数据结构与算法（Python版）

实现广度优先搜索

陈斌 北京大学 gischen@pku.edu.cn

实现广度优先搜索

- 在单词关系图建立完成以后，需要继续在图中寻找词梯问题的最短序列
- 需要用到“**广度优先搜索**Breadth First Search”算法对单词关系图进行搜索
- BFS是搜索图的最简单算法之一，也是其它一些重要的图算法的基础

实现广度优先搜索

- 给定图G，以及开始搜索的起始顶点s

  BFS搜索所有从s可到达顶点的边

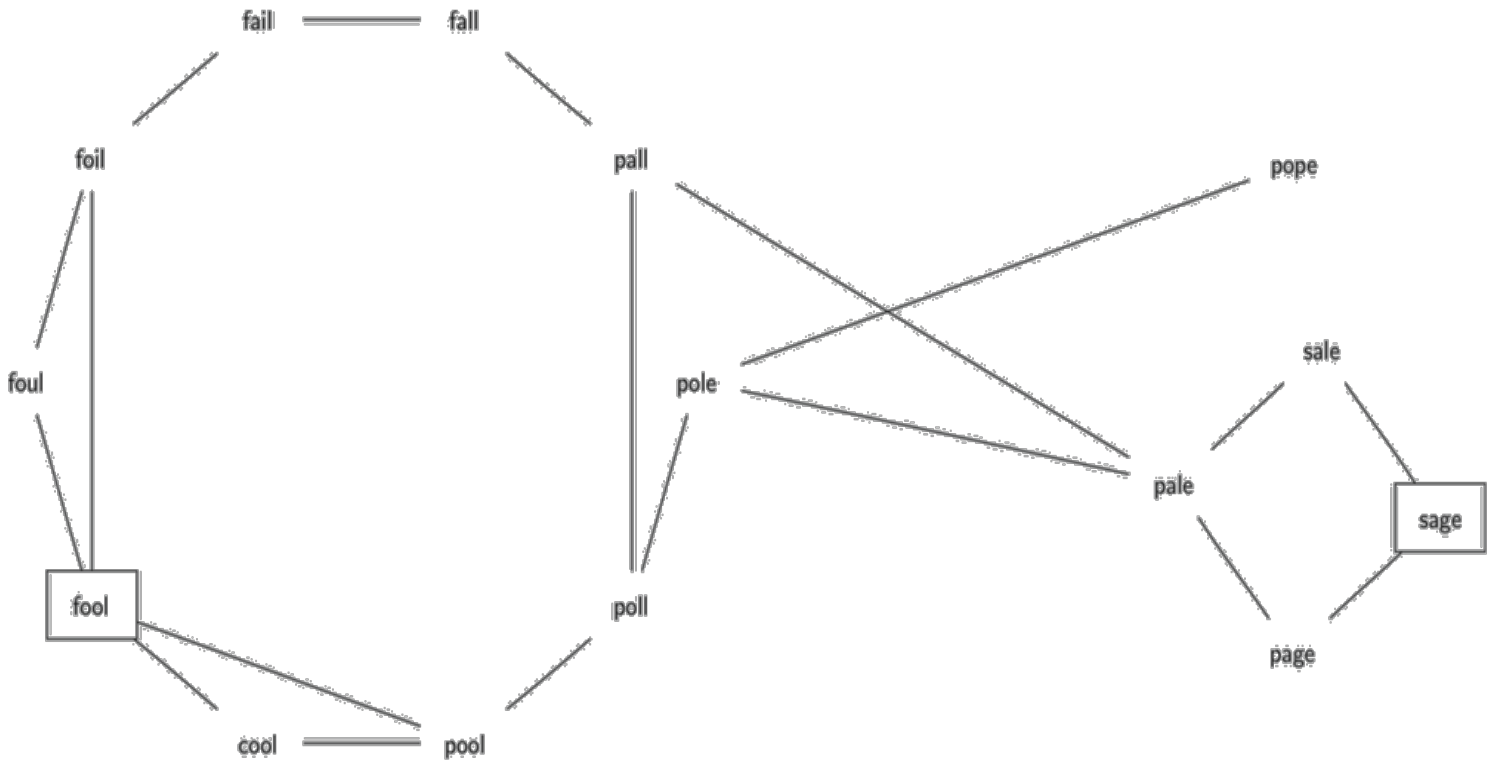
  而且在达到更远的距离k+1的顶点之前，BFS会找到全部距离为k的顶点

  可以想象为以s为根，构建一棵树的过程，从顶部向下逐步增加层次

  广度优先搜索能保证在增加层次之前，添加了所有兄弟节点到树中

BFS算法过程

- 我们从FOOL开始搜索



BFS算法过程

- 为了跟踪顶点的加入过程，并避免重复顶点，要为顶点增加3个属性

  距离distance：从起始顶点到此顶点路径长度；

  前驱顶点predecessor：可反向追溯到起点；

  颜色color：标识了此顶点是尚未发现（白色）、已经发现（灰色）、还是已经完成探索（黑色）

- 还需要用一个队列Queue来对已发现的顶点进行排列

  决定下一个要探索的顶点（队首顶点）

BFS算法过程

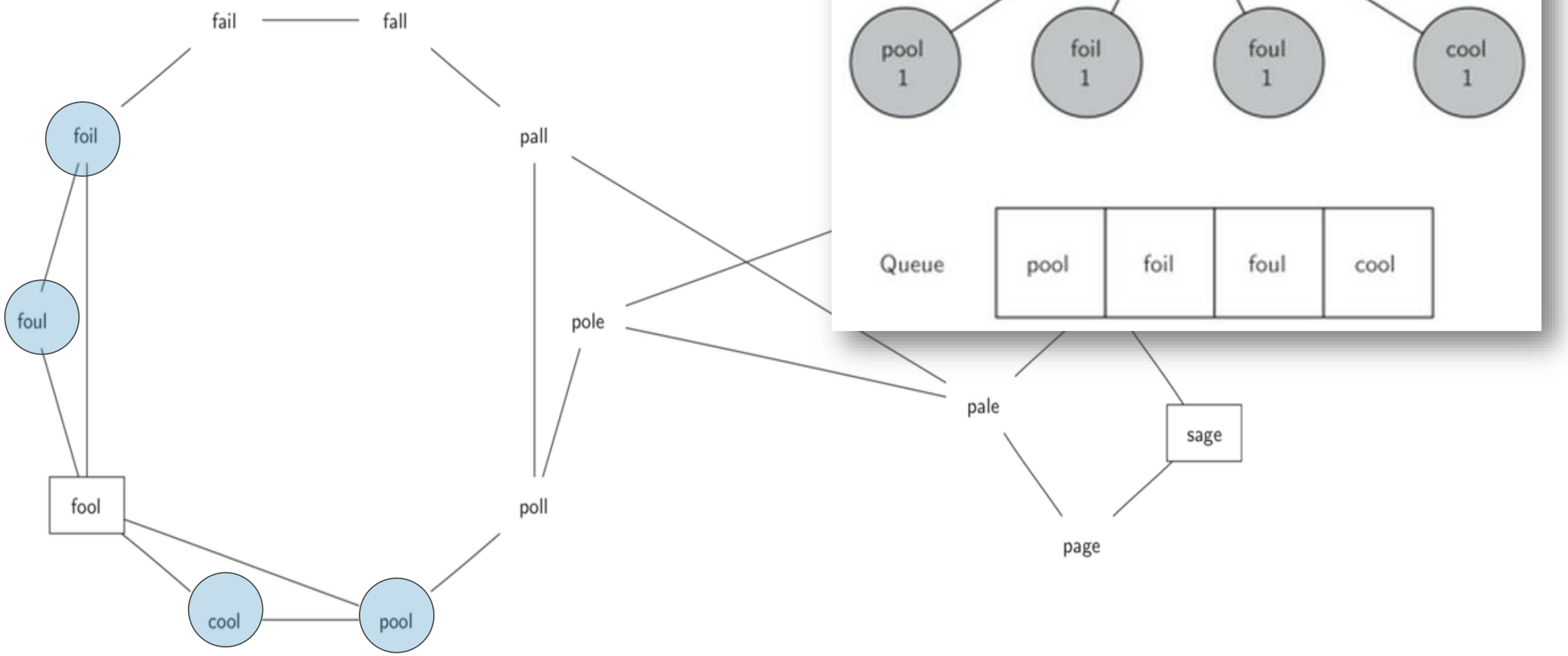
- 从起始顶点s开始，作为刚发现的顶点，标注为灰色，距离为0，前驱为None，加入队列，接下来是个循环迭代过程：

  从队首取出一个顶点作为当前顶点；

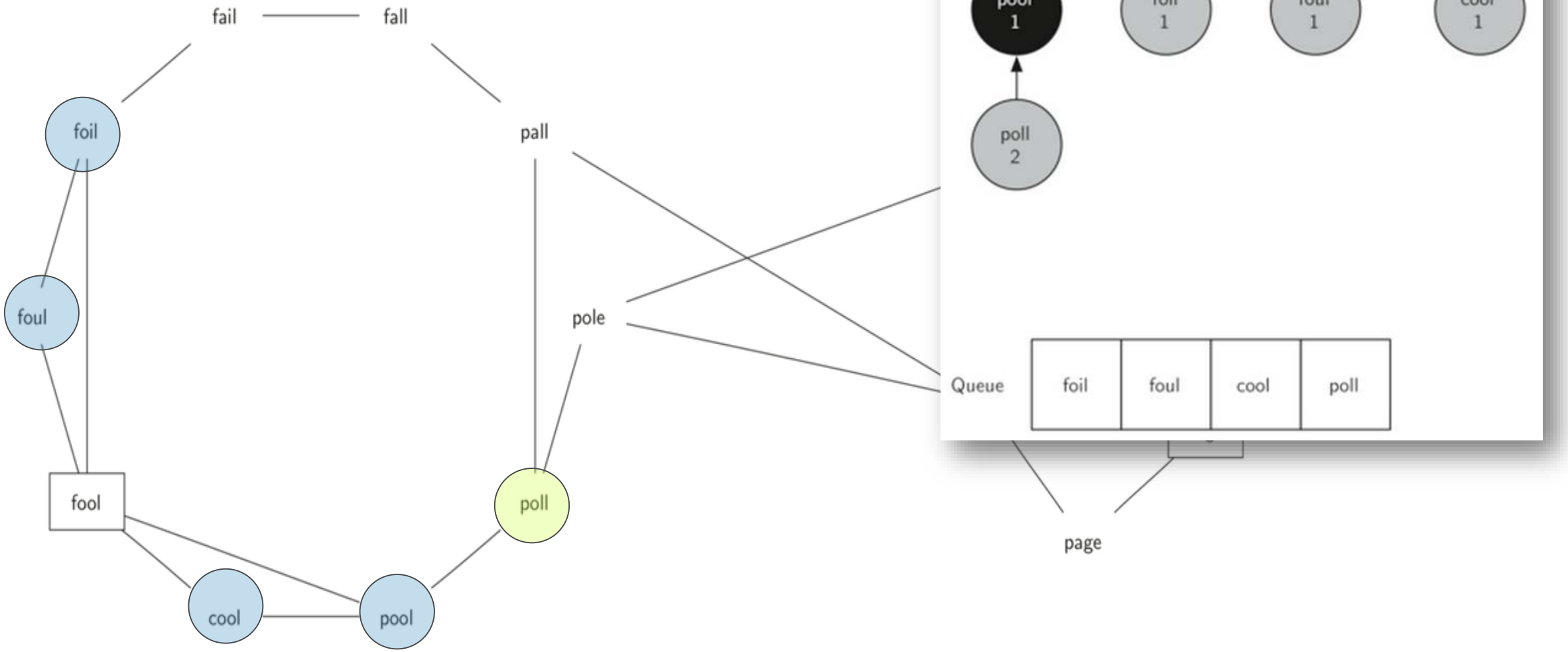
  遍历当前顶点的邻接顶点，如果是尚未发现的白色顶点，则将其颜色改为灰色（已发现），距离增加1，前驱顶点为当前顶点，加入到队列中

  遍历完成后，将当前顶点设置为黑色（已探索过），循环回到步骤1的队首取当前顶点

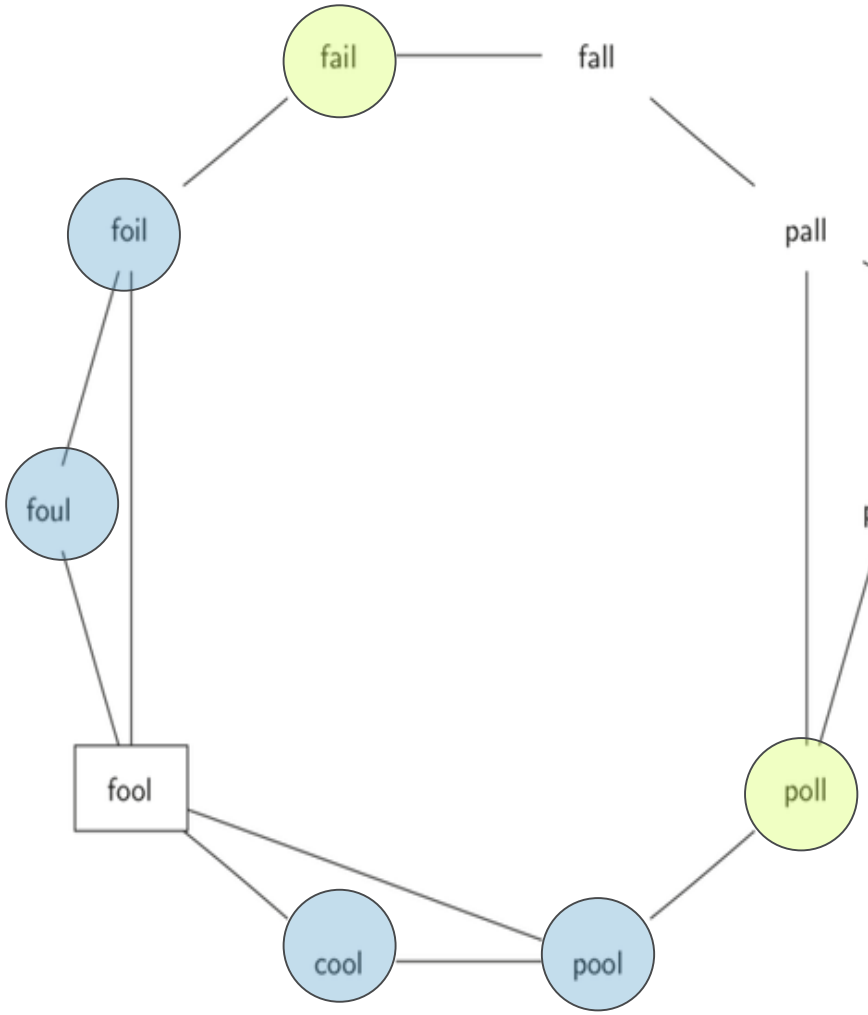
BFS算法实例运行

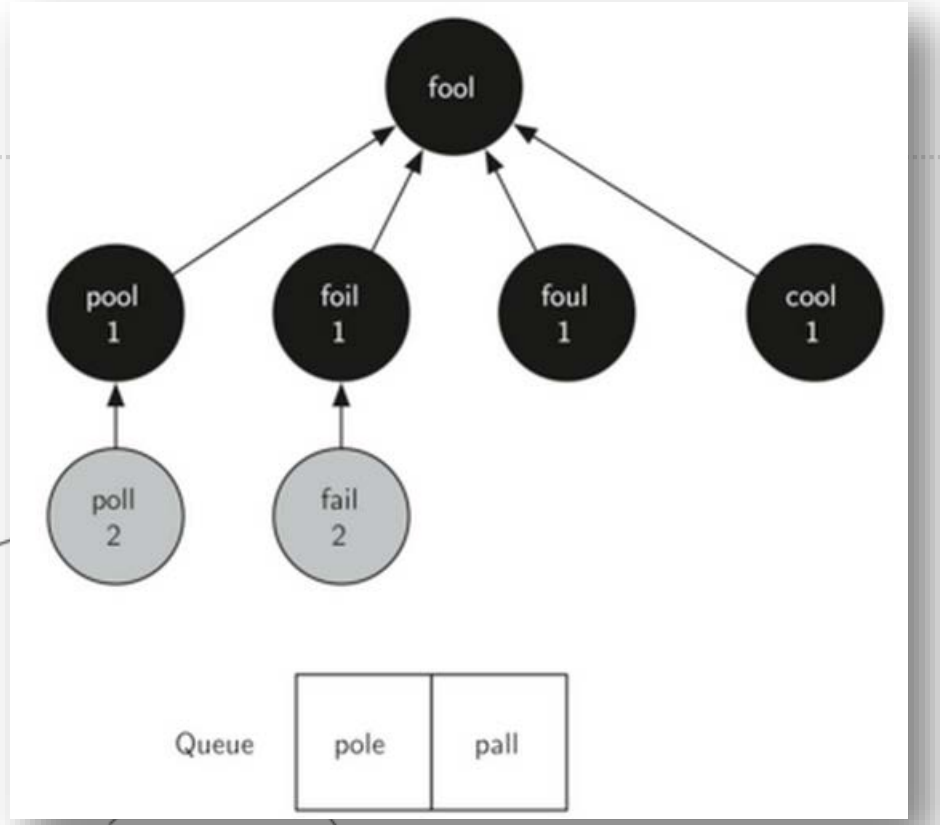

fail
fall
foil
pall
foul
pole
fool
poll
cool
pool
fool
pool
1
foil
1
foul
1
cool
1
Queue
pool
foil
foul
cool
pale
sage
page

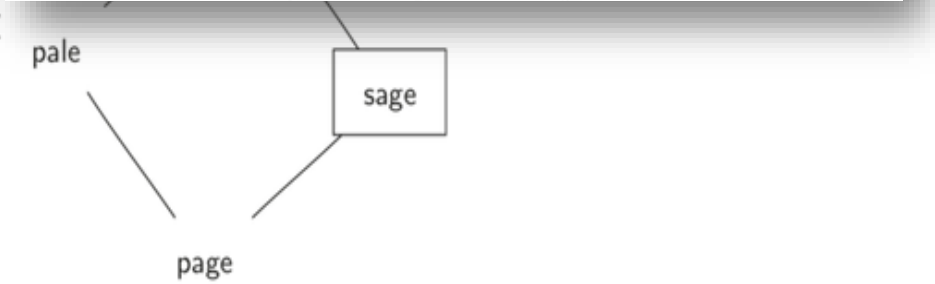
BFS算法实例运行


fail
fall
foil
pall
foul
pole
fool
poll
cool
pool
fool
pool
1
foil
1
foul
1
cool
1
poll
2
Queue
foil
foul
cool
poll
page

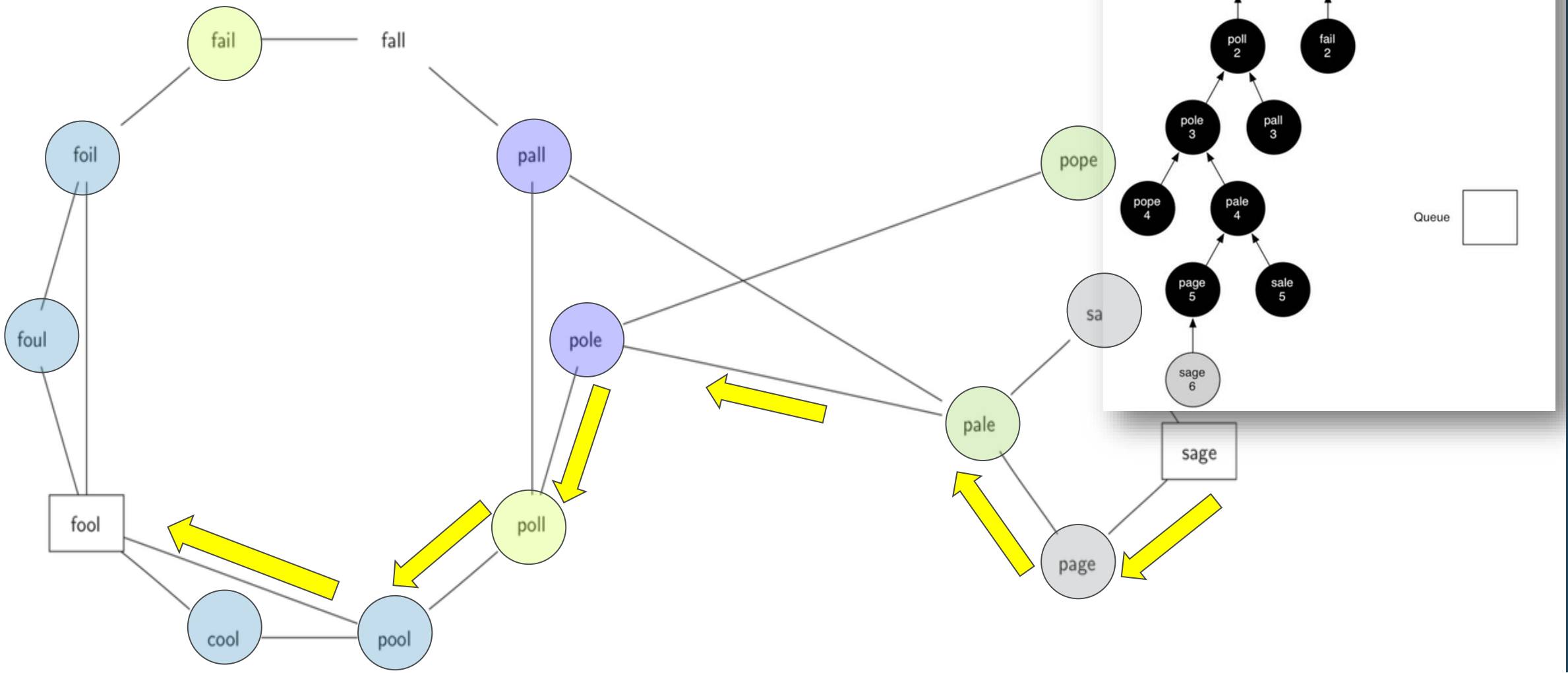
BFS算法实例运行


fail
fall
foil
pall
foul
pole
fool
poll
cool
pool


fool
pool 1
foil 1
foul 1
cool 1
poll 2
fail 2
Queue
pole
pall


pale
sage
page

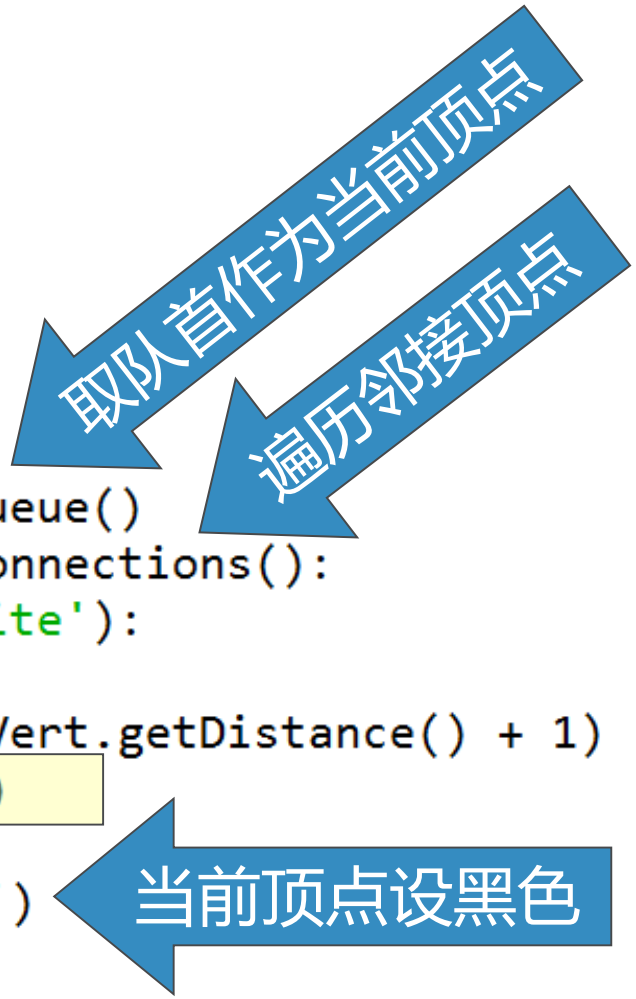
BFS算法实例运行


fail
fall
foil
pall
pope
foul
pole
sa
pale
fool
poll
sage
page
cool
pool

fool
pool 1
foil 1
foul 1
cool 1
poll 2
fail 2
pole 3
pall 3
pope 4
pale 4
page 5
sale 5
sage 6
Queue

BFS算法代码


```
def bfs(g,start):
  start.setDistance(0)
  start.setPred(None)
  vertQueue = Queue()
  vertQueue.enqueue(start)
  while (vertQueue.size() > 0):
    currentVert = vertQueue.dequeue()
    for nbr in currentVert.getConnections():
      if (nbr.getColor() == 'white'):
        nbr.setColor('gray')
        nbr.setDistance(currentVert.getDistance() + 1)
        nbr.setPred(currentVert)
        vertQueue.enqueue(nbr)
    currentVert.setColor('black')
```



BFS算法代码

- 在以FOOL为起始顶点，遍历了所有顶点，并为每个顶点着色、赋距离和前驱之后
- 即可以通过一个回途追溯函数来确定FOOL到任何单词顶点的最短词梯！

```
def traverse(y):
    x = y
    while (x.getPred()):
        print(x.getId())
        x = x.getPred()
    print(x.getId())


wordgraph = buildGraph("fourletterwords.txt")

bfs(wordgraph, wordgraph.getVertex('FOOL'))

traverse(wordgraph.getVertex('SAGE'))
```

广度优先搜索算法分析

- BFS算法主体是两个循环的嵌套

  while循环对每个顶点访问一次，所以是O(|V|)

  而嵌套在while中的for，由于每条边只有在其起始顶点u出队的时候才会被检查一次

  而每个顶点最多出队1次，所以边最多被检查1次，一共是O(|E|)

  综合起来BFS的时间复杂度为O(|V|+|E|)

广度优先搜索算法分析

- 词梯问题还包括两个部分算法

  建立BFS树之后，回溯顶点到起始顶点的过程，

  最多为O(|V|)

  创建单词关系图也需要时间，最多为$O(|V|^2)$