**1.使用BFS树进行无向图环检测：**

逐步构建BFS树，记录节点的父节点和结点层数，当发现该层节点中有节点存在指向同层或叔叔节点的边时，则说明有环。如果需要输出，则从该节点开始回溯，直到被指向的节点为止，就能找到整个环。

具体的，使用下面染色的代码，如果发现指向灰色节点出现（灰色节点包含了上述的两种情况，既有同层节点，又有叔叔节点），则说明有环。注意，子节点指向父节点的边不属于环，因此要记录父节点，以判断这种情况。另外，黑色的节点不会被再次访问。

|  |
| --- |
| bool hasCycle(int s){  for(int i = 1; i <= N; i++) {  nodesColor[i] = colorWhite;  }  queue<int> Q;  Q.push(s);  nodesColor[s] = colorGray;  while(!Q.empty()){  int v = Q.front();  Q.pop();  for(int index = 0; index < Graph[v].size(); index++){  int w = Graph[v][index];  if(nodesColor[w] == colorWhite){  Q.push(w);  nodesColor[w] = colorGray;  }else if(nodesColor[w] == colorGray){  return true;  }  }  nodesColor[v] = colorBlack;  }  return false;  } |

**2.无向图的最短路径树：**

一个结点的最短路径树是指，图中从该结点出发，到其他节点的最短路径就是从树中根节点出发，到其他节点的最短路径。

结论：BFS树一定是一个最短路径树，但最短路径树不一定能够通过BFS生成

例如这是一个图G：

BFS树：

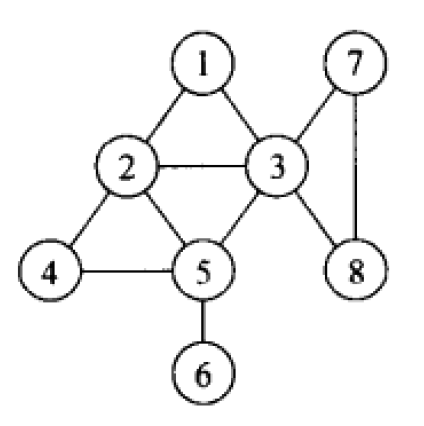
无法通过BFS生成的最短路径树：

即BFS算法有一定的贪心性质。

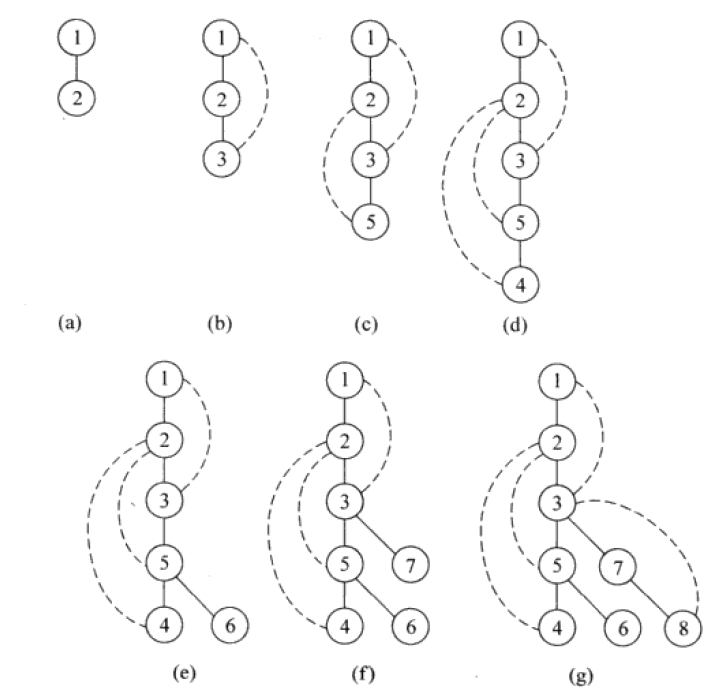
**3.DFS树**

使用DFS算法遍历图而产生的树

例如图G



的DFS遍历结果：



**图中不在DFS树上的边都是连接祖先和后代的边**

课后第6题（78页）

有向图反例：G不等于T

**4.无向图中DFS算法判断环**

需要记录父节点，一旦发现有指向祖先的边，就说明有环存在，仅判断的复杂度为θ(n)。注意，子节点指向父节点的边不属于环，因此要记录父节点，以判断这种情况。

**5.有向图中的BFS算法**

图中不在BFS树上的边包括无向图的两种，连接同一层顶点的边和连接子节点到叔叔节点的边（不分谁到谁，所以一共四种边）以及从子节点到父节点或祖先节点的边（一共两种）：

注意，不是从祖先节点或父节点到子节点的边，因为这样会将子节点拉上去。

即同层、相邻层向下的边（叔叔节点到子节点）、任意层向上的边（子节点到父节点、叔叔节点或祖先节点都可以）

**6.有向图中的DFS算法**

图中不在DFS树上的边包括无向图的一种，连接祖先节点和子节点的边（互联，两种）以及子节点到父节点的边，同层下，后遍历节点到先遍历节点的边（共两种）：

**7.有向图中的环检测**

BFS算法不适用，因为有一条连接祖先的边出现能推出有环，但有环不一定有一条连接祖先的边，例如：

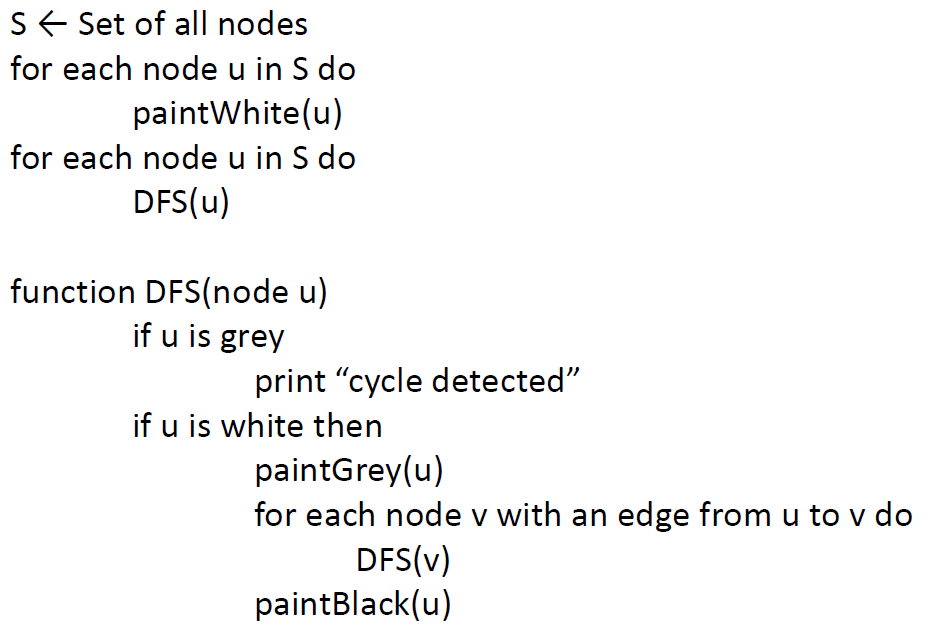
<https://stackoverflow.com/questions/2869647/why-dfs-and-not-bfs-for-finding-cycle-in-graphs>

中说得很好，BFS的记忆性不如DFS，DFS不仅能够记录哪些节点被访问过，还能记录到达这些节点的路径，而BFS不会记录这些信息。

上图中没有指向祖先节点的边，但是有环。

方法1：

使用DFS算法可以实现有向图中的环检测，如果有环则一定存在一条指向祖先节点的边（实现时就是，如果指向了灰色的节点，就说明有环）。



假设有一个n个节点的环，它们加入DFS树的顺序不一定，但是一定有一条指向第一个节点的边，才能构成环。

方法2：拓扑排序（逆序）：即得到的结果是拓扑排序的逆序，只有从后往前的边

