图的优先遍历

一、深度优先搜索算法DFS（depthFirstSearch）

1、算法的思想

从图中的某一个顶点x出发，访问x，然后遍历任何一个与x相邻的未被访问的顶点y，再遍历任何一个与y相邻的未被访问的顶点z……依次类推，直到到达一个所有邻接点都被访问的顶点为止；然后，依次回退到尚有邻接点未被访问过的顶点，重复上述过程，直到图中的全部顶点都被访问过为止。

2、算法实现的思想

深度优先遍历背后基于堆栈，有两种方式：

第一种是在程序中显示构造堆栈，利用压栈出栈操作实现；（非递归）

第二种是利用递归函数调用，基于递归程序栈实现　　（递归）

3、代码实现

1）访问初始结点 v，并标记结点 v为已访问。

2）查找结点 v的第一个邻接结点 w。

3）若 w存在，则继续执行 4，如果 w不存在，则回到第 1步，将从 v的下一个结点继续。

4）若 w未被访问，对 w进行深度优先遍历递归（即把 w当做另一个 v，然后进行步骤 123）。

5）查找结点 v的 w邻接结点的下一个邻接结点，转到步骤 3。

```
//对dfs 进行一个重载, 遍历我们所有的结点，并进行 dfs
public void dfs() {
        isVisited = new boolean[vertexList.size()];
        //遍历所有的结点，进行dfs[回溯]
        for(int i = 0; i < getNumOfVertex(); i++) {
                if(!isVisited[i]) {
                        dfs(isVisited, i);
```

```
                }
            }
        }

    //深度优先遍历算法
        //i 第一次就是 0
        private void dfs(boolean[] isVisited, int i) {
            //首先我们访问该结点,输出
            System.out.print(getValueByIndex(i) + "->");
            //将结点设置为已经访问
            isVisited[i] = true;
            //查找结点i的第一个邻接结点w
            int w = getFirstNeighbor(i);
            while(w != -1) {//说明有
                if(!isVisited[w]) {
                    dfs(isVisited, w);
                }
                //如果w结点已经被访问过
                w = getNextNeighbor(i, w);
            }
        }
```

二、广度优先搜索算法BFS（BreadFirstSearch)

1、算法的思想

从图中的某一个顶点x出发，访问x，然后访问与x所相邻的所有未被访问的顶点x1、x2……xn，接着再依次访问与x1、x2……xn相邻的未被访问的所有顶点。依次类推，直至图中的每个顶点都被访问。

2、算法实现的思想

广度优先遍历背后基于**队列**，下面介绍一下具体实现的方法：

1. 访问初始结点 v并标记结点 v为已访问。
2. 结点 v入队列
3. 当队列非空时，继续执行，否则算法结束。
4. 出队列，取得队头结点 u。
5. 查找结点 u的第一个邻接结点 w。

6. 若结点 u的邻接结点 w不存在，则转到步骤 3；否则循环执行以下三个步骤

i. 若结点 w尚未被访问，则访问结点 w并标记为已访问。

ii. 结点 w入队列

iii. 查找结点 u的继 w邻接结点后的下一个邻接结点 w，转到步骤 6。

3、代码实现

```
//遍历所有的结点，都进行广度优先搜索
    public void bfs() {
        isVisited = new boolean[vertexList.size()];
        for(int i = 0; i < getNumOfVertex(); i++) {
            if(!isVisited[i]) {
                bfs(isVisited, i);
            }
        }
    }


    //对一个结点进行广度优先遍历的方法
    private void bfs(boolean[] isVisited, int i) {
        int u ; // 表示队列的头结点对应下标
        int w ; // 邻接结点w
        //队列，记录结点访问的顺序
        LinkedList queue = new LinkedList();
        //访问结点，输出结点信息
        System.out.print(getValueByIndex(i) + "=>");
        //标记为已访问
        isVisited[i] = true;
        //将结点加入队列
        queue.addLast(i);

        while( !queue.isEmpty()) {
            //取出队列的头结点下标
            u = (Integer)queue.removeFirst();
            //得到第一个邻接结点的下标 w
            w = getFirstNeighbor(u);
            while(w != -1) {//找到
                //是否访问过
                if(!isVisited[w]) {
```

```
                    System.out.print(getValueByIndex(w) + "=>");
                    //标记已经访问
                    isVisited[w] = true;
                    //入队
                    queue.addLast(w);
                }
                //以u为前驱点，找w后面的下一个邻结点
                w = getNextNeighbor(u, w); //体现出我们的广度优先
            }
        }

}
```