时长：20 小时

## 和 bobo 老师一起，玩转图论算法

### 欢迎大家来到《玩转图论算法》

### 图论到底有什么用

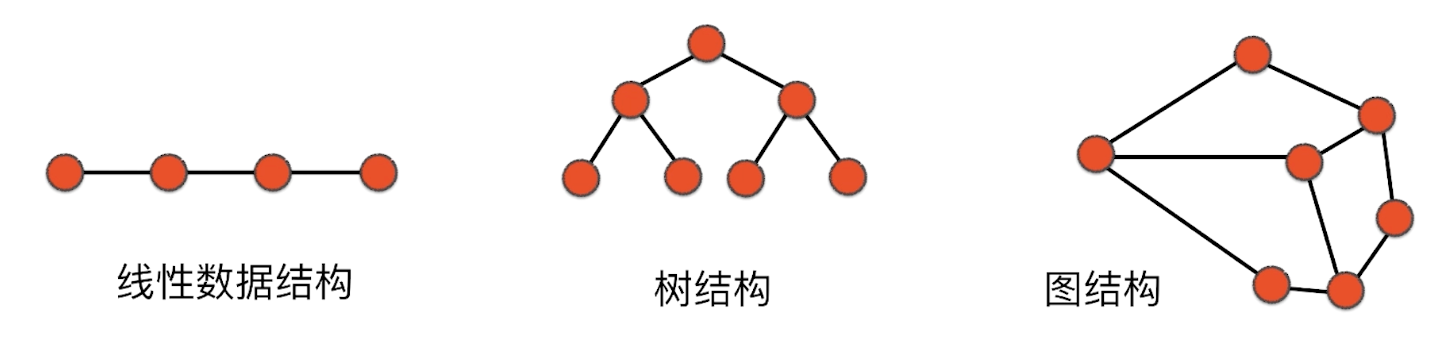
### 课程编程环境的搭建

## 图的基本表示

### 图的分类

一共是四类，以**有向无向**、**有权无权**进行组合的，四种分类

》》看一下下面的数据结构



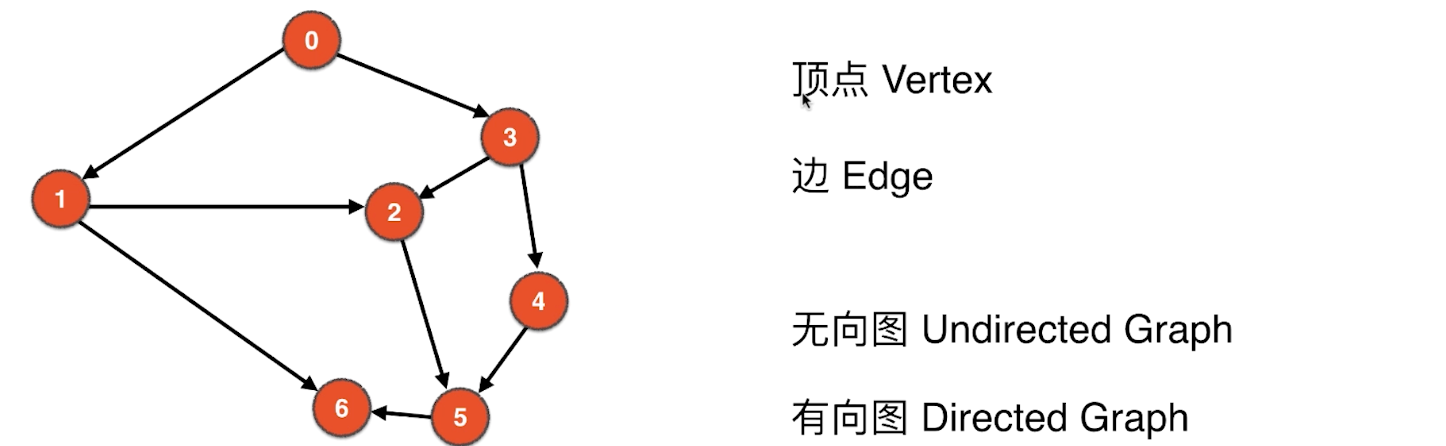
最大堆、最小堆就是用数组存储

线段树

》》图的两个基本的概念

**顶点 Vertex**

**边 Edge**



》》北京地铁建模运算

A close up of a map

Description automatically generated

\*\* 社交网络

好友关系可以用无向图，而关注的这个事情就需要用有向图了



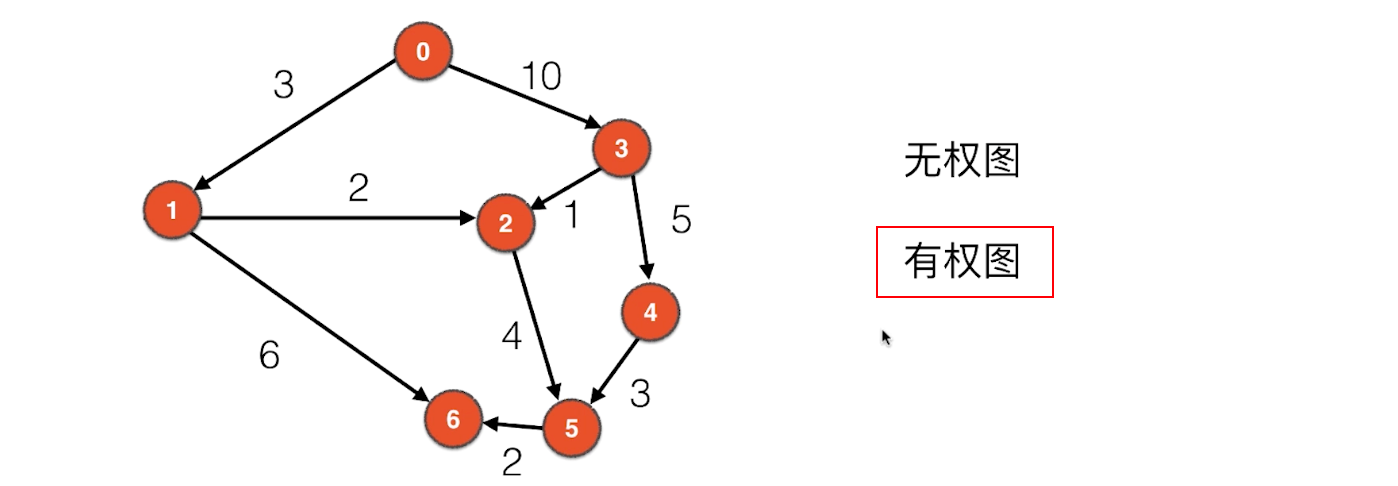
A picture containing object

Description automatically generated

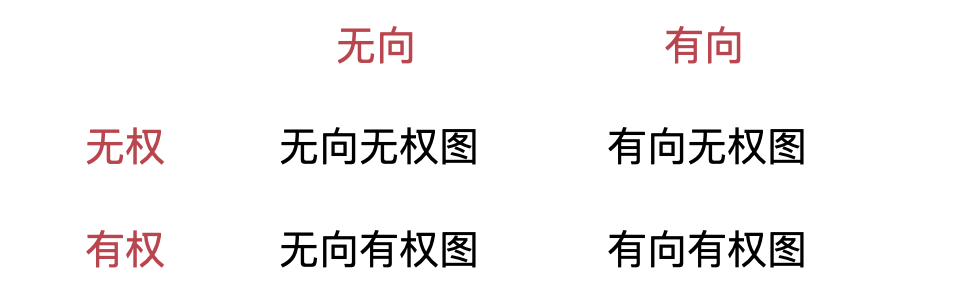
》》另外一个维度就是有没有权值

有权图，这个权就是图的边上的信息





》》最终的图的分类细分为



》》总结

每个算法需要作用在哪个类的图上来说

整个课程从**无向无权图**这个简单的来开始讲，后面后讲到其他的图分类算法。

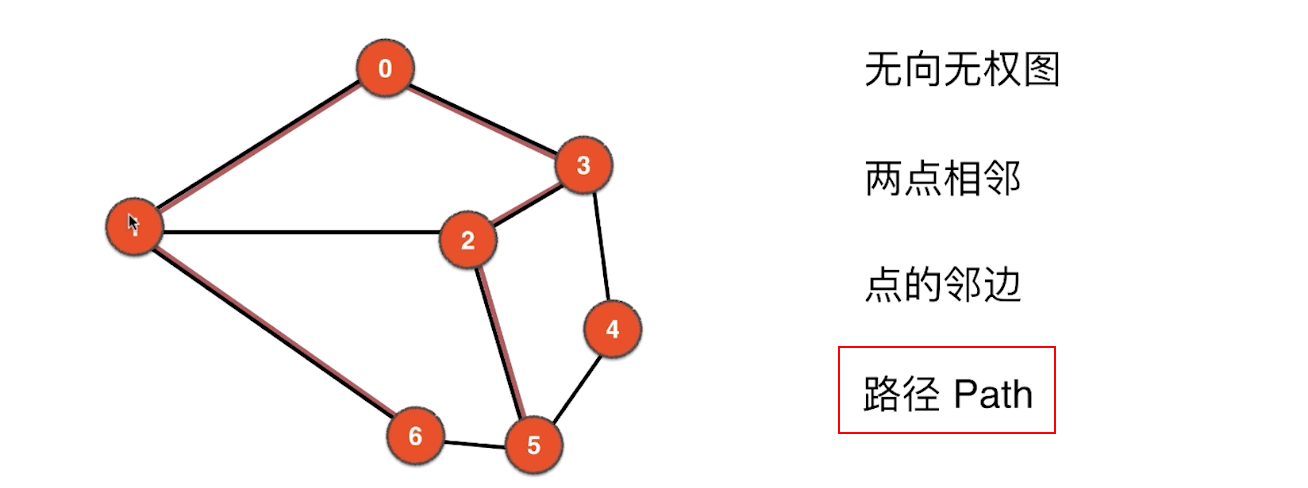
### 图的基本概念

》》主要介绍以下的几个概念-以**无向无权图**学习

两点相邻、点的邻边、路径 Path、环 Loop、自环边、平行边、联通分量、一个顶点的度 degree

**两点相邻、点的邻边**：相邻的点就知道了点的邻边，是相辅相成的，知道了一个就知道了另外的一个

**路径 Path**：一个点到另外一个点有多个路径，如 0-1-6



**环 Loop**：从一个点，经过若干条边后，又回到了这个点，就是一个环了。

**自环边**：之前说的一条边都是不同的点，那么边的两边都是自己的

**平行边**：两个顶点之间有两条边，就形成了平行边

自环边和平行边一般都是特殊的情况，都需要先处理下

在图论的知识中，没有自环边、没有平行边的图，被称为简单图

A screenshot of a cell phone

Description automatically generated

**联通分量：**一张图相互连接可以抵达的叫做一个联通分量

如下图，添加 7 和 8 两个顶点，

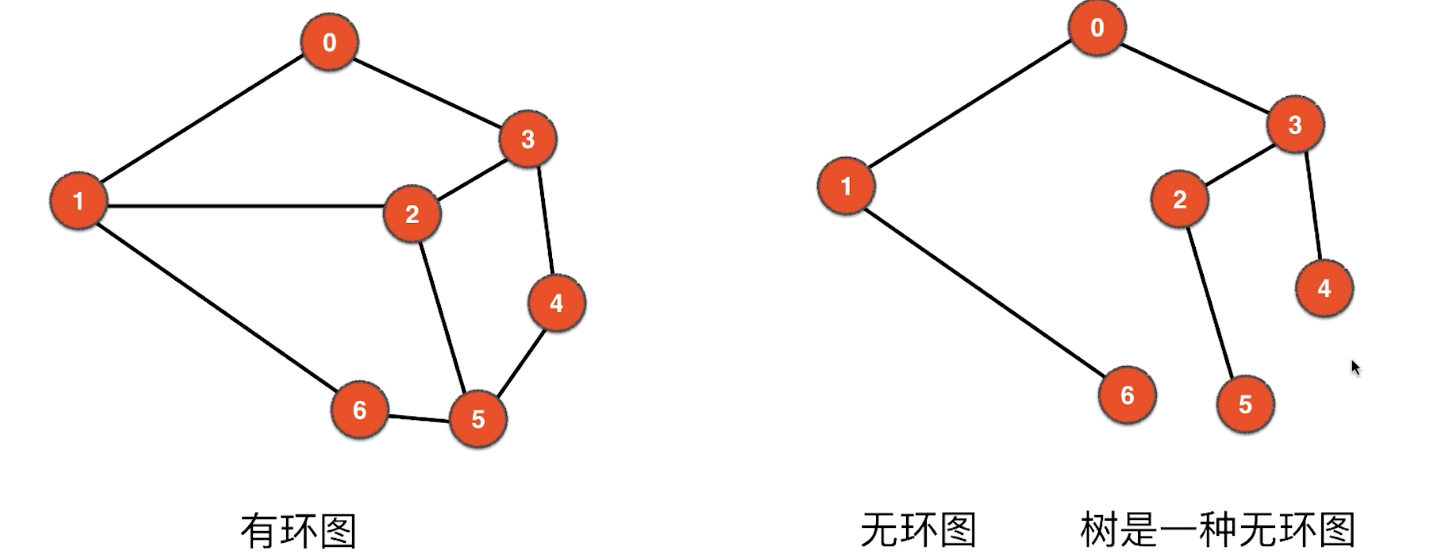
A screenshot of a cell phone

Description automatically generated

\*\* 一个图不一定是有环的，如下所示

树是满足图的定义的。

在讲树的数据结构是，根节点是唯一的，而在图论中讲到的树，的根节点是不唯一的，如下，0 可以是根节点，3 也可以是



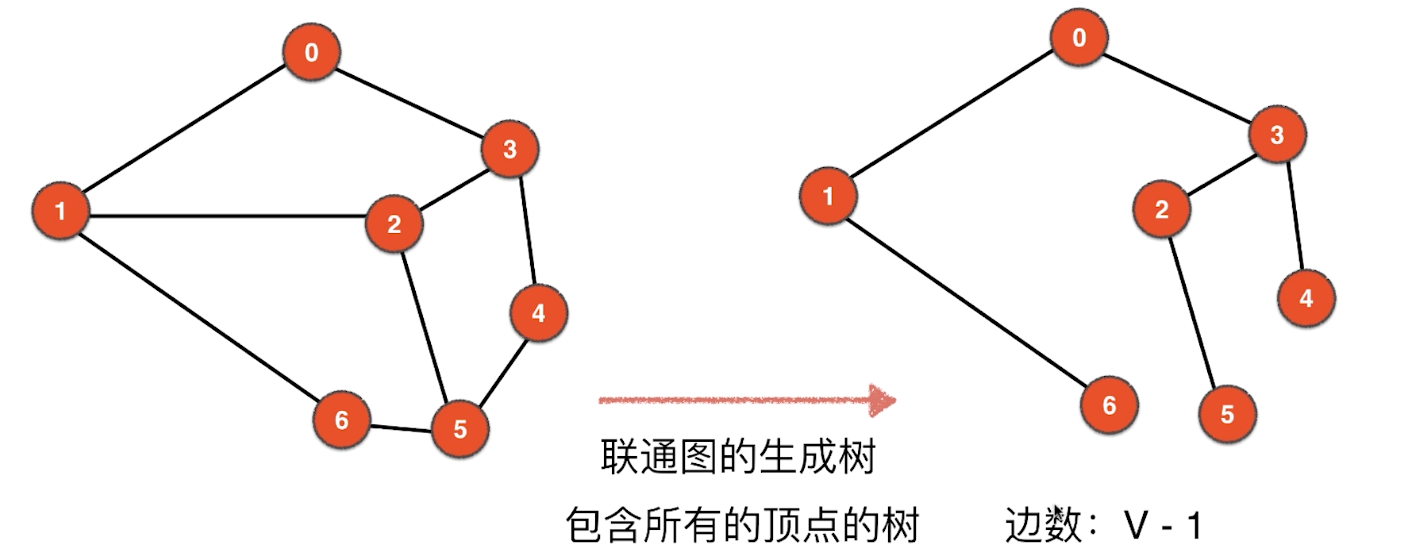
如下图，无环图一定是树是不对的。

A picture containing table

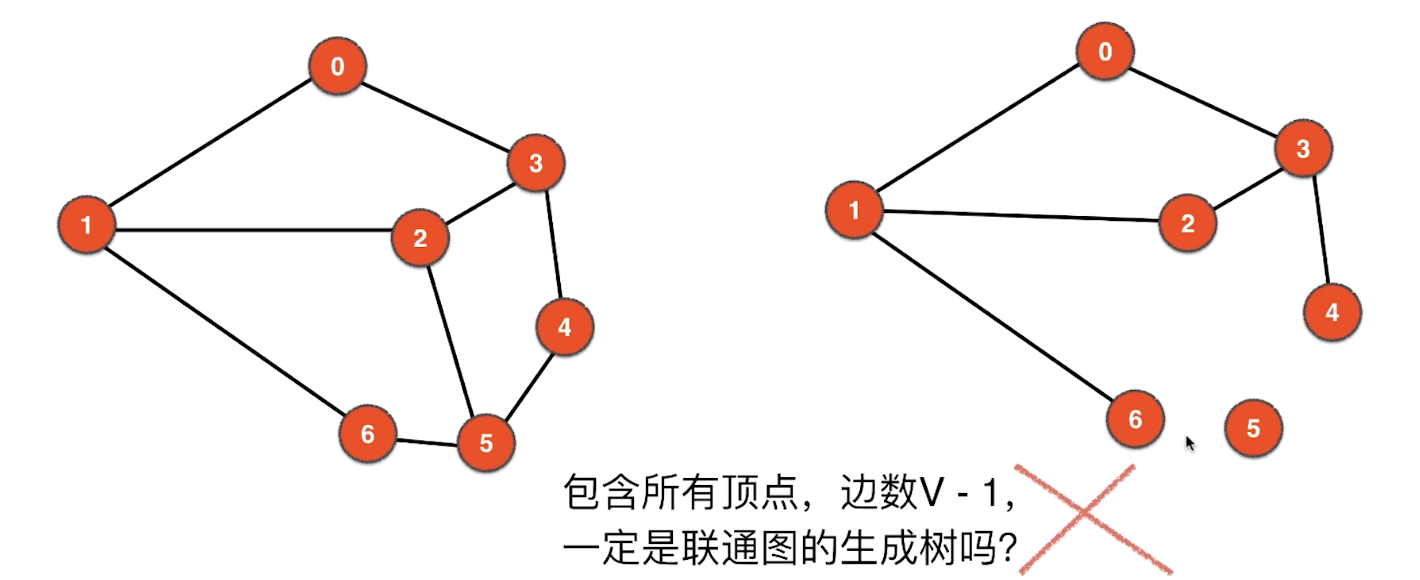
Description automatically generated

\*\* 联通图的生成树

这颗生成树

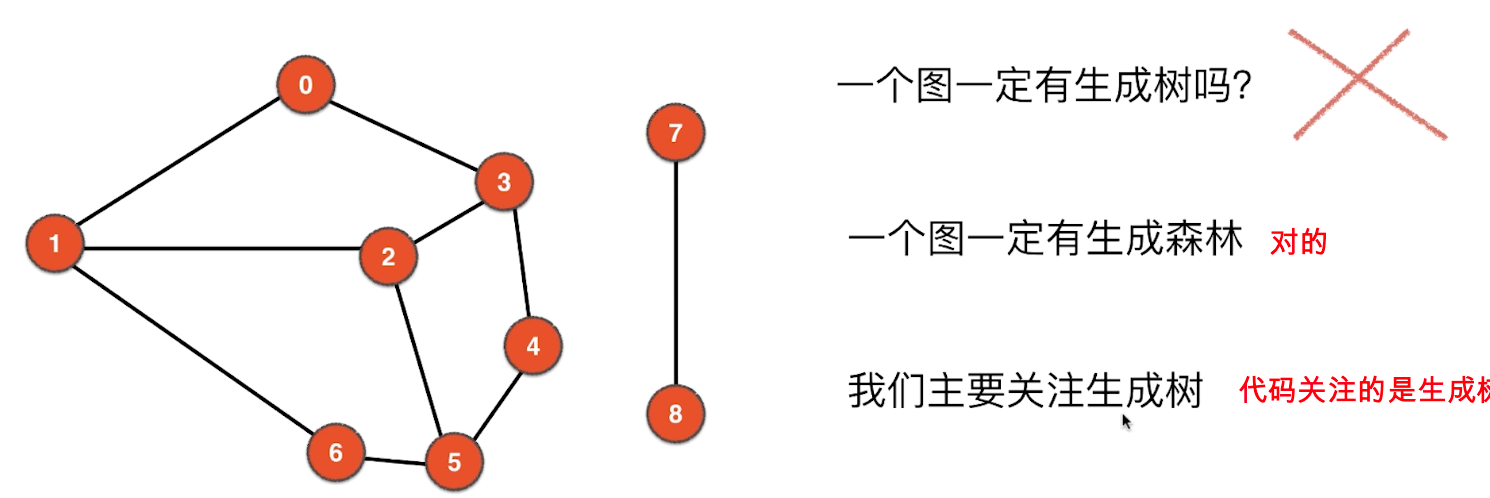


反过来问？



一个图一定有生成树吗？

答案是不对的，需要加上联通的条件，如下图所示



**顶点的度 degree**

对于有向图来说：会有些区别，后面会说到

对于无向图来说：就是顶点相邻的边数，这里不考虑自环边和平行边的情况

A screenshot of a cell phone

Description automatically generated

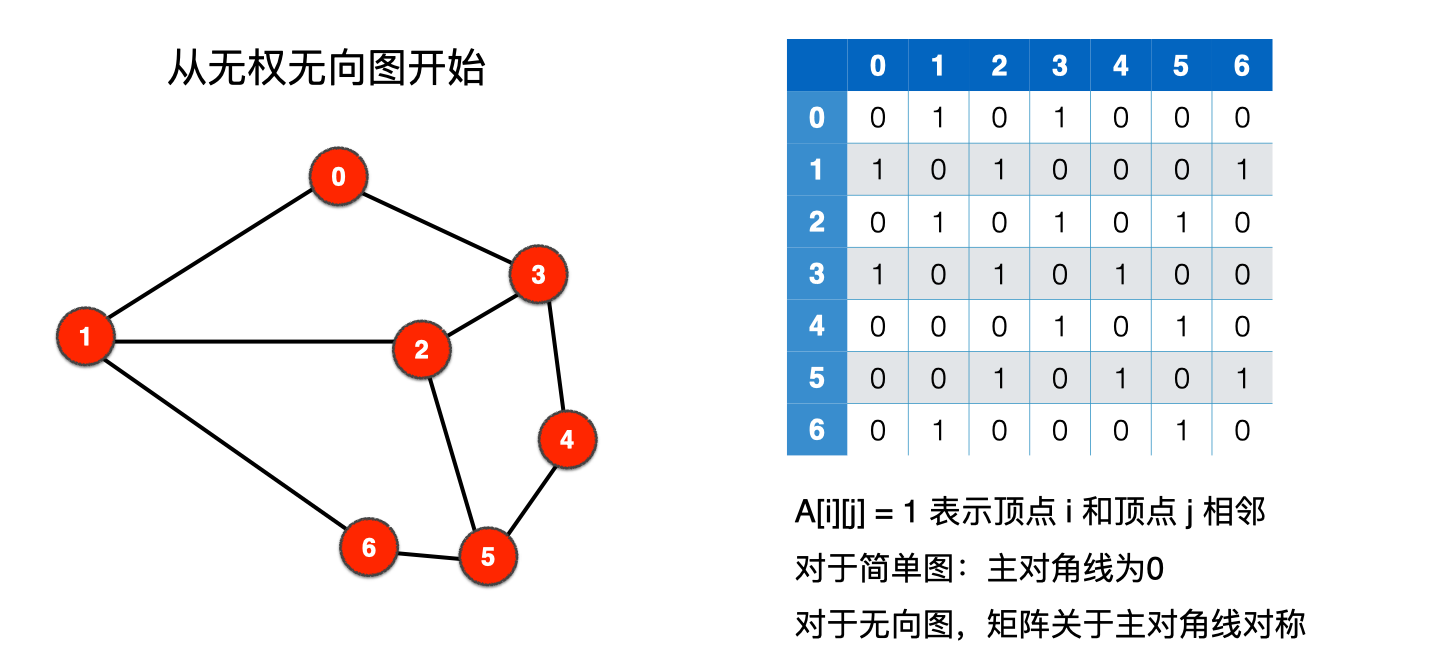
### 图的基本表示：邻接矩阵

》》先看下基本的表示方式以及概念

可以很简单的用一个方阵来表示，是一个 V\*V 的方阵，V 表示顶点的个数

对角线是自环边，也就是自己到自己的，是没有的，所以都是 0。

矩阵是关于多角线的对称的，因为都是互相连接的，1 和 2 连接，也就是 2 和 1 也是连接的。



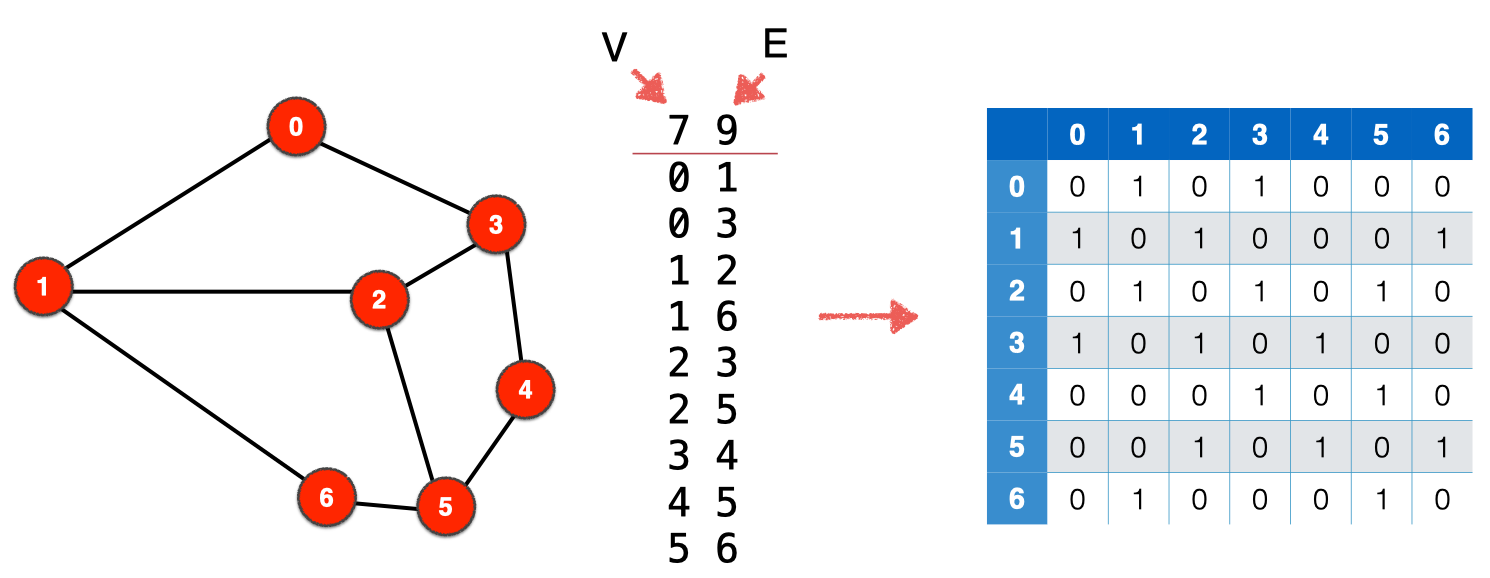
》》图的相关信息是从哪里来的？

这个矩阵可以用一个二维数组来表示，但是这个二维数组不是一上来就有的，我们从文件里面读取这个基本信息，然后构建这个二维数组

这个基本信息：*从文件给出，格式如下图的中间的样子。第一行表示有 7 个顶点，9 条边。下面的行就是表示哪两个顶点是连接的。如第二行就是表示 0 和 1 是有一条边连接的。*

这种格式是非常常规的，在竞赛中、考试中都是基于这样给出的。但是可能有些情况可能会不一样。但是都包括：有几个顶点、有几条边、哪些顶点是连接的。

其实理解了这种翻译的方法，不论怎么给出的图的形式，都是可以实现的，如 LeetCode 上就不是这么表示的，但是依然还是可以表示图的



》》我们接下来就是编程进行表示这个邻接矩阵

上面的两列数字用文本保存起来，然后读文本中的数据，进行图的构造

AdjMatrix.java

|  |
| --- |
| public class AdjMatrix {  private int V;  private int E;  private int[][] adj;  public AdjMatrix(String filename) {  File file = new Fiel(filename);  try(Scanner scanner = new Scanner(file);) {  this.V = scanner.nextInt();  this.adj = new int[V][V];  this.E = scanner.nextInt();  for(int i = 0; i < E ; i++) {  int a = scanner.nextInt();  int b = scanner.nextInt();  adj[a][b] = 1;  adj[b][a] = 1;  }  } catch(IOException e) {  e.printStackTrace();  }  }  @Override  public String toString() {  StringBuilder sb = new StringBuilder();  sb.append(String.format("V = %d, E = %d \n", V, E));  for(int i = i; i < V; i++) {  for (int j = 0; j < V; j++) {  sb.append(String.format("%d ", adj[i][j]));  }  sb.append("\n");  }  return sb.toString();  }  public static void main(String[] args) {  AdjMatrix ajdMatrix = new AdjMatrix("g.txt");  System.out.println(adjMatrix);  }  } |

### 更多图的方法

》》这里对上面的图添加其他方法，如下

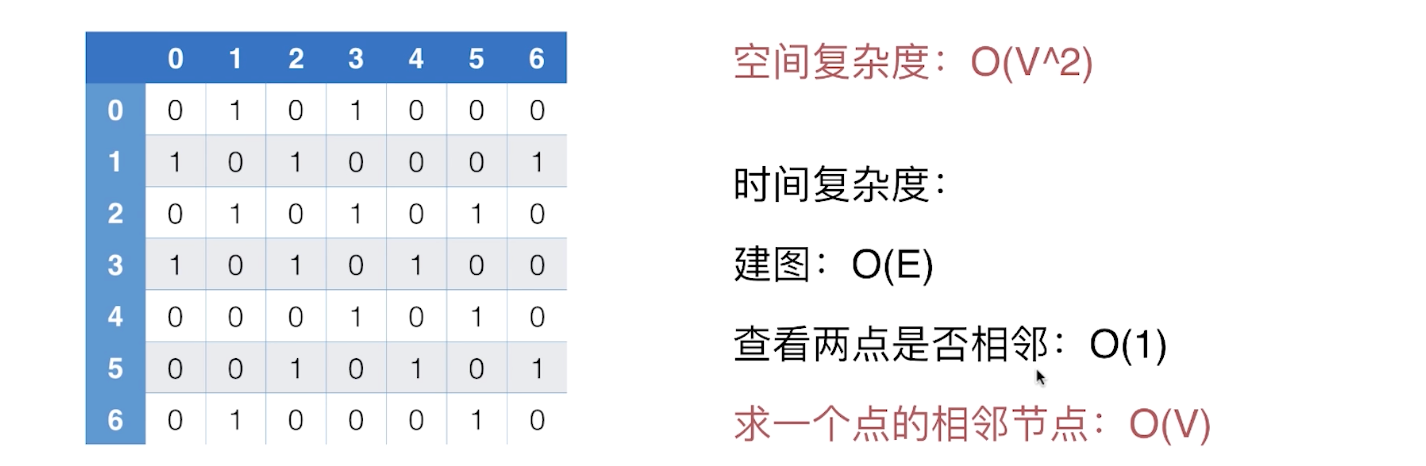
**1、错误处理；2、节点验证；3、V() E() 公共方法；4、hasEdge；5、求相邻节点；6、求一个点的度**

|  |
| --- |
| public class AdjMatrix {  private int V;  private int E;  private int[][] adj;  public AdjMatrix(String filename) {  File file = new Fiel(filename);  try(Scanner scanner = new Scanner(file);) {  this.V = scanner.nextInt();  if(this.v < 0 ) {  throw new IllegalArgumentException("V must be non-negative");  }  this.adj = new int[V][V];  this.E = scanner.nextInt();  if(this.E < 0 ) {  throw new IllegalArgumentException("E must be non-negative");  }    for(int i = 0; i < E ; i++) {  int a = scanner.nextInt();  int b = scanner.nextInt();  validateVertex(a);  validateVertex(b);  // 自环边  if(a == b) {  throw new IllegalArgumentException("Self Loop is Detected!");  }  // 平行边  if(adj[a][b] == 1) {  throw new IllegalArgumentException("Parallel Edge are Detected！");  }  adj[a][b] = 1;  adj[b][a] = 1;  }  } catch(IOException e) {  e.printStackTrace();  }  }    private void validateVertex(int v) {  if(v < 0 || v >= this.V) {  throw new IllegalArgumentException("vertex " + V " + "is invalid");  }  }  public int V() {  return this.V;  }  public int E() {  return this.E;  }  public boolean hasEdge(int v, int w) {  validateVertex(v);  validateVertex(w);  return adj[v][w] == 1;  }    // 返回 v 相邻的顶点，是一个及其有用的接口  public ArrayList<Integer> adj(int v) {  validateVertex(v);  ArrayList<Integer> res = new ArrayList();  for(int i = 0; i < V; i++) {  if(adj[v][i] == 1) {  res.add(i);  }  }  return res;  }  // 求一个顶点的度:无向图中的这个点有多少边  pulic 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 = i; i < V; i++) {  for (int j = 0; j < V; j++) {  sb.append(String.format("%d ", adj[i][j]));  }  sb.append("\n");  }  return sb.toString();  }  public static void main(String[] args) {  AdjMatrix ajdMatrix = new AdjMatrix("g.txt");  System.out.println(adjMatrix);  }  } |

### 图的基本表示：邻接表

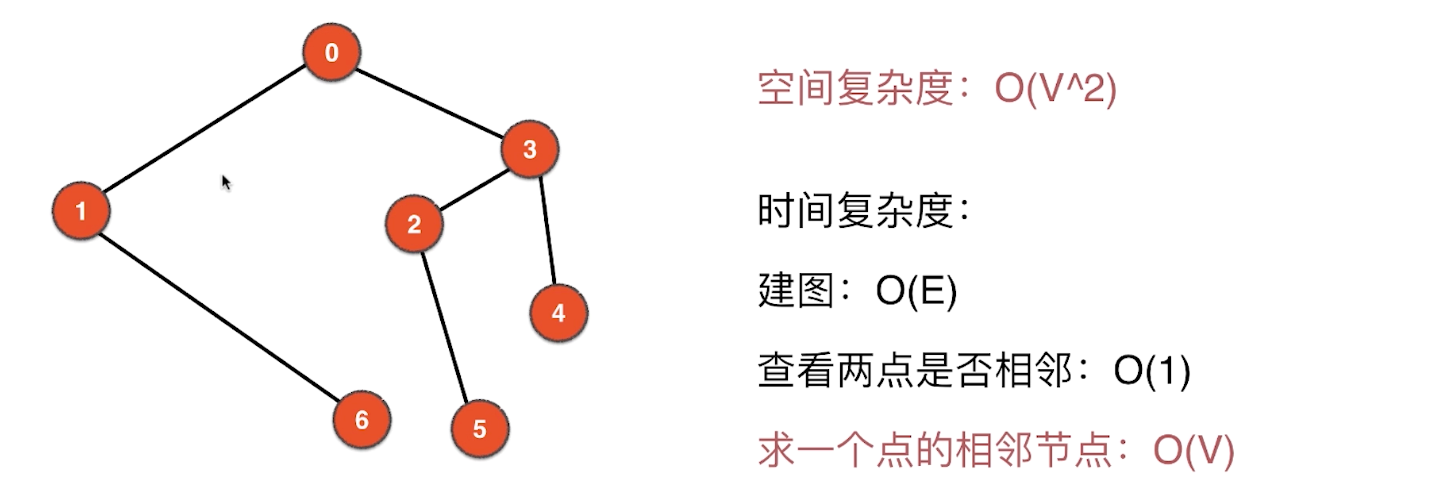
》》先看看上节课里面的邻接矩阵的一些情况

瓶颈在空间复杂度和求一个点的相邻节点的这两个上面。

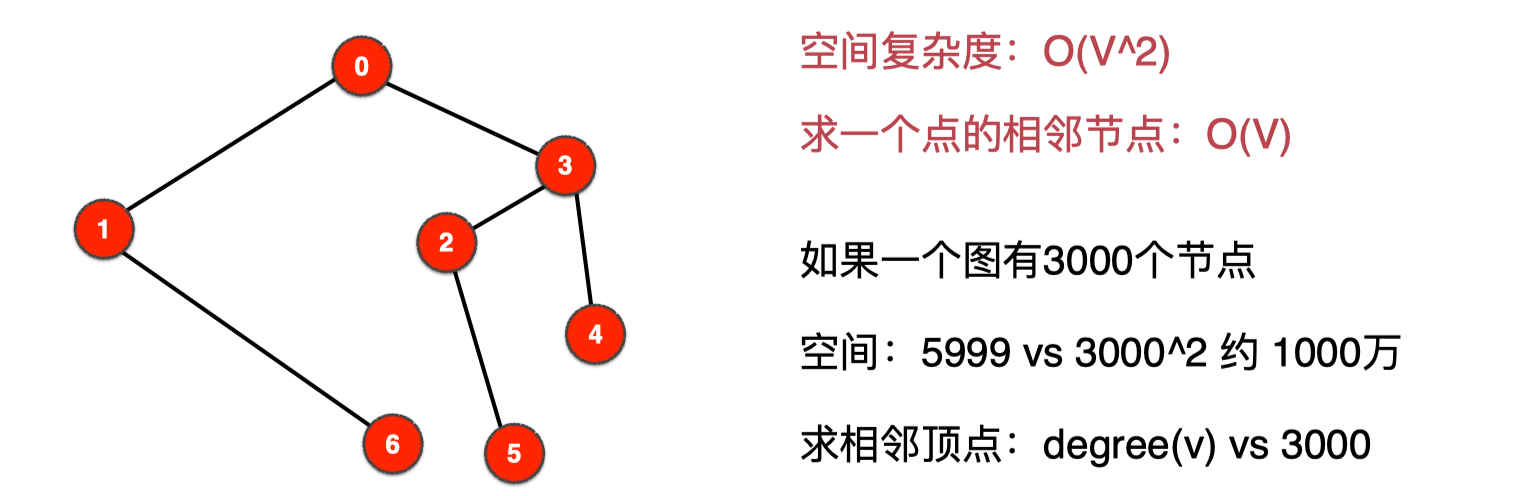


》》看看下面的例子

7 个节点，6 条边



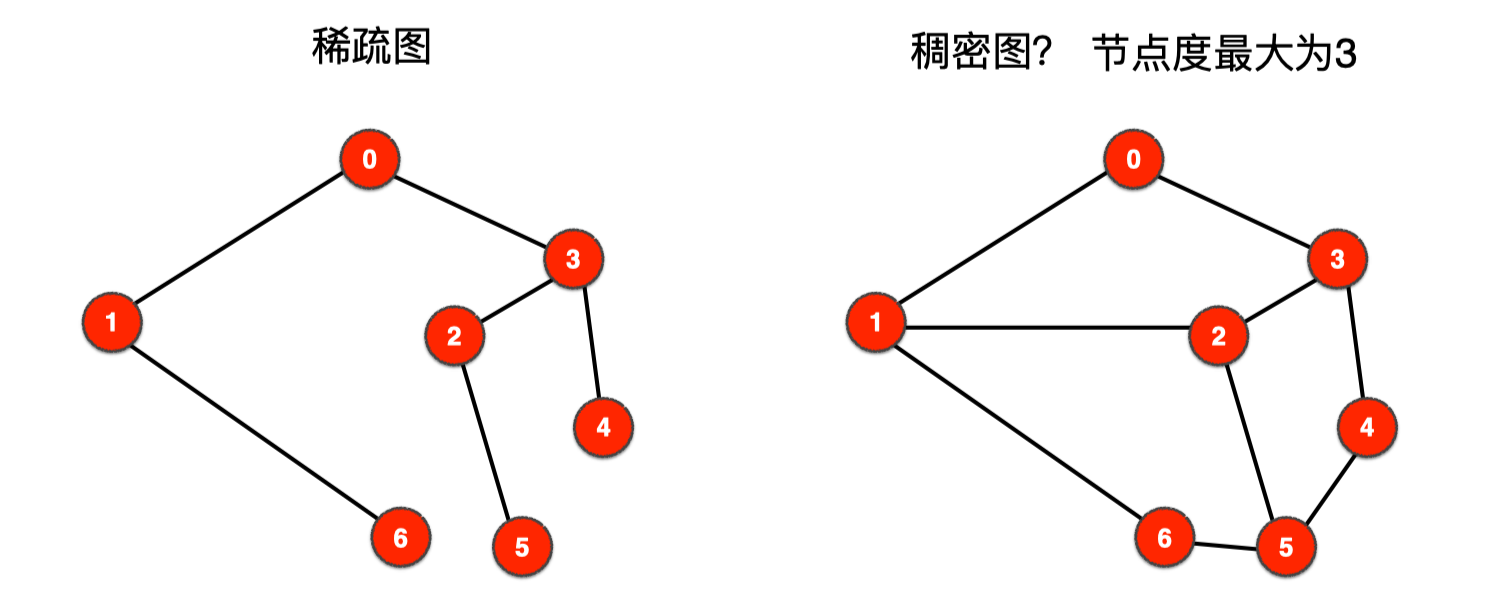
如果这个图有 3000 个节点，如果是树的话，就包括 5999 个信息（点数+边数），如果是用邻接矩阵表示的话就是 3000 的平方的数，大约是 1000 万，大概差 1000 多倍。



所以我们能找到 O(degree)的算法，就会比O(V)的算法快很多，这就要引出下面的稀疏图和稠密图的，如下

》》稀疏图和稠密图

系数和稠密是指边的多少？右边的这个图进行进一步的分析，如下下面的图

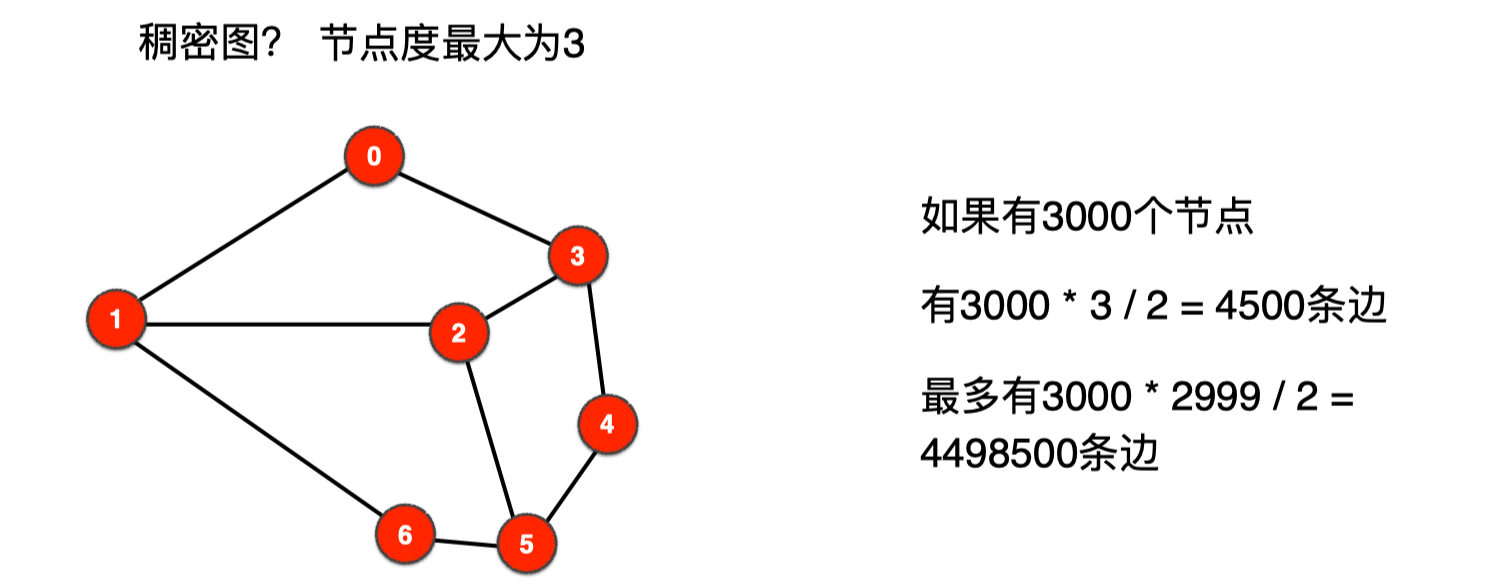


我们扩大到 3000 个节点来看看，进行边数的计算

如果节点度数是 3： 就是 4500条边

如果最多了有多少：就是 4 百万+的条边

*为什么要除以 2 了？是无向图，0 和 3 表示只连接一次*



\*\* 看一下下面的图的情况：

右边这个图还有一个称呼叫完全图，稠密图不一定是完全图

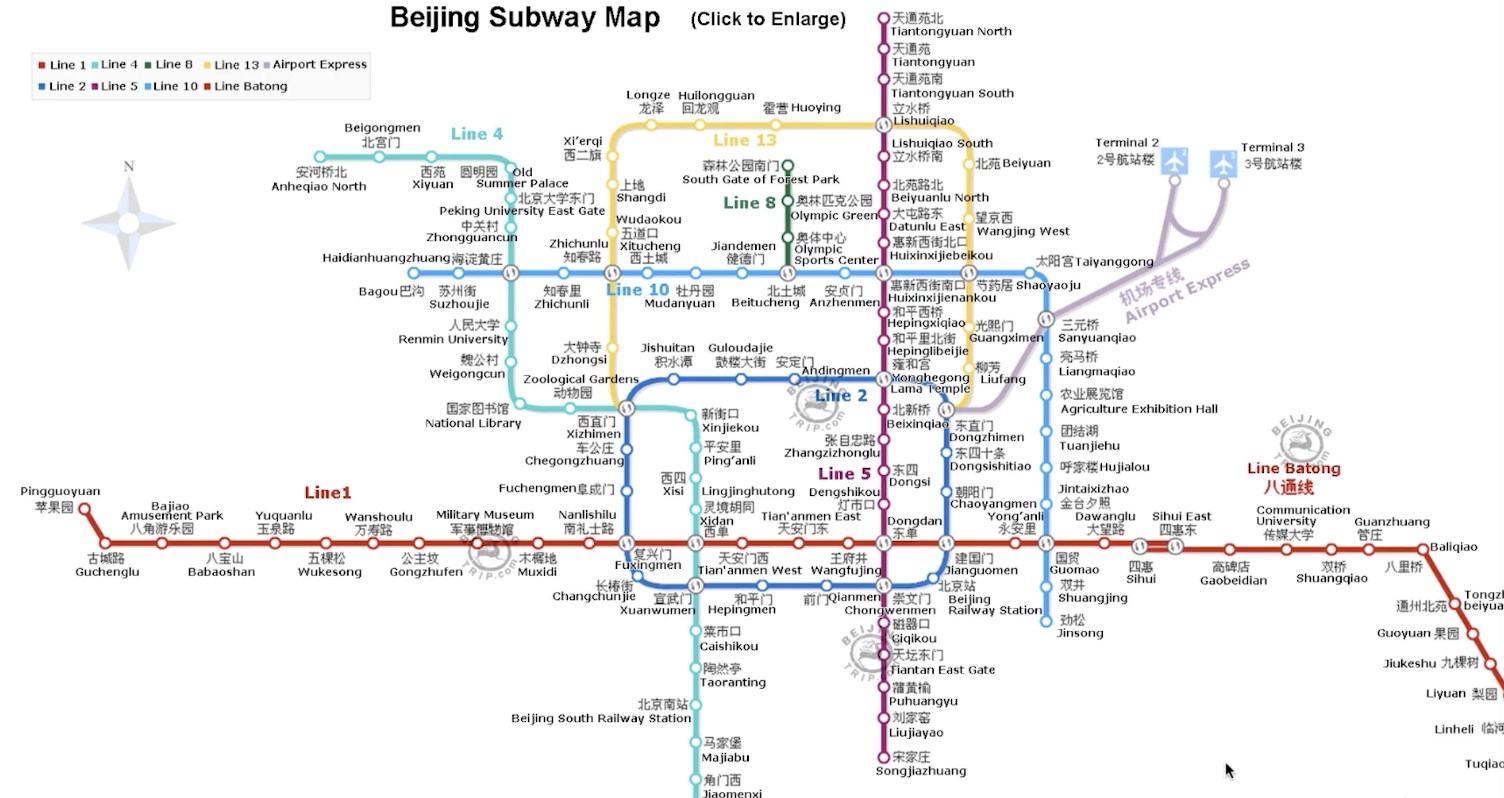
需要说明的是有时候看起来是稠密图，其实跟完全图比起来，就是稀疏图。

并且在实际的解决问题中，大部分的情况都是稀疏图

A close up of a logo

Description automatically generated

\*\* 如上面说的北京地铁的



\*\* 还有社交网络的，如微信和facebook 等这样的，如下图所示

A picture containing umbrella

Description automatically generated

虽然距了地铁和社交的例子，稀疏图和从稠密图是没有一个标准的，没有一个黑白分明的界限，很多的时候都是一个感觉，平均没个节点的度和度的最大值的比较，如果是小于二分之一的话，就肯定是稀疏图，对于这个概念了解一下就行。

因为下面介绍的存储无论对于稀疏图还是稠密图都是没有挑战的。

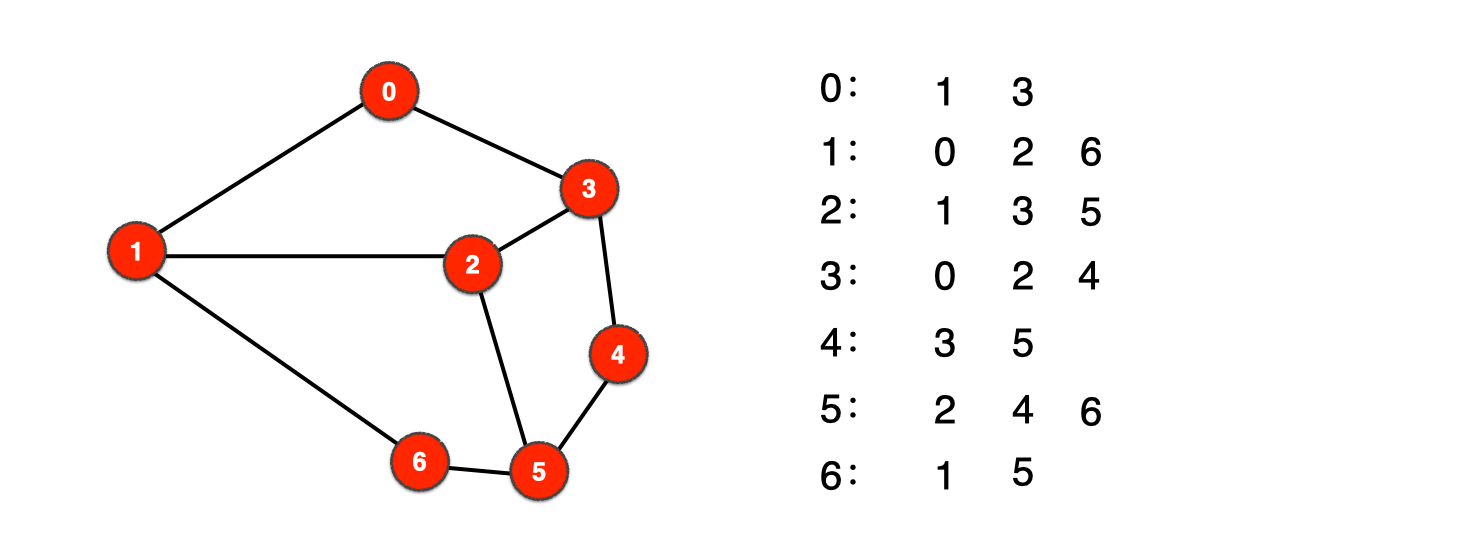
这里就总结了邻接矩阵的一些优点和缺点。针对稀疏图的解决方案

下面就解决了邻接表的表示方法

》》邻接表如下：

每个顶点后面都跟着一个链表，这个链表表示这个顶点与哪些顶点相连接

这样就表示这个顶点的存储至于相邻的顶点系那个关系，与图中有多少顶点是没有关系的。



### 邻接表的实现

AdjList.java

|  |
| --- |
| public class AdjList {  private int V;  private int E;  private LinkedList<Integer>[] adj;  public AdjMatrix(String filename) {  File file = new Fiel(filename);  try(Scanner scanner = new Scanner(file);) {  this.V = scanner.nextInt();  if(this.v < 0 ) {  throw new IllegalArgumentException("V must be non-negative");  }  this.adj = new LinkedList[V]; // 先申请数组的整个空间  for(int i = 0; i < V; i++) { // 然后每个元素申请空间  adj[i] = new LinkedList<>();  }  this.E = scanner.nextInt();  if(this.E < 0 ) {  throw new IllegalArgumentException("E must be non-negative");  }  for(int i = 0; i < E ; i++) {  int a = scanner.nextInt();  int b = scanner.nextInt();  validateVertex(a);  validateVertex(b);  // 自环边  if(a == b) {  throw new IllegalArgumentException("Self Loop is Detected!");  }  // 平行边  if(adj[a].contains(b)) { // a 点中是否包含 b 顶点，时间复杂度是O(a)  throw new IllegalArgumentException("Parallel Edge are Detected！");  }  adj[a].add(a);  adj[b].add(b);  }  } catch(IOException e) {  e.printStackTrace();  }  }    private void validateVertex(int v) {  if(v < 0 || v >= this.V) {  throw new IllegalArgumentException("vertex " + V " + "is invalid");  }  }  public int V() {  return this.V;  }  public int E() {  return this.E;  }  public boolean hasEdge(int v, int w) {  validateVertex(v);  validateVertex(w);  return adj[v].contains(w);  }    // 返回 v 相邻的顶点，是一个及其有用的接口  public ArrayList<Integer> adj(int v) {  validateVertex(v);  return adj[v];  }  // 求一个顶点的度:无向图中的这个点有多少边  pulic 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 = i; i < V; i++) {  sb.append(String.format("%d: ", i));  for (int w: adj[i]) { // 一般用 i j k 这种表示索引，而 v 和 w 表示顶点  sb.append(String.format("%d ", w));  }  sb.append("\n");  }  return sb.toString();  }  public static void main(String[] args) {  AdjMatrix ajdMatrix = new AdjMatrix("g.txt");  System.out.println(adjMatrix);  }  } |

### 邻接表的问题和改进

》》先总结下上节课用邻接表表示图的时候的一些情况  
空间复杂度可以写成 O(E)吗？也不是不行，V = E + 1，就是 2E+1，但是严格来说，是不行的，这里讲图的前提条件是联通的，但是如果不连通的情况下了？就是 7 个顶点，一条边没有？这个时候就不合适了

建图：为什么是 E\*V 了？关键是判断了包含是否是平行边的情况下

A screenshot of a cell phone

Description automatically generated

》》下面看到的就是建图的时间复杂度和查看两点是否相邻的，是需要优化的

A screenshot of a cell phone

Description automatically generated

》》缺点总结成如下的两点：

快速查重：就是新来一个点后，判断是否是相邻的

快速查看亮点是否相邻：hasEdge 方法

其实两个是一样的，都是判断是否是相邻的，也就是链表中快速查询，那么解决方法就是：

不要使用链表，而是使用哈希表或者红黑树

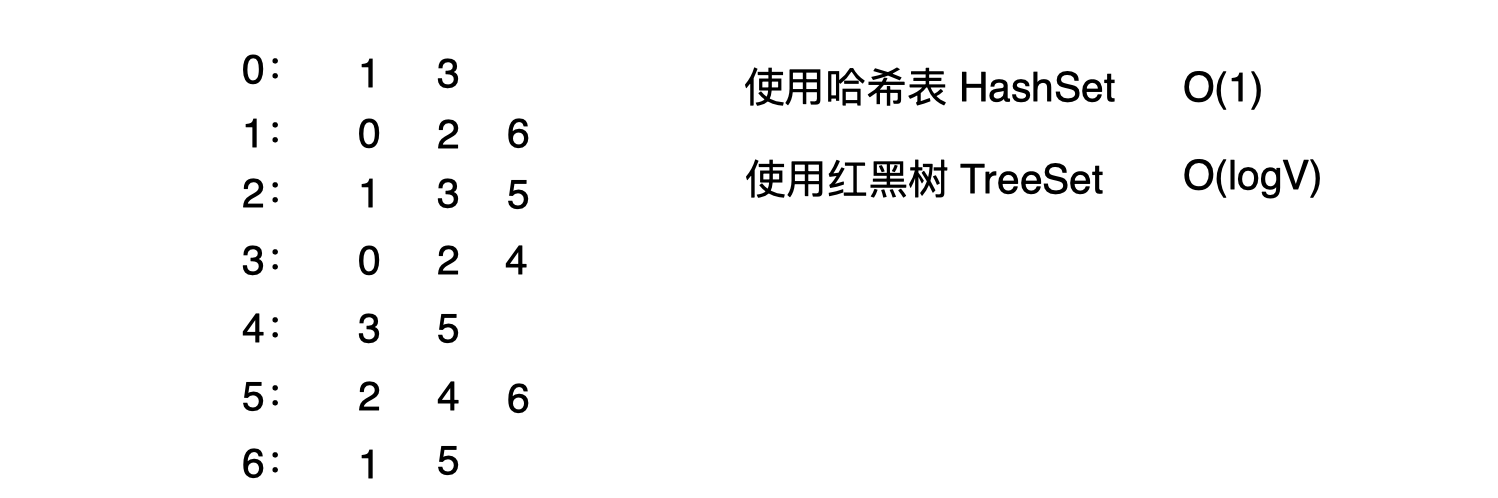
如果是哈希表就是 HashSet，那么就是 O(1) 级别

如果是红黑树就是 TreeSet，那么就是O(logV) 级别

A close up of a piece of paper

Description automatically generated

那么我们就可以将底层的数据结构换成 HashSet 和 TreeSet 就可以，看起来 O(1) 比O(logV)来看的话，确实比较多，但是实际起来这两者的差距是很微弱的。



下节课继续分析下决定使用哪个？是根据什么样的思路来进行的？

### 实现邻接表的改进

》》如果是使用哈希表了？

在视频里，LeetCode，比赛都是使用哈希表，那是没有问题的。

A screenshot of a cell phone

Description automatically generated

》》如果是使用红黑树了？

但是在这门课里面，选择用红黑树来进行实现

原因一：红黑树实现的结合是一个有序的，可以从小到大遍历，或者很快查找比元素大的或者小的，而哈希表是没有这个性质的。

为什么要选择这个算法了：后面的课程需要实现很复杂的算法，如果教学过程中，每次都不一样，就会带来一些疑惑。

原因二：红黑树更节省空间些。

A screenshot of a cell phone

Description automatically generated

》》代码实现：

AdjSet.java

|  |
| --- |
| public class AdjList {  private int V;  private int E;  private TreeSet<Integer>[] adj;  public AdjMatrix(String filename) {  File file = new Fiel(filename);  try(Scanner scanner = new Scanner(file);) {  this.V = scanner.nextInt();  if(this.v < 0 ) {  throw new IllegalArgumentException("V must be non-negative");  }  this.adj = new TreeSet[V]; // 先申请数组的整个空间  for(int i = 0; i < V; i++) { // 然后每个元素申请空间  adj[i] = new TreeSet<>();  }  this.E = scanner.nextInt();  if(this.E < 0 ) {  throw new IllegalArgumentException("E must be non-negative");  }  for(int i = 0; i < E ; i++) {  int a = scanner.nextInt();  int b = scanner.nextInt();  validateVertex(a);  validateVertex(b);  // 自环边  if(a == b) {  throw new IllegalArgumentException("Self Loop is Detected!");  }  // 平行边  if(adj[a].contains(b)) { // 时间复杂度由O(a) 变为了 O(logV)  throw new IllegalArgumentException("Parallel Edge are Detected！");  }  adj[a].add(a);  adj[b].add(b);  }  } catch(IOException e) {  e.printStackTrace();  }  }    private void validateVertex(int v) {  if(v < 0 || v >= this.V) {  throw new IllegalArgumentException("vertex " + V " + "is invalid");  }  }  public int V() {  return this.V;  }  public int E() {  return this.E;  }  public boolean hasEdge(int v, int w) {  validateVertex(v);  validateVertex(w);  return adj[v].contains(w);  }    // 返回 v 相邻的顶点，是一个及其有用的接口  // public TreeSet<Integer> adj(int v) { // 返回不同的类型，会造成一些困扰  public Iterable<Integer> adj(int v) { // 返回 Iterable，屏蔽内部实现细节，不需要关心用哪个实现  validateVertex(v);  return adj[v];  }  // 求一个顶点的度:无向图中的这个点有多少边  pulic int degree(int v) {  validateVertex(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 = i; i < V; i++) {  sb.append(String.format("%d: ", i));  for (int w: adj[i]) { // 一般用 i j k 这种表示索引，而 v 和 w 表示顶点  sb.append(String.format("%d ", w));  }  sb.append("\n");  }  return sb.toString();  }  public static void main(String[] args) {  AdjMatrix ajdMatrix = new AdjMatrix("g.txt");  System.out.println(adjMatrix);  }  } |

### 图的基本表示的比较

》》总的总结图如下所示

综合起来是 TreeSet 是一中更好的方法。

课程中也可以基于 HashSet 来设计的，建图和查看两点是否相邻都是 O(E)级别了。

小作业：底层基于 HashSet 做一个图的基本表示

图的表示方法，现在综合起来就有四种，基于现在的这种情况，我们可以做一个统一的接口，给一个默认的实现，然后用户使用接口的对象就可以，用户并不需要关心底层是用什么实现的。有兴趣的同学完全可以做一下这个事情

那么我们本门课程默认使用的是 TreeSet，大家关注的重点的在算法的层面就可以了

A screenshot of a cell phone

Description automatically generated

》》整理后的代码就是下面的样子：

后面的代码都会基于 Graph 来进行建图，以便于后面的操作

目前这个 Graph 的类只支持**无向无权图，**后面如果用到有权图的话也会进行改进

**Graph.java**

|  |
| --- |
| /// 暂时只支持无向无权图  public class Graph {  private int V;  private int E;  private TreeSet<Integer>[] adj;  public Graph(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 TreeSet[V];  for(int i = 0; i < V; i ++)  adj[i] = new TreeSet<Integer>();  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);  if(a == b) throw new IllegalArgumentException("Self Loop is Detected!");  if(adj[a].contains(b)) throw new IllegalArgumentException("Parallel Edges are Detected!");  adj[a].add(b);  adj[b].add(a);  }  }  catch(IOException 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);  }  public Iterable<Integer> adj(int v){  validateVertex(v);  return adj[v];  }  public int degree(int v){  validateVertex(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 v = 0; v < V; v ++){  sb.append(String.format("%d : ", v));  for(int w : adj[v])  sb.append(String.format("%d ", w));  sb.append('\n');  }  return sb.toString();  }  public static void main(String[] args){  Graph g = new Graph("g.txt");  System.out.print(g);  }  } |

## 图的深度优先遍历

### 数据结构遍历的意义

》》数据结构是需要可以遍历的

链表、树。具体看下图



A drawing of a person

Description automatically generated

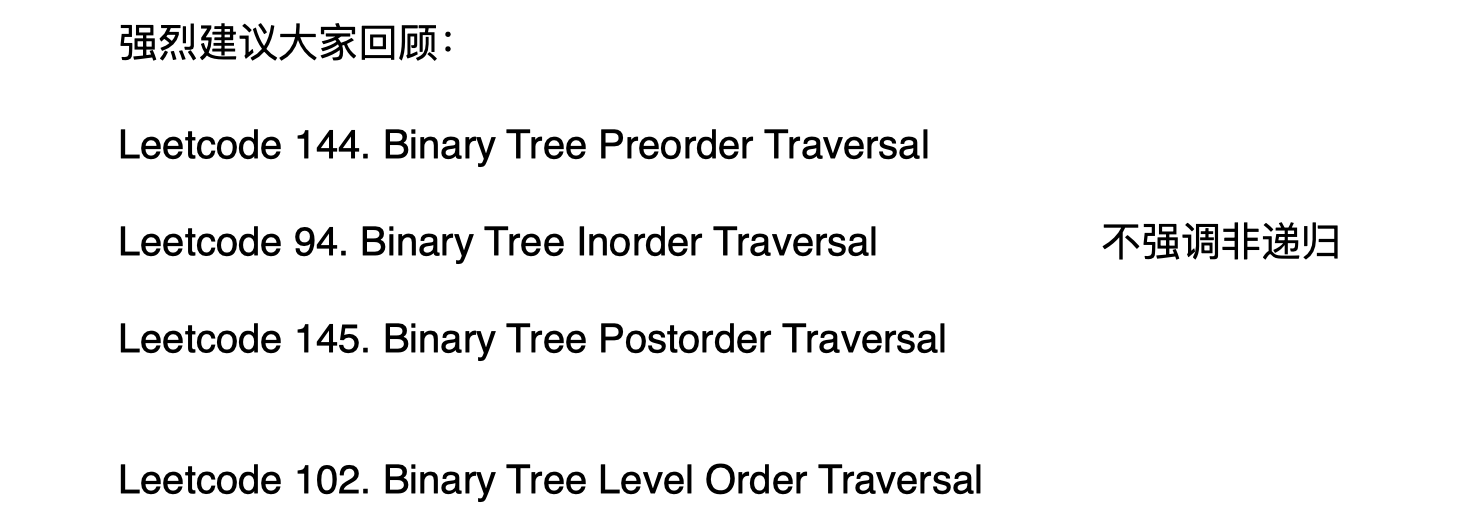
》》图的算法的遍历

A screenshot of a cell phone

Description automatically generated

》》二分树的遍历

这里建议大家先回顾下二叉树的前序、中序、后序以及层序遍历



并且也复习下 N 叉树的前序、后序以及层序遍历

A screenshot of a cell phone

Description automatically generated

### 从树的深度优先遍历，到图的深度优先遍历

》》图的深度优先遍历

先看下树的前序遍历的一个伪代码

A screenshot of a cell phone

Description automatically generated

》》然后，我们对比树的前序遍历的伪代码，我们进行图的遍历

图的遍历不同的就是有一个 visited 数组，用来维护已经访问过的节点

A screenshot of a cell phone

Description automatically generated

### DFS 逻辑的微观解读

》》这里先从简单的方法递归调用来诠释下递归的一个过程

右上是普通方法的调用：比如 A 调用了 B，在 B 里面调用了 C。

右下是递归：展开来看就是上面的调用的方法换成了自己

A screenshot of a cell phone

Description automatically generated

》》下面进行例子来微观解读一个树的 DFS 遍历的过程

A screenshot of a cell phone

Description automatically generated

\*\* 中间会经过很多的动画图，下面是最后一张图

A screenshot of a cell phone

Description automatically generated

### 实现图的深度优先遍历

》》这节课进行代码的编写来实现图的深度优先遍历

|  |
| --- |
|  |

### 图的深度优先遍历的改进

》》这节改进上面的 DFS 遍历

### 更多关于图的深度优先遍历

》》先回顾下二叉树的遍历

A screenshot of a cell phone

Description automatically generated

》》更多图的遍历

A screenshot of a cell phone

Description automatically generated

》》然后写一下深度优先序遍历

### 使用邻接矩阵进行图的深度优先遍历

### 使用图的接口

### 非递归实现图的深度优先遍历

## 图的深度优先遍历的应用

### 图的连通分量的个数

### DFS 中的一个技巧

### 求解连通分量

### 单源路径问题

### 单源路径问题的编程实现

### 单源路径问题的一个小优化

### 所有点对路径问题

### 提前结束递归：路径问题的另一个优化

### 无向图的环检测

### 二分图检测

### 实现二分图检测

### 本章小结和更多拓展

## 图的广度优先遍历

## 图论问题建模和 floodfill

## 图论搜索和人工智能

## 桥和割点，以及图的遍历树

## 哈密尔顿问题和状态压缩

## 欧拉回路和欧拉路径

## 最小生成树

## 最短路径算法

## 有向图算法

## 网络流

## 匹配问题

## 更广阔的图论世界