1.怎么布线最省钱：无向带权图的最小生成树

我们先来看一个和最小生成树相关的，现实生活中的案例：

某公司有6台交换机，分别部署在不同的网络中。交换机之间的通信距离不同，有的能够直接通信，有的必须经过其他交换机才能够连通

现在要求对交换机之间重新进行布线，并做如下要求：

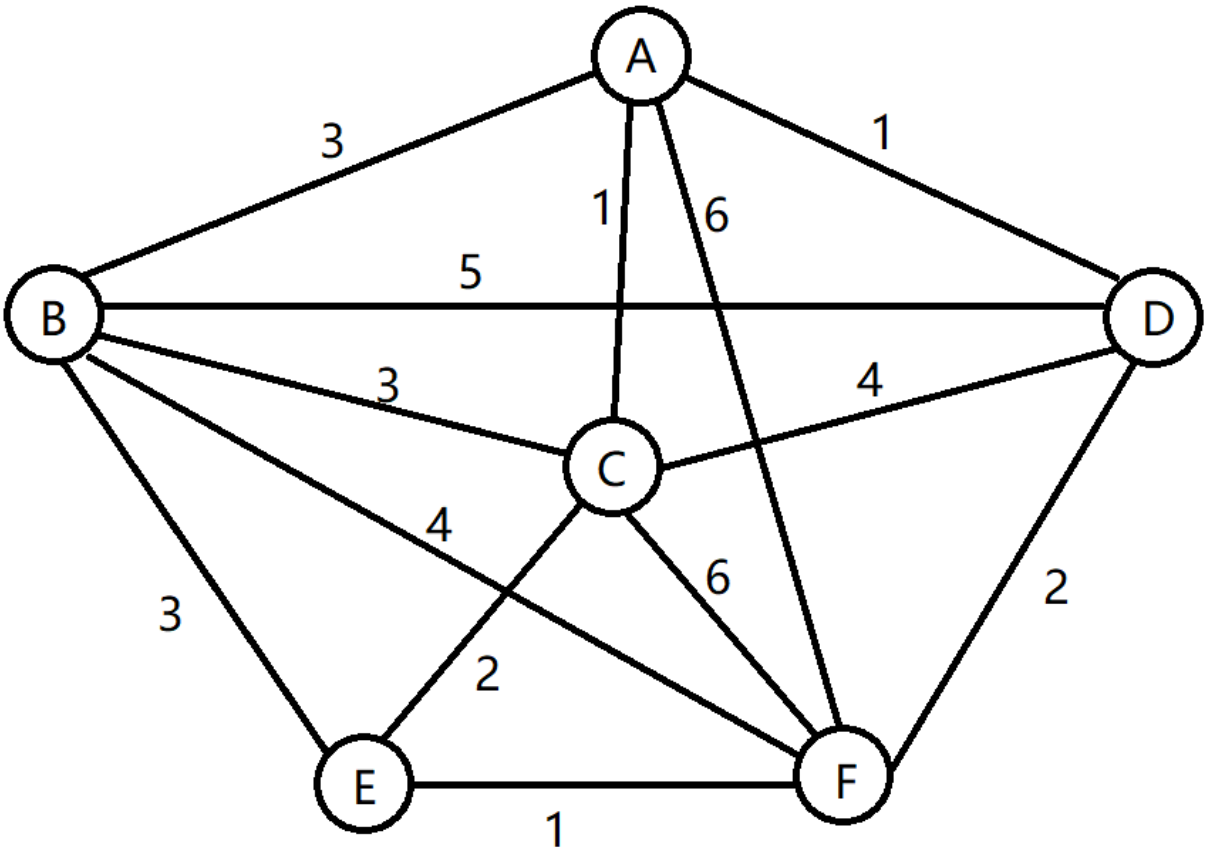
要求1：6台交换机之间必须能够互相进行通信，但是在通信过程中，可以经过其他交换机。例如交换机A向交换机B发送数据，可以经过交换机C的转发到达

要求2：省钱

那么也就是说，我们在布线的时候，一定要充分利用好“数据节点之间通信，可以经过其他交换机”这一条件

如果在两台交换机之间直接布线过于浪费成本，那么就想办法，通过其他交换机连接到这台交换机上，尽量节省布线成本

下面我们给出6个交换机节点之间现有布线布局的无向带权图



实际上，上面的问题，就可以使用无向带权图的最小生成树算法进行解决

下面，想让我们来了解一下什么是无向带权图的最小生成树问题

①最小生成树的概念

在之前我们曾经了解过：树是一种没有回路和环路的图

并且我们知道：只要我们对树结构进行遍历，就能够走遍树结构中的左右节点（比如二叉树的4种遍历方式）

并且，通过一张图所能够得到树结构的方案是有很多种的

那么在无向带权图中，我们为图中的每一条边都分配了权值

如果我们在删除一些边，去掉图中回路和环路之后保留下来的树结构中，所有节点之间边权的加和是所有方案中最小的

我们称这种树结构为无向图的最小生成树结构

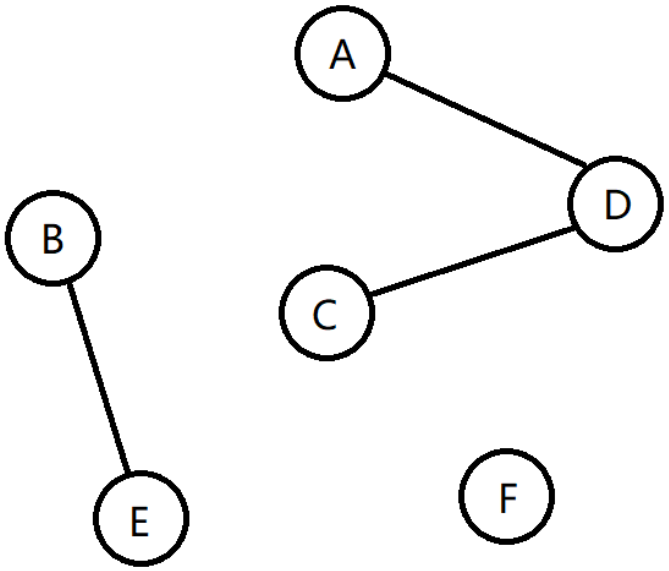
最小生成树中保留了原有图中所有的节点，这些节点保存在同一个树结构当中，并且这个数的所有边权之和如其他树结构相比，取值最小

在数据结构所涉及的相关算法中，有两种算法可以用来计算无向图的最小生成树：普里姆（Prim）算法和克鲁斯卡尔（Kruskal）算法

为了方便我们学习这两种算法，我们先来了解一下连通分量的概念

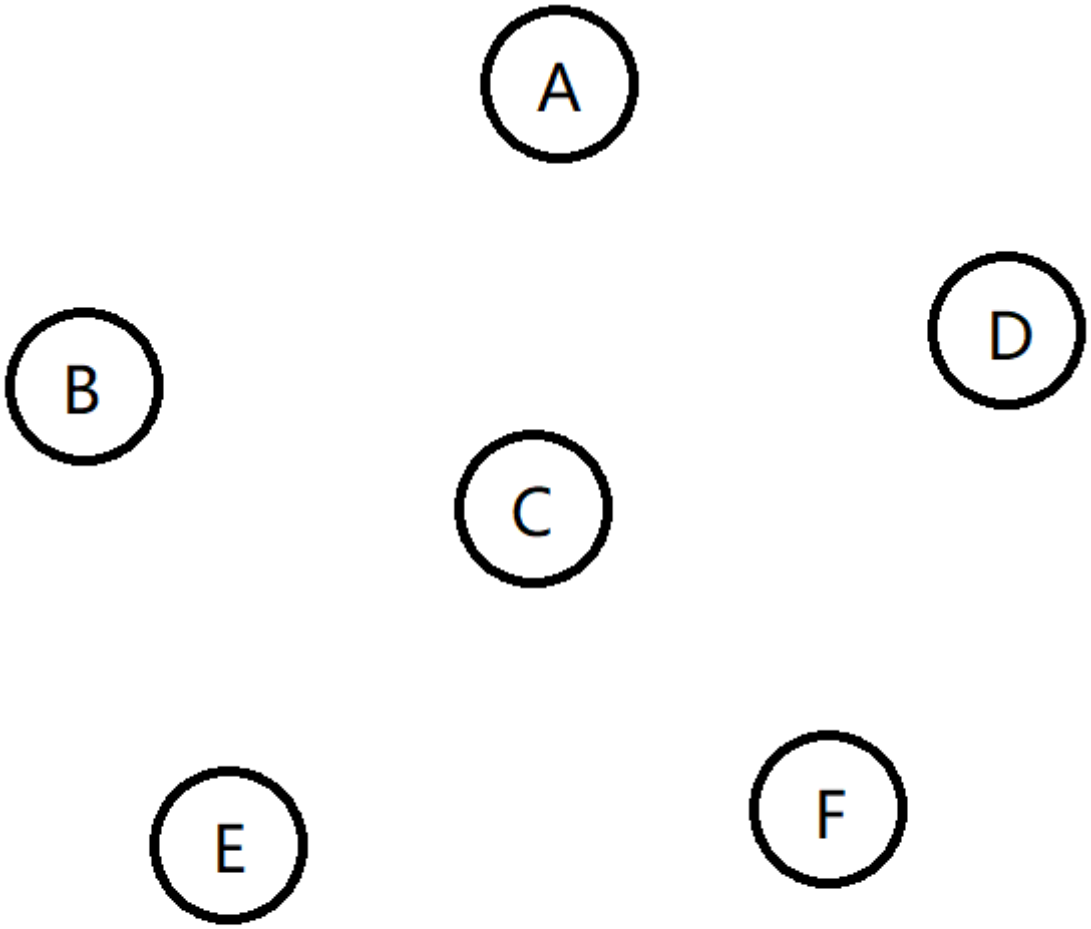
②连通分量的概念

我们将图中任意相连，且不与其他点相连的多个点之间，称之为一个连通分量



在这张图中：
节点ACD构成一个连通分量
节点BE构成一个连通分量
节点F自成一个连通分量
图中共有3个连通分量

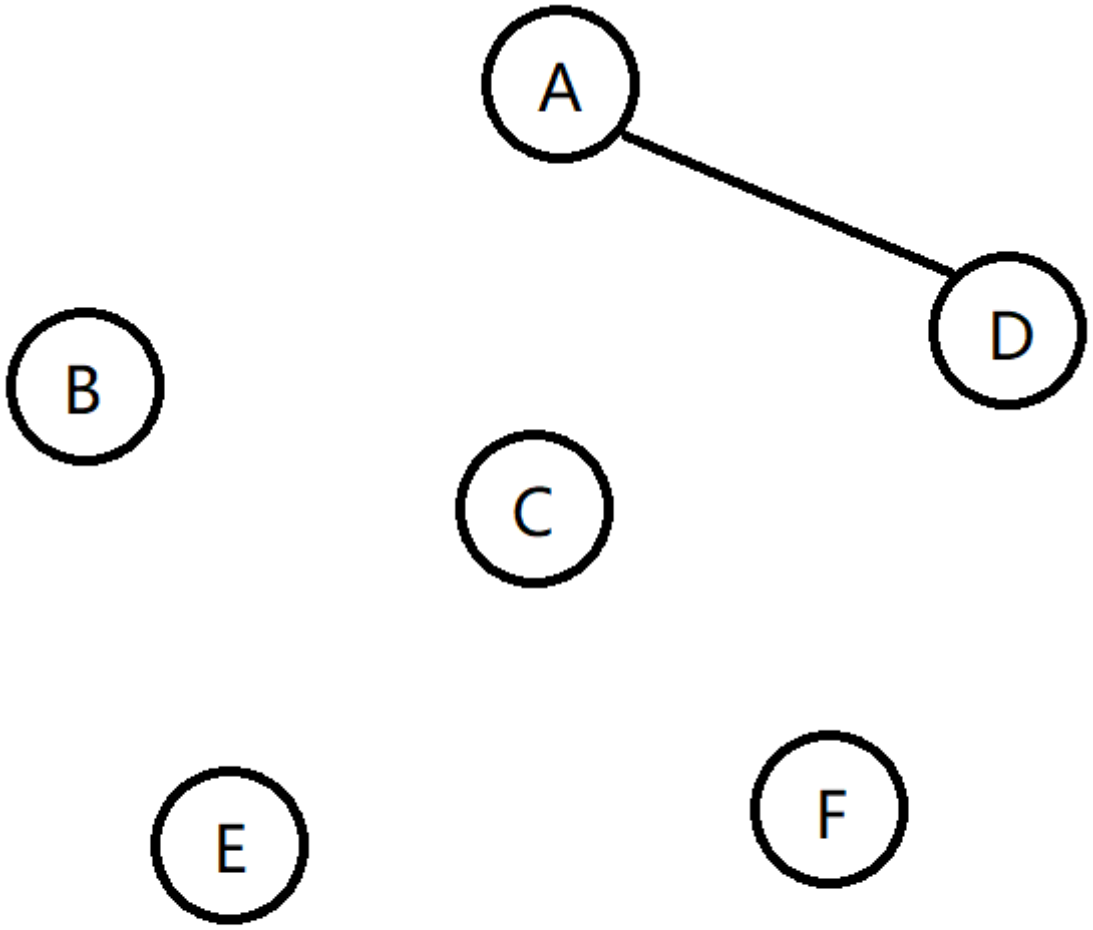
如果我们将一张图中所有的边都去掉，那么图中的每一个点都自成一个连通分量



图中所有节点都自成一个连通分量

图中共有6个连通分量

当我们使用一条边将其中的两个节点相连的时候，这两个点之间就构成了一个新的连通分量



图中连通分量总的个数就减去1个

在了解了连通分量的基础上，我们就可以来讨论普利姆算法和克鲁斯卡尔算法了

③普里姆算法

普里姆算法在开始之前，我们先将图中所有的边都“摘”出来，单独的放在一边，并且按照边权取值从小到大进行排序备用

此时图中所有的节点都是孤立的，自成一个连通分量，并建立一个变量，用来记录当前图中连通分量的总量

然后普里姆算法开始，算法的执行步骤如下：

步骤1：在所有的边中选择一个边权最小的边（如果具有相同最小权值的边有好几个，那么就随便选一个），使用这个边将两个节点连接起来，也就是将这个边加入生成树结构中

此时图中出现了一个包含两个节点的连通分量，我们暂且称之为当前最大连通分量，此时图中全部的连通分量的数量-1

步骤2：将上述步骤中用到的点，存放在一个集合中，这个集合称之为“用过的点的集合”（一会儿我们要用这个集合来判断新选择的边是不是会构成回路）

步骤3：查找和所有已经用过的点相关的边，在这些边中挑出一个权值最小的边，加入生成树结构中，

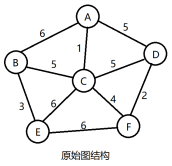
也就是从当前最大连通分量出发，通过当前最大连通分量中所有的节点，找出一个距离当前最大连通分量最近的点

但是在挑选的过程中我们需要保证：被选择的点，不能够出现在“用过的点的集合”中，因为如果出现这种情况，说明选择的边将使得图中出现回路，那就不是树结构了

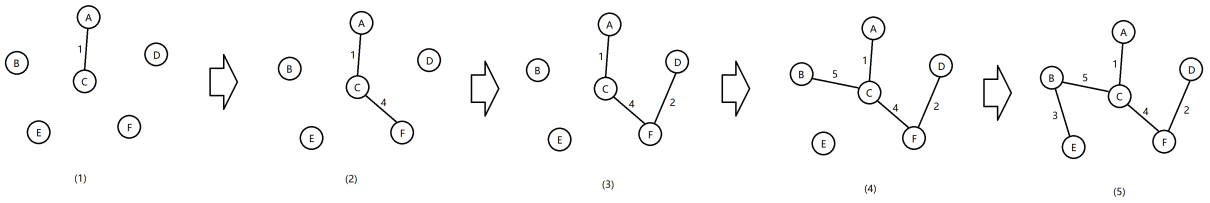
步骤4：将这个距离当前最大连通分量最近的点，同样加入“用过的点的集合”，图中连通分量数量-1

步骤5：重复步骤3-4，直到图中仅剩余1个连通分量的时候，说明所有的节点都已经加入最小生成树结构中了，程序终止

下面用一张图来表示上面的操作步骤：



原始图结构



实际上普里姆算法的思路和当年秦灭六国的思路是一样的：远交近攻！

从一座城开始为据点，找一个周边距离我最近的城，灭之，然后吞并成为大秦的国土！然后在之前国土和新吞并的国土的总和之上，再找一个离着最近的城，继续灭之！

最后，八荒荡尽，六合独尊，寰宇内外，唯我大秦！

但是在攻城略地的时候也需要注意，如果离着当前秦国国土最近的城，还是大秦自己的，那就不要攻打了！

这种情况在最小生成树中表现出来就是通过一条边，连接了一个已经存在于当前最大连通分量中的点，那就构成了回路，这是不行的！

那么我们是如何判断选择的边是否会导致构成回路呢？别忘了我们有“用过的点的集合”，如果一条边的两个端点都在这个集合中，那就说明这条边将会导致当前当前连通分量中出现回路

除了普里姆算法这种“由小做大，蚕食鲸吞”的创建最小生成树的方式之外，还有一种“各自为战，强强联合”的算法，那就是克鲁斯卡尔算法

④克鲁斯卡尔算法

正如上面所说的，计算图的最小生成树的算法还有一个，那就是克鲁斯卡尔算法

下面先让我们来看一下克鲁斯卡尔算法的执行步骤

在克鲁斯卡尔算法开始之前，我们同样将图中所有的边分别摘取出来单独放在一个集合中，并按照边权的大小进行排序

和普里姆算法不同的是，我们此时需要对所有的节点分别创建一个集合，每一个集合的长度都等于节点的数量，这些集合在最初的时候，都只保存一个不同的节点在其中

之后我们会使用这些集合，来判断一条边是否会导致生成树结构中产生回路的问题

然后同样创建一个变量， 用来记录当前图中连通分量的数量，最开始的时候，这个值等于图中节点的数量（因为此时图中的边已经全部摘除，所有的节点各自成为一个连通分量）

之后，克鲁斯卡尔算法开始：

步骤1：在边集合中，找出边权最小的一条边

步骤2：判断这条边的两个端点是否处在不同的两个集合中：

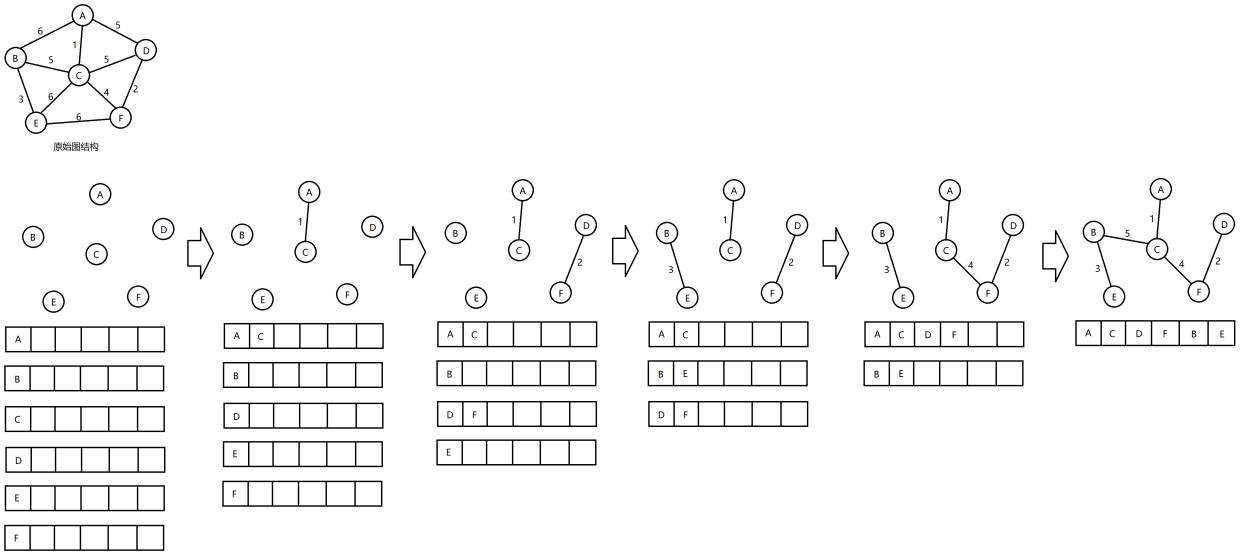
如果这条边的两个端点存在于同一个集合中，那么说明这条边的加入，将导致最小生成树结构产生回路，那么放弃这条边，重复执行步骤1

如果这条边的两个端点分别存在于两个不同的集合中，那么说明这条边的加入，可以沟通此时图中的两个不同的连通分量，并且不会导致回路的产生

步骤3：将选中的边所沟通的两个连通分量对应的节点合并到一个集合中，删除另一个空集合，并且图中连通分量的数量-1

步骤4：重复步骤1-3，直到图中仅剩余1个连通分量的时候，说明所有的节点都已经加入最小生成树结构中了，程序终止

下面用一张图来表示上面的操作步骤：



编者注：实际上上面的实现方案是我想到的一种用空间换时间的计算方式。我还想到一种使用节点二维数组表示节点间是否存在通路，也就是用时间换空间的算法，但是思路太麻烦了，在此略过不表

但是不管是哪种方法，都用到了动态规划算法的思想

克鲁斯卡尔算法的思想，则类似于中世纪的欧洲大陆，各个城邦之间各自为战，并在战争中不断互相合并，形成几个不相邻的几个小的帝国

起初，这些帝国之间并不相邻，就像最初的时候，图中形成的3个包含两个节点的连通分量之间，并不相互关联

但是随着战事的不断发展，帝国各自逐渐壮大，包含了更多的城邦的帝国之间也有了交集。

最终，各大帝国互相合并，形成更大的帝国，这就好比在图中，通过一个选中的边，将2个具有两个节点的连通分量沟通了起来

到最后，欧洲大陆统一，形成了一个完整的，城邦之间互相沟通的整体（实际上欧洲大陆根本没有统一过……）

但是在帝国之间进行合并的过程中还是要注意：如果帝国中的一座城市已经属于这个帝国了，那么就没有必要再去“合并”一次了

这就对应图中，如果一条边的两个端点正好处在同一个连通分量当中，那么如果再去加入这条边，就会构成生成树结构中的回路了

⑤总结

通过对上面两个算法的学习，我们发现：在图中边权取值重复率较低的情况下，普里姆算法和克鲁斯卡尔算法所计算得到的最小生成树结构，是能够互相验证的

但是不管是哪一种用来计算最小生成树的算法，实际上都使用了“贪心算法”（贪心算法我们将在后序的算法专题中进行讲解，敬请期待……）的思想：

普里姆算法是在构建一个当前最大连通分量的基础上，不断的贪距离这个当前最大连通分量最近的点；

而克鲁斯卡尔算法是通过不断贪当前图中，能够沟通两个不同连通分量的最小边来实现构建最小生成树结构的

最终，通过普利姆算法和克鲁斯卡尔算法得到的生成树结构中，边权相加的总和，一定比其他形式的生成树中边权相加的总和要小

所以经由这两种算法得到的图的生成树结构，称之为图的最小生成树