说明

图中实现最基本的操作之一就是搜索从一个指定顶点可以到达哪些顶点，比如从武汉出发的高铁可以到达哪些城市，一些城市可以直达高铁模拟图，要从某个城市（顶点）开始，沿着铁轨（边）移动到其他城市（顶点），有两种方法可以用来搜索图：深度优先搜索（DFS）会到达所有连通的顶点，深度优先搜索通过递归来实现，也可用栈，而广度优先搜索通过队列来实现，不同的实现机制导致不同的搜索方式

深度优先搜索（DFS）

思路

深度优先搜索（Depth First Search），简称DFS，该方法主要思想是：

- 从某一个顶点开始，选择一条没有到达过的顶点（布尔数组中对应的值为false）
- 标记刚选择的顶点为“访问过”（布尔数组中对应的值设置为true）
- 来到某个顶点，如果该顶点周围的顶点都访问过了，返回到上个顶点
- 当回退后的顶点依然是上述情况，继续返回

代码实现 - 递归

```java
package 图;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

/**
 * 图的搜索 - 深度优先搜索（DFS） - 递归
 */
public class DepthFirstSearch {
    // 用来标记已经访问过的顶点，保证每个顶点值访问一次
    private boolean[] marked;
    // s为搜索的起点
    public DepthFirstSearch(UndiGraph<?> graph, int s) {
        marked = new boolean[graph.vertexNum()];
        dfs(graph, s);
    }

    private void dfs(UndiGraph<?> graph, int v) {
        // 将刚访问到的顶点设置标志
        marked[v] = true;
        // 打印刚访问的顶点，可换成其他操作
        System.out.println(v);
        // 从v的所有邻接点中选择一个没有被访问过的顶点
        for (int w : graph.adj(v)) {
            if (!marked[w]) {
                dfs(graph, w);
            }
        }
    }

    public static void main(String[] args) {
        List<String> vertexInfo = Arrays.asList("v0", "v2", "v3", "v4", "v5");
```

```
        int[][] edges = {{0, 1}, {0, 2}, {0, 3},
                {1, 3}, {1, 4},
                {2, 4}};

        UndiGraph<String> graph = new UndiGraph<>(vertexInfo, edges);
        DepthFirstSearch search = new DepthFirstSearch(graph, 0);

    }
}
```

广度优先搜索（BFS）

思路

这个算法的思想大体是：

- 从起点开始，标记之并加入队列。
- 起点出列，其所有未被标记的邻接点在被标记后，入列。
- 队列头的元素出列，将该元素的所有未被标记的邻接点标记后，入列

代码实现

```
package 图;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

/**
 * 图的搜索 - 广度优先搜索（BFS）
 */
public class BreadthFirstSearch {
    // 用来标记已经访问过的顶点，保证每个顶点值访问一次
    private boolean[] marked;
    // 起点
    private final int s;
    // 到该顶点的路径上的最后一条边
    private int[] edgeTo;


    public BreadthFirstSearch(UndiGraph<?> graph, int s) {
        this.s = s;
        marked = new boolean[graph.vertexNum()];
        edgeTo = new int[graph.vertexNum()];
        bfs(graph, s);
    }

    public void bfs(UndiGraph<?> graph, int s) {
        marked[s] = true;
        // offer入列，poll出列
        Queue<Integer> queue = new LinkedList<>();
        queue.offer(s);
        while (!queue.isEmpty()) {
```

```
            int v = queue.poll();
            System.out.print(v+" ");
            for (int w: graph.adj(v)) {
                if (!marked[w]) {
                    edgeTo[w] = v;
                    marked[w] = true;
                    queue.offer(w);
                }
            }
        }
    }

    public boolean hasPathTo(int v) {
        return marked[v];
    }

    public Iterable<Integer> pathTo(int v) {
        if (hasPathTo(v)) {
            LinkedList<Integer> path = new LinkedList<>();
            for (int i = v; i != s; i = edgeTo[i]) {
                path.push(i);
            }
            // 最后将根结点压入
            path.push(s);
            return path;
        }
        // 到v不存在路径，就返回null
        return null;
    }

    public static void main(String[] args) {
        List<String> vertexInfo = Arrays.asList("v0", "v1", "v2", "v3", "v4", "v5");
        int[][] edges = {{3, 5},{0, 2}, {0, 1}, {0, 5},
                {1, 2}, {3, 4}, {2, 3}, {2, 4}};

        UndiGraph<String> graph = new UndiGraph<>(vertexInfo, edges);
        BreadthFirstSearch search = new BreadthFirstSearch(graph, 0);
    }
}
```