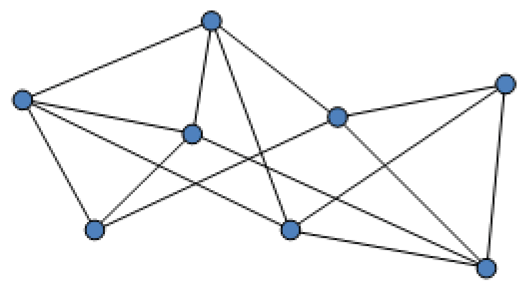
（伪代码算法有问题，visit mark true太晚会导致时间复杂度n^2）

图论（Graph Theory）是组合数学的一个分支，和其他数学分支，如群论、矩阵论、拓扑学有着密切关系。图是图论的主要研究对象。图是由若干给定的顶点及连接两顶点的边所构成的图形，这种图形通常用来描述某些事物之间的某种特定关系。顶点用于代表事物，连接两顶点的边则用于表示两个事物间具有这种关系。（from Wikipedia）



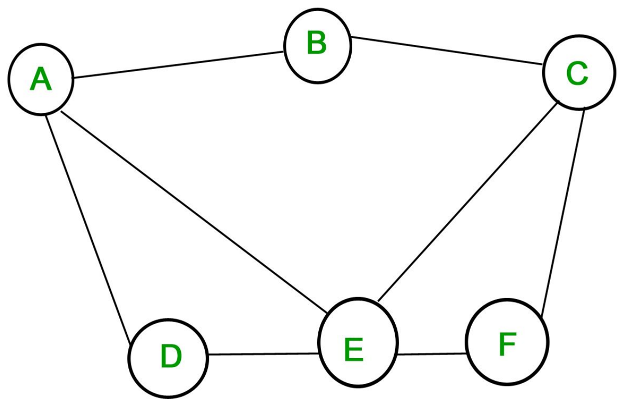
图分很多种类，比如有向图（directed graph）和无向图（undirected graph），有权图（weighted graph）和无权图（unweighted graph），连通图（connected graph）和非连通图（disconnected graph）；图有很多概念，常见的比如点（vertex），边（edge），面（face），权重（weight），入度（indegree），出度（outdegree），树（tree）；图有很多表达方式，比如邻接矩阵（adjacency matrix）和邻接链表（adjacency list）；图也有很多应用，比如数据库索引用到的B树，C++标准库里的map和set，分布式系统设计，等等。这些都是图的各个方面。

不同的算法基于不同的假设解决不同的问题。一般情况下一个算法假设越多，定义域越小，坏处是它能解决的问题越有限，好处是它变得更有针对性，因此通常具有更优的空间复杂度和时间复杂度。图论算法也一样，比如在图论里非常重要的Dijkstra算法，它的限制就是图里不能出现负环（即环上所有边权值之和为负）。当然Dijkstra算法有很多变种，这个以后再说。因此，图论可以分成很多子类别分别讨论，包括拓扑排序（Topological Sort），最短路（Shortest Path），生成树（Spanning Tree），欧拉回路（Eulerian Path），网络流（Flow），诸如此类，希望作者能在懒癌发作之前写完 :)

图论无论是在数学领域还是在计算机领域都占很大一部分，其中有趣的算法更是数不胜数，想入门的话强烈推荐David Eppstein教的CS 163 （此条0元）。这里先介绍一下两种最基本的图论算法——深度优先搜索（Depth First Search）和广度优先搜索（Breadth First Search），这两种算法都是遍历连通图的算法（连通图即任意两顶点都有路径相连，路径可以有一条或多条边组成），非连通图可拆分为多个连通子图。

1. 深度优先搜索（DFS）

   给定一个连通图G，从某一顶点v0出发，访问一个未访问过的相邻（共边）顶点v1，再从v1出发，访问一个未访问过的相邻顶点v2，以此类推，直到当前顶点没有任何未访问过的相邻顶点，这时不断返回上一个顶点，每回到一个顶点重复相同的操作访问未访问过的相邻顶点。简单来说就是一条路走到底，没路就往回走找别的岔路。



以上图为例，假设从顶点E出发，一种可能的访问顺序是E->A->D->B->C->F。

伪代码如下：

```
Set all nodes to "not visited";

s = new Stack();    ******* Change to use a stack

s.push(initial node);    ***** Push() stores a value in a stack

while ( s ≠ empty ) do
{
    x = s.pop();         ****** Pop() remove a value from the stack

    if ( x has not been visited )
    {
        visited[x] = true;         // Visit node x !

        for ( every edge (x, y)  /* we are using all edges ! */ )
            if ( y has not been visited )
                s.push(y);       ***** Use push() !
    }
}
```

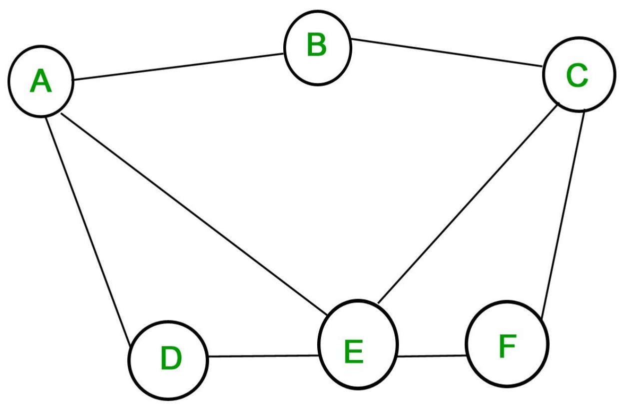
（from Emory）

时间复杂度（time complexity）：O(|V| + |E|)，其中|V|为顶点数，|E|为边数

空间复杂度（space complexity）：O(max(d))，其中max(d)为最大深度，即根节点（v0）到任意叶子节点（叶子节点是DFS访问时其相邻顶点均已被访问的顶点）的最长距离

2. 广度优先搜索（BFS）

   给定一个连通图G，从某一顶点v0出发，依次访问所有未访问过的相邻顶点，再依次访问刚刚访问的一层顶点中每个顶点的未访问过的相邻顶点，以此类推，直到遍历完所有节点。简单来说有点像病毒扩散。



以上图为例，假设从顶点E出发，一种可能的访问顺序是E->D->A->C->F->B。

伪代码如下：

```
Set all nodes to "not visited";

q = new Queue();

q.enqueue(initial node);

while ( q ≠ empty ) do
{
    x = q.dequeue();

    if ( x has not been visited )
    {
         visited[x] = true;         // Visit node x !

         for ( every edge (x, y)  /* we are using all edges ! */ )
             if ( y has not been visited )
                 q.enqueue(y);       // Use the edge (x,y) !!!
    }
}
```

（from Emory）

时间复杂度：O(|V| + |E|)，其中|V|为顶点数，|E|为边数

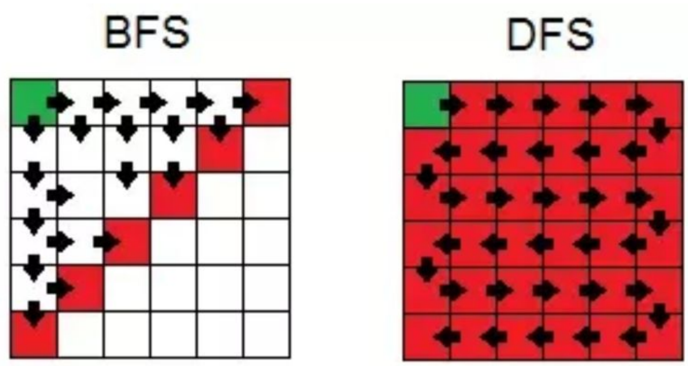
空间复杂度：O(|V|) ≈ O(avg(b)avg(d))，其中avg(b)为每个顶点平均相邻顶点数量，avg(d)为平均深度

两种算法功能一致，Big-O时间复杂度又一样，那它们有什么区别呢？以及如何判断什么时候用哪种呢？首先，所有的递归（recursion）都可以改写成迭代（iteration），反之亦然。上面两段伪代码用迭代实现BFS和DFS，其中唯一的区别在于BFS使用了队列（queue），而DFS使用了栈（stack）。从算法上讲，对于一个较为平衡的树（balanced tree，即从根节点出发到各个叶子节点的距离相近的图），例如完全二叉树（complete binary tree，每个非叶子节点都有两个子节点的树），BFS最差的情况（worst case）是在树的最后一层同时记录一半的节点，即空间复杂度为O(|V|)，而DFS最差的情况是同时记录从根节点到叶子节点最长路径上的所有节点，即空间复杂度为O(max(d))；从数据结构上来讲，当一层最多的节点数和一条路径上最多的节点数大致相当时，则根据不同的编程语言实现栈和队列的方法来选DFS或BFS，可以考虑的点包括内存限制，是否静态分配内存，操作成本，等等。因此同样使用迭代的情况下，主要根据图的形状（矮而宽还是高而窄）来选择DFS和BFS的使用。

如果用递归实现BFS和DFS，则两者的区别在于：BFS在遍历相邻顶点的循环结束之后递归所有相邻顶点，而DFS则是在遍历相邻顶点的循环内开头递归当前相邻顶点。不难看出

BFS即使用递归实现，它仍然需要一个类似数组（Array）的额外存储空间来记录所有相邻顶点，因此同样使用递归实现的情况下，BFS的空间复杂度比DFS高。值得一提的是，同样的步骤实现起来通常迭代比递归更有效率。

这时候问题就来了，目前看来BFS和DFS的最大区别就在于BFS在多数情况下空间复杂度比DFS高，那是不是任何情况都用DFS呢？大多数情况是的。但如果我们要找最短路（包含最少边数的路径），或者我们要在图中找到某个目标顶点而不必遍历全图时，一旦目标顶点离根节点很近，并且图中有多条不包含目标顶点还很长的分支，这个时候BFS就更加适合，DFS则容易在长分支里浪费时间。尽管描述很复杂，但实际上有很多这样的情况，比如网络爬虫在爬取相关网页时，重要的网页往往不会藏在很多层链接里。



除此之外还有一种结合BFS和DFS优势的遍历方法，因为它的中文名有点奇怪，所以还是用它的英文名Iterative Deepening Search（IDS）吧。这个算法和BFS目的相同，但Big-O空间复杂度和DFS相同，它本质上是一个具有深度限制的DFS，当所有深度限制内的顶点遍历完毕之后没找到目标顶点，就放宽深度限制继续DFS。总结一下就是IDS是一个优化过空间复杂度的BFS。

以上内容仅代表作者个人看法，如有错误欢迎指正。欢迎有兴趣的朋友来一起讨论算法！

相关材料：
https://en.wikipedia.org/wiki/Breadth-first_search
http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/bfs.html
https://en.wikipedia.org/wiki/Depth-first_search
http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/dfs.html