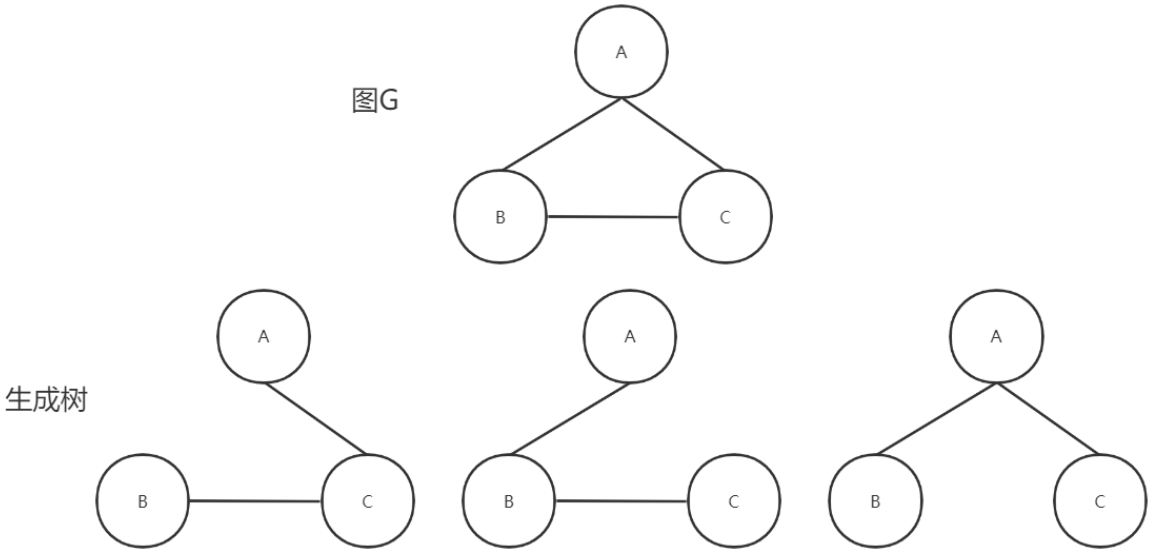
什么是最小生成树

我们今天，将讨论图中非常重要的一个概念，就是最小生成树。我们在讲图的概述的时候，讲过生成树的概念，从名字也可以看出来，和我们今天的主角---最小生成树有着密切关系，那我们先回顾一下生成树的概念。

生成树的定义

一个连通图的生成树是一个极小的连通子图，它包含图中全部的n个顶点，但只有构成一棵树的n-1条边。



可以看到一个包含三个顶点的完全图可以产生3颗生成树。对于包含n个顶点的无向完全图最多包含n的n-2次方颗生成树。

生成树的属性

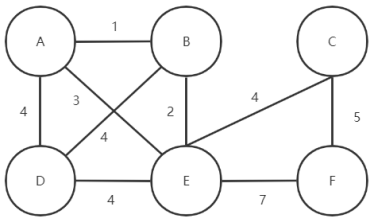
> - 一个连通图可以有多个生成树；
> - 一个连通图的所有生成树都包含相同的顶点个数和边数；
> - 生成树当中不存在环；
> - 移除生成树中的任意一条边都会导致图的不连通， 生成树的边最少特性；
> - 在生成树中添加一条边会构成环。
> - 对于包含n个顶点的连通图，生成树包含n个顶点和n-1条边；
> - 对于包含n个顶点的无向完全图最多包含 n^n-2颗生成树。

生成树的内容就这么一丢丢了，我相信大家已经完全掌握了，接下来就看我们的主角，最小生成树。

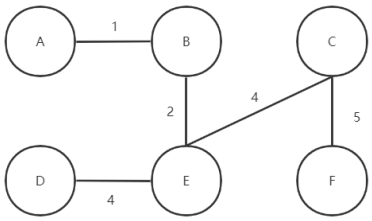
最小生成树

所谓一个 **带权图** 的最小生成树，就是原图中**边的权值最小的生成树** ，所谓最小是指边的权值之和小于或者等于其它生成树的边的权值之和。

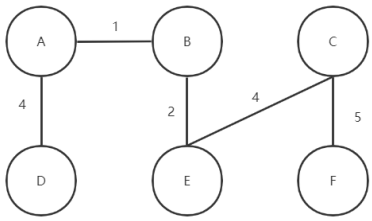
首先你明白最小生成树是和带权图联系在一起的；如果仅仅只是非带权的图，只存在生成树。其他的，我们看栗子解决就好了。



原图



最小生成树



最小生成树

上图中，原来的带权图可以生成左侧的两个最小生成树，这两颗最小生成树的权值之和最小，且包含原图中的所有顶点。

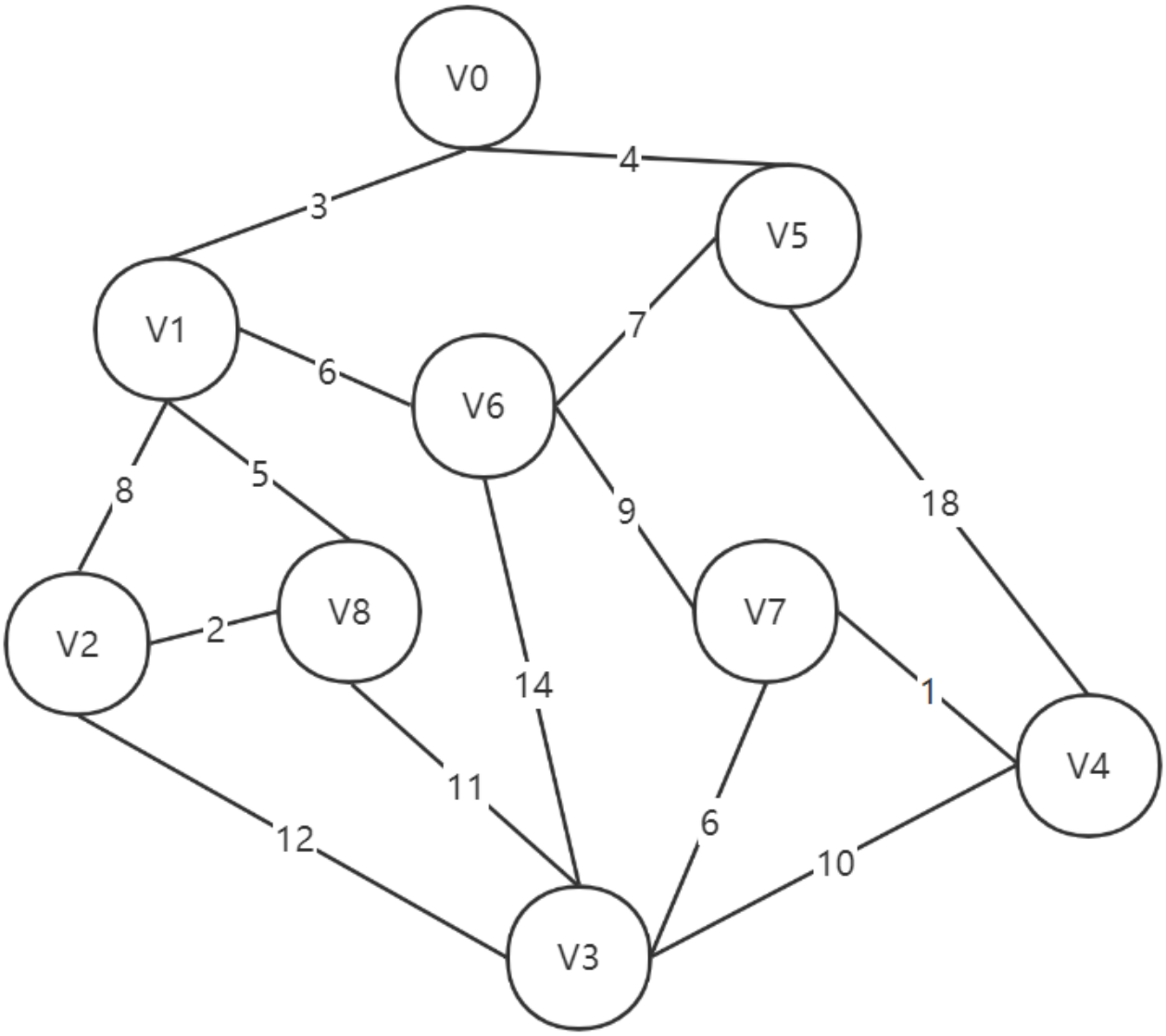
看图就是清晰，一下子理解了，但我们又如何从原图得到最小生成树呢?

这个确实是个问题，还是个好问题。最小生成树算法有很多，其中最经典的就是克鲁斯卡尔（Kruskal）算法和 普里姆（Prim）算法，也是我们考试、面试当中经常遇到的两个算法。

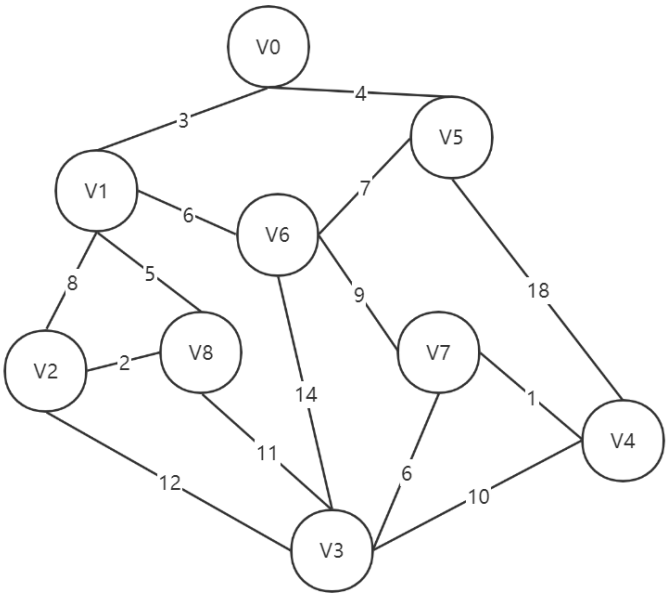
克鲁斯卡尔（Kruskal）算法

克鲁斯卡尔算法（Kruskal）是一种使用贪婪方法的最小生成树算法。该算法初始将图视为森林，图中的每一个顶点视为一棵单独的树。一棵树只与它的邻接顶点中权值最小且不违反最小生成树属性（不构成环）的树之间建立连边。

直接看例子：



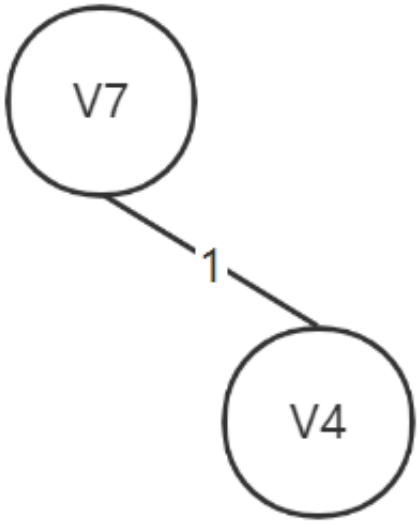
第一步：将图中所有的边按照权值进行非降序排列；



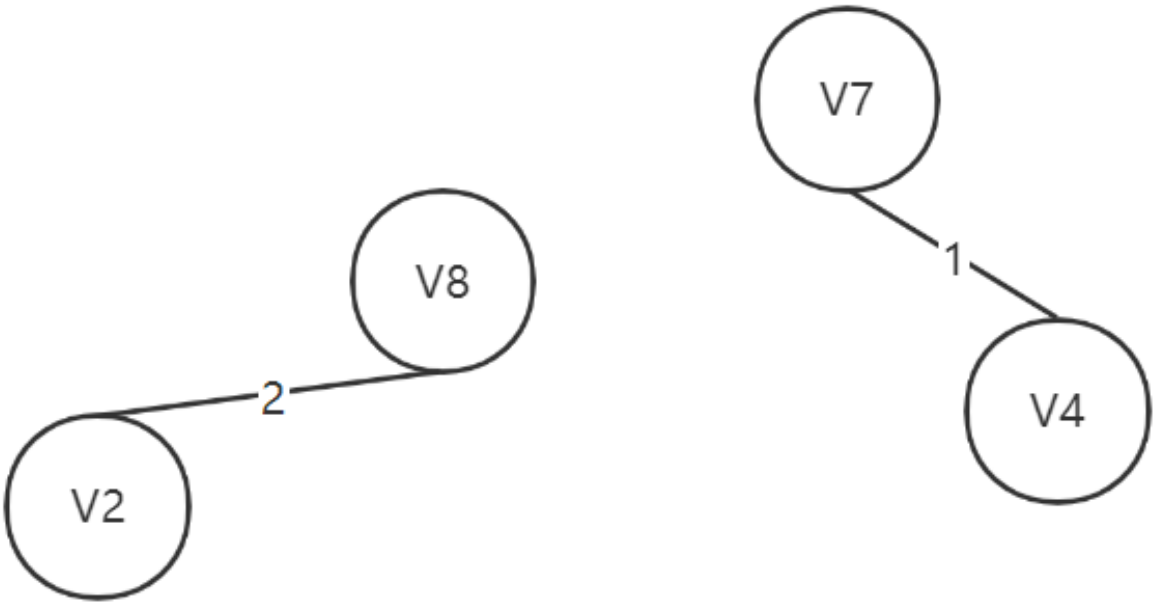
	begin	end	weight
edges[0]	4	7	1
edges[1]	2	8	2
edges[2]	0	1	3
edges[3]	0	5	4
edges[4]	1	8	5
edges[5]	3	7	6
edges[6]	1	6	6
edges[7]	5	6	7
edges[8]	1	2	8
edges[9]	6	7	9
edges[10]	3	4	10
edges[11]	3	8	11
edges[12]	2	3	12
edges[13]	3	6	14
edges[14]	4	5	18

第二步：从图中所有的边中选择可以构成最小生成树的边。

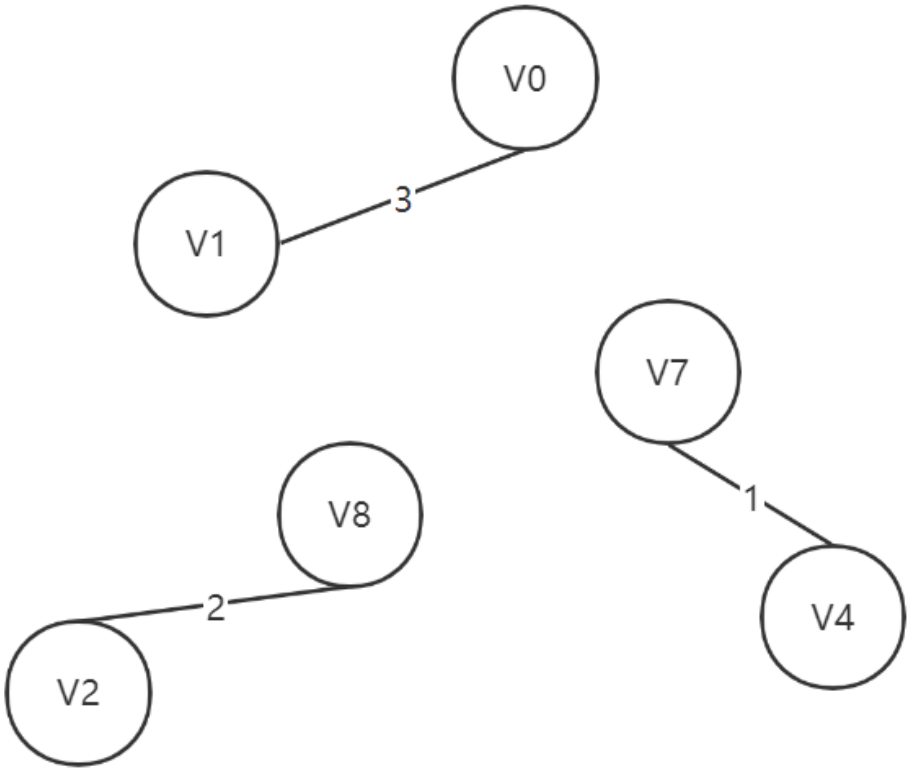
1、选择权值最小的边 V4-V7 ，没有环形成，则添加：



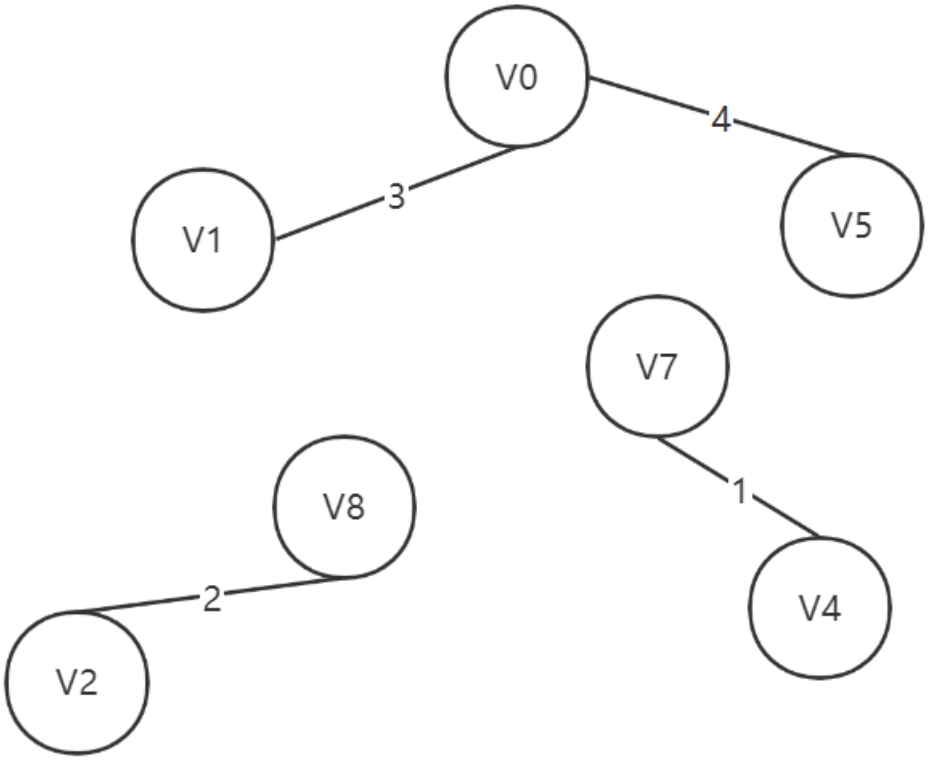
2、选择边 V2-V8，没有形成环，则添加：



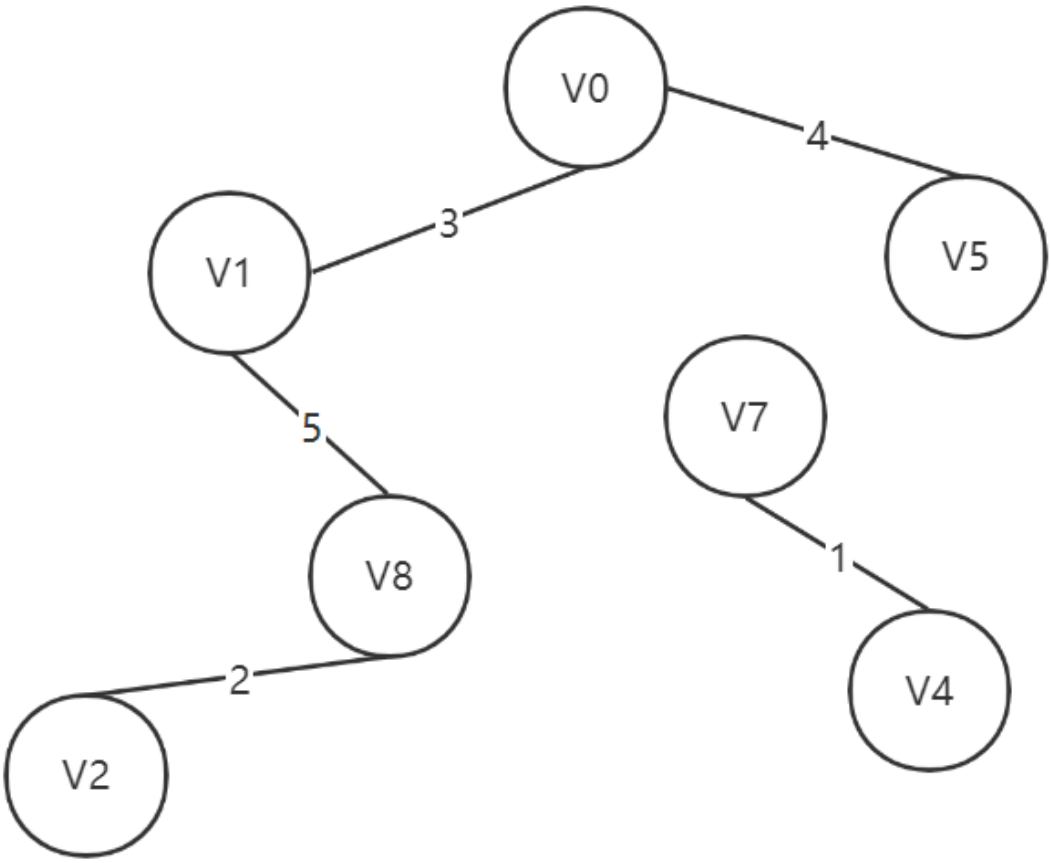
3、选择边V0 - V1没有形成环，则添加：



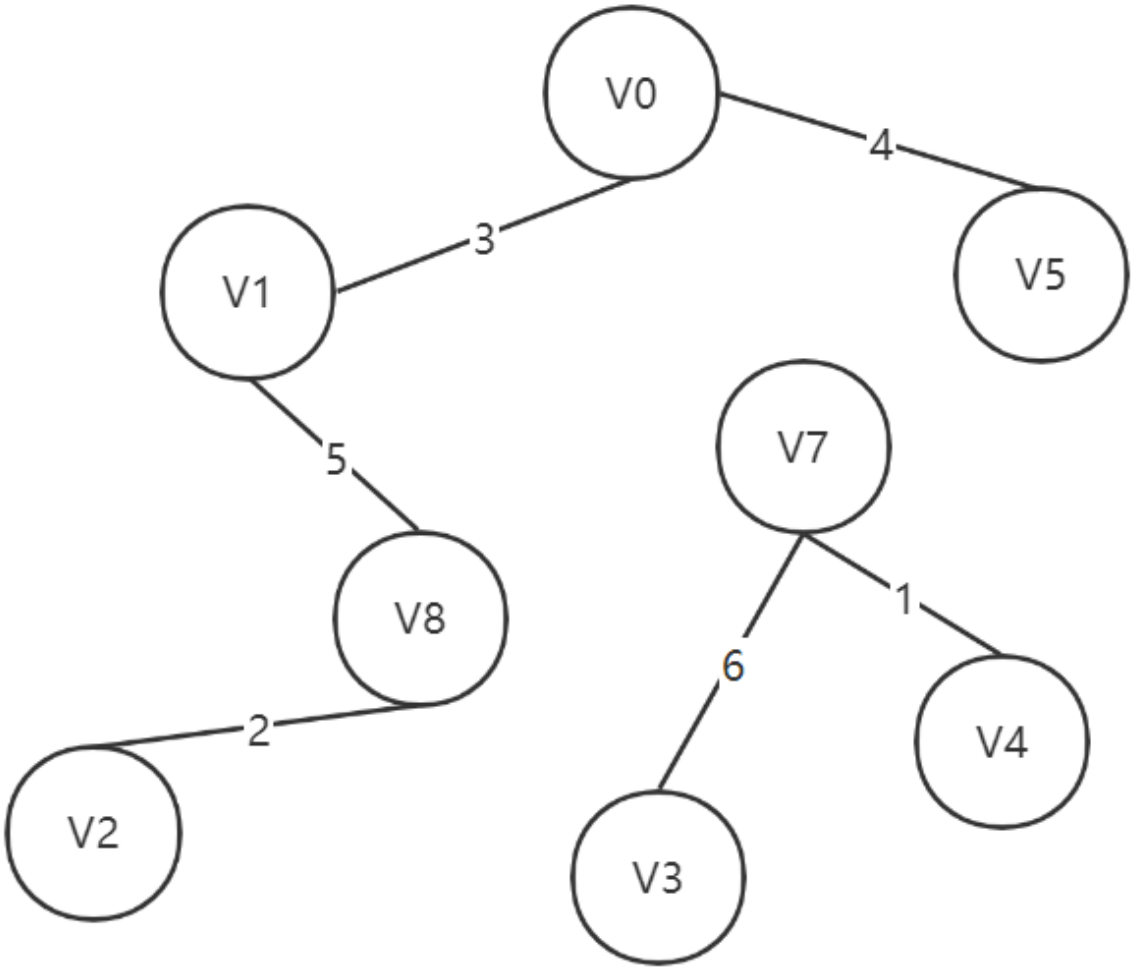
4、选择边V0 - V5，没有形成环，添加



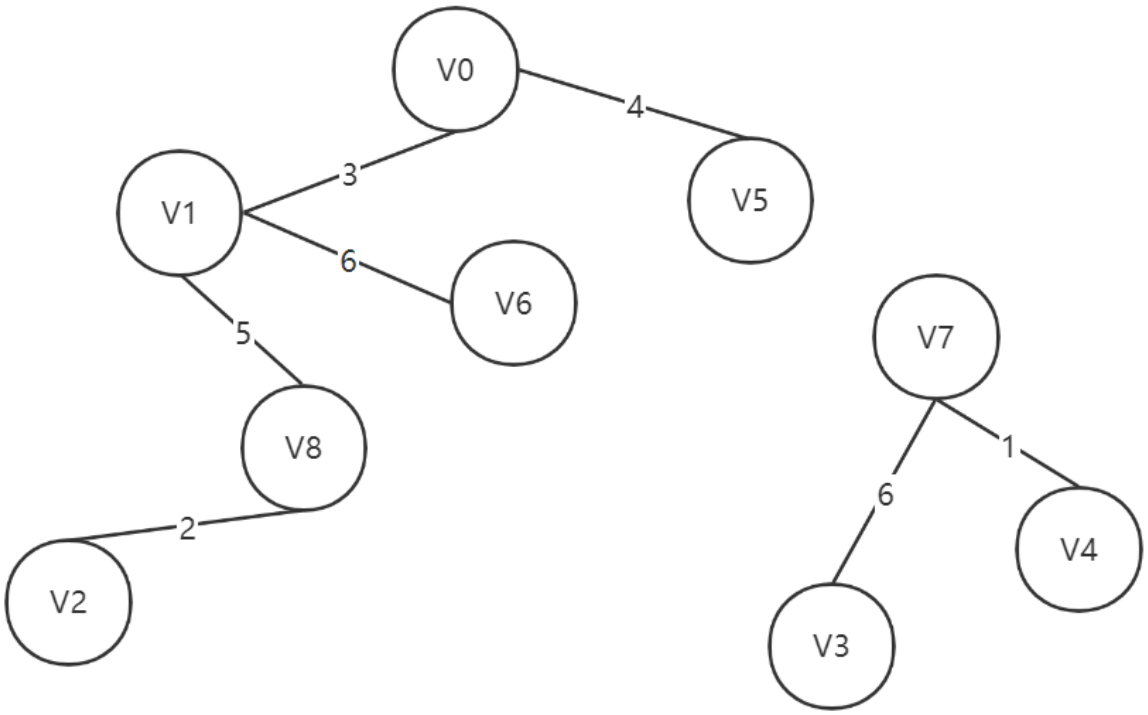
5、选择边V1 - V8，没有形成环，添加



6、选择V3 - V7，没有形成环，添加



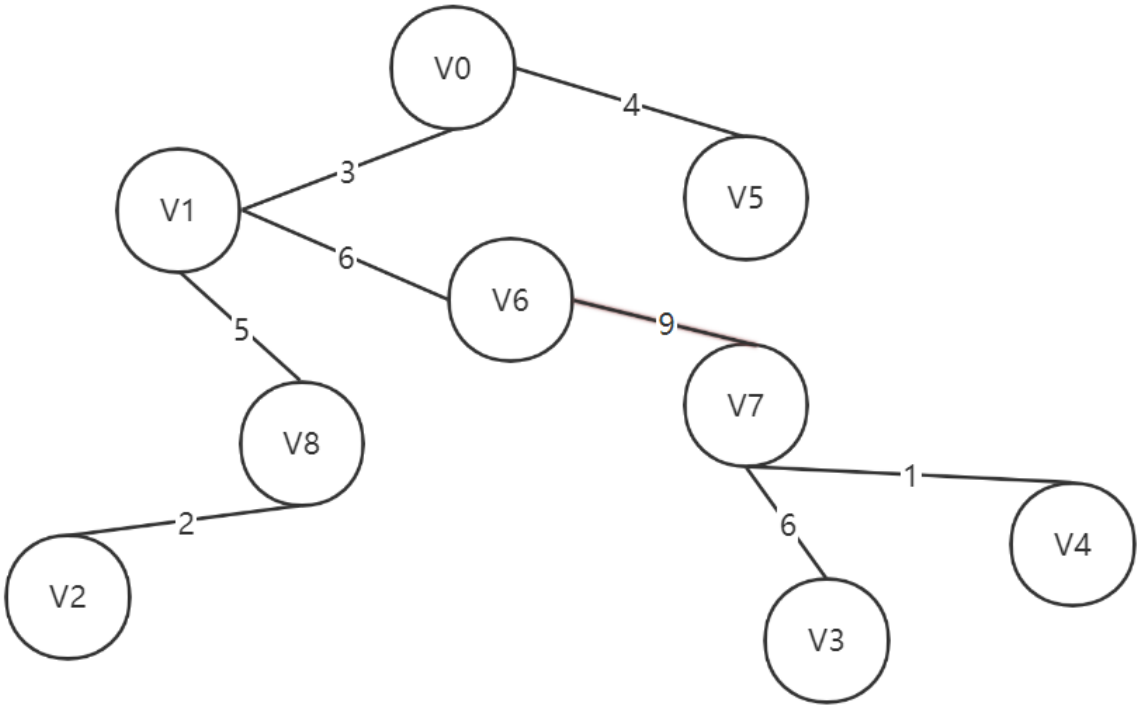
7、选择V1 - V6，没有形成环，添加



8、选择边V5 - V6添加这条边将导致形成环，舍弃，不添加；

9、选择边V1 - V2添加这条边将导致形成环，舍弃，不添加；

10、选择边V6 - V7，没有成环，添加



此时已经包含了图中顶点个数9减1条边，算法停止。

原来这样也可以，但我有一个问题，我们该如何判断添加一条边后是否形成环呢?

要判断添加一条边 X-Y 是否形成环，我们可以判断顶点X在最小树中的终点与顶点Y在最小生成树中的**终点是否相同**，如果相同则说明存在环路，否则不存环路，从而决定是否添加一条边。

所谓**终点**，就是将所有顶点按照从小到大的顺序排列好之后；某个顶点的终点就是"与它连通的最大顶点"。看下图，我们可以对图中顶点进行排序，排序后的顶点存放在一个数组中，每一个顶点则对应一个下标，同样的我们针对排序后的数组创建一个顶点的终点数组，初始时图中的每一个顶点是一棵树，每一个顶点的终点初始化为自身，我们用0来表示。

下标	0	1	2	3	4	5	6	7	8
数据	V0	V1	V2	V3	V4	V5	V6	V7	V8
终点	0	0	0	0	0	0	0	0	0

回到之前的算法执行过程，我们配合这个终点数组再来一次。

1、选择权值最小的边V4 - V7没有环形成（ V4的终点为4， V7的终点为7），则添加，并更新终点数组：

此时将4的终点更新为7；

下标	0	1	2	3	4	5	6	7	8
数据	V0	V1	V2	V3	V4	V5	V6	V7	V8
终点	0	0	0	0	7	0	0	0	0

2、选择权值最小的边V2 - V8没有环形成（V2的终点为2， V8的终点为8），则添加，并更新终点数组：

此时将2的终点更新为8；

下标	0	1	2	3	4	5	6	7	8
数据	V0	V1	V2	V3	V4	V5	V6	V7	V8
终点	0	0	8	0	7	0	0	0	0

3、选择权值最小的边V0 - V1没有环形成（V0的终点为0， V1的终点为1），则添加，并更新终点数组：

此时将2的终点更新为8；

下标	0	1	2	3	4	5	6	7	8
数据	V0	V1	V2	V3	V4	V5	V6	V7	V8
终点	1	0	8	0	7	0	0	0	0

4、选择权值最小的边V0 - V5没有环形成（V0的终点为1， V1的终点为5），则添加，并更新终点数组：

此时将1的终点更新为5；

下标	0	1	2	3	4	5	6	7	8
数据	V0	V1	V2	V3	V4	V5	V6	V7	V8
终点	1	5	8	0	7	0	0	0	0

5、选择权值最小的边V1 - V8没有环形成（V1的终点为5， V8的终点为8），则添加，并更新终点数组：

此时将5的终点更新为8；

下标	0	1	2	3	4	5	6	7	8
数据	V0	V1	V2	V3	V4	V5	V6	V7	V8
终点	1	5	8	0	7	8	0	0	0

6、选择权值最小的边V3 - V7 没有环形成（V3的终点为3， V7的终点为7），则添加，并更新终点数组：

此时将3的终点更新为7；

下标	0	1	2	3	4	5	6	7	8
数据	V0	V1	V2	V3	V4	V5	V6	V7	V8
终点	1	5	8	7	7	8	0	0	0

7、选择权值最小的边V1 - V6 没有环形成（V1的终点为8， V6的终点为6），则添加，并更新终点数组：

此时将8的终点更新为6；

下标	0	1	2	3	4	5	6	7	8
数据	V0	V1	V2	V3	V4	V5	V6	V7	V8
终点	1	5	8	7	7	8	0	0	6

8、选择权值最小的边V5 - V6 没有环形成（ V5的终点为6， V6的终点为6，两个顶点的终点相同，说明添加后会形成环），舍弃，不添加

9、选择权值最小的边V1 - V2 没有环形成（ V1的终点为6， V2的终点为6，两个顶点的终点相同，说明添加后会形成环），舍弃，不添加

10、选择权值最小的边V6 - V7没有环形成（ V6的终点为6， V7的终点为7），则添加，并更新终点数组：

此时将6的终点更新为7；

下标	0	1	2	3	4	5	6	7	8
数据	V0	V1	V2	V3	V4	V5	V6	V7	V8
终点	1	5	8	7	7	8	7	0	6

此时已经包含了图中顶点个数9减1条边，算法停止。

贪婪算法

从克鲁斯卡尔算法中，我们可以得出一个专门的算法思想，那就是贪心/贪婪算法。

贪心算法思想

顾名思义，贪心算法总是作出在当前看来最好的选择。也就是说贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的局部最优选择。当然，希望贪心算法得到的最终结果也是整体最优的。虽然贪心算法不能对所有问题都得到整体最优解，但对许多问题它能产生整体最优解。如单源最短路经问题，最小生成树问题等。在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似。

贪心算法的基本要素：

1、贪心选择性质。所谓贪心选择性质是指所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。这是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。

动态规划算法通常以自底向上的方式解各子问题（后面会讲），而贪心算法则通常以自顶向下的方式进行，以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题简化为规模更小的子问题。

对于一个具体问题，要确定它是否具有贪心选择性质，必须证明每一步所作的贪心选择最终导致问题的整体最优解。

2、当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。问题的最优子结构性质是该问题可用动态规划算法或贪心算法求解的关键特征。

贪心算法的基本思路

从问题的某一个初始解出发逐步逼近给定的目标，以尽可能快的地求得更好的解。当达到算法中的某一步不能再继续前进时，算法停止。

该算法存在问题：

1、不能保证求得的最后解是最佳的；

2、不能用来求最大或最小解问题；

3、只能求满足某些约束条件的可行解的范围。

实现该算法的过程：

从问题的某一初始解出发；

while 能朝给定总目标前进一步 do

求出可行解的一个解元素；

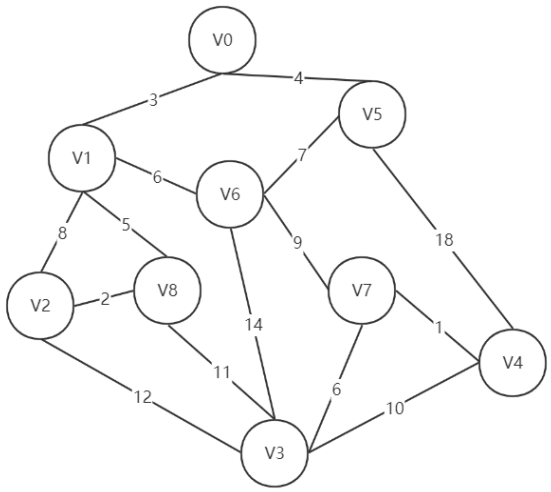
由所有解元素组合成问题的一个可行解。

普里姆（Prim）算法

普里姆算法在找最小生成树时，将顶点分为两类，一类是在查找的过程中已经包含在生成树中的顶点（假设为 A 类），剩下的为另一类（假设为 B 类）。

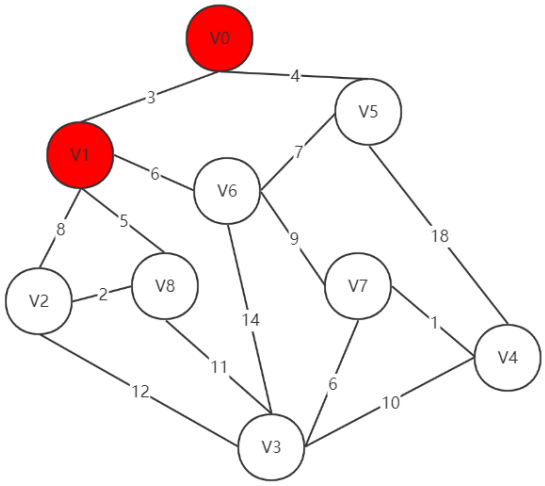
对于给定的连通网，起始状态全部顶点都归为 B 类。在找最小生成树时，选定任意一个顶点作为起始点，并将之从 B 类移至 A 类；然后找出 B 类中到 A 类中的顶点之间权值最小的顶点，将之从 B 类移至 A 类，如此重复，直到 B 类中没有顶点为止。所走过的顶点和边就是该连通图的最小生成树。

举个栗子：



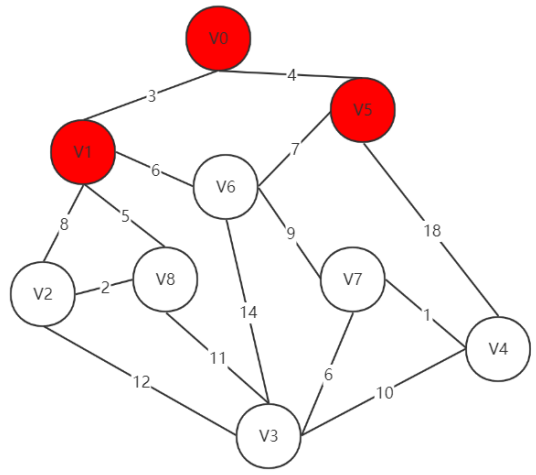
	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	3	∞	∞	∞	4	∞	∞	∞
V1	3	0	8	∞	∞	∞	6	∞	5
V2	∞	8	0	12	∞	∞	∞	∞	2
V3	∞	∞	12	0	10	∞	14	6	11
V4	∞	∞	∞	10	0	18	∞	1	∞
V5	4	∞	∞	∞	18	0	7	∞	∞
V6	∞	6	∞	14	∞	7	0	9	∞
V7	∞	∞	∞	6	1	∞	9	0	∞
V8	∞	5	2	11	∞	∞	∞	∞	0

假如从顶点 V0 出发，顶点V1 、V5的权值分别为3、4，所以对于顶点V0来说，到顶点V1的权值最小，将顶点V1加入到生成树中：



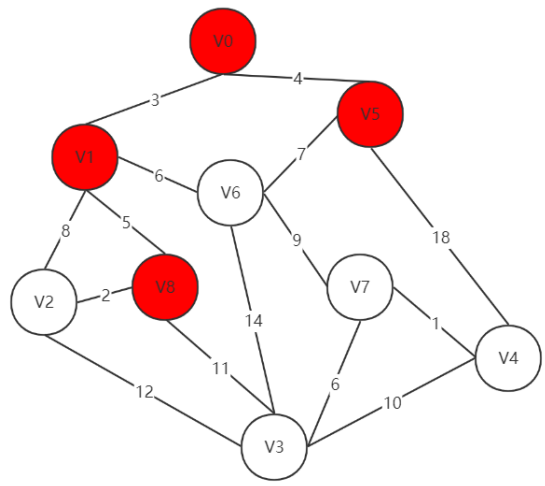
	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	3	∞	∞	∞	4	∞	∞	∞
V1	3	0	8	∞	∞	∞	6	∞	5
V2	∞	8	0	12	∞	∞	∞	∞	2
V3	∞	∞	12	0	10	∞	14	6	11
V4	∞	∞	∞	10	0	18	∞	1	∞
V5	4	∞	∞	∞	18	0	7	∞	∞
V6	∞	6	∞	14	∞	7	0	9	∞
V7	∞	∞	∞	6	1	∞	9	0	∞
V8	∞	5	2	11	∞	∞	∞	∞	0

继续分析与顶点V0和V1相邻的所有顶点（包括V2 、V5、V6、V8 ），其中权值最小的为V5， 将V5添加到生成树当中：



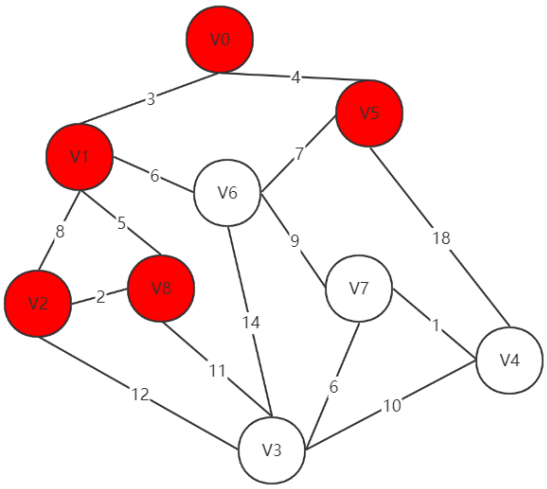
	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	3	∞	∞	∞	4	∞	∞	∞
V1	3	0	8	∞	∞	∞	6	∞	5
V2	∞	8	0	12	∞	∞	∞	∞	2
V3	∞	∞	12	0	10	∞	14	6	11
V4	∞	∞	∞	10	0	18	∞	1	∞
V5	4	∞	∞	∞	18	0	7	∞	∞
V6	∞	6	∞	14	∞	7	0	9	∞
V7	∞	∞	∞	6	1	∞	9	0	∞
V8	∞	5	2	11	∞	∞	∞	∞	0

继续分析与顶点V0和V1、V5相邻的所有顶点中权值最小的顶点，发现为V8，则添加到生成树当中。



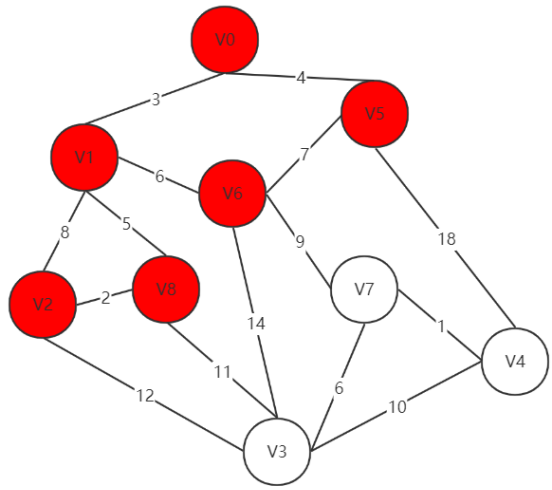
	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	3	∞	∞	∞	4	∞	∞	∞
V1	3	0	8	∞	∞	∞	6	∞	5
V2	∞	8	0	12	∞	∞	∞	∞	2
V3	∞	∞	12	0	10	∞	14	6	11
V4	∞	∞	∞	10	0	18	∞	1	∞
V5	4	∞	∞	∞	18	0	7	∞	∞
V6	∞	6	∞	14	∞	7	0	9	∞
V7	∞	∞	∞	6	1	∞	9	0	∞
V8	∞	5	2	11	∞	∞	∞	∞	0

继续分析与生成树中已添加顶点相邻的顶点中权值最小的顶点，发现为V2，则添加到生成树中：

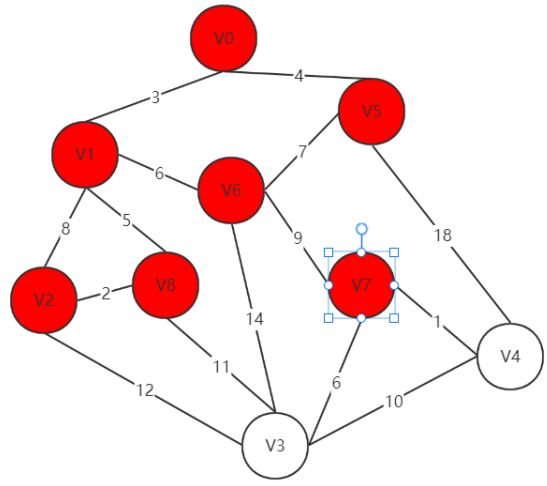


	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	3	∞	∞	∞	4	∞	∞	∞
V1	3	0	8	∞	∞	∞	6	∞	5
V2	∞	8	0	12	∞	∞	∞	∞	2
V3	∞	∞	12	0	10	∞	14	6	11
V4	∞	∞	∞	10	0	18	∞	1	∞
V5	4	∞	∞	∞	18	0	7	∞	∞
V6	∞	6	∞	14	∞	7	0	9	∞
V7	∞	∞	∞	6	1	∞	9	0	∞
V8	∞	5	2	11	∞	∞	∞	∞	0

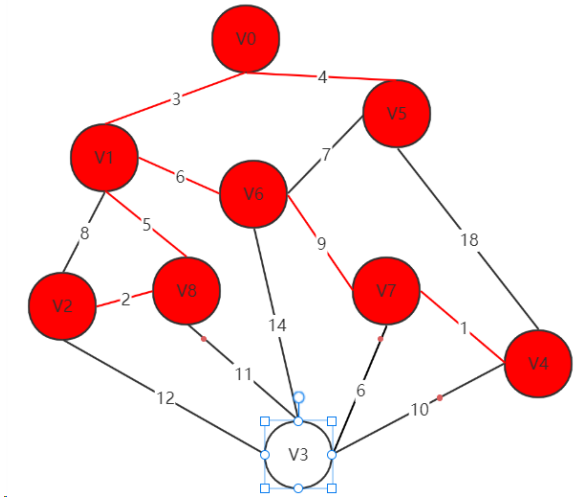
重复上面的过程，直到生成树中包含图中的所有顶点，我们直接看接下来的添加过程：



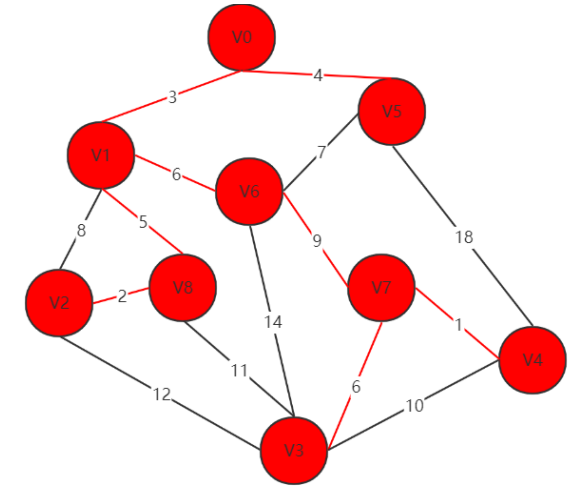
	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	3	∞	∞	∞	4	∞	∞	∞
V1	3	0	8	∞	∞	∞	6	∞	5
V2	∞	8	0	12	∞	∞	∞	∞	2
V3	∞	∞	12	0	10	∞	14	6	11
V4	∞	∞	∞	10	0	18	∞	1	∞
V5	4	∞	∞	∞	18	0	7	∞	∞
V6	∞	6	∞	14	∞	7	0	9	∞
V7	∞	∞	∞	6	1	∞	9	0	∞
V8	∞	5	2	11	∞	∞	∞	∞	0



	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	3	∞	∞	∞	4	∞	∞	∞
V1	3	0	8	∞	∞	∞	6	∞	5
V2	∞	8	0	12	∞	∞	∞	∞	2
V3	∞	∞	12	0	10	∞	14	6	11
V4	∞	∞	∞	10	0	18	∞	1	∞
V5	4	∞	∞	∞	18	0	7	∞	∞
V6	∞	6	∞	14	∞	7	0	9	∞
V7	∞	∞	∞	6	1	∞	9	0	∞
V8	∞	5	2	11	∞	∞	∞	∞	0



	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	3	∞	∞	∞	4	∞	∞	∞
V1	3	0	8	∞	∞	∞	6	∞	5
V2	∞	8	0	12	∞	∞	∞	∞	2
V3	∞	∞	12	0	10	∞	14	6	11
V4	∞	∞	∞	10	0	18	∞	1	∞
V5	4	∞	∞	∞	18	0	7	∞	∞
V6	∞	6	∞	14	∞	7	0	9	∞
V7	∞	∞	∞	6	1	∞	9	0	∞
V8	∞	5	2	11	∞	∞	∞	∞	0



	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	3	∞	∞	∞	4	∞	∞	∞
V1	3	0	8	∞	∞	∞	6	∞	5
V2	∞	8	0	12	∞	∞	∞	∞	2
V3	∞	∞	12	0	10	∞	14	6	11
V4	∞	∞	∞	10	0	18	∞	1	∞
V5	4	∞	∞	∞	18	0	7	∞	∞
V6	∞	6	∞	14	∞	7	0	9	∞
V7	∞	∞	∞	6	1	∞	9	0	∞
V8	∞	5	2	11	∞	∞	∞	∞	0

总结

最小生成树的问题，简单得理解就是给定一个带有权值的连通图（连通网），从众多的生成树中筛选出权值总和最小的生成树，即为该图的最小生成树。

最经典的两个最小生成树算法：Kruskal 算法与 Prim 算法。两者分别从不同的角度构造最小生成树，Kruskal 算法从边的角度出发，使用贪心的方式选择出图中的最小生成树，而 Prim 算法从顶点的角度出发，逐步找各个顶点上最小权值的边来构建最小生成树的。

最小生成树问题应用广泛，最直接的应用就是网线架设（网络G表示n各城市之间的通信线路网线路（其中顶点表示城市，边表示两个城市之间的通信线路，边上的权值表示线路的长度或造价。可通过求该网络的最小生成树达到求解通信线路或总代价最小的最佳方案。或者如果我们需要用最少的电线给一所房子安装电路。）、道路铺设。还可以间接应用于纠错的LDPC码、Renyi 熵图像配准、学习用于实时脸部验证的显著特征、减少蛋白质氨基酸序列测序中的数据存储，在湍流（turbulent）中模拟粒子交互的局部性，以及用于以太网桥接的自动配置，以避免在网络中形成环路。除此之外，最小生成树在聚类算法中也是应用广泛。

所以一定要搞懂最小生成树、Kruskal 算法及Prim 算法奥！！！