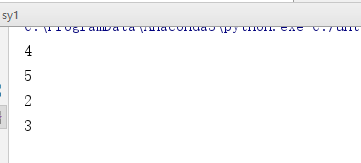
深度优先遍历

深度优先遍历算法dfs通俗的说就是“顺着起点往下走，直到无路可走就退回去找下一条路径，直到走完所有的结点”。

**import** queue  
**def** dfs(adj, start):  
 visited = set()  
 stack = [[start, 0]]  
 **while** stack:  
 (v, next\_child\_idx) = stack[-1]  
 **if** (v **not in** adj) **or** (next\_child\_idx >= len(adj[v])):  
 stack.pop()  
 **continue** next\_child = adj[v][next\_child\_idx]  
 stack[-1][1] += 1  
 **if** next\_child **in** visited:  
 **continue** print(next\_child)  
 visited.add(next\_child)  
 stack.append([next\_child, 0])  
  
  
graph = {1: [4, 2], 2: [3, 4], 3: [4], 4: [5]}  
  
  
dfs(graph, 1)



首次是“顺着起点往下走”中的起点当然就是函数传进来的参数start，第三行中把起点放到了栈中，此时栈就是初始状态，其中就只有一个元素即起点。那么栈中元素表示的语义是：下一次将访问的结点，由于使用邻接表来表示图，那么要表示一个结点表可以用<这个结点的父结点、这个结是父结点的第几个子结点来决定，由这们这里使用的图的存储方式-邻接表决定了，因为这样取第N个兄弟结点要容易了。因为邻接表中用list来表示一个结点的所有子结点，我们就用一个整数的索引值来保存下次要访问的子结点的list的下标，当这个下标超过子结点list的长度时意味着访问完所有子结点。

接着，“往下走”，看这句：next\_child = adj[v][next\_child\_idx]就是我们在这个while循环中每次访问的都是一个子结点，访问完当前结点后stack.append([next\_child, 0])将这个结点放到栈中，意思是下次就访问这个结点的子结点，这样就每次都是往下了。

“直到无路可走”，在程序中的体现就是 if (v not in adj) or (next\_child\_idx >= len(adj[v])):，栈顶元素表示即将要访问的结点的父结点及其是父结点的第N个子结点（有点绕），这里的意思是如果这个父结点都没有子结点了或者是想要访问第N个子结点但是父结点并没有这么多子结点，表示已经访问完了一个父结点的所有子结点了。

接着“就退回去找下一条路径”中的“退回去”，将栈顶元素弹出，新的栈顶元素就是它的父结点，那么就是退回去了，“去找下一条路径”就是弹出栈顶后下一次while中会沿着父结点继续探索，也就是去找下一条路径了。

最后“直到走完所有的结点“当然就是栈为空了，栈为空表示已经回退到起点，即所有结点已经访问完了，整个算法结束。