最短路径算法

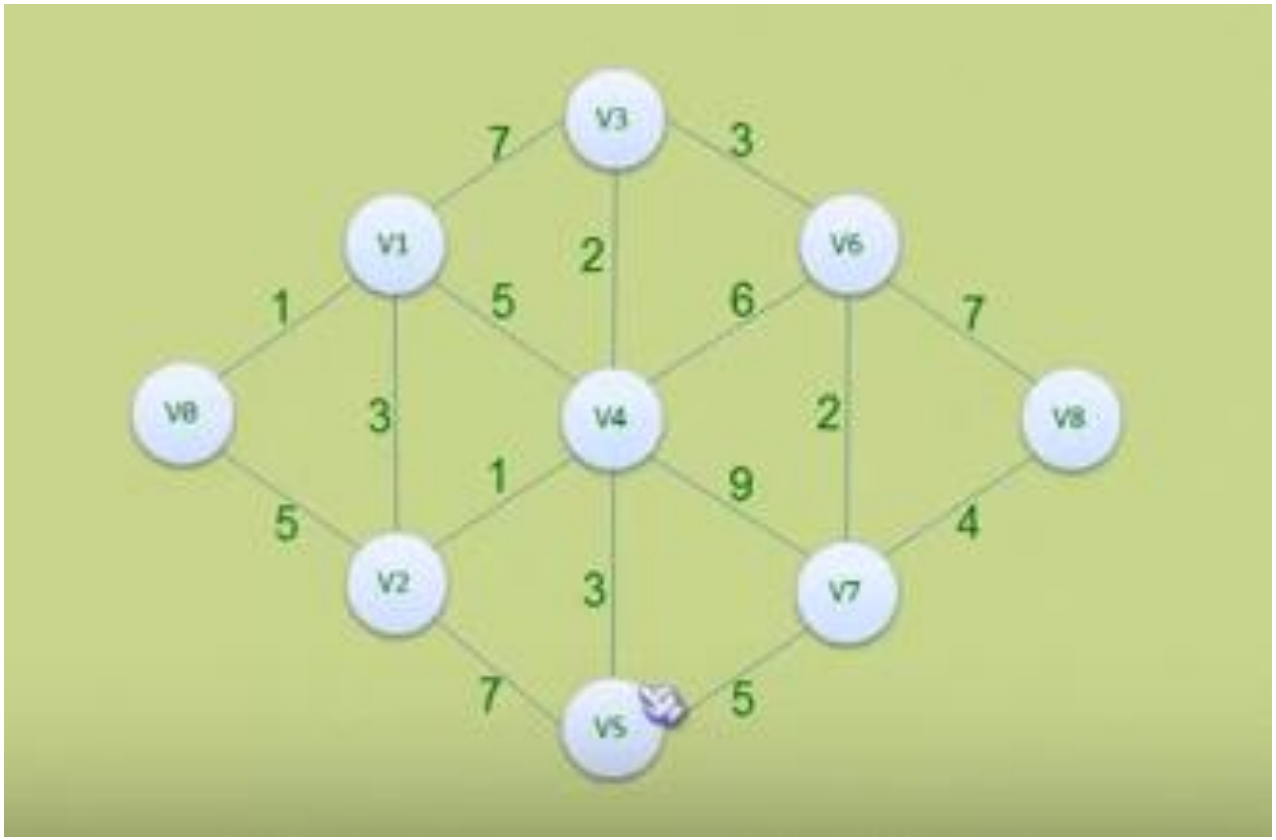
Dijkstra(迪杰斯特拉)算法

- 算法思想：设G=(V,E)是一个图，把图中顶点集合V分成两组，第一组为已求出最短路径的顶点集合（用S表示，初始时S中只有一个源点，以后每求得一条最短路径，就将加入到集合S中，直到全部顶点都加入到S中，算法就结束了），第二组为其余未确定最短路径的顶点集合（用U表示），按最短路径长度的递增次序依次把第二组的顶点加入S中。

Dijkstra(迪杰斯特拉)算法

- 在加入的过程中，总保持从源点v到S中各顶点的最短路径长度不大于从源点v到U中任何顶点的最短路径长度。此外，每个顶点对应一个距离，S中的顶点的距离就是从v到此顶点的最短路径长度，U中的顶点的距离，是从v到此顶点只包括S中的顶点为中间顶点的当前最短路径长度。

Dijkstra(迪杰斯特拉)算法



Dijkstra(迪杰斯特拉)算法

- 比如选自v_0作为起点(一般称为源点)，则根据图中的数据首先建立邻接矩阵和两个集合.第一个集合只包含点v_0，第二个集合包含剩余的所有点，此时，第二个集合为：

$$\{v_1(1)(v_1), v_2(5)(v_2), v_3(\infty), v_4(\infty), v_5(\infty), v_6(\infty), v_7(\infty), v_8(\infty)\}$$

小括号中的数字表示目前v_0 到该点的距离(只算直接距离，即邻接矩阵中的距离)，小括号中的点表示从v_0 到该点的最短路径中的倒数第二个点，如果没有中间点，则该点是最后一个点。

Dijkstra(迪杰斯特拉)算法

第二个集合中的距离都是临时距离，随着不断的更新可能会发生改变，但距离中最短的那个是准确的最短距离，不会再发生改变，所以将距离最短的顶点转移到第一个集合中，然后更新第二个集合中的距离。

得到:$\{v_0, v_1(1)(v_1)\}$

$\{v_2(4)(v_1), v_3(8)(v_1), v_4(6)(v_1), v_5(\infty), v_6(\infty), v_7(\infty), v_8(\infty)\}$

其中，更新距离时，计算每个点到第一个集合中的点的距离，比如对于v_2，可以得到其直接到v_0 的距离为5 ，到v_1 的距离为3(通过邻接矩阵计算)，而v_0到v_1的距离在集合1中已经得到了准确的最小值，为1 ，所以如果通过v_1中转，v_0到v_2的距离就只剩下了4 ，小于5 ，所以v_0 的距离更新为4 ，因为需要通过v_1中转，且v_1是v_2前面的第一个点，所以v_2 后面小括号中的顶点更新为v_1，需要注意的是，这个v_1表示的是v_2 前面的第一个点，不是v_0 后面的第一个点，不明白的看下一页。

Dijkstra(迪杰斯特拉)算法

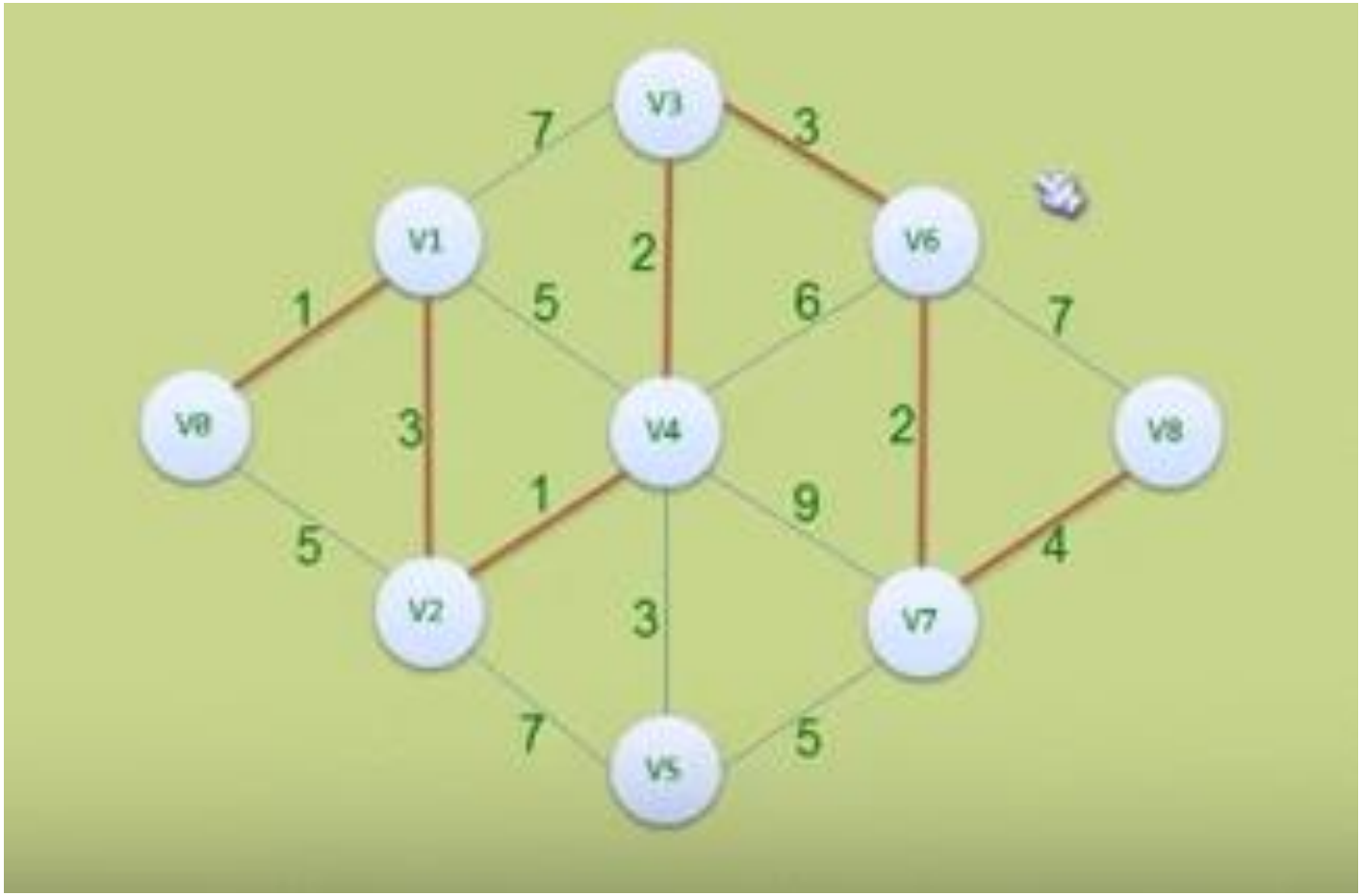
将v_2转移到第一个集合中，更新第二个结合可以得到

$\{v_0, v_1(1)(v_1), v_2(4)(v_1)\}$

$\{v_3(8)(v_1), v_4(5)(v_2), v_5(11)(v_2), v_6(\infty), v_7(\infty), v_8(\infty)\}$

其中v_4后面小括号中的v_2 表示从v_0 到v_4 最短路径中, v_4 前面的顶点是v_2,v_2前面的顶点通过第一个集合可以看出是v_1,然后通过第一个集合可以看出v_1 可以由v_0 直达, 通过这种方式得到v_0 到v_4 的最短路径为$v_0 \to v_1 \to v_2 \to v_4$, 距离为5.通过以上方法将第二个集合中的顶点全部转移到第一个集合，算法结束。

Dijkstra(迪杰斯特拉)算法



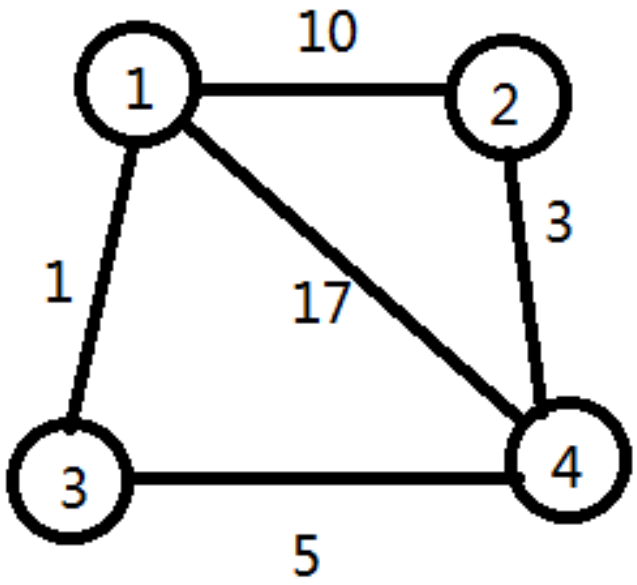
Floyd(弗洛伊德)算法

- 通过Floyd计算图G=(V,E)中各个顶点的最短路径时，需要引入两个矩阵，矩阵D中的元素d[i][j]表示顶点i(第i个顶点)到顶点j(第j个顶点)的目前已知的最短距离。矩阵P中的元素p[i][j]，表示顶点i到顶点j所经过的路径中最后一个顶点编号。假设图G中顶点个数为N，则需要对矩阵D和矩阵P进行N次更新。

Floyd(弗洛伊德)算法

- 初始时，矩阵D中顶点d[i][j]的距离为顶点i到顶点j的权值；如果i和j不相邻，则d[i][j]=∞，矩阵P的值为顶点p[i][j]的j的值。 接下来开始，对矩阵D进行N次更新。
- 第1次更新时，如果d[i][j]的距离 > d[i][0]+d[0][j]，则更新d[i][j]为d[i][0]+d[0][j],更新p[i][j]=p[i][0]。 同理，进行剩余的更新，更新N次之后，操作完成。

Floyd(弗洛伊德)算法



Floyd(弗洛伊德)算法

对于如图所示4个顶点，首先建立两个矩阵，距离矩阵为：

$$\begin{pmatrix} \infty & 10 & 1 & 17 \\ 10 & \infty & \infty & 3 \\ 1 & \infty & \infty & 5 \\ 17 & 3 & 5 & \infty \end{pmatrix}$$

路径矩阵为：

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

，该矩阵一般以0为第一列，此处为了演示方便以1开头。

Floyd(弗洛伊德)算法

以1号顶点为中间点, 更新矩阵, 首先计算顶点2 和顶点3 之间, 原距离为无穷, 如果以顶点1为中间点, 首先由2 到1, 再由1 到3, 可以得到两个距离分别为10 和1,所以通过顶点1中转后,2 和3 的距离降为11 , 更新两个矩阵为

$$\begin{pmatrix} \infty & 10 & 1 & 17 \\ 10 & \infty & 11 & 3 \\ 1 & 11 & \infty & 5 \\ 17 & 3 & 5 & \infty \end{pmatrix} \text{ 和 } \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 1 & 4 \\ 1 & 1 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

接着计算顶点2和顶点4的距离经顶点以中转后会不会减小, 如果会继续更新两个矩阵, 直到所有的顶点对计算完毕, 第一次更新结束. 将顶点2 作为中间点进行第二次跟新，直到所有的顶点都用过，算法结束。

第二个矩阵的数值与迪杰斯特拉算法中每个顶点后面第二个小括号中的顶点含义类似, 但有所不同, 具体哪里不同, 留作自己思考.

Dijkstra算法和Floyd算法比较

- Dijkstra(迪杰斯特拉)算法的时间复杂度是$O(n^2)$，Floyd(弗洛伊德)算法的时间复杂度是$O(n^3)$。
- Dijkstra算法的结果只能得到一个顶点到其他所有顶点的最短距离，Floyd算法得到的是所有顶点到所有顶点的最短距离。