# 1.开篇

完整学习图论领域的经典算法，培养的图论建模能力。

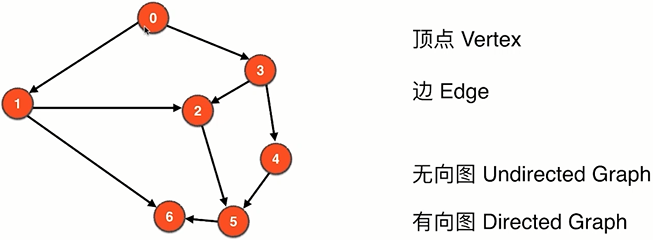
## 1.1 图论到底有什么用？

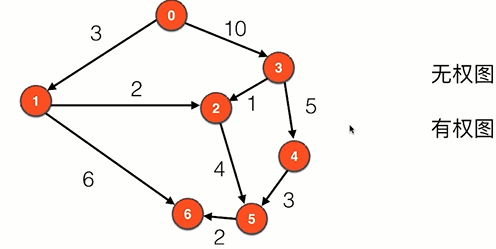
## 1.2 课程编程环境的搭建

# 2.图的基本表示

千里之行，驶于足下。解决任何有一个图论算法问题，首先需要用基本的数据结构来表示图——邻接矩阵和邻接表。

## 2.1 图的分类





顶点： Vertex

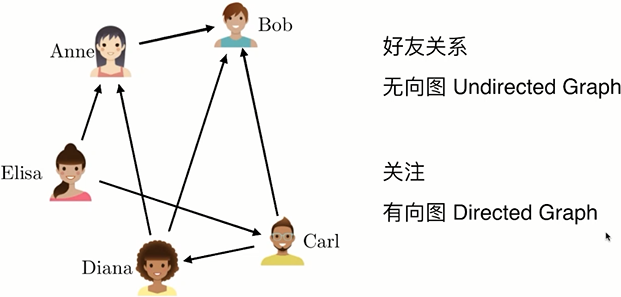
边：Edge

无向图：Undirected Graph

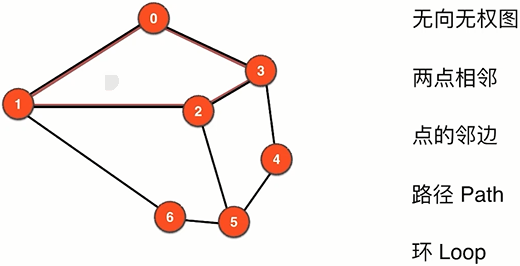
有向图：Directed Graph

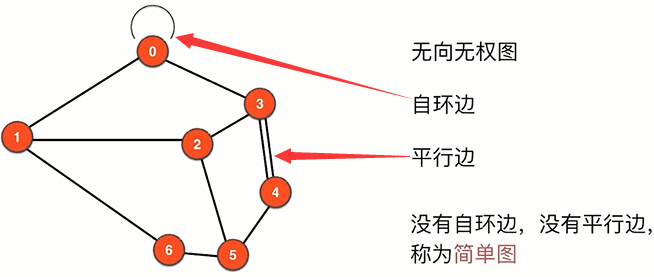
|  |  |  |
| --- | --- | --- |
| 图的分类 | 无向 | 有向 |
| 无权 | **无向无权图** | 有向无权图 |
| 有权 | 无向有权图 | 有向有权图 |

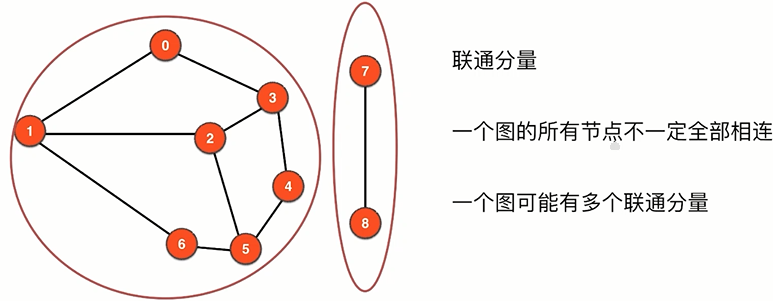


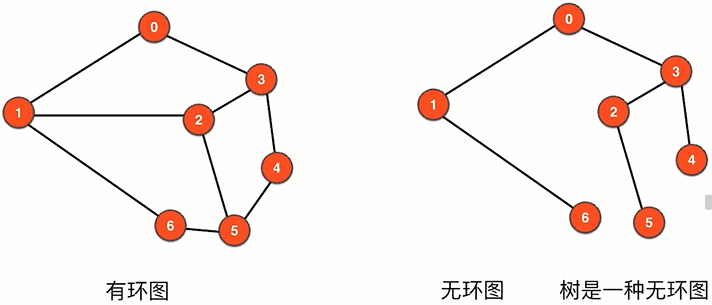


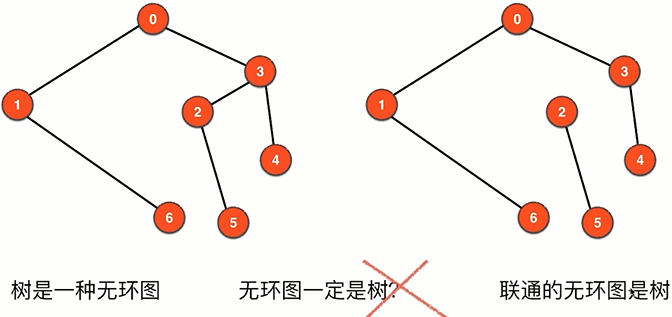
## 2.2 图的基本概念

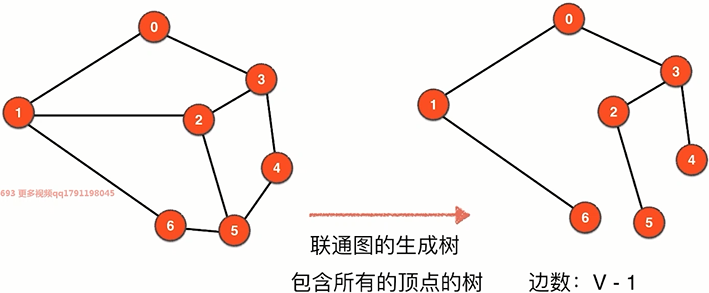


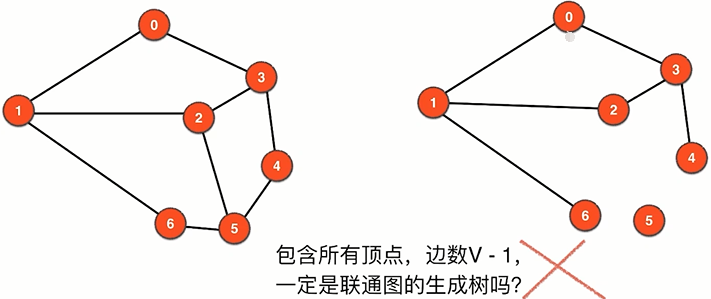


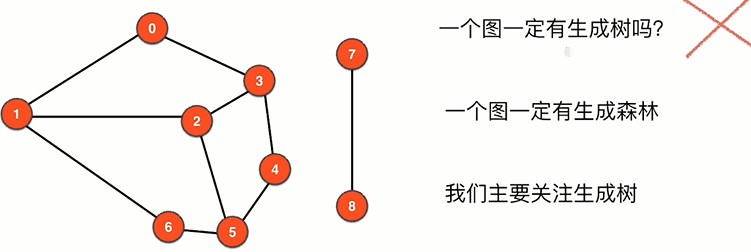


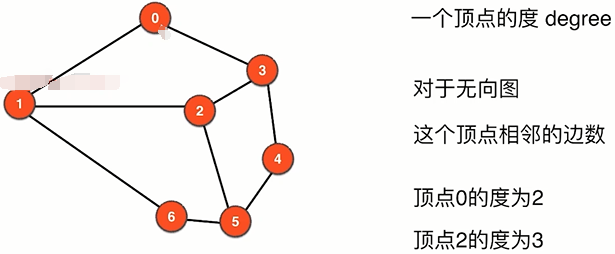




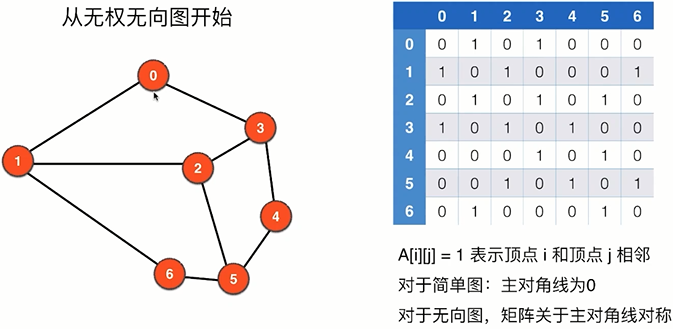


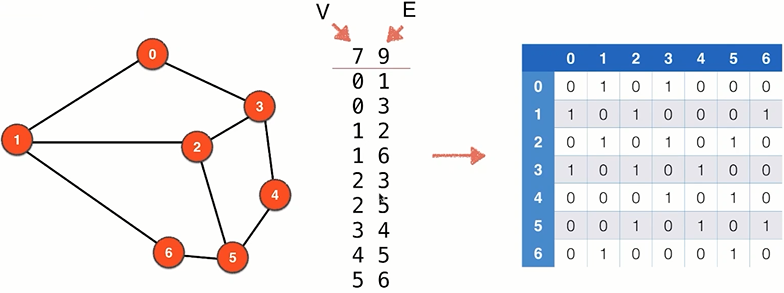






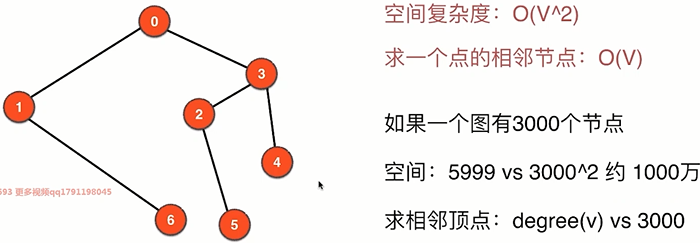
## 2.3 图的基本表示：邻接矩阵

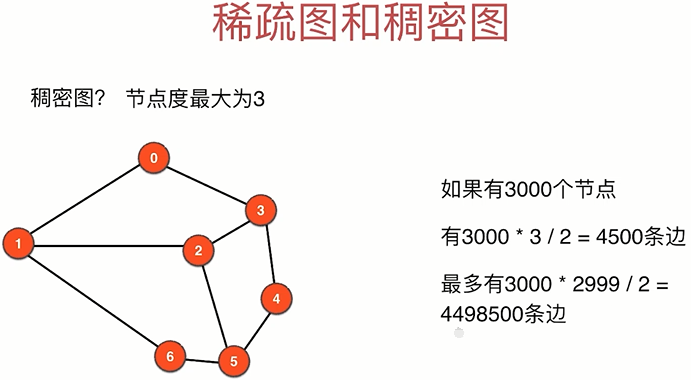
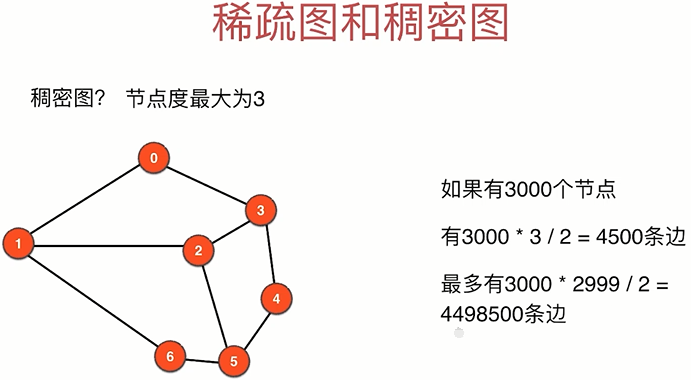


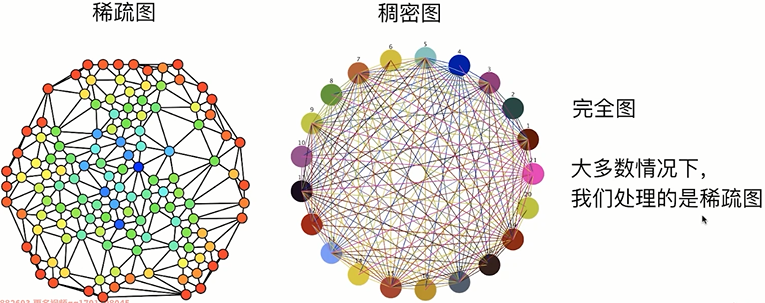


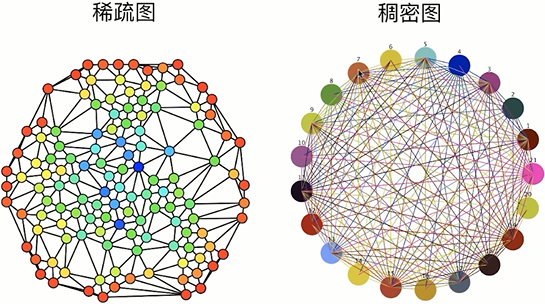
从文件读取图的基本信息，转化成图在计算机中的基本表示。







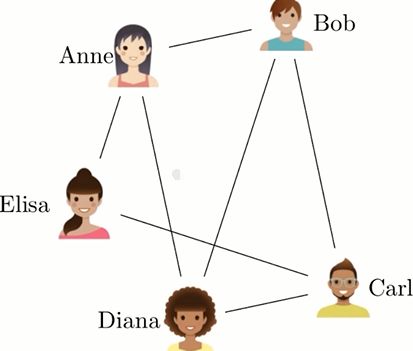




区分稀疏图与稠密图，没有固定的标准。



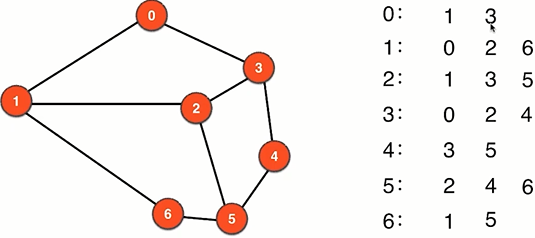
地铁线路图是稀疏图



社交网络图是稀疏图

## 2.4 邻接矩阵的实现

## 2.5 图的基本表示：邻接表



邻接表中，“表”指的是链表，图中每个顶点都有一个链表，链表中记录与这个顶点邻接的顶点。

每个顶点的邻接表的大小只和这个顶点的度相关，与整个图有多少顶点、多少边无关。

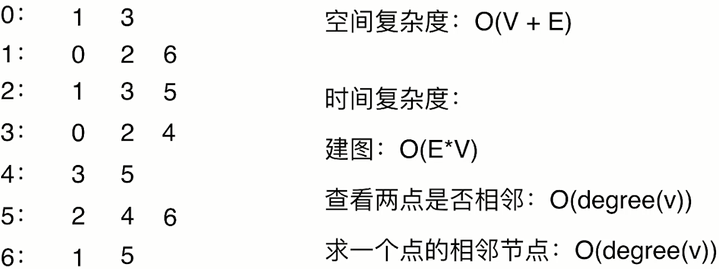
求一个顶点的邻接点，只与这个顶点的邻接表有关。

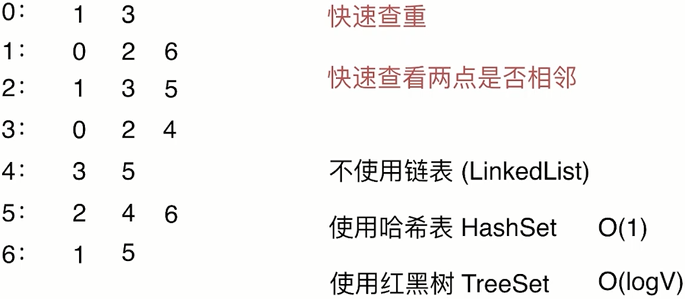
## 2.6 邻接表的实现

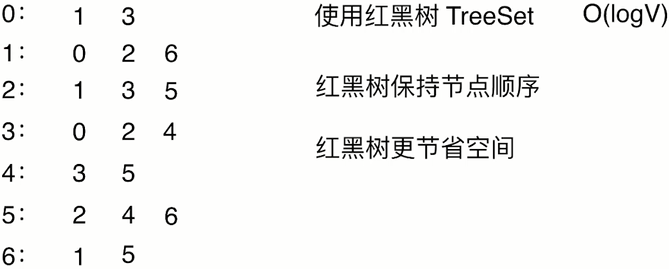
package com.sj.dsa;  
  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.LinkedList;  
import java.util.Scanner;  
  
*/\*\*  
 \* 邻接矩阵  
 \** ***@author*** *ShiJie  
 \** ***@since*** *2020-04-21  
 \*/*public class AdjList {  
 private int V;  
 private int E;  
 //private int[][] adj;  
 private LinkedList<Integer>[] adj;  
  
 public AdjList(String filename) {  
 File file = new File(filename);  
 try (Scanner scanner = new Scanner(file)) {  
 V = scanner.nextInt();  
 if (V < 0) throw new IllegalArgumentException("V must be non-negative");  
 adj = new LinkedList[V];  
 for (int i = 0; i < V; i++) {  
 adj[i] = new LinkedList<>();  
 }  
  
 E = scanner.nextInt();  
 if (E < 0) throw new IllegalArgumentException("E must be non-negative");  
 for (int i = 0; i < E; i++) {  
 int a = scanner.nextInt();  
 validateVertex(a);  
 int b = scanner.nextInt();  
 validateVertex(b);  
  
 // 简单图：不含自环边和平行边  
 if (a == b) throw new IllegalArgumentException("Self Loop Edge is Detected!");  
 if (adj[a].contains(b)) throw new IllegalArgumentException("Parallel Edges are Detected!");  
  
 adj[a].add(b);  
 adj[b].add(a);  
 }  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void validateVertex(int v) {  
 if (v < 0 || v >= V) {  
 throw new IllegalArgumentException("vertex " + v + " is invalid");  
 }  
 }  
  
 public int V() {  
 return V;  
 }  
  
 public int E() {  
 return E;  
 }  
  
 public boolean hasEdge(int v, int w) {  
 validateVertex(v);  
 validateVertex(w);  
 return adj[v].contains(w);  
 }  
  
 // 求一个顶点的邻点  
 public LinkedList<Integer> adj(int v) {  
 validateVertex(v);  
 return adj[v];  
 }  
  
 // 求一个顶点的度  
 public int degree(int v) {  
 return adj(v).size();  
 }  
  
 @Override  
 public String toString() {  
 StringBuilder sb = new StringBuilder();  
 sb.append(String.*format*("V = %d, E = %d\n", V, E));  
 for (int i = 0; i < V; i++) {  
 sb.append(String.*format*("%d: ", i));  
 for (int w: adj[i]) {  
 sb.append(String.*format*("%d ", w));  
 }  
 sb.append('\n');  
 }  
 return sb.toString();  
 }  
  
 public static void main(String[] args) {  
 AdjList adjMatrix = new AdjList("g.txt");  
 System.*out*.println(adjMatrix);  
 }  
  
}

## 2.7 邻接表的问题分析

复杂度分析







O(1) < O(logn) < O(n)

如果n = 100 万，1 < 20 < 100万

如果n = 10亿，1 < 30 < 10亿

哈希表保存元素是无序的，红黑树保存元素是有序的。

## 2.8 实现邻接表的改进

使用红黑树表示邻接表

## 2.9 图的基本表示的比较



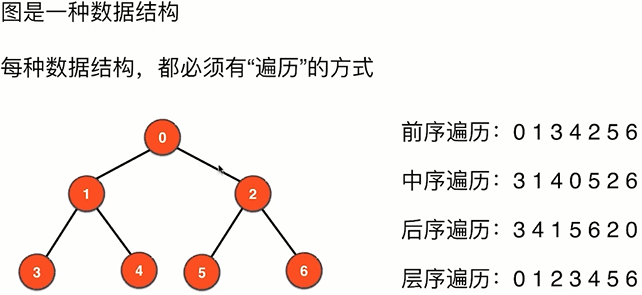
基于TreeSet的邻接表，元素有序；

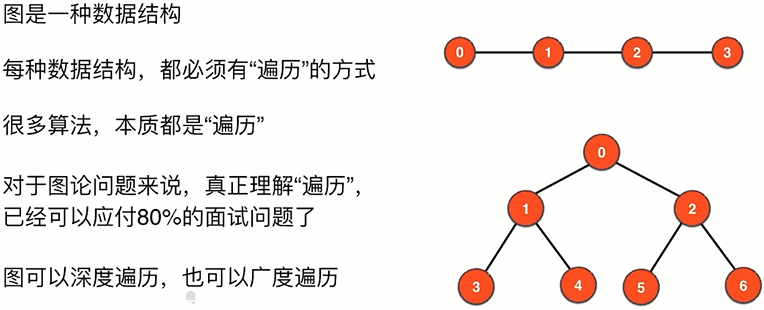
基于HashSet的邻接表，元素无序。

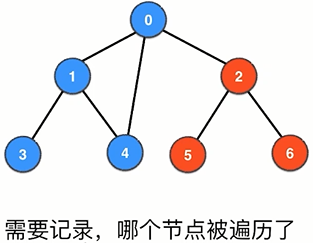
# 3.图的深度优先遍历

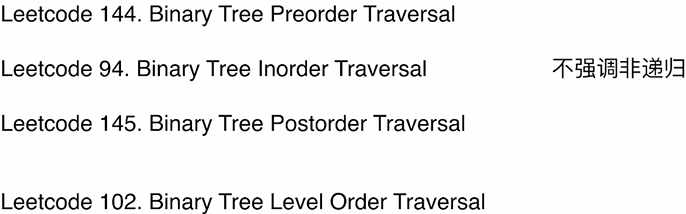
任何一种数据结构，都需要进行遍历。图也不例外。通过深入理解树的遍历，掌握图的遍历并不难，从树的深度优先遍历出发，到图的深度优先遍历，更加深刻地理解递归。

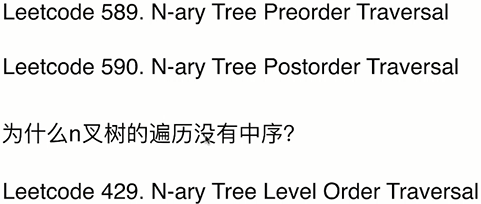
## 3.1 数据结构遍历的意义



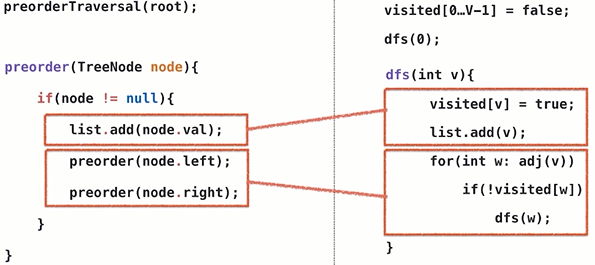




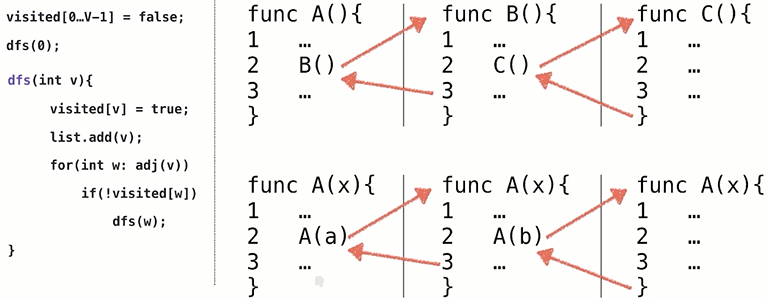


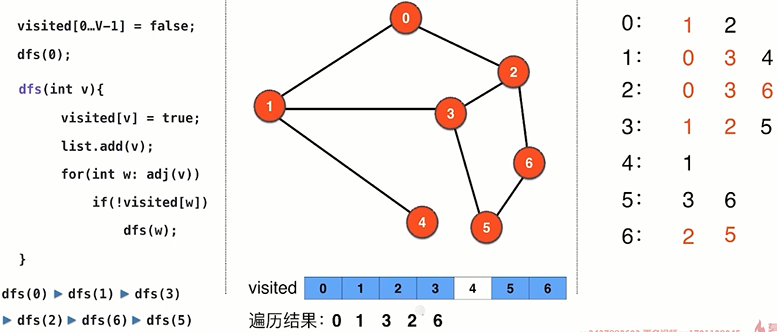


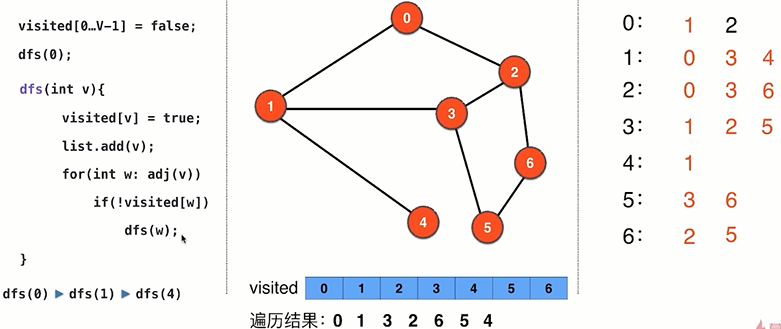
## 3.2 从树的深度优先遍历，到图的深度优先遍历



## 3.3 DFS逻辑的微观解读





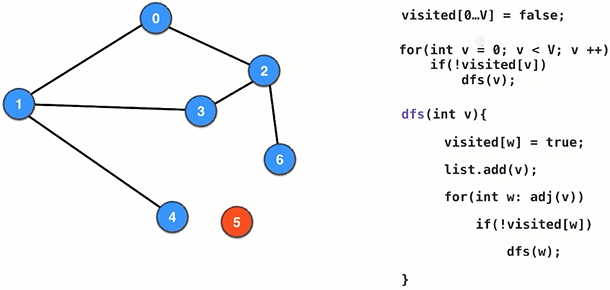


## 3.4 实现图的深度优先遍历

package com.sj.dsa;  
  
import java.util.ArrayList;  
  
*/\*\*  
 \** ***@author*** *ShiJie  
 \** ***@since*** *2020-04-21  
 \*/*public class GraphDFS {  
 private Graph graph;  
 private boolean[] visited;  
 private ArrayList<Integer> order = new ArrayList<>();  
  
 public GraphDFS(Graph graph) {  
 this.graph = graph;  
 visited = new boolean[graph.V()];  
 dfs(0);  
 }  
  
 private void dfs(int v) {  
 visited[v] = true;  
 order.add(v);  
 for (int w: graph.adj(v)) {  
 if (!visited[w]) {  
 dfs(w);  
 }  
 }  
 }  
  
 public Iterable<Integer> order() {  
 return order;  
 }  
  
 public static void main(String[] args) {  
 Graph graph = new Graph("g.txt");  
 GraphDFS graphDFS = new GraphDFS(graph);  
 System.*out*.println(graphDFS.order());  
 }  
}

存在Bug：只对连通图有效。

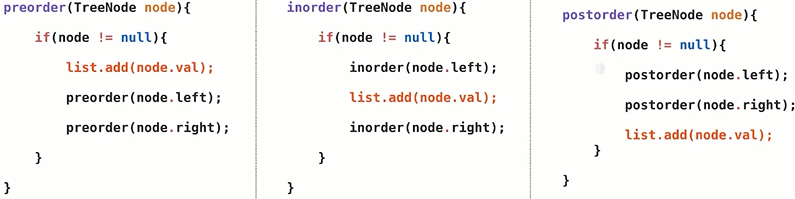
## 3.5 图的深度优先遍历的改进



解决存在连通分量的情况下遍历不全的问题。

## 3.6 更多关于图的深度优先遍历

**二叉树的深度优先遍历**



**图的深度优先遍历**



一个图有若干个相邻的节点，没有“中间”这个概念了，所以没有中序遍历。

**图的深度优先遍历，复杂度是：O(V + E)**

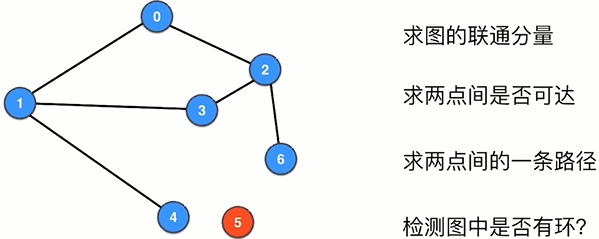
## 3.7 使用邻接矩阵进行图的深度优先遍历

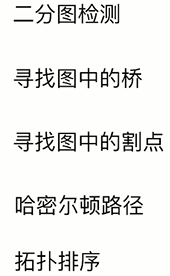
## 3.8 使用图的接口

## 3.9 非递归实现图的深度优先遍历

# 4.图的深度优先遍历的应用

别看图的深度优先遍历简单，用处可多了。联通分量，路径问题，环检测，二分图检测，都可以用DFS解决。不仅能够解决这些问题，还将进一步，对递归函数的设计与编写，有更深刻的体会。





## 4.1 图的连通分量的个数

## 4.2 DFS中的一个技巧

## 4.3 求解联通分量

## 4.4 单源路径问题

## 4.5 单源路径问题的编程实现

## 4.6 单源路径问题的一个小优化

## 4.7 所有点对路径问题

## 4.8 提前结束递归：路径问题的另一个优化

## 4.9 无向图的环检测

## 4.10 二分图检测

## 4.11 实现二分图检测

## 4.12 本章小结和更多拓展

# 5.图的广度优先遍历

图的广度优先遍历是图的另外一种遍历形式。图的广度优先遍历，不仅仅可以解决大多数DFS可以解决的问题，还拥有着独特的性质。这章还将揭示DFS和BFS的神奇联系。

## 5.1 从树的广度优先遍历，到图的广度优先遍历

## 5.2 图的 BFS 的实现

## 5.3 使用 BFS 求解路径问题

## 5.4 更多关于使用 BFS 求解路径问题

## 5.5 使用 BFS 求解联通分量问题

## 5.6 使用 BFS 求解环检测问题

## 5.7 使用 BFS 求解二分图检测问题

## 5.8 BFS 的重要性质

## 5.9 无权图的最短路径

## 5.10 BFS 和 DFS 的神奇联系

# 6.图论问题建模和 floodfill

别看我们只学习了图的DFS和BFS，但其实，已经能够解决80%的面试问题了。在这一章，我们就将通过几个经典算法面试问题，来说说图论问题建模的套路。同时，我们会接触图论领域的一个经典算法：floodfill。

## 6.1 算法笔试面试中的图论问题书写

## 6.2 图的建模和二维网格中的小技巧

## 6.3 编程实现图的建模

## 6.4 floodfill 算法

## 6.5 更多 floodfill 的问题

## 6.6 连通性和并查集

## 6.7 Flood Fill 的更多优化

# 7.图论搜索和人工智能

这章将重点关注算法面试中的BFS。不要小看BFS，求解图论面试问题的过程中，将在不经意间，接触到人工智能领域解决问题的一个重要思想：搜索。而BFS，则是解决一大类人工智能问题的基石。

## 7.1 算法笔试面试中的 BFS 问题

## 7.2 图论建模的核心：状态表达

## 7.3 实现转盘锁问题

## 7.4 一道智力题

## 7.5 代码实现一道智力题

## 7.6 Leetcode 上一个困难的问题

## 7.7 实现滑动谜题

## 7.8 图论搜索和人工智能

# 8.桥和割点，以及图的遍历树

对于一张图，我们可以分析出各种不同的指标。桥和割点就是一类很重要的指标，在很多问题中有着巨大的作用。这章，我们就来看看求解图中的桥和割点的算法。同时，大家也将更深刻的了解到：DFS决不仅仅是遍历这么简单。

## 8.1 什么是桥

## 8.2 寻找桥的算法思路

## 8.3 模拟寻找桥算法

## 8.4 实现寻找桥算法

## 8.5 图的遍历树

## 8.6 寻找割点的算法思路

## 8.7 实现寻找割点算法

## 8.8 关于变量语义，和如何书写正确的算法

# 9.哈密尔顿问题和状态压缩

在这一章，我们将接触大名鼎鼎的哈密尔顿问题。在解决哈密尔顿问题的过程中，我们还将回顾诸如回溯法，状态压缩，记忆化搜索等经典算法设计思想。

## 9.1 哈密尔顿回路和 TSP

## 9.2 求解哈密尔顿回路的算法

## 9.3 实现哈密尔顿回路的算法

## 9.4 哈密尔顿回路算法的一个优化

## 9.5 哈密尔顿路径算法

## 9.6 Leetcode 上的哈密尔顿问题

## 9.7 状态压缩

## 9.8 基于状态压缩的哈密尔顿算法

## 9.9 记忆化搜索

## 9.10 哈密尔顿回路和哈密尔顿路径小结

# 10.欧拉回路和欧拉路径

在这一章，我们将接触大名鼎鼎的欧拉问题。欧拉问题和哈密尔顿问题看起来极其相似，但是解决思路却完全不同。欧拉问题有极其优美的数学解法，在这一章，希望同学们也能领略数学之美。

## 10.1 什么是欧拉回路

## 10.2 欧拉回路的存在性及证明

## 10.3 实现欧拉回路存在性的判断

## 10.4 求解欧拉回路的三种算法

## 10.5 Hierholzer 算法模拟

## 10.6 实现 Hierholzer 算法

## 10.7 欧拉路径和本章小结

# 11.最小生成树

在这一章，我们将开始迈入有权图的世界，来看最小生成树问题。我们将介绍两种最小生成树算法：Prim和Kruskal。通过这两个算法的学习，大家也将看到高级数据结构，比如并查集和优先队列，在解决复杂算法问题中的作用。

## 11.1 带权图及实现

## 11.2 Map 的遍历

## 11.3 最小生成树和 Kruskal 算法;

## 11.4 切分定理

## 11.5 Kruskal 算法的实现

## 11.6 并查集动态环检测

## 11.7 Prim 算法的原理及模拟

## 11.8 实现 Prim 算法

## 11.9 Prim 算法的优化

## 11.10 本章小结和更多关于最小生成树问题的讨论

# 12.最短路径算法

最短路径问题应该是图论领域最典型，也是最古老的应用了。尽管如此，最短路径算法并没有那么简单，不同的最短路径算法，有着各自的优劣和适应场合。在这一章，我们就将系统地学习比较这些最短路径算法。

## 12.1 有权图的最短路径问题

## 12.2 Dijkstra 算法的原理和模拟

## 12.3 实现 Dijkstra 算法

## 12.4 Dijkstra 算法的优化

## 12.5 更多关于 Dijkstra 算法的讨论

## 12.6 Bellman.Ford 算法

## 12.7 负权环

## 12.8 实现 Bellman.Ford 算法.

## 12.9 更多关于 Bellman.Ford 算法的讨论

## 12.10 Floyd 算法

## 12.11 实现 Floyd 算法

## 12.12 本章小结和更多关于最短路径问题的讨论

# 13.有向图算法

在这一章，我们将迈入有向图的世界。我们将看有向图和无向图有什么本质的不同，进而深入研究 DAG 的性质，从而学习拓扑排序，关键路径，SCC等算法问题。

## 13.1 有向图的实现

## 13.2 有向图算法

## 13.3 有向图环检测和 DAG

## 13.4 有向图的度:入度和出度

## 13.5 有向图求解欧拉回路

## 13.6 拓扑排序

## 13.7 拓扑排序算法的实现

## 13.8 另一个拓扑排序算法

## 13.9 另一个拓扑排序算法的实现

## 13.10 有向图的强连通分量

## 13.11 Kosaraju 算法

## 13.12 Kosaraju 算法的实现

## 13.13 有向图算法小节

# 14.网络流

在这一章，我们将接触一种全新的结构：网络。在图论的世界中，对“网络”有着特殊的定义。同时，也能延伸出大名鼎鼎的“网络流”算法。在这一章，我们将学习网络流这一图论领域的“高级算法”，看如何应用它，解决大量实际中的问题。

## 14.1 网络流模型和最大流问题

## 14.2 Ford.Fulkerson 思想

## 14.3 Edmonds.Karp 算法

## 14.4 最大流算法的基本架构

## 14.5 实现 Edmonds.Karp 算法

## 14.6 Edmonds.Karp 算法的测试和更多讨论

## 14.7 网络流问题建模

## 14.8 本章小结和更多相关讨论

# 15.匹配问题

匹配算法可以看作是网络流算法的延伸，也有着自己独特的思想。在这一章，我们将仔细看一种特殊的图结构：二分图，进而，仔细研究其中所涉及的匹配问题。

## 15.1 最大匹配和完美匹配

## 15.2 使用最大流算法解决匹配问题

## 15.3 实现二分图匹配算法

## 15.4 通过 Leetcode 的一个 Hard 问题，看匹配算法建模

## 15.5 匈牙利算法

## 15.6 匈牙利算法的实现

## 15.7 基于递归实现的匈牙利算法

## 15.8 匹配问题小结

# 16.更广阔的图论世界

通过这个课程的学习，相信大家已经是图论领域的小牛了。但是，图论领域远远不止如此，甚至很多极其前沿的科学问题，都和图论这个领域有着千丝万缕的联系。希望这个课程是一个开始，让感兴趣的同学们，可以在更广阔的图论世界翱翔。大家加油！

## 16.1 更广阔的图论算法世界