DFS_recursion

May 5, 2020

1 Graph Depth-First Search With Recursion

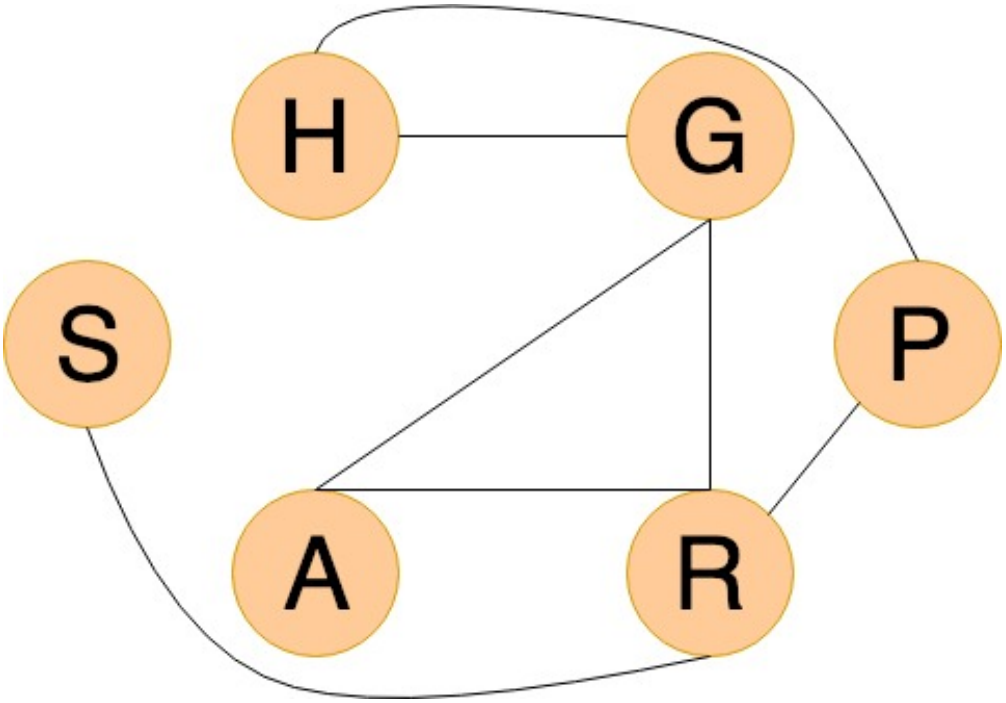
We've done depth-first search previously using an iterative approach (i.e., using a loop). In this notebook, we'll show how to implement a recursive soluton.

The basic idea is to select a node and explore all the possible paths from that node, and to apply this recursively to each node we are exploring.

You can see some helpful illustrations with various combinations here: https://www.cs.usfca.edu/~galles/visualization/DFS.html

In [1]:
```
# For this exercise we will be using an Adjacency List representation to store the graph

    # Class Node representation.
    class Node:
        def __init__(self,val):
            self.value = val
            self.children = []

        def add_child(self,new_node):
            self.children.append(new_node)

        def remove_child(self,del_node):
            if del_node in self.children:
                self.children.remove(del_node)

    class Graph():
        def __init__(self,node_list):
            self.nodes = node_list

        def add_edge(self,node1,node2):
            if(node1 in self.nodes and node2 in self.nodes):
                node1.add_child(node2)
                node2.add_child(node1)

        def remove_edge(self,node1,node2):
            if(node1 in self.nodes and node2 in self.nodes):
                node1.remove_child(node2)
                node2.remove_child(node1)
```

1.0.1 Initializing Graph with an example



Consider the above graph structure. The following code initializes all the edges according to the above structure.

```
In [2]: # Creating a graph as above.
        nodeG = Node('G')
        nodeR = Node('R')
        nodeA = Node('A')
        nodeP = Node('P')
        nodeH = Node('H')
        nodeS = Node('S')

        graph1 = Graph([nodeS,nodeH,nodeG,nodeP,nodeR,nodeA] )

        graph1.add_edge(nodeG,nodeR)
        graph1.add_edge(nodeA,nodeR)
        graph1.add_edge(nodeA,nodeG)
        graph1.add_edge(nodeR,nodeP)
        graph1.add_edge(nodeH,nodeG)
        graph1.add_edge(nodeH,nodeP)
        graph1.add_edge(nodeS,nodeR)
```

```
In [3]: # To verify that the graph is created accurately.
        # Let's just print all the parent nodes and child nodes.
        for each in graph1.nodes:
            print('parent node = ',each.value,end='\nchildren\n')
            for each in each.children:
                print(each.value,end=' ')
            print('\n')
```

```
parent node =  S
children
R

parent node =  H
children
G P

parent node =  G
children
R A H

parent node =  P
children
R H

parent node =  R
children
G A P S

parent node =  A
children
R G
```

1.0.2 Sample input and output

The output would vary based on the implementation of your algorithm, the order in which children are stored within the adjacency list.

1.0.3 DFS using recursion

Now that we have our example graph initialized, we are ready to do the actual depth-first search. Here's what that looks like:

```
In [4]: def dfs_recursion_start(self, start_node):
            visited = {}
            self.dfs_recursion(start_node, visited)

        def dfs_recursion(self, node,visited):

            if(node == None):
                return False

            visited[node.value] = True
            print(node.value)
```

```
            for each in node.children:
                if( each.value not in visited ):
                    self.dfs_recursion(each,visited)
```

In [5]:
```
Graph.dfs_recursion_start = dfs_recursion_start

Graph.dfs_recursion = dfs_recursion

graph1.dfs_recursion_start(nodeG)
```

```
G
R
A
P
H
S
```

In [ ]: