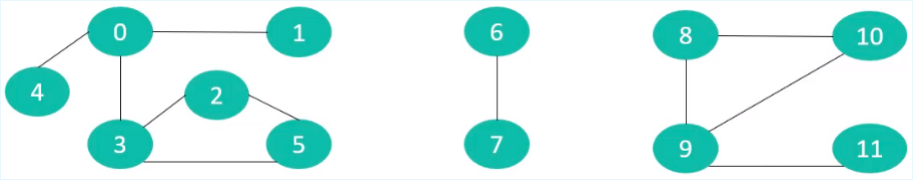
并查集和哈夫曼树

1. 并查集

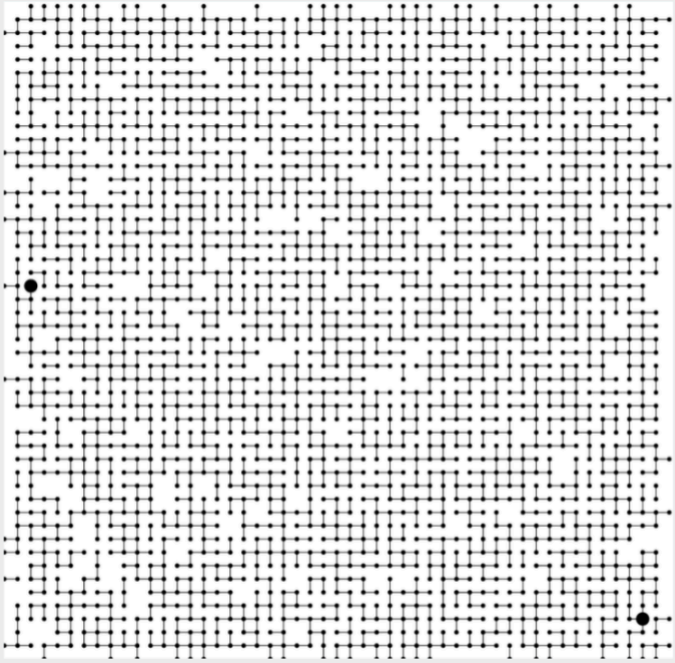
1.1 需求分析

假设现在有这样一个需求，如下图的每一个点代表一个村庄，每一条线就代表一条路，所以有些村庄之间有连接的路，有些村庄没有连接的路，但是有间接连接的路，根据上面的条件，能设计出一个数据结构，能快速执行下面2个操作：

1. 查询两个村庄之间是否有连接的路
2. 连接两个村庄



如何判断图中两点是否有**连接**



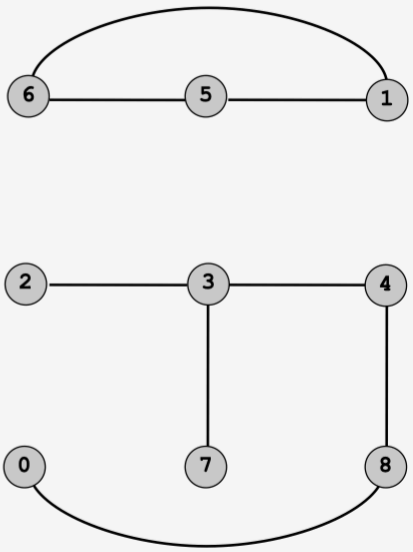
1.2 并查集概念

并查集（英文：Disjoint-set data structure，直译为不交集数据结构）是一种数据结构，用于处理一些不交集（Disjoint sets，一系列没有重复元素的集合）的合并及查询问题。并查集支持如下操作：

- **查询（Find）**：查询某个元素属于哪个集合，通常是返回集合内的一个“代表元素”。这个操作是为了判断两个元素是否在同一个集合之中。
- **合并（Union）**：将两个集合合并为一个。
- 添加：添加一个新集合，其中有一个新元素。添加操作不如查询和合并操作重要，常常被忽略。

由于支持查询和合并这两种操作，并查集在英文中也被称为联合-查找数据结构（Union-find data structure）或者合并-查找集合（Merge-find set）。

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)      no
 find(2, 4)      yes
union(5, 1)
union(7, 3)
union(1, 6)
union(4, 8)
 find(0, 2)      yes
 find(2, 4)      yes
```



1.3 并查集的算法介绍

- 并查集有2种常见实现思路

  a. Quick Find

  查找效率：O(1)

  合并效率：O(N)

  b. Quick Union

  查找效率：O（logN)

  合并效率：O（logN)

1.3.1 QuickFind的思路

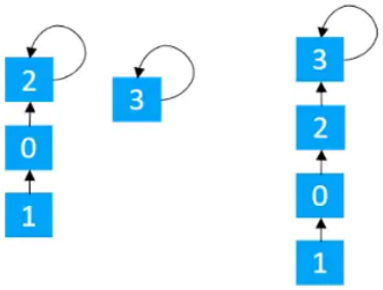
- 核心是将所有元素进行ID分组管理，每个元素对应一个ID号
- find的时候，只需要返回这个元素对应的分组ID
- 合并的时候Union(a,b)时候，将是属于a分组的元素，都改为b分组的ID号

1.3.2 QuickUnion的思路

- 将集合分为根结点和父节点的思想，所有节点保存他的父节点信息，当发现某个节点的父节点就是他自己的时候，这个节点就是根结点
- find操作就是找到这个元素的根结点，判断两个元素的根结点是不是一致，来判断是否连通。
- 合并操作的时候，Union(a, b)的时候，不是合并a和b，而是将a的根结点和b的根节点进行合并

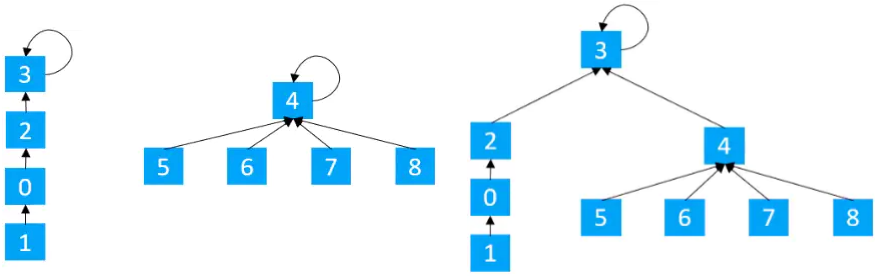
1.3.3 基于size的算法改进

- 在Union过程中，可能会出现不平衡的情况，甚至退化成为链表，Union(1,3)



- 将元素少的树，嫁接到元素多的树

1.3.4 基于rank的算法改进



- 矮的树，嫁接到高的树

1.3.5 路径压缩

- 在find时使路径上的所有节点都指向根节点，从而降低树的高度
    - 指向父节点的父节点
    - 都指向根结点

1.4 并查集代码实现

2. 哈夫曼树

2.1 哈夫曼树相关的几个名词

- 路径

> 在一棵树中，一个结点到另一个结点之间的通路，称为路径。
>
> 从根结点到结点 a 之间的通路就是一条路径。

- 路径长度

> 在一条路径中，每经过一个结点，路径长度都要加 1 。
>
> 例如在一棵树中，规定根结点所在层数为1层，那么从根结点到第 i 层结点的路径长度为 i – 1 。
>
> 从根结点到结点 c 的路径长度为 3。

- 节点的权

> 给每一个结点赋予一个新的数值，被称为这个结点的权。
>
> 例如，结点 a 的权为 7，结点 b 的权为 5。

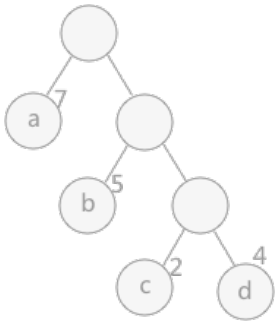
- 节点的带权路径长度

> 指的是从根结点到该结点之间的路径长度与该结点的权的乘积。
>
> 例如，结点 b 的带权路径长度为 2 * 5 = 10 。

树的带权路径长度为树中所有叶子结点的带权路径长度之和。通常记作“WPL”。

例如图中所示的这颗树的带权路径长度为：

> WPL = 7 * 1 + 5 * 2 + 2 * 3 + 4 * 3



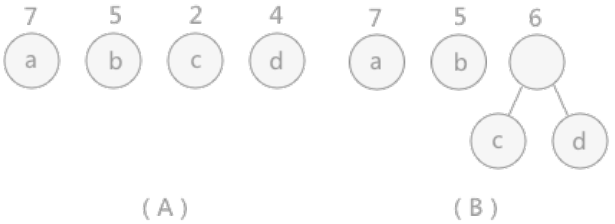
2.2 什么是哈夫曼树

当用 n 个结点（都做叶子结点且都有各自的权值）试图构建一棵树时，如果构建的这棵树的带权路径长度最小，称这棵树为“最优二叉树”，有时也叫“赫夫曼树”或者“哈夫曼树”。

2.3 构建哈夫曼树

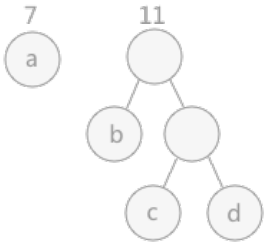
对于给定的有各自权值的 n 个结点，构建哈夫曼树有一个行之有效的办法：

1. 在 n 个权值中选出两个最小的权值，对应的两个结点组成一个新的二叉树，且新二叉树的根结点的权值为左右孩子权值的和；
2. 在原有的 n 个权值中删除那两个最小的权值，同时将新的权值加入到 n−2 个权值的行列中，以此类推；
3. 重复 1 和 2 ，直到所以的结点构建成了一棵二叉树为止，这棵树就是哈夫曼树。

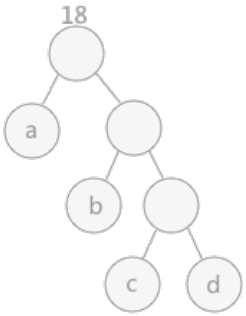


(A)

(B)



(C)



(D)