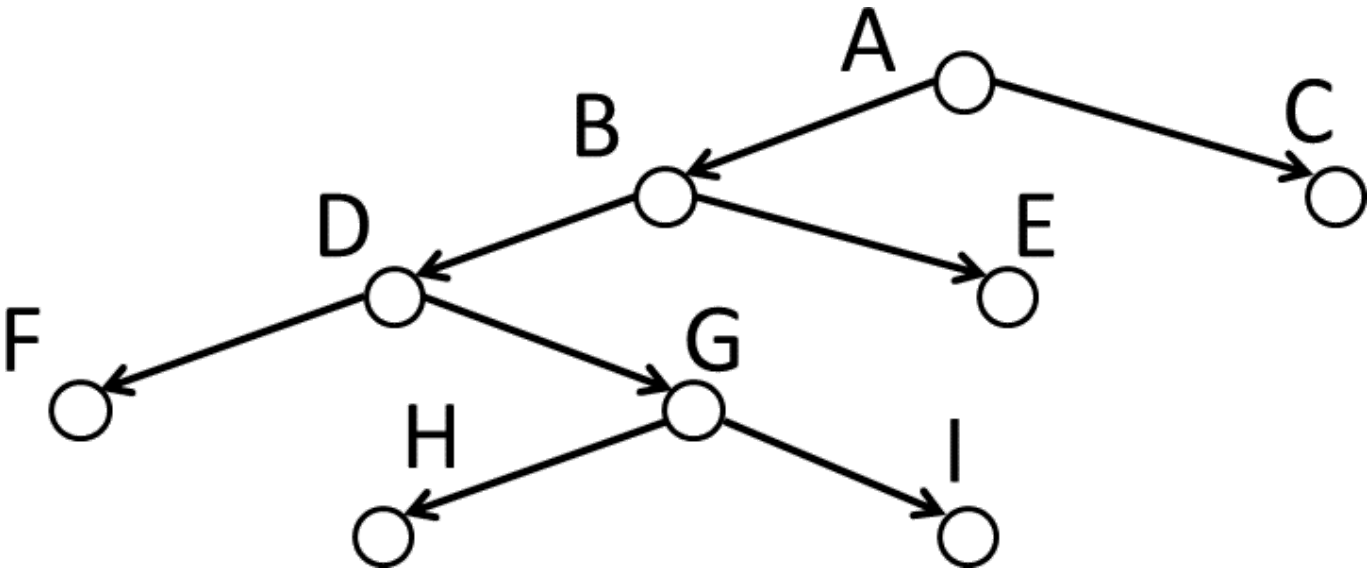
課題番号02-02



上記のグラフを、今回は簡易的に木構造で表現し、深さ優先探索(dfs), 幅優先探索(bfs)ができる関数をpython3で作成した。

ソースコードは以下のとおりである。また、ソースコードの解説はコメントにて記述した。

```
class TreeNode:
  # TreeNode(ノードにもたせる値)
  # のようにして使う。
  # 各インスタンスは そのノードの値となるval, 左側の枝への参照となるleft,
  # 右側の枝への参照となるright をインスタンス変数に持つ。
  # 探索プログラムを簡易化する都合上、末尾にはNoneを値に持つノードを設定する。
  def __init__(self, val):
    self.val = val
    if val != None:
      self.left = TreeNode(None)
      self.right = TreeNode(None)

  # 左右の枝の値を設定する。
  def set_left_and_right(self, left=None, right=None):
    self.left = TreeNode(left)
    self.right = TreeNode(right)
    return self

# 今回の課題となるグラフを木構造で表現する。
def init_tree():
  root = TreeNode('A')
  b = root.set_left_and_right('B', 'C').left
  d = b.set_left_and_right('D', 'E').left
  g = d.set_left_and_right('F', 'G').right
  g.set_left_and_right('H', 'I')
  return root

# 深さ優先探索の関数。
```

```
def dfs(node, search_val):
  open_list = []
  closed_list = []
  # (1) 探索を開始する点をopen_listに入れる。closed_listは空。
  open_list.insert(0, node)

  count = 1
  while open_list: # (2) open_listが空なら探索は終了し、Noneを返す。
    print(f"loop: {count}")
    print(f"n: {open_list[0].val}, Left: {open_list[0].left.val}, Right:
{open_list[0].right.val}")
    print(f"open_list: {list(map(lambda x: x.val, open_list))}")
    print(f"closed_list: {list(map(lambda x: x.val, closed_list))}")
    print("----------------------")
    subject_node = open_list.pop(0) # (3) open_listの先頭のn点を取り除き、
    closed_list.insert(0, subject_node) # closed_listに入れる。
    if subject_node.val == search_val: # (4) n が目標点なら探索は成功して終了
      return subject_node.val
    # 深さ優先探索固有のロジック
    if subject_node.right.val: # n の左右の枝（子）が存在すれば、
      open_list.insert(0, subject_node.right) # 全てopen_listの先頭に入れる
    if subject_node.left.val:
      open_list.insert(0, subject_node.left)
    # ここまで
    count += 1
  return None

# 幅優先探索の関数。
def bfs(node, search_val):
  open_list = []
  closed_list = []
  # (1) 探索を開始する点をopen_listに入れる。closed_listは空。
  open_list.insert(0, node)

  count = 1
  while open_list: # (2) open_listが空なら探索は終了し、Noneを返す。
    print(f"loop: {count}")
    print(f"n: {open_list[0].val}, Left: {open_list[0].left.val}, Right:
{open_list[0].right.val}")
    print(f"open_list: {list(map(lambda x: x.val, open_list))}")
    print(f"closed_list: {list(map(lambda x: x.val, closed_list))}")
    print("----------------------")
    subject_node = open_list.pop(0) # (3) open_listの先頭のn点を取り除き、
    closed_list.insert(0, subject_node) # closed_listに入れる。
    if subject_node.val == search_val: # (4) n が目標点なら探索は成功して終了
      return subject_node.val
    # 深さ優先探索固有のロジック
    if subject_node.left.val: # n の左右の枝（子）が存在すれば、
      open_list.append(subject_node.left) # 全てopen_listの末尾に入れる
    if subject_node.right.val:
      open_list.append(subject_node.right)
    # ここまで
    count += 1
  return None
```

```
def main():
  root = init_tree()
  print('================')
  print('dfs start')
  print('================')
  result = dfs(root, 'C')
  print(result)

  print('================')
  print('bfs start')
  print('================')
  result = bfs(root, 'I')
  print(result)

main()
```

探索の過程は以下のようになった。

深さ優先探索

```
================
dfs start
================
loop: 1
n: A, Left: B, Right: C
open_list: ['A']
closed_list: []
---------------------
loop: 2
n: B, Left: D, Right: E
open_list: ['B', 'C']
closed_list: ['A']
---------------------
loop: 3
n: D, Left: F, Right: G
open_list: ['D', 'E', 'C']
closed_list: ['B', 'A']
---------------------
loop: 4
n: F, Left: None, Right: None
open_list: ['F', 'G', 'E', 'C']
closed_list: ['D', 'B', 'A']
---------------------
loop: 5
n: G, Left: H, Right: I
open_list: ['G', 'E', 'C']
closed_list: ['F', 'D', 'B', 'A']
---------------------
loop: 6
n: H, Left: None, Right: None
```

```
open_list: ['H', 'I', 'E', 'C']
closed_list: ['G', 'F', 'D', 'B', 'A']
--------------------
loop: 7
n: I, Left: None, Right: None
open_list: ['I', 'E', 'C']
closed_list: ['H', 'G', 'F', 'D', 'B', 'A']
--------------------
loop: 8
n: E, Left: None, Right: None
open_list: ['E', 'C']
closed_list: ['I', 'H', 'G', 'F', 'D', 'B', 'A']
--------------------
loop: 9
n: C, Left: None, Right: None
open_list: ['C']
closed_list: ['E', 'I', 'H', 'G', 'F', 'D', 'B', 'A']
--------------------
C
```

幅優先探索

```
================
bfs start
================
loop: 1
n: A, Left: B, Right: C
open_list: ['A']
closed_list: []
--------------------
loop: 2
n: B, Left: D, Right: E
open_list: ['B', 'C']
closed_list: ['A']
--------------------
loop: 3
n: C, Left: None, Right: None
open_list: ['C', 'D', 'E']
closed_list: ['B', 'A']
--------------------
loop: 4
n: D, Left: F, Right: G
open_list: ['D', 'E']
closed_list: ['C', 'B', 'A']
--------------------
loop: 5
n: E, Left: None, Right: None
open_list: ['E', 'F', 'G']
closed_list: ['D', 'C', 'B', 'A']
--------------------
```

```
loop: 6
n: F, Left: None, Right: None
open_list: ['F', 'G']
closed_list: ['E', 'D', 'C', 'B', 'A']
--------------------
loop: 7
n: G, Left: H, Right: I
open_list: ['G']
closed_list: ['F', 'E', 'D', 'C', 'B', 'A']
--------------------
loop: 8
n: H, Left: None, Right: None
open_list: ['H', 'I']
closed_list: ['G', 'F', 'E', 'D', 'C', 'B', 'A']
--------------------
loop: 9
n: I, Left: None, Right: None
open_list: ['I']
closed_list: ['H', 'G', 'F', 'E', 'D', 'C', 'B', 'A']
--------------------
I
```

参考文献

Python Software Foundation."5. データ構造 — Python 3.11.4 ドキュメント".https://docs.python.org/ja/3/tutorial/datastructures.html