最短路径

最短路径之迪杰斯特拉算法（Dijkstra Algorithm）

今天我们主要看一个最短路径算法，迪杰斯特拉算法（Dijkstra Algorithm）。

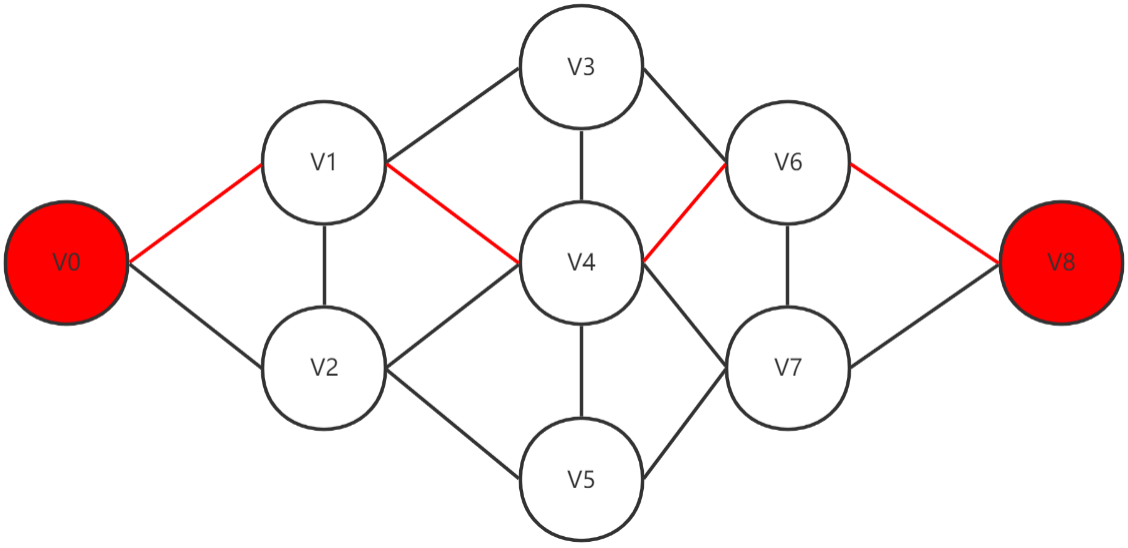
这个算法的主要运用就是计算从某个源点开始到其他顶点的最短路径的算法。

什么是最短路径，什么又是源点，还有最短路径算法有啥子用呢？我们来一个一个的看

基础概念

什么是源点?

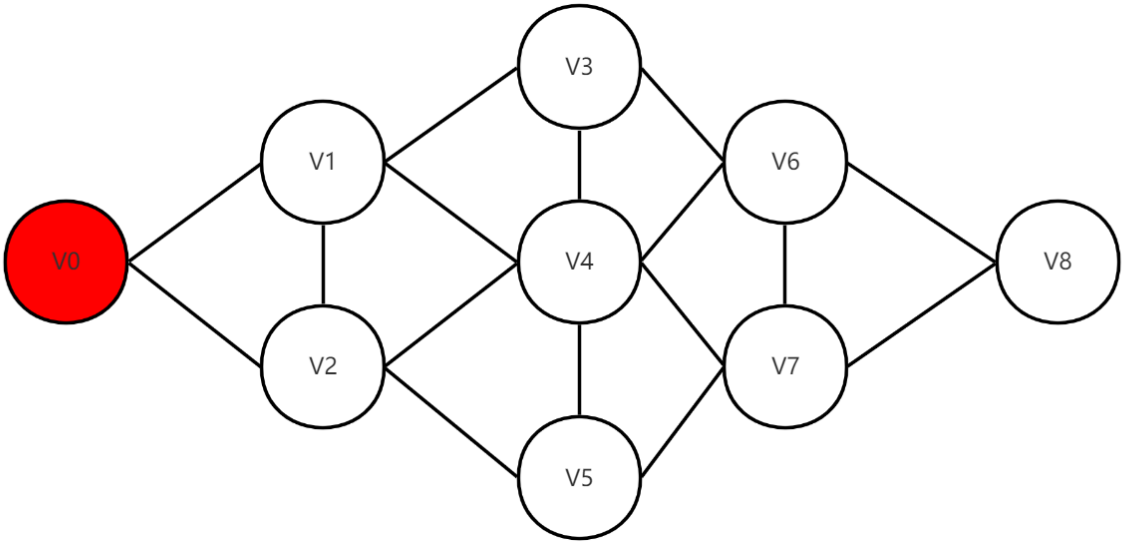
路径起始的第一个顶点称为源点（Source），最后一个顶点称为终点（Destination）。图下图中，我们用红色标注出的就可以认为是一个路径（V0 ->V1 ->V4 ->V6 ->V8）的源点和终点，但不要有误区，其实图中的任何一个顶点都可作为源点或者终点，源点与终点只是相对一条路径而言的。



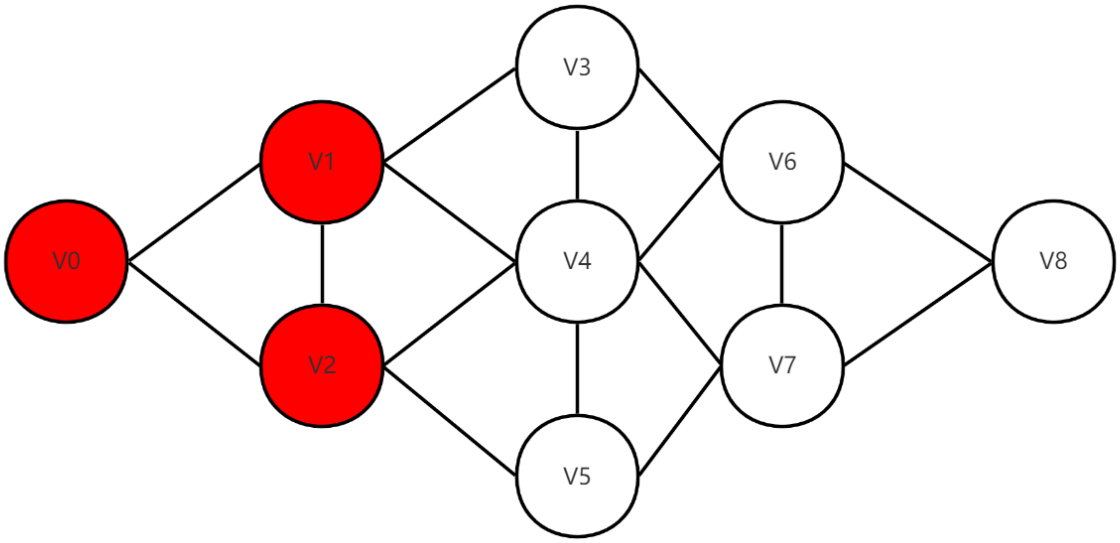
什么是最短路径

对于无向图而言，从源点V0到终点V8的最短路径就是从源点V0到终点V8所包含的边最少的路径。我们只需要从源点V0出发对图做广度优先搜索，一旦遇到终点V8就终止。我们可以具体来看看如何得到无向图中源点V0到终点V8的最短路径。

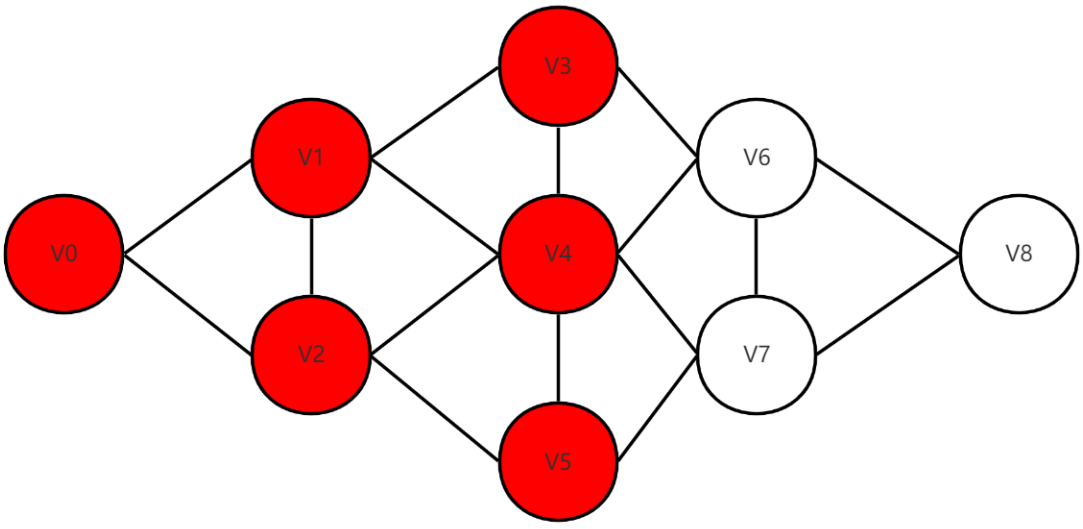
第一步：遍历顶点V0：



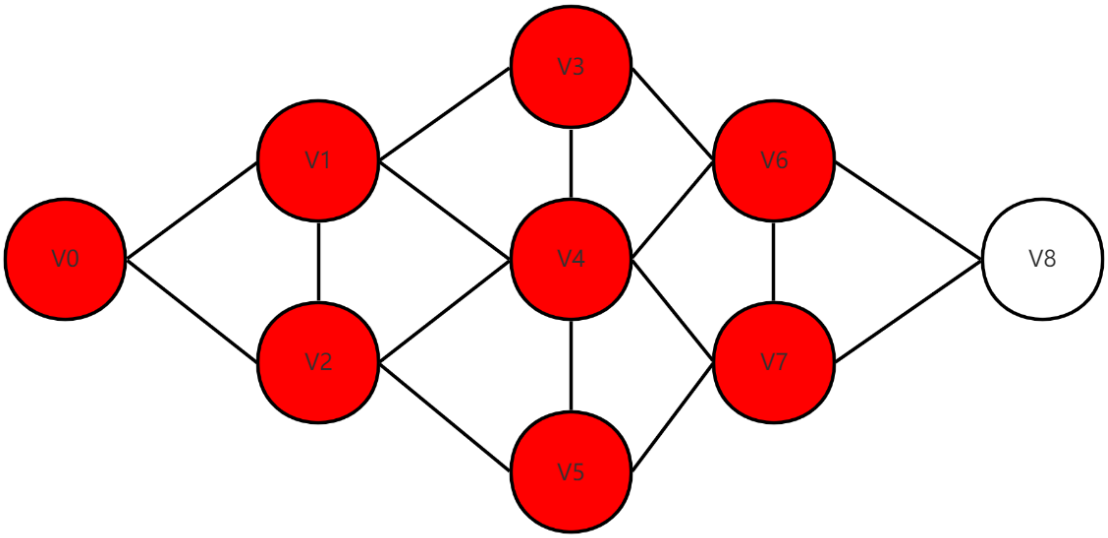
第二步：遍历顶点V0的邻接顶点V1和V2（具体操作中我们使用队列来进行实现）：



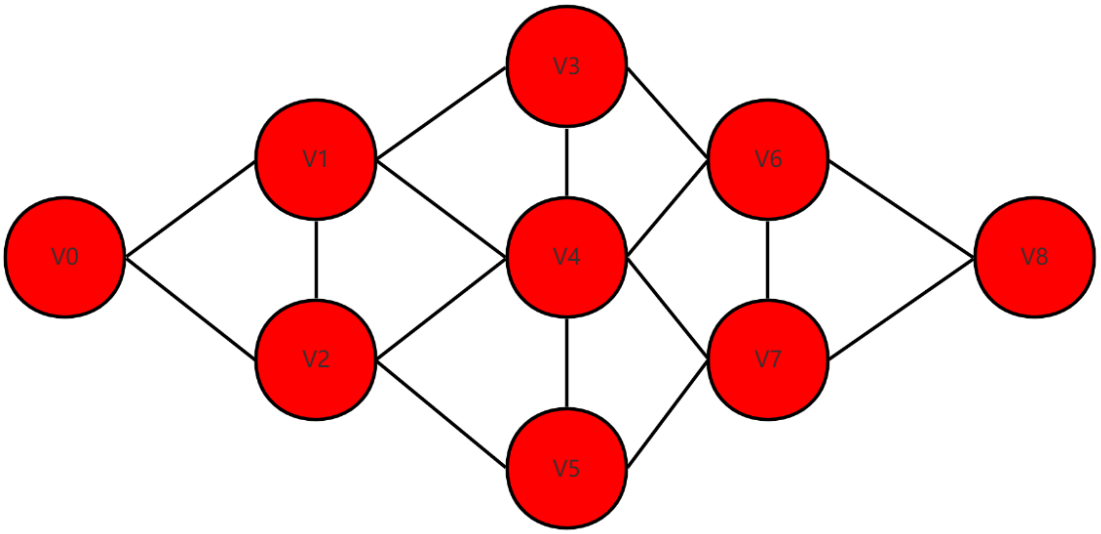
第三步：遍历顶点V1的邻接顶点V3和V4，遍历顶点V2的邻接顶点V5：



第四步：遍历顶点V3的邻接顶点V6和V4的邻接顶点V7：



第五步：遍历顶点V6的邻接顶点V8，发现正好是终点



由此可以得到，图中从源点V0到终点V8的（第一条）一条最短路径 （V0 ->V1 ->V3 ->V6 ->V8 ）。

最短路径的用处有哪些

简单来说，我们要从大兴机场到北京天安门，如何规划路线才能换乘最少，并且耗时最少呢？这时候，最短路径算法就派上用场了，你每天使用的导航系统进行道路规划也同样依赖于底层的算法！虽然现实情况可能更复杂一些，但是学习最基础的算法对于我们日后的提升总有莫大的帮助。我们接着看今天的主角：迪杰斯特拉算法。

迪杰斯特拉算法

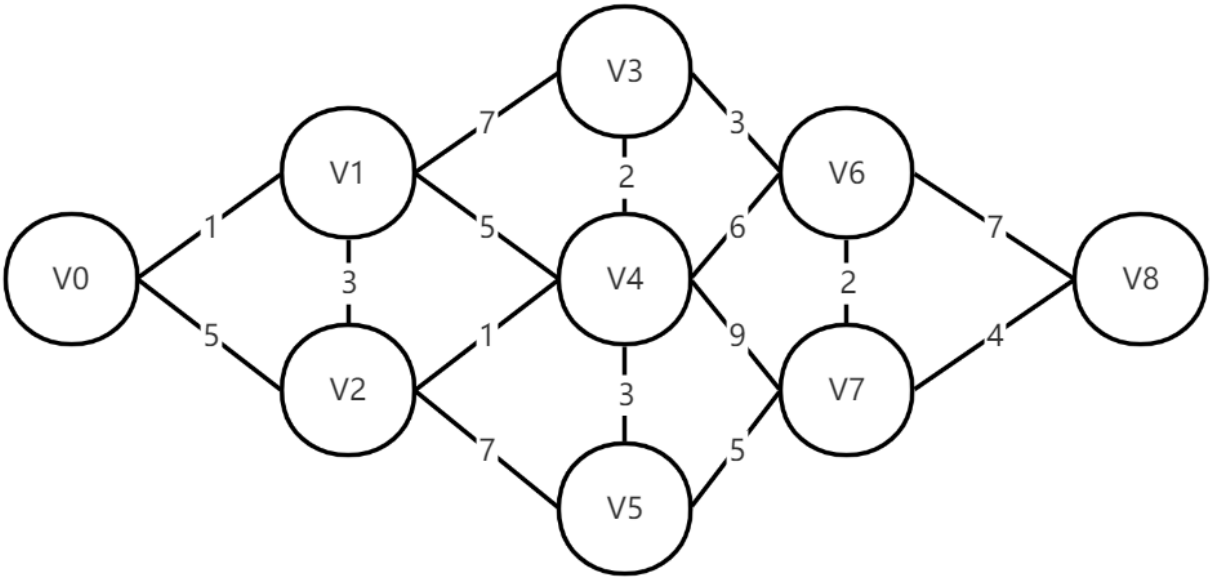
迪杰斯特拉算法是一个单源点的一个最短路径算法，也就是说，我们这个算法会求得从一个顶点到其所有顶点的最短路径。

首先，引入一个辅助变量D，它的每一个分量D[i]表示当前所找到的从源点v到每一个顶点Vi的最短路径的长度，它的初始状态为：若从V到Vi有弧，则D[i]为弧上的权值。否则D[i]为无穷大。显然，长度就为：

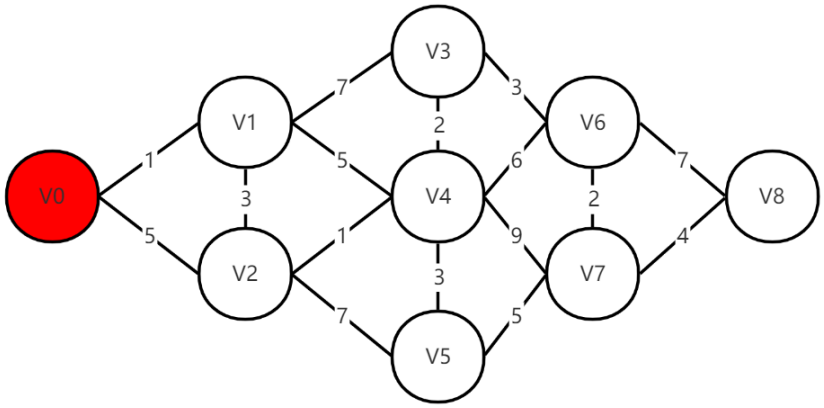
$$D[i] = Min_j\{D[i] \mid v_i \in V\}$$

的路径就是从 v 出发的长度最短的一条最短路径。此路径为（V，Vj） 。

我们直接看例子，以下图为例：



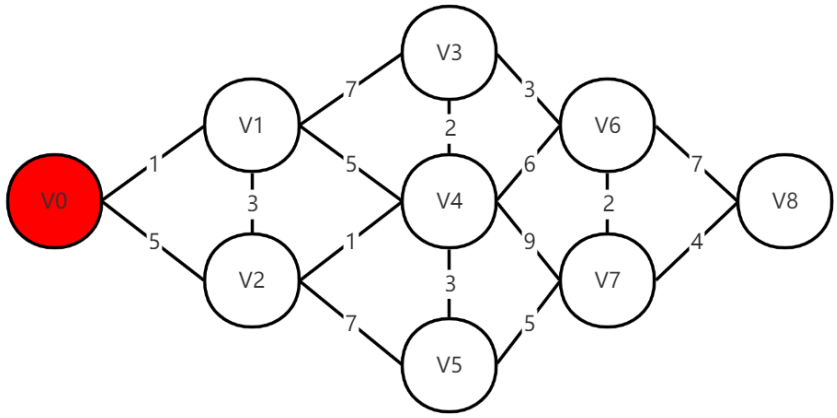
第一步：第一步：初始化辅助向量D，路径向量P 和 当前已求得顶点V0到Vj的最短路径 的向量Final。



	D	P	Final
V0	0	0	1
V1	1	0	0
V2	5	0	0
V3	∞	0	0
V4	∞	0	0
V5	∞	0	0
V6	∞	0	0
V7	∞	0	0
V8	∞	0	0

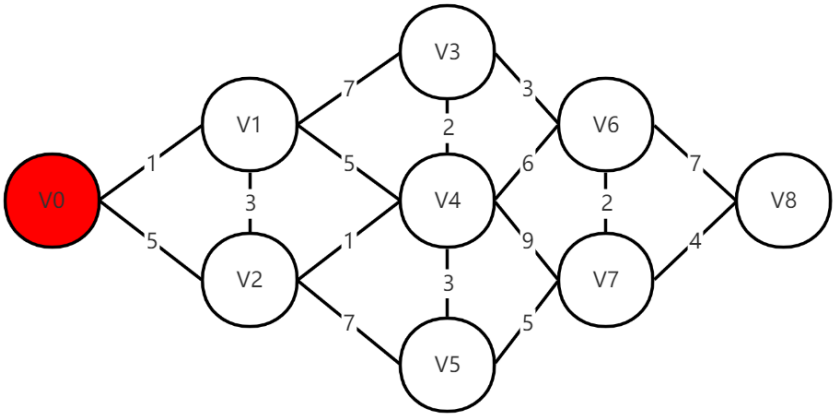
辅助向量D的初态为：若从V0到Vj有弧，则D[j]为弧上的权值；否则D[j]为无穷大（INF）。对应到图中， V0到V1有弧，则D[1] = 1 ； V0 到V2有弧，则D[2] = 5 ； 到其他顶点没有弧，则相应的用无穷（INF）表示。路径向量P用于存储最短路径下标的数组，初始时全部置为零；向量Final中值为1表示顶点V0到Vj的最短路径已求得，V0到V0的最短路径当然是已求得，所以将Final[V0]设置为1。接下来就是迪杰斯特拉算法的核心了，认真看奥。

第二步：遍历源点V0, 找到源点V0的邻接顶点中距离最短的顶点，即V1， V0到V1的最短路径为1已经求出，更新Final[1] = 1 。



	D	P	Final
V0	0	0	1
V1	1	0	1
V2	5	0	0
V3	∞	0	0
V4	∞	0	0
V5	∞	0	0
V6	∞	0	0
V7	∞	0	0
V8	∞	0	0

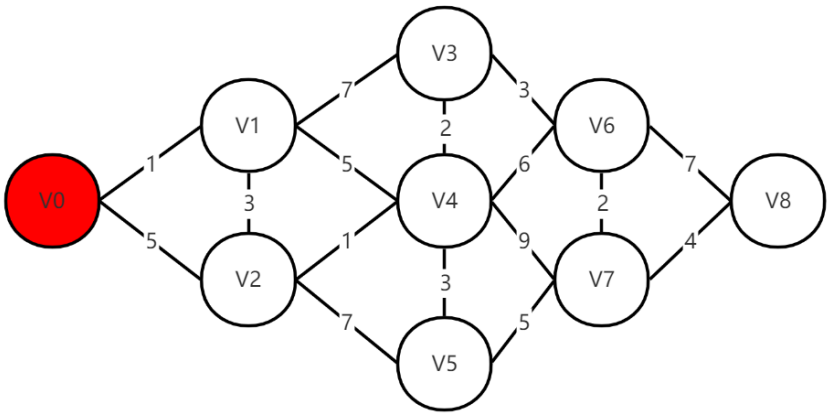
第三步：遍历顶点 V1，找到顶点V1的邻接顶点V0 、V2 、V3 、V4 （其中V0已经遍历过了，不需要考虑）。从V1到V2的距离是 3，所以V0到V2的距离是 1+3=4，小于辅助向量D中的距离 5，则更新 D[2] = 4；从V1到V3的距离是 7，所以V0到 V3 的距离是 1+7=8，小于辅助向量中的D[3]，则更新 D[3] = 8 ； 从V1到V4的距离是 5，所以V0到V4的距离是 1+5=6，小于辅助向量中的D[4] ，则更新D[4] = 6；相应的将顶点V2 、V3 、V4 的前驱结点更新为V1的下标 1。



	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	0
V3	8	1	0
V4	6	1	0
V5	∞	0	0
V6	∞	0	0
V7	∞	0	0
V8	∞	0	0

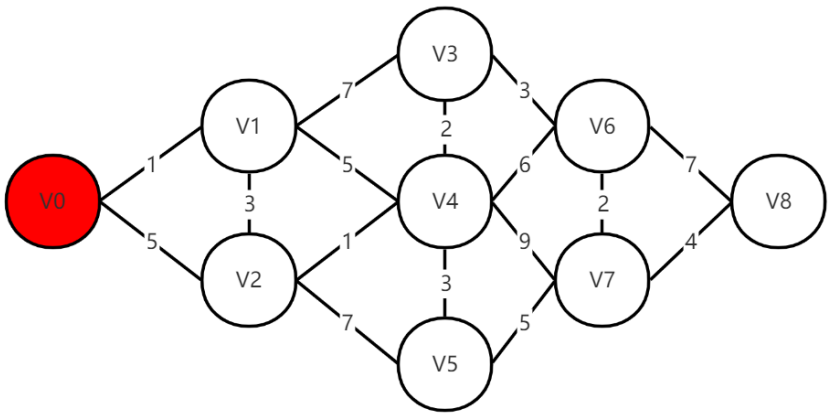
接下来就是重复第二步、第三步

第四步：遍历源点 V0, 找到从源点V0出发距离最短的且final=0的顶点，发现为 V2, 更新 Final[2] = 1



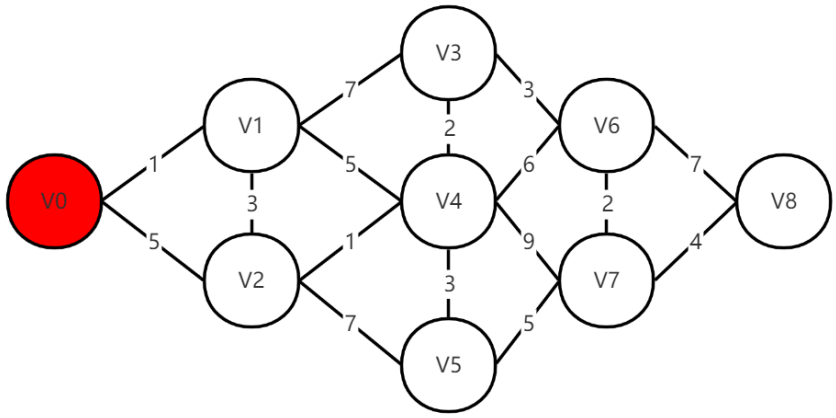
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	8	1	0
V4	6	1	0
V5	∞	0	0
V6	∞	0	0
V7	∞	0	0
V8	∞	0	0

第五步：遍历顶点V2并更新辅助向量 D 和 路径向量 P 。



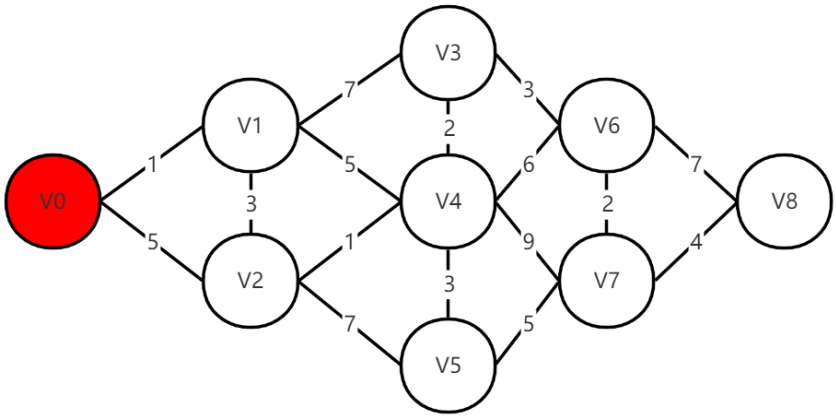
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	8	1	0
V4	5	2	0
V5	11	2	0
V6	∞	0	0
V7	∞	0	0
V8	∞	0	0

第六步：找到从源点V0出发距离最短的且final=0的顶点，发现为V4 ，更新final[4] = 1 ．



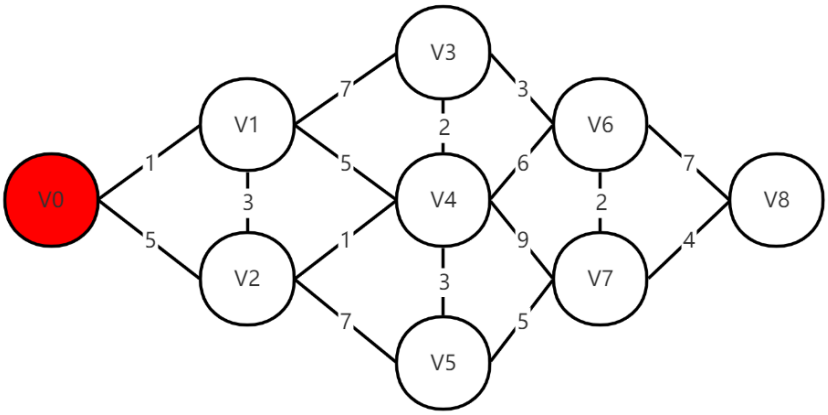
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	8	1	0
V4	5	2	1
V5	11	2	0
V6	∞	0	0
V7	∞	0	0
V8	∞	0	0

第七步：遍历顶点V4并更新辅助向量D和路径向量P



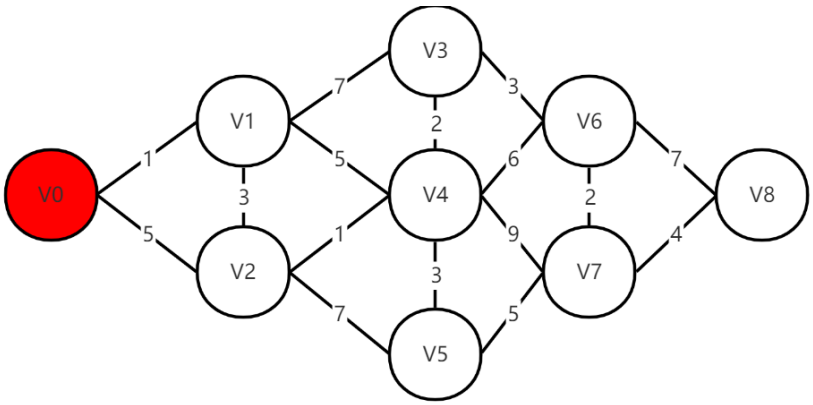
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	7	4	0
V4	5	2	1
V5	8	4	0
V6	11	4	0
V7	14	4	0
V8	∞	0	0

第八步：找到从源点V0出发距离最短的且final=0的顶点，发现为 V3 , 更新 Final[3] = 1.



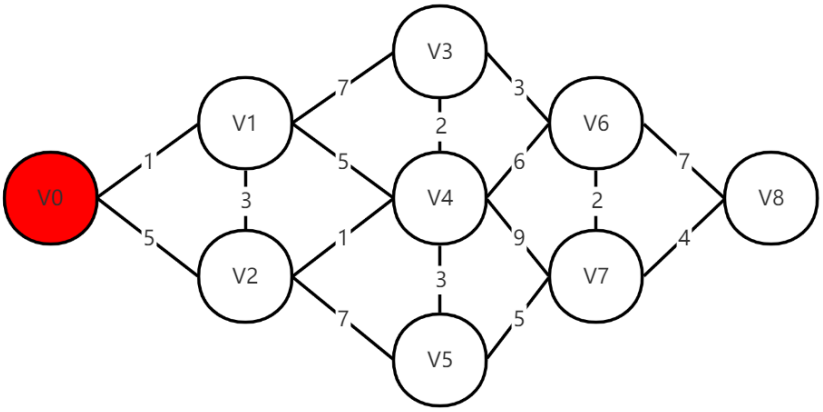
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	7	4	1
V4	5	2	1
V5	8	4	0
V6	11	4	0
V7	14	4	0
V8	∞	0	0

第九步：遍历顶点V3并更新辅助向量 D 和 路径向量 P 。



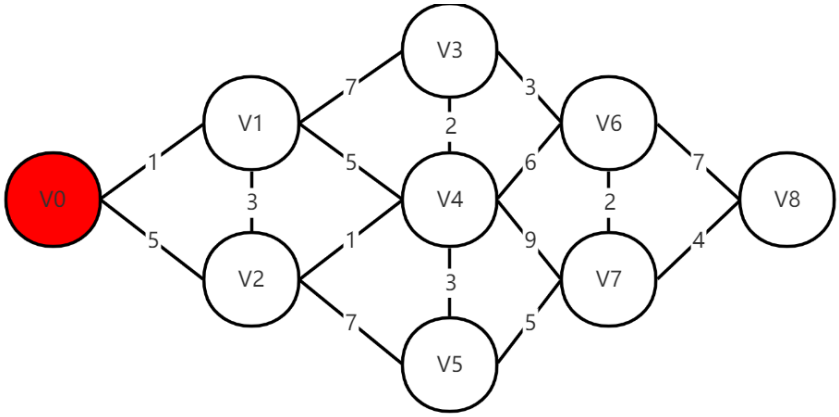
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	7	4	1
V4	5	2	1
V5	8	4	0
V6	10	3	0
V7	14	4	0
V8	∞	0	0

第十步：找到从源点V0出发距离最短的且final=0的顶点，发现为V5 ，更新final[5] = 1 ．



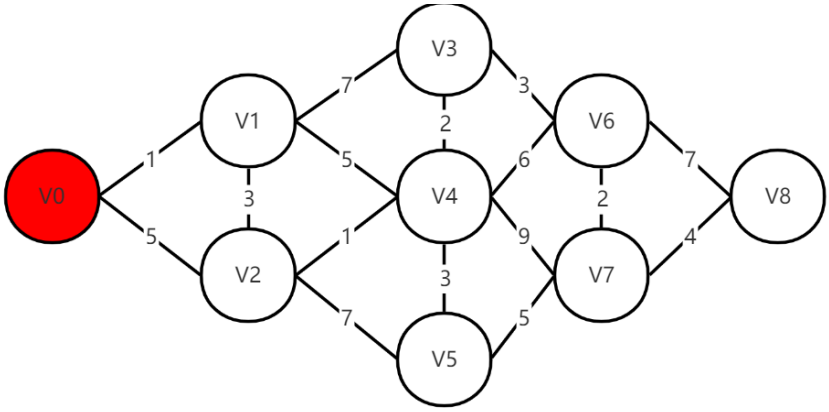
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	7	4	1
V4	5	2	1
V5	8	4	1
V6	10	3	0
V7	14	4	0
V8	∞	0	0

第十一步：遍历顶点 V5 并更新辅助向量 D 和 路径向量 P 。



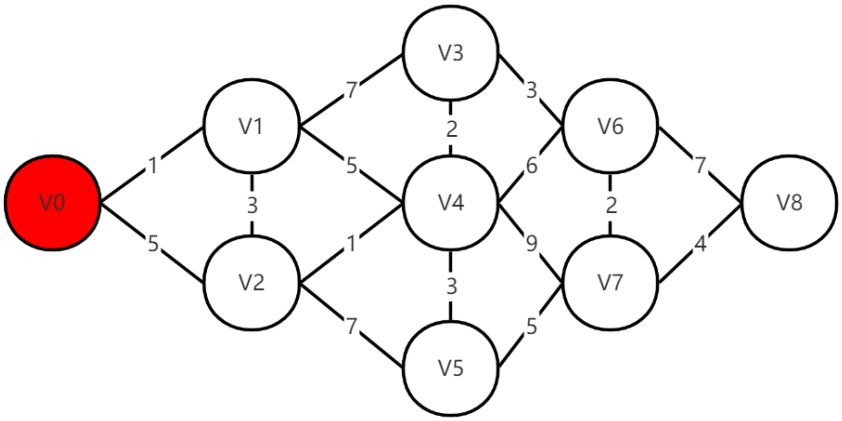
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	7	4	1
V4	5	2	1
V5	8	4	1
V6	10	3	0
V7	13	5	0
V8	∞	0	0

第十二步：找到从源点 V0出发距离最短的且final=0的顶点，发现为 V6 , 更新 final[6] = 1 .



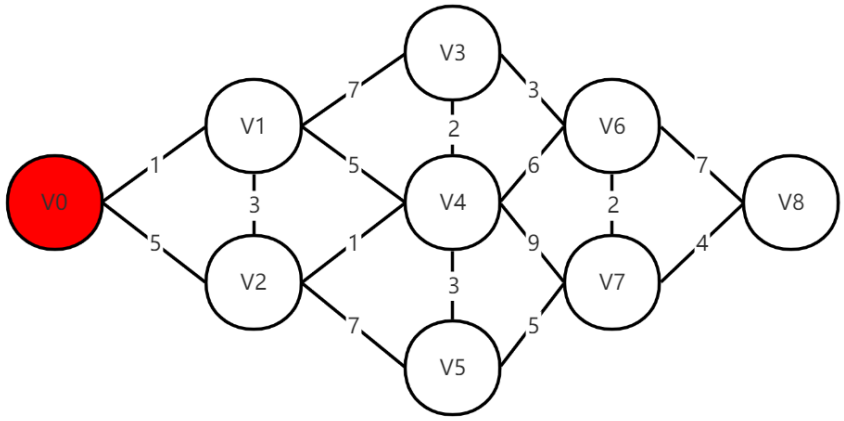
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	7	4	1
V4	5	2	1
V5	8	4	1
V6	10	3	1
V7	13	5	0
V8	∞	0	0

第十三步：遍历顶点V6并更新辅助向量 D 和 路径向量 P 。



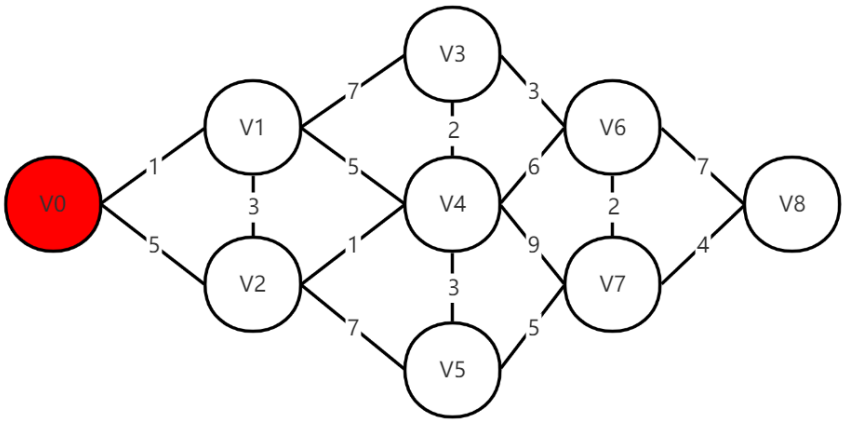
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	7	4	1
V4	5	2	1
V5	8	4	1
V6	10	3	1
V7	12	6	0
V8	17	6	0

第十四步：找到从源点 V0出发距离最短的且final=0的顶点，发现为 V7 , 更新 final[7] = 1 .



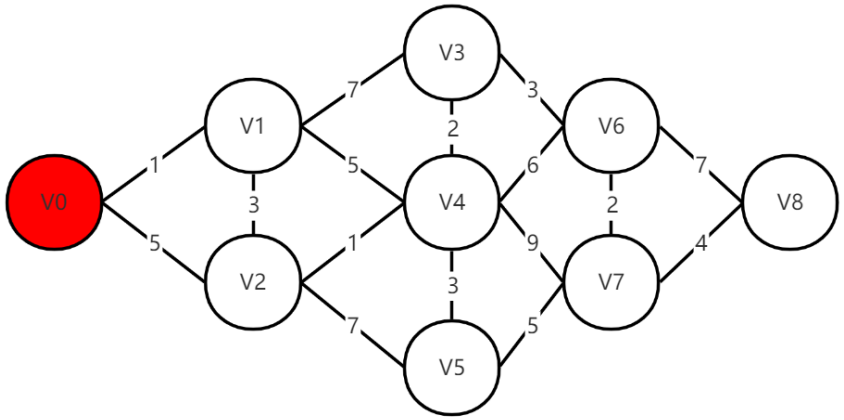
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	7	4	1
V4	5	2	1
V5	8	4	1
V6	10	3	1
V7	12	6	1
V8	17	6	0

第十五步：遍历顶点 V7 并更新辅助向量 D 和 路径向量 P 。



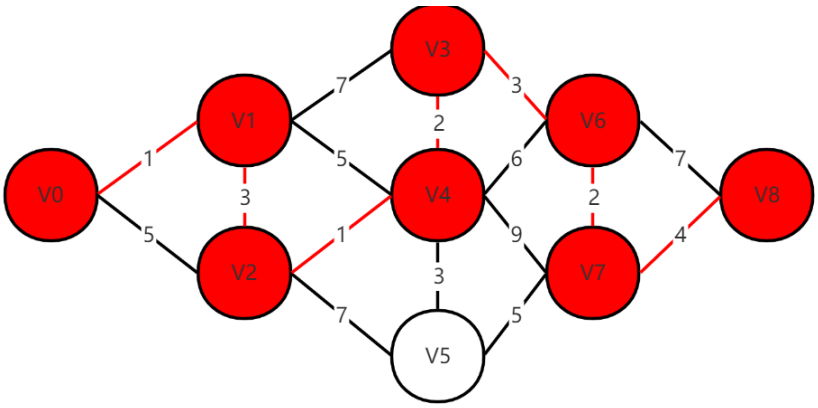
	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	7	4	1
V4	5	2	1
V5	8	4	1
V6	10	3	1
V7	12	6	1
V8	16	7	0

第十五步：找到从源点 V0出发距离最短的且final=0的顶点，发现为 V8 , 更新 final[8] = 1 .此时，全部结点完毕，算法结束。



	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	7	4	1
V4	5	2	1
V5	8	4	1
V6	10	3	1
V7	12	6	1
V8	16	7	1

根据路径向量我们则可以得到从源点V0到终点V8的最短路径。



	D	P	Final
V0	0	0	1
V1	1	0	1
V2	4	1	1
V3	7	4	1
V4	5	2	1
V5	8	4	1
V6	10	3	1
V7	12	6	1
V8	16	7	1

最短路径之弗洛伊德算法

对于迪杰斯特拉算法中谈到的是如何解决图中任意两个顶点之间最短路径的计算问题，今日就是围绕这个问题展开。

迪杰斯特拉算法可以计算指定顶点到其他顶点之间的最短路径，那么我就可以通过指定网络中的每一个顶点为源点，重复执行迪杰斯特拉算法 n 次，这样便可以得到每一对顶点之间的最短路径，显然这种方式的时间复杂度为O（n^3） 。

今日所讲的算法时间复杂度也为O（n^3） ，但是她的实现比那种方式更加简洁优雅，重要的是思想也很巧妙，她就是 **弗洛伊德（Floyd）算法** 。

迪杰斯特拉算法计算指定顶点到其他顶点之间的最短路径需要维护一个长度为 n 的辅助向量 D。而弗洛伊德算法既然可以求得每一对顶点之间的最短路径，自然要维护一个n ✖n 的二维辅助向量 D，同理也需要维护一个 n ✖n 的二维路径向量 P。接下来你所看到的就是弗洛伊德算法最核心的部分了。

现定义一个 n 阶方阵序列：

$$D^{(-1)},\ D^{(0)},\ D^{(1)},\ D^{(2)},\ \ldots\ldots,\ D^{(k)},\ \ldots\ldots,\ D^{(n-1)}$$

其中：

$$D^{(-1)}\ [i]\ [j] = G.arcs[i][j]$$

$$D^{(k)}\ [i]\ [j] = Min\left\{D^{(k-1)}\ [i]\ [j], D^{(k-1)}\ [i]\ [k] + D^{(k-1)}\ [k]\ [j]\right\},\ 0 \leqslant k \leqslant n-1$$

其中方阵

$$D^{(-1)}$$

就是我们图的邻接矩阵

$$D^{(1)}\ [i]\ [j]$$

表示从顶点Vi到顶点Vj的中间顶点的序号不大于 1 的最短路径的长度；

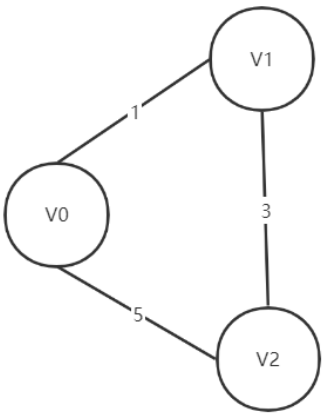
$$D^{(k)}\ [i]\ [j]$$

表示从顶点Vi到顶点Vj的中间顶点的序号不大于 k 的最短路径的长度；

$$D^{(n-1)}\ [i]\ [j]$$

表示从顶点Vi到顶点Vj的最短路径的长度；

接下来，让我们愉快的来看栗子：



D^0	V0	V1	V2
V0	0	1	5
V1	1	0	3
V2	5	3	0

我们先以上面包含三个顶点个无向图来讲解弗洛伊德算法的思想，然后再使用在将迪杰斯特拉算法时用到的图上人脑模拟一遍。

图中包含三个顶点的邻接矩阵的右上角标注的是 D^0,这是为什么呢？不是说好的 n 阶方阵初始化为图的邻接矩阵吗？ 这就得感慨数学的严谨性质了，除 n 阶方阵D^(-1)之外，其他任何一个 n 阶方阵都可以使用它的前一个状态获得,则

$$D^{(0)}\left[i\right]\left[j\right]=Min\left\{D^{(-1)}\left[i\right]\left[j\right],D^{(-1)}\left[i\right]\left[0\right]+D^{(-1)}\left[0\right]\left[j\right]\right\}=Min\left\{D^{(-1)}\left[i\right]\left[j\right],D^{(-1)}\left[i\right]\left[j\right]=D^{(-1)}\left[i\right]\left[j\right]\right\}$$

故 n 阶矩阵D^-1和D^0是同一个矩阵。那么D^1如何求得呢？当然是用她的前一个状态D^0求得了

$$D^{(1)}\left[i\right]\left[j\right]=Min\left\{D^{(0)}\left[i\right]\left[j\right],D^{(0)}\left[i\right]\left[1\right]+D^{(0)}\left[1\right]\left[j\right]\right\}$$

比如计算顶点V0到V2的最短路径：

$$D^{(1)}\left[0\right]\left[2\right]=Min\left\{D^{(0)}\left[0\right]\left[2\right],D^{(0)}\left[0\right]\left[1\right]+D^{(0)}\left[1\right]\left[2\right]\right\}$$

其中

$$D^{(0)}\left[0\right]\left[2\right]$$

的值为5，即顶点V0到顶点V2的直接距离为 5；而

$$D^{(0)}\left[0\right]\left[1\right]+D^{(0)}\left[1\right]\left[2\right]=1+3=4$$

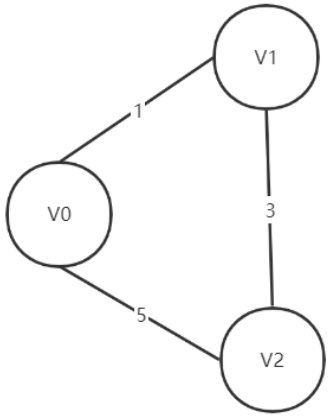
表示顶点V0到顶点V2经顶点V1中转以后的距离为4，从而将

$$D^{(1)}\left[0\right]\left[2\right]$$

更新为4，表示顶点V0到顶点V2的最短路径为4。

这就是弗洛伊德算法的精妙所在，在一个图中，要求得任意两个顶点Vi和Vj之间的最短路径，我们则可以通过两个顶点Vi和Vj之间的顶点进行中转，对于算法本身而言，则是不断得尝试所有的中转结点，从而确定两个顶点之间的最短路径。

根据公式计算完之后，获得 n 阶矩阵D^1 .

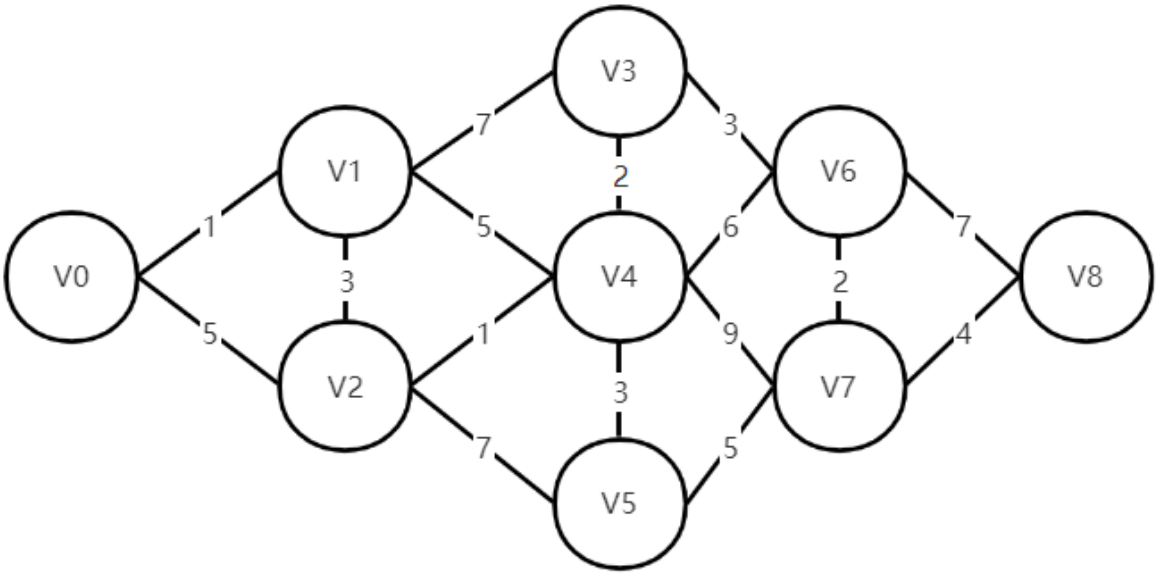


D^1	V0	V1	V2
V0	0	1	4
V1	1	0	3
V2	4	3	0

过程演示

为了更清楚弗洛伊德算法的执行过程，和弗洛伊德算法的精妙所在，我们以下图为例，一步一步得脑子过一遍。

依旧用之前的图



算法第一步，初始化 n 阶矩阵 D（辅助向量） 和n 阶矩阵P（路径向量）：

D^0	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	5	∞	∞	∞	∞	∞	∞
V1	1	0	3	7	5	∞	∞	∞	∞
V2	5	3	0	∞	1	7	∞	∞	∞
V3	∞	7	∞	0	2	∞	3	∞	∞
V4	∞	5	1	2	0	3	6	9	∞
V5	∞	∞	7	∞	3	0	∞	5	∞
V6	∞	∞	∞	3	6	∞	0	2	7
V7	∞	∞	∞	∞	9	5	2	0	4
V8	∞	∞	∞	∞	∞	∞	7	4	0

P^0	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	2	3	4	5	6	7	8
V1	0	1	2	3	4	5	6	7	8
V2	0	1	2	3	4	5	6	7	8
V3	0	1	2	3	4	5	6	7	8
V4	0	1	2	3	4	5	6	7	8
V5	0	1	2	3	4	5	6	7	8
V6	0	1	2	3	4	5	6	7	8
V7	0	1	2	3	4	5	6	7	8
V8	0	1	2	3	4	5	6	7	8

第二步：k == 1，此时

$$D^{(1)}\left[i\right]\left[j\right]=Min\left\{D^{(0)}\left[i\right]\left[j\right],D^{(0)}\left[i\right]\left[1\right]+D^{(0)}\left[1\right]\left[j\right]\right\}$$

我们主要就是计算

$$D^{(0)}\left[i\right]\left[1\right]+D^{(0)}\left[1\right]\left[j\right]$$

，其中

$$D^{(0)}\left[i\right]\left[1\right]$$

就是下方我们所标记的绿色列，而

$$D^{(0)}\left[1\right]\left[j\right]$$

则表示我们所标记的绿色行。

这样我想大家就可以进行轻松计算了，我们以取 i == 0 为例进行计算，则

$$D^{(0)}\left[0\right]\left[1\right]=1$$

分别与

$$D^{(0)}\left[1\right]\left[j\right]$$

所表示的绿色行中的每一个元素进行相加，即得到 [2,1,4,8,6,1+∞,1+∞,1+∞,1+∞]，然后与V0的行 [0,1,5,∞,∞,∞,∞,∞,∞] 进行大小比较，将更小值保留，即得到

$$D^{(1)}$$

中的V0行；此时相应的只要将

$$P^{(1)}\left[0\right]\left[j\right]$$

更新为

$$P^{(0)}\left[0\right]\left[1\right]$$

。红色表示被更新过的元素。

D^1	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	4	8	6	∞	∞	∞	∞
V1	1	0	3	7	5	∞	∞	∞	∞
V2	4	3	0	10	1	7	∞	∞	∞
V3	8	7	10	0	2	∞	3	∞	∞
V4	6	5	1	2	0	3	6	9	∞
V5	∞	∞	7	∞	3	0	∞	5	∞
V6	∞	∞	∞	3	6	∞	0	2	7
V7	∞	∞	∞	∞	9	5	2	0	4
V8	∞	∞	∞	∞	∞	∞	7	4	0

P^1	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	1	1	1	5	6	7	8
V1	0	1	2	3	4	5	6	7	8
V2	1	1	2	1	4	5	6	7	8
V3	1	1	1	3	4	5	6	7	8
V4	1	1	2	3	4	5	6	7	8
V5	0	1	2	3	4	5	6	7	8
V6	0	1	2	3	4	5	6	7	8
V7	0	1	2	3	4	5	6	7	8
V8	0	1	2	3	4	5	6	7	8

第三步：k == 2，

$$D^{(1)}\ [i]\ [2]$$

表示绿色一列

$$D^{(1)}\ [2]\ [j]$$

表示绿色的行。

0 <= i <= 8，分别与

$$D^{(1)}\ [2]\ [j]$$

表示的绿色行进行运算并比较更新，获得 D^2.

D^2	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	4	8	5	11	∞	∞	∞
V1	1	0	3	7	4	10	∞	∞	∞
V2	4	3	0	10	1	7	∞	∞	∞
V3	8	7	10	0	2	17	3	∞	∞
V4	5	4	1	2	0	3	6	9	∞
V5	11	10	7	17	3	0	∞	5	∞
V6	∞	∞	∞	3	6	∞	0	2	7
V7	∞	∞	∞	∞	9	5	2	0	4
V8	∞	∞	∞	∞	∞	∞	7	4	0

P^2	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	1	1	1	1	6	7	8
V1	0	1	2	3	2	2	6	7	8
V2	1	1	2	1	4	5	6	7	8
V3	1	1	1	3	4	1	6	7	8
V4	2	2	2	3	4	5	6	7	8
V5	2	2	2	2	4	5	6	7	8
V6	0	1	2	3	4	5	6	7	8
V7	0	1	2	3	4	5	6	7	8
V8	0	1	2	3	4	5	6	7	8

第四步：k == 3，

$$D^{(2)}\ [i]\ [3]$$

表示绿色一列，

$$D^{(2)}\ [3]\ [j]$$

表示绿色的行。0 <= i <= 8，分别与 表示的绿色行进行运算并比较更新，从而获得 D^3.

D^3	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	4	8	5	11	11	∞	∞
V1	1	0	3	7	4	10	10	∞	∞
V2	4	3	0	10	1	7	13	∞	∞
V3	8	7	10	0	2	17	3	∞	∞
V4	5	4	1	2	0	3	5	9	∞
V5	11	10	7	17	3	0	20	5	∞
V6	11	10	13	3	5	20	0	2	7
V7	∞	∞	∞	∞	9	5	2	0	4
V8	∞	∞	∞	∞	∞	∞	7	4	0

P^3	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	1	1	1	1	1	7	8
V1	0	1	2	3	2	2	3	7	8
V2	1	1	2	1	4	5	1	7	8
V3	1	1	1	3	4	1	6	7	8
V4	2	2	2	3	4	5	3	7	8
V5	2	2	2	2	4	5	2	7	8
V6	3	3	3	3	3	3	6	7	8
V7	0	1	2	3	4	5	6	7	8
V8	0	1	2	3	4	5	6	7	8

第五步：k == 4，

$$D^{(3)}\ [i]\ [4]$$

表示绿色一列，

$$D^{(3)}\ [4]\ [j]$$

表示绿色的行。

0 <= i <= 8，分别与

$$D^{(3)}\ [4]\ [j]$$

表示的绿色行进行运算并比较更新，从而获得D^4.

D^4	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	4	7	5	8	10	14	∞
V1	1	0	3	6	4	7	9	13	∞
V2	4	3	0	3	1	4	6	10	∞
V3	7	6	3	0	2	5	3	11	∞
V4	5	4	1	2	0	3	5	9	∞
V5	8	7	4	5	3	0	8	5	∞
V6	10	9	6	3	5	8	0	2	7
V7	14	13	10	11	9	5	2	0	4
V8	∞	∞	∞	∞	∞	∞	7	4	0

P^4	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	1	1	1	1	1	1	8
V1	0	1	2	2	2	2	2	2	8
V2	1	1	2	4	4	4	4	4	8
V3	4	4	4	3	4	4	6	4	8
V4	2	2	2	3	4	5	3	7	8
V5	4	4	4	4	4	5	4	7	8
V6	3	3	3	3	3	3	6	7	8
V7	4	4	4	4	4	5	6	7	8
V8	0	1	2	3	4	5	6	7	8

第六步：k == 5，

$$D^{(4)}\ [i]\ [5]$$

表示绿色一列，

$$D^{(4)}\ [5]\ [j]$$

表示绿色的行。

0 <= i <= 8，分别与

$$D^{(4)}\ [5]\ [j]$$

表示的绿色行进行运算并比较更新，从而获得D^5.

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

P^5	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	1	1	1	1	1	1	8
V1	0	1	2	2	2	2	2	2	8
V2	1	1	2	4	4	4	4	4	8
V3	4	4	4	3	4	4	6	4	8
V4	2	2	2	3	4	5	3	5	8
V5	4	4	4	4	4	5	4	7	8
V6	3	3	3	3	3	3	6	7	8
V7	4	5	5	5	5	5	6	7	8
V8	0	1	2	3	4	5	6	7	8

第七步：k == 6，

$$D^{(5)}\ [i]\ [6]$$

表示绿色一列，

$$D^{(5)}\ [6]\ [j]$$

表示绿色的行。

0 <= i <= 8，分别与

$$D^{(5)}\ [6]\ [j]$$

表示的绿色行进行运算并比较更新，从而获得D^6.

D^6	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	4	7	5	8	10	12	17
V1	1	0	3	6	4	7	9	11	16
V2	4	3	0	3	1	4	6	8	13
V3	7	6	3	0	2	5	3	5	10
V4	5	4	1	2	0	3	5	7	12
V5	8	7	4	5	3	0	8	5	15
V6	10	9	6	3	5	8	0	2	7
V7	12	11	8	5	7	5	2	0	4
V8	17	16	13	10	12	15	7	4	0

P^6	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	1	1	1	1	1	1	1
V1	0	1	2	2	2	2	2	2	2
V2	1	1	2	4	4	4	4	4	4
V3	4	4	4	3	4	4	6	6	6
V4	2	2	2	3	4	5	3	3	3
V5	4	4	4	4	4	5	4	7	4
V6	3	3	3	3	3	3	6	7	8
V7	6	6	6	6	6	5	6	7	8
V8	6	6	6	6	6	6	6	7	8

第八步：k == 7，

$$D^{(6)}\ [i]\ [7]$$

表示绿色一列，

$$D^{(6)}\ [7]\ [j]$$

表示绿色的行。

0 <= i <= 8，分别与

$$D^{(6)}\ [7]\ [j]$$

表示的绿色行进行运算并比较更新，从而获得D^7.

D^6	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	4	7	5	8	10	12	16
V1	1	0	3	6	4	7	9	11	15
V2	4	3	0	3	1	4	6	8	12
V3	7	6	3	0	2	5	3	5	9
V4	5	4	1	2	0	3	5	7	11
V5	8	7	4	5	3	0	8	5	9
V6	10	9	6	3	5	8	0	2	6
V7	12	11	8	5	7	5	2	0	4
V8	16	15	12	9	11	9	6	4	0

P^6	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	1	1	1	1	1	1	1
V1	0	1	2	2	2	2	2	2	2
V2	1	1	2	4	4	4	4	4	4
V3	4	4	4	3	4	4	6	6	6
V4	2	2	2	3	4	5	3	3	3
V5	4	4	4	4	4	5	4	7	7
V6	3	3	3	3	3	3	6	7	7
V7	6	6	6	6	6	5	6	7	7
V8	7	7	7	7	7	7	7	7	8

第九步：k == 8，同样的道理进行计算，我们注意到绿色区域划分出的左上角的区域都是不需要被更新的，绿色区域本身也是不更新的，所以 D^7与D^8 是一样的了。

D^6	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	4	7	5	8	10	12	16
V1	1	0	3	6	4	7	9	11	15
V2	4	3	0	3	1	4	6	8	12
V3	7	6	3	0	2	5	3	5	9
V4	5	4	1	2	0	3	5	7	11
V5	8	7	4	5	3	0	8	5	9
V6	10	9	6	3	5	8	0	2	6
V7	12	11	8	5	7	5	2	0	4
V8	16	15	12	9	11	9	6	4	0

P^6	V0	V1	V2	V3	V4	V5	V6	V7	V8
V0	0	1	1	1	1	1	1	1	1
V1	0	1	2	2	2	2	2	2	2
V2	1	1	2	4	4	4	4	4	4
V3	4	4	4	3	4	4	6	6	6
V4	2	2	2	3	4	5	3	3	3
V5	4	4	4	4	4	5	4	7	7
V6	3	3	3	3	3	3	6	7	7
V7	6	6	6	6	6	5	6	7	7
V8	7	7	7	7	7	7	7	7	8

这样，整个弗洛伊德算法的执行过程就结束了，我知道看完，可能还是有一点儿模糊，不过我希望你能多看几遍这个例子，最好是自己也和给你们绘制的图一样，自己手工地人脑模拟一遍，只有这样，你才会真正理解算法的精妙所在。

总结

最短路径算法最经典的是迪杰斯特拉算法与弗洛伊德算法，这两个算法也是严蔚敏老师书中所提及的两个算法，当然也有一些其他算法，Bellman-Ford 与 SPFA（Bellman-Ford算法的队列优化）；只要大家把迪杰斯特拉算法与弗洛伊德算法掌握了，不论面试考试绝对不在话下。

迪杰斯特拉算法用于计算一个顶点到其他顶点的最短路径的 “单源点” 最短路径算法，其实我们可以采用小顶堆进行优化。弗洛伊德算法用于计算图中任意顶点之间的最短路径的 “ 多源点” 最短路径算法。两者都存在无法处理带负权值图的最短路径算法。但现实情况中也很少存在负权值的情况了，所以还是学好这个算法，哈哈，加油奥！