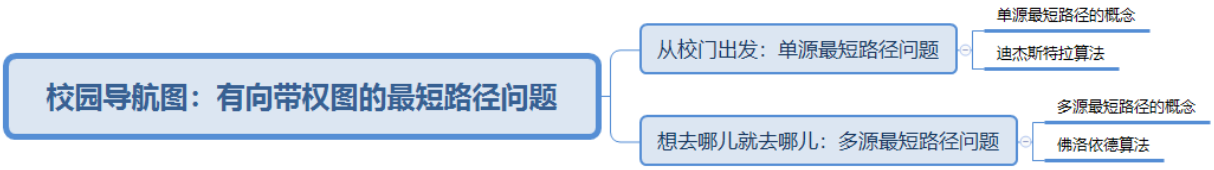


1.校园导航图：有向带权图的最算路径问题

在之前的学习当中，我们已经知道了什么是无向图的最小生成树问题。

无向图的最小生成树结构，适用于同一条边连接的两个节点之间，能够双向往来的情况。

在这种情况下，我们通过普里姆算法和克鲁斯卡尔算法计算得到的最小生成树结构，具有边权总和最小的特点

这也使得在一些特定场合下，例如双向通信的交换机集群布线的场景，使用最小生成树结构能够将布线成本降至最低

但是这两种算法，充其量只能够应付无向图，如果将无向图结构升级成为有向图结构，那么就会遇见新的问题，并且需要使用新的算法进行解决

下面我们想想一个场景：

你站在学校的大门口，心里想着：今天我得去主楼上课，然后还想去食堂吃饭，或者……回寝室睡一会儿也行，对了！还得去图书馆还借的书……

这个时候你犯难了……因为上面提到的几处场景，互相之间都有点儿距离，并且有些道路，还都是单行道……（为啥学校里头会有单行道……）

你站在学校大门口有点儿无所适从，不知道怎么走才最近……

上面你遇到的问题，实际上就是有向图中的最短路径问题，而根据最短路径的起点是否固定，我们又能够将之分别描述为：

有向带权图的单源最短路径问题：即在图中去往其他节点的起点是固定的，都是从同一个节点出发，然后计算从这个点出发，如何才能够通过最短的路程，到达其他的点

有向带权图的多源最短路径问题：即需要计算，从任何一个节点出发，到达其他节点的最短路径是什么，距离为多少

上面的这两个问题，适用于有向带权图的环境，而这两个问题的解决算法分别是：

有向带权图的单源最短路径问题：迪杰斯特拉算法

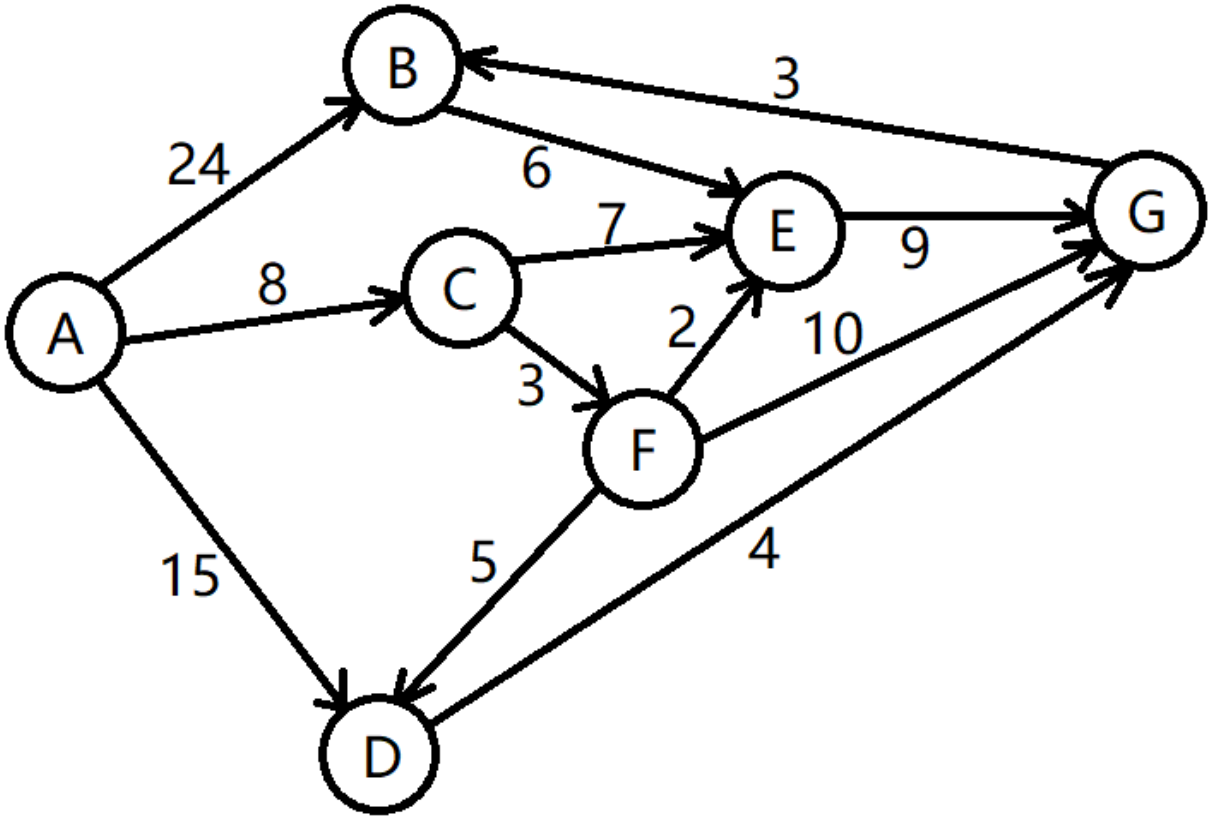
有向带权图的多源最短路径问题：佛洛依德算法（不是玩儿心理学那哥们儿……）

那么接下来，就让我们一起研究下，这两个算法解决的问题，以及执行的流程

①从校门出发：单源最短路径问题

1.单源最短路径的概念

在解释单源最短路径的概念之前，我们先来给出一张有向带权图，方便我们后序的讲解：



在这张图中我们不难看出，点与点之间的边，不仅带有边权，而且还具有指向，有的节点之间能够往来，有的节点之间只能够单向通行

但是从这张图中我们也能够大概看出来：如果从A节点出发的话，基本上能够走遍图中的所有节点

那么就让我们如上面所说，以A节点为起点，计算出从A节点到达其他节点的最短路径，这就是单源最短路径问题

2.迪杰斯特拉算法

在迪杰斯特拉算法开始之前，我们先给出这样一个矩阵，并通过这个矩阵表示各个节点之间是否具有通路、如果具有通路的话，边权是多少：

	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞

D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

上面的这张表，我们称之为图的邻接矩阵表，下面我们对图的邻接矩阵表进行说明：

1.表中的m行n列表示从节点m到节点n之间的距离，也就是连接m节点和n节点之间边的边权

2.表中取值为∞的部分，表示在原图中两个节点之间没有直接相连的边，可以理解为两点之间的距离无穷大

3.表中取值为正整数的部分，表示在原图中两点之间具有直接相连的边，并且这个值就是边的边权

4.表中取值为0的部分，都是m行m列的部分，也就是表示一个点从自己出发，到达自己的边长，所以取值为0

5.因为在有向图中，从节点m到节点n之间具有边相连，但是并不代表从节点n出发返回节点m同样具有相连的边，既是具有相连的边，边的权值也有可能不同

所以，有向图的邻接矩阵表是一个不对称的矩阵

6.注意：邻接矩阵表同样可以用来表示无向带权图的节点连接情况（只不过是之前的普里姆算法和克鲁斯卡尔算法中并没有用上而已）

但是在无向带权图中，因为两个节点之间的边是可以往返的，所以只要从节点m到节点n之间具有相连的边，那么就表示从节点n出发，同样能够通过这条边回到节点m

进一步说也就是，在无向带权图的邻接矩阵表中，m行n列的取值和n行m列的取值一定是相同的，这也就表明，无向带权图的邻接矩阵表是一个以对角线对称的矩阵

好的，在了解完有向带权图的邻接矩阵表的概念之后，下面让我们列出这样一张表，用来辅助计算迪杰斯特拉算法：

节点取值	A	B	C	D	E	F	G	从A节点能抵达的

								点的集合（S集合）
最短距离								
最短路径								

下面我们来解释一下这张表的含义：

节点取值：表示从A点出发，到达对应点的各种信息，如：B列用来记录从A点出发，到达B点的最近距离和最短路径

其中A列是不记录信息的，因为A列表示从节点A出发，自己到达自己的最短距离和最短路径，没有任何意义

最短距离：即从A点出发，直接连通或者经过其他节点到达这一节点的边权之和

最短路径：记录一个字符串，用来表示从A点出发，直接连通或者经过其他节点到达这一节点的路径，字符串结尾一定是这个节点的取值

例如：B列最终的最短路径是ADGB，表示从A节点出发，途经D节点和G节点最终达到B节点，就是从A节点到B节点的最短路径

从A节点能抵达的点的集合（S集合）：相当于最小生成树中普里姆算法下的“用过的点的集合”，表示当前已经计算出来最短路径的节点

这一列中的点的最短路径是已经计算完毕的，不会再去进行更改

好的，上面表格的含义已经解释完成了，下面让我们一起说明一下迪杰斯特拉算法的具体步骤：

步骤1：列出从起点出发，能够直接连通的所有其他节点

步骤2：找出这些节点中，距离A节点最近的节点，加入“从A节点能抵达的点的集合”中，即加入S集合

步骤3：从这一新加入S集合的点出发，通过邻接矩阵表，扫描对应行，列出所有从这个点出发，能够抵达的其他节点的边权和最短路径

步骤4：通过扫描这一点在邻接矩阵表中，与其他节点相连的边权取值，更新表中从A点出发抵达其他节点的最短距离和最短路径

例如：原先A节点与F节点相连，最短距离为25，最短路径为AF；在C节点加入之后，A节点与C节点相连，最短距离为11，最短路径为AC

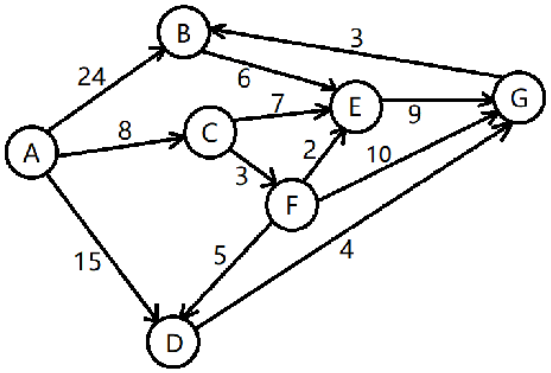
C节点同样与F节点相连，最短距离为10，最短路径为CF，此时：AC + CF < AF，则更新从A节点到F节点的最短距离为AC + CF = 21

最短路径从AF更新为ACF

步骤5：重复上述步骤2-4，直到所有的节点都加入S集合中为止，从A节点出发抵达其他节点的最短路径查找完成

好的各位蒙圈的同学们，下面就让我们根据上面给出的有向带权图以及其邻接矩阵表，来演算一遍找到上述图中从A节点出发，到达其他节点的最短路径的过程

第1步：因为是从A节点出发，所以在表中列出所有从A节点出发能够直接连通的节点的最短距离和最短路径，并将A节点纳入S集合

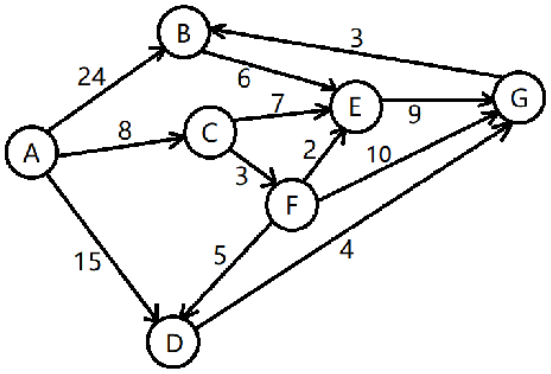


	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集

								合）
最短距离		24	8	15				A
最短路径		AB	AC	AD				

第2步：找到目前从A节点出发，能够抵达的3个节点BCD中，距离A最近的节点，将其纳入S集合，确定到达这个点的最短路径

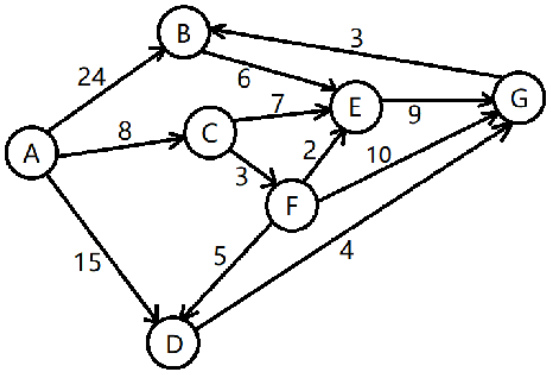


	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集合）
最短距离		24	8	15				A C

最短路径		AB	AC	AD				

第3步：从C节点出发，列出从C节点出发，能够抵达的其他新节点的距离和路径，加在AC这条路径的基础上



	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

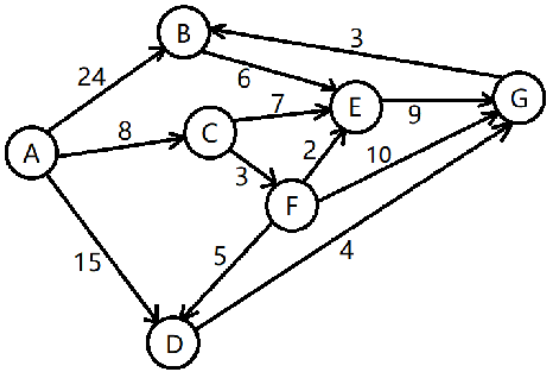
节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集合）
最短距离		24	8	15	15	11		A C
最短路径		AB	AC	AD	ACE	ACF		

第4步：考虑从C节点出发，能否到达B节点或者D节点，如果能够到达，是否存在AC + CB < AB或者AC + CD < AD的情况

如果存在，则更新最短距离和最短路径。通过观察发现，从C节点出发，并不能够到达B节点或者D节点，则这两个节点的最短路径不变

第5步：从上表中选出最短距离最小的一项（用过的不再考虑），将终点纳入S集合

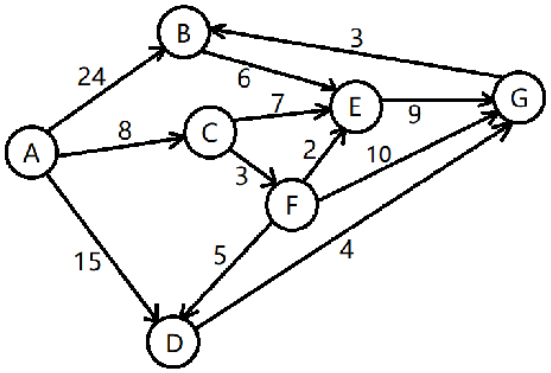
这一步同样可以解释为：找出从A节点出发，途径C节点，能够到达最近的节点



	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集合）
最短距离		24	8	15	15	11		A C F
最短路径		AB	AC	AD	ACE	ACF		

第6步：考虑从F节点出发，列出能够抵达的其他节点的距离



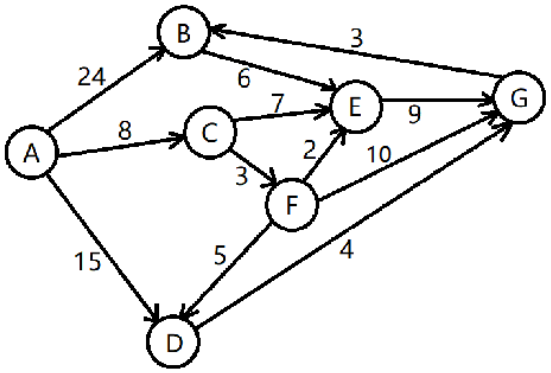
	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集合）
最短距离		24	8	15	15	11	21	A C F
最短路径		AB	AC	AD	ACE	ACF	ACFG	

第7步：考虑从F节点出发，能够缩短从A节点到达其他节点的路径

因为从F节点出发，能够抵达D节点和E节点，也就是说现在要考虑是否存在ACF + FE < ACE或者ACF + FD < AD的情况

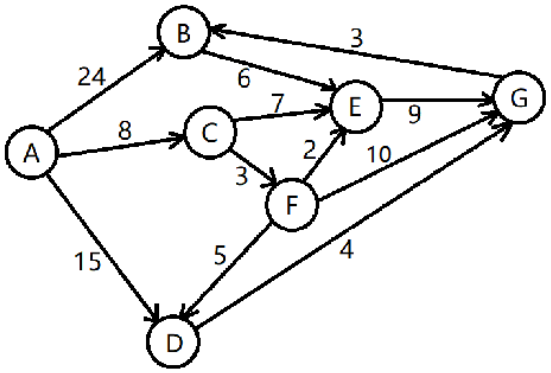
根据上述情况描述，存在ACF + FE = 13 < ACE = 15的情况，所以对上表中由A节点出发，抵达E节点的情况进行更新



	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集合）
最短距离		24	8	15	13	11	21	A C F
最短路径		AB	AC	AD	ACFE	ACF	ACFG	

第8步：继续在上表中找出最短距离最小值，即从A节点到达E节点的情况，将E节点纳入S集合



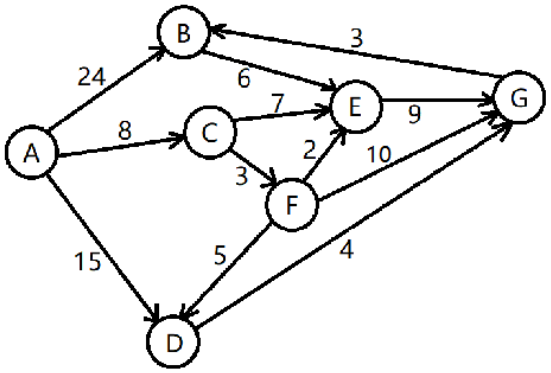
	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集合）
最短距离		24	8	15	13	11	21	A C F E
最短路径		AB	AC	AD	ACFE	ACF	ACFG	

第9步：考虑从E节点出发，抵达其他节点的情况，因为从E节点出发，不会抵达任何新节点（即本表中出A列外，其他列均已填满）

所以直接考虑从E节点出发，会不会缩短从A节点抵达B、D、G节点的路径的情况，发现这种情况不存在，所以直接略过

第10步：继续在上表中找出最短距离最小的情况，即从A节点直接到达D节点的情况，将D节点纳入S集合

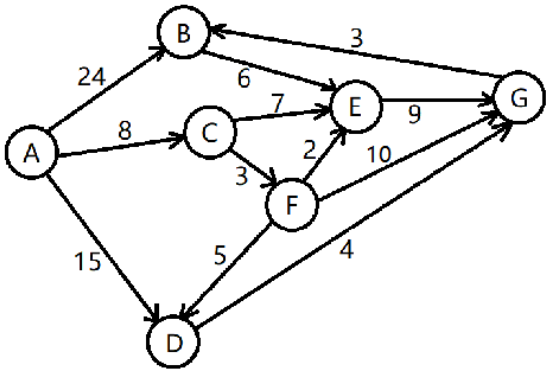


	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集合）
最短距离		24	8	15	13	11	21	A C F E D
最短路径		AB	AC	AD	ACFE	ACF	ACFG	

第11步：考虑从D节点出发，能够抵达的其他节点的路径长度，因为从D节点出发不会抵达其他新节点，所以直接考虑从D节点出发

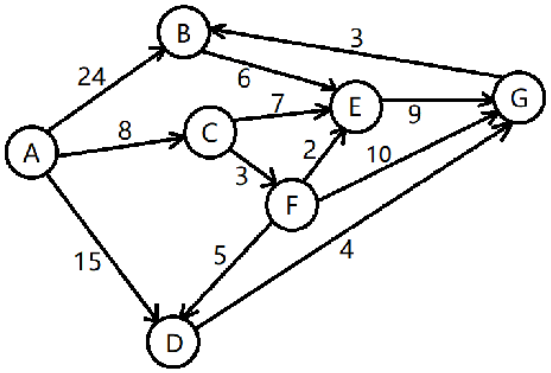
能否缩短从A节点出发，抵达B、D、G节点的距离，发现：AD + DG = 19 < ACFG = 21，所以更新从A节点抵达G节点的最短路径



	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集合）
最短距离		24	8	15	13	11	19	A C F E D
最短路径		AB	AC	AD	ACFE	ACF	ADG	

第12步：继续从上表中寻找最短距离最小的列，将终点纳入S集合中

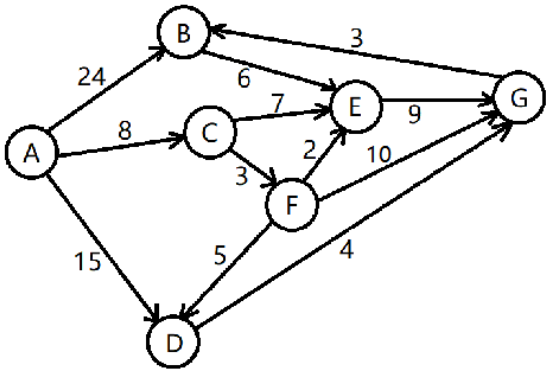


	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集合）
最短距离		24	8	15	13	11	19	A C F E D G
最短路径		AB	AC	AD	ACFE	ACF	ADG	

第13步：考虑从G节点出发，能否缩短从A节点抵达B节点的路径

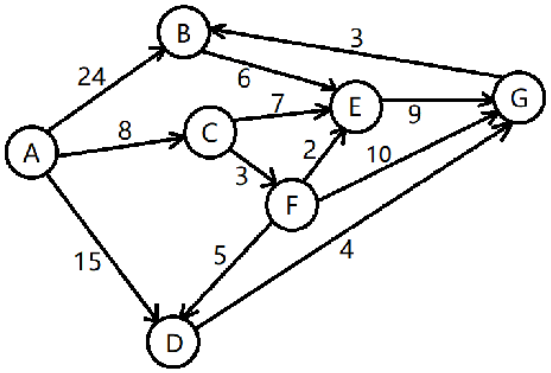
经过计算得到：ADG + GB = 22 < AB = 24，所以更新上表为：



	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集合）
最短距离		22	8	15	13	11	19	A C F E D G
最短路径		ADGB	AC	AD	ACFE	ACF	ADG	

第14步：将最后剩余的B节点纳入S集合中，表示从A节点抵达其余所有节点的最短路径均已计算完毕，迪杰斯特拉算法运行结束



	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

节点取值	A	B	C	D	E	F	G	从A节点能抵达的点的集合（S集合）
最短距离		22	8	15	13	11	19	A C F E D G
最短路径		ADGB	AC	AD	ACFE	ACF	ADG	

总结一下上面步骤体现出来迪杰斯特拉算法的核心思想：迪杰斯特拉算法实际上就是在不断的将新的节点纳入S集合的过程中

考虑经过这个新节点，能不能进一步缩短从起点抵达这个节点的距离，如果能，那么我们必须更新从起点到达这个节点的路径和距离

实际上这种情况，玩过大富翁的同学应该比较清楚：到达下一个点的路径长度，等于你之前走过的路径 长度加上摇骰子的点数

实际上在迪杰斯特拉算法中也是一样的，假设有ABCD4个节点，你已经计算出到达B节点的最短路径是ACB，长度为10，目前从A节点到D节点的路径AD长度是20

而且恰好从B节点到D节点有一条长度为5的路径，并且你正好站在B节点上，那么相比起直接从A走到D的距离为20，不如多经过几个节点，走ACBD，距离为15

注意：有的同学可能发现，迪杰斯特拉算法和普里姆算法步骤上和思想上具有很大的相似性：都是查找一个距离最近的点，所以有些同学对这两个算法总是傻傻分不清楚

实际上，这两个算法之间，一般来讲，存在着两个比较明显的差异：

差异1：普里姆算法用于解决无向图的最小生成树问题，主要针对无向图使用；迪杰斯特拉算法用于解决有向图的单源最短路径问题，主要针对有向图使用

差异2：普里姆算法得到的最小生成树结构，是一种“全局最优”的结构，也就是说最小生成树的边权加和相对于其他生成树结构来说是最小的，

但是在最小生成树中，我们可以发现：从某一个固定起点出发，抵达其他点的距离，不一定是最短的，可能存在其他更短路径，所以普里姆算法属于一种“放眼全局，舍小博大”的算法；

而迪杰斯特拉算法与普里姆算法正好相反，只求当前距离起点最近的点，并不关心整体路径边权之和的大小，这就有可能引起在单源最短路径中，边权总和比其他生成树结构更大

所以，我们也可以将迪杰斯特拉算法得到的单源最短路径集合，看做是一种特定树根的生成树结构，而其思想正好与普里姆算法相反，是一种“贪图眼前，舍大换小”的算法思想

②想去哪儿就去哪儿：多源最短路径问题

1.多源最短路径的概念

在上面，我们已经讨论了单源最短路径问题的迪杰斯特拉算法。

单源最短路径指的是从一个固定的起点出发，计算到达其他节点路径最短的算法。那么，有没有一种可能，我们能够计算出从任何一个节点

抵达其他节点，距离都最近的情况呢？答案是肯定的，那就是佛洛依德算法（注意：此佛洛依德非彼佛洛依德，不是玩儿心理学解梦那哥们儿……），

这个算法根据其计算方式，也被称之为“插点法”（注意是插点不是插眼……）

综上所述，多源最短路径问题，指的就是在有向带权图中，计算从任何一个点出发，怎么走才能够以最短的路径抵达其他节点，也可以理解成为升级版的单源最短路径问题

虽然在问题形式上有所升级，但是佛洛依德算法不论在原理层面上还是在代码实现层面上，都比迪杰斯特拉算法来的简单

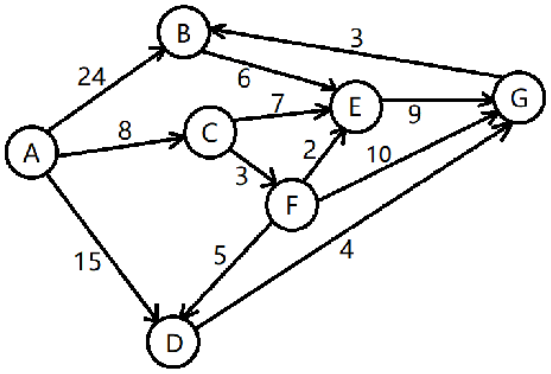
甚至在不记录最短路径经历节点，仅计算最短路径距离的情况下，使用5行代码就能够实现，所以我们也戏称佛洛依德算法是一种仅有5句代码的算法

（实际上如果记录最短路径经历节点的话，也只要6句代码就能够实现……）

下面，让我们一起来学习一下，这个仅有5条代码的算法，是如何实现的

2.弗洛伊德算法

老规矩，在学习佛洛依德算法之前，我们还是使用上面的那张有向图，并使用相同的邻接矩阵：



	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞
B	∞	0	∞	∞	6	∞	∞
C	∞	∞	0	∞	7	3	∞
D	∞	∞	∞	0	∞	∞	4
E	∞	∞	∞	∞	0	∞	9
F	∞	∞	∞	5	2	0	10
G	∞	3	∞	∞	∞	∞	0

不同的是，我们除了邻接矩阵之外，还要额外提供一个中转点矩阵，矩阵结构如下所示：

	A	B	C	D	E	F	G
A	A	B	C	D	E	F	G
B	A	B	C	D	E	F	G
C	A	B	C	D	E	F	G
D	A	B	C	D	E	F	G
E	A	B	C	D	E	F	G
F	A	B	C	D	E	F	G
G	A	B	C	D	E	F	G

实际上上面这个矩阵的结构只是一种初始化结构，至于为什么这么填充这个矩阵：规定！

这个中转点矩阵表示的含义就是：m行n列表示的值就是从m节点去往n节点的中转节点

例如图中A行E列的取值是E，那么就表示从A点到达E点的中转点是E

但是有同学会说：不对啊：从A节点出发，并没有直接通往E节点的路径啊！不存在中转点这一概念！

没错，所以这个中转点矩阵还要和邻接矩阵一起看才能看懂：

在邻接矩阵中，A行E列的取值是∞，那么也就意味着，目前还不能够从A直接走到E

这个矩阵会在后序过程中和邻接矩阵一起进行调整，直到最后，能够正确的表示每两个节点之间路径的中转点为止

下面我们来说一下佛洛依德算法的执行过程：

步骤1：按照有A-G的顺序，选中图中的节点，每一次向图中插入一个节点，插入的节点称之为“中转点”

步骤2：在插入中转点后，判断每一个起点经过此中转点到达其他终点的路径边权会不会减小（对角线除外）

假设向图中插入的是第k个节点，那么判断条件就是：if(邻[i][j] > 邻[i][k] + 邻[k][j])

如果边权没有减小，则保持邻接矩阵中的这个位置取值不变，中转点矩阵保持不变（邻[i][j] <= 邻[i][k] + 邻[k][j]的情况）

如果边权有所减小，则将邻接矩阵下这个位置上的边权进行更新，同时更新中转点矩阵（邻[i][j] > 邻[i][k] + 邻[k][j]的情况）

（更新中转点矩阵的代码：转[i][j] = 转[i][k]，实际上在初始中转矩阵中的第k列上的取值是一样的，为了取值方便，我们使用转[i][k]表示）

实际上这个步骤就是插点的步骤，也就是试探在这个点加入图中之后，会不会影响其他路径的权值取值

步骤3：重复上述步骤1-2，直到所有的节点都被插入到图中为止

嗯嗯……佛洛依德算法的步骤确实简单……但是貌似还是有些不好理解……那么废话少说，上案例！

还是以上面的图和临界矩阵为例，我们来研究一下佛洛依德算法的步骤：

第1步：初始化邻接矩阵和中转矩阵是这个样子的：

为了方便起见，我将两个矩阵画在了一起，邻表示邻接矩阵的部分；转表示中转点矩阵的部分

邻	A	B	C	D	E	F	G	转	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞	A	A	B	C	D	E	F	G
B	∞	0	∞	∞	6	∞	∞	B	A	B	C	D	E	F	G
C	∞	∞	0	∞	7	3	∞	C	A	B	C	D	E	F	G

D	∞	∞	∞	0	∞	∞	4	D	A	B	C	D	E	F	G
E	∞	∞	∞	∞	0	∞	9	E	A	B	C	D	E	F	G
F	∞	∞	∞	5	2	0	10	F	A	B	C	D	E	F	G
G	∞	3	∞	∞	∞	∞	0	G	A	B	C	D	E	F	G

第2步：插入A节点到图中，因为此时图中相当于仅仅具有一个节点，所以也不会对上述两个矩阵产生任何影响

所以上面的两个矩阵保持不变，直接进入下一步

邻	A	B	C	D	E	F	G	转	A	B	C	D	E	F	G
A	0	24	8	15	∞	∞	∞	A	A	B	C	D	E	F	G
B	∞	0	∞	∞	6	∞	∞	B	A	B	C	D	E	F	G
C	∞	∞	0	∞	7	3	∞	C	A	B	C	D	E	F	G
D	∞	∞	∞	0	∞	∞	4	D	A	B	C	D	E	F	G
E	∞	∞	∞	∞	0	∞	9	E	A	B	C	D	E	F	G
F	∞	∞	∞	5	2	0	10	F	A	B	C	D	E	F	G
G	∞	3	∞	∞	∞	∞	0	G	A	B	C	D	E	F	G

第3步：插入B节点到图中，开始判断其余节点经过B节点，到达另一节点的路径权值有没有变小

当节点B加入之后，我们发现从A节点到E节点之间有了通路，取值为30（邻接矩阵A行E列改为30）

从G节点到E节点之间也有了通路，取值为9（邻接矩阵G行E列改为9）

此时，我们还要更新中转点矩阵，让矩阵中的A行E列和G行E列的取值为B

邻接矩阵和中转点矩阵之间变换完毕后，得到：

邻	A	B	C	D	E	F	G	转	A	B	C	D	E	F	G
A	0	24	8	15	30	∞	∞	A	A	B	C	D	B	F	G
B	∞	0	∞	∞	6	∞	∞	B	A	B	C	D	E	F	G
C	∞	∞	0	∞	7	3	∞	C	A	B	C	D	E	F	G
D	∞	∞	∞	0	∞	∞	4	D	A	B	C	D	E	F	G

E	∞	∞	∞	∞	0	∞	9	E	A	B	C	D	E	F	G
F	∞	∞	∞	5	2	0	10	F	A	B	C	D	E	F	G
G	∞	3	∞	∞	9	∞	0	G	A	B	C	D	B	F	G

第4步：插入节点C到图中，开始判断其余节点经过C节点，到达另一节点的路径权值有没有变小

当插入C节点之后，我们发现A节点经过C节点到达E节点的距离15比经过B节点到达E节点的距离30要小

且在A节点与F节点之间建立了通路，距离为11

所以：更改邻接矩阵A行E列改为15，A行F列改为11

中转点矩阵的A行E列修改为C，A行F列改为C

邻接矩阵和中转点矩阵之间变换完毕后，得到：

邻	A	B	C	D	E	F	G	转	A	B	C	D	E	F	G
A	0	24	8	15	15	11	∞	A	A	B	C	D	C	C	G
B	∞	0	∞	∞	6	∞	∞	B	A	B	C	D	E	F	G
C	∞	∞	0	∞	7	3	∞	C	A	B	C	D	E	F	G
D	∞	∞	∞	0	∞	∞	4	D	A	B	C	D	E	F	G
E	∞	∞	∞	∞	0	∞	9	E	A	B	C	D	E	F	G
F	∞	∞	∞	5	2	0	10	F	A	B	C	D	E	F	G
G	∞	3	∞	∞	9	∞	0	G	A	B	C	D	B	F	G

第5步：插入节点D到图中，开始判断其余节点经过D节点，到达另一节点的路径权值有没有变小

当插入节点D之后，A节点和G节点之间建立了通路，长度是19

从F节点经过D节点到达G节点的距离9小于直接从F节点出发到达G节点的距离10

所以，我们将邻接矩阵的A行G列改为19，F行G列改为9

将中转点矩阵的A行G列改为D，F行G列改为D

邻	A	B	C	D	E	F	G	转	A	B	C	D	E	F	G
A	0	24	8	15	15	11	19	A	A	B	C	D	C	C	D

B	∞	0	∞	∞	6	∞	∞	B	A	B	C	D	E	F	G
C	∞	∞	0	∞	7	3	∞	C	A	B	C	D	E	F	G
D	∞	∞	∞	0	∞	∞	4	D	A	B	C	D	E	F	G
E	∞	∞	∞	∞	0	∞	9	E	A	B	C	D	E	F	G
F	∞	∞	∞	5	2	0	9	F	A	B	C	D	E	F	D
G	∞	3	∞	∞	9	∞	0	G	A	B	C	D	B	F	G

第6步：插入节点E到图中，开始判断其余节点经过E节点，到达另一节点的路径权值有没有变小

当插入E节点之后，B节点与G节点之间建立通路，距离为15；C节点与G节点之间建立通路，距离为16

且没有使得其他节点之间距离变小，所以：

更新邻接矩阵B行G列为15；C行G列为16

更新中转点矩阵B行G列为E；C行G列为E

邻	A	B	C	D	E	F	G	转	A	B	C	D	E	F	G
A	0	24	8	15	15	11	19	A	A	B	C	D	C	C	D
B	∞	0	∞	∞	6	∞	15	B	A	B	C	D	E	F	E
C	∞	∞	0	∞	7	3	16	C	A	B	C	D	E	F	E
D	∞	∞	∞	0	∞	∞	4	D	A	B	C	D	E	F	G
E	∞	∞	∞	∞	0	∞	9	E	A	B	C	D	E	F	G
F	∞	∞	∞	5	2	0	9	F	A	B	C	D	E	F	D
G	∞	3	∞	∞	9	∞	0	G	A	B	C	D	B	F	G

第7步：插入节点F到图中，开始判断其余节点经过F节点，到达另一节点的路径权值有没有变小

当插入F节点之后，A节点到经过C节点经过F节点到达E节点的距离13，比A节点经过C节点直接到达E节点的距离15小；

C节点经过F节点建立与D节点之间的通路，距离为8；

C节点经过F节点到达E节点的距离5，比C节点直接到达E节点的距离7更近；

且从C节点出发经过F节点经过D节点到达G节点的距离12小于从C节点出发经过E节点到达G节点的距离16

所以：

更新邻接矩阵A行E列为13；C行D列为8；C行E列为5；C行G列为12

更新中转点矩阵A行E列不变；C行D列为F；C行E列为F；C行G列为F

邻	A	B	C	D	E	F	G	转	A	B	C	D	E	F	G
A	0	24	8	15	13	11	19	A	A	B	C	D	C	C	D
B	∞	0	∞	∞	6	∞	15	B	A	B	C	D	E	F	E
C	∞	∞	0	8	5	3	12	C	A	B	C	F	F	F	F
D	∞	∞	∞	0	∞	∞	4	D	A	B	C	D	E	F	G
E	∞	∞	∞	∞	0	∞	9	E	A	B	C	D	E	F	G
F	∞	∞	∞	5	2	0	9	F	A	B	C	D	E	F	D
G	∞	3	∞	∞	9	∞	0	G	A	B	C	D	B	F	G

注意：中转点矩阵的A行E列之所以不变还是C，是因为我们从A节点出发，依然首先到达C节点，然后转而从C节点到达F节点最后才抵达E节点

所以我们也可以认为中转点矩阵记录的是从m节点到n节点之间最短路径上的第一个中转点的取值

第8步：插入节点G到图中，开始判断其余节点经过G节点，到达另一节点的路径权值有没有变小

当插入G节点之后，A节点经过D节点经过G节点到达B节点的距离22，比从A节点出发直接到达B节点的距离24更近；

并且在C节点到B节点之间建立了通路，从C节点出发经过F节点经过D节点经过G节点到达B节点，距离为15；

在D节点和B节点之间建立了通路，从D节点出发经过G节点到达B节点，距离为7；

在D节点和E节点之间建立了通路，从D节点出发经过G节点到达B节点到达E节点，距离为13；

在E节点和B节点之间建立了通路，从E节点出发经过G节点到达B节点，距离为12；

在F节点和B节点之间建立了通路，从F节点出发经过D节点经过G节点到达B节点，距离为12

所以：

更新邻接矩阵：

A行B列为22，C行B列为15，D行B列为7，D行E列为13，E行B列为12，F行B列为12

更新中转点矩阵：

A行B列为D，C行B列为F，D行B列为G，D行E列为G，E行B列为G，F行B列不变

邻	A	B	C	D	E	F	G	转	A	B	C	D	E	F	G
A	0	22	8	15	13	11	19	A	A	D	C	D	C	C	D
B	∞	0	∞	∞	6	∞	15	B	A	B	C	D	E	F	E
C	∞	15	0	8	5	3	12	C	A	F	C	F	F	F	F
D	∞	7	∞	0	13	∞	4	D	A	G	C	D	G	F	G
E	∞	12	∞	∞	0	∞	9	E	A	G	C	D	E	F	G
F	∞	12	∞	5	2	0	9	F	A	D	C	D	E	F	D
G	∞	3	∞	∞	9	∞	0	G	A	B	C	D	B	F	G

经过上面的一番折腾，佛洛依德算法结束

实际上佛洛依德算法计算从m点到n点之间的最短路径，都是在前一步的基础上，根据新节点的插入，计算能不能缩短路径而计算得到的

所以每一次在插入一个新节点的时候，都需要知道之前的邻接矩阵结构和中转点矩阵结构

③总结迪杰斯特拉算法和佛洛依德算法

在上述两种算法当中，所有使用到的表格和矩阵的结构，都是在不断变化、不断更新的

也就是说，随着算法的逐步进行，答案的整体结构也在不断变化，只有到算法的最后一步，我们带能够得知最终结果的样貌

这种算法思想，实际上就是算法中常见的动态规划算法思想

可是有的同学不禁会问：尤其是在迪杰斯特拉算法中，我们不总是寻找例起点最近的点加入S集合中吗？这不应该是贪心算法的思路吗？

没错！我们可以认为，贪心算法是一种极端情况下的动态规划算法，至于二者之间的关系，我们将会在后面的算法专题中进行详细描述