # 標示為已拜訪

else: # 否則 (已訪問過)

return # 不繼續搜尋，直接返回。

print(node, '=> ', end = '') # 印出節點

neighbors = g[node]['n'] # 取出鄰居。

for n in neighbors: # 對於每個鄰居

if not g[n]['v']: # 假如該鄰居還沒被拜訪過

enqueue(q, n) # 就放入 queue 中

bfs(g, q)

print('dfs:', end = '')

init(g)

dfs(g, 'A') # 呼叫深度優先搜尋。

print('')

print('bfs:', end = '')

init(g)

bfs(g, queue) # 呼叫廣度優先搜尋。

print('')