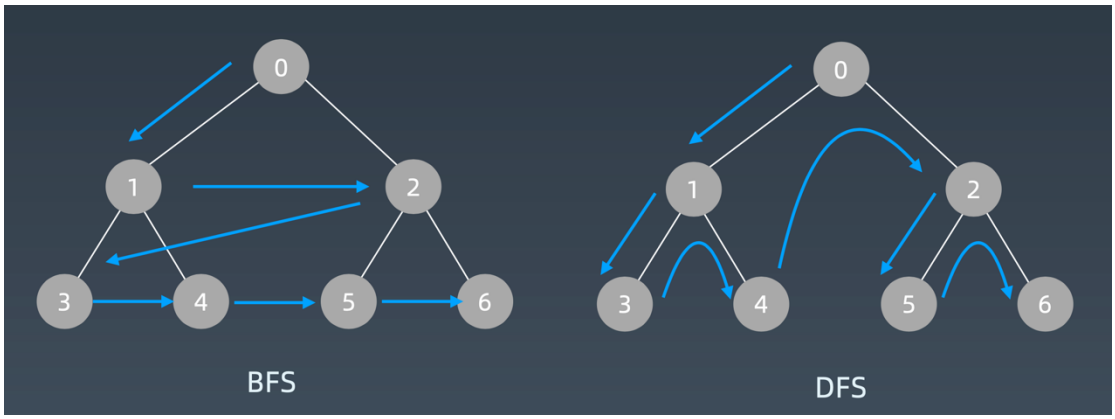
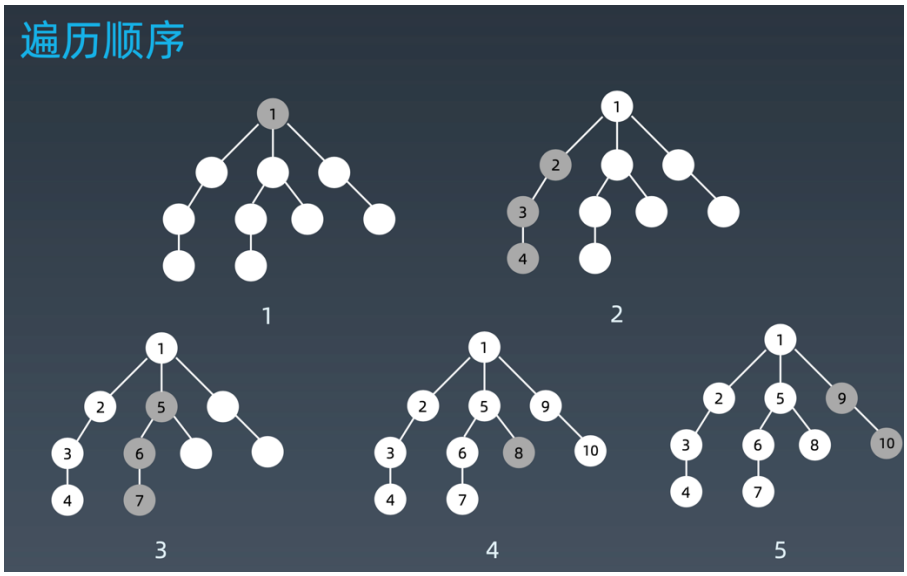
Week4：DFS/BFS 遍历



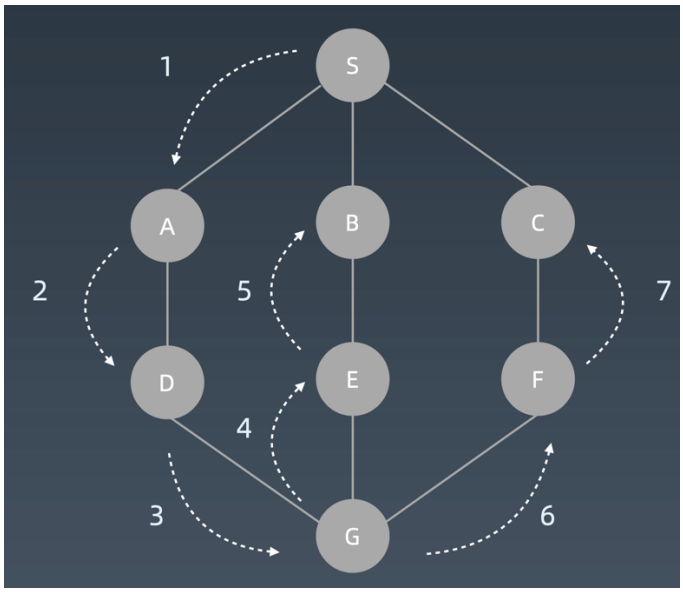
1) DFS 深度优先

-遍历顺序：*一冲到底 stack*

-树



-图



-递归写法：

DFS 代码 - 递归写法

```
visited = set()

def dfs(node, visited):
  if node in visited: # terminator
     # already visited
     return

   visited.add(node)

   # process current node here.
   ...
   for next_node in node.children():
      if not next_node in visited:
         dfs(next node, visited)
```

-迭代写法：自主维护一个 *stack* 来模拟系统的 stack

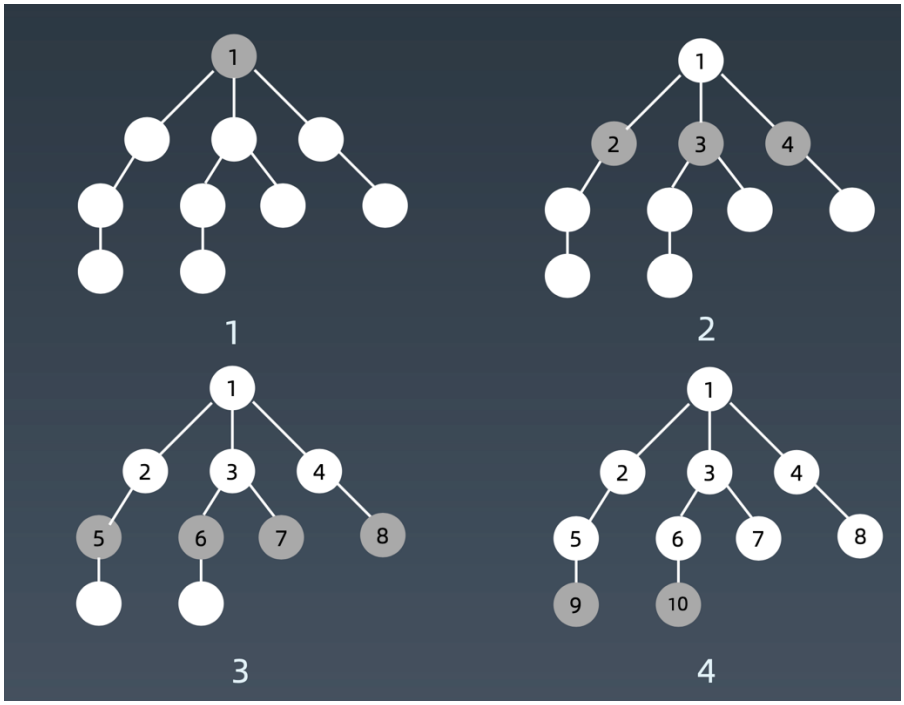
DFS 代码 - 非递归写法

```
def DFS(self, tree):

    if tree.root is None:
        return []

    visited, stack = [], [tree.root]

    while stack:
        node = stack.pop()
        visited.add(node)

        process (node)
        nodes = generate_related_nodes(node)
        stack.push(nodes)

    # other processing work
    ...
```

2) BFS 广度优先

-遍历顺序：*层层展开 queue*

-树



-写法：维护一个 *queue*

BFS 代码

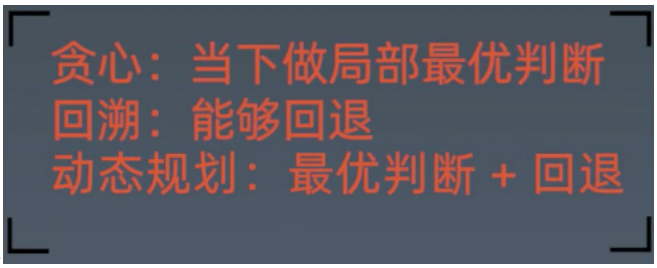
```
def BFS(graph, start, end):

    queue = []
    queue.append([start])
    visited.add(start)

    while queue:
        node = queue.pop()
        visited.add(node)

        process(node)
        nodes = generate_related_nodes(node)
        queue.push(nodes)

    # other processing work
    ...
```

3）贪心算法

-局限：鼠目寸光？

-定义：贪心算法是一种在每一步选择中都采取在当前状态下最好或最优(最有利)的选择，从而希望导致结果是全局最好或最优的算法

-与动态规划区别：贪心对子问题的解决方案（解决子问题的方案一旦设定，则每次都会运用相同的方案）不能回退；动规则会保存以前的运算结果，并可以根据以前结果对当前进行选择，有回退功能

- 

-主要解决最优化问题

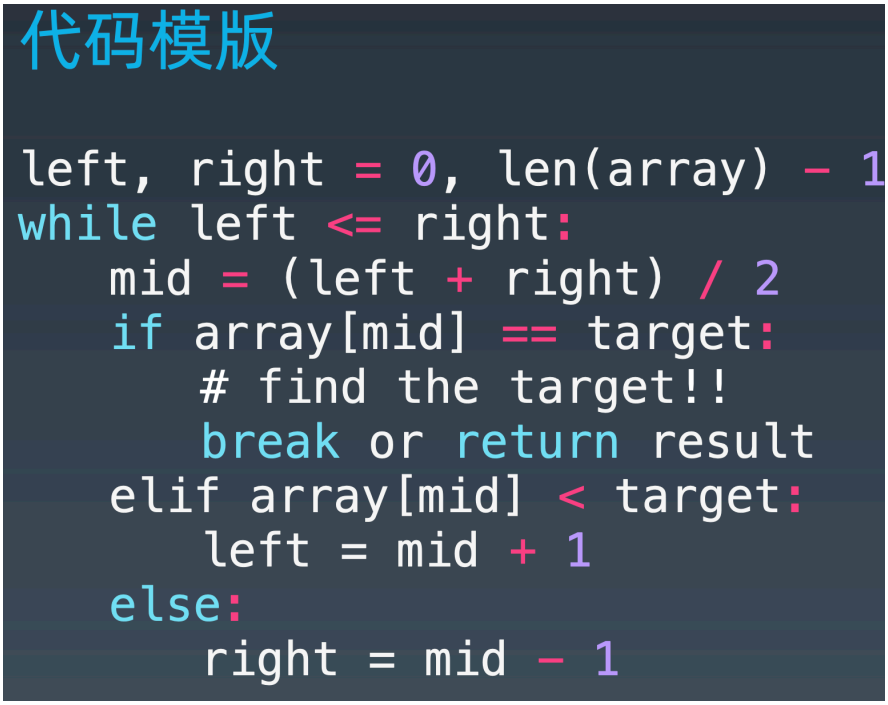
-适用场景：问题能够分解成子问题来解决，子问题的最优解能递推到最终问题的最优解；这种子问题最优解称为最优子结构；

4）二分查找

-使用前提 *为什么能用？？？*：

1）目标函数单调性
2）存在上下界
3）能够通过索引访问

-代码模版：



-牛顿迭代法

http://www.matrix67.com/blog/archives/361

-四步做题：

审题（细节，边界条件，input，output）；

考虑所有解法得出最优法；

写代码；

测试 case；

作业：

1）用广度优先搜索写 leetcode22