小码哥教育 SEEMYGO

遍历

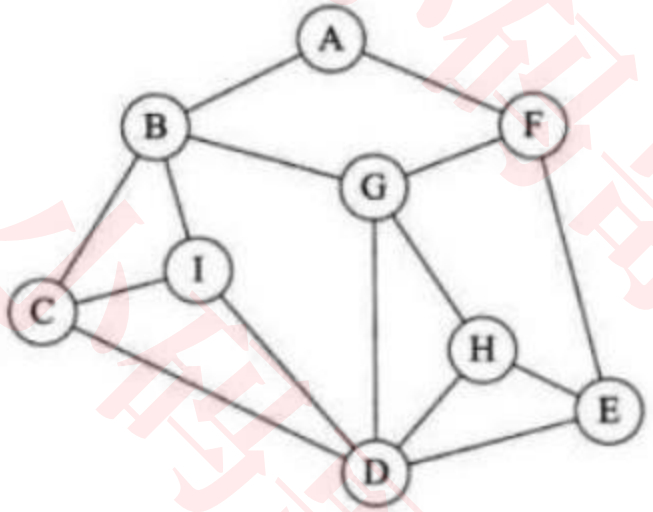
- 图的遍历
  - 从图中某一顶点出发访问图中其余顶点，且每一个顶点仅被访问一次

- 图有2种常见的遍历方式（有向图、无向图都适用）
  - 广度优先搜索（Breadth First Search，BFS），又称为宽度优先搜索、横向优先搜索

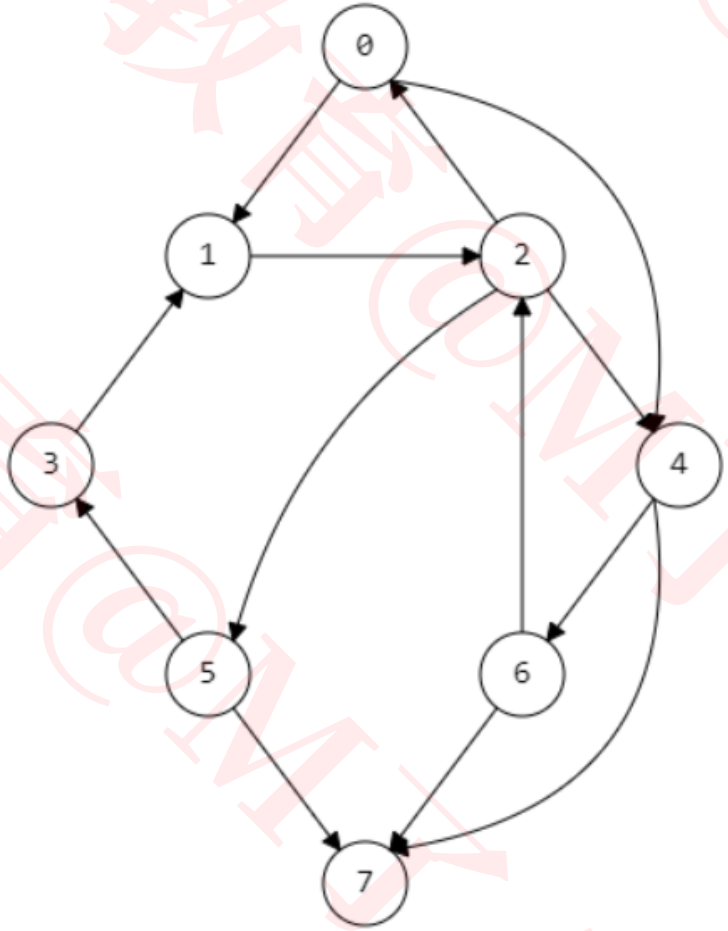
  - 深度优先搜索（Depth First Search，DFS）
    - ✓ 发明“深度优先搜索”算法的2位科学家在1986年共同获得计算机领域的最高奖：图灵奖

广度优先搜索（Breadth First Search）

- 之前所学的二叉树层序遍历就是一种广度优先搜索



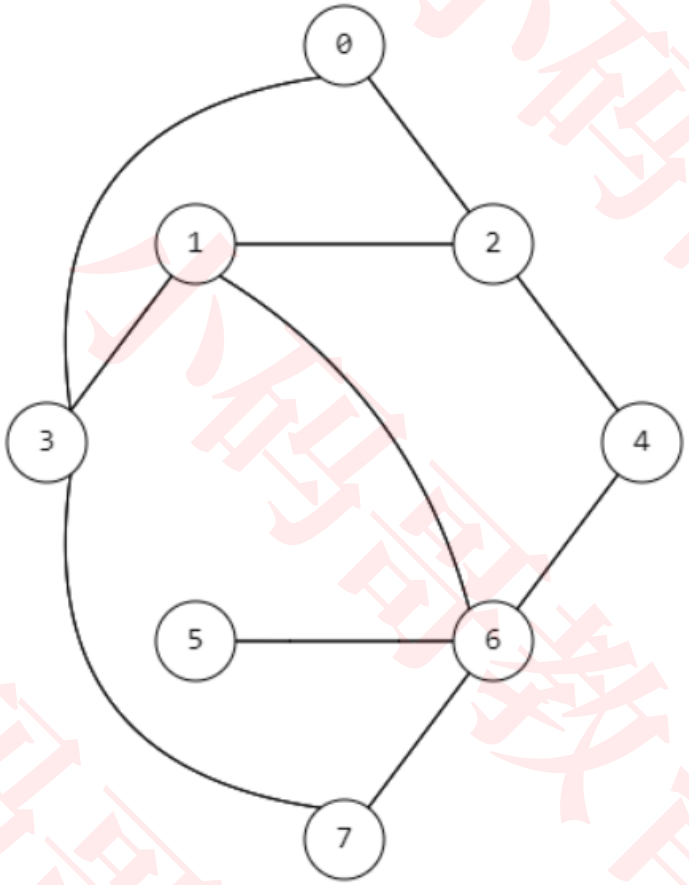
第1层	A
第2层	B、F
第3层	C、I、G、E
第4层	D、H



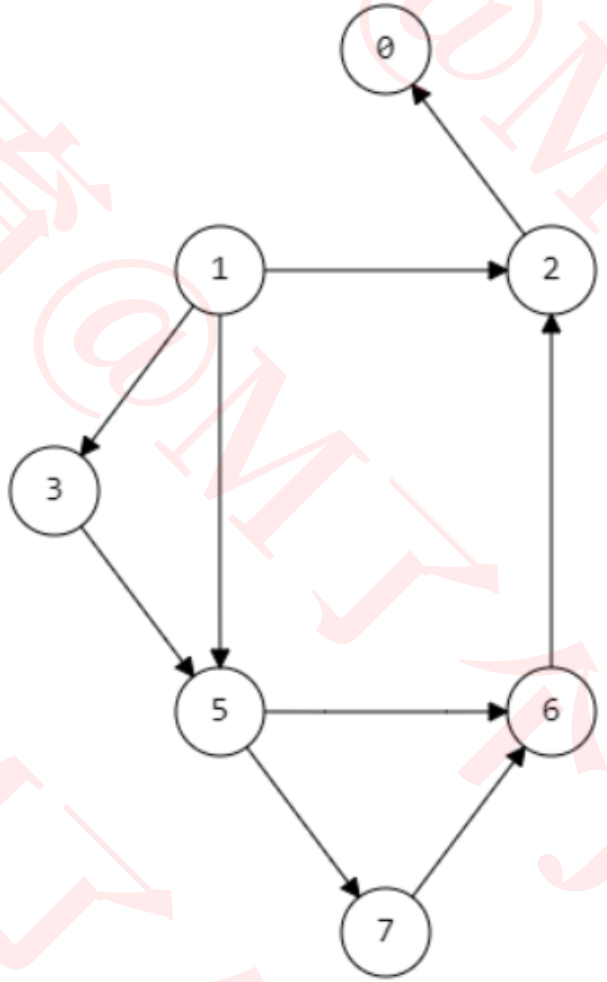
第1层	0
第2层	1、4
第3层	2、6、7
第4层	5
第5层	3

小码哥教育 SEEMYGO

广度优先搜索

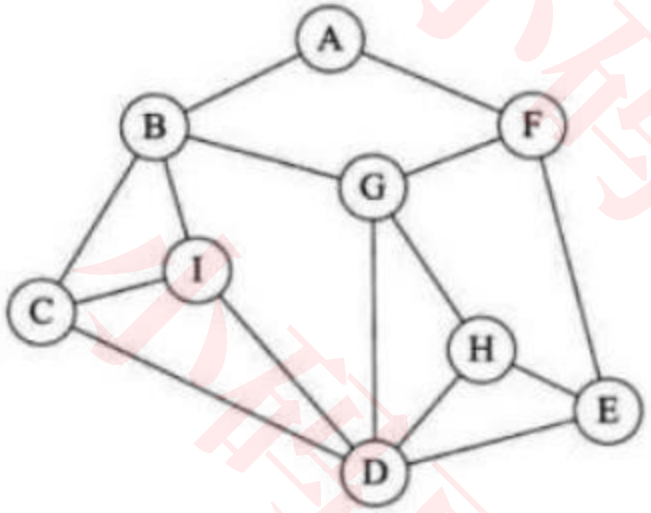


第1层	0
第2层	3、2
第3层	7、1、4
第4层	6
第5层	5



第1层	5
第2层	7、6
第3层	2
第4层	0

广度优先搜索 – 思路



← A	←
← B、F	←
← F、C、I、G	←
← C、I、G、E	←
← I、G、E、D	←
← G、E、D	←
← E、D、H	←
← D、H	←
← H	←
←	←

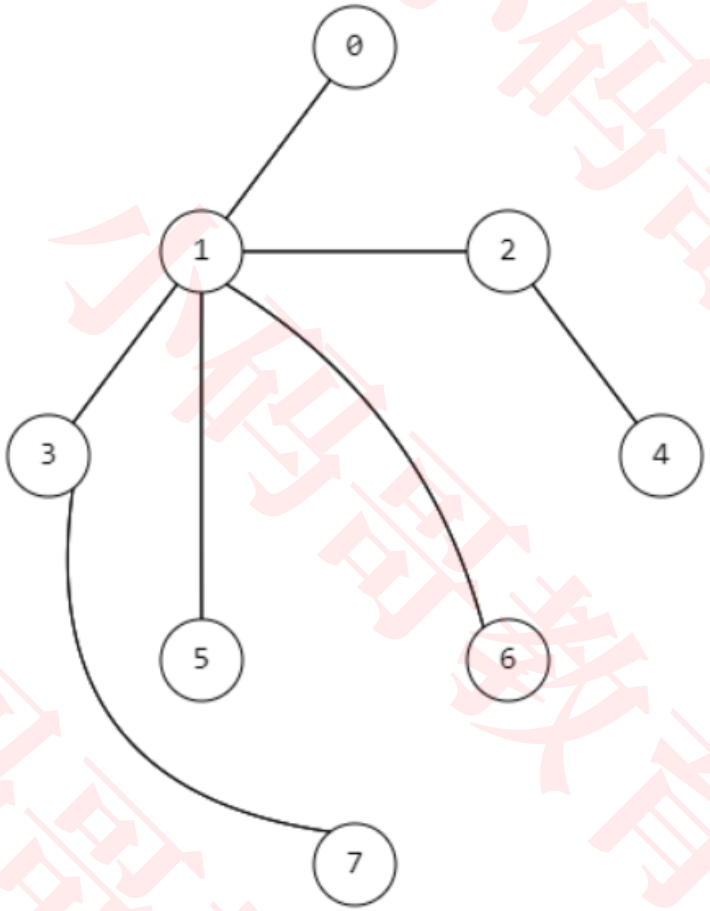
小码哥教育 SEEMYGO

广度优先搜索 - 实现

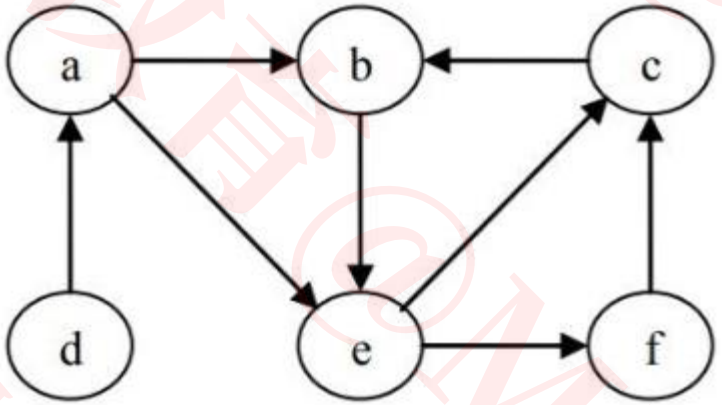
```
private void bfs(Vertex<V, E> beginVertex) {
    Set<Vertex<V, E>> visitedVertices = new HashSet<>();
    Queue<Vertex<V, E>> queue = new LinkedList<>();
    queue.offer(beginVertex);
    visitedVertices.add(beginVertex);
    while (!queue.isEmpty()) {
        Vertex<V, E> vertex = queue.poll();
        System.out.println(vertex.value);

        for (Edge<V, E> edge : vertex.outEdges) {
            if (visitedVertices.contains(edge.to)) continue;
            queue.offer(edge.to);
            visitedVertices.add(edge.to);
        }
    }
}
```

深度优先搜索（Depth First Search）

- 之前所学的二叉树前序遍历就是一种深度优先搜索



1、3、7、5、6、2、4、0



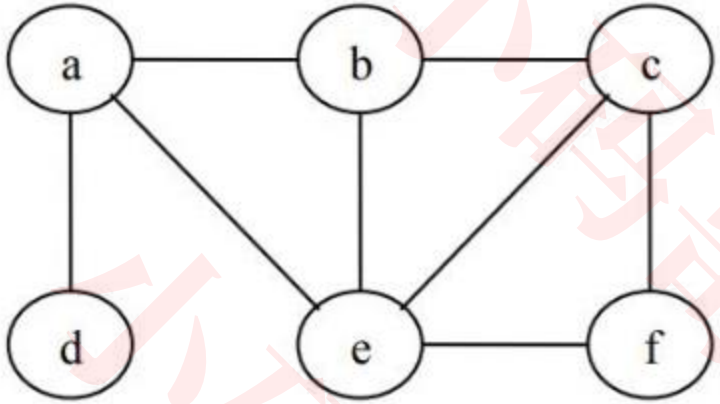
a、e、f、c、b

a、e、c、b、f

a、b、e、f、c

a、b、e、c、f

深度优先搜索



e、f、c、b、a、d

e、c、f、b、a、d

e、c、b、a、d、f

e、b、c、f、a、d

e、a、b、c、f、d

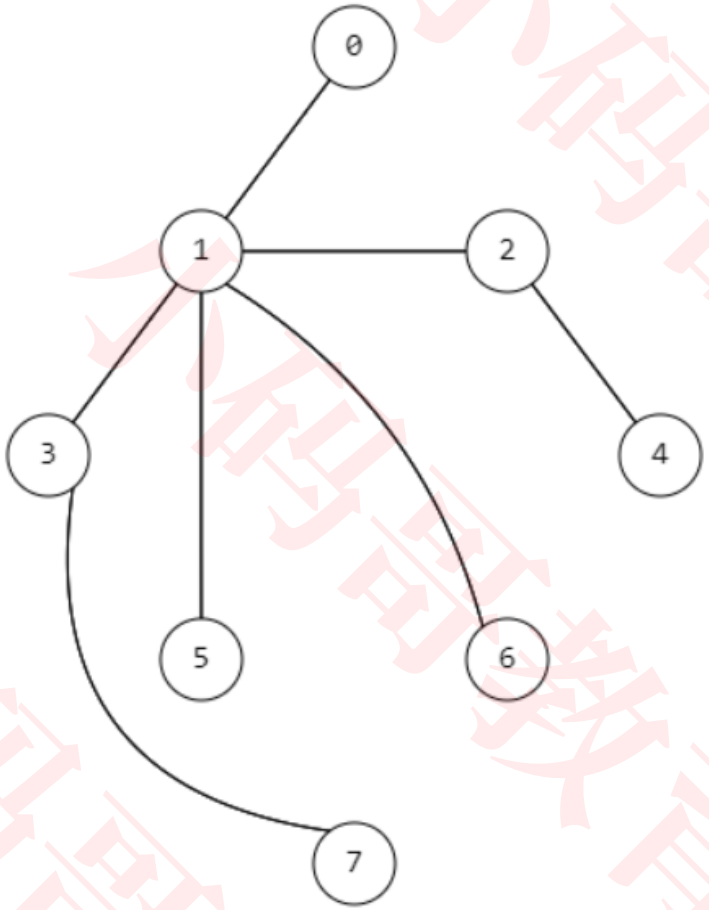
e、a、d、b、c、f

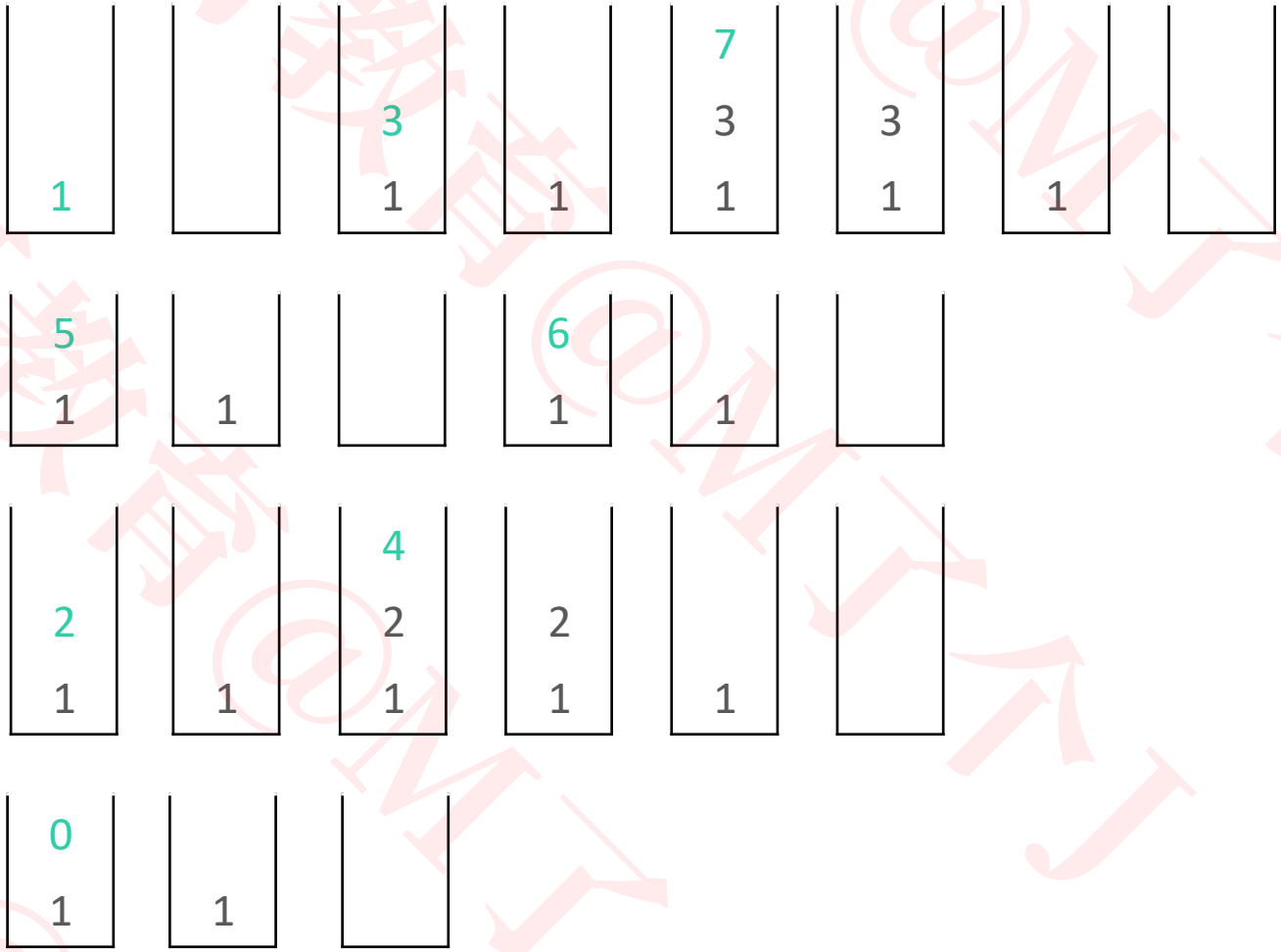
小码哥教育 SEEMYGO

深度优先搜索 - 递归实现

```
private void dfs(Vertex<V, E> vertex, Set<Vertex<V, E>> visitedVertices) {
    System.out.println(vertex.value);
    visitedVertices.add(vertex);

    for (Edge<V, E> edge : vertex.outEdges) {
        if (visitedVertices.contains(edge.to)) continue;
        dfs(edge.to, visitedVertices);
    }
}
```

深度优先搜索 – 非递归思路





小码哥教育 SEEMYGO

深度优先搜索 - 非递归实现

```
private void dfs(Vertex<V, E> beginVertex) {
    Set<Vertex<V, E>> visitedVertices = new HashSet<>();
    Stack<Vertex<V, E>> stack = new Stack<>();
    stack.push(beginVertex);
    visitedVertices.add(beginVertex);
    System.out.println(beginVertex.value);

    while (!stack.isEmpty()) {
        Vertex<V, E> vertex = stack.pop();
        for (Edge<V, E> edge : vertex.outEdges) {
            if (visitedVertices.contains(edge.to)) continue;
            stack.push(edge.from);
            stack.push(edge.to);
            visitedVertices.add(edge.to);
            System.out.println(edge.to.value);
            break;
        }
    }
}
```